

A Robot Interface Using WAP Phone

Pablo d'Angelo* and **Peter Corke**

CSIRO Manufacturing Science & Technology
PO Box 883, Kenmore, Qld 4069, Australia
<http://www.cat.csiro.au/cmst/automation>

Abstract

Mobile phones are now ubiquitous and many of these now include micro-browsers. In this work we explore the potential, and the limitations, of widely available WAP technology for online control and monitoring of a robot.

1 Introduction

Today, without realizing it, most people carry with them a robot control interface — a mobile phone. WAP enabled mobile phones are increasingly common (and powerful) and provide a possible alternative to the classical teach pendant or a UI running on a computer connected to the robot. Compared to a normal robot teach pendant phones are small and wireless.

This paper describes our initial steps in developing a mobile phone interface for a 4 joint hydraulic arm, shown in Figure 1.



Figure 1: Experimental hydraulic robot arm.

The Internet has been used for remote of machines. Taylor and his group at the University of Western Australia built a web interface [Taylor and Dalton, 1997] for their ASEA IRb-6 industrial robot arm, it went online in

*University of Applied Sciences Ulm, Germany

September 1994. It is still running and can be reached on <http://telerobot.mech.uwa.edu.au/>. As the Internet technologies advanced the interface has been improved from a pure HTML interface to a more interactive JAVA applet that gives the operator better feedback and control by providing interactive 3D models to plan the motions visually. The a big problem when using the Internet (and other high latency networks) as part of the robot control structure are the long and unpredictable time delay introduced by the network. To archive a usable interface a high level of control (supervisory control) is needed to reduce the number of interactions needed for a desired result.

The mobile Internet is based around the Wireless Application Protocol (WAP) which has been developed by the WAP Forum <http://www.wapforum.org>. WAP itself defines the needed infrastructure for communication between mobile client and service provider (telephone company). From the content providers point of view it looks very similar to the “traditional” WWW environment, the only difference is the markup language, which is WML instead of HTML. An overview of the general architecture is shown in Figure 2. The WAP gateway is normally run by the phone company and translates the WML from its normal verbose XML text representation into a binary XML representation called WBXML which is transmitted to the phone using WAP. Some gateways also support transcoding of HTML to WML, which enables you to browse the normal HTML pages with your phone.

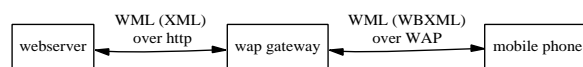


Figure 2: From webservice to phone.

The use of WAP for monitoring and control has also received recent attention, for instance CSIRO is developing a Mobile SCADA [Taylor, 2001] (Supervisory Control and Data Acquisition) infrastructure. It can be used to build distributed applications driven by simple wireless devices, one example application is the Sydney Coke Vending Machine Demonstration, which allows you to buy a coke from just by

dialing its phone number from your mobile phone. The Mobile SCADA architecture itself is very flexible, it's based on a database that receives incoming events and knows how to react to them.

The next section describes the architecture of our WAP-based robot control system.

2 Architecture

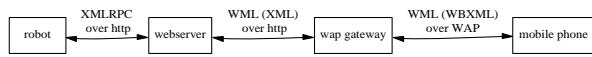


Figure 3: From robot to phone.

An overview of the architecture used on this particular application is shown in Figure 3 where the salient difference compared to Figure 2 is the robot device shown on the left.

We consider the problem in two parts:

- robot to server, and
- server to phone

for the following reasons:

- **Security.**
 - The HTTP server to serve WML pages to the WAP gateway must be reachable over the public Internet, but the operating system run on the robot is generally not as carefully examined for security bugs and misconfiguration as a dedicated machine running a public webserver.
 - Since most organisation use firewalls shield the internal network from the outside, in general the robot is not directly reachable by the WAP gateway.
 - Network security requires different expertise to robot control and these functions should be logically partitioned. In general robot software programmers are not computer security experts and the main goal while writing the software was implementing a good robotic system and not building a bulletproof architecture and programs which withstand a careful examination for security bugs.
- **Keeping the robot simple.**
 - Removing unnecessary programs from the robot makes the system easier to build and maintain, increases the stability and keeps CPU power for more important tasks. Changes to the UI can be done without touching the robot at all.
 - Fully functional web servers (such as Apache+PHP) are not well supported on real-time platforms such as LynxOS or QNX Neutrino.
- **Conserving network bandwidth.** Mobile robots will utilise slow wireless data links such as 802.11 or GSM. A lot of time and bandwidth can be saved if only RPC data is shipped over that line and the more verbose

markup languages are delivered over the faster connections between webserver and WAP gateway.

2.1 Robot to Server

Traditionally our robot systems have used an inhouse developed interprocess communication layer called RTC (Remote Tool Control [Roberts *et al.*, 1999]). RTC is a string transport protocol layered over SUNRPC. Server/Client libraries exist for C and Tcl and a client interface for Java has been written using the Java native interface (JNI). Tcl/Tk allows for extremely rapid development of GUI for robotic systems. However the portability of both Java and Tcl/Tk is limited by the need for SUNRPC which is only commonly available for UNIX-like platforms.

For this project we decided to try XMLRPC [Winer, 1999; Martin *et al.*, 2000], an XML-encoded RPC Protocol sent over HTTP. XMLRPC is supported by over a dozen languages across most platforms and also offers typed values instead of plain strings as RTC does. XMLRPC was chosen over SOAP (Simple Object Access Protocol, the XML-based RPC standard by W3C) because it is lighter and has more implementations. Robot "middleware" is an area of growing interest for which some have proposed the use of CORBA (Common Object Request Broker Architecture). Although CORBA is widely supported and offers a truckload of features, it has larger system resource requirements and involves a steep learning curve. Since our focus is a simple, easy to use and medium featured RPC mechanism we decided not to use CORBA.

A significant attraction to us of XMLRPC is the existing code base, but it was quickly found that the C server implementation used a slow forking HTTP server. A new HTTP implementation that uses threads and also supports some HTTP 1.1 features like persistent connections to avoid establishing a new TCP/IP connection for every function call was written. This server runs on all our platforms: LynxOS, Solaris, QNX and Linux. XMLRPC has bindings for languages such as C, Java, PHP and Tcl though not all support persistent connections.

A problem with XMLRPC itself is the very verbose XML grammar it uses. For example a double array containing 360 elements (modeled after a dataset returned from a laser rangefinder) is encoded to 14 kB of raw XML, whereas the raw numbers (as strings) take up around 3-4kB, and the data itself is only 1.4kB. Additionally it also takes time to create and parse this message so that XMLRPC is not suited to transfer larger data structures at high rates (> 20 Hz) and for usage over low bandwidth links like a GSM data connection. A possible solution would be the usage of a binary XML format like WBXML, WBXML was designed by the WAP Forum to save bandwidth over wireless links and to allow easy and fast parsers. Unfortunately this is not supported by existing XMLRPC implementations. Sending compressed gzip XMLRPC bodies over the network could be done easily and is directly supported by HTTP 1.1, but it adds some more processing at each end.

The normal HTTP Basic authentication is used to control

XMLRPC access to the robot. This is easy to implement but is not considered a secure method, since username and password are sent over the network in cleartext. Since the webserver and the robot are both located inside a private LAN this was not a significant concern. If direct XMLRPC access from outside networks is desired, then HTTPS should be used instead of HTTP.

The XMLRPC interface to the robot consists mainly of exported variables that can be read and set through two XMLRPC function calls. Exported variables include the Cartesian position, status, joint values and internal control parameters. For our demonstration only Cartesian position, joint values and status are used for the WAP UI.

A PHP script on the webserver accesses these variables using XMLRPC and builds WML pages by simply filling the information in predefined templates. The PHP script itself contains only the application logic, not the markup code, and so can be easily used to generate interfaces for markup languages other than WML. Figure 4 shows relationship the overview. At the moment we are replacing the simple template substitution based output generation with an XSLT [W3C, ; Martin *et al.*, 2000] based system.

The robot control software (written in C++) already included HTML and Java interfaces discussed in [Honegger and Corke, 2001]. These were built on top of a simple HTTP 1.0 server integrated into the robot software, used to:

- deliver HTML to a web browser;
- deliver XML encoded data to the JAVA interface;
- receive commands with HTTP GET requests;
- set parameters using the HTTP POST command.

Communication with the robot was changed to use XMLRPC. Extension of the robot control software was done in a few hours simply by replacing one class. Our other (C based and RTC enabled) programs can export their variables through XMLRPC by just calling two functions during startup. Wrapper software will be developed to make this new communications available to legacy applications.

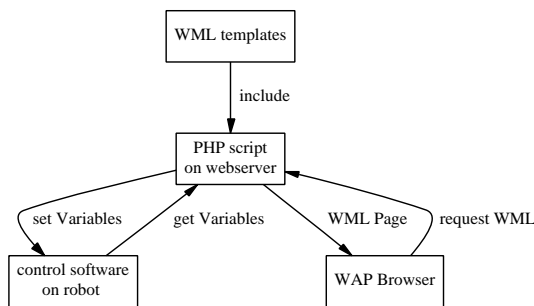


Figure 4: Script on the webserver.

2.2 Server to phone

The communication between webserver and the phone is defined by the WAP standard. Access control is again done

by HTTP Basic Authentication. In order to protect the password transmitted to the robot we need an end-to-end encrypted communications channel. In practice we cannot quite do this. We can use SSL (Secure Socket Layer) encrypted connection between WAP gateway and the webserver. We can use WTLS (Wireless Transport Layer Security) to ensure secure communications from the WAP gateway to the phone. However the gateway must decrypt and the re-encrypt and is a small potential security risk that is unavoidable, see Figure 2.

The server is implemented as PHP scripts running on the main QCAT webserver (<http://www.cat.csiro.au>).

3 UI

Special care must be taken when designing the user interface since the the “Mobile Internet” is quite different and the devices (mostly mobile phones which were designed primary as phones and not as browsers) have many constraints. [Singhal *et al.*, 2000] contains a more complete discussion about this subject as well as good chapters about the current and coming WAP standards. The major problem areas are:

- **Small screen size** Today the typical WAP enabled phone has a monochrome display of around 4 lines each of 12 characters. In the future this may change a bit but there is still a big difference between a computer monitor and a handheld device.
- **Low data rate and high latency.** The wireless communication adds a lot of latency when compared to wired network connection. A normal round trip may take around 5-10 seconds (sometimes even more), compared to roundtrips of a few hundred milliseconds on the current Internet.
- **Uncomfortable data input.** Especially on mobile phones it is hard to input text, and even floating point numbers have to be entered in a complicated way on most devices, since there is no special dot character that can be used in number entry mode (phones are designed for integer phone numbers).
- **Expensive.** Most existing services are time-based, the packet based (and volume accounted) GPRS is available from Telstra but only one GPRS Phone (Motorola Timeport P7389i) is currently available on the market and the service is still in geographically limited testing.

These limiting factors force a changed view of the user who does not want to use the device like a scaled down web. For example “surfing the web” is unlikely to happen from a mobile phone since it is not convenient. Instead the tasks must be easy and straight to the point and tailored to the current situation. Often it makes sense to offer services as an addition to other more featured and less mobile interfaces. These could be used by the user to customise the WAP interface to his specific needs.

Since our hydraulic arm was mainly used to develop and test a model-based control of hydraulically actuated manip-

ulators [Honegger and Corke, 2001], the user interface is very basic and was developed to prove the point. It allows the user to turn the robot on or off and to move the actuator to a specific position in Cartesian coordinates. The coordinates can be either entered by hand or taken from a list of saved positions.

The control flow is shown in figure 5. Information and actions are offered by the frontpage only if they are currently available. The thick lines indicate an action on the robot or some other device outside the browser and links without processing on the “server” side are shown as thin lines.

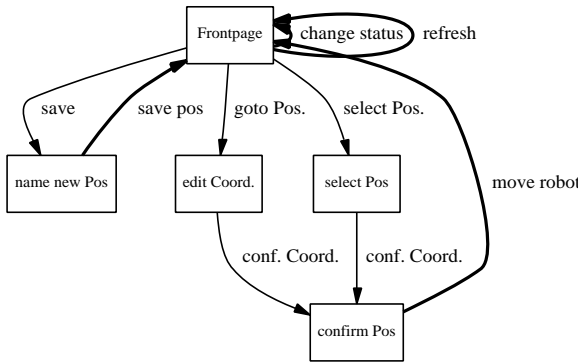


Figure 5: Control flow of the sample interface.

A quick look at the diagram shows that lots of page navigation can be done without affecting the robot (for example: Frontpage to edit Pos.). One WML (Wireless Markup Language) file consists of a single “deck” that can hold multiple “cards” (one card is roughly the same as a page). These can be used for wizard style entry of data, without having to fetch the pages individually (which is slow). In our case we use one deck with two cards and 3 single card decks, shown in Figure 6. The *confirm Coord.* card doesn’t need to be in both *Goto Coord* and *Goto Pos* decks, we can simply link from *select Pos.* inside the *Goto Coord.* deck to the *confirm Coord.* card inside the *Choose Pos.* deck.

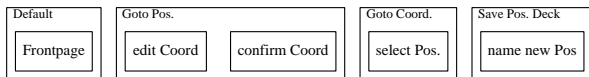


Figure 6: Decks are used to group cards.

Some screenshots of the interface are shown in Figures 7 and 8 for two different microbrowsers (Openwave SDK (left) and Nokia WAP Toolkit (right)). This highlights another significant problem with WAP which is that the look and feel is not easily controllable and is largely dictated by the phone. Figure 7 and Figure 8 show the top of the Frontpage, the selection of a known Goal is shown if Figure 9.

4 Conclusions

It worked! However the current technology is very limited and suffers from numerous problems that make the solu-

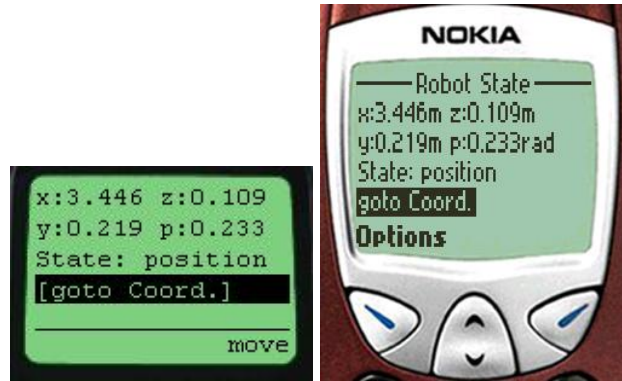


Figure 7: Frontpage

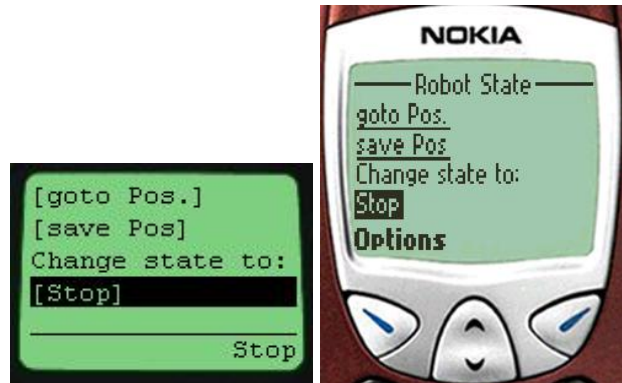


Figure 8: Frontpage, second screen.

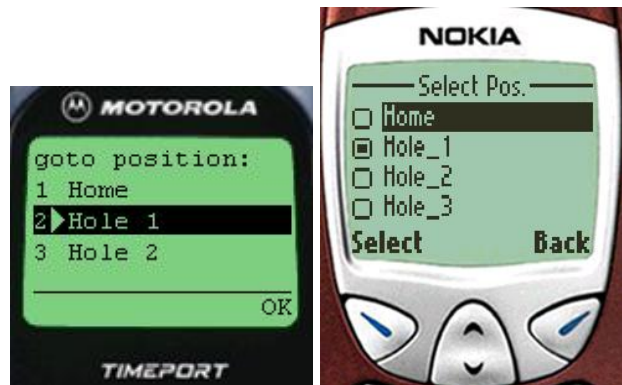


Figure 9: Selection of a known goal.

tions not very intuitive. The most important ones have been discussed above in Section 3.

Additionally there are some gaps in the WAP 1.1 standards, for example the WML 1.1 (which is the version supported by all browsers) specifies no way to force a reload of a URL once it has been cached, one must resort to some small WMLScript to create unique URL’s in that case. To add to this, every browser is different, so some constructs work very well on Nokia’s browser and not on the Openwave browser (used by most other phones) and vice versa, because WML does not define the look&feel but the intention and semantics of its constructs. Thus every browser

manufacturer has some freedom (which is needed, because WAP caters for a wide range of devices) while implementing WML. To make sure a reasonably complex page works fine on different browser it has to be tested on every browsers (see also [Openwave Systems Inc, 2001] and [Singhal *et al.*, 2000]). [Openwave Systems Inc, 2001] even encourage the developer to tailor content for the different browsers.

5 Future work

Future work may include:

- **Graphics.** First tests have been made with including small graphics into the application to give a visual feedback of the robot arm. Given the very low resolution of the typical mobile phone display, simple figures like a series of 3 views (top, side, front) of the robot arm seem to make most sense. PHP supports dynamic generation of WBMP images and a small script can create these images on the fly from the current joint values.
- **XSLT.** The current version uses templates to generate WML. Switching to XSLT based transformations would simplify the scripts and provide more flexibility (eg. creating a HTML & WML page directly from the same application). These transformation could take the robots XMLRPC response and create the output directly from the response. We have written XSLT for most of the needed transformations already.
- **Generic UI description for robots.** Create a generic description of a robot that allows the UI to adjust to changes or a new robot automatically based on the description provided by the robot. The description could be stored on the robot and fetched by the UI to create an interface for this robot. This method is not limited to WAP UI's but could also be used for other (possibly GUI) interfaces.
- **Open-pit excavation.** We are designing a gateway to allow online access to the status of machines such as shovels and draglines via WAP.
- **Autonomous outdoor vehicle.** This is a new project (jointly with UNSW and QUT) to look at reactive navigation strategies for mobile robots in semi-structured surface environments. The WAP interface could be used to report vehicle status, accept commands to travel to defined points in the environment, and to transmit low-resolution images from the vehicle's camera.

Acknowledgements

We would like to thank our colleagues Pavan Sikka, Jonathan Roberts and Elliot Duff for help and valuable hints.

References

- [Honegger and Corke, 2001] Marcel Honegger and Peter Corke. Model-based control of hydraulically actuated manipulators. In *ICRA 2001*, 2001.
- [Martin *et al.*, 2000] Didier Martin, Mark Birbeck, Michael Kay, Brian Loesgen, Jon Pinnock, Steven Livingstone, Peter Stark, Kevin Williams, Richard Anderson, Stephen Mohr, David Baliles, Bruce Peat, and Nikola Ozu. *Professional XML*. WROX, 2000.
- [Openwave Systems Inc, 2001] Openwave Systems Inc. GSM Application Style Guide. <http://www.openwave.com>, 2001.
- [Roberts *et al.*, 1999] Jonathan M. Roberts, Peter I. Corke, Robin J. Kirkham, Frederic Pennerath, and Graeme J. Winstanley. A real-time software architecture for robotics and automation. In *ICRA*, pages 1158 – 1163, 1999.
- [Singhal *et al.*, 2000] Sandeep Singhal, Thomas Bridgman, Lalitha Suryanarayana, Daniel Mauney, Jari Alvinen, David Bevis, Jim Chan, and Stefan Hild. *WAP - The Wireless Application Protocol*. Addison Wesley, 2000.
- [Taylor and Dalton, 1997] Ken Taylor and Barney Dalton. Issues in internet telerobotics. 1997.
- [Taylor, 2001] Ken Taylor. Mobile SCADA infrastructure. <http://mobile.act.cmis.csiro.au/>, 2001.
- [W3C,] W3C. Extensible stylesheet language. <http://www.w3.org/Style/XSL/>.
- [Winer, 1999] David Winer. XMLRPC Website. <http://www.xmlrpc.com>, 1999.