Web-Based Deployment of Text Locating Algorithms

Simon M. Lucas and Carlos R. Jaimez González
Computer Science Department
University of Essex, UK
{sml,crjaim}@essex.ac.uk,

Abstract

This paper describes a simple yet novel approach to the web-based deployment and evaluation of text locating algorithms.

Web-based deployment allows algorithms to be evaluated by end users or researchers, without the need to install the algorithm. This is a major advantage both for the end user, and for the algorithm developer. The end user is protected from lengthy installation procedures, which may also leave one's machine in a corrupted state. The algorithm developer is protected from theft of software or intellectual property.

Our system provides access to a deployed algorithm in two ways: interactive mode via a web browser, and program access mode via a special kind of web service architecture. The system is demonstrated with the deployment and testing of one of the entries for the ICDAR 2005 text locating competition.

1 Introduction

The Web has already dramatically improved the efficiency of the research process, offering searchable access to a vast number of papers, on-line articles, and discussion forums. For those engaged in empirical computer science research, however, the best may be yet to come. Currently, the *modus operandi* in fields such as computer vision is for researchers to evaluate their own algorithms on public datasets, and then publish the results in a paper, which is subject to a delay of at best several months, but at worst two years or more, before publication. Competitions have been associated with several research communities and conferences, and these help in establishing the state of the art in a particular field, but do to the effort of running them, and of participating in them, are usually run only annually or biennially. We argue that the software technology is now ready to enable researchers to deploy their algorithms as special kinds of web services as a matter of standard practice. Evaluation and comparison on a potentially vast number of datasets can then be an automatic, and on-going process.

Web-based deployment allows algorithms to be evaluated by end users or researchers, without the need to install the algorithm. This is a major advantage both for the end user, and for the algorithm developer. The end user is protected from lengthy installation procedures, which may also leave one's machine in a corrupted state. The algorithm developer is protected from theft of software or intellectual property.

In recent years there has been a great deal of interest in web services, and *Service Oriented Programming* [1] has been proposed as a new programming paradigm. However, so far, the reality has not lived up to the hype, and in most web services perform only simple functions, such as retrieving the current price of a specified stock, for example.

On the other hand, pattern recognition researchers have long recognized the benefits of offering web-based demonstrations of their software. A very relevant example of this is the Carnegie Mellon University Robotics Institute (CMU-RI) Face Detection demonstration[1]. The demonstration system works as follows. Users upload images to the system using a simple two-field form, specifying their email address in one field, and the URI of the image in the other field. This latter detail means that users must be able to upload an image to a web server before interacting with the application. The demonstrator then downloads the image from the URI, informs the user of this, and specifies that an email containing the results will be sent the next day.

We tested the system, and the email did indeed arrive the next day, with a hyperlink to the result image, as shown in Figure 1. Note the hyperlink to the numerical results. An example of these numerical results is shown in Table 1. While this data is useful, and could be used by an evaluation algorithm, it would be even better if it published in XML, instead of *ad hoc* plain text. The problem with plain text output is that it requires manual effort to write parsers for it, and is extremely version sensitive i.e. if a decision is made to change the format of the output or to add a new attribute (e.g. such as the time taken to process the image), then any programs used to read such data must be modified accordingly, which can be tedious, and failure to make such

[1] http://vasc.ri.cmu.edu/cgi-bin/demos/findface.cgi

modifications could lead to either to a program that fails to run, or worse still, or one that appears to run but produces incorrect results.

There are many positive aspects of this CMU-RI Demo. It has been up and running for many months, and has successfully processed hundreds of images. It is simple to operate, though rather slow, (this may have been done deliberately to reduce server load). The slow turnaround allows the possibility of human intervention in the generated results, a possibility that can in practical terms be avoided with an immediate turn-around.

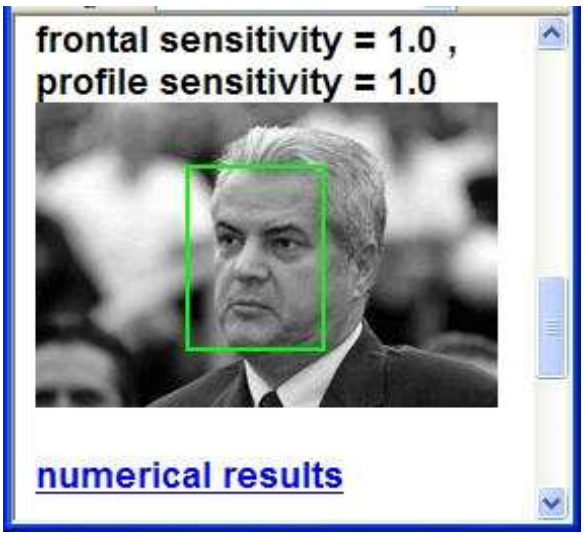

Figure 1. A sample gallery image from the CMU-RI web-based face detector demonstration.

```
1 -- number of faces

 1. Frontal Face
   107    76      -- Position
     6          -- size
    0.44       -- confidence
```

Table 1. Sample Numerical Output from the CMU-RI Face Detector Demonstration

Our system provides access to a deployed algorithm in two ways: interactive mode via a web browser, and program access mode via a special kind of web service architecture. Both deployment modes aim to offer immediate results, subject to server load. Web-browser mode is useful for users wishing to casually test systems on selected images. Program mode, on the other hand, can be used to test deployed algorithms on hundreds of test-cases, or even to build complete systems, where each component is a special kind of web service. The work reported here builds on previous efforts to deploy and evaluate pattern recognition algorithms using the Internet [7] or the Web (meaning HTTP over the Internet) [6]. Compared to previous efforts, the current work differs in two main ways. Firstly, the system is object-based rather than service based. This overcomes limitations in offering parameterized systems as services, since in our system, objects can be directly constructed using the same complex sets of parameters as for local objects. Secondly, the use of URIs to refer to all objects means that the results of all method invocations can be serialized to an XML file and stored on a web-site for future use. In addition to offering web-based access to client programs, we also think that it is important where possible to offer interactive access to deployed programs via a standard web browser.

The first author is currently in the process of evaluating the software submitted to the ICDAR 2005 text locating competition. Of the five entries received, only one of them worked first time. Others failed to operate due to missing library files, or compatibility problems with different versions of Java. While all the problems were eventually overcome, this costs effort both for the competition entrant and for the organizer. The web-based evaluation mode puts the responsibility for making the program run firmly with the entrant. Due to the effort involved in running competitions, for a given field these only tend to happen once every year, or perhaps once every two years (such as the highly successful Fingerprint Verification Contests [9]). The web-based deployment mode makes it practical for algorithms to be continuously deployed and evaluated. Web-based deployment has the potential to turbo-charge research in this area, by allowing good ideas that are well implemented to gain immediate acceptance by the research community.

The main contribution of this paper is to describe a relatively simple yet powerful system that we have developed for distributed object programming over the web. The system is called WOX, which stands for Web Objects in XML. The rest of the paper describes why existing distributed programming protocols are not yet ideal for our purposes, the ideas behind WOX, and how we have already used it to deploy a text locating algorithm as a web service. Note that the problem of finding all the text regions in an image is also referred to as *text detection*, but we prefer the term text locating, as detection could mean simply indicating whether or not an image contained some text.

2 Distributed Programming and Web Services

There are two goals to the work reported here: to make text-locating algorithms available to end users via web-browsers, and also to make them available to end-user client programs.

The latter mode requires some form of distributed programming, whereby a program on the client machine makes a remote method call or a remote procedure call to a remote machine, where for the current exercise, the remote machine can be situated anywhere on the Internet. Actually, we make a further requirement. Due to security measures, many institutions only allow calls to port 80 or 8080 through their firewall, typically to machines running web servers (such as Apache HTTPD), or web application servers (such as Tomcat).

While there are many remote procedure call mechanisms, such as PVM, MPI, RMI, etc. none of these are ideally suited to the task at hand.

The requirement that all traffic be routed through a web server or web application server is quite restrictive, and restricts us to using HTTP. While it is possible to tunnel CORBA or Java RMI calls through HTTP, this exercise may be technically quite demanding, and also rather opaque: we have developed a simpler, and more transparent system.

To gain a degree of language independence and HTTP compliance, we have settled on using XML to encode the procedure call and return messages. This also has the advantage of being human readable, which can aid debugging.

XML-RPC offers a simple mechanism for making remote procedure calls through the web, but only allows for a fixed set of data types to be used. This means that any application specific data types must be translated into the pre-defined types before transmission, which incurs an unnecessary overhead in deploying a new application in this way.

Simple Object Access Protocol (SOAP) might seem like the obvious choice. The acronym is something of a misnomer, however, since it is not simple, and it does not provide access to objects. In other words, there is no SOAP supported way for a client to instantiate an object on a remote server, and then subsequently refer to it just as they would a local object, although of course it is possible to program one's way around this if necessary. Having direct access to objects is important when dealing with stateful programs, but is not an issue when dealing with stateless ones. The text locating programs we have in mind for this application are expected to be stateless, since the result of processing an image is expected to be independent of the previous images seen.

If we were evaluating trainable text locaters, however, then maintaining the state of a text locater (i.e. what it had learnt during training) would be a vital consideration.

Unlike XML-RPC, SOAP does allow for user-defined (application specific) data-types. This is done by installing specific serializers for specific data-types. However, the default encodings used by SOAP for arrays of primitive data-types are extremely inefficient, and hence it is questionable as to whether SOAP would provide any benefits for this kind of application.

By default, SOAP uses an XML element for each element of an array of primitive elements (such as `int`). This means that SOAP-encoded arrays can be over 40 times the size of their binary encoding. The exception to this are byte arrays, that are encoded efficiently using base-64 (which WOX also uses for byte arrays). Given the speed of modern computers, and the fact that many of us have access to high bandwidth Internet connections, this difference in encoding efficiency might seem unimportant. However, table 2 emphasizes how significant this difference is, both in time and space usage. For arrays of more than $30,000$ `int`, the SOAP server crashed with an out of memory error.

method	Time (ms)	Size (kb)
WOX	80	106
SOAP	3,300	4,200

Table 2. Time and space usage for passing an array of $20,000$ `int` using WOX, and SOAP.

2.1 Representational State Transfer (REST)

Many analysts and developers have become extremely frustrated with SOAP's shortcomings, and there is significant interest in an alternative paradigm called REST, which stands for Representational State Transfer [4], [5]. Proponents of REST argue that what made the Web really take off was HTTP, and the notion of a URI - a Uniform Resource Indicator. This gave a standard way to refer to any item of information.

SOAP hides a set of services at a site behind a single URI endpoint used for remote procedure calls, and the details of the service required are encoded in the message instead of the URI.

The idea behind REST is that URIs should be used to name service, data structures and objects directly, in order to exploit the full power of the web. However, REST is an architectural style rather than a concrete protocol.

2.2 Web Objects in XML (WOX)

We have designed a new protocol for making remote method invocations over the web, and we call this WOX, for Web Objects in XML. WOX is based largely on the principles of REST, and each object on a server can be uniquely identified with a URI.

While SOAP does not allow references to objects (i.e. all parameters must be passed by value), WOX allows call by value or call by reference. Call by reference can in some cases make huge savings on network traffic.

The basic ideas behind WOX are as follows. A remote method call to a WOX server specifies the URI of the object (which could be local to that server or remote from it), the name of the method to invoke, and the parameters to that method. Each parameter can be passed by value, or by reference - again for references, URIs are used). The XML

encoding of the object specifies the object's class. This can then be used by the WOX server to load the appropriate class, deserialize the object of that class, and invoke the method on it. When the method invocation has finished, the WOX Server will then serialize the result to XML, which can either be sent directly back to the client, or be saved on the server, and it's URI sent back to the client.

The WOX system currently exists as an operational prototype, which has been implemented only in Java, though we expect that all the concepts used could be applied to any object oriented language.

3 Requirements

In designing our system we began with a set of requirements functional and non-functional requirements for our system to deploy text locating algorithms:

3.1 Functional

The functional requirements are as follows:

Interactive Testing The system must support an interactive mode of usage.

Client Program Access The system must support access to a text locating algorithm to a remote client program.

Graphical Results : the system must provide graphical results of running a text locater on an image.

XML results the system must provide detailed results in XML, which are stored on the web server for future reference and processing.

3.2 Non-Functional Requirements

Simplicity the system should be simple to deploy and use, requiring no expert knowledge of web services.

Efficiency The system should not impose any unnecessary CPU or network bandwidth overheads

Generality The system should be able to deploy algorithm implementations written in any language

Free / Open Source The system should not rely on any commercial tools - it must be free to deploy, and open source to enable others to extend it freely.

Platform Independent The system must be easily runnable on a variety of platforms. For example, a system implemented using Microsoft .Net would be unacceptable to the community.

Although still in prototype, the system already meets its main requirements. Both the web browser interface[2] and the WOX Server[3] are currently running and freely available to interact with as a service, and to download and install as one's own service.

4 The Text Locating Interface

We define the interface to a text locating algorithm in as simple a way as possible, defining data structures where necessary, but avoiding the use of Collection classes as far as possible, as these may be complex to serialize.

We use Java to define the interface, as this is the language that we work with most often, and the one we always use for prototype development. The fact the Java has an `interface` keyword makes the syntax especially appropriate. This does not mean, however, that service implementations are in any way restricted to be in Java, and non-Java programs can either by invoked by starting a new process from within Java, or by using the Java Native Interface.

Table 3 shows the Java interface for a text locating service. Note that this interface assumes that the text locater implementation is pre-trained. An interesting alternative would be to allow for a trainable interface, whereby images tagged with ground-truth rectangles could be uploaded to the service in order to train it. This would lead to two interesting possibilities: either that a common text locater object could be shared among all users, or that an individual text locater be made available to each user (or indeed many per user). The former allows for a community-wide training process reminiscent of the *OpenMind* concept [10], while the latter allows text-locaters to be trained and tested for individual needs.

```
public interface TextLocater {
   public Rect[] locateText(byte[] encodedBytes);
}
```

Table 3. Text Locater Interface.

The only class that the interface depends on is the `Rect` class, shown in Table 4. This simply codes the coordinates of a single rectangle. The interface specifies that a Text Locater should return an array of such rectangles. The actual `Rect` class used by client and server may differ, but providing that they use the same fields, they will be serialized and de-serialized in the same way by WOX, and hence be compatible.

5 The WOX Client-Server Architecture

Figure 2 shows the WOX system architecture, and how it relates to the text locating application. To interact with a

[2] http://algoval.essex.ac.uk:8080/textloc/

[3] http://algoval.essex.ac.uk:8080/WOXServer.jsp

```
public class Rect {
   int x, y;
   int w, h;
}
```

Table 4. The `Rect` class used to store rectangle coordinates.

text locating WOX web service, the client application program first instantiates an object of the appropriate class on the server. To do this, the client passes the class name of the object, together with any parameters, to the WOX server. The WOX Server then attempts to instantiate an object of that class, and then returns either a copy of the instantiated object, or a reference to the instantiated object. In our case, we wish to make all subsequent calls to the object on the server, so we specify that a reference should be returned.

When the WOX client receives the URI of the newly instantiated object, it instantiates a special proxy object, that implements the text locater interface, and will send all client-side method invocations on to the server object for processing. In the case of the rectangle results, in our client program we now wish to receive all the encoded rectangles directly (instead of just receiving URI references to them), so the WOX method call policy is specified accordingly.

Now the client has a reference to the server side text locater implementation object, it can call the `locate` method to find the rectangles in an image. When making this call, we suppose that the image is stored on the client as an array of byte. This is then encoded using the WOX serializer, and send to the WOX server as part of a WOX method call. Note that the client program does all the work through a proxy: it just sees the `textLocater` interface.

The WOX server then deserializes the XML method call to a set of Java objects, attempts to find the object to invoke the method on, makes the invocation, and in this case returns an array of `Rect`, serialized in XML. To get the required logging behaviour that would allow researchers with the aid of a web browser to interrogate the results of any text locating method invocations, we had to install an adapter for this service. The reason for using an adapter is that we wanted all the rectangles found by each text locater to be overlaid on a copy of each original image uploaded to the WOX server. This is not part of the normal WOX method invocation logging process: hence the need for an adapter, as shown in Figure 2.

5.1 XML Result File

Table 5 shows a sample XML result file (Figure 3 shows a different image with the rectangles overlaid on it). There are a number of issues that arise when choosing a format for the XML results file. Currently, our main emphasis is on rapid prototyping, so we are using the default WOX object serialization format to produce all our XML. This eliminates the necessity to produce helper classes to read and write XML from objects, and avoids the need to design any XML representations. The other issue that arises when producing the result images with the rectangles overlaid on them, is when to do this. Here, there are two choices. One option is to store only the original images, then dynamically overlay the rectangles on them for each request. This costs more time, and is potentially brittle, since the image results now requires special software in order to be properly interpreted. Here we chose the less space efficient but simpler option of saving the overlaid images immediately when the text locater has found the rectangles.

```
<object type="problems.roi.TextLocResult" id="0">
  <field name="elapsedTime" type="int"
         value="1063"/>
  <field name="rectangles">
    <array type="problems.roi.Rect" length="6"
           id="1">
      <object type="problems.roi.Rect" id="2">
        <field name="x" type="int" value="102"/>
        <field name="y" type="int" value="142"/>
        <field name="w" type="int" value="182"/>
        <field name="h" type="int" value="158"/>
      </object>
      <object type="problems.roi.Rect" id="3">
        <field name="x" type="int" value="101"/>
        <field name="y" type="int" value="98"/>
        <field name="w" type="int" value="226"/>
        <field name="h" type="int" value="114"/>
      </object>
      <!-- rest of array omitted -->
      </array>
    </field>
</object>
```

Table 5. An example XML results file created by the WOX Text Locating Service Adapter.

5.2 Deployment

We believe that a strong feature of WOX is the simplicity with which algorithms can be deployed. In order to deploy a Java algorithm as a text locater, for example (assuming that the WOX server has already been installed on a web application server such as Tomcat), one has only to copy the Java classes or `jar` file to the appropriate directory on the server (web-based upload is also possible, but we've not yet enabled this, as it as potential security risks). If the algorithm is not implementation in Java, then a Java wrapper can be used to interact with the implementation, either using the Java Native Interface, or by reading/writing files.

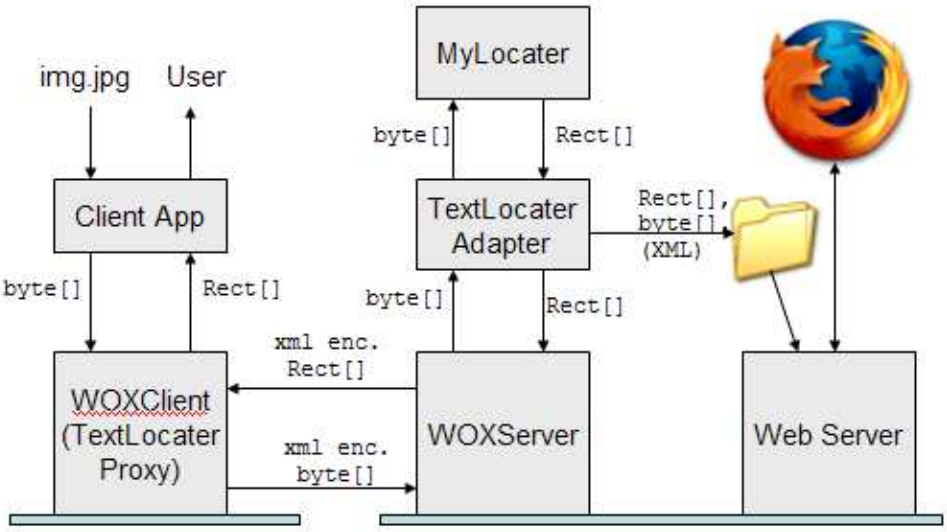

Figure 2. The WOX Client-Server Architecture.

6 Test Results

We tested the system first of all with some trivial text locaters, that would simply decompress an uploaded JPEG image. When the system passed this test, we then installed a non-trivial text locating algorithm with reasonably high performance. With the permission of Alex Chen, we installed his submission to the ICDAR 2005 text locating contest [3] [2]. Of all the submissions, this one is relatively fast, and had proven to be trouble-free in operation. As we only had access to the executable file, we used a file-based mechanism to interact with it. The `TextLocater` object that is exposed on the WOX server is a simple wrapper that takes in the submitted image, saves it to a file, invokes the Chen algorithm, then reads the results of the text detection (which are in XML), and returns the result to the adapter. Recall that the adapter then overlays the detected rectangles on the image, saves that image to a file (together with the XML), and returns the XML-encoded rectangles to the client program.

6.1 Evaluating Text Locaters

The system described so far caters for the web-based deployment and usage of algorithms, in this case, text locating algorithms. So far, we have made no mention of how the accuracy of these algorithms should be evaluated. Our proposal is to keep the usage and evaluation of these algorithms entirely separate. The way this works in our prototype is that the WOX text locating client uses the service to locate the text rectangles in a set of images. In order to assess the accuracy of the service, it is necessary to have access to the ground-truth data. If the client does have access to this, then it may run the necessary evaluation algorithms to score the algorithm. The separation of the running of the algorithm from its evaluation is a good idea, since it allows any number of evaluation algorithms to be run on the detected rectangles. This is especially important for text locating, where the community has yet to agree on the most appropriate measures. The write-up of the ICDAR 2003 text locating competition discussed two alternative metrics, for example [8].

7 Conclusions

We have described a relatively simple method for deploying text locating algorithms as interactive web applications, and as REST-based web services. We believe that web-based deployment of these algorithms is of great importance in order to speed up the way that new ideas are propagated and assimilated in the research community. It offers faster and easier access to newly developed programs than would be possible with other means.

The system is reasonably efficient, and stores all the results in XML files, and as JPEG images with the rectangles overlaid. The XML results can later be processed by a variety of evaluation algorithms, and the overlaid images can be visually inspected using a web browser.

The system is freely available, and we encourage researchers to use it to make their algorithms accessible in this way. Not only would this lead to more rapid dissemination and evaluation of new algorithms and their implementations, but it would also enable new systems to be constructed from a range of web-based components, by merely specifying the connections between the URIs of those components.

Figure 3. Sample image of shop front that has been uploaded to the Alex Chen text locating algorithm. Only a cropped version of the image is shown to save space, but the retrieval rate is very good, though there are a few false positives (which will lower the precision).

Acknowledgments

We thank Alex Chen for his cooperation in allowing us to install his text locating algorithm in our text locating web service.

References

[1] G. Bieber and J. Carpenter. Introduction to service-oriented programming (rev 2.1). `http://www.openwings.org/download/specs/ServiceOrientedIntroduction.pdf` .

[2] X. Chen and A. L. Yuille. Detecting and reading text in natural scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages II:366 – II:373, 2004.

[3] X. Chen and A. L. Yuille. A time efficient cascade for real-time object detection: with applications for the visually impaired. In *Proceedings of the CVAVI05, IEEE Conference on Computer Vision and Pattern Recognition Workshop*, page to appear, 2005.

[4] R. L. Costello. Building web services the rest way, Accessed May 2005. http://www.xfront.com/REST-Web-Services.html.

[5] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. University of California at Irvine.

[6] S. Lucas. Web-based evaluation and deployment of pattern recognizers. *Proceedings of International Conference on Pattern Recognition*, pages 419 –422, 2002.

[7] S. Lucas and K. Sarampalis. Automatic evaluation of algorithms over the internet. *Proceedings of International Conference on Pattern Recognition*, 4:434–437, 2000.

[8] S. M. Lucas and et al. Icdar 2003 robust reading competitions: Entries, results, and future directions. *International Journal of Document Analysis and Recognition*, page to appear, 2005.

[9] D. Maio, D. Maltoni, R. Cappelli, J. Wayman, and A. Jain. Fvc2000: Fingerprint verification competition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:402 – 412, (2002).

[10] D. Stork. The open mind initiative. `http://www.openmind.org`.