LIGHT

(xmL-Innovative Generation for Home networking Technologies)

Deliverable D2.2: Requirements analysis for footprint and power constrained devices

---

Editor(s):	*Vittorio Miori, Luca Tarrini, Rolando Bianchi Bandinelli*
Document Name:	*Deliverable D2.2: Requirements analysis for footprint and power constrained devices*
Date:	May 23, 2005

Scope

The ubiquitous nature of XML makes it a data interchange format promising universal access to any XML-enabled device by any XML-enabled application. XML has recently made inroads into the data-oriented world of machine-machine interaction, most prominently in the form of SOAP and Web services. In fact there is a greater demand to use Web services technology as a way to deliver content to small (wireless) devices. But this opportunity has created new challenges for embedded systems. In fact rarely these systems have enough memory and processing power to run an HTTP server, SOAP engine, and XML parser in comparison to the normal desktop PC. In addition, the quality of wireless networks is much lower than that of fixed networks, especially considering bandwidth and latency. Therefore the processing and memory limitations of embedded devices may severely compromise the usability of XML, unless several optimizations are considered.

Executive Summary

This deliverable carries out a significant number of technical studies on the applicability of Web services in distributed embedded systems, that is investigate if resource-limited devices are capable and fast enough to run Web services. The main purpose of this document is to summarize the results of the evaluation of existing technologies that could be of interest to WSED (Web Services for Embedded Devices) architecture.

Our approach isn't to develop a new prototype implementation of a tool for Web services but to experiment the modern solutions for testing them on our architecture. In this work we have concluded that Web services are applicable in distributed embedded systems, but applying it introduces an overhead, in terms of CPU, memory, and network usage.

1.0 Introduction

An embedded system [1] is a combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a specific function. An embedded system is any system consisting of one or more embedded devices, which contains software that interacts with an external environment, such as a human user, physical objects and sensors, in order to solve a dedicated task. A good example is the microwave oven. Almost every household has one, and tens of millions of them are used every day, but very few people realize that a processor and software are involved in the preparation of their launch or dinner.

The embedded systems are becoming smarter and are increasingly shipping with network connectivity. It would be extremely useful to remotely configure or monitor an embedded device, to have an embedded system integrated with other systems on a home network. It seems inevitable that the number of embedded systems will continue to increase rapidly. Already there are promising new embedded devices that have enormous potential: light switches and thermostats that can be controlled by a central computer, intelligent air-bag systems, palm-sized electronic organizers and personal digital assistants and so on.

At the same time a new technology, XML Web Services, is becoming a powerful mechanism for connecting distributed and intelligent systems. In a near future, information and services should be available everywhere and anytime and with Web services with anyone or any machine. As discussed in

the Deliverable 2.1 [2], all of these intelligent embedded systems are capable of service description languages that allow platform independent description of the service interfaces, service registration mechanisms that allow services advertise themselves once deployed, service discovery mechanisms that allow user applications to find and choose appropriate services at run time, and authorization, authentication and accounting mechanisms to establish trusts among service providers and service users. SOAP, WSDL, UDDI, and WS-* are initially created for hardware with relatively a lot of memory and CPU resources. For example SUN is working on the mobile counterpart to add Web services to their J2ME platform.

In synthesis, we can say certainly that Web Services and embedded systems are hot topics nowadays. As both technologies are still not in their mature form and the applicability of Web Services Architecture as middleware technology to integrate distributed embedded systems in a loosely coupled environment arises whether both these technologies must be combined.

In the LIGHT project, the devices of interest are:

- Embedded control systems, including consumer devices, intelligent sensors, and smart home controls.
- Communication oriented devices such as digital cell phones and mobile devices.

The target platforms for this use case include a broad range of devices (PDAs, handhelds, smartphones, TV set, Set-to Box) that limit code size to 64K and heap size to 230K. For the reasons above, a scaled-down standard for Web services catering specifically for low-powered devices are required, and it has to be also compatible with the original architecture.

The problem of hosting Web services in an embedded system entails the high requirements of resources in terms of processing power, memory availability, disk usage, and network bandwidth. This is due to the overheads associated with execution of applications, data processing and marshalling techniques, and communication protocols. For example, the XML-based messaging is at the heart of the current Web services technology. But these messages are very verbose and require more processing than other protocols, and they are therefore not well suited for a domain having small resources. An aspect in contrast with the wired network, is that the main difficulties in wireless communication are lower bandwidth, which limits the amount of data that can be sent, and higher latency, which limits the number of round trips that a protocol can effectively make within a communication.

In the LIGHT project, the requirements of WSED architecture must be able to run on top of limited-resources devices. It will be essential to distinguish between controller devices and services devices because the capabilities could change. For this last, the component software footprint is in the range of Kbytes while the typical hardware resources of controllers are 32 bits microprocessor (16 and even 8 bits microprocessor are possible), with 128 Kbytes of Flash memory and 64 Kbytes of RAM memory.

2.0 Design Issues

Each new technology seems to bring with it a whole new set of design trade-offs and issues. Web services are no different and the combination with embedded systems increase the problem. Limited resources, severe cost constraints, and operational considerations combine to create a complex set of engineering trade-off. To make devices useful as services to big machines and web services accessible from the small devices it seems logical to speak the common language, that is XML. As if specialized protocols are more efficient, in many situation they mandate complicated and costly proxy or gateway. A discussion of some of the purported issues surrounding Web services follows.

Verbosity: One of Web services main drawback concluded from the studies is the use of non efficient XML messages. Character based XML data takes more space in comparison to binary encoded message and parsing complex XML data is processor intensive. In recent years, there has been much work in measuring the performance of SOAP as a messaging protocol. For example, some experiments [3] proved that the bottleneck for a high-performance SOAP toolkit is the marshaling and unmarshaling overhead of converting floating point data to text and vice versa. Implementing the networking, XML parsing, and SOAP encapsulation can be a bit intimidating but there are tools designed to abstract away much of the complexity. We have analysed a number of alternative serializations developed to deliver XML content to small devices and in the next section we discuss on some of these tools and however many of these are not interoperable.

The verbosity of the text-oriented HTTP solution has multiple system impacts, affecting RAM usage, bandwidth requirements, and operating costs. Because XML is a text based language, providing the significant advantage of making platform independence, the downside is that text-based systems are inherently less efficient than a binary system. This leads to more data being transmitted and larger buffers required to both prepare outbound messages and receive inbound messages.

Resource Constrained: One of the main features that defines an embedded systems is that you don't have enough memory, processing power, or some other resource to do what you want. The thought of adding an XML parser and SOAP encoding engine to a system seems problematic at best. In many cases that might be true – a full-blown XML parser can easily add over 180Kb of code. The table describes the technologies and the features used to build Web services on Windows CE.NET.

Component	Size
XML/HTTP	15 KB – this is pulled in by the SOAP Server component
COM (inprocess)	111 KB
SOAP Server	270 KB – This include HTTPD server and XML/HTTP
ATL	44 KB
HTTPD Server	61 KB – this is pulled in by the SOAP Server component

Table 1. Web Services Components on Windows CE.NET

A feature about Windows CE is the support of SOAP Toolkit functionality that consists of a client-side component that allows an application to invoke Web service operations and a server-side component that maps invoked Web service operations to COM object method calls as described by the WSDL and WSML (Web Services Meta Language) files. The WSML file provides information that maps the operations of a service to specific methods in the COM object.

Fortunately, many of the features of the XML standard are not required for SOAP encoding. A well-pared XML parser that fully supports SOAP can be under 20 Kbytes in size. For those truly constrained devices that can't handle even that small amount of code, Web services can also be invoked without using SOAP at all. The WSDL specification also allows a port to be bound to an HTTP POST or GET verb targeted at a specific address. This allows the invocation of a Web service to be a simple as sending an HTTP POST or GET URL-encoded request to a specific URL. Web service frameworks targeted towards embedded systems exist that provide an HTTP client and server and support URL encoding while consuming under 5Kb of code space.

8- and 16-Bit Support: Many platforms that claim to support embedded systems really mean they support 32-bit systems that run a real-time operating system (RTOS). In other words, they support only

about 15% of the processors out in the field. What about the other 85% of the processors out there, are they doomed to proprietary networking systems at best or standalone operation at worst? Certain proprietary networking software companies that say an 8-bit processor simply can't handle a full-blown TCP/IP stack have been proven wrong (there are at least a half-dozen 8-bit TCP/IP solutions available today). If the view of Web services as a distributed computing model is subscribed to, many new applications and features become possible. For example, addressing the slowly deteriorating performance of an industrial compressor can be a complex statistical calculation based on historical data and current environmental factors. If a more capable application server has access to this information it can handle the calculations, generate a service request for a field technician, and also provide feedback to the device, such as going into a lower rpm mode to minimize vibrations in order to extend the life of the asset until help arrives. This gives the end customer continued use of that asset but perhaps at a lower capacity. Eight-bit applications that can invoke two different Web services over a TCP/IP stack (with full gateway routing support) have been demonstrated in under 20Kb of code on an 8051 type architecture. Consider the impact of even the most mundane devices, such as a compressor or smart sensor, being able to participate in the community of a business enterprise.

3.0 Evaluation

In this section we illustrate the main features of gSOAP, kSOAP, cSOAP tools and MMLite system architecture. MMLite is not only a simple tool for the development of Web services on embedded systems but it is also a complex architecture system on which we are going to experiment WSED. Because of limited amount of memory, the traditional packages such as Xerces [4] or Axis [5] are far too large and resource-intensive to work on microdevice. For example, Xerces.jar file is over one megabyte in size. It is impossible for our target devices. To transport Web services on small (wireless) devices, Sun is currently in the stage of finalizing JSR172 [6], a specification that address the use of XML, SOAP, and Web services on these devices.

3.1 gSOAP

Within the context of LIGHT, we agree our interest on gSOAP [7] project. In this section, we gives a brief overview of this tool. The first stage illustrates the gSOAP design characteristics while the second stage introduces the runtime optimizations [8] for developing light-weight Web services for embedded devices. The optimizations in gSOAP are included in the IBM alphaWorks Web Services Tools Kit Mobile Devices (WSTKMD) [9]. The gSOAP Web services is an open source development environment for Web Services and provides a SOAP/XML to C/C++ language binding to ease the development of SOAP/XML Web services and client application in C and or C++. Nowadays the specification is the 2.7.2 and it include an interesting feature such as SOAP-over-UDP. This SOAP binding will result interesting when we will develop the Discovery process in WSED architecture. GSOAP is portable to most platforms, Unix, Linux, Windows, Mac OS X, Pocket PC, Palm, Symbian, and cell phones. Then it could be imported on PC or embedded system.

The main design characteristic include:

- RPC compiler generates code for XML serialization of an application's native C/C++ application data types, including graph-based data structures.
- Compiler generates compact code and SOAP/XML engine has a small memory footprint.

- It is important the support for pure C because many embedded systems kernels and systems applications are developed in C.
- The efficiency of the parsing is enhanced by using XML pull parsing.
- A service can be defined in WSDL file or gSOAP header file.

Figure shows the development and deployment of gSOAP services.

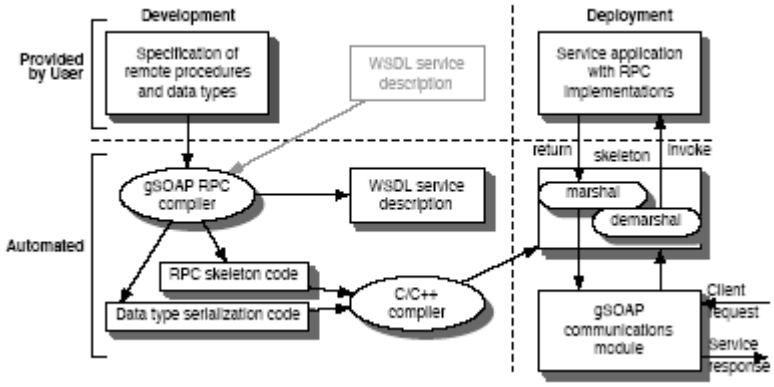


Figure 1. Development and deployment of a Service

It is tedious task to write a stub routine in C or C++ especially in the situation in which the input or output parameters of a remote method contain elaborate data structures. GSOAP provides '*wsld2h*' WSDL parser and '*soapcpp2*' stub and skeleton compiler to automate the development of Web services and server applications. The gSOAP WSDL importer and gSOAP compiler are used to parse the WSDL service definitions to create the RPC stub code and XML serialization routines for parameter (de)marhaling by the stub. Optionally, the client stub can be created from the gSOAP header file service definitions. The client application is compiled and linked with the RPC stub and gSOAP communications module to invoke SOAP/xml service functions over the Internet or local network.

To limit memory usage, processing time, and bandwidth requirements gSOAP has implemented additional features that they include improved portability, compact compiler-based code generation methods, and reduced memory overhead with novel streaming techniques for efficient messaging. This tool can be supported on embedded Linux and Windows CE. This aspect is important in the choice of tool whereby we will develop WSED. For example kSOAP is bound to a particular platform. The gSOAP runtime library, 'stdsoap2.c', contains an embedded HTTP Web server and XML parser and the memory space is about 80K.

Two are the optimized phases for the XML serialization. The first phase determines which data components belong to the data structure and which pointers are used to reference them. The second phase emits the XML encoded form of the entire data structure, with all sub-components of the structure serialized recursively. XML parsing is further optimized using look-aside buffers to store XML attribute names with their corresponding values. This permits a minimized dynamic allocation.

GSOAP utilizes a XML pull parsing techniques to achieve the streaming enhancing the performance of XML communications. Therefore to implement efficient binary data transfers, gSOAP has integrated DIME [10] (Direct Internet Message Encapsulation) protocol enabling the transport of attachments with a SOAP/XML request or response. DIME results better performance compared to MIME attachments.

Tests have revealed a good performance about the strategy of gSOAP and it is preferable ahead cSOAP approach regard to execution speed and memory consumption. CSOAP is illustrated in the next section. In particular it performs better than cSOAP for marshalling large messages.

3.2 MMLite

MMLite [11] is not only a tool to develop Web services on small devices. MMLite explores new ways to create compact software systems for the next generation of computing devices. We are going to experiment the final WSED architecture on this system. It is an object-based, modular system architecture, suitable for a wide variety of hardware and components, that provides a menu of components for use at compile-time, link-time, or runtime to construct a wide range of applications. A component in MMLite consists of one or more objects. Multiple objects can reside in a single namespace. When an object needs to send a message to an object in another namespace for the first time, a proxy object is created in the sending object's namespace that transparently handles the marshalling of parameters. Componentization reduces the development time and led to a flexible and understandable system.

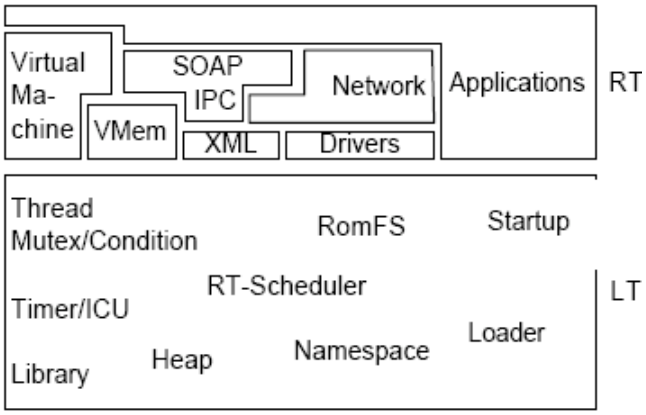


Figure . A sample system configuration. Link time (LT), and run time (RT) Loadable components.

A unique aspect of MMLite is its focus on support for transparently replacing components while these components are in use. MMLite uses COM interfaces, which in turn support dynamic reconfigurability on a per-object and per-component basis. The base menu of the system contains components for heap management, dynamic on-demand loading of new components, machine initialization, namespace, file system, and virtual memory. These components are typically very small (500 to 3,000 bytes on x86), although the network component is much larger (84,832 bytes on x86). The resulting MMLite system can be quite small: the base system is 26 Kbytes on x86, and 20 Kbytes on ARM. A desirable aspect of the MMLite system architecture is that it supports real time constraints irrespective of the type of microprocessor or network connection.

In MMLite, we have are investing the approach to realize Web services on small embedded systems. We depict the components that make it possible for an MMLite device to communicate with Web service.

SAX Parser: [12] Parses XML data as it is being received and calls event driven handler functions through a COM interface.

Tokenizer: It is used by the HTTP server and SAX parser. The tokenizer reads text data from a network stream and splits it into units of text, based on context. The tokenizer is used by the HHTP server and SAX parser. It facilitates processing of network data while it is being received, meaning that the entire request does not have to be present at once. Footprint is therefore reduced.

BAX processor: Deals with pre-tokenized XML. This component is similar to the SAX parser but handles binary XML thus saving significant processing that is otherwise associated with textual data. A standard interface and format has not yet been defined.

HTTP Server: Handles simple HTTP requests such as reading and writing files. URLs map easily to MMLite namespaces. The HTTP server also allows sending SOAP messages embedded into HTML pages, in which case the HTTP server delegates the work to the SOAP marshaler and SAX parser.
SOAP COM Marshaler: Provides automation for accessing COM objects through SOAP. Note that SOAP messages can be handled directly as messages as well as marshalled into COM objects.

3.3 cSOAP

CSOAP [13] is essentially a SOAP engine for resource-constrained devices which is able to deploy Web services and manage RPCs from SOAP clients and dispatch them to services. CSOAP is divided in two parts: cSOAP server part running into the server-side and cSOAP client part running into the client-side. Server-side manages the deployed services, listen to the SOAP messages coming from clients.

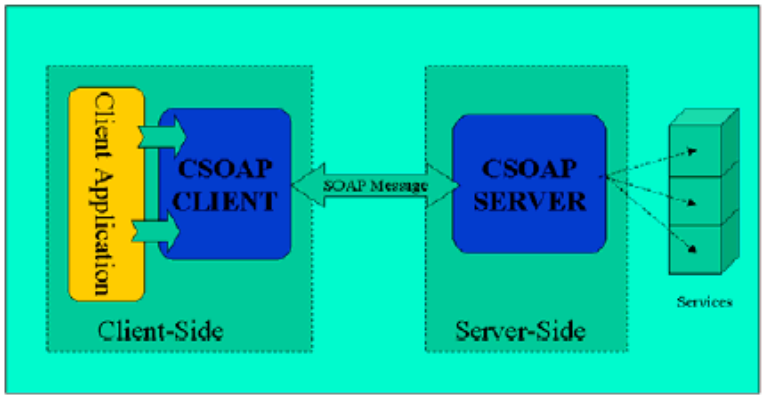
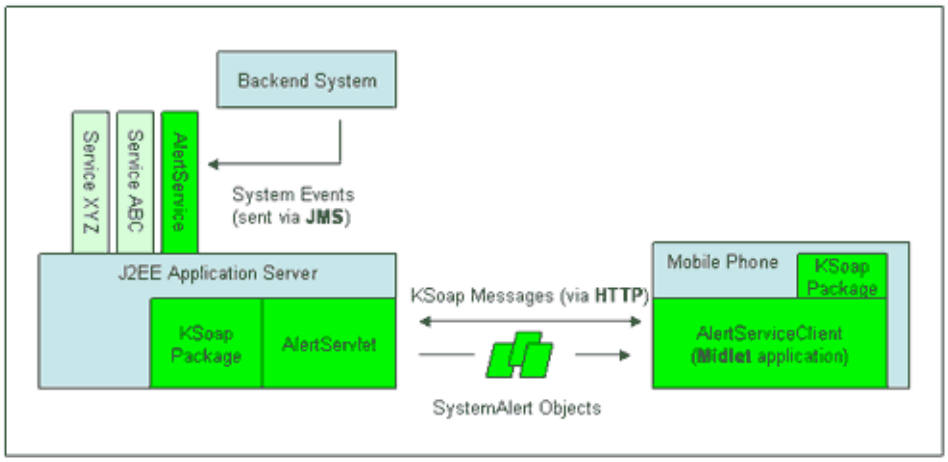


Figure 2. CSOAP Infrastructure

Client-server, CSOAP provides a set of methods which allows a simple development of client application that invokes services deployed on CSOAP server. CSOAP is organized into subsystems which make it modular and extensible. CSOAP takes a different approach. Instead of generating middleware layers, it provides SOAP specific functionality through a library and an associated API. CSOAP does not provide a mapping of types between C and XML Schema. The marshalling and unmarshalling data types such as structs and array must be handled by the application developer. All service specific code resides in the client and server applications themselves.

3.4 kXML and kSOAP

kXML [14] is an open source project that provides a *XML pull parser* to transform XML data to all Java platforms. kXML is suited for small devices because of small footprint size. The popularity of this parser is its adoption on J2ME/MIDP platforms and specially designed for constrained environment. In Sosnoski article, [15], it depicts the performances of different Java XML document and XPP (XML Pull Parser) is the performance leader in most respect. For middleware type applications that do not require validation, entities, processing instructions, and other, XPP looks to be an excellent choice despite its newness. KSOAP [16] (SOAP for the kVM) is an SOAP API with a smaller footprint suitable for J2ME/MIDP, and is based on kXML.



There are three basic tasks involved in working with kSOAP:

- Implement the serialization logic of data objects via Kvmserializable interface
- Register data objects with kSOAP ClassMap
- Integrate services with the KsoapServlet.

The minimum needs of memory space to support kXML-SOAP library is **40Kbytes**.

4.0 Abbreviations

DIME	Direct Internet Message Encapsulation
J2ME	Java 2 Micro Edition
SOAP	Simple Object Access Protocol
WSDL	Web Services Description Language
WSED	Web Services for Embedded Devices
WSML	Web Services Meta Language
UDDI	Universal Description, Discovery and Integration
XML	eXtensible Markup Language
XPP	XML Pull Parser

5.0 Reference

[1] Coulorius G., and others. *Distributed Systems: Concepts and Design*. Addison-Welsey.

[2] Miori V., Tarrini L., Bandinelli R. *Requirements analysis of the WSED architecture from the SOA's perspective.* Deliverable D21.

[3] Chiu, L. Govindaraju, M. Bramley, R. *Investing the limits of SOAP performance for scientific computing*. In proceeding of the 11th IEEE International Symposium on High-Performance Distributed Computing, 2002.

[4] Xerces. http://xml.apache.org/xerces-j/

[5] Axis. ws.apache.org/axis/

[6] *J2ME Web Services Specification*.
http://www.jcp.org/aboutJava/communityprocess/review/jsr172/

[7] gSOAP Project. http://www.cs.fsu.edu/~engelen/soap.html

[8] Engelen, R. *Code Generation Techniques for Developing Light-Weight XML Web Services for Embedded Devices.* Proceeding of the ACM SIGAPP SAC Conference, 2004.

[9] IBM alphaWorks. *Web services tool kit for mobile devices*. www.alphaworks.ibm.com/tech/wstkmd

[10] *DIME*. http://msdn.microsoft.com/library/en-us/dnglobspec/html/draft-nielsen-dime-02.txt

[11] *Invisibile Computing*. http://research.microsoft.com/invisible/

[12] *SAX Parser*. www.saxproject.org/

[13] *CSOAP Project*. http://csoap.sourceforge.net/

[14] *KXML Project*. http://kxml.sourceforge.net/

[15] Sosnoski, D. *A look at features and performance of XML document models in Java*.

[16] KSOAP Project. http://kobjects.org/auto?self=%2381d91ea1000000f5b22b3fc4