Towards Chaining Geo-Computational Web Applications Across Multiple Sites

Song Xianfeng, Kono Yasuyuki, Hirata Masahiro, Yanagisawa Masayuki, Cao Guifa[*]

Institute of Geography, Chinese Academy of Sciences, Beijing 100101, China[*]
Center for Southeast Asian Studies, Kyoto University, Kyoto 606-8501, Japan

Abstract

Geo-computational web applications, like other web applications, provide browsers with a human interactive interface that incorporates all the parameters required to access their backend functions. Through this interface, web geo-applications can serve users well within their own repositories. However, while a particular Geo-computational task may involve the cooperation of multiple repositories, there are no established methods to chain them together.

In this paper, we propose a lightweight function-oriented markup language, FuncTag, that can precisely define a programmatic web interface for aggregating web geo-applications across multiple sites. Accordingly, an intelligent task agent for interpreting FuncTag script is developed for Java, and data transfer is based on XML-RPC and HTTP protocol. Collaborating with the portable task agent, FuncTag markup tags may directly feed web geo-applications on user-targeted sources, automate interaction among those applications, and chain them together as a batched job does.

Keywords XML, Geo-Computation, Web Programmatic Interface, Intelligent Task Agent

1 Introduction

In the past, a great deal of effort was made to deliver GIS to the Internet, and GIS sites have spread spatial data everywhere online. The potential now exists to chain backend geo-applications or aggregate spatial data from those disparate web repositories. This task raises many complicated issues. Initially, each GIS web site is built for a specific purpose, and little attention is paid to cooperation with other sites. The generic difficulties of integrating those sites include the processing of the function entry, the encoding of the parameters, the transport method, and time outs for the web connection. The developers of geo-applications must consider those issues when constructing their applications with the goal of integrating sites. Furthermore, certain programs may be proprietary and may require maintenance each time one of the sites is changed.

To chain relevant geo-computational applications on the web for a specific task, the simplest way is to manually feed one application on the feedback from another application or the targeted resource downloaded from another site. Such an approach is verbose and violates our motivation towards function shipping and data sharing on the web. In addition, Web Interface Definition Language (WIDL), an XML-based language that implements a service-based architecture over the document-based resources of the Web, allows for interaction with web servers to be defined as functional interfaces that can be accessed by remote systems via standard web protocols (Philllip, 1997). However, WIDL script does not work by itself; it has to map its defined function into a conventional programming language code as IDL does. Those established applications are available only through the application developers mentioned above.

The most appropriate approach would be to build an intelligent task agent to automate the interaction among web geo-applications. This task agent would provide users with a simple programmatic interface which submits the request and accepts the response, instead of using human intervention and complex programming. Users would then be unaware of the implementation details, and just simply code their macroscripts to feed the task agent.

In order to achieve this, we present a programmable function way to aggregate the distributed geo-applications over the World Wide Web. A function-oriented markup language, FuncTag, is designed to describe the web programmatic interface. Users can code a simple FuncTag script to integrate web geo-applications for a specific task. An intelligent task agent, the counterpart of FuncTag, is developed to run the FuncTag script. The task agent incorporates the complicated implementations encountered within a distributed system, and leaves a simple programming entry open to the general public.

In order to attain FuncTag application independence and task-agent platform independence, portability and compatibility must be addressed for the designation and the implementation of the system. We adopt XML to define FuncTag and Java to code the task agent, because separately they are the portable data and the portable code, in other words, a perfect match for the above independence (Brett, 2000). To make the data exchange very efficient, task agents are designed for cooperation with each other. Within the task-agent system's architecture, agent only speaks HTTP with another's agent or web geo-application, which ensures that the entire system is web-enabled.

2 Web Programmatic Interfaces

2.1 Programmable Function vs. Web Application

A function within the conventional programming language is sometimes called a procedure or a subroutine; it encapsulates uniform code blocks for specific computational tasks. It is often used to break a large computation into smaller ones in a convenient manner in order to maintain the whole program. In recent years, the web has become a distributed application platform, and most web pages are generated dynamically from web applications, instead of from a static document system. Those web applications can be regarded as programmatic functions, while we treat the web as a universal computational context. This is because web applications maintain all the complex details of the implementation behind the web and just allow entry for input and output as the programmable function does in a conventional program. With respect to this, we define a programmatic web interface to automate interaction among web applications.

2.2 HTML Form Specifications

On the web, a web application employs a generic HTML form-presenting interface in order for users to manually pass their request to the server. The web application, or part of it, behaves as a form-processing servant that accepts submitted parameters and returns responses. The servant generally runs on the server side as a CGI, Fast CGI, Java Servlet, etc., and constructs its functional entry according to the following HTML form- interface specifications (W3C, 1998).

> 1) *Form controls* specify the valid name/value pairs as a form data set. The control vision may be different in type, but each successful control has its name paired with a value as part of the submitted form data set. (2) The *Form submission method* specifies the HTTP method used to send the form data set to the form processor. The HTTP methods' "post" and "get" are the best ways to transport uploaded data to remote processors (Fielding, 1997). (3) The *Form submission target* specifies the remote processor used to accept the form data set. It is a Uniform Resource Identifier (URI) wherein the form-processing servant resides. (4) *Form content types* have to be used to encode the form data set for submission to the server, as HTTP is a character-based transparent protocol. The "application/x-www-form-urlencoded" is the default content type for encoding the ascii data blocks and allowing the reserved characters to escape by HTTP. The "multipart/form data" is often used for large arbitrary binary data sets (Nebel, 1995). There are many other types available, but they seldom appear here.

Referencing the above HTML form declaration, a web programmatic interface is defined to automate interactions among web applications, instead of using human intervention. This renewal interface reserves the main perspectives of the HTML form and presents its structure with eXtensible Markup Language (w3c, 2000a). With the help of this interface, users can write simple programming scripts to incorporate parameters and other properties that are necessary to drive the desired web applications. The execution of the scripts is finished within the intelligent task agent where the dedicated runner interprets the user-coded scripts and interacts with web applications automatically.

2.3 DTD Definition of Web Programmatic Interface

We use a lightweight XML-based markup language, FuncTag, to complete the web programmatic interface and its basic syntax and usage. All the specifications are towards function prototype and function calling; there is no control flow or complex expression. FuncTag Document Template Definition (DTD), rather than XML schemas, describes in detail the encoding of each element, because FuncTag is XML v1.0 compliant and W3C XML schemas' specifications are still candidate recommendations. It is expected that FuncTag will be improved in the future following in the footprints of XML schemas.

```
<!-- ==================================================================
 This is the DTD for Function Markup Language (FuncTag) (draft 2001/01/20).
 Copyright (c) 2001 CSEAS, All Rights Reserved.
 For this working draft:
  Namespace:    http://130.54.102.241/2001/funcml
  URI for the DTD: http://130.54.102.241/2001/funcml/dtd/funcml-20010120.dtd
================================================================== -->

<!-- common attributes for reference -->
<!ENTITY %stdAttrs
“id ID #IMPLIED
idref URI #IMPLIED”>

<!-- function attributes -->
<!ENTITY %funcAttrs
   ”url CDATA #IMPLIED
   enctype (application/x-www-form-urlencoded | multipart/form-data) #IMPLIED
   method (get|post) #IMPLIED”>

<!-- the root element holding all other data content -->
<!ELEMENT FUNCML:FuncTag ((FUNCML:Param | FUNCML:Function)*,FUNCML:Main) >
<!ATTLIST FUNCML:FuncTag
   xmlns:FUNCML CDATA #REQUIRED>

<!-- main function entry like the main() in ANSI C -->
<!ELEMENT FUNCML:Main (FUNCML:Param)*>
<!ATTLIST FUNCML:Main
   %stdAttrs;
   % funcAttrs ; >

<!-- generic function entry or function reference -->
<!ELEMENT FUNCML:Function (FUNCML:Param)*>
<!ATTLIST FUNCML:Function
   %stdAttrs;
  % funcAttrs; >

<!-- parameter or its reference -->
<!ELEMENT FUNCML:Param (FUNCML:Name,FUNCML:Value)?>
<!ATTLIST FUNCML:Param
  %stdAttrs; >

<!-- argument name -->
<!ELEMENT FUNCML:Name (#PCDATA)>

<!-- argument value -->
<!ELEMENT FUNCML:Value ( FUNCML:Literal | FUNCML:URI | FUNCML:Function ) >
<!ATTLIST FUNCML:Value
  type (ascii | binary) #REQUIRED >

<!-- simple literal -->
<!ELEMENT FUNCML:Literal (#PCDATA)>

<!-- reference of web source -->
<!ELEMENT FUNCML:URI EMPTY>
<!ATTLIST FUNCML:URI
   href CDATA #REQUIRED >
```

In this specification, the value part of the name/value parameters is enriched with a supplementary value reference in order to code functional scripts in a flexible programming manner. We define two atomic data types, namely, "ascii" and "binary", and three schemes for data location representation, namely, simple string (Literal), reference to web resource (URI), and reference to the feedback from another function calling (Function). The references to parameters or functions make it possible to feed web applications on a user-targeted web source or to pipe one application's output to another's.

It comes as no surprise that this specification is quite simple, but very practical, in aggregating web applications. It supports the organization of compact function structures in a recursive way in which the function embeds all its declared parameters and also those parameters that may embed other function declarations again. Furthermore, the arameters are also allowed to be declared outside of their function body and referenced by several functions; this makes parallel function structure possible. In the next section, we will illustrate how an intelligent task agent takes off FuncTag scripts independently or cooperatively.

3 Intelligent Task Agent

3.1 Basic Procedure For Processing Tasks

FuncTag script, standing for a computational task, is firstly parsed into a DOM Tree. Then, the entire task is split into a set of sub-tasks with respect to the function or to the parameter node within the DOM Tree. Accordingly, a series of function and parameter objects are generated where each object stands for a particular sub-task. Those objects usually reside on differing sites and are nowadays referred to as RPC handlers. They consist of a task-oriented distributed context at the object level rather than at the application level. Each object contains all its properties and a set of methods ruling its interaction with other remote objects. They can locate each other for computational cooperation by the name lookup service. Once a running context is ready, the task starts while the object of its main function is invoked; other objects will be triggered later according to their dependency. If an object depends on another object's feedback, it will go to sleep after invoking its dependants, and will continue its work until notice arrives from its last dependant. The computation continues to stop at the main function object and returns to the user as the last result.

3.2 System Architecture

The FuncTag task agent is an intelligent agent system in a distributed environment on which FuncTag scripts depend for their concrete execution. The system architecture is shown in Figure 1. It is a typical client/server structure that consists of a task-feeding client and a task-processing server. The core part, the task-processing server, includes five major components, namely, the FuncTag Parser, the RPC-Handler Management Service, the RPC-Handler Pool, the Intra-Communication Bridge, and the HTTP Form Agent.

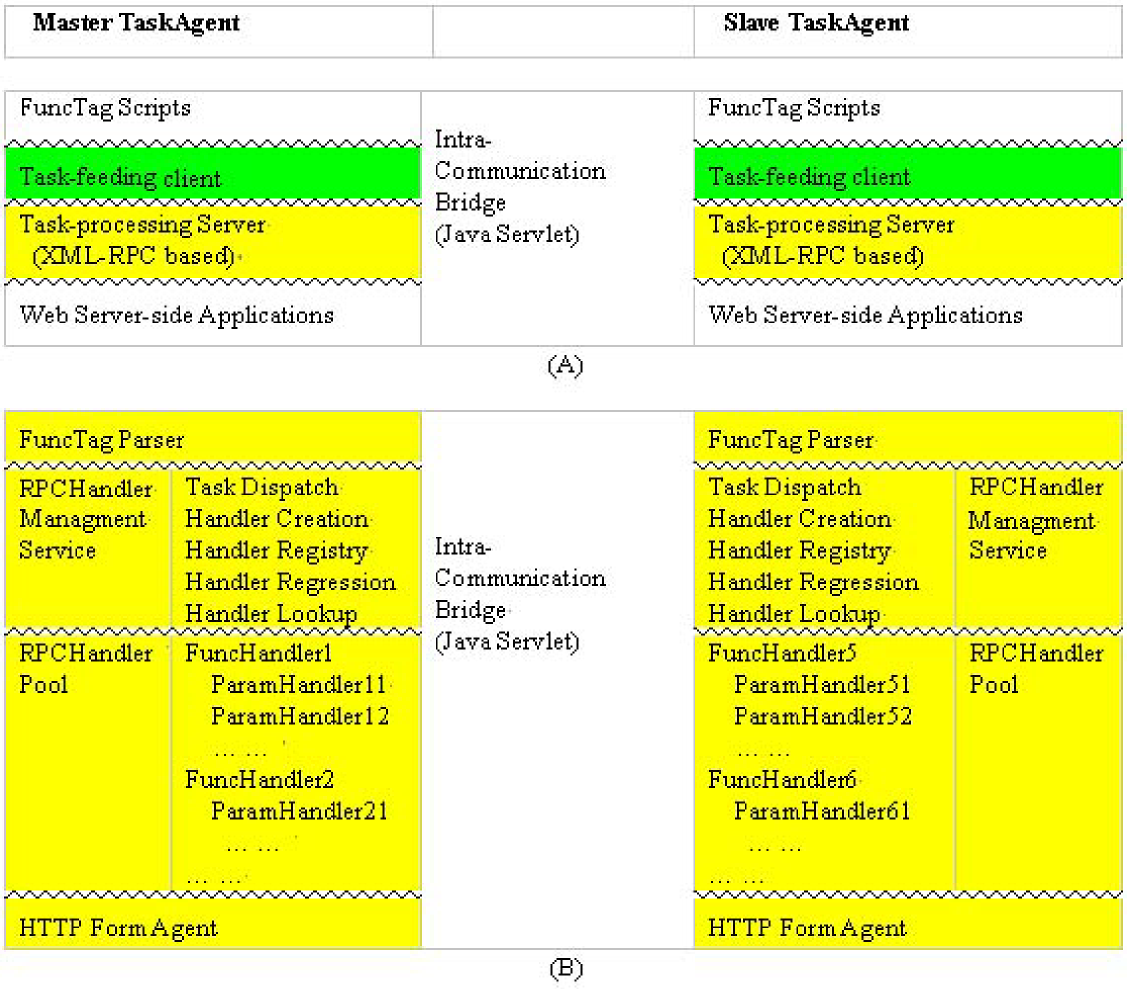

Figure 1 Architecture of the Intelligent Task Agent

The task-feeding client provides users with an entry to feed the task-processing server the FuncTag scripts and to fetch the computational results. This client may be either a standalone program or a server-side servant; both can connect to the task-processing server via HTTP protocol.

The task-processing server is a simple HTTP server embedding the XML-RPC components where the computational task has actually been executed (while the term “task” is mentioned, we are referring to the task described within FuncTag scripts). When several task agents work on the same task, the task-processing server, which has received the task scripts, behaves as a master; the others work as slaves that have to follow the master’s directives and retrieve its lookup service. When there is no slave available, the master can work well by itself, except that it will do so with low efficiency. The cooperative computation between multiple task agents only occurs while some task-feeding servers are being hosted by the web servers where the web applications that have been coded into FuncTag scripts also reside.

The FuncTag Parser is an XML parser that is responsible for checking the validity and the well-formality of the FuncTag script, and for parsing the FuncTag XML content into the standard DOM Tree.

The RPC-Handler Management Service is a set of permanent RPC handlers that are charged for the services of managing other dynamically generated RPC handlers. After receiving the FuncTag DOM Tree, built by the parser, the task dispatch service regenerates the pedigree for each function and parameter according to the function or the parameter node within the DOM. Each function or parameter is assigned a unique identifier (ID) which is generated with CRC32 methodology. Then, the slave server is detected. If a slave exists, the creation service of an RPC handler will be invoked to create and initialize the handler with an ID and a pedigree for the corresponding function or parameter. Otherwise, the master server will do it by itself. RPC-handler registry services on/for the master will memorize the IDs and host servers of all the RPC handlers so that the lookup service can serve queries on those handlers' positions. After completing the computational task, all the handlers associated with that task will be removed by the RPC-handler regression service by the master or a slave.

The RPC-Handler Pool hosts all the functions and the parameter handlers generated dynamically, keeps track of their active status, and maintains the integrity with the Handler Registry Table on the master server.

The Intra-Communication Bridge only becomes active while the cooperative task agents are separated by the firewall where links between task agents cannot be built. Each task agent also employs HTTP protocol, but it has to choose another port number (i.e., 8585), except for official 80. While the firewall closes those illegal ports, the Intra-Communication Bridge may act as a proxy to accept input from another task agent via the normal HTTP/80 and then re-feed it to the accompanying task agent.

The HTTP Form Agent can directly automate interaction among web applications as a HTTP browser does. It is an external service that helps task agents interact indirectly with web backend applications. While the Function Handler completely prepares its parameters, the Form Agent will be invoked. It will encode all the name/value pairs with the specified content type, open the HTTP connection to the web server on which the application resides, pass on the encoded content, and receive the response via the specified HTTP method.

3.3 Miscellaneous Perspectives

(1) Communication protocol

The distributed application is similar in behavior to a communication system in which data and directive transmissions have to depend on certain specifications. It is difficult to say whether one protocol is better than the other. Which should be selected depends on the targeted application. The task agent chooses XML-RPC as its protocol, instead of the popular CORBA (Letho, 1998), for many reasons. RPC allows literally any type of application intercommunication, because its transport protocol can be HTTP, while CORBA allows for the use of IIOP for connecting CORBA clients and its servers (OMG, 1997). As is well known, HTTP is the protocol with the most legacy for passing the firewall. RPC is more lightweight than CORBA, because CORBA often has to load an entire object over the network, whereas RPC only has to pass across the request input and the resulting output. RPC has also jumped over its traditional obstacle of encoding while XML provides a simple textual representation of data (Userland, 2000).

(2) Trigger-style function calling

The looseness of the web often causes time out errors while the browser fetches a remote source. Programs trying to link several sites and pipe them together may fail in cases where one of the web connections is broken. We apply multiple threads and function triggers to improve the performance of our task agent. The parent function always goes to sleep while its dependants are working as the new threads. The parent does not wake up to build its own web connection for data fetching until all the dependants prepare the required data content. This ensures that only the active function opens its needed web connection and no connection is holding online waiting for another function. Therefore, the problems caused by time outs can be resolved here.

(3) Cooperation between multiple task agents

Cooperation between multiple task agents may make task performance very efficient. Suppose that only one task agent is involved in several function callings. All the function inputs and outputs have to pass that agent, because it is the central processing unit that controls all the data flowing between the functions. While the FuncTag script links several functions that are located at different web sites, the central task agent has to accept one site's output and then forward it to another site. If there are other task agents residing on the relevant sites, the site-to-site transmission can bypass the central task agent and greatly improve the data transportation.

(4) Portability from Java and XML

To enable the task agent to work on multiple platforms and interoperate completely, we employ both Java and XML to construct the system. Many articles have been written which mention the outstanding aspects of Java and XML technology, but the authors here just want to emphasize their portability, which is the real reason why we apply them for system building. Java's portability enables an application to run on multiple platforms, but the data exchange among those applications suffers from the proprietary data formats.

XML promises to bring to a data format what Java brought to programming language, namely, complete portability (Brett, 2000). Since XML allows systems to communicate using a standard means of data representation, it is really an ideal counterpart for Java. Therefore, the perfect match between the portable code and the portable data adds to our application along with the dramatic ability of the system-independent and standards-based data transportability.

4 Performance Validation and Analysis

4.1 Performance Validation

The following example is borrowed from research material on the vegetation distribution in the Abdal Aziz Mountain region of northeastern Syria. The thematic map on vegetation was achieved from the classification of the Landsat TM5 image and is stored in the ArcView Shape format. In order to allow users to understand the vegetation coverage by its background, we provide users with information on the housing, traffic, and water distribution in that area, which were retrieved from the DEMIS mapping server by following the Open GIS web mapping server interface implementation specifications.

Centering on this topic, we construct three toy-like web servants for data format transmissions and spatial data overlay. (1) A GML generator is applied to translate the ArcView shape data set into the Geographical Markup Language (GML) format that is the recommendation specification of Open GIS for both spatial data storage and transport (Ron, 2000). There is no additional rendering material available in GML coding, just the necessary data-shelf markup and the data contents. (2) The SVG generator is built to transform the GML data set into the Scalable Vector Graphics (SVG) format that is the candidate recommendation of the W3C organization for rendering two-dimensional graphics in XML (W3C, 2000b). We employ the XSL Transformation (XSLT) to complete the conversion between GML and SVG.. (3) The SVGZ generator is coded to combine a SVG map and an image into a compact SVG document with gzip and zlib compressions, where the image exactly stands for the same region as on the SVG map and becomes the background for the newly generated map.

To achieve a SVGZ map that integrates the vegetation coverage and the background information, there are several steps needed while manually interacting with those relevant web programs. (1) We need to obtain the vegetation coverage shape file and background image separately from our public server and DEMIS server. (2) The shape file should be fed to the GML generator for the GML code. (3) The SVG generator is employed to convert the GML code into the SVG code again. (4) Both the SVG code and the background image are fed to the SVGZ generator for the expected results.

To validate our designation and system implementation, we can code the following FuncTag scripts to automate the interactive task mentioned above. The whole package of the intelligent task agent and this example are available at the site, namely, http://photolib.cseas.kyoto-u.ac.jp/funcvm/funcvm.tar.

4.2 Performance Analysis

As mentioned in Section 3.3, several task agents cooperating with each other can make the performance efficient. There is no reduction in the quantity of the data exchange, because the computational procedures among multiple task agents are not any different from those within a single task agent, except that the function objects are distributed among separate task agents. The good efficiency is mainly achieved from the optimal communication, although there are a number of other related factors.

In general, the master task agent has a different host server than the web geo-computational function, and the communication between the master and the geo-function may be called inter-communication across web sites. When there is a slave task agent installed on the web server, where the geo-function also resides, the slave may feed the geo-function and receive its feedback instead of the master. At the same time, the inter-communication between sites also becomes the intra-communication within the same site. The cooperative task agents may then improve the task performance, since the intra-communication is usually much faster than the inter-communication. Certainly, there is another negation that was not mentioned above. The slave has to pass on the feedback of its function to the next slave on another site, but this transmission would occur within the master site if the slave were not available. This means that intra-communication is changed into inter-communication. Since the interaction between the slave task agent and the web geo-function includes round data transmission, input/output, and data transport between task agents which is one way, the summary should be positive while the cooperation occurs.

5 Conclusion

In this paper, we have proposed a programmatic web interface and have implemented an intelligent task agent based on Java, XML-RPC, and HTTP technology. They cooperate with each other to provide a simple programmable way to chain multiple geo-computational web applications across separate sites. The system architecture of the intelligent task agent is comprised of the FuncTag parser, the HTTP form agent, and the functional handlers; it provides a distributed computational context at an object level. The system implementation also facilitates the best portability. Our ongoing and future work consists of the following issues. One is the improvement of the programmatic interface description while XML schemas become mature. The second is the fact that security has to be considered in order to keep the task agent from being used against its host or other systems.

6 Acknowledgments

This research itself was part of the on-going project titled “The Construction of Metadatabase in Southeast Asian Studies” supported by JSPS Foundation. And the dataset for validation was from the project of “The Study on Vegetation Distribution in the Dry Area of Northeastern Syria”.

7 References

Brett M, 2000, Java and XML, O’Reilly & Associates, Inc. Pp. 277-316

Fielding R, Irvine UC, Gettys J and Mogul J etc. January 1997, Hypertext Transfer Protocol -- HTTP/1.1, http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2068.html

Letho, L., 1998, Java/CORBA interation – a new opportunity for distributed GIS on the Web. In Proceedings of the ACSM Conference (Baltimore: ACSM), pp.474-481

Nebel E, Masinter L and Xerox Corporation, November 1995, Form-based File Upload in HTML, Http://www.cis.oio-state.edu/htbin/rfc/rfc1867.html

OMG (Object Management Group), 1997, What is CORBA? Document on the Internet, http://www.omg.org/corba/whatiscorba.html

Phillip M and Charles A, 1997, Web Interface Defination Language (WIDL), http://www.w3.org/TR/NOTE-widl

RON LAKE and ADRIAN CUTHBERT, 2000, Geography Markup Language (GML) v1.0, OpenGIS Consortium, Inc. OGC Document Number: 11-029

USERLAND, 2000, XML-RPC Library for Java, http://classic.helma.at/hannes/xmlrpc/

W3C (World Wide Web Consortium), 1998, HTML 4.01 Specification. W3C Recommendation, revised on 24 December 1999. http://www.w3.org/TR/1999/REC-html401-19991224/interact/forms.html

W3C (World Wide Web Consortium), 2000a, Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, revised on 6 October 2000. http://www.w3.org/TR/2000/REC-xml-20001006

W3C (World Wide Web Consortium), 2000b, Scalable Vector Graphics (SVG) 1.0 Specification, W3C Candidate Recommendation, revised on 2 November 2000. http://www.w3.org/TR/2000/CR-SVG-20001102/