State of the Art of SOAP Libraries in Python and Ruby

Pekka Kanerva

Helsinki Institute for Information Technology

August 6, 2007

HIIT
Technical Report
2007-02

State of the Art of SOAP Libraries in Python and Ruby

Pekka Kanerva

Helsinki Institute for Information Technology

HIIT Technical Reports 2007-2

ISSN 1458-9478

State of the Art of SOAP Libraries in Python and Ruby

Pekka Kanerva
<Pekka.Kanerva@HIIT.FI>
Helsinki Institute for Information Technology

August 6, 2007

Abstract

Web services are gaining more and more attention in the business field and in the academic research. Simple Object Access Protocol (SOAP) is the standard messaging format for Web services. The single services are described in Web Services Description Language (WSDL). More recently, the REST architecture specified by Roy T. Fielding has received more attention in creating better Web services. This technical report describes our experiments on building simple, composable Web services. We describe our findings on using both Python and Ruby SOAP libraries for prototyping. A simple REST interface is created for a commercial Web service called SyncShield.

Chapter 1

Introduction

The ITEA Services for all (S4All) research project aimed to create a world of easy-to-use, easy-to-share, and easy-to-develop services from a user point of view.

S4All describes a visionary software component called a Service Composer which is used to combine public small-scale web-services into a more complex series of meaningful series of simple tasks enqueued into a workflow.

HIIT's contribution to the project was to experiment the Service Composer on mobile end-user terminals with rapid development of both services and service composer with tools such as Python [30] and Python for S60 [29] and Ruby [33].

This Technical Report is organized as follows... In Section 1.1 we describe Web services and the technologies involved briefly. In Chapter 2 we describe the programming languages and libraries being used. In Section 3.1 we describe the architecture of our prototype. In Chapter 4 we draw up the conclusions of our experiment. Last, we give directions for further work. Abbreviations and notations which are being used on this report are listed on Chapter 5, Abbreviations and Notations.

1.1 Web Services

The W3C has defined a *Web service* to be a software system designed to support interoperable M2M interaction over a network [16]. A Web service[1] has a machine processable interface. Typically this interface is described in WSDL. Networked nodes communicate with a Web service by exchanging SOAP messages, usually over HTTP. The conceptual figure of a *Web service* is depicted in Figure 1.1.

[1] We are following the W3C's convention of writing the *Web service* with a capital letter

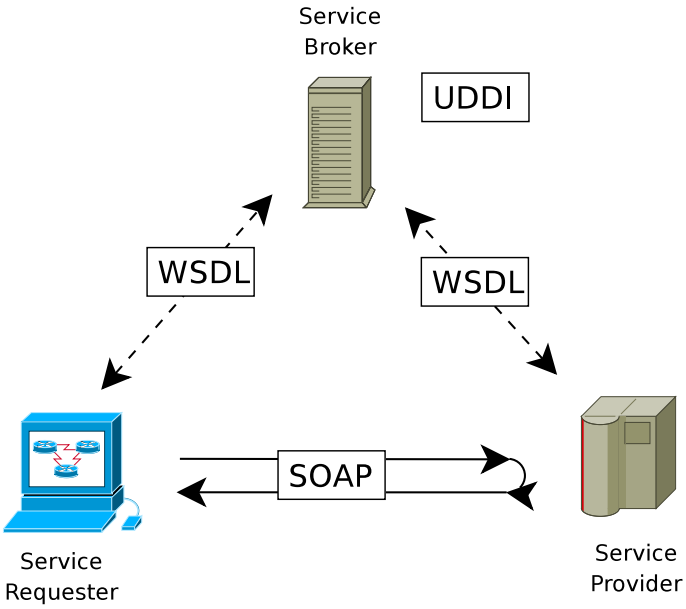

Figure 1.1: Web services

The service provider creates an API for the services he/she publishes in the network. The services are described in Web Service Description Language (WSDL) which is an XML formatted document describing the service method calls, the structures, and so on.

The Service Provider may register his/her service API into a global repository of businesses and services into the Service Broker. The global registry is implemented as Universal Description, Discovery and Interaction (UDDI) [38], an XML-based registry. UDDI stores the WSDL descriptions of Service Providers' service APIs. UDDI service consists of three parts. *White pages* describe the identifiers, addresses and contacts for the services. *Yellow pages* describe the services categorized into common standard taxonomies. Finally, UDDI *Green pages* provide more technical information about the services.

An end user searching for some service first commits a search for the UDDI service for a service he/she is searching for. If the search is successful, the UDDI service returns a WSDL description of the service to the Requester. The Requester processes the WSDL which describes the access point for the desired service. Now the Requester may contact to the Service Provider's service API and start to use the service.

1.2 Representational State Transfer (REST)

Representational State Transfer is a software architecture which was originally described in Roy T. Fielding's doctoral dissertation [13]. The REST architecture is commonly used in WWW-like systems.

REST architecture consists of a group of principles which describe an interface to the WWW services being used. REST requires no additional messaging protocol like SOAP.

> The name "Representational State Transfer" is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through the application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use. [13]

One of the key concepts of REST is the existence of *resources* which then can be referenced by unique ids, i.e. URIs [6]. These resources can be then retrieved, stored, modified, etc. by using some agreed messaging protocol, like HTTP. As a result, the service returns *a representation* of the resource, i.e. an actual text document, a PNG image, an OGG music file, etc.

In REST architecture, the collection of resources to be referenced can be considered as *nouns* in difference with the operations on the resources which act as *verbs*. In both cases, it's usually so that the noun or verb needs some additional parameters, or *adjectives* to describe the request more carefully. As a simple example, consider the following HTTP request

```
GET /rest/apple?cultivar=goldendelicious
```

The typical set of verbs consists of HTTP operations, i.e. `GET`, `PUT`, `POST`, and `DELETE`, when HTTP is used as a transport protocol. The example above refers to an apple object, i.e. a noun, with a defining adjective `goldendelicious`.

Another vital characteristic of REST architecture is *layering*. This means that any number of proxies, caches, clients, servers, etc [2] can participate in mediating the request transparently through a network without any knowledge of the endpoints.

Additionally, REST suggests services which are *client-server*, *stateless*, and *uniformness of interfaces*. Client server model refers to the most common architectural style of services offered in networks. Statelessness refers to

[2] These are usually referred as a general notion *connectors*

the requirement that the server has no internal states to keep track. This simplifies the implementation and improves server scalability and efficiency significantly through the smaller memory consumption and more simple server internal structure.

1.3 REST Calls for SOAP API

Typically, a SOAP API is a set of methods inside a single service which is described in a WSDL file. If the provided Web service is more complex and thus larger, it is reasonable to set up a number of different services described in different WSDL descriptions. Each of these services provides an disjoint set of methods of their own. An analogue to the object oriented programming is a number of classes each of which implement a set of methods to modify the object instances.

REST hierarchy of identifiers, i.e. URLs which name the set of resources is somewhat different from the SOAP approach. URLs define a hierarchical structure for the resources which takes a shape of a tree. Although it is possible that the set of resources is a flat, one level set of IDs, like `<URL: http://www.someservice.org/{a, b, c, ..., n} >`, tree structure is more typical. The difference is depicted in Figure 1.2.

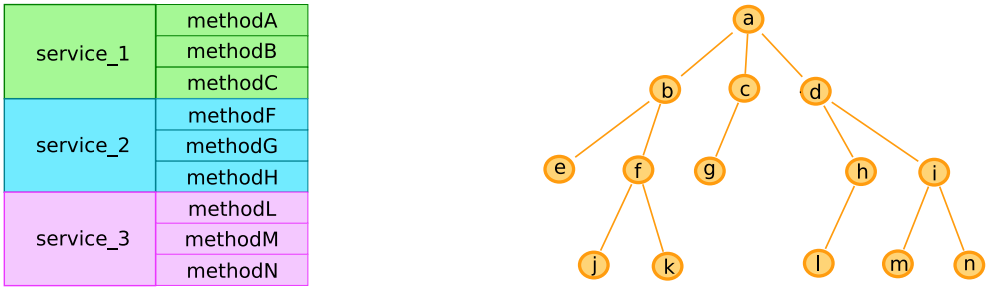

Figure 1.2: SOAP-REST mapping

In SOAP case, the Web service has two distinct services, `service_1`, ... `service_3`. These provide the following methods $methodA, ..., methodC$, $methodF, ..., methodG$, and $methodL, ..., methodN$ respectively. Still, the SOAP API is more or less flat, a group of methods accessing the resources.

In RESTful architecture, the resources form a categorized hierarchy for the Web service. In case of the situation in Figure 1.2, for example, the resources `j` and `k` belong to a upper-level group of resources denoted with `f`. This, in turn, belongs into a broader category `b` together with `e`. Assuming

the Web service is provided by A Company Ltd. Now the resource `k`'s unique identifier could be `<URL: http://www.acompany.com/service/a/b/f/k >`.

For creating a RESTful interface to a SOAP service, careful design must be done when drawing the hierarchy up.

1.4 Use Case Scenarios

We were considering a number of use cases to get a grip on every-day life situations in which mobile service composition would be useful. In the following we will briefly describe some scenarios in which a composable Web services could easen up daily tasks.

1.4.1 Fire Fighting

As a first scenario we considered a group of fire fighters in a mission to extinguish a forest fire. We will now suppose that a fire fighter has a handy new application composed of various Web services on his PDA.

When the fire alarm comes our fire fighter have the location of the forest fire on his location application on the PDA. He/She will have the exact coordinates of the fire which he/she sends to a national weather service which gives the current short term weather forecast for the given location. The essential pieces of information are the precipitation and winds. The fire department has implemented a web service which estimates how fast and to which direction the fire spreads. This gives the fire fighters valuable information how to coordinate the extinction task.

1.4.2 Travel reservation

A second scenario goes as follows. Consider a sales manager of a large software company who is attending to a trade fare abroad. First he/she contacts the flight reservation to book a flight to the destination. Having the flight booked the manager contacts to a local taxi company and reserves a taxi to the airport after a reasonable time slot for collecting the luggage after the flight has landed.

The organizer has listed several hotels near the fare venue. The sales manager browses their home pages and selects one which pleases him most. The reservation time is taken from the fair organizers web service. The manager reserves the hotel he/she selected with the handy reservation Web service.

1.5 Considerations

The above mentioned scenarios describe briefly a couple of different cases of daily work.

In the fire fighting scenario we can identify the following independent, single services. 1) the locationing service to signal the exact region of the forest fire, 2) the weather service, 3) the service for approximating the pace of the propagation of the fire.

In the travel reservation use case we recognize 1) the on-line flight booking service, 2) taxi reservation service, 3) hotel booking system.

These use cases depict clearly that Web services easen every day living. Some of the services mentioned in the scenarios are here already. Flights can be booked online already as well as taxis can be ordered with a web form.

However, www pages do not provide ability to compose several distinct services into a larger series of tasks nor make possible mobile service composition. Quite often it is also the case that large web pages get loaded so slowly on a mobile terminal that nobody really wants to use such services with a smartphone. The services must have some common well defined API which is fast enough to use even with low band-width devices. REST and SOAP provide such well defined APIs.

Chapter 2

Technologies

We wanted to prototype a piece of software which can be used to compose simple Web services into a series of tasks which are executed successively. Each single task reads its input from the output of the predecessor. Apple has a distantly similar application included in the Mac OS X called Automator [4].

2.1 Mobile Terminals

We have several year's of experience in working with Nokia S60 mobile terminals as programming platforms in HIIT in various research projects. Several research prototypes are implemented both in native Symbian platform in C++ and on J2ME Java platform as well as in Python with Python for S60. Therefore it was a natural choice of continuing to experiment on S60 platform with Nokia phones.

Computation and storage capacity of mobile phones and PDA devices has grown up rapidly during last few years. As the performance of mobile devices is rapidly reaching for the capabilities of desktop PCs, these new high-performance hand-held devices are more often called as smartphones. Thus the traditional PDAs and mobile phones have merged into a single device.

Software development in such a device is usually a more laborious task due to the limited capabilities of the small device. This is due to the limited memory resources which requires more diligent memory management on the programming phase. The requirement of the smartphone be running for months or years requires conserving memory management and rare memory leaks. Secondly, the exception handling in various error conditions must be done in a more accurate manner to avoid the piece of software to hang.

2.2 Python

Python is a *powerful*, *dynamic*, *platform-independent*, and *object-oriented* programming language [30]. It has an over a decade long history on various application domains. We had an extensive experience on working with Python in previous research projects with positive experience.

Python's strength lies in the versatile module library which comes in the standard Python distribution. No extra libraries are needed - at least normally - to search for to do the job. Python offers powerful introspection capabilities which makes it easy to inspect and modify code at runtime.

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print n, 'equals', x, '*', n/x
...             break
...     else:
...         # loop fell through without finding a factor
...         print n, 'is a prime number'
...
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

Listing 2.1: Sample Python code [31]

Python integrates easily with several object models in wide use like CORBA and .NET. Python runs under Linux/UNIX, Windows, Mac, OS/2, Amiga, under .NET, Java virtual machine, and S60 platform among others. For more info about Python, see [30].

Python's availability to multiple platforms, especially on S60 made it an easy decision as an implementation language.

2.2.1 Python for S60

Above mentioned specialities of Symbian environment in Section 2.1 makes the Symbian C++ flavour of C++ programming very special task. Therefore it takes a considerable time to learn oneself as a productive Symbian

programmer. To lower the learning curve of programming the smartphones, development of easy-to-use higher-level programming languages as Java or Python on Symbian platform was done.

We chose Python for S60 over J2ME for the easy of use of python scripts over Java programming environment and for the Python's wider support for mobile devices native features such as built-in camera, telephoning api, SMS sending, and contacts database.

We do not go into deep in details of the more laborious setup of the Java programming environment in comparison with Python. To give the reader a glimpse of the difference of the setup of the two language environments we mention that to start programming on the J2ME platform one needs to get some implementation of the Mobile Information Device Profile (MIDP) and Connected Limited Device Configuration (CLDC) class libraries, like Sun Wireless Toolkit [37] or Nokia's Carbide.j [1] [26]. To have a more non-restricted access to the platform from Java, MIDlets [2] needs to be signed [20] by a known trusted third party (TTP) to access the file system, for example. To install the Java MIDlet into the mobile device, the compiled class files must be packed up into a Java application archive (JAR) and Java application description (JAD) files which, in turn, can be downloaded to the mobile device from a web page.

The reader may consult Sun's J2ME homepages [19] and Java Community Process's homepages [18] for more information.

The Python world is much more simple without requiring multi-step compilation and installation process. The only required steps to start Python application development for smartphone is to get the Python for S60 environment and install it in the smartphone and a text editor for the desktop PC to write the program code. The Python code can be transferred into the smartphone by a bluetooth link, for example.

2.3 Ruby

Despite of its history of over a decade, as far as from 1995, Ruby programming language [33] has gained mass acceptance during only a few recent years. Ruby's creator, Yukihiro "matz" Matsumoto, mixed up parts of his favourite languages Perl [27], Smalltalk [35], Ada [1], and Lisp [21] to create a new language.

Ruby's highly *object-oriented* character, *open* and flexible structure and *blocks* are considered as its most important strengths. In Ruby, everything

[1] Formerly known as Nokia Developer's Suite for J2ME

[2] Java applications dedicated to be executed in embedded devices are called *MIDlets*.

is an object. Compared to many other programming languages in which the primitive types at the very lowest level like `int`s and `char`s are not object, this is not the case in Ruby. Everything is an object as is the case in one of the model languages, Smalltalk.

Every class in Ruby's libraries are open. This means that users can modify, augment or delete any parts at will. For example, traditional addition is done with plus sign. Let us say that you'd like to do addition with written method name `plus` instead of the common $a + b$ notation. It's easy:

```
class Numeric
  def plus(x)
    self.+(x)
  end
end

y = 5.plus 6
# y is now equal to 11
```

Listing 2.2: Sample Ruby code [32]

Ruby's blocks implement a closure functionality with which a programmer can pass a closure to any method. Blocks are considered one of the Ruby's most compelling characteristics.

Chapter 3

Experiments

In the first phases, we experimented with the publicly available web services provided in XMethods [1]. A great deal of these services which are implemented for testing SOAP implementations are sensible only on certain geographical areas. There were several services providing city address information, a number of delayed quote info on several stock exchanges, and several different phone number services, to mention some of the services with locational constraints. These services are just fine for testing the SOAP application or library but provides little use for further application prototyping, especially globally.

However, there are also several nice services for experimental usage without any locational limits. To mention some of them, a currency conversion service and several encryption services [2] are the most useful of them.

3.1 Architecture

The architecture of our prototype setup is depicted in Figure 3.1.

The Service Composer running in mobile terminal is described in Ville Mäntysaari's master's thesis [22]. We will not go into details here.

The Web services we were exploring in our prototype were provided by an early version of SyncShield [9] service implemented by Capricode [8]. The SyncShield service is targeted to mobile device management to support wireless business processes. SyncShield offered a group of concise, similar services for mobile device management which made it possible to create simple series of tasks from a set of primitive operations.

[1] <URL: http://www.xmethods.org >

[2] A Caesar cipher or a Vigenere cipher cannot be considered as a true encryption solution, but they are nice for prototyping though.

Due to the company confidential information we do not go into details of SyncShield. For more information about the details of SyncShield, the reader is advised to contact Capricode.

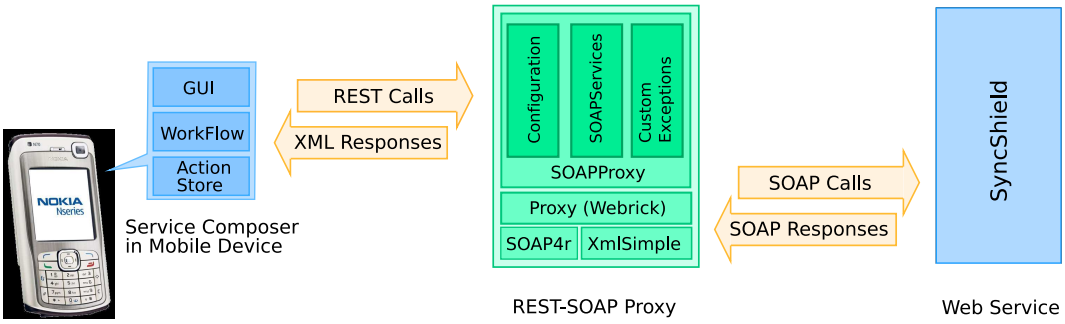

Figure 3.1: REST-SOAP Proxy Architecture

The strengths of the REST architecture made it an intriguing way to approach the prototype implementation. However, according to Capricode, they have planned to implement a REST API for SyncShield but at the time we implemented our prototype, we couldn't wait for the REST API. We implemented a simple proxy which implemented a REST interface for SyncShield. The proxy transforms the REST calls into SOAP messages which are sent to SyncShield server.

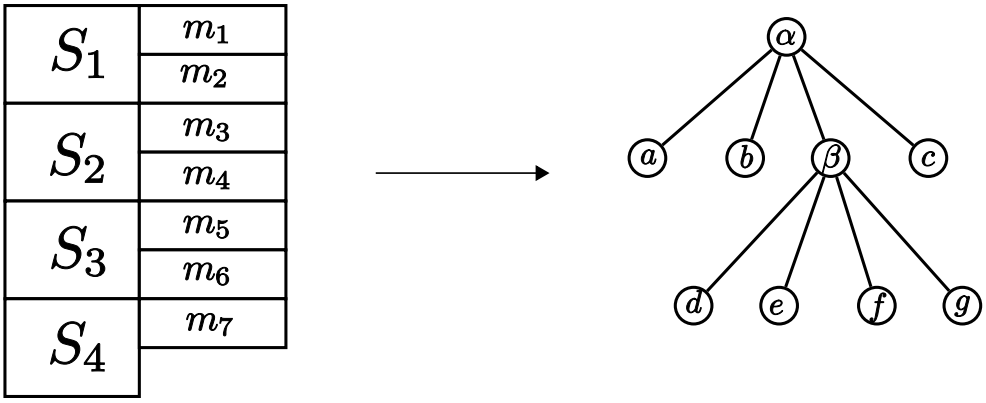

Figure 3.2: Implemented SOAP-REST Mapping

The SyncShield version we were using provided four different SOAP services. The system is depicted in Figure 3.2. The services are numbered as $S_1, ..., S_4$ There are seven methods implemented in these services, numbered

as $m_1, ..., m_7$. The following REST contains two groups of resources, i.e. the top-level group α which contains four leaf resources. One of these leaves is a lower-level group, denoted β. The independent resources are labeled as $a, ..., j$.

To keep our proxy as lightweight as possible, we needed to make a slight violation on the REST principles thus avoiding to create a stateful proxy. To do a complete SOAP-REST transformation, SOAP methods should be translated into operations which modify the resources, i.e. nouns. In the leaf nodes we have the REST equivalents of SOAP methods. We didn't consider this such a big matter since our primary target was to investigate mobile service composition, not the subtleties of REST.

The disadvantage is that we didn't get the full understanding on the complexity of the SOAP-REST transformation. However, there's not so much work to be done in transforming the Web services from method orientation to the resource orientation. The tree structure remains analogous to the one we implemented. The remaining step would be replace the leaf nodes with the resource identifiers and the operation, i.e. the verb moved into an URL parameter. The number of parameters is the only part which might grow when the leaf nodes are replaced by the resources.

3.2 Implementation

Our first selection as an implementation language in the proxy side was naturally Python since we had already done the Composer with Python for S60.

3.2.1 Choosing the Python SOAP Library

When selecting the SOAP library, during mid-2006, there were two major projects implementing SOAP for Python, namely ZSI (Zolera SOAP Infrastructure) [34] and SOAPpy [17]. SOAPpy was nice and small library and easy to use with some simple examples. ZSI was considerably larger library with more functionality.

Both the SOAPpy and ZSI were on heavy development phase during the time, ZSI probably the most. The release of ZSI had been coming "real soon now" for some time but was delayed. It seemed though that no prominent development work was done during the last year.

For creating simple-to-use, as much automated service discovery and utilization, we needed a SOAP library which can process WSDL descriptions of Web services. ZSI library provides a command line tool `wsdl2py.py` which is designed to generate the Python code to access the Web service exposed by

```
url = 'http://www.xmethods.org/sd/2001/TemperatureService.wsdl'
zip = '01072'
proxy = SOAPpy.WSDL.Proxy(url)
temp = proxy.getTemp(zip)
print 'Temperature at', zip, 'is', temp
```

Listing 3.1: WSDL handling sample from SOAPpy package

the WSDL description. The design choice is quite interesting in that sense, that the idea behind WSDL descriptions is that it should be possible to handle the descriptions automatically instead of a manual command line script. Further, ZSI contains another script `wsdl2dispatch` which should be run after the `wsdl2py` has executed. The `wsdl2dispatch` will generate a module containing an interface to the Web service.

SOAPpy provides a simple interface for dynamic WSDL processing. A sample code taken from an example in the SOAPpy package is described in Listing 3.1.

SOAPpy was still approaching the 1.0 release which can be understood to mean that it was still growing and maturing towards the first official release. The latest release was 0.11.6 from Sept, 2004 with a 0.12_rc1 dated on Feb 2005. Since only the SOAPpy library included the support for dynamic WSDL handling, SOAPpy was the only choice we had.

Problems with SOAPpy's capability to process WSDL descriptions occurred when we tried to process SyncShield's service descriptions. Our tests with simple WSDL files were successful, but more complex descriptions were problematic. If a `complextype` contained an array of elements the WSDL parser threw a fault. Numerous problems with WSDL handling were questioned on the pywebsvcs-talk mailing list on Python Web Services web pages [28] without any noteworthy solutions nor quick patches so far.

We had no resources to start fixing the SOAPpy library. Therefore we also took a look on Ruby and its SOAP libraries. Before going into Ruby, we list some of our expressions on installing SOAPpy into Python for S60.

3.3 Installing SOAP libraries into Python for S60

In the testing phase we did some experiments on dropping the SOAPpy libraries into the Python for S60. We wanted to have only the SOAP support so we didn't even consider trying to cut the SOAP specific libraries out of

the ZSI even if the libraries would have been more finished at that time. SOAPpy was considerably smaller compared with the ZSI, almost five times.

We didn't manage to isolate SOAPpy from dependencies to external Python libraries. SOAPpy was quite much dependent on libraries not included on Python for S60 distribution[3]. After we had added `cgi.py`, `fpconst.py`, `smtplib.py`, `threading.py`, `UserDict.py`, `UserList.py`, and `weakref.py` into the Python for S60 library directory in the mobile phone, SOAPpy didn't run properly on Python for S60. The weakref library was the most problematic part. It made references to the Python's core parts which were implemented as native Symbian libraries. Tero Hasu kindly provided us a work around implementation of the `weakref` library which which used strong references instead of weak references. Unfortunately, the strong reference patch gave no help.

3.3.1 SOAP Libraries for Ruby

We had an existing small web server implemented in Ruby and Webrick [39] so we started to expand it for implementation of the Proxy.

The experiments were done in standard Ubuntu Linux 6.06 Server Edition. The standard Ruby distribution, version 1.8.5 on Ubuntu 6.06 LTS, ships with SOAP libraries. However, we didn't get the SOAP library work well enough. We then started to experiment with SOAP4r.

The latest release of SOAP4r was 1.5.5 at the time. We first installed it as a Ruby gem which didn't work either. There was no success with compiling the version 1.5.5 from the sources either. There were unrecoverable errors on processing the WSDL descriptions. Arrays containing `simpletype`s didn't map correctly into Ruby objects. The same problem was reported in [36] also. Wilson reports similar kind of problems on his blog [40]. His experiment was somewhat different from ours, though. We tried the latest nightly build which was on that time version 20061022 which fixed the problems. The current version today, both in the SOAP4r web page and in Ruby gem repository are the version 1.5.5.20061022.

We used XmlSimple [41] for generating the required additional XML messages from SOAP calls. This was the case mainly done on generating simple exceptions from SOAP exceptions originated from the Web Service. Ruby's native SOAP implementation generated well-formed XML for the smartphone with `SOAP::Marshal` module.

[3]See `API_Reference_for_Python.pdf` for modules included in Python for S60 available in Python for S60 download package [29] in Forum Nokia website: `<URL: http://www.forum.nokia.com >`

XmlSimple origins goes back to a Perl module `XML::Simple` [23] originally written by Grant McLean. Using XmlSimple is one of the easiest-to-use XML libraries we have encountered. Ruby data structures, lists and dictionaries, are directly mapped into XML structures and vice versa. No heavy-weight DTD or DOM handling is required. For a simple XML generation XmlSimple library is an excellent library tool for easy usage and fast adoption.

Chapter 4

Experiences and Conclusion

SOAP has gained a wide acceptance in business world as a language implementing application protocol messaging. Various implementations exists for many different programming languages. More work should be done on enhancing the interoperability of different SOAP libraries. There are problems within single programming language - two different implementations of SOAP will not necessarily work without some clashes. This is the case also between different programming languages. On our conversations with Capricode we found out, that they have encountered the same problems with Java SOAP libraries. Some implementations work better on some aspects of SOAP features, others exhibit their strengths in another features.

The lack of documentation is a real problem in SOAP4r. The reason for this is mainly the original author of SOAP4r, Hiroshi Nakamura's feelings of his insufficient knowledge on English [25]. There is very little documentation available, if any. A number of examples is provided. Some third party docs are listed in SOAP4r's Trac pages. These are, of course, with the sample code, useful but provides not very much help in general. SOAPpy is documented a lot more carefully which is in line with Python's standard of good documentation availability.

The REST-based approach in architectural design of Web services is a serious attempt to re-claim the www back as it was around 1993 when the first popular web browser Mosaic [24] was released. Web was just URLs and www pages, i.e. universal locators and resources. RESTful approach of Web service architecture is promising at least in the client side due to the simplicity of creation of calls into resources.

Our experience on creating a simple REST API for a SOAP service was positive. It required only a small amount of code to parse the RESTful request URLs. Careful design must be done to create a proper URL directory structure for the resources.

However, there may be need for processing XML messages resulting from queries, which means that SOAP and XML libraries may be needed in any case when receiving results. This may increase the complexity of the client applications in the mobile devices where getting the required application libraries is more laborious than in a desktop PC.

Various actors have established the services and goods the actors are offering also into the web. Usually the services are provided as web pages, net stores, www reservation services, and so on. Far more few services are implemented as Web services in a sense as W3C describes it. In general, Google, Amazon, eBay, and Flickr can be considered the only actors who have published their services also as Web service APIs [2,10,14,15]. Further, free blogging services like Bloglines [1], personal organizer Backpack [2], and event search site Eventful [3] offer their Web service API [5, 7]. REST APIs are provided by Amazon, Eventful, Flickr, and eBay [3, 11, 12, 14].

The services mentioned above are a nice opening for general-purpose Web services for the public. However, more services are needed to create a truly open, multi-purpose field of diverse, composable web services.

4.1 Further Work

Our implementation of the REST proxy for a SOAP service consisted a relatively small portion of a larger service. It would be interesting to repeat the study with a larger service with more features and services offered. The question about the need for a stateful proxy is another interesting issue. With the time constraint in hand, we didn't want to jump into that train in the experiment. Still it remains unanswered whether the statefullness would be a must in our case if the full REST compliance was to be implemented.

Another interesting topic is the composer GUI. We experimented with textual GUI widgets, but the German study conducted under the S4ALL project experimented with graphical UI in desktop PC. Doing some experiments with a graphical UI in the mobile phone would be fascinating experiment. Graphics is somewhat heavier solution instead of text-based elements so it would be interesting to know could it be done with a mobile device and RAD tools. Another limiting factor is the small size of the mobile device, and the possible need for touch screen.

[1] <URL: http://www.bloglines.com>
[2] <URL: http://www.backpack.com>
[3] <URL: http://www.eventful.com>

Chapter 5

Abbreviations and Notations

BEEP	Blocks Extensible Exchange Protocol
HTTP	Hypertext Transfer Protocol
J2ME	Java 2 Platform, Micro Edition
MIDP	Mobile Information Device Profile
PDA	Personal Digital Assistant
REST	Representational State Transfer
RESTful	Web design which complies with the REST principles
M2M	Machine to Machine
RPC	Remote Procedure Call
SOA	Service-oriented architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
WSDL	Web Services Description Language

Bibliography

[1] *Ada Home: The Website for Ada.* <URL: http://www.adahome.com/ >. (referred 23.5.2007).

[2] *Amazon Web Services.* <URL: http://www.amazon.com/AWS-home-page-Money/b/ref=gw_br_websvcs/103-7241410-8536661?\%5Fencoding=UTF8&node=3435361&pf_rd_m=ATVPDKIKX0DER&pf_rd_s=left-nav-2&pf_rd_r=13NQ5V1J4K0SBBKWV4MD&pf_rd_t=101&pf_rd_p=285790601&pf_rd_i=507846 >. (referred 15.5.2007).

[3] *Amazon S3 - Using the REST API.* <URL: http://docs.amazonwebservices.com/AmazonS3/2006-03-01/ >. (referred 16.5.2007).

[4] *Automator - Your personal automation assistant.* <URL: http://www.apple.com/macosx/features/automator/ >. (referred 25.4.2007).

[5] *Backpack API.* <URL: http://www.backpackit.com/api/ >. (referred 16.5.2007).

[6] T Berners-Lee, R Fielding, and L Masinter. *Uniform Resource Identifiers (URI): Generic Syntax.* Request For Comments, <URL: http://ftp.funet.fi/rfc/rfc2396.txt >, August 1998. (referred 26.4.2007).

[7] *Bloglines Services.* <URL: http://www.bloglines.com/services/ >, 2007. (referred 16.5.2007).

[8] *Capricode Oy.* <URL: http://www.capricode.com >. (referred 4.5.2007).

[9] Capricode. *SyncShield.* <URL: http://www.capricode.com/index.php?197 >, 2007. (referred 4.5.2007).

[10] *Developers Program.* <URL: http://developer.ebay.com/index_html >. (referred 15.5.2007).

[11] *eBay REST API.* <URL: http://developer.ebay.com/developercenter/rest/ >, Dec 2006. (referred 16.5.2007).

[12] *Eventful API.* <URL: http://api.eventful.com/ >. (referred 21.5.2007).

[13] Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, 2000. Available online: <URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm > (referred 25.4.2007).

[14] *Flickr Services.* <URL: http://www.flickr.com/services/api/ >, 2007. (referred 16.5.2007).

[15] *Code Google.* <URL: http://code.google.com/ >. (referred 15.5.2007).

[16] Web Services Architecture Working Group. *Web Services Architecture.* <URL: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/ >, Feb 2004. (referred 4.5.2007).

[17] SOAPpy homepage. <URL: http://sourceforge.net/project/showfiles.php?group_id=26590&package_id=18246 >. (referred 24.4.2007).

[18] *Java Community Process.* <URL: http://www.jcp.org >. (referred 25.4.2007).

[19] *The Java ME Platform.* <URL: http://java.sun.com/javame/index.jsp >. (referred 26.4.2007).

[20] Jonathan Knudsen. *Understanding MIDP 2.0's Security Architecture.* <URL: http://developers.sun.com/techtopics/mobility/midp/articles/permissions/ >, Feb 2003. (referred 2.5.2007).

[21] *Association of Lisp Users.* <URL: http://lisp.org/alu/home >. (referred 23.5.2007).

[22] Ville Mäntysaari. *Service Composition on a Mobile Phone.* Master's thesis, University of Helsinki, 2007. (working title).

[23] Grant McLean. *XML-Simple.* <URL: http://search.cpan.org/dist/XML-Simple/ >, Oct 2006. (referred 5.6.2007).

[24] *NCSA Mosaic – September 10, 1993 Demo*. <URL: http://www.totic.org/nscp/demodoc/demo.html >. (referred 15.5.2007).

[25] Hiroshi Nakamura. *Re: wsdl2ruby.rb for Amazon EWS - advanced problems*. <URL: http://groups.google.com/group/soap4r/msg/6e9beae5a06c05f5?dmode=source&output=gplain >, Jul 2006. a message on soap4r@googlegroups.com mailing list, (referred 29.5.2007).

[26] *Carbide Development Tools*. <URL: www.forum.nokia.com/carbide >. (referred 2.5.2007).

[27] *The Perl Directory at Perl.org*. <URL: http://www.perl.org/ >. (referred 23.5.2007).

[28] *Python Web Services*. <URL: http://pywebsvcs.sourceforge.net/ >. (referred 24.4.2007).

[29] *Python for S60*. <URL: http://opensource.nokia.com/projects/pythonfors60/ >. (referred 25.4.2007).

[30] *Python Programming Language*. <URL: http://www.python.org/ >. (referred 25.4.2007).

[31] *Python Tutorial*. <URL: http://docs.python.org/tut/node6.html >, Sept 2006. (referred 29.5.2007).

[32] *About Ruby*. <URL: http://www.ruby-lang.org/en/about/ >. (referred 27.4.2007).

[33] *Ruby Programming Language*. <URL: http://www.ruby-lang.org >. (referred 25.4.2007).

[34] Rick Salz and Christopher Blunck. *ZSI: The Zolera SOAP Infrastructure*. <URL: http://downloads.sourceforge.net/pywebsvcs/zsi-2.0.pdf?modtime=1170404063&big_mirror=0 >, Feb 2007. (referred 24.4.2007.

[35] *Smalltalk.org*. <URL: http://www.smalltalk.org >. (referred 23.5.2007).

[36] *SOAP4r - Changes in version 1.5.5 in SOAP4 Trac*. <URL: http://dev.ctor.org/soap4r/wiki/Changes-155 >. (referred 26.4.2007).

[37] *Sun Java Wireless Toolkit for CLDC*. <URL: http://java.sun.com/products/sjwtoolkit/ >, Apr 2007. (referred 2.5.2007).

[38] *OASIS UDDI Spesifications TC - Committee Spesifications.* <URL: http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm >. (referred 29.5.2007).

[39] *Webrick - A Ruby library for build HTTP servers.* <URL: http://www.webrick.org/ >. (referred 30.5.2007).

[40] Brendon Wilson. *Ruby + SOAP4R + WSDL Hell.* <URL: http://www.brendonwilson.com/blog/2006/04/02/ruby-soap4r-wsdl-hell >, April 2006. (referred 11.5.2007).

[41] *XmlSimple - XML Made Easy.* <URL: http://xml-simple.rubyforge.org/ >. (referred 14.5.2007).