Wireless Microservers

With Bluetooth components getting smaller and cheaper, we might soon integrate wireless microservers into all kinds of electronic devices. The authors explore applying a general-purpose, pluggable microserver, based on wireless application protocol and Bluetooth technology, for remote control purposes.

Stephan Hartwig and Jan-Peter Strömann
Nokia

Peter Resch
University of Dortmund

Since the early days of the Web, server-side executable content has been an important ingredient of server technology. It has turned simple hypertext retrieval into real applications. Not surprisingly, the idea of remotely controlling devices through the Web[1,2] has always seemed near at hand. Because hypertext user interfaces can run on any Web browser, UI development boils down to Web content creation. Furthermore, thanks to the HTTP standard's smart and scalable nature, we can fit embedded servers into simple 8-bit microcontrollers with only a few Kbytes of RAM and ROM (see the "Embedding Servers into Devices" sidebar).[3]

Ever since we started integrating hypertext browsers into mobile phones, people have proposed using mobile phones as remote controls. Now, with the provision of short-range wireless connectivity—for example, through Bluetooth—mobile phones and other handhelds might substantially change the way people interact with electronic devices. Here, we report on our effort to create a low-power wireless microserver with a very small form factor and connect it to mobile devices using standard consumer technology.

Candidate applications and technologies

Consumer electronics have used wireless low-cost remote controls for decades. Adding embedded servers to devices will create a new range of use cases beyond the limited capabilities of infrared remotes:

- Browsers could then allow real interaction instead of just sending one-way commands such as infrared remotes.
- We could harmonize similar operations in terms of a Web UI, even if the built-in UIs of the particular devices differ.[4] A good example is setting a device's clock, which is a common operation in many consumer devices but always requires a unique—and often error-prone—implementation.
- We could distribute the UI among the device's built-in UI and the handheld's UI.[4] For example, we could export parental control functions—such as "edit play-time budget or game type"—of game consoles to a mobile phone.
- A server with memory could personalize its UI and service by collecting and interpreting its own usage patterns and those of adjacent servers.
- Devices that lack a UI, have a restricted UI, or are purposely hidden could export a UI to a handheld.
- While the handheld is connected to the device, the device could connect to the Internet, in case the handheld supports cellular data calls.

Eventually, we could turn each compatible handheld into a general-purpose remote control.

In a joint project, Nokia and the University of

1536-1268/02/$17.00 © 2002 IEEE

Embedding Servers into Devices

There are several ways to make things visible on the Web. In the simplest case, a server hosts an item's Web presence without a physical connection to the item. A handheld device reads links between the item and its Web presence, connects to the respective URL, and retrieves information about the item. A well-known example for this approach is the Cooltown Museum,[1] where small infrared transceivers are located close to the pictures. When coming close, the visitor's PDAs receive Web links that point to the information pages for the particular picture. Unfortunately, interacting with the item itself is impossible.

Interaction with a device would be possible if the device had a wireless control interface to its internal logic. For example, the mobile terminal could download a device-specific user interface application from the Web and use it to control the device through a device-dependent protocol (see Figure A1). This approach might become feasible when we can download Java applications into mobile terminals with access to Bluetooth APIs (see http://jcp.org/aboutJava/communityprocess/review/jsr082). Accessing the device immediately and locally without an Internet connection would be possible only if the device contained an embedded Web server (see Figure A2). An execution environment, such as server-side scripting, would be required to interact with the device's logic.

Short-range connectivity seems to be an obstacle, but it empowers location-aware applications through the wireless link's limited reach. If a user wants to adjust a microserver-equipped TV's volume, he or she does not want to accidentally interact with somebody else's TV. Therefore, short-range wireless radio links, preferably using unlicensed bands, are well suited for networking things and people.

REFERENCE

1. T. Kindberg et al., "People, Places, Things: Web Presence for the Real World," *Proc. 3rd IEEE Workshop Mobile Computing Systems and Applications* (WMCSA '00), IEEE CS Press, Los Alamitos, Calif., 2000, pp. 19–21.

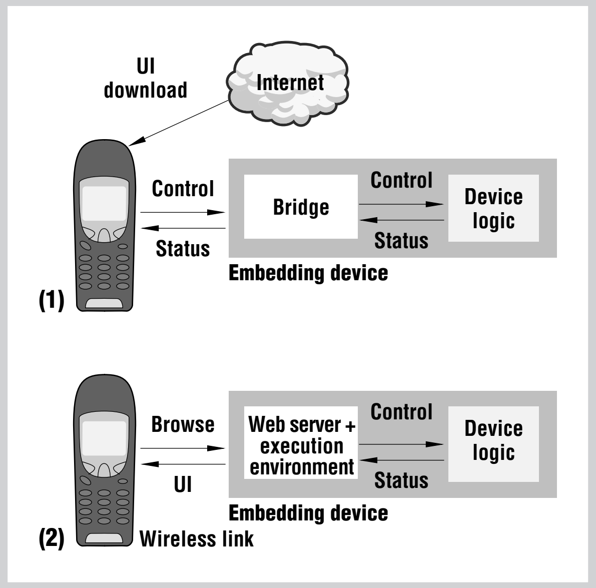


Figure A. (1) UI application (such as Java Midlet) is downloaded and controls the device by a control protocol. The internal and wireless link for the control protocol must be bridged. (2) The device hosts the server comprising UI and execution environment.

Dortmund created a low-power wireless microserver. (We use the term microserver to refer to small and cost-efficient embedded Web server implementations that are either integrated or plugged directly into the device.) Because mobile phones have a much higher population than PDAs and will likely be the first truly ubiquitous computing devices, we chose to connect the microserver to a mobile phone.

Although there are many technologies available for embedded middleware for distributed computing and service discovery, such as Java, Jini, and UPnP,[5] we decided not to use such middleware components. For our project, these technologies have two important drawbacks: they are not widely applied today and their implementation takes more processing resources then we were willing to spend. For example, a small implementation of Jini that doesn't even use Java-RMI would require approximately 100 Kbytes of ROM and 50 Kbytes of RAM plus another 100 Kbytes of RAM and 70 Kbytes of ROM for a lean Java virtual machine that includes basic packages (see www.psinaptic.com).

We chose to use Bluetooth because it is the only short-range radio technology currently deployed in mobile phones. Our implementation fit into the free memory of a commercial Bluetooth module, which was about 40 Kbytes of ROM and 4 Kbytes of RAM. For the phone, we used existing implementations of the browser and Bluetooth software without adding a new middleware component. For remote method invocation, we based our solution (discussed later) on the popular yet old-fashioned common gateway interface.

WAP and Bluetooth

The wireless application protocol is an industry-wide standard to connect mobile phones to the Web (www.wapforum.org). It is designed especially for the mobile phone environment and its limited battery power and memory, small display, and low transmission rates. Similar to the Japanese I-mode, WAP provides Web access for mobile devices. Usually, browsers directly request HTML content from a Web server using HTTP. A typical WAP infrastructure also needs HTTP (see Figure 1), but the WAP protocol stack (see Figure 2) and the WML (Wireless Markup Language) are tailored for the limited transmission capacity and resources of mobile devices. In addition, unlike HTTP, WAP allows server push operation, so the server can send a WML page to a browser without a request.

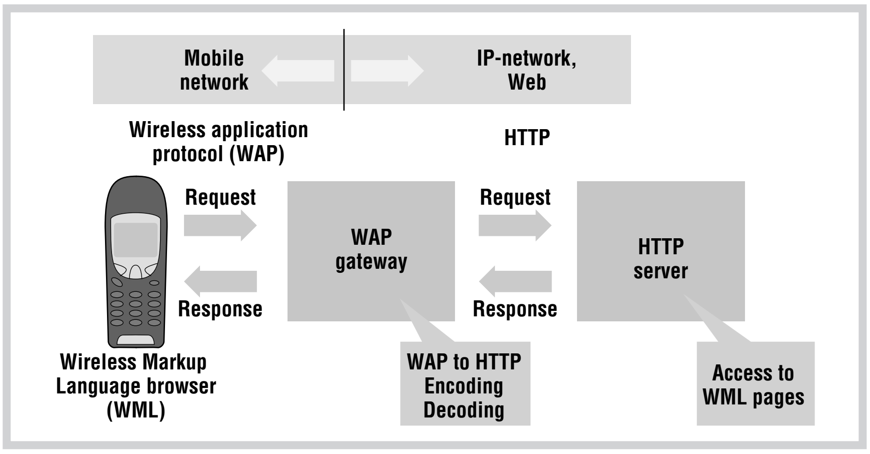


Figure 1. A wireless application protocol request and response model. A WAP request is directed to a WAP gateway. The gateway decodes the request and transforms it into an HTTP request for an ordinary Web server. The gateway then encodes the Wireless Markup Language page response from the HTTP server and sends it as a WAP response back to the requesting WAP device.

The WAP standard also specifies different levels of security that enable data encryption and trusted communication—for example, for electronic payments or banking applications.

Because future mobile phones will have larger screens and more processing power, the next-generation WAP standard (WAP 2.0) will support the HTTP protocol and XHTML as a markup language. Another reason for this support is the general trend toward all-Internet protocol (IP) infrastructures. Currently, most Web browsers available in Europe are based on WAP 1.1, which we selected for our implementation. However, you could apply our concept to other mobile Web technologies, such as I-mode and WAP 2.0 without a substantial change in complexity.

Bluetooth is an industry standard for short-range, low-power, wireless communications and networking (www.bluetooth.com/dev/specifications.asp). It uses radio transmission in the license-free band of 2.4 GHz and can transmit voice and data with bit rates up to 720 kbits/sec within approximately 10 meters.

Bluetooth technology supports point-to-point and point-to-multipoint connections. We can actively connect a Bluetooth device to seven devices simultaneously. These devices build a so-called *piconet*, and every piconet contains up to seven slaves and is controlled by a master. Several piconets can be linked together to form a *scatternet*.

Bluetooth was not envisaged to just replace cables with interconnecting mobile phones, PCs, and peripheral devices such as headsets or printers. It aims to let arbitrary electronic devices form ad hoc networks to jointly advertise and use each other's services. It specifies its own service discovery protocol, and every Bluetooth device usually implements its own service

Glossary

API	Application programmers' interface
BCU	Bus Coupling Unit
CDMA:	Code Division Multiple Access
DUN:	Dial-up networking
EIB	European Installation Bus
GPP	General purpose port
GPRS	General packet radio service
IP	Internet protocol
LAP	LAN access profile
PAN	Personal area networking
PEI	Physical external interface
UART	Universal asynchronous receiver transmitter
UI	User interface
WDP	Wireless datagram protocol
WAP	Wireless application protocol
WML	Wireless Markup Language
WSP	Wireless session protocol

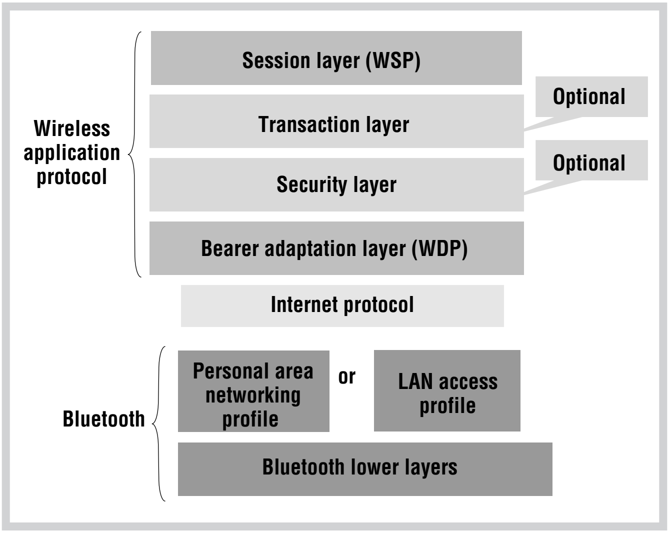


Figure 2. A WAP-over-Bluetooth protocol stack. Either the LAN access profile or the personal area networking profile is used for Internet protocol adaptation.

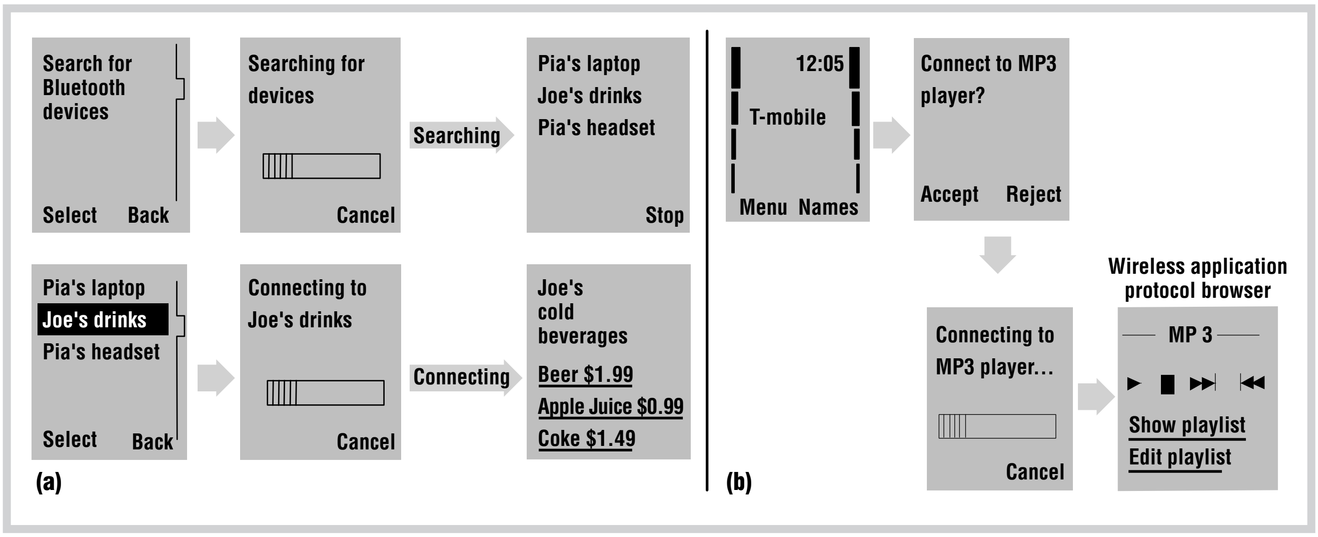

Search for Bluetooth devices
Select Back
Searching for devices
Cancel
Searching
Pia's laptop
Joe's drinks
Pia's headset
Stop
Pia's laptop
Joe's drinks
Pia's headset
Select Back
Connecting to Joe's drinks
Cancel
Connecting
Joe's cold beverages
Beer $1.99
Apple Juice $0.99
Coke $1.49
(a)
12:05
T-mobile
Menu Names
Connect to MP3 player?
Accept Reject
Connecting to MP3 player...
Cancel
Wireless application protocol browser
MP 3
▶ ■ ⏭ ⏮
Show playlist
Edit playlist
(b)

Figure 3. Consecutive screenshots of a mobile phone (a) while a user actively connects to a vending machine and (b) when the server initiates connection (the connection request—"Connect to MP3 player?"—comes without user interaction).

discovery server and client.

The Bluetooth community has identified many application areas for Bluetooth. To assure application-level interoperability between devices of different manufacturers, it is not sufficient to specify the Bluetooth technology; we also need to agree on the technology's use by higher-level protocols and applications. Bluetooth profiles address these aspects.

WAP over Bluetooth

The WAP protocol stack includes at the lowest level the wireless datagram protocol (WDP). This layer implements the bearer adaptation and is defined for a variety of bearers, such as GSM, Code Division Multiple Access, and the general packet radio service. The WDP doesn't cover Bluetooth but it does cover IP. In fact, WDP is identical to the well-known user datagram protocol, in case IP is the bearer. So the challenge is to define an IP encapsulation protocol for Bluetooth, which has been done in both the *LAN access profile* and the *personal area networking* (PAN) profile (see Figure 2). The LAP makes Bluetooth behave like a standard serial port that legacy software and devices can use. Consequently, the LAP uses the same mechanisms as a terminal that accesses a network over a serial connection, namely point-to-point protocol and IP. When using the PAN profile, Bluetooth behaves like a direct LAN connection: the Bluetooth Network Encapsulation Protocol makes Bluetooth behave similar to Ethernet using the Bluetooth device address as a hardware address.

The PAN profile implements a lighter stack than the LAP. The LAP is already approved in the Bluetooth 1.1 specification, but future implementations might favor PAN, because it doesn't require serial port emulation and point-to-point protocol.

WAP-over-Bluetooth user experience

At first glance, Bluetooth is just another bearer for an existing service. After we established the Bluetooth and WAP connection between the client and server, browser operation is the same. However, there are some major differences in the WAP user experience between cellular and Bluetooth connections.

The first difference concerns location awareness. Bluetooth's limited range is not necessarily a disadvantage. A WAP-over-Bluetooth server knows that once a user connects, he or she is close. Thus, it can offer location-aware services, and servers in different places can reuse the same URLs.

A second difference is that bandwidth is typically higher in Bluetooth than in cellular systems, allowing richer content and better performance.

Finally, there are two scenarios for establishing a connection. In the first scenario, the user (WAP client) initiates the connection (see Figure 3a). Upon user request, the WAP client searches for all Bluetooth devices in the user's proximity, then creates a WAP connection to the selected WAP service. The user might select a function such as "search for Bluetooth devices." Subsequently, the phone displays the Bluetooth devices found. The user selects the desired WAP service and the phone displays this server's starting page.

In the second scenario, the WAP gateway initiates the connection (see Figure 3b). In this case, we assume the WAP client is in "discoverable" mode—that is, other Bluetooth devices can find it. Once the user terminal comes into the gateway's proximity, the WAP gateway actively connects and displays its starting page on the user's phone. It is either an implementation choice or a user option whether to confirm the connection.

In both examples, we assume that neither the client nor server require authentication.

Implementation options for embedded WAP servers

If we leave out the WAP standard's optional features and merge the server with

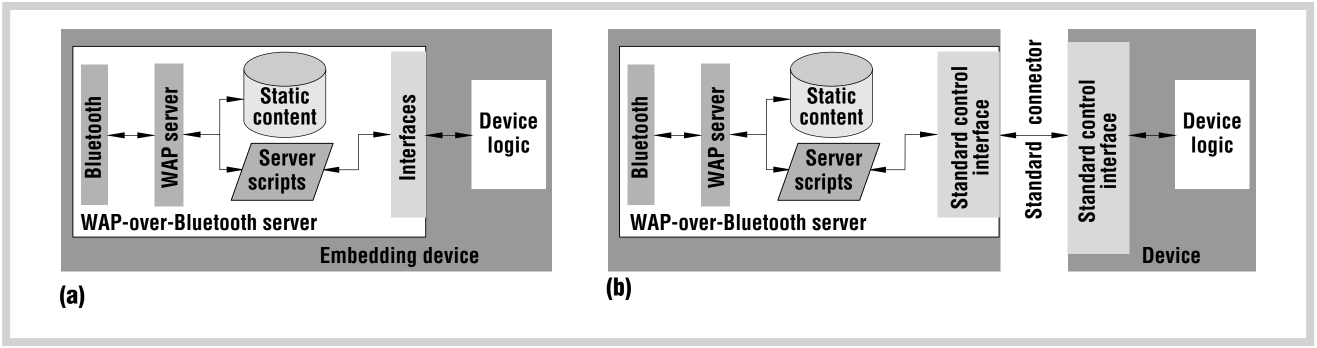


Figure 4. (a) An embedded WAP server versus (b) a pluggable server composed of a standard connector and a standard control interface.

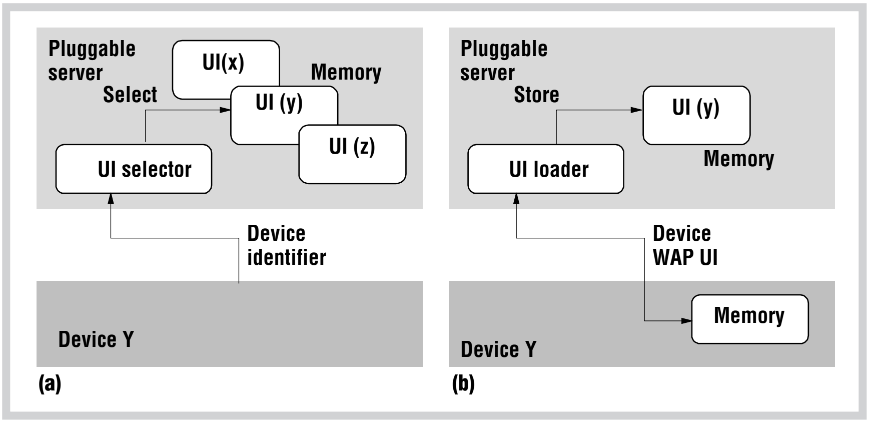


Figure 5. (a) A preprogrammed UI versus (b) a UI downloaded from the device.

the gateway, we can implement an embedded WAP-over-Bluetooth server in an 8-bit microcontroller using a few Kbytes of RAM and Flash, provided that Bluetooth (including the stack) is running on a dedicated component. Such optional features mainly include the WAP security layer and the transaction layer (see Figure 2). Bluetooth implements flow control, retransmission of lost packets, authentication, and encryption—thus replacing some of the sacrificed WAP features.

Because the embedded server does not require an internal separation between the WAP gateway and HTTP server, we can directly store the content in the final WAP-encoded message format, thus making WAP encoding and decoding obsolete. Figure 4a depicts a simple model of such an embedded WAP server. Typically, the WAP server would either run on the baseband processor inside a Bluetooth module or on a host processor connected to the Bluetooth module. The interfaces include general-purpose ports, a universal asynchronous receiver transmitter (UART), and the like. Logically, part of the WAP UI would be implemented as server-side scripts, triggering actions on user interaction.

The concept of embedded WAP servers is appealing at first glance. Any mobile phone can act as a remote control for Bluetooth devices, which simply export their UIs in a standard hypertext language. However, once these servers are embedded in low-cost devices, they can unduly increase the devices' costs. The server's cost amounts to at least the cost of the Bluetooth module, including external circuitry. This can be 10 Euros and more, and will remain so at least until mid 2003. Customers using the embedded server feature might be willing to pay more, but many people might end up paying for a feature they never use. Moreover, Bluetooth technology and hypertext UI development is clearly outside the core competence of many device manufacturers.

Instead of embedding the server into the device, we could provide a robust and cheap standard connector where a pluggable server can be retrofitted (see Figure 4b).[6] Such a connector would, for example, provide several general-purpose ports and a UART option. Unlike embedded servers, we could cost-efficiently manufacture general-purpose pluggable servers in large quantities. Moreover, if we could separate the UI content from the corresponding devices and pluggable servers, then independent marketplaces for UIs, servers, and devices could emerge. We could move pluggable servers from one device to another—for example, if a device is not used for some time, we could remove the server and plug it into some other device. This concept could, for instance, be interesting for toy construction kits (such as Lego), which increasingly use digital technology to construct sophisticated machines, vehicles, and robots. Because these kits consist entirely of pluggable components, a pluggable server would be a natural extension. Eventually, we'll have to decide whether to use an embedded or pluggable server on a case-by-case basis. However, the question remains, "How do we get the UI into the server if we purchased the server independent of the device?"

Distributing the UI

We must either preload general-purpose, pluggable servers when selecting a candidate UI or externally load the UI when we first plug it in. In addition, for built-in servers, we'll want to be able to remotely update UI content. Here, we present four methods for provisioning UIs.

We borrow our first method, which applies to pluggable servers, from general-purpose IR remote controls: Every server can be preprogrammed for many devices. Once the server is plugged into a device, the device identifies itself to the server (either on startup or on the device's request) and the server selects the corresponding UI from several available UIs (see Figure 5a). This method is not very flexible and adds undue cost to the server because it reserves memory for UIs that a particular customer might never use.

More desirable are servers that download the desired UI from some source. This method assumes that the device simply stores its UI in memory (for example, ROM) and uploads it to the server once it is plugged in (see Figure 5b).[6] Compared to the previous method, the total cost is lower, because only the required UI must be stored in memory. However, it makes the device slightly more expensive, and the server—but not the UI—is provided separately from the device. Moreover, we can't easily update the UI.

Obviously, the most flexible method is to download the UI over a network (see Figure 6). One simple implementation is to connect the server to a PC that is connected to the Internet. The PC downloads the UI file from the manufacturer's Web site and then stores it in the pluggable or built-in server.

The fourth and most convenient method assumes that the server creates a Bluetooth dial-up network connection to the UI server through a Bluetooth phone. Advantageously, this phone would be the same phone that is used to access the server. This method consists of several steps (see Figure 7):

1. The user plugs the server into the device and creates a Bluetooth connection to it.
2. The server retrieves the device ID and manufacturer URL from the device.
3. The server discovers that the UI is not in memory. It creates a dial-up network connection to an ISP and downloads the UI from the manufacturer or some service provider. Alternatively, the pluggable server directly dials up the manufacturer. Dial-up networking over Bluetooth is supported by the dial-up networking profile.
4. After the server has downloaded the UI, it presents the UI starting page to the user.

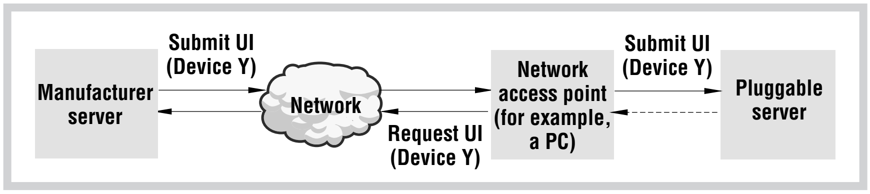


Figure 6. Downloading a UI over a network.

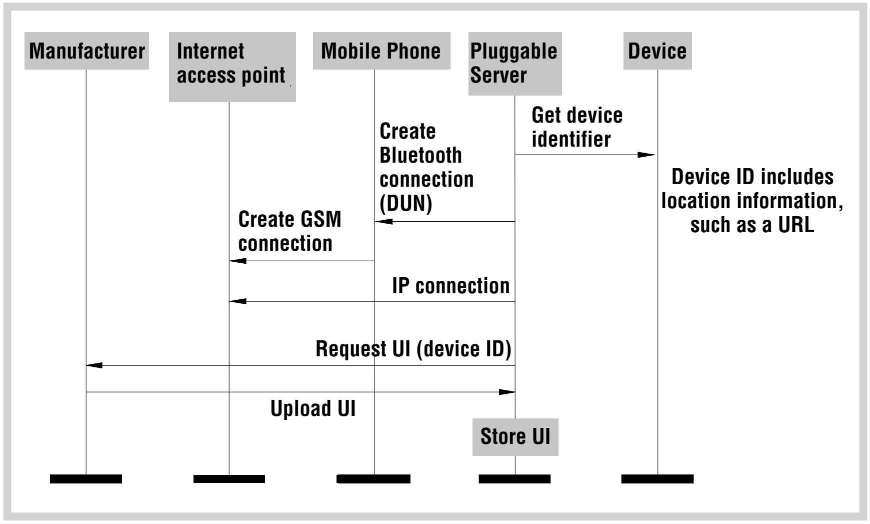


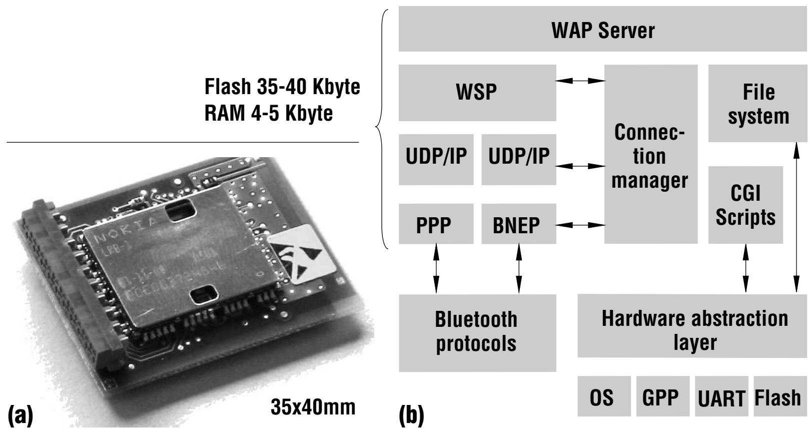
Figure 7. Downloading a UI over a network (method 4).

Clearly, the UI does not have to be prestored in the server, and no other devices (such as a PC) are needed for the download. This procedure could also be useful for regularly updating the UI or downloading informative pages into the server (perhaps to advertise the device's new model). Only when the server is plugged into a device for the first time will the user have to wait until the download completes. However, the server could cache the UIs of recently connected devices to avoid delays when being moved from one device to another. Typical WML pages occupy only a few hundred bytes of memory.

A simple method invocation procedure through CGI scripts

We cannot realize remote control applications without server-side scripting, because user interaction must trigger I/O operations on the server's interface to the device's logic. The challenge is to achieve some basic means for invoking device functions without using distributed computing middleware such as Jini or UPnP.

We implemented a general-purpose CGI script, `call.cgi`, as part of the server software. We used the script to invoke methods on the server's interface. We called it with a query string for selecting and parameterizing the desired operations. For example, loading the following URL



WSP	Wireless session protocol
UDP/IP	User datagram protocol/Internet protocol
PPP	Point-to-point protocol
BNEP	Bluetooth network encapsulation protocol
OS	Operating system
GPP	Generate purpose ports
UART	Universal asynchronous receiver transmitter

Figure 8. (a) Microserver prototype implementation and (b) the most important blocks of the software architecture.

```
call.cgi ? Port1=1 & serial=myString &
resultURL=myURL.wmlc
```

would set Port1 to one, read a string from the serial port, store the string into the browser variable myString, and load URL myURL.wmlc. (The actual syntax of the implemented script is different. This example just explains the idea.) Besides reading and writing to the interfaces, the implemented script can also increase and decrease counters and read and write string tokens (like cookies) to the server's flash memory. This approach is rather primitive, but it works fine for many applications without requiring custom server scripts. We implemented the script in native code as part of the server and did not require any script engine for execution. The ROM usage was 3 Kbytes.

WAP-over-Bluetooth demonstrator implementations

We developed two working implementations of the WAP-over-Bluetooth server—a PC reference implementation using a Bluetooth PC card and an embedded implementation. Figure 8 shows the server hardware (the dimensions are 32×40 mm), with the server implemented inside a Bluetooth module. The server's total footprint is about 4 or 5 Kbytes of RAM and 35 or 40 Kbytes of ROM, depending on the profile used (LAP or PAN). This includes a connectionless WAP stack without WAP security. The connector implements two UARTS, several I/O ports, and a power supply. We can update the server's content and software through a PC application but not yet over server-initiated dial-up networking. We also modified a commercial Bluetooth phone's software to support WAP-over-Bluetooth.

The first ideas many people had were about remote-controlled PC applications, even though these are not embedded applications. We implemented a WAP application to control PowerPoint presentations using a Bluetooth phone. Another interesting demo application we did was to jointly edit an MP3 player's play list (running on a Bluetooth-equipped laptop) for clubs and private parties. Interestingly, these kinds of demos were rather easy to implement once the server platform was available. Two students were able to do both the PowerPoint and MP3 demos in just a few days. However, the implementations that used the embedded microserver hardware were the most interesting.

The toy crane demonstrator

The toy crane demonstrator is a typical candidate for a pluggable server. We bought a toy crane and removed the cable remote to connect the motors to relays. We then connected these relays to a microserver's general-purpose ports, including the corre-

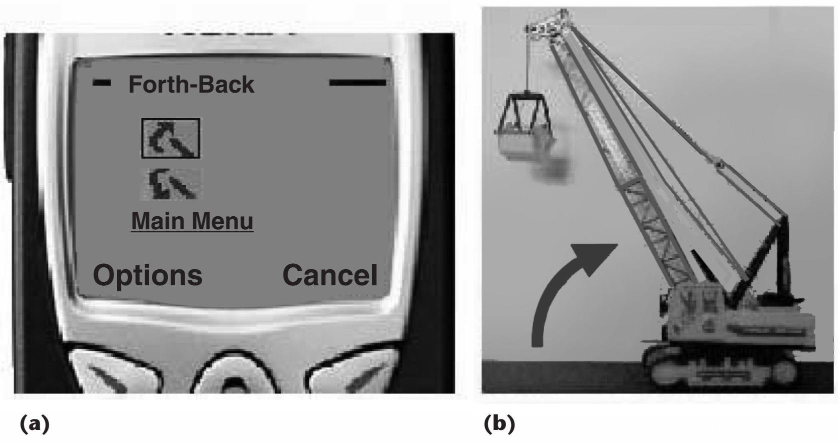


Figure 9. The toy crane demonstrator. The server is located in the driver's cab and connected through relays to the crane's motors. The (a) current WAP menu displayed is used to (b) lift and lower the crane's boom.

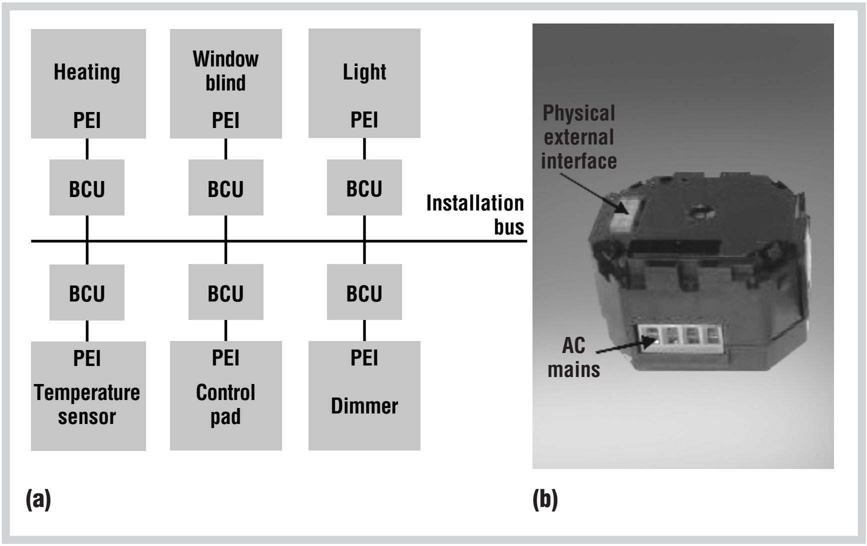


Figure 10. (a) The installation bus provides a network between devices and controls; (b) all devices and controls are connected via a Bus Coupling Unit (BCU).

sponding UI. We purposely selected the toy crane to see whether a WAP UI could sufficiently replace the powerful joystick operation.

We showed the crane at trade shows and to the Bluetooth congress. Users familiar with WAP browsers could operate the crane intuitively without major instructions. We displayed the links as small icons representing individual control options. The phone UI didn't allow any proportional control (similar to a joystick), but we implemented the WAP content (see Figure 9) such that users could repeatedly start and stop a selected movement (for example, "lower the boom") by pressing the same button. The delay between key presses and motor activation was negligible.

Home automation

For the next demonstrator, we connected a bluetooth server to a home bus. The goal was to operate and control devices on the home bus via Bluetooth phones. We implemented a working version of the demo using the PC reference implementation.

The most widely deployed installation bus for home automation in Europe is the European Installation Bus (see www.eiba.org). An installation bus system aims to decouple network control and AC power distribution. This is achieved by providing two logically independent networks: first, the 110/220V power distribution network, and second, a low-voltage network for control purposes. Alternatively, the control information could be transmitted through the power distribution network using modulation techniques, which saves extra wiring (a "powerline EIB"). The advantage of separating power distribution and control is that the connection between control points (switches, dimmers, and control pads) and devices is not hardwired but can be configured. Similar to a computer network, datagrams are sent between control points and devices to trigger the desired actions in the devices—like switching a light on and off.

Every control point (switch, button, control pad, and so forth) can be set up to control every device or clusters of devices (see Figure 10a). In the simplest case, a light switch is configured to switch a particular lamp on or off. A PC or a dedicated EIB device connected to the installation bus configures the network.

EIB control points usually consist of two components. The main part, normally not visible to the user, is the Bus Coupling Unit (see Figure 10b). The second part, the application module, contains the UI and the control logic. This could be a simple light switch but also a heating regulator. The application unit is plugged directly into the BCU's physical external interface (PEI) and thus hides the BCU in the wall.

EIB bus systems are widely deployed, mainly in office buildings and hospitals but also in houses. Although these systems provide some convenience, they still suffer from the fact that the UI to the home network is largely unchanged compared to plain old installations: switches, dimmers, and so forth.

Here the idea of the pluggable server

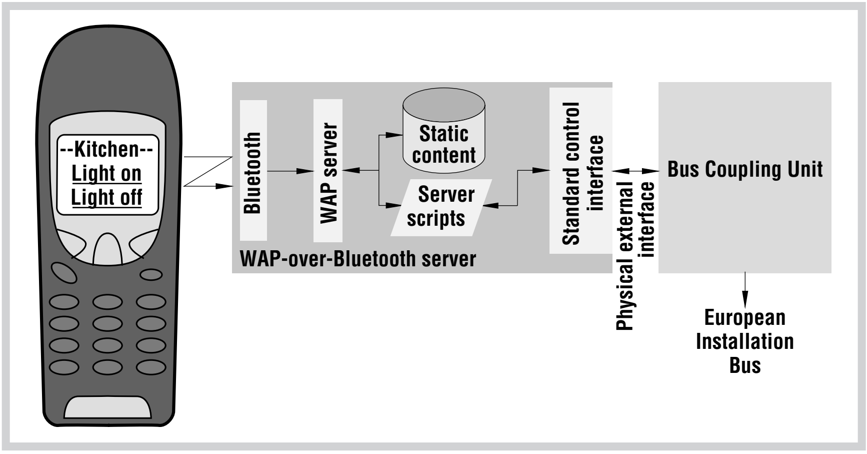

--Kitchen--
Light on
Light off
Bluetooth
WAP server
Static content
Server scripts
Standard control interface
WAP-over-Bluetooth server
Physical external interface
Bus Coupling Unit
European Installation Bus

Figure 11. Controlling the home network via a plugged-in WAP-over-Bluetooth server.

comes into play: Every BCU can send control datagrams to any device connected to the installation bus. Logically, we could go into an EIB networked building, unplug an arbitrary application module, and plug a WAP-over-Bluetooth microserver into the BCU using the PEI interface (see Figure 11). Once the microserver is loaded with the correct content, a user can control any device connected to the installation bus through his or her Bluetooth phone's WAP browser. For our demonstrator, we connected a Bluetooth-equipped laptop to the BCU. In Cooltown terms, this could be called a *placeManager*,[7] because it provides a Web presence for the devices and organizes them in its Web presentation.

To ease content creation, we created a software tool that automatically converts the network configuration data into server content. The configuration data is available as a binary file, which is created by the commercial EIB software tool used to configure the installation bus.

The most interesting lesson learned from this demonstrator is that, in some areas, deploying pervasive computing applications does not suffer from the famous hen-and-egg problem. These installation busses, for example, are deployed today and a small additional investment for the pluggable server turns such a building into an attentive environment. Typical installation busses also contain sensors such as light meters and thermometers that might provide useful context information. It would be interesting to study how to make the WAP UI context aware, also taking into account personal usage patterns.

Future implementations of both mobile and embedded servers will be more powerful and thus will use true distributed computing middleware. However, for now, we have shown that connecting electronic devices to the Web with inexpensive standard technology is possible and is sufficient for many applications. A major challenge will be to identify application areas where we can deploy pervasive computing technology in consumer domains without major investments in infrastructure. One possibility is to determine where we can add value to legacy systems by adding embedded server technology, like we did with the EIB implementation. Finally, we have to give device manufacturers a cost efficient option for making their products ready for integration into a pervasive computing environment without committing to a particular pervasive computing technology or middleware. A simple—yet to be standardized—control interface is such an option.

the AUTHORS

Stephan Hartwig is a project manager for Nokia Mobile Phones and is currently involved in Bluetooth software development for mobile phones. He has been working as SW designer and project manager in projects related to digital TV systems, mobile multimedia, and bluetooth solutions. Other research interests include adaptive filters, image sequence compression, and formats conversion. He received his diploma degree in electrical engineering from the University of Bochum and a PhD from the University of Dortmund, Germany. He is member of the VDE, ITG, and IEEE.Contact him at Nokia GmbH, Rensingstr. 15, 44807 Bochum, Germany; stephan.hartwig@nokia.com.

Jan-Peter Strömann is a system design engineer at the Product Creation Center of Nokia Mobile Phones. His development interests include system integration of messaging components such as WAP and MMS. He studied electrical engineering at the University of Dortmund. His diploma thesis dealt with embedded WAP servers as gateway systems for installation bus systems. Contact him at Nokia GmbH, Rensingstr. 15, 44807 Bochum, Germany; jan.stroemann@nokia.com.

Peter Resch works as an electrical engineer at the Computer Engineering Institute, University of Dortmund. His research interests include microprocessor design and the design of mobile and ubiquitous devices and their applications. He helped to create a new ubiquitous computing research group at the institute. He received his graduate engineer at the Dortmund University of Applied Sciences. Contact him at University of Dortmund, Computer Engineering Institute; peter.resch@udo.edu; www-ds.e-technik.uni-dortmund.de.

ACKNOWLEDGMENTS

We thank the Nokia thin server team members who contributed to these demonstrations, especially Eugen Palnau, Stefan Kapsokefalos, Ossi Rabina, Ari-Pekka Peltola, and Jari Kokkonen. We also thank Uwe Schwiegelshohn at the University of Dortmund for his support. The work on the EIB system would have been impossible without the cooperation and support of Christian Heite and Martin Gräfe of ABB Busch Jäger, who provided expertise on EIB systems. The authors alone are responsible for the contents of this article. In addition, the article presents research activities and does not make any statement about the commercial availability of such technology by any vendors.

REFERENCES

1. US Patent 5,956,487, "Embedding Web Access Mechanisms in an Appliance for User Interface Functions Including a Web Server and Web Browser,"1996, document US 5956487; www.depatisnet.de.
2. S. Hartwig et al., "WAP over Bluetooth: Technology and Applications," *IEEE Int'l Conf. Consumer Electronics*, IEEE Press, Piscataway, N.J., 2001.
3. J. Bentham, *TCP/IP Lean*, CMP Books, 2000.
4. B.A. Myers, "Using Hand-Held Devices and PCs Together," *Comm. ACM*, vol. 44, no. 11, Nov. 2001, pp. 34–41.
5. J. Burkhardt et al., *Pervasive Computing*, Addison Wesley, Reading, Mass., 2002.
6. Patent Application WO 01/41408 01 A1, "A Device and a Method for Operating an Electronic Utility Device from a Portable Telecommunication Apparatus," document W0200141408; www.depatisnet.de.
7. T. Kindberg et al., "People, Places, Things: Web Presence for the Real World," *Proc. 3rd IEEE Workshop Mobile Computing Systems and Applications* (WMCSA'00), IEEE CS Press, Los Alamitos, Calif., 2000, pp. 19–21.