

Smartcards for Ubiquitous Web Services

Changkyu Kim, Karthikeyan Sankaralingam, Youngin Shin
University of Texas, Austin
{ckkim, karu, codeguru}@cs.utexas.edu

May 5, 2002

Abstract

Web services have now become the hottest topic in the software market. Most web services have the capability of tailoring their output based on the client. Dynamic content on the web is then created by using multiple web services and using different data from different services. Smart cards have been predominantly used for authentication and in some niche areas like banking. However they are getting increasingly powerful in terms of computation and programmable smarts are currently available. In this project we propose using smart cards as a vehicle for ubiquitous computing using web services and propose a system that can be implemented in hardware and software to realize this goal. The key benefits being that state information is saved entirely in the smart card - making servers stateless thereby increasing their scalability and enhancing privacy/security. We developed a working software prototype to show that the system can be built on a Schlumberger Cyberflex smart card and that it works as model for ubiquitous computing. We propose a hardware strawman design which can be used as a universal communication module that can be plugged into any consumer device making it internet-visible with the smart card being used both for storing all the user state information and for computation.

1 Introduction

Web services have now become the hottest topic in the software market. Many software vendors including Microsoft, IBM, Sun and HP are now developing and deploying their own web service products [8], [4], [3], [9]. Web services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes [11]. In most cases, web service clients are *web servers*, rather than actual users. It is then the service client's responsibility to create dynamic content using one or more web services. In contrast to the platforms of web service servers or web servers, those of actual end users vary from typical desktop PC's to mobile devices such as PDA's or cellular phones. Some web service platform, such as Microsoft.NET, therefore support the ability to tailor content according to the end-user's platform.

So far there has been little attempt to use smart cards as one such platform. Although smart cards are currently used to authenticate users they have the computation power to do much more than that. Though limited in speed and space, these chips contain microprocessors, ROM and memory and run their own COS(Chip Operating System), on which various applications may execute. Hence, with this computing power, it may be possible to use Smart cards as mobile access points to web services; Smart cards can store user information, invoke web services and process temporary service results.

In this project, a general model of accessing web services using Smart cards is proposed and a software prototype is developed. Users, in this model, should be able to access web services from any place with their Smart cards. Furthermore, users do not require any high performance computing mobile devices such as PDA's. Rather, simple smart card readers will be enough. We also propose a specification for a communication module that can be plugged into any consumer device making it visible on the internet. The contributions are three-fold; 1) Manufacturers of consumer devices do not need to be bothered with the implementation of web services, network stack and XML processing. 2) The smart card is used as the storage point making servers stateless - thereby increasing their scalability and enhancing privacy of users.

3) Communication is entirely handled by the communication module - freeing device manufacturers from implementing various communication components in their device.

In this paper, first we will briefly cover the history of Smart cards and web services and provide our motivation for tying the two in section 2. In section 3 we describe our architecture for ubiquitous computing using smart cards and web services. In section 4 we describe the details of a web services description language. In section 5 we describe our software prototype built as a proof of concept. In section 6 we provide our conclusion and a discussion about future work.

2 Related work

Based on their main purpose of user authentication, most kinds of smart card applications are related to on line security and payment. In the past, when memory cards were used, user keys were stored on the chip for user authentication. Later, with the increase in the computing power of smart cards simple applications related to cryptography, banking, and authentication were developed. Recently Schlumberger, one of the largest smart card companies, released a Javacard, which has a built-in Java Virtual Machine and has the capability of running Java applets [6].

Smart cards have also been doubling their memory size and chip operation clock speed every year since they were introduced in 1996 – when the chip speed was only 3MHz with 4kbytes of memory. They now operate at 50MHz with 512Kbytes of memory and are expected to run at 400MHz with 100MBytes of memory by 2006. It is also expected that more devices such as fingerprint recognizers, small display devices, and film batteries, will show up on the card. As part of the General Service Administration SmartData initiative several smart card projects are being pursued [2]. Javacards have been used to do complex tasks and in fact a full fledged web server has been implemented on a GSM smart card [12]. Urien [10] proposed an innovative method where smartcards run well defined server and client applications to adapt to internet applications. A comprehensive list of smart card projects can be found in [1].

3 Architecture

We propose an architecture for utilizing smart cards as the medium of ubiquitous web services. The overall scenario we visualize is as follows: *Users get most of the computation done through web services. A web service workflow description language specifies the list of web services to be executed, the manner in which they share data between themselves and the control flow amongst these web services. The flow of information between web services and user devices is also specified. Users should be able to “suspend” execution at anytime and resume execution at any time, at any other place.*

Figure 1 shows our system organization. The behavior of the different blocks is as follows:

- Web services are implemented using any of the standards provided by the different software vendors. For our project we used the Microsoft.NET platform.
- The user device here represents any consumer device that needs to access web services, or it could be a web browser showing a web page built using different web services.
- The communication module orchestrates communication between the web services, the smart card and the device. If the communication module is written entirely in software, a translation layer needs to be used to interface with the device drivers of the card reader and convert text message into smart card commands to communicate with the smart card.
- The smart card is used for two purposes - storage and computation. The smart card is used to store the web services that need to be accessed. These web services are stored in a web service workflow description language. These web services are then accessed using SOAP calls. The smart card also stores the information returned by the web services in the storage space. The information is saved in named records which are then used to pass information between web services. By saving state on the smart card we allow easy migration for users between sessions where these sessions could be at different access points. An interpreter for the web service workflow description language is written as a

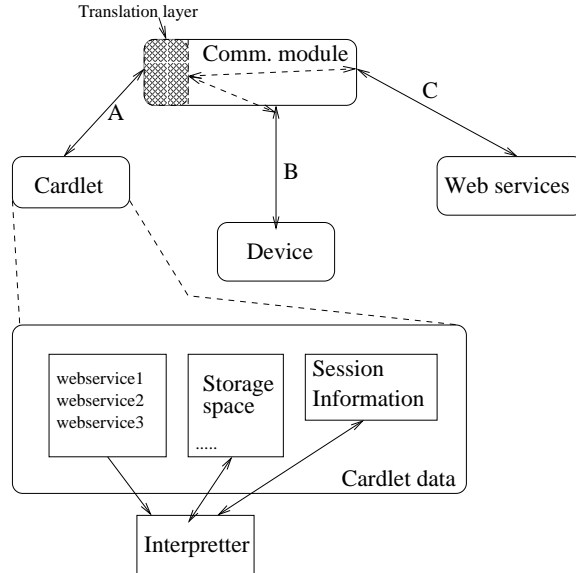


Figure 1: System overview. A) Plain-text messages. USB based communication through the card reader. B) Communication with user device; device-dependent. Use operating system communication services if necessary. C) HTTP/SOAP - TCP/IP based communication.

smart card cardlet and runs on the card. This interpreter (referred to henceforth as cardlet interpreter) examines the web service calls and generates SOAP calls to be sent to the web service servers. These calls are first sent to the communication module and from there forwarded to the web service servers. The storage space resides in the data space of this cardlet interpreter.

We first outline two example scenarios using our architecture and then describe possible hardware and software only implementations. The hardware implementation section describes possible solutions using data obtained from processor specification sheets. We examine 2 embedded processor cores.

3.1 Examples

We provide here two simple example uses for our system and how the different pieces fit together. Consider the first example of an “active-printer” which is a postscript printer with an attached keyboard input device and our communication module. The printer itself could be installed in a place like Kinkos for example. The user types in a document name as a URL in the keyboard. The document itself could be a Postscript file, HTML file, a Word document etc. What then happens is as follows:

- The communication module first makes a HTTP connection and fetches the document.
- It then sends this document to a file convert web service and converts this file to postscript format.
- The postscript file is then sent to the printer and is printed out.

The pseudo-code is shown in figure 2. The syntax — every call is either a webservice call or a device call. The first argument is the input and the second argument is a record name where the output from the web service or device is stored. The first argument has 2 parts - a command and the data record name - separated by a “,” in the example. Device calls are sent to the device and webservice calls are converted into HTTP GET requests and sent to the network layer. Note here that the user can suspend execution at any point and continue on any other active-printer where he left off. The records are stored in a non-volatile storage in the smart card in the storage space shown in figure 1.

Consider the second example of a air-ticket purchase terminal. The terminal itself consists of a display device and our communication module orchestrating communication. Consider the 3 simple steps in a ticket

```
device_call [getinput] <R1>
webservice_call [fetch, <R1>] <R2>
webservice_call [convert, <R2>] <R3>
device_call [print, <R3>]
```

Figure 2: Pseudo code for webservice workflow description

purchase - search for flight availability, find seating availability, make payment. In the case that the user needs to be put on a waiting list, then we want to defer payment until the ticket is confirmed - all the intermediate information is saved on the smart card instead of the servers. Also each of these 3 services is offered by different web service providers. The pseudo-code in figure 3 shows how this can be done. The records encapsulate all the information that needs to be passed between web services calls. Note also that each web service is totally oblivious to what processing is done on the record i.e. in the example the web service “findseating” does not have to talk to web service “makepayment” and the two need not share any common communication format. Instead the cardlet interpreter matches information from one web service to another and makes sure that they can pass data between each other.

3.2 Software implementation

In a software implementation of our system the communication module runs entirely in software and the device has to be a software component like a web browser and it should interface to the communication module using the operating system¹. When processing web service calls, the smart card cardlet interpreter generates SOAP calls which are sent to the communication module as plain text strings. The communication module receives these messages and makes the actual HTTP network connections. XML data returned by the web service servers is returned back to the cardlet interpreter which parses the XML data. When processing *device calls* plain text messages are sent to the communication module which forwards these messages to the device. In section 5 we describe our implementation details.

3.3 Hardware implementation

In a hardware implementation of our system the communication module is developed as an independent hardware component. It has two I/O ports, and a smart card slot. One of the I/O ports is a USB port² which is connected to the consumer device. The second I/O port is either a standard 10 Base-T Ethernet or 802.11. The module also incorporates a standard smart card reader interface and an embedded communication oriented processor core. The requirements of the embedded core are - TCP/IP processing, Ethernet connectivity, HTTP/XML processing capability and ability to interface with USB.

There are several embedded cores that provide this functionality. The Motorola MPC823 core [7] provides all of this functionality plus added LCD and video controllers. It provides for 2 additional serial communication channels also. The RISC core on this processor is a 32-bit processor with 32 32-bit registers and a 2KB instruction cache implementing the PowerPC ISA. This core can easily be programmed with network processing required. Uvicom provides a system-on-chip solution integrating USB, TCP/IP-HTTP processing and Ethernet on a single chip called the IP2022 Internet Processor [5]. The IP2022 CPU provides 64Kbyte flash program memory and 16Kbyte RAM which can be used to program the RISC core to provide the network processing. In the event that more data memory is required the core can access up to 128Kbytes of external memory.

The IP2022 processor has been used before for enabling communication between consumer devices in a system-on-chip implementation called PhantomServer. The authors are not aware of its use as a smart card

¹The device can connect to the machine through any sort of I/O port and use device drivers to interface with the communication module.

²This port could be any type of port really. We choose USB because of its attractive properties of bandwidth (up-to 1Mbps/sec) and low power. Moreover USB is by far the most popular protocol for PC devices. We believe it should be suitable for general consumer devices.

```

device_call [getinput] <R1> /* get destination, date etc */
webservice_call [getavailability, <R1>] <R2>
device_call [showavailability, <R2>]
device_call [getchoice] <R3> /* get users choice */
webservice_call [findseating, <R3>] <R4>
if (R4.waitinglist) {
    device_call [waitinglist, <R4>] /* inform user */
    /* we are done here. Record R4 encapsulates all information
       user checks back later at any access point and he can resume from
       here */
    while (true) {
        webservice_call [checkwaitinglist, <R4>] <R5>
        if (R5.waitinglist) {
            device_call [waitinglist, <R4>] /* inform user */
            /* R4 and R5 encapsulate all information. User can terminate
               here and resume later */
        } else {
            break;
        }
    }
}
device_call [showdetails, <R4>]
webservice_call [makepayment, <R4>], <R5>
device_call [showpaymentconfirmation, <R5>]
device_call [printticket, <R4>]
device_call [printpayment, <R5>]

```

Figure 3: Pseudo code for ticket purchase example

interface enabling ubiquitous computing and saving all the state information with the user instead of on centralized servers. Our proposed hardware model has 2 important advantages:

- It frees consumer device manufacturers from having to design a communication layer and the network stack. It also frees the device manufacturers from have to design HTTP/XML processing. Devices which use this module could be as simple as vending machines that need to connect to the internet to a complex air-ticket dispensing systems.
- It frees consumer device manufacturers from having to design mechanisms for storing state information and simultaneously frees servers from having to store state information and pass it along between services.

4 Webservice language description

Traditionally, a web application is made available through a web server and can be accessed by client browser. But instead of providing information via dynamically generated HTML, the server generates a series of function calls through HTTP and uses XML to transfer messages which are consumed by the clients. Therefore, a web service can be defined as an interface that describes a group of operations which can be accessed by XML messages. This interface hides the implementation details of the service by removing restrictions on what platform it is running on or what language it should be written in. The second characteristic of the web service is composability. The developer can develop new web applications by searching for necessary web services and combining them in the right order. Some long-running transactions like purchasing an airline ticket and managing inventory in a warehouse consist of a series of the web service components. For example, managing inventory in a warehouse consists of a credit validation service, an inventory management service and a customer accounting activity.

In this project, we focus on the latter characteristics of the web service by developing a language which describes the ordering of the multiple web service components. Our language is derived from IBM's web service flow description language called WSFL [14]. The reader is encouraged to read the WSFL specification for details as we are limited by space into going into a detailed description. The second component in the language specification is describing a single web service. Again there are several implementations. The W3C standard for this is WSDL which defines an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages [13]. Again the reader is encouraged to read the WSDL working draft for details.

5 Software prototype

Our software prototype was written as 4 separate modules - the cardlet interpreter, the communication module, the user device and the web services. Since this is a full software implementation we needed one extra software component that would actually interface with the smart card and send/receive smart card commands to/from the USB card reader. We call this module the smart card reader application and this sits as a filter between the smart card and the communication module labeled as the translation layer in Figure 1. We describe each of these in detail below.

5.1 Language specification and interpreter

The storage model, language specification and session information for the cardlet interpreter were simplified because of the lack of processing power and space in the smart card. Our storage model consists of a set of 32 *byte registers* half of which are used as predicate registers. The session information is entirely captured by the values in all of these registers and a program counter - which points to which point in the web service language we are in. Each instruction in our is fixed length - 6 bytes each.

We defined the format of each instruction for describing a web service workflow as shown below. The instruction fields are explained in Table 1.

Target(1byte) | MID(1byte) | Imm/Reg (1byte) | SRC (1byte) | DR (1byte) | PR (1byte)

Notation	Meaning
Target	Denotes the target for an instruction execution. 0 represents a device and 1 represents a web service call.
MID	Message ID; The communication identifies the message ID and sends the corresponding request to either a device or a web service server.
Imm/Reg	Denotes the type of the source data. If Imm/Reg == 0, SRC contains the immediate value for the source data. If Imm/Reg == 1, SRC contains the register location of the source data
SRC	Source data value; either the immediate value or the register location.
DR	Register location for the destination data. The return value from either a device call or a web service call is stored.
PR	Predicate Register; If PR == 1, the current instruction is skipped and the program counter is increased

Table 1: Instruction Format

The following is the request message format between the cardlet interpreter and the card reader application

`Target(1byte) | MID(1byte) | data (1byte)`

We implemented the cardlet interpreter for this simple instruction set using Cyberflex Access SDK 4.1. The cardlet interpreter is written in Java and runs on a Schlumberger 32K cyberflex smart card. Our simple web service workflows are described using this instruction set. After the smart card is inserted into the card reader, the card reader application uploads the set of instructions into the cardlet interpreter. The cardlet interpreter then starts execution when it receives the start message from the card reader application. After “executing” each instruction it sends the request message to card reader application and waits for a reply from card reader. When it receives a reply, the return data is stored into the corresponding register denoted by the instruction and the next instruction is “executed”.

5.2 Smartcard reader application

The smart card software reader application communicates with the cardlet interpreter executing in the smart card. The card reader software is written in Java and uses the smart card reader device drivers to communicate with the card through USB. The Schlumberger SDK provides a Java inter-operation layer for developing smart card readers using Java. The reader application sends and receives smart card commands to/from the smart card. The card reader application has a GUI for interfacing with the user. It has the capability of controlling the smart card and sequencing the instructions interactively. The card reader software module also has the capability of examining the storage space used by the cardlet interpreter - all the registers can be examined and the program counter can be obtained. A new program can be typed in and uploaded into the smart card. The card reader application connects to software communication module through sockets and basically forwards messages from the card to/from the communication module which interfaces with the device and web services. A continuous log of messages to and from the card is displayed for debugging purposes.

5.3 Communication Module

The communication Module(CM) sits in the middle of all components, talking to the smart card reader application, devices, and web services. It is stateless and just sends and receives all messages between those entities. Currently, it is implemented with Visual C++, and runs under Microsoft Windows environment, as a socket server waiting for messages from the card reader application.

From the card reader application, the CM receives two different messages; 1) **web service call**, and 2) **device call**. When it receives a web service call, it sends an HTTP/SOAP request message of the following format.

```
GET /UBQuards/TestWebService.asmx/TestWebMethod?nData={data}\n\n
```

where `data` is an integer value. It is a simple HTTP GET request calling `TestWebMethod`. Then, the web service server responds with the following message.

```
Received data : HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Mon, 04 Mar 2002 22:57:13 GMT
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 89
```

```
<?xml version="1.0" encoding="utf-8"?>
<int xmlns="http://tempuri.org/">1</int>
```

We parse the above response to extract the return value of the web service, which is 1 in this example. The CM sends this one byte data back to the card reader application that requested this web service call. Since the smart card's computation power is limited we move the XML parsing and SOAP call generation responsibility to the communication module in our implementation although our proposed architecture model was to do all of this processing entirely in the smart card.

From the smart card reader application, the CM receives device calls also which tell the device to either respond with its value or write a value to its register. The CM forwards both types of messages to the device, the format of which is shown below.

```
[ read | write ] [ data ]
```

The device responds to a `read` message with data. On receipt of write messages, the data is written into the device's register.

5.4 Device

Devices are implemented as socket server applications that respond to device call messages from the CM, modeling real hardware behavior. Currently, we have one device running at `localhost:2000`. Values received with write messages are displayed on a window. The device has one register value that can be modified by the user. On receipt of a read message this register's value is returned.

5.5 Web services

We used Microsoft.NET as our web service platform. It runs at `codeguru.csres.utexas.edu:8080`. Our prototype web service is accessible at `/UBQuards/TestWebService.asmx`. Currently, this web service receives an integer, increments it and returns the incremented value. All communication between CM and web services are done through SOAP/HTTP as previously explained.

6 Conclusion and Discussion

In this project we examined the possibility of using smart cards as a medium for providing ubiquitous web services. We developed a software prototype that has 4 components. 1) A java cardlet interpreter that implements a stripped down web service workflow description and executes on the smart card. 2) A java card reader module that interfaces with the cardlet interpreter and provides a GUI for user interaction. The card reader module also acts as a communication bridge. 3) A few example web services and 4) a prototype "user device" written as a software component. The card reader module orchestrates communication between the smart card, the device and web service according to our specification.

We also provided a discussion on how a full hardware communication module can be built to provide ubiquitous computing using smart cards. As future work, it should be interesting to pursue development of

the required network processing on one of the embedded cores and examine how the serial communication channels can be programmed to act as a smart card reader. On the hardware side it is an interesting possibility to see if all of the communication module can be integrated into the smart card itself. WSFL is only the first attempt at describing web service work flows. We need to examine in detail if it is powerful enough to capture the majority of state information that need to communicated between web services. Strong encryption schemes are required if all user state information is saved on the card - to prevent the user from tampering with the information!

We came to understand that smart cards as of now have computation limitations that prevented us from doing the full XML processing on the card. Also we did not implement any encryption schemes to prevent the user from modifying state information on the smart card. We believe in the future this should be possible and ubiquitous computing using smart cards has several benefits.

References

- [1] Center for information technology intergration: Projects: smart cards. <http://www.citi.umich.edu/projects/smartcard/>.
- [2] GSA Smart Card Project Viewer. http://egov.gov/scripts/sc_viewer.asp.
- [3] HP Webservices platform 2.0. <http://www.hpmiddleware.com/webservices/>.
- [4] IBM Webservices. <http://www.alphaworks.ibm.com/webservices>.
- [5] IP2022 Internet Processor: Data Sheet. http://www.ubicom.com/pdf_files/processors/IP2K-DDS-IP2022DS-12.pdf.
- [6] Schlumberger Javacard. http://www1.slb.com/smartcards/infosec/cyberflex_access.html.
- [7] Motorola 823 and 823e Communication Processors: Data Sheet. <http://e-www.motorola.com/brdata/PDFDB/docs/MPC823FACT.pdf>.
- [8] Microsoft.NET. <http://www.microsoft.com/net>.
- [9] Sun Open Net Environment - Sun ONE. <http://www.sun.com/software/sunone/>.
- [10] P. Urien. Programming internet smartcard with xml scripts. In *Proceedings of the International Conference on Research in Smart Cards, E-smart 2001*.
- [11] Web services - the web's next revolution. <http://www-105.ibm.com/developerworks/education.nsf/webservices-onlinecourse-bytitle/BA84142372686CFB862569A400601C18?OpenDocument>.
- [12] How to turn GSM SIM into a Web Server, EURESCOM P1005 Project Internal Result. <http://www.eurescom.de/public-webspace/P1000-series/P1005/websim/websim-howto.pdf>.
- [13] Web Services Description Language (WSDL) 1.1. <http://www.w3.org/TR/wsdl>.
- [14] Web Services Flow Language. IBM Webservices toolkit.