OSGeo

Your Open Source Compass

OSGeo Journal Volume 8

February 2011

FOSS4G 2009
Conference Proceedings

OSGeo Community
News & Announcements
Case Studies
Integration Examples

FOSS4G
DENVER 2011
SEPTEMBER 12-16

The Annual International

FREE & OPEN SOURCE
SOFTWARE FOR GEOSPATIAL

Conference Event

2011.FOSS4G.ORG

OSGeo
Your Open Source Compass

Volume 8 Contents

From the Editor

OSGeo has just past its 5th birthday, along with this 8th volume of the OSGeo Journal! With this edition we bring a few news headlines from the past couple months, a few general articles and, most significantly, several top papers from the **FOSS4G 2009** conference event held in Sydney, Australia.

The Journal has become a diverse platform for several groups and growth in each area is expected to continue. The key groups that read and contribute to the Journal include software developers sharing information about their projects or communities, power users showing off their solutions, academia seeking to publish their research and observations in a peer-reviewed, open source friendly medium. OSGeo also uses the Journal to share community updates and the annual reports of the organisation.

Welcome to those of you who are new to the OSGeo Journal. Our Journal team and volunteer reviewers and editors hope you enjoy this volume. We also invite you to submit your own articles to any of our various sections. To submit an article, register as an "author" and sign in at http://osgeo.org/ojs. Then when you log in you will see an option to submit an article.[1]

We look forward to working with, and for, you in the upcoming year. It's sure to be an interesting year as we see OSGeo, Open Source in general and all our relate communities continue to grow. Nowhere else is this growth more apparent than at our annual conference: **FOSS4G 2011 Denver**, September, 2011.[2] Keep an eye on your OSGeo mailing lists, blogs and other feeds to follow the latest FOSS4G announcements, including the invitation to submit presentation proposals.[3] It will be as competitive as ever to get a speaking slot, so be sure to make your title and abstract really stand out.

Wishing you the best for 2011 and hoping to see you in Denver!

[illegible]

Tyler Mitchell
tmitchell@osgeo.org
Editor in chief, OSGeo Journal
Executive Director, OSGeo

[1]The direct URL for article submission is: https://www.osgeo.org/ojs/index.php/journal/author/submit
[2]FOSS4G 2011 Denver: http://2011.foss4g.org
[3]FOSS4G 2011 Abstract Submission: http://2011.foss4g.org/program

FOSS4G 2009 Conference Proceedings

From the Academic Track Chair

Prof. Thierry Badard

The FOSS4G 2009 academic track aimed to bring together researchers, developers, users and practitioners – all who were carrying out research and development in the free and open source geospatial fields and who were willing to share original, recent developments and experiences.

The primary goal was to promote cooperative research between OSGeo developers and academia, but the academic track has also acted as an inventory of current research topics. This track was the right forum to highlight the most important research challenges and trends in the domain and let them become the basis for an informal OSGeo research agenda. It has fostered interdisciplinary discussions in all aspects of the free and open source geospatial domains. It was organized to promote networking between the participants, to initiate and favour discussions regarding cutting-edge technologies in the field, to exchange research ideas and to promote international collaboration.

In addition to the OSGeo Foundation[23], the ICA (International Cartographic Association) working group on open source geospatial technologies[24]) was proud to support the organisation of the track.

The coordinators sought to gather paper submissions globally that addressed theoretical, technical, and practical topics related to the free and open source geospatial domain. Suggested topics included, but were not limited to, the following:

- State of the art developments in Open Source GIS
- Open Source GIS in Education
- Interoperability and standards - OGC, ISO/TC 211, Metadata
- Spatial Data Infrastructures and Service Oriented Architectures
- Free and open source Web Mapping, Web GIS and Web processing services
- Cartography and advanced styling
- Earth Observation and remote sensing
- Spatial and Spatio-temporal data, analysis and integration
- Free and Open Source GIS application use cases in Government, Participatory GIS, Location based services, Health, Energy, Water, Urban and Environmental Planning, Climate change, etc.

In response to the call for papers, 25 articles were submitted to the academic track. The submissions were highly diversified, and came from USA, Canada, Thailand, Japan, South Korea, Sri Lanka, Australia, New Zealand, Italy, Denmark, France, Germany, Switzerland, Romania and Turkey. Selection of submissions were based on the full papers received. All submissions were thoroughly peer reviewed by two to three members of the international scientific committee and refereed for their quality, originality and relevance. The scientific committee selected 12 papers (48% acceptance rate) for presentation at the FOSS4G 2009 conference. From those, 6 papers were accepted for presentation in the proceedings of the academic track, which are published in this volume of the OSGeo Journal. They correspond to the 6 best papers assessed by the international scientific committee.

The accepted and published papers covered a wide

[23]OSGeo: Open Source Geospatial Foundation: http://osgeo.org
[24]ICA open source working group: http://ica-opensource.scg.ulaval.ca/

range of cutting-edge research topics and novel applications on Free and Open Source Geospatial technologies. I am particularly proud and happy to see some very high quality scientific contributions published in the OSGeo Journal. This will undoubtedly encourage more interesting research to be published in this volume, as our OSGeo journal is an open access journal. In addition, it helps draw attention to this important project of the OSGeo Foundation. I hope the publication of these proceedings in the OSGeo journal will encourage future scientists, researchers and members of academia to consider the OSGeo Journal as an increasingly valuable place to publish their research works and case studies.

As a concluding note, I would like to take the opportunity to thank the individuals and institutions that made the FOSS4G 2009 academic track possible. First, I would like to thank the international scientific committee members and external reviewers for evaluating the assigned papers in a timely and professional manner. Next, I would like to recognize the tremendous efforts put forward by members of the local organising committee of FOSS4G 2009 for accommodating and supporting the academic track. Finally, I want to thank the authors for their contributions, efforts, patience and support that made this academic track a huge success.

January, 2011
Prof. Thierry Badard
Laval University, Canada
Chair, FOSS4G 2009 Academic Track
Co-chair, ICA Working Group on Open Source Geospatial Technologies

Geoprocessing in the Clouds

Bastian Baranski, Bastian Schaeffer, Richard Redweik

Abstract

Cloud Computing is one of the latest hypes in the mainstream IT world. Spatial Data Infrastructures (SDIs) with its classical publish-find-bind paradigm have not been affected yet by this emerging trend. This paper reviews this novel technology and tries to identify the paradigm behind it. In particular, the scalability aspect for a cloud enabled 52°North Open Source Web Processing Service is challenged and proven in the exemplary Google Cloud. On this basis, future direction for SDIs and the Cloud Computing paradigm are identified.

Introduction

Cloud Computing is one of the latest trends in the mainstream IT world (5). The term Cloud Computing uses a cloud metaphor to represent the internet or other large networking infrastructures. From a provider perspective, the key aspect of the cloud is the ability to dynamically scale and provide computational power, storage, and other applications, even complete infrastructures in a cost efficient and secure way over the internet. From a client perspective, the key aspect of a cloud is the ability to access the cloud facilities on-demand without managing the underlying infrastructure and dealing with the related investments and maintenance costs.

However, existing Spatial Data Infrastructures (SDI) are mostly focused on data retrieval and data visualization (8). Migrating the data processing part from classical desktop application to a distributed environment could be regarded as the next step. The step after migrating to a distributed environment would be the adoption of Cloud Computing principles. While the processing part in SDIs has already been tackled (12) (3) (13), Cloud Computing has not been regarded in the context of SDIs yet. This was the starting for this paper to explore the capabilities of Cloud Computing with a special focus on the processing part in SDIs.

In general, there are two options for realizing Cloud Computing in SDIs. First, adopting Cloud Computing principles and standards to SDIs. Second, migrating SDI elements amongst other services on top of a Cloud Computing infrastructure. Following the first option, the geodomain would once again create their own separate standards and markets and therefore establishing new barriers for utilizing SDIs. From our perspective the second option would be more effective and would allow the Geoinformation (GI) domain to be open to the mainstream IT world and thereby broaden the limited GI market. By leveraging these core propositions, we believe that the paradigm behind the Cloud Computing buzzword is promising for geospatial applications in order to enable new and promising business models for building up, operating and utilizing SDIs. In order to get hands-on experience, the 52°North WPS implementation [25] was migrated as a proof-of-concept study into the Google Cloud (namely the Google App Engine platform).

The remainder of this paper is structured as follows. First, a review of the basic concepts and related technologies is provided. This is followed by a description of the technical concept of the WPS migration into the Google Cloud. In the next section, our technical concept is evaluated in terms of scalability as one of the key aspects of Cloud Computing. Finally, the paper ends with a conclusion about the described framework and a discussion about interesting topics for a further research agenda.

Background

This section provides a review of related work in the context of Cloud Computing and SDI concepts.

Cloud Computing

Cloud Computing is one of the latest trends in the mainstream IT world (4) (5) and several companies such as Amazon, Google, Microsoft and Salesforce have already build up significant effort in this direction. The term Cloud Computing uses a cloud metaphor to represent the internet or other large networking infrastructures and the paradigm behind the buzzword hints at a future in which the storage of data and computations are no longer performed on local computers, but on distributed facilities operated by third-party storage and computational utilities (2). The term Cloud Computing overlaps with some concepts of Distributed Computing and Grid Computing (6). Grid Computing and Cloud Computing are both scalable infrastructures and provide sufficient computational resources like storage or computational power. But the target audience of Grid Computing is typically the scientific community running large-scale simulations and resource- and time-consuming applications (for example a global climate change model or the aerodynamic design of engine components), whereas with Cloud Computing small and medium-sized companies can scale their web-based applications in an instant fashion without having to invest in infrastructure for storing or processing large amounts of data (10). Furthermore, national

[25] http://www.52north.org/wps
[26] http://lcg.web.cern.ch/LCG/

and international Grid infrastructures (for example the Worldwide LHC Computing Grid [26]) are typically governmentally funded and driven by international joint research projects (for the example the Large Hadron Collider, LHC project at CERN [27]), whereas cloud infrastructures are operated by large enterprises under economic aspects.

Characteristics

The key characteristics of the cloud are the ability of datacenter providers to scale and provision computational resources, storage, and other applications even complete infrastructures dynamically in a cost efficient and secure way over the internet. Besides the consumer is given the ability to use these resources without having to manage the underlying complexity of the technology. These characteristics open up new perspectives for tackling different problems and lead to the following set of core value propositions.

Efficiency Cloud Computing enables IT organizations to increase hardware utilization rates enormously and to scale up to massive capacities in an instant without heavily investing in infrastructure in advance. Datacenter providers are now able to utilize their infrastructure more efficiently by dynamically distributing their applications and processes to free available resources.

Outtasking By outtasking software and data to scalable facilities operated by third parties, users and customers don't have to operate their own datacenters anymore. Therefore, enterprises of all types - from Web 2.0 startups to global enterprises - can decrease their infrastructure costs enormously. They can take advantage of transforming their fixed IT costs into variable costs as a business advantage by focusing on their core business (rather spending time on developing mature software and innovative business models than managing their physical hardware and purchasing costly licenses for rarely used software).

Scalability The allocation of cloud resources (for example high capacity storage or computing power) is done in real-time and most cloud infrastructures scale the deployed applications automatically on demand (for example in case of high request rates). This gives cloud users and cloud application providers the option for handling peak load very efficiently without operating their own datacenter and without managing their own infrastructure. For example, load-balancing or the development of high availability solutions for their software does not need to be regarded because it is provided by the cloud implicitly. By deploying their software and data in the cloud, they are automatically able to scale up their business capacities (for example from a few to hundreds of servers) in an instant and on demand fashion.

On-demand Allocating cloud resources on a real-time and on-demand basis helps enterprises to scale up their business capacities in an instant and efficient way. The absence of long-term contracts in combination with pay-per-use revenue models allows the low-cost start-up of new ideas for business models. The total cost of ownership (including hardware, software licenses, energy, fail-safety and technical engineers) of self-hosted datacenters minimizes start-up costs and helps enterprises to put new promising business models into the market.

Additional features of Cloud Computing infrastructures are the application of Service Level Agreements (SLA) defining service quality guarantees and contractual penalty clauses if the providers fail to meet the guaranteed service quality goals. Such contracts are important for cost-performance ratio transparency and therefore an essential skill for all kinds of IT and in this sense also IT based geospatial business models.

In essence, Cloud Computing is not a completely new concept. It moreover collects a family of well known and established methods and technologies (for example SaaS as a model for software packaging and deployment and Virtualization as an efficient hosting platform (7)) under the umbrella of the term Cloud Computing. Besides, it describes a paradigm of outsourcing applications and specific tasks to a scalable infrastructure and therefore consequently enabling new business models with less up-front investments. Keeping in mind that these technologies and general concepts existed in the IT industry for years, the emergence of high network bandwidth and mature virtualization technologies has now enabled this paradigm for a broader audience and leads to new application development models.

There are still a number of open issues for Cloud Computing. One deals with the general barriers of adopting Cloud Computing and is examined for example in the so-called "Open Cloud Manifesto". Beside data backup and recovery responsibilities the outsourcing of confidential and economically relevant data from data owners facilities to third party infrastructures is problematic in context of trust. Using public clouds as a deployment platform for applications and services in a risk management scenario is already a security issue in situations when the underlying cloud suffers an outage. But the problems regarding outsourcing of data and reliability of infrastructures are not specific only for cloud infrastructures. They must be addressed for all kinds of distributed architectures.

Projects and Initiatives

A lot of enterprise corporations are trying to get into the Cloud Computing business by offering services to ac-

[27] http://lhc.web.cern.ch/lhc/

cess their huge and over years grown infrastructures to the public. Microsoft with the Azure Services Platform [28] and its upcoming operating system Windows Azure [29] for operating cloud infrastructures, IBM introduced their "Blue Cloud" platform [30] and SUN for example offers Cloud Computing solutions as well [31]. In this chapter we describe the two cloud solutions from Amazon and Google more detailed, showing clearly that cloud providers could realize the different layers and characteristics of a cloud infrastructure at a different level of detail.

The Amazon Web Services (AWS) product is a collection of services that are offering Infrastructure as a Service (IaaS), Datastorage as a Service (dSaaS) and some aspects of Platform as a Service (PaaS). The Amazon Elastic Compute Cloud (Amazon EC2) provides a web service interface to manage virtual machines (IaaS) that are used to host customer specific applications and can be scaled on-demand to handle peak load. The Amazon Simple Storage Service (Amazon S3) provides a web services interface that can be used to store and retrieve large amounts of data (dSaaS). The Amazon Elastic MapReduce is a web service that offers computational power to process efficiently vast amounts of data. It utilizes the Hadoop [32] framework and dynamically distributes data and processing tasks across an automatically scaled cluster of computation nodes.

In contrast of AWS, the Google App Engine is an adequate example for pure PaaS. The Google App Engine provides a sandbox for running Java- and Python-based web applications. The web applications are deployed on the Google infrastructure and so they can take advantage of the same scalable and load balancing technologies that Google applications are built on. On the one hand, the key advantage of Google App Engine over AWS is that Google App Engine offers an easy way of deploying web applications in the cloud. In particular, the overhead of dealing with virtual machines and entire (virtual) server systems could be neglected. The Google App Engine offers also a data storage service (dSaaS) and different bindings to existing Google applications for authentication and accounting. Besides, the free default quota for testing purposes (for example data transfer and CPU time) lowers also the barrier for a first trial experiment. On the other hand, applications deployed in the Google App Engine are restricted to a specific (Java- or Python-based) application framework that runs in a restricted sandbox. This sandbox forbids the creation of threads and the web service request duration is limited to 30 seconds. Furthermore, the Google App Engine platform does not support the MapReduce programming model (1) or related methods for distributed processing and generating efficiently large data sets. Therefore, the Google App Engine platform is currently not suitable for performing large-scale and time-consuming geospatial processes.

Beside these and other commercial cloud providers, different projects and initiatives drive the general development of Cloud Computing technologies and especially the development of open standards for interoperability in clouds. The Open Cloud Consortium (OCC) [33] for example is an initiative dedicated to Cloud interoperability and initiated the Open Cloud testbed [34]. The Open Cirrus Project [35] is cloud computing research testbed between research and industry partners. In the Eucalyptus [36] initiative, an open source based implementation of the Amazon API is under development.

Web Processing Service

The Open Geospatial Consortium (OGC) Web Processing Service interface specification (11) describes a standardized method to publish and execute web-based processes for any type of geoprocesses. According to the WPS interface specification, a process is defined as any calculation operating on spatially referenced data. In detail, the WPS interface specification describes three operations, which are all handled in a stateless manner: GetCapabilities, DescribeProcess and Execute. GetCapabilities is common to any type of OGC Web Service and returns service metadata. In case of WPS it also returns a brief description of the processes offered by the specific service instance. To get more information about the hosted processes, the WPS provides process metadata through the DescribeProcess operation. This operation describes all parameters, which are required to run the process. Based on this information the client can perform the Execute operation upon the designated process. As every OGC Web Service, the WPS communicates through HTTP-GET and HTTP-POST based on an OGC-specific XML-message encoding. Besides this basic communication pattern, the WPS interface provides functionality for scalable processing such as asynchronous processing (implemented using the pull model), storing of process results and processing of data references encoded as URLs. The application of URL references as input for specific processes is a promising feature, as it limits the volume of data sent

[28] http://www.microsoft.com/azure/default.mspx
[29] http://www.microsoft.com/azure/windowsazurefordevelopers/default.aspx
[30] http://www.ibm.com/ibm/cloud/
[31] http://www.sun.com/solutions/cloudcomputing/index.jsp
[32] http://hadoop.apache.org/core/
[33] http://www.opencloudconsortium.org/
[34] http://www.opencloudconsortium.org/
[35] https://opencirrus.org/
[36] http://www.eucalyptus.com/

between client and service and allows the service to apply specific caching strategies. The service retrieves the data once and reuses it multiple times, by using the reference as an identifier for data.

Concept

On a technical level, the classical 52°North WPS is implemented as a Java Servlet. Due to platform independence gained by Java programming language and the Google App Engine Java support, the WPS components could be easily compiled with a standard Java compiler on a local machine and the resulting package could be deployed on the Google App Engine platform which runs with its own java virtual machine.

As our first test case, we implemented a simple buffer process, which takes two inputs. First, geographic features to be buffered encoded as GML (for example provided by an OGC Web Feature Service, WFS) and second, a distance for the buffer calculation. As a result, geographic features representing the buffers around the input geographic features are computed. The resulting dataset could be fetched either encoded as GML (as exercised with uDig [37]) or KML (as exercised with Google Earth [38]). According to the number of requests, the deployed application is able to scale up by means of the Google cloud mechanisms. Furthermore, native Google cloud services such as authentication could be used directly in the cloud from the deployed application.

In general, the deployed WPS provides geoprocesses to customer, which is the classical SaaS aspect. This is built on the PaaS aspect, which fosters the automatic scalability.

By deploying the WPS in the Google Cloud, the enduser still is able to find and bind a single URL representing the WPS, even though multiple instance exist on the sever side to maintain a scalable service. Therefore the classical publish-find-bind SDI paradigm (9) is not modified by using cloud technologies. However, the use of standardized interfaces such as a WPS ensures interoperability from the client perspective, cloud interoperability from a provider perspective is not given, since every cloud infrastructures has its own APIs and requirements.

Scalability Evaluation

Scalability is one of the key aspects of Cloud Computing. Therefore, we tested our approach and the Google cloud in this direction. We used a stress test to simulate a high demand of simultaneous requests and expected a constant response time by the WPS deployed in the cloud in contrast to a linear rising response time by a non-cloud setting.

Methodology

The WPS was stress tested with the simple buffer algorithm, deployed on the Google App Engine as well as on a local and non cloud enabled Tomcat installation. The geometric data for that process were also delivered via a web service (deployed at the Google App Engine platform in the first case and deployed on the local and non cloud enabled machine in the second case). A cumulative approach was used, starting with 1 and up to 200 requests that were sent simultaneously to the deployed services. The elapsed time from sending the request to receiving the response on its own, as well as for the cumulative sum of the requests/response times was measured. In order to compare the local setting with the remote cloud setting, the results are normalized by only regarding the response time relatively to the maximum/minimum interval of all requests to the specific machine.

Results

Figure 1 shows the normalized response time of the online as well as of the local deployed WPS over the number of simultaneously sent requests. The response time of the remote WPS (blue) stays nearly constant up to 200 simultaneous requests whereas the local WPS response time (red) grows linearly.

Evaluation

The performance evaluation shows to some degree that Google App Engine's scale at high request rates, as the response time for many simultaneous requests stays nearly constant in contrast to the non-cloud deployment.

The slight increase of the response time of the WPS deployed at the Google App Engine platform could be explained by some bottlenecks concerning the data allocation from an external server, laborious internal processing steps in the performance testing tool and high traffic at the local machine and in the local subnetwork when running the performance testing tool. A slight overhead for replicating new service instances on the server side could also be assumed.

Conclusion and Outlook

This paper presented and tested an approach of bringing the OGC Web Processing Service to the cloud. On a conceptual level, we showed that Cloud Computing is not a completely new concept and applied to SDI, the classical publish-find-bind pattern does not have to be modified. Therefore, we see a paradigm shift

[37] http://udig.refractions.net/
[38] http://earth.google.de/

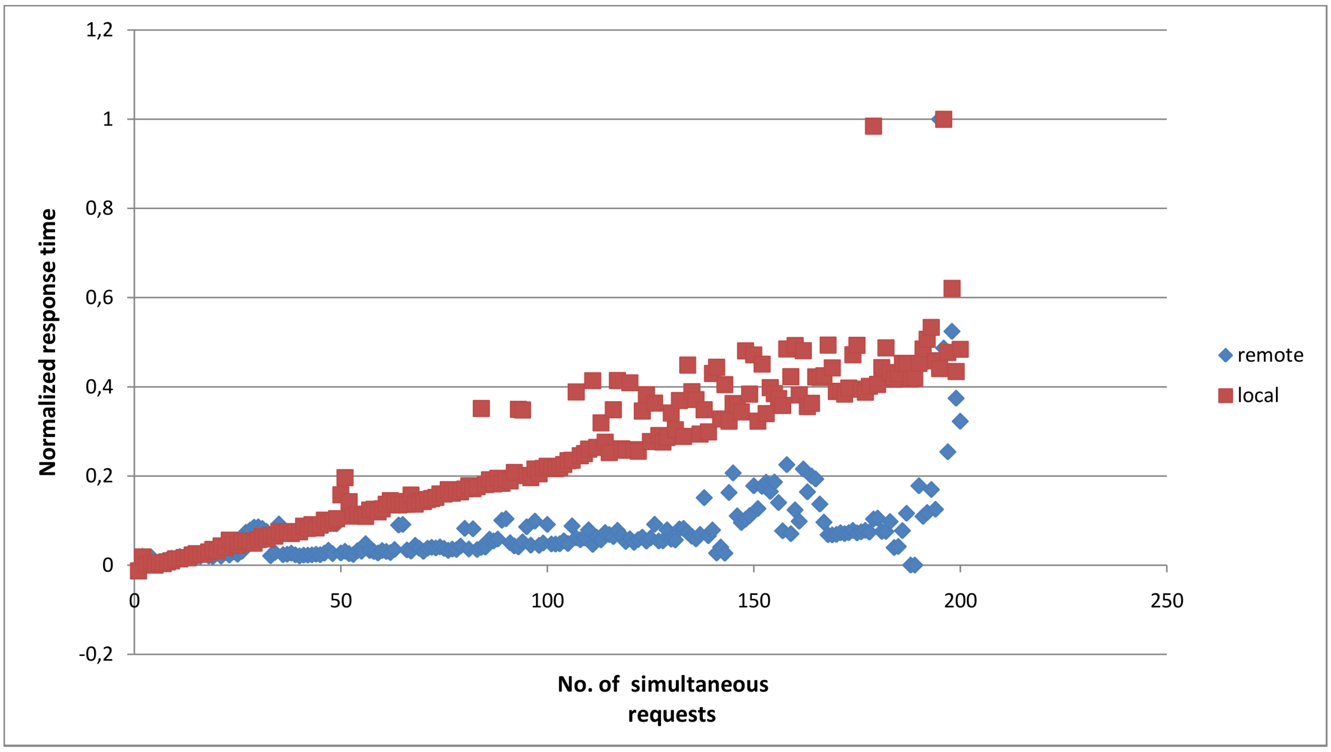


Figure 1: Comparison of normalized response time of remote (blue) and local (red) deployed WPS over number of simultaneously requests.

from technological to economical aspects in contrast to a complete paradigm change. On a technical level, our tests showed that by using the Google Cloud, response times could be held almost constant in contrast to a non-cloud approach. However, our tests also showed, that for the cloud approach, bottlenecks outside the cloud have to be taken into account and could eliminate the positive cloud effects if not carefully evaluated. Nevertheless, the tests showed that Cloud Computing keeps its promises and should be regarded further in sophisticated setups.

Thus, we plan in the next evolution phase to extend the described scenario, which mainly incorporates SaaS aspects, towards a more complex scenario, which takes near real-time air quality sensor data, stored already in the cloud (IaaS or dSaaS) and provided through standardized OGC interfaces (for example OGC Sensor Observation Service, SOS), and interpolate these data in the cloud (for example via WPS). Thereby, we aim at keeping the response time constant using efficient (despite possible high request rates). Besides, another goal for the next iteration phase will be the integration of existing Google App Engine services (for example Mail for alerting and Google Accounts for authentication) and efficient methods for distributed processing as well as storing large dataset (for example MapReduce and the Hadoop platform) into the framework.

Nevertheless, the presented approach is to our knowledge the first OGC compliant cloud service ever and could pave the way for a paradigm shift in SDIs. On the basis of our past experience we still believe that Cloud Computing is promising for building up, operating and utilizing SDI in an effortless way and promising for geospatial applications to enable new business models with less up-front investments. Furthermore, Cloud Computing could be potentially the missing element to popularize SDIs to a broader non-expert community (for example in an effortless way by means of Web 2.0 applications, such as mashups, open collaboration, social networking and mobile e-commerce). In particular, we could think of using OGC interfaces as the standardized way for obtaining geospatial resources (data/processing) similar to added-value services already provided in clouds such as Google Mail. However, by using OGC interfaces, cloud interoperability even from a provider perspective in regard to geospatial resources could be gained.

To further advance the adoption and combination of Cloud Computing and SDI, the 52°North Geoprocessing Community members will continue their basic research by addressing the following questions and topics:

- How can Cloud Computing lower the barriers for building, operating and utilizing SDIs?
- How can Cloud Computing promote innovative and promising geospatial e-commerce models?
- How can Cloud Computing popularize geospatial applications to a broader and collaborating commu-

nity?

- How can SDI elements be mapped to the Cloud Computing paradigm?
- Development and implementation of a fully Cloud Computing enabled SDIs by extending our approach with other Cloud Computing aspects.
- Security aspects such as Authentication, Authorization, Accounting and Delegation.

Bastian Baranski
Research Associate, Institute for Geoinformatics at University of Muenster, Germany
baranski AT uni-muenster.de
http://ifgi.uni-muenster.de/

Bastian Schaeffer
Research Associate, Institute for Geoinformatics at University of Muenster, Germany
schaeffer AT uni-muenster.de
http://ifgi.uni-muenster.de/

Richard Redweik
Bachelor Student, Institute for Geoinformatics at University of Muenster, Germany
richard.redweik AT uni-muenster.de
http://ifgi.uni-muenster.de/

Bibliography

[1] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters., 2008.
[2] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared, 2008.
[3] A. Friis-Christensen, N. Ostlnder, M. Lutz, and L. Bernard. *Transactions in GIS*, chapter Designing service architectures for distributed geoprocessing: Challenges and future directions., pages 799–818. 2007.
[4] Gartner. Gartner says cloud computing will be as influential as e-business. Gartner press release, Gartner, 2008.
[5] Gartner. Gartner says cloud application infrastructure technologies need seven years to mature. Gartner press release, Gartner, 2009.
[6] K. Hartig. What is cloud computing? the cloud is a virtualization of resources that maintains and manages itself. *.NET Developers Journal*, 2008.
[7] Sun Microsystems Inc. Cloud computing at a higher level. Technical report, Sun Microsystems Inc., 2009.
[8] C. Kiehle, K. Greve, and C. Heier. *Transactions in GIS*, chapter Requirements for Next Generation Spatial Data Infrastructures-Standardized Web Based Geoprocessing and Web Service Orchestration, pages 819–834. 2007.
[9] I. Masser. *GIS worlds: Creating spatial data infrastructures*. ESRI Press, Redlands, California, USA, 2005.
[10] J. Myerson. Cloud computing versus grid computing - service types, similarities and differences, and things to consider. Technical report, IBM Corporation, 2008.
[11] OGC. Opengis web processing service, version 1.0.0. Technical report, Open Geospatial Consortium (OGC), 2007.
[12] B. Schaeffer, B. Baranski, T. Foerster, and J. Brauner. A service-oriented framework for realtime and distributed geoprocessing. In *International Opensource Geospatial Research Symposium*, 2009.
[13] A. Weiser and A. Zipf. *Lecture Notes in Geoinformation and Cartography*, chapter Web Service Orchestration (WSO) of OGC Web Services (OWS) for Disaster Management, pages 239–254. Springer, New York, 2007.

Media Mapping

Using Georeferenced Images and Audio to provide supporting information for the Analysis of Environmental SensorDatasets

Phil Bartie, Simon Kingham

Abstract

Field based environmental monitoring projects often fail to gather supporting temporal information on the surroundings, yet these external factors may play a significant part in understanding variations in the collected datasets. For example when sampling air quality the values may change as a result of a bus passing the sampling point, yet this temporal local information is difficult to capture at a consistently high resolution over extended time periods. Here we develop an application which runs on a mobile phone able to capture visual and audio data with corresponding time and location details. We also develop a desktop analysis tool which synchronises the display of this dataset with those captured from environmental sensors. The result is a tool able to assist researchers in understanding local changes in environmental datasets as a result of changes in the nearby surrounding environment.

Introduction

The analysis of temporal datasets in Geographic Information Systems (GIS) is often hampered by a lack of supporting relevant information on local conditions at the time of data capture. Being able to explain unpredictable variations in temporal datasets may depend on being able to understand the nature of the local environment at a very local scale. For example a passing vehicle may be the cause of a noted spike in airborne particulate matter, but unless this situational information is recorded the spike may never be explicitly explained. Sensor networks are able to supply background information revealing the wider situation, but a co-located synchronized set of sensors are required to understand the local situation during mobile data capture. Here we develop an application able to assist researchers in storing information on the local environment at the time of data capture.

The solution uses a mobile phone to store audio, visual, and location details against time such that during analysis the researchers are able to view the local environment at the time of data capture. A custom playback and analysis tool was also developed to combine this situational data with other time stamped data captured from the same location, allowing researchers fast access to the relevant contextual information during the analysis of the environmental data. In this paper we describe the function of the mobile and desktop applications, their design, and report on their usefulness in an air pollution monitoring study conducted in an urban area. The tools proved useful in explaining local spikes in air pollution data due to local events documented in the supporting images and audio data. The audio stream was also useful for allowing the researcher to take spatially and temporally attributed verbal notes in the field.

Background

Many studies have examined the relationship between air pollution and modes of urban commuting (Fruin et al. 2008, Briggs et al. 2008, O'Donoghue et al. 2007, Gulliver and Briggs 2007, Van Roosbroeck et al. 2006, Kingham et al. 1998). Air pollution data is highly temporal, changing across time and space, affected by global and local events. Local events, such as a bus passing the recording equipment, are hard to document and traditionally paper based records are kept. However manual references to such events are hard to integrate into any analysis, and sampling frequency is often inconsistent. Yet these factors are important as has been strongly argued by Briggs et al (2008) who state that local factors could be the cause of differences reported between studies including such things as "building configuration, road layout, monitoring methods, averaging periods, season, meteorological conditions, vehicle, driving and walking behaviours, and the strength of in-vehicle sources" (Briggs et al. 2008, 20).

To provide a better understanding of the surrounding environment at the time of data capture a second set of sensors can be used to automatically capture contextual information. This contextual information is not the primary data for research purposes, but a supporting dataset for the analysis phase. Technological advancements have made it possible to sense the environment more accurately, at higher sampling densities than ever before. Sensor networks may consist of electronic devices (Culler and Mulder 2004, Microsoft Corporation 2006), or citizens volunteering local environmental information via the internet (Goodchild 2007). En mass citizens may provide data without realizing it, such as the monitoring of mobile phones to estimate population movement (Ratti et al. 2006), or to estimate travel delays on motorway sections (Astarita et al. 2006).

The majority of electronic wireless sensor networks are static, distributed across a region at fixed sites, feeding information to a central facility which combines the data to build a picture of the surrounding conditions. For the purpose of mapping a commuter's exposure to air pollution the local level changes are also important, therefore a set of sensors should remain co-located with the air pollution sampling equipment. The data samples must also be at a high enough temporal resolution to record significant local events, and the data streams need to remain synchronized during data capture.

Mobile computing devices and smartphones have been proven useful in environmental monitoring enabling participants to collect and share data in real-time (Rudman et al. 2005). The MESSAGE consortium (Polak and Hoose 2008) have undertaken a number of projects using mobile phones as personal environmental sensors and data loggers. The mobile phones were equipped with a payload of environmental sensors able to record carbon monoxide, carbon dioxide, traffic volume, and nitrogen dioxide levels. Researchers exploring the city could feed data in real time to a data centre for processing, revealing current city wide air pollution trends. The individual trip data could also be replayed and mapped so that air pollution trails could be reviewed to visualize areas of poor air quality in the city. However a key aspect missing from this research was the facility to store video or image data from the user's surroundings. Therefore any analysis carried out at a later date would lack documented contextual detail. Although audio was used, it only provided an estimate of traffic volume and could not differentiate between vehicle types, or allow field researchers to take temporally and spatially attributed verbal notes.

Another research group developed a prototype system known as GeoMobSense (Kanjo et al. 2007). This toolkit allows private users to equip their own phones with the necessary facilities to log data from connected sensors. The phones themselves are used to display information, as well as log sound levels, while separate data loggers are used to record the environmental data. The resulting datasets can be exported and displayed on Google Earth, and other GIS applications. Again this toolkit fails to store continuous image sequences, or to save a spatially attributed audio file. Therefore any post-capture analysis is hampered by a lack of documentation on the surrounding situation during the field study.

Multimedia files provide a useful companion dataset for prompt recall of events which occurred during data capture. They provide an extra channel of information useful when linked to GISs (Cartwright et al. 2007), however the video, image, or sound files are normally linked to a point, as in Media Mapper (Red Hen Systems 2009). This creates a one way relationship, only allowing the corresponding multimedia clip to be found when the user clicks on a map location, essentially using the map as document retrieval interface. The content of the multimedia file itself is not spatially attributed, and the ability to jump to the corresponding position in the video file for a given map location has to be performed manually.

There have been a number of attempts to more closely link the multimedia content to space through dynamically geo-referencing multimedia files. Spatial information is encoded into the file through an appropriate technique such that at any point in the video or audio the corresponding location may be referenced. For example specialist equipment can turn GPS location information into audio data, in a similar way that a modem is able to turn computer data into audio to send it across a telephone line, which can then be recorded to the audio track alongside the video data. An example of this technology is CamNav Mapper (Blueglen Ltd 2009). This allows a user to search through a video file, and at all times be able to display the corresponding recording location in a GIS. However the connection is uni-directional, meaning the video is able to provide location information to the GIS, but it is not possible to initiate a search for the corresponding part in the video from the GIS.

Jaejun (2002) developed an application which supports bi-directional searching, permitting the user to find relevant video information from selecting a GIS location, or for finding the filming location by searching the video file. Similarly Zeiner (2005) developed an application able to fuse GPS location and video using timestamp information collected from synchronized clocks. For still images a set of points are created in the GIS, however for video recorded while moving a track denotes which parts of the video correspond to which geographic location. They also explore the use of data standards in providing geo-multimedia tools via the World Wide Web, with particular focus on the overlap between web mapping standards, metadata standards, and video streaming standards. These tools are not however designed with the ability to integrate other temporal datasets such as required for environmental research.

Other studies have used or developed analysis tools to visualize environmental datasets with local situational data (Kaur et al. 2006, Terwoert 2009, Arnold et al. 2004). However while these often include the ability to link photographic images with environmental data, they appear to lack a tightly integrated mapping facility.

For our research the requirement was for an application which could provide a high level of integration between temporal multimedia and location datasets, with the ability to support additional datasets collected from synchronized sensors. The capture device needed to be small, lightweight, and mobile such that it could be carried by a pedestrian or cyclist easily for extended periods of time. The datasets for location, audio and image needed to be tightly coupled such that no synchronisation issues could occur during long field trials. The analysis tools needed to be easily operated by an untrained GIS user, such that they could search through the datasets to interact with any of the captured data streams while maintaining sync with the other linked data sources. We therefore looked at developing the data logging tool on a GPS equipped smartphone, which is a highly portable programmable device available at low cost. As a result of using a single programmable device the GPS, audio, and image datasets are tightly coupled, removing the need to synchronize clocks, and guaranteeing data streams remain in sync indefinitely. Additionally the multimedia files are georeferenced at the time of data capture removing the need for any post-capture data processing. In contrast to other applications which use laptops to capture and process media, the smartphone approach offers a robust, small, and very portable platform which may be easily carried by pedestrians. Finally our analysis tool supports a tri-directional search mechanism, such that users may drive the search by moving through the audio media, mapping interface, or by interrogating charts of the additional sensor data. This means the user is able to easily capture and analyze data using any of three mediums (location, time, graph value). In the next section we look in more detail at the applications developed during this research.

Application Development

In this study we developed two applications to assist in the process of recording the surrounding situational conditions. The first application runs at the time of data capture in the field on a smartphone equipped with Assisted GPS (A-GPS), and stores both an audio and visual record of the surroundings. A-GPS is particularly useful for urban based research as it provides a faster start-up location solution throughout a greater range of urban environments, such that a position could be found more quickly and maintained more consistently. Furthermore the phone selected for this research had a high sensitivity GPS chipset, enabling locations to be calculated across a high proportion of the city, including inside some single storey buildings. The second application developed for this research runs on a desktop computer and assembles independent data streams against a common timeline, such that the user may easily browse through multiple datasets whilst maintaining sync between them, at all times being able to refer to the corresponding situational image and audio data. We discuss each application in more detail in Sections 3.1 (Mobile Data Capture Application) and 3.2 (Data Analysis and Playback), starting with the mobile data capture tools.

Mobile Data Capture Application

The main design criterion for the data capture device was that it would be used in urban studies everyday over an extended period of a few months. It therefore had to be small, robust, light weight, offer a large data storage capacity, be able to capture audio, imagery, and run on battery for at least 90 minutes to ensure an entire urban commute could be captured in a single session. We decided to use a Nokia N82 smartphone to carry out these tasks as they can be programmed easily using the Python language, incorporate a high sensitivity A-GPS able to function adequately in urban canyons, and have a high quality camera. Furthermore they are able to use micro-SD cards for data storage, are smaller than any laptop or netbook computer, and have good battery life.

Nokia Series 60 smartphones can be programmed in three main languages, which are C, Java, and Python. Python is very suitable for rapid development and allows the developer to access core phone hardware through supported Application Program Interfaces (APIs). The hardware access required for this project included GPS hardware, audio, and screen display. Our initial application was designed to record a continuous video and audio feed to the micro-SD card at 15 frames per second, while logging GPS locations every second. The phone supports the ability for Python applications to request the position of the current playhead in the video file during recording, enabling the application to log the GPS position information along with the current video position to ensure a tight coupling of the location and video datastreams. The data capture application performed well, and as it used MP4 compression a full one hour video with GPS log files occupied only around 70MB. However the continuous video capture depleted a fully charged battery in 60 minutes, and made the mobile phone run fairly hot. After discussions with the air pollution research team we looked at an alternative solution to record still images at regular intervals.

The next iteration of our application, and the one used in field trials, records an audio file continuously but captures still images at the rate of 1 image every 3 seconds. The audio file is recorded to a WAV file at 8kHz, again the playhead position is stored with each GPS update such that for every image the location and position in the audio file is known explicitly.

To link the audio and image files with GPS a common timeline primary key is required. GPS time was considered unsuitable for the base timeline as the user may lose the GPS signals when moving inside buildings. Therefore the Python time from the phone clock was considered more reliable, and forms the baseline to which all other datasets are synced. GPS time is however stored as an additional attribute in the log file in case it is required later.

Specialist sensors were used to sample the air quality, recording the concentration of particulate matter at various sizes (PM10, PM2.5 and PM1), ultrafine particles (UFP), temperature and carbon monoxide levels. These devices all have internal clocks which were synced to the nearest second to the clock on the mobile phone before each journey. During transit the mobile phone was mounted facing forwards on either the car dashboard, bike handlebars, or the strap of a rucksack

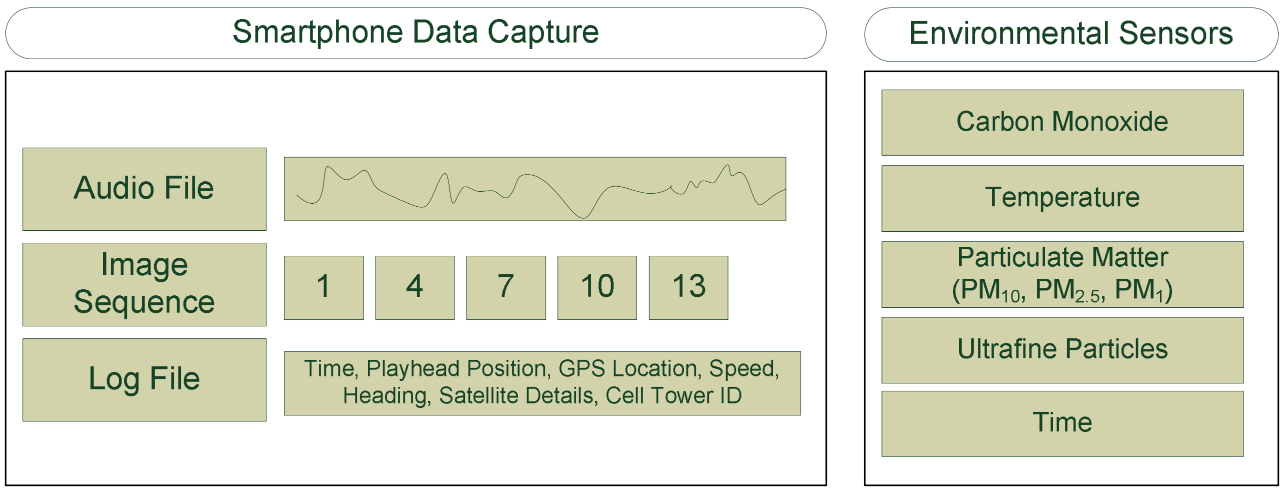

Smartphone Data Capture
Audio File
Image Sequence
1
4
7
10
13
Log File
Time, Playhead Position, GPS Location, Speed, Heading, Satellite Details, Cell Tower ID
Environmental Sensors
Carbon Monoxide
Temperature
Particulate Matter (PM_{10}, $PM_{2.5}$, PM_1)
Ultrafine Particles
Time

Figure 1: An Overview of the Data Captured

to ensure the camera could capture a clear view of the oncoming route.

Figure 1 shows the data collected with each journey. The smartphone tags each image with a unique identification number based on the Python time in seconds, ensuring that images correspond to a single log entry. Log entries record the position of the playhead in the sound file, Python time, GPS time, and the GPS latitude and longitude. Additionally we stored the cell tower identification value so that approximate locations can be determined if GPS positioning is lost. As well as recording GPS position we also record speed, GPS accuracy, heading, the number of satellites visible, and number used for the position solution. This enables us to carry out analysis on the location accuracy if required at a later date.

Data Analysis and Playback

To be able to efficiently analyse the large volume of time series data collected it was necessary to build a custom application which allowed non-GIS users the ability to review the data easily from a single interface. To do this we wrote a custom application using C# .NET which made use of a number of open source libraries for charting, map display, and coordinate projection (Figure 2). One of the key criteria in our application design was that all the datasets should be synced and remain in sync across all dataset viewers while the user explored the data. For example selecting a map point should display the relevant air pollution data, the corresponding street image, and move the sound file playback to the correct location so any relevant audio notes and background noises could be heard. In the following section we discuss each of these data visualisation elements, and describe the methods through which the datasets are kept in sync.

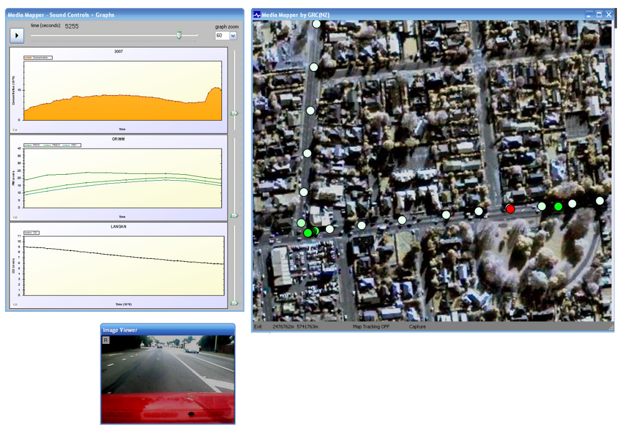


Figure 2: Data Analysis and Playback Tool

Mapping

The mapping element makes use of the Piccolo Zoomable User Interface (ZUI) graphics framework. This open source library is not specifically designed for map display but offers powerful functions enabling rapid development of ZUIs. Piccolo is able to create scene graphs consisting of both vector and raster nodes, which may be easily animated to change location, shape or colour. It also provides the functionality to smoothly zoom and pan around the data, capturing mouse and keyboard input. Events can be assigned to graphical objects such that a user may interact with map items and trigger a custom action. In our application we use this functionality to link map objects to custom search functions, so for example clicking on a GPS track point moves the charting tool along to display the air quality values at that location, and moves the audio playhead forwards or backwards to the corresponding audio sample.

The mobile phone logs GPS data using the WGS84 coordinate system, however as Piccolo is unaware of geographic coordinate systems it requires projected datasets. We used New Zealand Map Grid (NZMG) as our projected coordinate system for all datasets. The background mapping layers included Quickbird satellite imagery, and Land Information New Zealand road centrelines transformed using GRASS to NZMG. The vector datasets, typically ESRI Shapefiles, were converted to a BNA text format using OGR libraries before being loaded and displayed by Piccolo, while the raster datasets (JPEG) were natively supported. ESRI World files were required with each raster image to provide pixel resolution and location coordinates values, so that raster datasets could be loaded into the correct position.

The GPS data, once downloaded from the mobile application to the desktop application, is transformed into the NZMG coordinate system using the most accurate NTv2 projection transformations provided by the Proj4 libraries. These points were then displayed as Piccolo vector nodes over the base mapping, each node with a hidden tag holding the corresponding capture time primary key. This tag corresponds to the phone capture time in seconds, and is also recorded in the log file against the playhead position and location details. These tag data enable the software to instantly locate corresponding relevant information in other datasets when a user clicks on a map object ensuring the system is very responsive.

Audio

The audio playback is handled using Microsoft DirectSound libraries, which enable the rapid development of software able to control audio datasets. Here we use basic playback control features (i.e. play, pause, forward, rewind), and the functionality to read the current playhead position during playback. From this we can calculate the current play time in seconds from the start of the sampling period, and therefore find the corresponding log entries which hold the geographic location and relevant image filenames. As the audio file is captured at 8000 samples per second it has the highest resolution of any of the datasets. Therefore any searches performed from the user cueing or reviewing this dataset require an additional step to find the most relevant (i.e. closest) timestamp in the log files. This was performed by simply ranking the difference in time from the audio position to all log entries, the first item being selected as the nearest. This functionality allows the user to review the audio file and also see corresponding imagery, location and chart data.

Due to memory issues when loading a 90 minute long audio file, a buffered playback method was required. Only a small section of the file is loaded into a memory buffer, and this buffer is constantly filled from the disk file as audio playback progresses. The performance impacts of this technique were minimal, and rapid audio reviewing and cueing are still possible.

Street Images

The smartphone captures an image every 3 seconds in JPEG format, and labels it with the appropriate timestamp. The log file holds a list of these timestamp filenames along with their corresponding location and other GPS details. When the user selects a location on the map the tag (with each Piccolo node object) holds the timestamp details, and therefore the corresponding image can instantly be loaded without performing any search other than for the filename in the file system. When the user controls the audio playback the nearest timestamp information is found in the log file, and from this the corresponding picture name can be generated. Similarly when the user moves through the chart information the nearest timestamp in the log file is found and used to determine the appropriate image to display.

The image display is supported using native .NET libraries, with functionality to rotate the image sequence which is useful if the phone has been placed on its side during image capture.

Charting

The environmental sensor datasets are charted using the open source ZedGraph libraries. These provide sophisticated charting capabilities, and allow the programmer to link into many key events such as when the user pans across the chart, clicks in the charting area, or changes the scale on a chart axis. In our case the x-axis was allocated to time in seconds since the start of sampling. Ideally the smartphone would be turned on first to ensure the audio file timeline starts before the environmental data, but negative time values are also supported. The y-axis displays the values from the relevant sensor. The display automatically scales the y-axis according to the values in the entire loaded dataset, although manual scaling is also possible to zoom into values for smaller sample periods.

The ZedGraph library was used to create line graphs for each of the environmental sensor types. These can remain static to allow the researcher to view the entire datasets while replaying map, image and audio datasets. Alternatively the graphs can be dynamically linked to the playback such that they pan along the x-axis (time) automatically as the data log is replayed. In this case the current playback time is shown on the far left of the chart.

If the user clicks within the chart area the application retrieves the corresponding time (x-axis value) and moves the playback position to that value. This allows the researcher to instantly find the current geographical location for any spikes noted in the environmental datasets. Also as the audio and image files remain in sync the researcher is also able to look and listen to information from the surroundings at that point.

For reporting purposes the system supports high quality output of the graphs, by simply double clicking on them. There is also functionality to allow the user to export snapshots of any interesting results, effectively using this tool to produce a filtered dataset. To do this they simply click a button during playback and the GPS, picture link reference, time and date, and graph data are exported to a text file. As each environmental sensor operates at a different sampling frequency the application interpolates values between readings (i.e. straight line between known values). When the data is output the interpolated value is used if an actual reading value is not present for the current playback position.

User Interaction

One of the key differences in the analysis application developed for this research to those reviewed earlier, is the ability to initiate a search from any of three linked interfaces. Figure 3 summarises the processes required to provide this functionality.

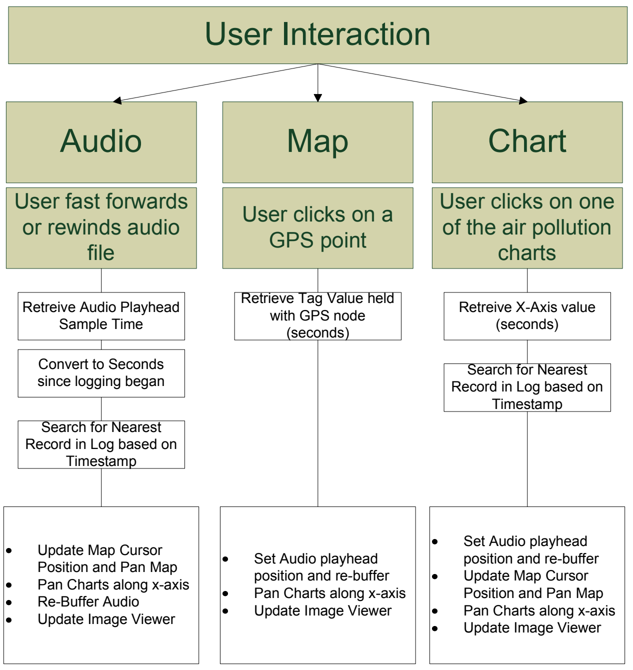

User Interaction
Audio
Map
Chart
User fast forwards or rewinds audio file
User clicks on a GPS point
User clicks on one of the air pollution charts
Retreive Audio Playhead Sample Time
Convert to Seconds since logging began
Search for Nearest Record in Log based on Timestamp
Retrieve Tag Value held with GPS node (seconds)
Retreive X-Axis value (seconds)
Search for Nearest Record in Log based on Timestamp
• Update Map Cursor Position and Pan Map
• Pan Charts along x-axis
• Re-Buffer Audio
• Update Image Viewer
• Set Audio playhead position and re-buffer
• Pan Charts along x-axis
• Update Image Viewer
• Set Audio playhead position and re-buffer
• Update Map Cursor Position and Pan Map
• Pan Charts along x-axis
• Update Image Viewer

Figure 3: User Interaction via Audio, Map, or Chart Interfaces

During normal playback the system carries out an audio update event every 4000 samples (500ms). This ensures that the displays of each dataset remain in sync, without impacting the performance on slower computers.

In the next section we look at a number of examples which demonstrate the usefulness of this application.

Examples

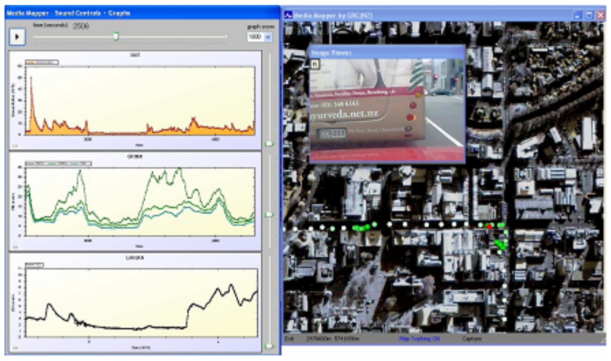


Figure 4: Analysis of Peak Resulting from Bus Pulling Away.

In the following section we look at data collected in air pollution studies from Christchurch, New Zealand. The figures illustrate how useful the contextual information was in explicitly explaining peaks in the air quality datasets. In Figure 4 we can see that the air quality spikes in the top left graph occur just after the bus (pictured) pulls away from the field observer. In addition the location information and map allow the researchers to easily and quickly identify where in the city these interesting results occurred.

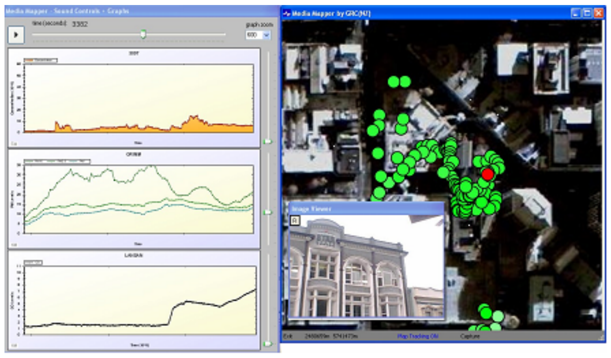


Figure 5: Analysis of Air Data while Walking in Pedestrian Precinct.

In the next example the air quality levels are much better while exploring a pedestrian precinct, with low particulate matter concentrations (Figure 5). However following this a sharp rise in particulate matter can be noted. By clicking on the spike in the graph display area the researcher is able to see, from the map location and supporting images, that this spike correlates to when the field observer entered a multi-storey car park.

In the final example data were collected while travelling on a bus around the city. A series of spikes can be noted in the carbon monoxide levels at fairly regularly spaced intervals, as shown in Figure 6. By reviewing the audio data, just before each spike, it was possible to hear the sounds of the door opening, ticket machine being operated and so on, and therefore conclude that these spikes may be related to door opening events. It is also worth noting that a camera facing forwards would not have picked up this information, thus demonstrating the value of collecting audio data.

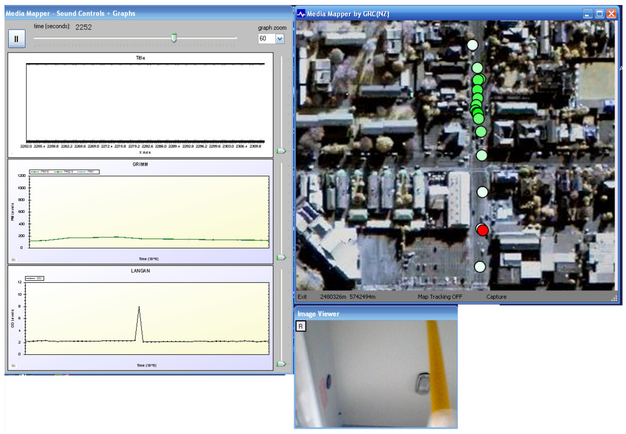


Figure 6: Bus Trials of Air Pollution Monitoring Equipment

Conclusion

In this research we have demonstrated that a smartphone may be used as a suitable data capture tool, able to accurately and consistently capture location attributed audio and visual data while operating in an urban space. The onboard Assisted-GPS (A-GPS) was able to rapidly locate the user when the device was first turned on, and maintain position throughout exploration of outdoor urban areas. The audio and image quality was suitable for later analysis to identify key events which may be associated with changes in the local environment as a result of short temporal events (such as a bus pulling out) or changes in the nature of the physical environment (such as entering a car park). Python provided an excellent programming language for rapid application development on mobile phone, with the necessary functionality to use the phone's hardware such that explicit geo-referenced image and audio could be processed on board the phone, rather than during later post-processing in the desktop environment. This ensures that data sync between audio, location, and imagery datasets may be maintained indefinitely. The smartphone platform also proved to be rugged enough for daily trials over a period of several months, and had battery and storage facility to cater for long urban commutes (tested up to 120 minutes).

We also demonstrated that a simple desktop application able to maintain sync between mapping, environmental, audio, and visual datasets proved useful in the analysis phase of the research. The tool allowed non-GIS researchers the functionality to easily explore environmental datasets while maintaining links to the corresponding contextual information.

The project was completed using a number of open source tools and libraries, without which the development would not have been possible in the allotted time or within budget. Future versions of the application should include the ability to store bookmarks against time, allowing the user to add keywords or notes, which would be particularly useful when revisiting previous datasets, or to store comments for other project collaborators to view. More powerful query tools would be useful too, such that map points may be highlighted or hidden based on the closest environmental data. Finally the ability to search audio files for speech would be useful in longer sampling runs such that the system could automatically identify where audio notes have been taken.

Acknowledgements

The authors would like to acknowledge the contributions made by Kreepa Shrestha, and Woodrow Pattinson who carried out the extensive field trials. Also Justin Harrison for setting up and supporting the environmental sensor equipment. This research would not have been possible without funding support from the Geospatial Research Centre (NZ) and New Zealand Transport Agency.

Phil Bartie
Phd Student
University of Canterbury, New Zealand
philbartie@gmail.com

Simon Kingham
Associate Professor of Geography
University of Canterbury, New Zealand
simon.kingham@canterbury.ac.nz

Bibliography

Arnold SJ, ApSimon H, Barlow J, Belcher S, Bell M, Boddy JW, Britter R, Cheng H, Clark R, Colvile RN (2004) Introduction to the DAPPLE Air Pollution Project. Science of the Total Environment 332: 139-153

Astarita V, Bertini RL, d[92?]Elia S, Guido G (2006) Motorway traffic parameter estimation from mobile phone counts. European Journal of Operational Research 175: 1435-1446

Blueglen Ltd (2009) CamNavMapper. Retrieved 20 May 2009 from http://www.blueglen.com/prod_camnav_single.htm

Briggs DJ, de Hoogh K, Morris C, Gulliver J (2008) Effects of travel mode on exposures to particulate air pollution. Environment International 34: 12-22

Cartwright W, Peterson MP, Gartner GF (2007) Multimedia cartography, Springer Verlag,

Culler DE, Mulder H (2004) Smart sensors to network the world. Scientific American 290: 84-91

Fruin S, Westerdahl D, Sax T, Sioutas C, Fine PM (2008) Measurements and predictors of on-road ultrafine particle concentrations and associated pollutants in Los Angeles. Atmospheric Environment 42: 207-219

Goodchild MF (2007) Citizens as voluntary sensors: Spatial data infrastructure in the world of Web 2.0. International Journal of Spatial Data Infrastructures Research 2: 24-32

Gulliver J, Briggs D (2007) Journey-time exposure to particulate air pollution. Atmospheric Environment 41: 7195-7207

Jaejun YOO, Joo T, Park JH, Lee J (2002) A video geographic information system for supporting bi-directional search for video data and geographic information. Proceedings of International Symposium 2002

Kanjo E, Benford S, Paxton M, Chamberlain A, Fraser DS, Woodgate D, Crellin D, Woolard A (2007) MobGeoSen: facilitating personal geosensor data collection and visualization using mobile phones Personal and Ubiquitous Computing

Kaur S, Clark RDR, Walsh PT, Arnold SJ, Colvile RN, Nieuwenhuijsen M (2006) Exposure visualisation of ultrafine particle counts in a transport microenvironment. Atmospheric Environment 40: 386-398

Kingham S, Meaton J, Sheard A, Lawrenson O (1998) Assessment of exposure to traffic-related fumes during the journey to work. Transportation Research Part D-Transport and Environment 3: 271-274

Microsoft Corporation (2006) Sensors and Devices - SenseCam. Retrieved 22 May 2006 from http://research.microsoft.com/sendev/project_sensecam.aspx

O'Donoghue RT, Gill LW, McKevitt RJ, Broderick B (2007) Exposure to hydro-carbon concentrations while commuting or exercising in Dublin. Environment International 33: 1-8

Polak J, Hoose N (2008) Mobile Environmental Sensing System Across Grid En-vironments.

Ratti C, Pulselli RM, Williams S, Frenchman D (2006) Mobile Landscapes: using location data from cell phones for urban analysis. Environment and Planning B: Planning and Design 33: 727[96?]748

Red Hen Systems (2009) MediaMapper. Retrieved 20 May 2009 from http://www.afds.net/mediamapper.html

Rudman P, North S, Chalmers M (2005) Mobile Pollution Mapping in the City. Proceedings UK-UbiNet Workshop on eScience and Ubicomp. Edinburgh,

Terwoert J (2009) EU-project VECTOR: Visualising cyclists[92?] exposure to fine particles. Velo-City 2009. Brussels,

Van Roosbroeck S, Wichmann J, Janssen NAH, Hoek G, van Wijnen JH, Lebret E, Brunekreef B (2006) Long-term personal exposure to traffic-related air pollution among school children, a validation study. Science of the Total Environment 368: 565-573

Zeiner H, Kienast G, Derler C, Haas W (2005) Video documentation of urban areas. Computers, Environment and Urban Systems 29: 653-668

MapWindow 6.0

An Extensible Architecture for Cartographic Symbology

Harold A. Dunsford Jr., Daniel P. Ames

Abstract

A robust, extensible architecture is critical to open source projects that have a distributed developer and user base. The MapWindow 6.0 project is using a new architectural paradigm where extensibility is handled from several different plug-in points, rather than a single, application wide design. This allows new kinds of extensibility to be explored such as tools and data providers in addition to the more conventional application wide extensibility. This presentation outlines some of the improvements in the built in cartography, but primarily addresses the .Net architectural decisions that permit run-time discovery of new kinds of custom symbology. Improvements include layering of different kinds of symbols to make a compound symbol as well as establishing cartographic sub-categories based on vector attributes or raster values. The open ended framework allows for an extremely flexible system of run-time discovery so that the core libraries do not have to be recompiled each time an external cartographic improvement is developed.

Introduction

The problem addressed in this paper is the inability for an open source core application to anticipate all of the symbolic requirements for new sorts of data. Changes in the design model for the 6.0 version of the open source MapWindow GIS project allow for new kinds of plug-ins. One new kind of plug-in actually allows for external libraries to control the business logic of data management for a specific data format. The software that controls the business logic that runs the interface based run-time recognition of these new data providers has also been encapsulated in the form of a non-graphical component that can be easily added to a new project as easily as dragging the Map control or the legend onto that project. The inevitable consequence for this is that eventually there maybe new styles of data which need to be symbolized in an unconventional way. This paper seeks to address the problem of how we can design an architecture that is at once extremely flexible and versatile, supplying a built in symbol set that is as rich as professional software, but that is also malleable, so that future developers can easily extend the symbolic capabilities without having to recompile the architectural core.

The techniques outlined in this paper are important because architectures that support extensibility form a robust framework for successful open source GIS platforms to build on. While this quality is important for both proprietary and open source systems, it is essential when a spatially distributed developer base must coordinate their efforts. Multi-tiered, modular and transparent design standards allow for greater security and design control of the low-level, shared libraries, and also act as a contract to unify a wide range of extensibility and customization that is added on top of that core. This provides future and co-developers with a common platform that can be extended without fear of breaking other parts of the code – hence saving time and development costs.

We propose that it is non-obvious as to how to use a common interface or custom attribute to allow the core library and other developers to use extension classes effectively if the most critical content is not described by the interface itself. The conventional, run-time discovery of extension classes traditionally relies on using either System.Reflection or the Microsoft Add-In framework in order to specify light weight contract interfaces that then can be fleshed out with code that implements those interfaces. For something like a data provider, where the end result is an in-memory data format that always matches a given interface, this sort of traditional extensibility interface is ideal because you know what to expect from each additional data provider. Symbology, however, is something that should support a rich and versatile collection of properties that can't be predicted in advance. For instance, extending a point symbol to be represented by an image requires completely different attributes and properties than describing the point using a color, size and shape. What's wrong with previous proposed solutions?

Previous GIS architectures fail to address these versatile areas of extensibility. The conventional open source approach is that tasks like handling data formats and rendering those data formats are handled internally and modified or developed by only a select few that are privileged to be working on the core library. Even the most cutting edge advancements in Microsoft's Add-In Framework simply allow contract interfaces to be updated to newer versions, and don't address a way to easily extend the project with effectively unbounded components.

The first key component of our approach is the design of the base interface. This includes an ISymbol, for points, an IStroke for lines and an IPattern for polygons. These basic elements have minimal common attributes, but they all have methods that allow for those symbols to render themselves, given the set of coordinates, or a graphics path, to draw. The important thing from the standpoint of the rendering is that it won't matter what kind of properties control that rendering method, since all the symbols, for instance, evoke the same basic drawing method. These base interfaces also allow for run-time identification of classes that extend symbology.

The second topic of discussion by this paper is how to design a corresponding user interface that can allow for largely unpredictable symbol elements to be successfully modified by the user without knowing them in advance. The Dot Net Property Dialog component will be discussed as an option for a default user interface if one is not specified, but more importantly we will illustrate how software can allow developers to provide a custom control, or tab-page where they handle their own symbology.

Finally, we will address using extension methods (first made available with the 3.0 version of the .Net Framework) to allow developers to create new methods that are programmatically discoverable to intellisense, but also allow easy configuration of their new symbols without requiring the core library to be recompiled.

Background

MapWindow

The MapWindow project was first established in 1998 at Utah Water Research Lab in Logan as an alternative to using MapObjects LT 1.0. (1) The proprietary controls provided by ESRI prevented users from modifying the underlying data, however, which was of limited use for research oriented applications. The project requirements included being able to dynamically alter the shapes of vector data, or access the data values for grids. They created the core MapWinGIS.ocx component, an ActiveX control that could provide the low level access to the data formats that developers could then rapidly turn into successful projects. This ultimately led to the development of the MapWindow application because many of the common features that were shared between projects ended up being replicated over and over again. The fully developed project today is called MapWindow 4.x.

The MapWindow 6.0 project is focused on developing modular components written in C#. (3) Since everything written for MapWindow 6.0 is in a managed, Dot-Net language, it will be far more portable in terms of using the project in web applications or across platforms using the open source Mono framework. The MapWindow 6.0 version of the project began in the summer of 2007 as an effort to develop a topology toolkit for MapWindow 4.x. In 2007, the team added the Net Topology Suite into the project. This required such a serious re-thinking of the underlying objects that it was beneficial to start development of a completely new version of MapWindow from scratch, which is now MapWindow version 6.0. (2)

Other GIS Extensibility Architectures

ArcGIS

The ESRI ArcGIS object model consists of a host of interconnected objects, each with a very precisely defined role. The paradigm is to provide programmatic, macro-style access to the underlying objects through interfaces that restrict the functionality. In the Visual Basic for Applications (VBA) Macro development environment that is associated with ArcGIS versions 8.0 and larger, directly accessing an object doesn't expose the majority of its properties or methods. An example is the MxDocument object. In order to access more, you must first dimension a new IMxDocument interface, and point it at the object. This allows tight control over what aspects of the software external developers can control by exposing only a limited subset of members. Access permissions could be adjusted so that the full object is only viewable to developers working within the ArcMap project itself. There is no model in place to allow other developers to provide the base application with support for new data formats, though they are beginning to rely on the open source community projects like GDAL for some of their data member support. They also do not provide direct access to outside developers to low level functionality like rendering.

Grass

The extensibility model for the GRASS project involves creating new libraries that can perform new, independent operations. As is evident from their book Open Source GIS A GRASS GIS Approach, GRASS 6 is written in the ANSI C programming language and hosts more than 350 modules for management, processing, analysis and visualization of GIS data. The strategy that they have adopted is to require that all data formats be converted to their standardized raster and vector formats before any other module can work with the data. This allows for analysis of data directly from a file that might be too large to store in memory, while reducing the complexity of the analysis methods to working with a single data format. They also recognize that not every user will be an expert coder. In order to support this intermediate level of programmer, the GRASS supports script programming. UNIX Shell, PERL and Python scripts are supported, allowing repetitious tasks to be handled through their scripting language. They are the most mature version of truly open-source GIS today, and can be thought of as a textbook example of how to run a successful, long-term open source venture. GRASS shows us that in order to be a good, open-source project; you must have a framework that allows for future development and expansion over a long time. Our extensibility model explores how to give the .Net libraries a common GIS framework to use in order to talk to each other through the use of common interfaces, rather than the use of a common data format, but ultimately the long term goal is the same.

Quantum GIS

Given that there are hundreds of open source projects that feature GIS today, some of which are featured at http://opensourcegis.org/, it is hard to choose one to best illustrate the extensibility models that are currently available from applications that fall in the middle spectrum. One of the more complete and well known systems is Quantum GIS. The approach from Quantum GIS has always been more like a standard windows-style programming environment, instead of the traditional, Linux-style command prompt that was the principal mechanism for working with GRASS until just recently.

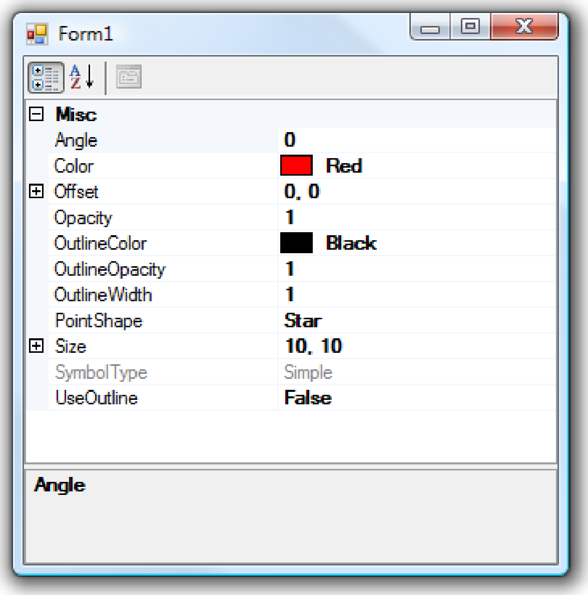
Quantum GIS supports a plug-in architecture that is more reminiscent of what you would likely find in a traditional software package. They even support a special type of Data Provider plug-in that allows developers to specifically extend the data formats that can be supported by the project. There are posts about Data Providers on their bulletin going back as far as 2006, so even though a big part of this presentation is about introducing plug-ins with distinct capabilities, this isn't an entirely unproven concept. The plug-in architecture for QGIS works by using python script or C++ files that satisfy certain criteria, i.e. hosting a particular script file with the name plugin.py and in essence writing script to match specific method names or schema. The weakness of this model is that the powerful development environment tools like intellisense are not generally available when working with a scripting language. With an interface, modern development environments are able to flesh out a skeleton with the correct signature that is type-checked at compile time. Further, while the special plug-ins exist to support data formats, those must first be accepted by the community and then manually added into the core library.

Programming Methods

Interfaces

An interface acts as a kind of skeleton framework for classes. In the C# language, dual-inheritance is not allowed, but it is possible to implement as many interfaces as you want. (5) Therefore, it has become fairly conventional to devise small interfaces that can be re-used in many different contexts. Many examples are built into the .Net Framework, such as the ICloneable interface which simply specifies that there will be a Clone() method that returns a new object. There are many classes that support this method, and those classes come from very diverse sets of class hierarchies. An interface can be thought of as a contract that designates what a particular class will do, regardless of the actual code that is used to fulfill the contract. As was illustrated in an earlier paper published in Position IT (2) the minor performance characteristics of virtual calls through an interface is insignificant compared to the performance penalty of using property accessors. There is a limitation in the sense that public variables called fields cannot be defined on an interface, and so using an interface forces the use of property accessors or methods, and using property accessors can slow down performance significantly in large loops.

Property-Grid



⊟ Misc	
Angle	0
Color	🟥 Red
⊞ Offset	0, 0
Opacity	1
OutlineColor	⬛ Black
OutlineOpacity	1
OutlineWidth	1
PointShape	Star
⊞ Size	10, 10
SymbolType	Simple
UseOutline	False

Angle

Figure 1: Property-Grid Control

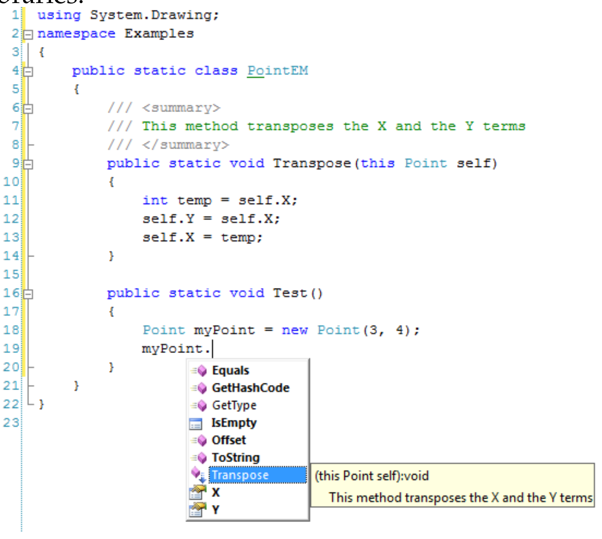
The Property Grid control shown in Figure 1 is a .Net control that creates a kind of tabular layout that itemizes each of the public properties on a class, and next to that, provides an interactive spot where a value can be changed. The default behavior of the editing region is type dependant, so that simple values might only allow a text-box style editing, while more complex members might have a drop-down control or even provide a button that launches an entire dialog. These behaviors can be customized for new class data types using so called 'Editor' classes that control how the property grid behaves during the editing process. The editor to use for a new class is controlled by the use of attributes.

The significance of this control is that it does not require any pre-existing knowledge of what properties exist on a variable. While we don't recommend that everything relies on property grids, we are using it as an example of an open ended user interface design. Without relying on anything more complex than a .Net property grid, it then becomes possible to provide a default user interface for any new symbol classes that do not explicitly define a user interface editor.

Extension Methods

Extension methods have been a part of the of the .Net framework since version 3.0. These methods appear to extend the number and type of methods that can be accessed programmatically from an existing class or interface, even if the class is a sealed class and cannot be modified through inheritance. (4) The methods and properties that appear in intellisense are normally limited by the type definition of that variable. However, with extension methods, new methods can be 'appended' to the existing class, without ever modifying the source code for the class. In the visual studio development environment, these extension methods are designated by a purple down arrow to the left of the method. The importance of extension methods for this paper is that they represent the means by which intermediate level developers can design programmatic access to control new symbology members, making them programmatically discoverable through intellisense to future developers.

Between open ended user interface components like the property grid dialog and the extension methods, applications like MapWindow can support a new design concept which can easily build from or extend core libraries.



```
using System.Drawing;
namespace Examples
{
    public static class PointEM
    {
        /// <summary>
        /// This method transposes the X and the Y terms
        /// </summary>
        public static void Transpose(this Point self)
        {
            int temp = self.X;
            self.Y = self.X;
            self.X = temp;
        }

        public static void Test()
        {
            Point myPoint = new Point(3, 4);
            myPoint.
        }
    }
}
```

(this Point self):void
This method transposes the X and the Y terms

Figure 2: Transpose Extension Method

MapWindow 6.0 Symbology

Symbol Class Hierarchy

As of 6/29/2009, the architecture for the symbology interfaces follows a standard idea where a random but simple coloring scheme is applied to an entire layer as soon as the layer is added to the map. The layer does not host the descriptive characteristics directly, but rather stores a reference to a single scheme. The type of scheme depends on the type of features being represented. A PointScheme, for instance, works with point data, while PolygonScheme and LineScheme represent schemes that are specific to describing polygons and lines. The classes use a collection concept, so that each Scheme represents a collection of Categories. The strict role of a category is that it combines a query string with a Symbolizer. The string is used to select the members that the symbology will be applied to. This enables an easy programmatic access to hosting complex attribute based symbology. Finally, a PointSymbolizer is made up of at least one, but potentially several overlapping Symbols. A LineSymbolizer is made up of Strokes. A PolygonSymbolizer is made up of Patterns.

Point Symbology

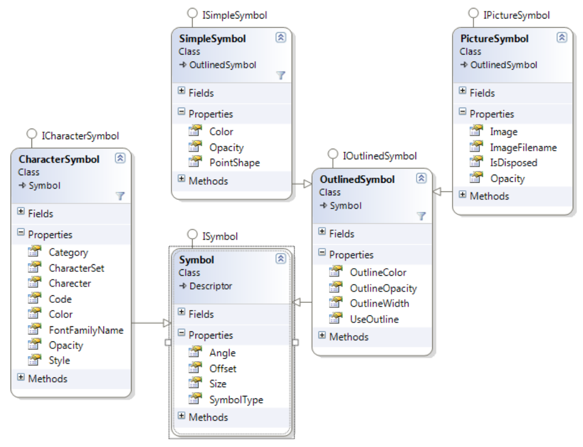


Figure 3: Point Symbol Classes

While each of the members above has a corresponding interface, it should be pointed out that only the lowest level need be developed in order for the developer to extend the graphical representations for vector features. For instance, if you wanted to design a new class to draw point types, implementing the ISymbol is sufficient, allowing it to use the pre-existing structure of symbolizers, categories and schemes. The existing symbol classes implement this interface and currently control their own drawing. All of the point symbol classes provide an offset, size and angle, but even characteristics like color are not universal. A PictureSymbol uses an image to control the drawing and has no information about coloring, though it can have an outline. A CharacterSymbol provides properties to control the font family name, the style and the character. The Characters can consist of artistically created symbol sets which support vector drawing and so will look good at any scale. A SimpleSymbol provides the most basic point symbology with an enumeration of default shapes. This simply uses GDI+ drawing methods to build the shapes programmatically.

Line Symbology

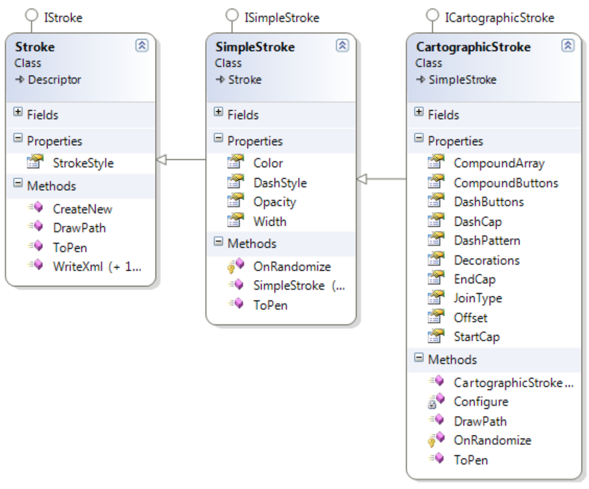


Figure 4: Line Stroke Classes

The equivalent concept for lines uses the IStroke interface. Each stroke represents a pass with a pen. At the moment, there is a SimpleStroke and CartographicStroke. The simple stroke allows for a color, width and dash pattern selected from an enumeration. It uses a default choice of rounded end caps. The Catographic stroke extends the capabilities of the simple stroke. It adds a custom dash pattern, custom contour pattern, line cap styles, and the ability to specify a number of point symbols as decorations along the line. Since these strokes are then layered one on top of the other, very complex line structures are now possible. If, however, a user wanted to design a new kind of stroke that was designed entirely by point symbols drawn at fixed distances, it could be done easily by creating a plug-in that implements the IStroke interface. Currently, the drawing is handled by passing the GraphicsPath for the lines to draw (after they have been translated to screen coordinates) to each stroke in sequence where it handles its own drawing using GDI+ graphics calls.

Polygon Symbology

The final set of descriptive symbols is for polygons. The PolygonSymbolizer is slightly more complex than the symbolizers for points or lines because in addition to having a collection of patterns, it also specifies a line symbolizer to use for drawing the borders. In this way, we get to re-use the drawing code for the lines when drawing the borders of the polygon. The individual patterns include a SimplePattern that only specifies a fill color. A PicturePattern is created using a collection of tiled images. Finally a GradientPattern uses linear, circular or rectangular gradients with various rotations and colors. The gradient can be controlled programmatically to work with many colors and positions (ranging from 0 to 1).

These new cartographic capabilities give an extraordinarily rich way to draw, color, or depict vector feature content, but what is most innovative is that the symbology itself is extensible. Raster symbology also exists, but is not tremendously different from techniques dating back to earlier versions of MapWindow, where several color breaks subdivide a raster.

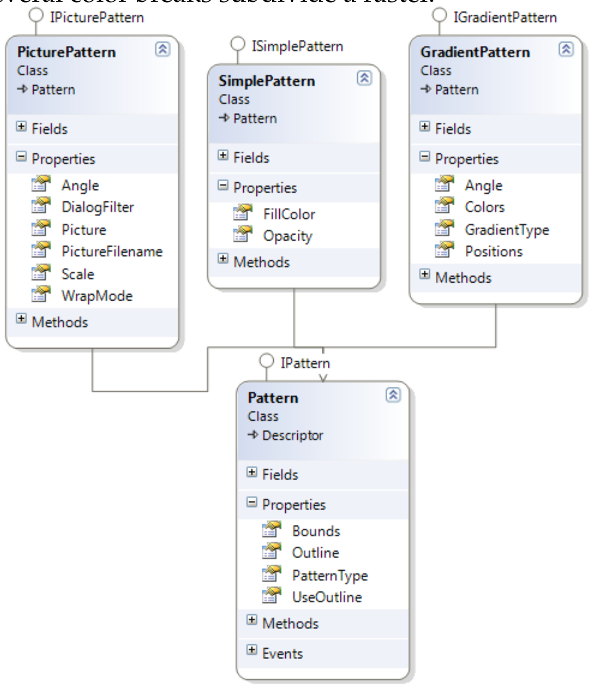
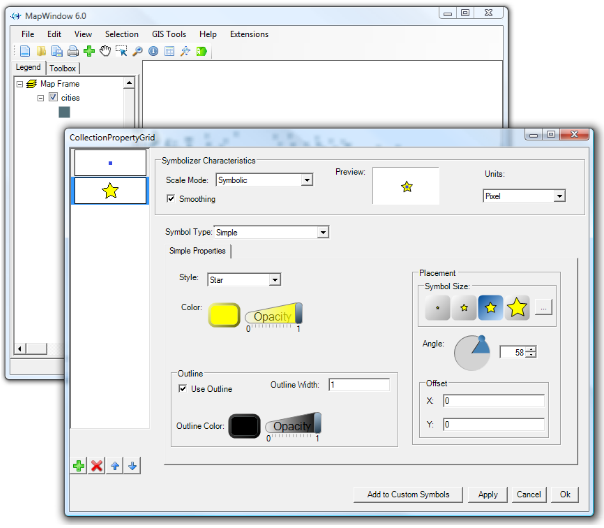


Figure 5: Polygon Pattern Classes

User Interface Design



Scale Mode: Symbolic
☑ Smoothing
Units: Pixel
Symbol Type: Simple
Style: Star
☑ Use Outline
Outline Width: 1
Angle: 58
X: 0
Y: 0

Figure 6: Point Symbolizer User Interface

Double clicking on the representation below the layer automatically launches a complex dialog that allows the customization of that symbol. As is illustrated in Figure 3, a fairly straight forward collection of properties exists described on the Simple Properties tab. A Drop-down currently set to 'Simple' controls what elements appear in the tab control. The extensibility allows new Symbol types to appear in this drop-down, and, while still in the planning stages, we plan on supporting a ISymbolUIEditor interface that allows users to specify the exact layout of a tab control for editing their custom symbol type chosen from the drop-down. In such a case, the layering (shown in the graphical list representations on the left side of the form) and content in the placement group would be re-used as common content, while new controls would appear where the style, color and outline groups currently appear. Similar interfaces exist for specifying character and picture symbols as well as all the types of stroke or pattern.

Discoverable Extension

It would be elegant for this paper if we could use examples of existing discoverable extension methods that control properties on symbol classes in order to best illustrate its usefulness in connection with an extensible symbology. Unfortunately, these haven't been written yet. However, we can gain important insights by drawing an analogy from an extensible data provider interface instead. At the time of submitting this paper, we have mostly begun using the extension methods for adding topology methods to the IFeature. The original intent of extension methods was to allow an easier way to work with sealed enumerable classes when working with System.Linq. The most common use of extension methods is simply to extend an existing class without changing it. However, the benefit that we have found to be the most useful from an extensibility standpoint is that we can make the interfaces that new developers have to implement much leaner. For instance, an IFeature currently specifies a coupling between vector information and attributes. This would be fairly straight forward to implement. However, adding an apparently simple new method called Intersects on this interface would force every individual that only wanted to support a new data format to also implement their own intersection code. Since topology methods are quite advanced and require a sizeable infrastructure to draw on, we chose to support the intersect behavior using extension methods. Any IFeature can now intersect with another IFeature, though it must yield to the definition provided by the extension method.

The danger of extension methods is that they can only be replaced with 'new' functionality and can't ever be overridden. This means that if a feature class implements its own Intersects logic, if an instance of that class is cast as an IFeature interface, then it will call the Intersects extension method, and not allow the new logic to replace the built in logic.

Summary and Conclusions

Symbology is a critical part of feature representation, but in both proprietary and open source GIS systems, this fundamental aspect of the representation is seldom extensible. The usual mechanism for managing extensibility tends to be limited to allowing automation of repetitive tasks, working well within the existing data management, analysis and rendering architectures. The extensible architecture used by this version of MapWindow allows symbolizers to be discovered at run-time, opening up a vast new domain for customization and personalization of the open source framework. The user interface design allows a coupling between customizable forms and smart default controls that work reasonably well even if no custom form is provided. The difficulty of working with open ended interfaces programmatically can largely be addressed by the introduction of new extension methods that access or set values. These extension methods can function at many different levels.

Harold A. Dunsford Jr.
Department of Geosciences
Idaho State University
Idaho, USA
dunsharo@isu.edu
Daniel P. Ames P.E.
Department of Geosciences
Idaho State University
Idaho, USA
amesdani@isu.edu

Bibliography

[1] D. P. Ames. *MapWinGIS Reference Manual: A function guide for the free MapWindow GIS ActiveX component*. Lulu.com, Morrisville, North Carolina, 2007.

[2] H. Dunsford. Restructuring of the mapwindow gis project. *PositionIT*, April/May:54–59, 2009.

[3] H. Dunsford et al. Community code development: A new paradigm for geospatial software in support of the data for environmental modeling(d4em) project. In *AWRA Spring Specialty Conference GIS and Water Resources V*, San Mateo, California, 2008.

[4] Microsoft. Extension methods (c# programming guide). 2008. URL http://msdn.microsoft.com/en-us/library/bb383977.aspx.

[5] MSDN. Explicit interface implementation: C# programming guide. 2008. URL http://msdn.microsoft.com/en-us/library/ms173157.aspx.

A Data System for Visualizing 4-D Atmospheric CO2 Models and Data

Tyler A. Erickson, Anna M. Michalak, John C. Lin

Abstract

This paper describes a geospatial data system that produces KML representations of complex spatio-temporal datasets related to modeling the atmospheric carbon cycle. KML is an open standard language for transferring annotated geospatial data that can be used by many modern geospatial software packages, particularly virtual globe applications. The server component of the data system is built using a variety of open source software packages, which provide flexibility for creating custom geospatial representations of the datasets. The paper shows examples of how KML representations of atmospheric CO2 datasets and model outputs can be visualized with virtual globe client applications, allowing a diverse group of users to explore the complex scientific datasets that are central to the discussion of climate change and global warming.

Introduction

The general population's awareness of, and interest in, climate change has increased dramatically in recent years. The consensus among climate scientists is that climate change is occurring, and that there is "very high confidence (9 out of 10 chance of being correct) that the global average net effect of human activities since 1750 has been one of warming" (19). Although the need for further work on specific scientific aspects remains, the discussion has largely progressed from "is climate change occurring?" to "how will climate change progress in the future?" and "how can human society mitigate or adapt to climate change?"

Several factors affect the energy balance of the climate system, including greenhouse gases, aerosols, and land surface properties. Of all the components, the increase in the atmospheric concentration of carbon dioxide (CO2) has been responsible for the largest increase in radiative forcing, or tendency to warm the Earth's surface. In 2007, the International Panel on Climate Change (IPCC) published a summary report stating that "carbon dioxide is the most important anthropogenic greenhouse gas that contributes to climate change" (19).

CO2 is continuously exchanged between the atmosphere and the Earth's surface, including land and oceans. The rate of exchange, or flux, is spatially and temporally variable, with this variability itself changing across scales. Overall, approximately half of current anthropogenic emissions of CO2 are taken up by land and oceans, which act as natural carbon "sinks." However, there is a lack of understanding of where these sinks occur, how they vary in time, and how they are affected by climate variability and other processes. This, in turn, limits the skill of existing models in predicting future changes in net carbon balance (12), and, therefore, the future atmospheric concentrations of CO2. Because CO2 flux can only be measured directly at relatively small spatial scales and at a limited number of sites (e.g. (2)), the estimation of CO2 fluxes on regional to continental scales relies heavily on models and indirect measurements.

One approach that carbon cycle scientists use to characterize the spatial and temporal variability of CO2 fluxes is to relate concentrations of atmospheric CO2 measured in the atmosphere to fluxes occurring in upwind regions, through a process called inverse modeling (e.g. (10)). This approach couples atmospheric CO2 observations with numerical models describing winds and weather patterns, and any additional information relevant to estimating carbon exchange, in order to trace fluctuations in atmospheric concentration measurements of CO2 backwards in space and time to the sources and sinks. This process allows scientists to characterize variability in upwind carbon exchange between the Earth's surface and the atmosphere.

Modeling and understanding the sensitivity of available CO2 observations to upwind fluxes is a key component of the inverse modeling framework. One approach, described in this paper, involves the use of a Lagrangian model, which simulates large numbers of particle trajectories backwards in time, starting at the location and time of the measurement, to identify the regions (in space and time) that influence the measured concentration. This modeling approach produces a diverse set of spatial and temporally varying datasets, with concentration measurement locations that are fixed in 3-D space and variable in time, particle trajectories that are variable in 3-D space and time, and sensitivity maps of measurements to surface fluxes that are variable in 2-D space and time.

The focus of this paper is the development of an approach for sharing these complex spatial and temporal datasets with a diverse set of users, ranging from the general public to carbon cycle scientists and decision makers. These spatial and temporal datasets can be difficult to explore and visualize, due to the high dimensionality of the data. An ideal tool for exploring these datasets would:

- possess capability for displaying 3-D spatially and temporally referenced data;

- have an easy-to-use interactive user interface, that allows the user to navigate the data in both space and time, and to query attributes of the data;
- have support for overlaying other georeferenced datasets; and
- be freely available and be compatible with commonly used software.

While many past approaches for visualizing complex geospatial data have some of these characteristics, all of the characteristics are important for communicating the data to a diverse set of users.

Fortunately, recent advances in geospatial software have reduced the barriers for visualizing 3-D and 4-D datasets. Virtual globes, which are interactive client applications, can be used to present custom 4-D datasets over a richly detailed reference model of the Earth (3). Accessing this type of visualization no longer requires advanced training in geospatial information systems or computer science, but rather is accessible by typical computer users. Google Earth, a popular virtual globe with a simple user interface, has been downloaded over 350 million times (31).

This paper presents a client-server geospatial information system that allows users to visualize several complex datasets used for understanding terrestrial carbon fluxes. The geospatial data server is built using free and open source software (FOSS) components that store, process, and format spatial and temporal datasets so that they can be easily visualized, using modern virtual globe software packages. The system described in this paper incorporates the desired characteristics listed above, and is built on a flexible platform that can be easily enhanced in the future.

Background

This section presents an overview of the scientific background for the models used to generate datasets included in the visualization and of open source software development.

Atmospheric CO2 Data and Modeling

Regular atmospheric measurements of CO2 began in 1958 with observations taken at Mauna Loa, Hawaii (19). Since that time, global atmospheric CO2 monitoring networks have expanded significantly. The National Oceanic and Atmospheric Administration (NOAA) Earth System Research Laboratory (ESRL) Global Monitoring Division (GMD) maintains the NOAA-ESRL Cooperative Air Sampling Network, which currently includes over 150 sites globally (28). More recently, NOAA-ESRL-GMD has developed a Tall Tower Network of sites with continuous observations of CO2 and related gases (29). This network focuses on the continental United States, and currently includes eight tall tower sites. Continuous measurements are particularly useful for inverse modeling studies, because they allow individual sampling locations to "see" large regions, as a function of changing wind directions and weather patterns.

Data from two of the Tall Tower Network sites are used for illustration purposes in the presented application. The first site is the LEF tower in Park Falls, Wisconsin, a 396m tower that has been operating since 1994. The second site is the 107m AMT tower in Argyle, Maine, which has been operating since 2003. In the presented analysis, the concentration measurements from these towers were averaged to 3-hour intervals over the time period of June 1 to July 8, 2004. Meteorological information derived from the WRF model and atmospheric trajectories from the STILT model (described later in this section) were used to calculate the influence of atmospheric CO2 arriving from outside the examined domain on the available observations, and this impact was pre-subtracted from observations. As a result, the CO2 concentration variations examined here are influenced only by carbon sources and sinks within the examined North American domain (15).

As discussed in Section 1, one method used by carbon cycle scientists to quantify the sensitivity of atmospheric measurements to surface fluxes involves simulating collections of particles backwards in time from the measurement location, while modeling the turbulent dispersion as a stochastic process (30). The Stochastic Time-Inverted Lagrangian Transport Model (STILT) is one such model that estimates subgrid particle movement by interpolating gridded meteorological fields to the location of the particle and parameterizing the turbulent motions as functions of these meteorological variables (23). The STILT model is an adaptation of the HYSPLIT trajectory model (7) but incorporates four key improvements:

- a modified turbulence scheme that ensures adherence to the "well-mixed criterion", a manifestation of the 2nd Law of Thermodynamics (49);
- close coupling to atmospheric models to minimize deviations from mass conservation (27);
- capability to account for errors in the meteorological fields using a Monte Carlo method (22);
- a revised method for estimating the height of the planetary boundary layer (PBL) that generalizes to unstable, neutral, and stable conditions (50). The PBL is the lower portion of the atmosphere in which trace gas concentrations are most sensitive to surface fluxes (47).

The STILT model runs can utilize gridded meteorological datasets from a variety of sources, including high-resolution limited area models like the Regional Atmospheric Modeling System (RAMS) (5) and the Weather Research and Forecasting (WRF) model (45). For this study, WRF v2.2 was used to generate meteorological fields used by STILT (27). A 3-level nested grid domain was used, with base grid of 40-km spanning 10°N to 70°N, and 170°W to 50°W with higher resolution grids of 10-km resolution over the Eastern half of the United States, and 2-km resolution grids surrounding three tall towers, including the two examined in the current study. 10 day back-trajectories of 500 particles per hour from each receptor (tall tower location) were simulated. From these trajectories, a temporal grid was produced, representing the sensitivity of atmospheric concentrations to upwind surface fluxes on a 3-hour time interval and a 1°×1° grid. The sensitivities were derived using times and locations where the particles were below the planetary boundary layer, indicating that the air parcel is sensitive to fluxes occurring at that location. These sensitivity grids, a.k.a. *footprints*, provide the linkage between locations measuring atmospheric concentrations with upwind fluxes.

Biospheric fluxes play a critical role in the carbon cycle and atmospheric CO2 concentrations, imposing marked diurnal and seasonal cycles on CO2 variations. The terrestrial biosphere absorbs CO2 through photosynthesis during the daytime of the growing season and releases CO2 back to the atmosphere through respiration during the nighttime and the winter season (43). To simulate these biospheric processes, a wide variety of biospheric models have been developed over the past several decades (44).

In this project, CO2 surface fluxes generated using the CASA terrestrial carbon cycle model (39) were used, which simulates ecosystem processes and is driven by satellite observations and meteorology. The surface fluxes, which were presented in Olson and Randerson (32), were mapped every 3 hours and at 1°×1° resolution, matching the sensitivity datasets.

The biospheric processes controlling carbon dioxide fluxes are complex functions of a large number of variables, including solar radiation, temperature, vegetation type, nutrient availability, disturbance history, and soil moisture, among other factors. The interactions between these variables result in a heterogeneous distribution of fluxes that vary both in space and time.

Geovisualization

Geovisualization is an emerging field that includes approaches from a variety of disciplines, including cartography, scientific visualization, exploratory data analysis, and geographic information science, which provide tools for the visual exploration, analysis, synthesis and presentation of data that contain geographic information (9, 24). Geovisualization systems may include support for temporal information, which is often lacking in traditional geographic information systems (26) but may be of critical importance for understanding temporally-variable environmental datasets.

Many geovisualization systems allow for significant levels of interaction, which allows users to explore data, synthesize, confirm and communicate ideas through guided discovery (8). In recent years the general public has become familiar with interactive visualization in several forms such as online maps used for driving directions and virtual globes used by television news programs to give spatial context to remote events.

The scientific community has begun to utilize freely available virtual globe applications as geovisualization tools to communicate scientific results (3) such as meteorological data (46), disease transmission observations (18), and vertical profile data obtained from satellite sensors (4). Many virtual globes can display custom user content that is spatially (and may also be temporally) referenced, provided the data is in a standard format.

The Keyhole Markup Language (KML) is an open standard XML-based language for exchanging georeferenced feature data, styling, and annotation. KML was originally created by Keyhole, Inc. and further expanded by Google after it acquired Keyhole, Inc. in 2004. The KML 2.2 specification was submitted to the Open Geospatial Consortium (OGC) standards organization, and became an official OGC standard (the OpenGIS KML Encoding Standard) on April 14, 2008 (34). The adoption of the standard by the OGC should encourage the development of visualization clients and server software applications that use KML to exchange spatial and temporal data.

One import feature of the KML language is the ability to access additional KML-formatted data at a specified URL using the Network Link functionality. This allows KML viewers to hierarchically link to large external datasets stored on or generated by remote servers. This paper describes a data system, built with open source software components, that produces KML formatted data with spatial and temporal attributes. Open source is a software development method that allows wide accessibility to the software's code base for use, improvement, and redistribution in modified or unmodified forms.

Open source software has been adopted in many areas of academic research, such as the R language for statistical analysis (16). Rey (42) describes past interactions between academic geospatial researcher and the open source communities, as well potential opportunities for future cross-collaboration. In many ways, the open source software development process is similar to the scientific process of knowledge development (21) in that it promotes peer review by external developers (37) and the open source licenses allow for continuous enhancements and improvements to a body of knowledge.

Data System

A flexible three-tier data system architecture was designed to enable visualization of the carbon cycle datasets (Figure 1). The database and application tiers store, manipulate, and format the carbon cycle datasets as requested by client applications via a HTTP connection, enabling access by any client computer that has access to the application server. The database and application tier are created using open source geospatial components, and communicate with the client applications using KML, an open standard language for transferring geospatial data.

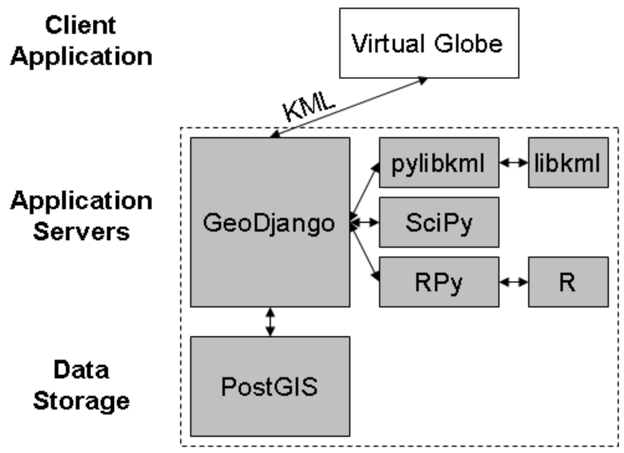


Figure 1: Overview of the open source data system components (shaded grey) and their relationship to the Google Earth virtual globe client application. The core of the application server, GeoDjango, uses the functionality of several open source libraries for formatting and manipulating data.

Data Storage

The foundation tier of the data system is PostGIS (41), a spatial extension to PostgreSQL (38), a client-server relational database. The PostGIS extension adds geographic data types and spatial operators to PostgreSQL, which enables the database to store spatial, temporal and attribute information as records. PostGIS was selected because it is a widely used, free and open source, and follows the *Simple Feature Access Specification for SQL* (35), which is an open international standard for storing and accessing geographic features.

Datafile Format	Packages Used	Dataset Examples
text file	SciPy (1)	land-water mask
MATLAB (48) data file	SciPy	sensitivity maps, biospheric flux maps
R formatted data file	RPy (25) & R (16)	particle locations

Table 1: Summary of the open source packages used to import datafiles.

Application Server

The GeoDjango web framework is the core of the data system. GeoDjango is an integration of the Django (6) web framework with several open source geospatial libraries (GEOS, proj.4, GDAL) that includes support for spatial databases and provides spatial processing functionality. GeoDjango exposes the geometric data types and operators provided by PostGIS at the database level. GeoDjango is written in Python (40), a general-purpose object-oriented programming language that can be used for many kinds of software development and is known for its code readability and its ability to integrate with other languages and tools. GeoDjango is a web server that shares content using the Hypertext Transfer Protocol (HTTP). While the content is typically a web page, for this data system we have customized GeoDjango to serve KML documents.

The functionality of GeoDjango is enhanced by integrating with several open source libraries, as shown in Figure 1. The libkml C++ library (20) provides a structured way of authoring valid KML documents. The pylibkml library (11) is a Python wrapper for libkml, which simplifies the use of the libkml objects from within Python code. Key elements of the KML language that are used by this data system are: (1) features that have temporal attributes to denote a specific time or interval of time, (2) network link elements that allow for accessing remote datasets, and (3) model elements that can incorporate 3-D models in the virtual globes to symbolize attributes.

Measurement and model data used by the data system originate from a variety of sources and occur in a variety of formats. Several open source packages, used by the data system, are summarized in Table 1.

Client Application

The client application's role is to present spatially and temporally referenced data to the user. Although any application that implements the OGC KML standard could be used to view the data, the visualizations shown in this paper were produced using the Google Earth virtual globe (version 5.0) (14).

Application: Visualizing Atmospheric CO2 Data

This section describes how the data system was configured to manage the atmospheric CO2 datasets.

Model Representation

The atmospheric transport and biospheric models produce datafiles with numerous attributes. Data models were created in the GeoDjango framework to describe data objects, their attributes, and relationships between

Conceptual Object	Description	Spatial & Temporal Attributes
Sensor	A tall tower measurement location, which includes the static 3D location of the sensor.	3D point
Particle	A single simulated particle that represents a parcel of air at a specific time that will later arrive at the sensor location	time instant
Location	A simulated 3D point location of a particle at a specified time that corresponds to a specified concentration measurement. Also includes the height of the planetary boundary layer.	3D point; time; corresponding to a specific time interval (measurement)
Sensor Measurement	Average CO2 concentration measured over a time interval	time interval (measurement)
Sensitivity	The sensitivity of a measurement to a surface flux	time interval (measurement) & time interval (surface flux)
Surface Region	A discretized portion of the Earth's surface	2D polygon
Surface Region Flux	A modeled surface flux value	time interval (surface flux)

Table 2: Summary of the GeoDjango data model objects used to manage the measured and modeled datasets.

objects. Table 2 gives a summary of the conceptual objects that are used in this visualization and highlights the spatial and temporal attributes of each object, while Figure 2 shows how the conceptual data objects are interrelated.

In conjunction with the imported data, GeoDjango uses the data model objects to create, populate, and save instances of the data records to the PostGIS database. Similarly, in order to access the data for the visualizations, GeoDjango uses the data model objects to query data stored in the PostGIS database.

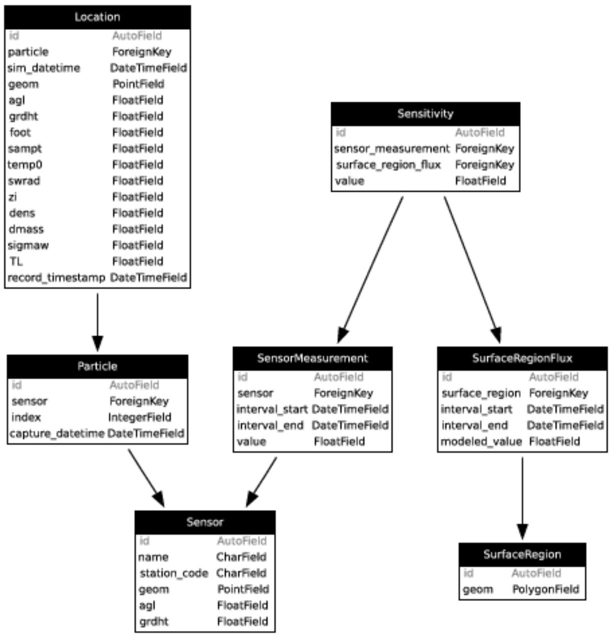


Figure 2: Overview of the GeoDjango data models and their relationships that are used to model the CO2 measurement and model output datasets.

Atmospheric Measurements of CO2

The data for the atmospheric CO2 measurements, described in the background section, was obtained as MATLAB data files (one per measurement sensor). A Python script is used to parse the data files and import records into the PostGIS data tables, using the SensorMeasurement object of the GeoDjango data models (Figure 2). KML-formatted representations of the CO2 concentrations are obtained by submitting a URL request to the GeoDjango server application. For example, a request for a KML representation of a measurement data series for the LEF Tall Tower sensor between June 1, 2004 and August 8, 2004 would be:

```
http://localhost/measurement/station=LEF/
start=2004-06-01T00:00:00Z/end=2004-07-08T00:
00:00Z/series.kml
```

An example rendering of the CO2 concentration relative to background measured at the LEF Tall Tower is shown in Figure 3(a), and an example KML representation is shown in Table 3. The KML model representation uses the <*Location*> element to set the 3-D position and the <*Link*> element to reference an external COLLADA model of a unit-sized, colored sphere (green_sphere.dae). The volume of the sphere is set to be proportional to the absolute value of the difference between the measured and background concentrations using the <*Scale*> element. CO2 concentrations that are greater (less) than the background concentration are symbolized in green (blue). Although KML is itself an OGC standard, this encoding of the sensor data does not conform to any of the current OGC Sensor Web Enablement specifications (33).

When a user navigates time using the time slider control in Google Earth, the color and size of the sphere change according to the selected time, conveying the temporal variability of the concentrations.

Table 3: Example excerpt of KML for displaying a CO2 measurement observation.

```
<Placemark>
 <name>2004-06-0101:00</name>
   ...
 <TimeSpan>
   <begin>2004-05-31T23:30:00Z</begin>
   <end>2004-06-01T02:30:00Z</end>
 </TimeSpan>
 <Modelid="model_1366">
   <altitudeMode>relativeToGround</altitudeMode>
   <Location>
     <longitude>-90.273157</longitude>
     <latitude>45.945048</latitude>
     <altitude>396</altitude>
   </Location>
   <Scale>
     <x>28.1536</x>
     <y>28.1536</y>
     <z>28.1536</z>
   </Scale>
   <Link>
     <href>green_sphere.dae</href>
   </Link>
 </Model>
</Placemark>
```

Table 4: Portion of the KML document that defines a particle location for a specified time and particle paths between consecutive locations. Although the location is valid for an instant in time for a moving particle, the KML <TimeSpan> tag for the particle location specifies a short interval to enhance the visualization. Paths between consecutive locations are extended to the ground surface.

```
<Document>
  <Styleid="style_particle">
    <IconStyle>
      <Icon>
        <href>http://maps.google.com/mapfiles/kml/shapes/shaded_dot.png</href>
      </Icon>
    </IconStyle>
  </Style>
  <Style id="style_particle_path_above_bnd">
    <LineStyle>
      <color>ff0000ff</color>
      <width>4</width>
    </LineStyle>
    <PolyStyle>
      <color>400000ff</color>
    </PolyStyle>
  </Style>
  <Style id="style_particle_path_below_bnd">
    <LineStyle>
      <color>ff00ff00</color>
      <width>4</width>
    </LineStyle>
    <PolyStyle>
      <color>4000ff00</color>
    </PolyStyle>
  </Style>
  <Folder>
    <name>locations</name>
    <visibility>0</visibility>
    <open>1</open>
    <Placemark>
      <visibility>0</visibility>
      <TimeSpan>
        <begin>2004-06-03T04:30:00Z</begin>
        <end>2004-06-03T04:50:00Z</end>
      </TimeSpan>
      <styleUrl>#style_particle</styleUrl>
      <Point>
        <extrude>1</extrude>
        <altitudeMode>relativeToGround</altitudeMode>
        <coordinates>-83.0984,77.8721,4240.77</coordinates>
      </Point>
    </Placemark>
  </Folder>
    <Folder>
    <name>path</name>
    <Placemark>
      <visibility>0</visibility>
      <TimeSpan>
        <begin>2004-06-03T04:40:00Z</begin>
        <end>2004-06-10T19:00:00Z</end>
      </TimeSpan>
      <styleUrl>#style_particle_path_above_bnd</styleUrl>
      <LineString>
        <extrude>1</extrude>
        <tessellate>1</tessellate>
        <altitudeMode>relativeToGround</altitudeMode>
        <coordinates>  -83.0984,77.8721,4240.77  -83.1489,77.8293,4255.398
                                                              </coordinates>
      </LineString>
    </Placemark>
    <Placemark>
       ...
  </Folder>
</Document>
```

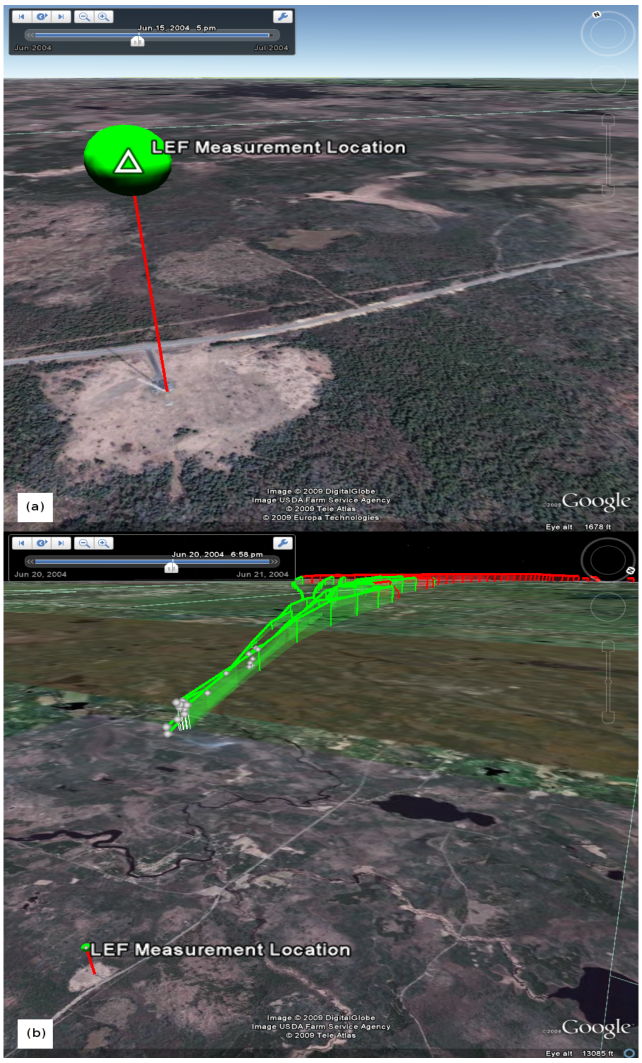


Figure 3: Screenshots of Google Earth renderings of the KML generated by the data system. (a) CO2 concentrations measured at the LEF Tall Tower. The volume of the sphere is proportional to the difference between the measured and the background concentrations. The sphere color (green/blue) is used to denote whether the measured concentrations are below/above the background concentration. (b) Simulated particle locations (grey) representing air parcels that are sampled by the LEF Tall Tower. The paths between simulated locations are colored green/red to indicate whether the particle is below/above the atmospheric boundary layer.

Air Parcel Simulation

The source of the particle location data are R-formatted datafiles produced by the STILT atmospheric transport model. A Python script is used to parse the data files and import records into the PostGIS data tables, using the GeoDjango data models.

KML-formatted representations of the particle data are obtained by submitting a URL request to the GeoDjango server application. For example, a request for single particle trajectory captured by the LEF Tall Tower measurements on 2004-06-10 at 19:00 (UTC) is:

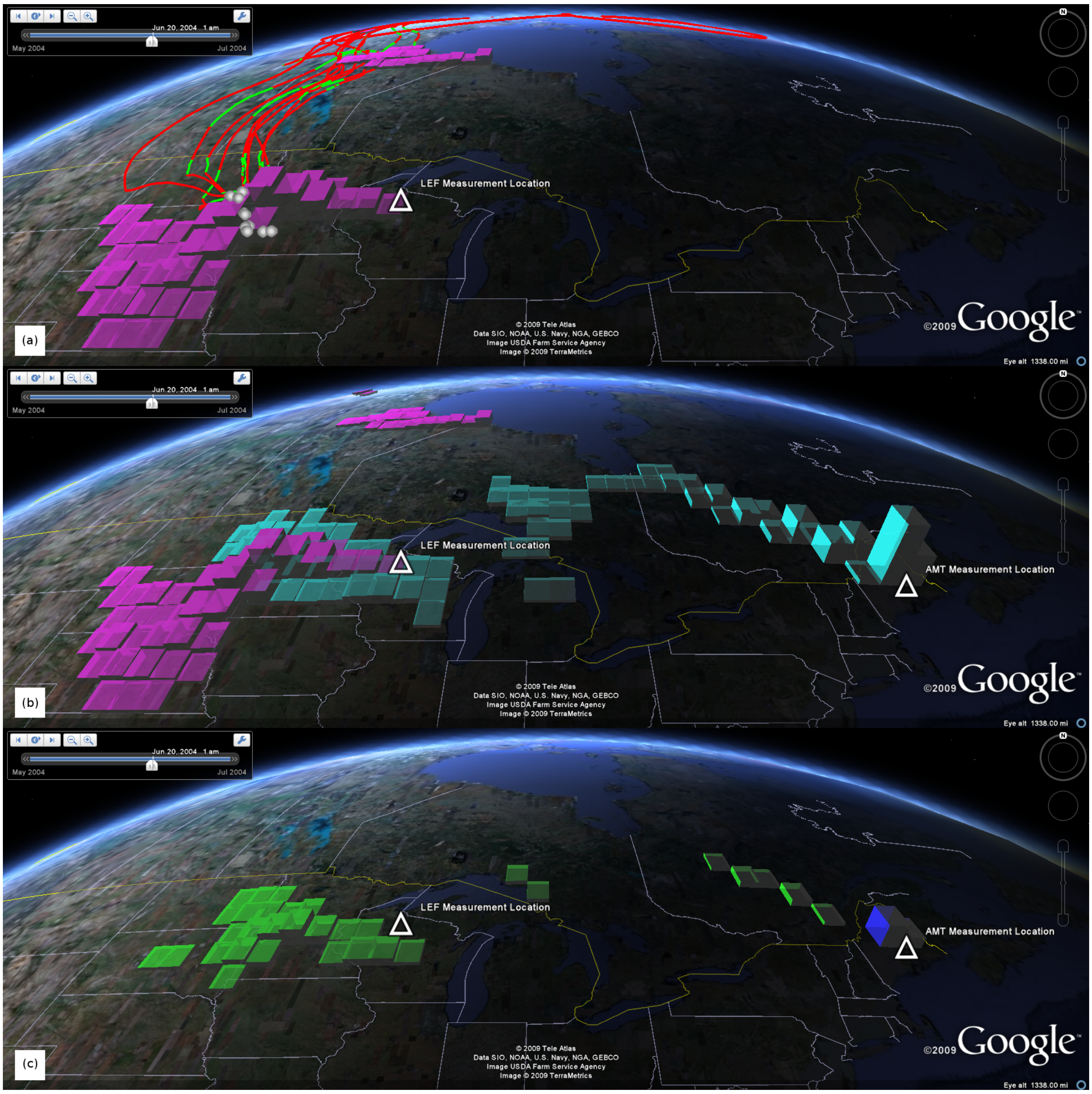


Figure 4: Screenshots of Google Earth renderings of the KML generated by the data system. The screenshots all represent the conditions at the same time: 6/20/2004 at 1:00 (UTC). (a) Particle locations (grey) and trajectories (red and green) for air parcels that are sampled by the LEF Tall Tower, along with the estimated sensitivity (purple) of the LEF tower measurements to surface fluxes. The height of the column is proportional to the sensitivity. (b) The overlaid sensitivity maps for the LEF and AMT tall towers. The sensitivity of CO2 measurements taken at the LEF/AMT tall tower to surface flux are symbolized in purple/cyan. (c) The sensitivity of CO2 measurements to the modeled biospheric flux of CO2 for measurements taken at the LEF and AMT tall towers. The height of the columns is proportional to the sensitivity multiplied by the model biospheric flux. Fluxes from the ground surface to atmosphere (respiration) are symbolized blue, while fluxes from the atmosphere to the ground surface (photosynthesis) are symbolized green.

http://localhost/particle/station=LEF/capture=2004-06-10T019:00:00Z/index=1/track.kml

A user may also request a series of particle trajectories. For example, a request for a series of 100 particle trajectories is:

http://localhost/particle_tracks/station=LEF/capture=2004-06-10T19:00:00Z/index_start=1/index_end=100/track.kml

Sample KML representations of the particle locations and paths between consecutive locations are shown in Table 4. An example Google Earth rendering of the particle tracks KML corresponding to measurements collected at the LEF Tall Tower is shown in Figure 3(b) and a screenshot of several paths is shown in Figure 4(a). The particle paths are symbolized, using the <*StyleUrl*> and <*Style*> elements, according to whether the particle location is above (red) or below (green) the atmospheric boundary layer. Particle locations that are below the atmospheric boundary layer contribute to the sensitivity of the sensor measurements to surface flux. Similarly, the path between consecutive particle locations is constructed with posts extending from each measurement location to the ground surface.

Sensitivity Maps

Spatially discretized sensitivity maps are produced by aggregating particle locations that are below the atmospheric boundary layer for a specified time interval (flux time), and that correspond to a specified measurement time interval (measurement time).

Table 5: Portion of the KML document that symbolizes the sensitivity of CO2 measurements made at the LEF Tall Tower to a surface flux for a 1°×1° ground region and 3-hour time interval.

```
<Document>
  <Style id="style_positive">
    <PolyStyle>
      <color>bfff00ff</color>
      <outline>0</outline>
    </PolyStyle>
  </Style>
  <Folder>
    <name>LEF sensitivity</name>
    <Placemark>
      <TimeSpan>
        <begin>2004-06</begin>
        <end>2004-06-01T03:00:00Z</end>
      </TimeSpan>
      <styleUrl>#style_positive</styleUrl>
      <MultiGeometry>
        <Polygon>
          <extrude>1</extrude>
          <tessellate>1</tessellate>
          <altitudeMode>relativeToGround</altitudeMode>
          <outerBoundaryIs>
            <LinearRing>
        <coordinates> -96,44,15036.9 -95,44,15036.9 -95,45,15036.9 -96,45,15036.9
                -96,44,15036.9 </coordinates>
            </LinearRing>
          </outerBoundaryIs>
        </Polygon>
      </MultiGeometry>
    </Placemark>
    <Placemark>
      ...
</Document>
```

A sample KML representation of a sensitivity map element is shown in Table 5. The TimeSpan element refers to the time interval of the surface flux, and the symbolized map elements correspond to aggregate sensitivity of the all measurements taken 0-10 days after the surface flux time interval.

Examples of Google Earth screenshots rendering the sensitivity map KML files are shown in Figure 4(a) and 4(b). The extruded height of the sensitivity map elements is proportional to the sensitivity of the CO2 measurements to the surface flux, and the elements are uniquely colored to correspond to the measurement sensor (magenta – LEF Tall Tower; cyan – AMT Tall Tower). As can be seen by the overlapping regions shown on the center section of Figure 4(b), more than one measurement sensor may be sensitive to surface fluxes occurring for a particular region.

While the sensitivity maps describe regions that may have affected the concentration measurements, a more informative variable to visualize is the sensitivity value multiplied by a modeled CO2 flux. The bottom section of Figure 4(c) shows a Google Earth rendering of this variable using biospheric fluxes of CO2 estimated by the CASA biospheric model. The extruded height of the map elements is proportional to the absolute value of the sensitivity multiplied by the biospheric flux. The elements are colored according to the direction of the flux, with green indicating transfer of CO2 from the atmosphere to the land surface (photosynthesis), and blue indicating release from the land surface to the atmosphere (respiration).

Discussion

CO2 Visualizations

The use of a user-friendly virtual globe application, such as Google Earth, makes the datasets accessible to a wide group of users. One use of the data system has been to create data layers that can introduce carbon cycle science to non-specialists such as educators. A KML document that included many of the datasets discussed in the paper was selected as a winner of Google For Educators 2009 KML in Research competition because "it represented a novel and compelling representation of science using Google Earth and the KML language." [39] The ease of use of virtual globe applications enables other non-specialist groups, such as the general public and decision makers whom are neither familiar with carbon cycle science or geospatial information tools, to access and explore the datasets.

Although carbon cycle scientists have long had other tools such as mapping applications for understanding the datasets, this group of advanced users can also benefit from having access to visualization tools for viewing their data. Virtual globe applications can be used for exploratory data analysis, helping scientists to identify issues with their data that are difficult to detect using traditional tools. For example, the virtual globe interface allows users to easily change perspective to view both the 'forest' (carbon sensitivity variations across North America) and the 'trees' (particle tracks that the sensitivity values are based on). This has been used to identify potential issues with the simulation of individual particle tracks which are not apparent when viewing the sensitivity footprints at the continental scale. Also, due to the three-dimensional nature of trajectories and the spatially and temporally varying footprints, the visualization software described in this

[39]Ryan Falor (Google), personal communication, 2 March 2009

paper provides a valuable tool to probe changes due to dynamic changes in the wind patterns. The tool enables the user to investigate the dispersion pattern of trajectories and quickly reveals the land areas whose emissions are sampled by the trajectories, whenever they dip within the PBL.

A key advantage of using a virtual globe to visualize spatio-temporal data is the ability to interactively navigate the temporal aspect of the datasets. The ability to select a specific time and to play forward and backward in time allows users to explore the temporal variability in the 'footprints' of each concentration measurement location. This conveys to users the effect that constantly changing meteorological fields have on the potential information can be extracted from the concentration measurements, a concept that is difficult to fully convey with a non-temporal representation that can only contain data for a single pre-specified time interval. While past work has used geovisualization tools to create movies to present changes over time (13), the non-interactive nature of movies does not facilitate exploration of the data. The sensitivity dataset actually has two temporal dimensions, which cannot be directly represented in KML because KML objects can only contain a single primitive element for time.

The sensitivity dataset relates a concentration measured over a time interval (time 1) to a surface flux that occurred over a previous time interval (time 2). The approach taken in this paper is to generate a KML representation of sensitivity integrated over the entire time that concentrations were measured (time 1), so that the temporal primitive element in the KML refers to the time of the surface flux (time 2). This allows the user to use the Google Earth time slider to see variations in surface flux sensitivity. A complementary and equally useful approach would be to create a second KML representation that integrates the sensitivity data over the surface flux time (time 2), so that users could use the time slider to see variations in regions that a particular concentration measurement is sensitive to, regardless of when the surface flux occurred.

Using Open Source Software for Research

The data system described in this paper was designed to manage a variety of spatially and temporally referenced datasets, which is a typical need for scientists that monitor the Earth's environment. The data system was constructed using a variety of open source software components because of the numerous advantages that open source software has over proprietary components in terms of flexibility, maintainability, simplicity, and the developer community.

Because the source code is available, modifications can be made to extend the functionality provided by the component. If these modifications are contributed back to the open source project, they may be incorporated into the core product. This can be advantageous to the author of the modification, because future enhancements to the core product will be compatible with the modifications. This is particularly important for code developed as part of academic research projects, which generally have a finite project length and do not support long-term maintenance of software.

In the lead author's experience, the process of designing prototype geospatial applications for research with open source software is quite different than with closed source proprietary systems. When an issue is encountered with closed source software, a user is restricted in their options for resolving the issue because they are prevented from inspecting and modifying the source code. When developing with open source software, there is always a way forward. If the user has sufficient technical skills they can debug and fix the issue themselves, or if not they can hire someone to fix the issue. Online forums and chat groups for both types of systems provide support to programmers, but for open source projects there is less separation between the developers and the users of the software, resulting in quicker and more relevant answers to questions.

Google Earth, a freely available proprietary closed source application, was primarily chosen as the visualization client because of its support of the full KML specification and its availability on multiple operating systems (Linux, Mac, Windows). Other beneficial features are the easy-to-use interface, the direct access to detailed vector data layers and high-resolution imagery that provide spatial reference, and the availability of rendering effects such as atmosphere and sun options that enhance the user's perception of spatial and temporal change, However the KML documents created by this data system can be rendered by any virtual globe application that supports the full OGC-KML specification (36).

Conclusions

This paper has described a prototype data system for producing visualizations of datasets related to atmospheric CO2 modeling. The intent has been to present an example of using open source geospatial software to manage complex spatial and temporal data, and to produce datasets in an open standard format that can be viewed in virtual globe applications as well as used by other geospatial software. While this paper focused on datasets related to modeling the atmospheric CO2 cycle, the data management and visualization techniques are appropriate for other regional to global-scale spatial-temporal datasets.

Providing a means of visualizing spatial-temporal datasets is important step toward increasing the non-specialists' knowledge of the complex processes that cause climate change. Geovisualizations, such as those presented in this paper, can be used to familiarize the general public, decision makers, and future researchers with an understanding the current state of knowledge and challenges in modeling Earth's systems and predicting future responses.

The data system presented in this paper is a work in progress, with numerous enhancements envisioned. Additional KML representations of the sensitivity datasets could be added to allow users to visualize according to the time of concentration measurement (in addition to the currently implemented time of surface flux). The current approach of storing discretized spatial variables (i.e. sensitivity maps or biospheric flux maps) as polygons could be improved by implementing raster data storage. Additional complex datasets used for inverse modeling, such as best estimate maps and covariance matrices, could be included in the visualization.

Acknowledgements

The authors gratefully acknowledges Adam Hirsch for discussions related to particle tracking as well as Sharon Gourdji, Kimberly Mueller, Vineet Yadav, and Deborah Huntzinger who provided the modeling datasets used in this visualization. The modeling datasets were created with support from the National Aeronautics and Space Administration under grant NNX06AE84G *Constraining North American Fluxes of Carbon Dioxide and Inferring Their Spatiotemporal Covariances through Assimilation of Remote Sensing and Atmospheric data in a Geostatistical Framework* issued through the ROSES A.6 North American Carbon Program to the University of Michigan.

Tyler A. Erickson
Michigan Tech Research Institute
Michigan Technological University
3600 Green Court, Suite 100
Ann Arbor, MI 48105
tylerickson@gmail.com

Anna M. Michalak
Department of Civil and Environmental Engineering, and
Department of Atmospheric, Oceanic and Space Sciences
The University of Michigan
183 EWRE Building
Ann Arbor, MI 48109-2125
anna.michalak@umich.edu

John C. Lin
Department of Earth and Environmental Sciences
University of Waterloo
200 University Avenue West
Waterloo, ON, Canada N2L 3G1
jcl@uwaterloo.ca

Bibliography

[1] Anonymous (2009) SciPy - Scientific Tools for Python. http://www.scipy.org/ Accessed 29 June 2009

[2] Baldocchi D, Finnigan J, Wilson K, Paw U KT, Falge E (2000) On Measuring Net Ecosystem Carbon Exchange Over Tall Vegetation on Complex Terrain. Boundary-Layer Meteorology. 96(1):257-291

[3] Butler D (2006) Virtual globes: The web-wide world. Nature. 439(7078):776-778

[4] Chen A, Leptoukh G, Kempler S, et al. (2009) Visualization of A-Train vertical profiles using Google Earth. Computers & Geosciences. 35(2):419-427

[5] Cotton WR, Pielke Sr. RA, Walko RL, et al. (2003) RAMS 2001: Current status and future directions. Meteorology and Atmospheric Physics. 82(1-4):5-29

[6] Django Software Foundation (2009) Django | The Web framework for perfectionists with deadlines. http://www.djangoproject.com/ Accessed 28 June 2009

[7] Draxler RR, Hess GD (1998) An overview of the HYSPLIT_4 modelling system for trajectories, dispersion and deposition. Aust. Meteor. Mag. 47(4):295-308

[8] Dykes J (2005) Facilitiating Interaction for Geovisualization. In: Exploring geovisualization. Kidlington, Oxford, UK: Elsevier.

[9] Dykes J, MacEachren AM, Kraak MJ (2005) Exploring geovisualization. In: Exploring geovisualization. Kidlington, Oxford, UK: Elsevier.

[10] Enting IG (2002) Inverse problems in atmospheric constituent transport. Cambridge University Press

[11] Erickson TA, Kemker R (2009) pylibkml - a python wrapper for the libkml library. http://code.google.com/p/pylibkml/ Accessed 29 June 2009

[12] Friedlingstein P, Cox P, Betts R, et al. (2006) Climate Carbon Cycle Feedback Analysis: Results from the C4MIP Model Intercomparison. Journal of Climate. 19:3337

[13] Gardiner N (2006) High Definition Geovisualization: Earth and Biodiversity Sciences for Informal Audiences. In: Geographic Hypermedia. http://dx.doi.org/10.1007/978-3-540-34238-0_24 Accessed 26 June 2009

[14] Google Earth. http://earth.google.com/ Accessed 29 June 2009

[15] Gourdji S, Hirsch A, Mueller K, et al. (2009) Regional-scale geostatistical inverse modeling of North American CO2 fluxes - A synthetic data study. Atmospheric Chemistry and Physics, 9, 22407-22458.

[16] Ihaka R, Gentleman R (1996) R: A Language for Data Analysis and Graphics. Journal of Computational and Graphical Statistics. 5(3):299-314

[17] IPCC (2007) Summary for Policymakers. In: Climate Change 2007: The Physical Science Basis. Contribution of Working Group I to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change. Cambridge, United Kingdom and New York, NY, USA: Cambridge University Press.

[18] Janies D, Hill AW, Guralnick R, et al. (2007) Genomic Analysis and Geographic Visualization of the Spread of Avian Influenza (H5N1). Syst Biol. 56(2):321-329

[19] Keeling CD, Bacastow RB, Bain-Bridge AE, et al. (1976) Atmospheric carbon dioxide variations at Mauna Loa Observatory, Hawaii. Tellus. 28:538-551

[20] kml.mashbridge, kml.bent (2009) libkml - a KML library written in C++ with bindings to other languages. http://code.google.com/p/libkml/ Accessed 29 June 2009

[21] von Krogh G, Spaeth S (2007) The open source software phenomenon: Characteristics that promote research. Journal of Strategic Information Systems. 16(3):236-253

[22] Lin JC, Gerbig C (2005) Accounting for the effect of transport errors on tracer inversions. Geophys. Res. Lett. 32(1)

[23] Lin JC, Gerbig C, Wofsy SC, et al. (2003) A near-field tool for simulating the upstream influence of atmospheric observations: The Stochastic Time-Inverted Lagrangian Transport (STILT) model. Journal of Geophysical Research (Atmospheres). 108(D16):ACH 2-1

[24] MacEachren AM, Kraak MJ (1997) Exploratory cartographic vi-

sualization: advancing the agenda. Computers and Geosciences. 23(4):335-343
[25] Moreira W, Warnes GR (2009) RPy - A simple and efficient access to R from Python. http://rpy.sourceforge.net/ Accessed 29 June 2009
[26] Mountain D (2005) Visualizing, Querying and Summarizing Individual Spatio-Temporal Behavior. In: Exploring geovisualization. Kidlington, Oxford, UK: Elsevier.
[27] Nehrkorn T, Eluszkiewicz J, Wofsy S, et al. (2010) Coupled Weather Research and Forecast-Stochastic Time-Inverted Lagrangian Transport (WRF-STILT) Model. Meteorology and Atmospheric Physics, 107, 51-64.
[28] NOAA ESRL GMD (2009) Observation Sites >> Listing by Project. ESRL Global Monitoring Division. http://www.esrl.noaa.gov/gmd/dv/site/site_table.html Accessed 28 June 2009
[29] NOAA ESRL GMD (2009) NOAA ESRL GMD Tall Tower Network. ESRL Global Monitoring Division - Carbon Cycle Group. http://www.esrl.noaa.gov/gmd/ccgg/towers/ Accessed 26 June 2009
[30] Obukhov AM (1959) Description of Turbulence in Terms of Lagrangian Variables. Advances in Geophysics. 6:113
[31] Ohazama C (2008) Truly global. Google Lat Long Blog. http://google-latlong.blogspot.com/2008/02/truly-global.html Accessed 26 June 2009
[32] Olsen SC, Randerson JT (2004) Differences between surface and column atmospheric CO2 and implications for carbon cycle research. J. Geophys. Res. 109:D02301
[33] Open Geospatial Consortium, Inc. (2009) Sensor Web Enablement WG | OGC®. http://www.opengeospatial.org/projects/groups/sensorweb Accessed 29 July 2009
[34] Open Geospatial Consortium, Inc. (2008) OGC® Approves KML as Open Standard | OGC®. OGC Website. http://www.opengeospatial.org/pressroom/pressreleases/857 Accessed 26 June 2009
[35] Open Geospatial Consortium, Inc. (2006) Simple Feature Access - Part 2: SQL Option | OGC®. http://www.opengeospatial.org/standards/sfs Accessed 28 June 2009
[36] Open Geospatial Consortium, Inc. KML | OGC®. http://www.opengeospatial.org/standards/kml Accessed 29 July 2009
[37] Open Source Initiative (2007) Home | Open Source Initiative. Open Source Initiative website. http://www.opensource.org/ Accessed 26 June 2009
[38] PostgreSQL Global Development Group (2009) PostgreSQL: The world's most advanced open source database. http://www.postgresql.org/ Accessed 28 June 2009
[39] Potter CS, Randerson JT, Field CB, et al. (1993) Terrestrial Ecosystem Production: a Process Model Based on Global Satellite and Surface Data. Global Biogeochem. Cycles. 7:811-841
[40] Python Software Foundation (2009) Python Programming Language – Official Website. http://www.python.org/ Accessed 28 June 2009
[41] Refractions Research (2009) PostGIS : Home. http://postgis.refractions.net/ Accessed 28 June 2009
[42] Rey S (2009) Show me the code: spatial analysis and open source. Journal of Geographical Systems. 11(2):191-207
[43] Schlesinger W (1997) Biogeochemistry: An analysis of global change. San Diego: Academic Press
[44] Sellers PJ, Dickinson RE, Randall DA, et al. (1997) Modeling the Exchanges of Energy, Water, and Carbon Between Continents and the Atmosphere. Science. 275(5299):502-509
[45] Skamarock WC, Klemp JB, Dudhia J, et al. (2005) A Description of the Advanced Research WRF Version 2. Boulder, Colorado: National Center for Atmospheric Research Boulder, Colorado, Mesoscale and Microscale Meteorology Division
[46] Smith TM, Lakshmanan V (2005) Utilizing Google Earth as a GIS platform for weather applications. In: 22nd International Conference on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology. Atlanta, Georgia, USA: American Meteorological Society. http://ams.confex.com/ams/pdfpapers/104847.pdf
[47] Stull RB (1988) An introduction to boundary layer meteorology. Dordrecht, The Netherlands: Kluwer http://adsabs.harvard.edu/abs/1988aitb.book.....S Accessed 28 June 2009
[48] The Mathworks (2009) MATLAB - The Language Of Technical Computing. http://www.mathworks.com/products/matlab/ Accessed 29 June 2009
[49] Thomson DJ (1987) Criteria for the Selection of Stochastic Models of Particle Trajectories in Turbulent Flows. Journal of Fluid Mechanics Digital Archive. 180(-1):529-556
[50] Vogelezang D, Holtslag A (1996) Evaluation and model impacts of alternative boundary-layer height formulations. Boundary-Layer Meteorology. 81(3):245-269

Collaborative Web-Based Mapping of Real-Time Flight Simulator and Sensor Data

Rabih Dagher, Cristian Gadea, Bogdan Ionescu, Dan Ionescu and Robin Tropper

Abstract

Google Maps is an example of how Web 2.0 technology such as AJAX can be used to create online map services that are easy to access, user-friendly and fast. Thanks to flexible web-based mapping APIs, it is now possible for non-experts to plot and distribute GIS (Geographic Information System) data to a large audience. Most data plotted so far, however, has been relatively static. In addition, the typical webpage layout has limited the interaction possibilities for online maps when compared with windowed desktop applications. This paper will present a JEE-based publish/subscribe architecture that allows real-time sensor data to be displayed collaboratively on the web, requiring users to have nothing more than a web browser and Internet connectivity to gain access to that data. The architecture is tested using live data from Microsoft Flight Simulator and data conforming to the OGC Sensor Observation Service (SOS) standard. By using the latest web-based technology from open source projects like OpenLayers and 52North, this paper shows how maps and GIS data can be made more accessible, more social and generally more useful.

Introduction

With the growing adoption of social websites like Twitter, the demand for real-time data on the Internet is now higher than ever before. New web browser technologies have made it easy for users to access and publish dynamic data, such as what they are doing and where they are currently located. However, while there exist online services that promise "real-time" geographic data, a closer look will almost always reveal that the data is delayed or otherwise not presented in a useful way. In addition, collaborating with others in real-time on maps themselves containing real-time data has yet to be attempted from inside a web browser.

This paper introduces a new framework that allows displaying real-time sensor data within a collaborative web-based environment. The framework extends the publish/subscribe messaging model to meet the reliability and scalability demands of a social and collaborative online environment based on real-time data. While the framework can be adapted to a large variety of real-time streaming data sources, this paper will focus on live data provided by Microsoft Flight Simulator 2004 and data that conforms to the Sensor Observation Service (SOS) standard. SOS is one of the Sensor Web Enablement (SWE) standards of the Open Geospatial Consortium (OGC), an international standards organization with nearly 400 supporting companies and institutions (**?**).

The real-time data will be rendered by using the open source OpenLayers API and will be displayed inside an in-house platform for real-time web-based collaboration known as UC-IC ("you see I see"). UC-IC recreates a familiar desktop environment within the web browser, allowing maps to be organized within windows and manipulated in user-friendly ways. Additionally, the UC-IC platform was designed from the ground up to provide a social environment that enables the real-time transfer of applications and information to and from collaborators. This paper will show how the UC-IC platform allows for novel ways of interacting with maps containing real-time GIS data.

The rest of this paper is organized as follows. *Background* introduces the technology involved in displaying real-time data on the web and analyzes how the approach presented in this paper differs from existing solutions. *System Design* then discusses the unique client and server design of the proposed system. *Results* offers a look at two different deployments of the architecture: one with real-time data provided by running instances of Microsoft Flight Simulator, and the other using live SOS sensor data. Finally, *Conclusion* reflects on the contributions of this paper and proposes topics for future research.

Background

It was just over a decade ago that the Internet was limited to static content accessed using archaic web browsers. Despite these limitations, Geographic Information System (GIS) data quickly found its place on the Internet. Released in 1996 as a free service, MapQuest (**?**) made it possible for users to search for a location by name and to navigate the resulting map by using several buttons that surrounded the static map image (including buttons to select the zoom level). Users would see the entire webpage refresh for each navigation operation. Even with these inconveniences, there was something appealing about the interactivity and accessibility brought forth by the Internet that made it a good fit for geographic information.

As the Internet evolved to support more dynamic webpage content through the introduction of Rich Internet Application (RIA) technologies such as Asynchronous JavaScript and XML (AJAX), it came as no surprise that one of the first websites to make the most out of this new technology was a mapping site, namely Google Maps (?). Launched in 2004, Google Maps combined visually appealing maps with a very accessible user interface. It used AJAX to dynamically load sections of the map as the user dragged the map with the mouse cursor, a defining characteristic of what are now known as "slippy maps" (?).

While Google Maps offered many benefits (including that the service and developer tools were available at no charge), it also had some shortcomings. Its simplified interface, while pleasant to use, omitted support for the addition and selection of user-defined layers, such as layers based on the Web Map Service (WMS) standard defined by the OGC. The API available to developers required obtaining a key from Google, which was only functional on one domain and imposed numerous restrictions on how the map was to be used, including the following as described in the Google Maps Terms and Conditions:

> Except where you have been specifically licensed to do so by Google, you may not use Google Maps with any products, systems, or applications installed or otherwise connected to or in communication with vehicles, capable of vehicle navigation, positioning, dispatch, real time route guidance, fleet management or similar applications.(?)

Another limitation of the Google Maps API was that the API code itself could not be deployed on servers not owned by Google, meaning that developers had to rely on Google's uptime and availability. If developers wanted to demo their browser-based map application in an area without Internet access, they would be out of luck.

Several open source "slippy maps" emerged to address these drawbacks, with OpenLayers currently being the largest after absorbing many of the developers from the now-defunct MapBuilder project (?). OpenLayers has appeared in several academic papers (?)(?). There are no references to any "slippy map" being used for real-time data within a flexible collaborative environment, however.

Several papers have looked into the implications of real-time data delivery to a browser (?). In addition, online "WebOS" environments have existed for some time (?), although interaction with other users in these environments is usually limited to basic sharing of media, and they do not allow for full real-time collaboration of entire web-based applications and their data.

The publish/subscribe messaging model is a well known solution for real-time communication and is used in numerous examples, including as part of the OGC Sensor Alert Service (SAS) implementation of 52North. In this SAS architecture, both Publisher and Subscriber register themselves on the SAS server and communicate with each other using the open Extensible Messaging and Presence Protocol (XMPP) through a Multi-User Chat (MUC) channel (?). This paper, however, will present an approach that uses a registry service and other techniques to ensure that the architecture is scalable and robust enough for streaming data to a real-time social networking environment.

System Design

This section presents the client-side and server-side design that allows real-time GIS data to be delivered to a collaborative web-based environment.

Client-Side Design

The client-side user interface is a web-based implementation of the desktop metaphor, consisting of familiar windows, icons and menus that can be manipulated by using the mouse. The entire environment, however, is built on top of a collaborative platform that allows any window to be "sent" to another user. This is done simply by dragging a window onto an icon on the web-based desktop representing the friend who is to receive the application (and who must acknowledge a dialog to accept it). This sending process is unique in that the entire application logic, in addition to the current data within that application, is sent as part of the window. This is made possible by an advanced XML-based syntax and dynamic resource loading techniques (AJAX-Push) that go beyond the scope of this paper.

What is important to note, however, is that this concept of "sending" does not necessarily mean that the user doing the sending no longer retains the application. Rather, both users can have the same window open, and the inherent collaboration built into the system ensures that any actions performed within that application are automatically synchronized to the other user. For example, any text being typed into a text field will be communicated character-by-character to the other user's browser so that, as much as possible, both users always see the same application state. The environment was named "UC-IC" to highlight these collaborative characteristics.

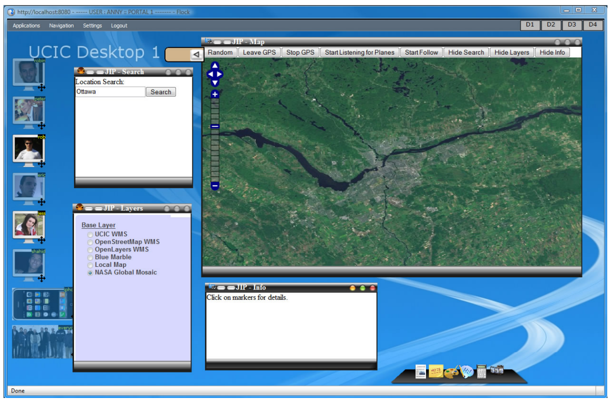


Figure 1: UC-IC environment with JIP Windows open.

Real-time collaboration on UC-IC is supported for applications programmed in, or able to communicate through, DHTML (AJAX). This includes Java Applets and Adobe Flash components, all of which can communicate through the DHTML-based UC-IC platform, offering a flexible environment for collaborative application development. Existing applications built on the UC-IC platform include a videoconferencing/chat application, a rich-text editor for live co-authoring, a collaborative video player, and a drawing application that can be reused for annotations on top of other applications.

The real-time nature of the platform makes it ideal for GIS applications dealing with sensor data. The GIS application implemented on the UC-IC platform was named Joint Intelligence Picture (JIP) and is shown in Figure 1. It consists of four different windows that communicate with each other through methods provided by the platform:

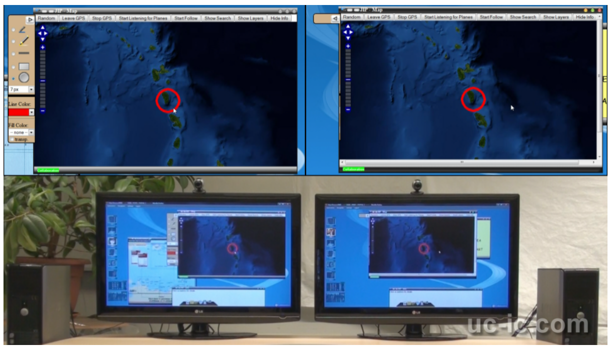


Figure 2: Two users collaborating on a JIP Map Window inside UC-IC.

JIP Map Window The Map Window contains an interactive "slippy map" generated by the OpenLayers API (?). It is the main window that appears when JIP is selected from the UC-IC applications list and it contains buttons for hiding and showing the other three windows that make up JIP. Inviting other users to collaborate on a JIP Map Window allows all of those users to see actions such as zooming, panning and drawing on the map. Figure 2 shows how a red circle drawn by one user appears on the collaborative Map Window of another user. A close-up of each screen is provided in the top half of the image. Other JIP Windows can also affect the contents of the Map Window; for example, using the Search Window can invoke the Map Window to show a specific location. In addition, markers may be displayed on the map based on real-time sensor data (for example, GPS coordinates of a moving vehicle).

JIP Search Window The Search Window contains a search box and list area where search results appear. Clicking on a search result updates the JIP Map Window. A custom geocoding solution provided by M3Data (?) is used to generate the results.

JIP Layers Window Part of the OpenLayers API, the Layers Window allows users to choose from a list of predefined map base layers. The list contains several custom map layers conforming to the OGC WMS standard and hosted on a local GeoServer (?) deployment, as well as free WMS layers from NASA (?) and OpenSteetMaps (?). In addition, the OpenLayers API supports loading layers from commercial services such as Google Maps, although these were avoided for reasons related to ease of deployment as mentioned in section *Background*. The Layers Window also contains checkboxes for showing and hiding the sensor data markers that are to appear on the map.

JIP Data Window The Data Window contains information based on user clicks in the Map Window. For example, a user can click on a barometric sensor marker on the map and see a real-time dial indicating barometric pressure in the Data Window. Like all windows in the UC-IC environment, the Data Window can be shared with other users on its own (for example, if a receiving user only needs to watch the live dial move and does not need the corresponding map).

Server-Side Design

In order to obtain real-time SOS data and display it within the JIP Map Window or Data Window, the web browser has to be in constant communication with the UC-IC server using AJAX. The Java Enterprise Edition (JEE) technology found in the open source JBoss Application Server (?) is used to make this possible. Of particular importance to the real-time architecture is JBoss' built-in Java Message Service (JMS) Server called JBoss Messaging.

JMS allows for asynchronous messaging based on the publish/subscribe messaging software pattern. The purpose of a JMS Server (sometimes also called a JMS Provider) is to route messages between JMS Clients, which can be either JMS Publishers or JMS Subscribers. JMS Publishers publish messages to a certain "topic" on the JMS Server, and JMS Subscribers subscribe to that topic to asynchronously receive the messages. The JBoss Application Server includes a JMS Server, while JMS Clients can be Java applications that use the JMS API.

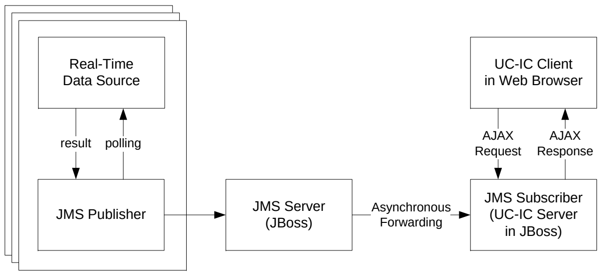


Figure 3: Basic real-time architecture using JMS.

A basic real-time architecture is shown in Figure 3. To initiate the transfer of real-time data, the client (the user in a web browser) must send an AJAX request that causes the UC-IC server (the JMS Subscriber) to subscribe to the real-time stream via the JMS Server. A JMS Publisher component is tasked with polling a real-time data source and receiving its response. The JMS Publisher then publishes the response to the JMS Server using a pre-arranged topic. The JMS Server then uses asynchronous messaging to forward the results to any JMS Subscribers who have subscribed to receive messages on that specific topic. Finally, the subscribed UC-IC Server sends the message to the client using AJAX-Push. The browser based client then parses the data and displays it within JIP.

This sequence of events can be seen in Figure 4. Note that, as was shown in Figure 3, multiple JMS Publishers can be sending real-time data to the JMS Server at the same time. Additionally, standard JMS methods exist for unsubscribing (based on an AJAX request from a user) and terminating the publishing process.

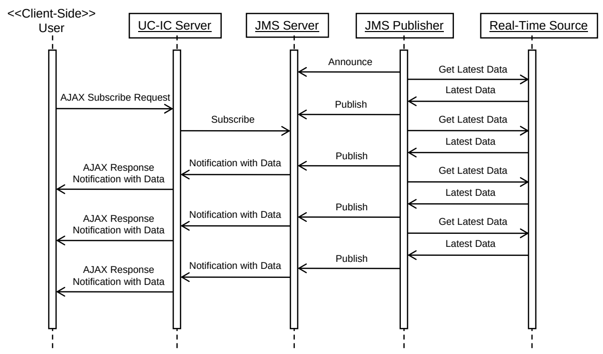


Figure 4: Event sequence for basic real-time data delivery to web browser.

While the above architecture would be sufficient for basic real-time data transmission, our final architecture features several enhancements that offer additional scalability and robustness required for real-time delivery to a social and collaborative platform like UC-IC. Having one centralized JMS Server, for example, is not ideal for real-time data delivery since the JMS Server could suffer an outage or become overloaded. A JNDI (Java Naming and Directory Interface) Registry Server is therefore introduced. The JNDI Registry Server stores a list of topics and the network address of the corresponding JMS Server where the topic is managed. JMS Publishers and Subscribers, which are pre-configured with the topics that they are to access, must first request the location of a JMS Server via the JNDI Registry Server. Once they obtain the location of the JMS Server, they can proceed as above.

By using a JNDI Registry Server, new JMS Servers can be added to the system with ease, allowing for much better scalability of the system as the number of topics increases. In addition, the support for redundancy of the system is increased since topics can be reassigned to different JMS Servers by changing the values stored by the JNDI Registry Server. By changing the location of the topics, JMS Clients can be dynamically assigned to the best JMS Server for optimal real-time data delivery.

The architecture robustness is further increased by keeping a list of alternate JNDI Registry Server locations in the JMS Servers and JMS Clients. The architecture can also support security through the use of SSL certificates, although security implications are beyond the scope of this paper.

Results

The design described in section *System Design* was applied to two different deployments: one with real-time data from Microsoft Flight Simulator, and the other using live sensor data based on the Sensor Observation Service (SOS) standard.

Flight Simulator

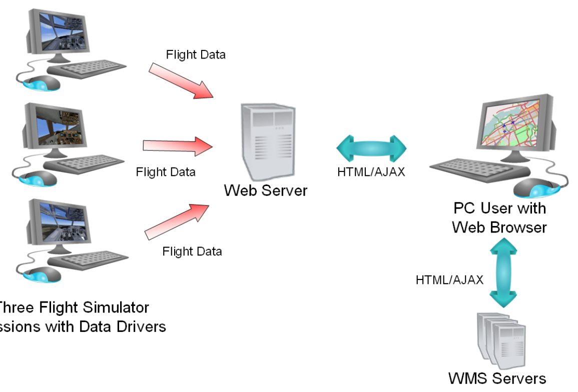
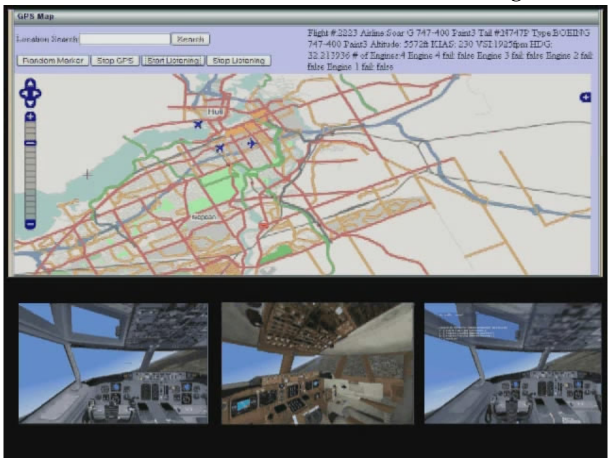


Figure 5: Flight Simulator test setup overview.

To test how the real-time architecture functions within a collaborative web-based environment, a controllable real-time data source was needed. Microsoft Flight Simulator 2004 was selected since the real-time aircraft data from inside the game could be accessed through a driver called FSUIPC (?). By setting up the JMS Publisher with the driver as the data source, XML data could be obtained for asynchronous sending to the UC-IC Server and display on the JIP client. A high-level view of this architecture is summarized in Figure 5.



Flight #2223 Airline:Soar G 747-400 Paint3 Tail #N747P Type:BOEING 747-400 Paint3 Altitude: 5572ft KIAS: 230 VSI:1925fpm HDG: 32.213936 # of Engines:4 Engine 4 fail: false Engine 3 fail: false Engine 2 fail: false Engine 1 fail: false

Figure 6: Real-time data from three different Microsoft Flight Simulator sessions displayed inside JIP.

The system was tested with three different Flight Simulator instances running on three different computers, each with their own JMS Publisher. This is shown in Figure 6. As the planes moved, a marker on the JIP client would move to show each plane's latest location. In addition, clicking a marker would populate the JIP Data Window with additional streaming information, such as the plane's current altitude, air speed and heading.

Although it was clear that a stream of data from the game was being received by the browser, the exact delay in the data was difficult to gauge. An additional feature was added that would change the plane marker color to red in the case of a simulated engine failure, which could be quickly activated from within Flight Simulator. The delay was observed to be less than two seconds when testing on a local network. The JIP Map Window was shared with five other UC-IC users, who could navigate and draw on the map while the planes were moving.

SOS Data

To test the system with more typical GIS sensor data, a standard SOS server was set up by hosting the open source 52North SOS Server component (?). The JMS Publisher component was tasked with polling the SOS Server using GetObservation requests. The JMS Publisher then received a response in the OGC Observation & Measurements (O&M) format. The JMS Publisher component then sent the O&M response to the JMS Server component, which would asynchronously forward it to the UC-IC Server, and the UC-IC Server would send it for parsing and display on the client using AJAX. In this case, an open source library called JFreeChart was used to display the data as a real-time dial (?).

Again, the real-time performance was very good with almost no noticeable delay, and the collaborative nature of the system made it easy to monitor and annotate the data as a group.

Conclusion

This paper introduced an architecture used to deliver real-time Flight Simulator and SOS sensor data to a social and collaborative UC-IC environment accessible from within a web browser. This was accomplished by using open source technologies, including OpenLayers to display the real-time data and JBoss to make asynchronous communication possible. The system was tested by using real-time data from Microsoft Flight Simulator and a locally-hosted SOS server using open source components from 52North. In both cases, the real-time data streamed smoothly to the JIP client application and was available for real-time collaboration with other users of UC-IC.

Future work will attempt to address limitations of JavaScript memory management techniques used by browsers, which can cause performance issues if the real-time data is left streaming for long periods of time. The SOS standard, for example, offers a less-verbose GetResult request which is more ideal for real-time data and would allow for more complex sets of real-time data to be delivered to the client. Although experiments with mobile devices have already been undertaken, the great breadth of mobile web browsers has given inconsistent results when attempting to load the DHTML-based UC-IC environment, and are worth exploring further. Finally, other standards such as the Sensor Alert Service could be implemented to communicate scenarios like the plane engine failure.

Rabih Dagher, Cristian Gadea, Bogdan Ionescu, Dan Ionescu and Robin Tropper,
NCCT Laboratory
University of Ottawa
161 Louis Pasteur Room B-306
Ottawa ON K1N-6N5
CANADA
rdagher AT ncct.uottawa.ca
cgadea AT ncct.uottawa.ca
bogdan AT ncct.uottawa.ca
dan AT ncct.uottawa.ca
rtropper AT ncct.uottawa.ca

A Modular Spatial Modeling Environment for GIS

Brian Marchionni & Daniel Ames, Idaho State University

Abstract

Development of an open source modeling environment for use with spatial-temporal data in a Geographic Information System (GIS) is presented. MapWindow GIS, a free and open source desktop GIS, has been used extensively in watershed modeling and is the underlying engine of the U.S. EPA BASINS system. To date, legacy versions of MapWindow have lacked an integrated modeling environment suitable for linking together geospatial and temporal independent processes at a granular level. Development efforts focused on creating an extensible graphical, open source modeling environment with easy to use programming objects.

This development was made possible due to the new design of the MapWindow GIS 6 project. This new modeling environment allows users and developers to easily create models which can take advantage of spatial and temporal data objects and analytical tools. The design approach involves the extensive use of interfaces, which are essentially skeleton programming tools that detail how an object programmatically interacts with other objects, but not necessarily how it works internally. By using interfaces, the new MapWindow GIS modeler makes it relatively simple to take existing modeling processes, wrap them in an appropriate interface, and execute them as part of a more complex model.

The central underlying design consideration of the newest version of MapWindow GIS was to keep the entire project as modular as possible, this has been extended to encapsulate the development efforts of the modeler as well. The new modeling environment allows developers to automatically generate user interfaces for their processes. Because all tools in the MapWindow modeler must implement the same interface, developers wishing to use a tool directly in their own application need not add the graphical modeler if they do not so desire.

MapWindow GIS 6 and the modeler are entirely developed using the Microsoft .NET Framework which allows it to be run on a variety of operating systems including Windows, Linux or OS X (via the Mono compiler).

Introduction

The goal of the project described here was to create a cutting edge open source modeling environment that worked within the Microsoft .Net framework and contained both a graphical user interface and easy to use programming object. The project was designed to run along side the next generation of the MapWindow project, MapWindow GIS 6. MapWindow GIS 4 is the current version of the project and is under continued development at Idaho State University in the Department of Geosciences. MapWindow GIS 5 was a short-lived prototype project that was never released publicly.

Originally developed at Utah State University, and now maintained primarily at Idaho State University with an international development team, MapWindow GIS is a free and open source software project that is downloaded over 6000 times per month. It has an active community of users and developers on the MapWindow.org web site. The community collaborates on making updates and introducing new features.

The existing project is divided into two components: the MapWindow GIS desktop application, and the ActiveX map control. These two components work together to form the entire project. This modularity allows the ActiveX control to be used in other stand alone applications as well as within the main MapWindow GIS desktop application. (1)

The need to develop a modeling environment arose from other developments in the GIS community. A general need to simplify the task of using spatial and temporal processes had been brought forward by many MapWindow users and by several other communities who are using the MapWindow components in their own projects. Furthermore, the use of modeling in a GIS environment has been suggested by several other researchers including Xie and Brown in their 2007 paper noting, 'simulation in spatial analysis and modeling has been one of the key approaches of many researchers of GeoComputation' (9).

The modeler requirements are closely tied to other developments in the MapWindow 6 project and include:

- all user interfaces need to be as simple to use and as well documented as possible;
- users should need no programming experience to use the software;
- the software must have high portability: software should work on many different systems including MS Windows, Linux and Macintosh OS X;
- the code needs to be highly extensible and reusable;
- the code should be easy to maintain for new developers.

The modeling environment should also be designed such that it can be integrated into other applications with minimal dependency on external libraries, including the MapWindow library itself. Since the design of MapWindow GIS 6 was well underway at the time of the modeling environment's conception, it was decided that components and data types from this new architecture would be used because of the advantages that it afforded, including being memory managed and highly extensible. The data types are not directly link to their sources, which allows many different data formats to be present as a single data type.

MapWindow GIS 6

MapWindow GIS 6 is the next generation of the MapWindow open source project. Early in the planning stages of MapWindow GIS 6 it became apparent that the technology behind the original MapWindow ActiveX map component would not be capable of meeting all of the new project's requirements. Specifically since the original code was written as a Microsoft COM object it could never be cross platform compatible. For this reason it was decided that a complete rewrite of the map component would be required.

The design of the new architecture focused on an extremely modular system using class interfaces. Figure 1 highlights the interface architecture of MapWindow GIS 6. This design allows for any single component to be replaced by another component that uses the same interface. This design stemmed from the successful plug-ins methodology from the original MapWindow GIS 4 that allowed third party developers to extend the functionality of the application by writing their own class which implements the plug-in class interface (2). The improvement presented in MapWindow GIS 6 is that this interface based architecture is extended to every modular component of the architecture and not just to plug-ins.

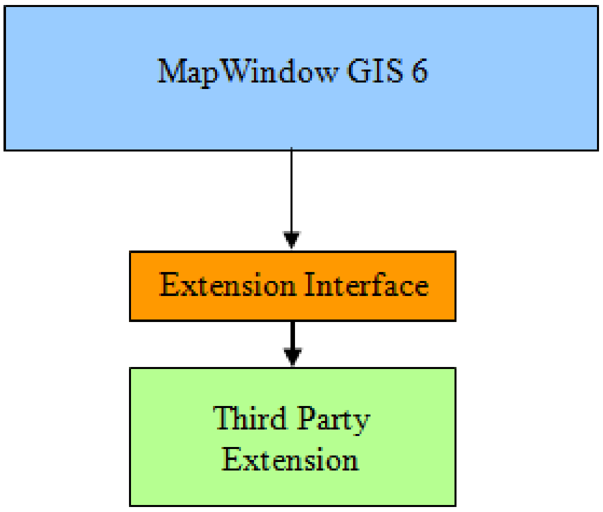


Figure 1: MapWindow GIS 6 Architecture

The MapWindow Modeler Project

Project Requirements

The MapWindow Modeler environment has been developed specifically to meet the requirements of several use cases identified by the United States Environmental Protection Agency (EPA) Data for Environmental Modeling (D4EM) project. Some of the key requirements and constraints of the system are defined as follow:

- the tool should be written in Microsoft .NET so that it can be ported to Windows Mobile and Mono for Linux;
- the available tools and available data types should be extensible;
- the tool should integrate both spatial and temporal components;
- the tool should be easy to use for end users;
- the tool should be compatible with existing versions of MapWindow GIS;
- the tool should be robust and easy for new developers to add to and enhance.

Several existing open source projects were identified to see if they could meet the requirements for a modeling tool for MapWindow. Sextante (7) meets some of the requirements, however it lacks temporal data type support and is written in the Java language and hence would not meet the requirement of being written in Microsoft .NET. No other open source modeling products that were available were written in Microsoft .NET and could handle both spatial and temporal data interaction. The OpenMI system (4) was examined, but it too failed to meet all of the requirements of the system as its scope was well beyond a simple graphical tool for linking spatial and temporal processes, but rather is designed to link larger complex models.

Use Cases

There are three primary use cases for the modeling environment. The first covers the modelers' use while integrated into MapWindow GIS 6. In this mode a standard extension to the MapWindow GIS 6 desktop application will include the modeling environment. These two environments are tightly linked to allow data from the MapWindow GIS 6 map component to be added seamlessly from the modeler, and conversely they allow data from the map to be used in models. The second use case involves integration with the legacy code of MapWindow GIS 4. This is similar to integration with version 6 of MapWindow, but only a specific subset of the data will be made available to the MapWindow GIS 4 application because of format compatibility issues. The final use case covers using the modeler as a stand alone component for use in third party applications. Since all possible uses of the modeler in other applications cannot be considered, the modeler must be as versatile and customizable as possible.

Software Design Technique

Since many different users and developers will be working with the system, the initial design specifications needed to be well defined at the outset. Once this initial design was completed a small group of developers created a set of simple tools to test the general design. It was at this stage that critical modifications were made to the design to address specific problems that developers and users were facing.

The rapid prototyping development technique allowed for feedback from testers and other developers while ensuring a quick time to deployment. Initial development efforts took only six months. Once this critical initial development stage was completed the second phase continued until such time as all parties involved were satisfied with the resulting architecture. Once completed most major design considerations were done and the overall architecture finalized. While changes can still be made at this point they must take into account the existence of other dependant components that need to be integrated and cannot have their functionality impaired. For example, if a new version of an interface is created once this second stage of development is completed, it must ensure that any components using the already existing interface must continue to function seamlessly.

This second stage of development is potentially the most important. Feedback from developers creating tools that will ensure the ease of use of the interface for new developers wishing to create tools or data types for the system.

Software Design

Modeler Design

The MapWindow modeler is composed of two interrelated parts: the ToolManager and the Modeler. The ToolManager lists all of the available tools to the user while also providing access to tools in the Modeler. The Modeler displays, loads, saves, and executes models in a graphical environment.

The Modeler and ToolManager itself are actually .NET form components. Like other programming objects in the .NET environment they have a graphical representation that allows programmers to drag and drop it onto a form without writing any code. This greatly reduces the time needed for programmers to develop an application that uses the modeler. Figure 2 shows the class diagram on the ToolManager and Modeler. Many of the classes are interdependent and used by both components. Figure 3 shows an instance of the ToolManager on the left running within MapWindow 6 displaying the tools it has found, and the Modeler on the right displaying a simple model containing one tool.

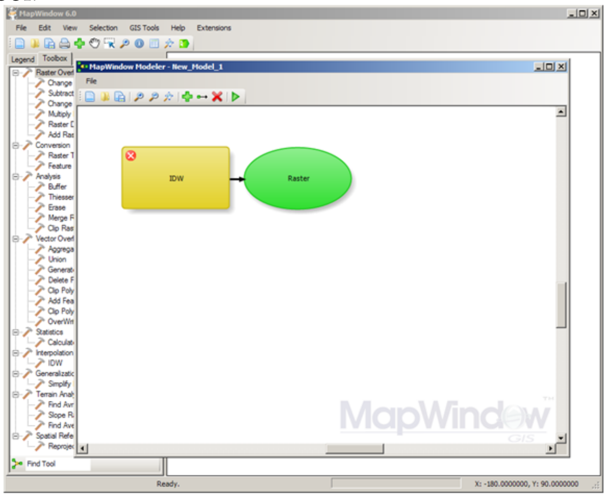

MapWindow 6.0
File Edit View Selection GIS Tools Help Extensions
Legend Toolbox
MapWindow Modeler - New_Model_1
File
IDW
Raster
MapWindow
Find Tool
Ready.
X: -180.0000000, Y: 90.0000000

Figure 3: The ToolManager and Modeler running with MapWindow 6 on Microsoft Windows

Building for Extensibility

Since the use cases for the modeler cover many different applications it was imperative that the modeler be designed such that it can be extended in several different ways, such as:

- tool definitions;
- parameter definitions;
- user interface representation of parameter definitions.

To allow each of these areas to be expanded upon, several programming concepts needed to be employed. These concepts are widely used through the architecture of MapWindow GIS 6 so programmers familiar with this environment can more easily add functionality to the modeler.

To accomplish this, a class interface was defined for tool definitions and parameter definitions called ITool and IParameter, respectively. Using interfaces, blank class templates which programmers can populate with functions (5), allows developers to rapidly develop software which implements the needed operations of software they are interacting with (3). 'A well-recognized method for reducing program complexity involves structuring the model as a set of distinct modules with well-defined interfaces' (6).

Since tools and parameter types can be generated in a variety of different ways, the Modeler never loads ITools or IParameters from disk directly; rather, it relies on a ToolManager to handle loading, and instantiating tools, and parameter types as needed. The ToolManager loads tools by scanning specified folders for assemblies that implement the IToolProvider interface. Once a class that implements this interface is found, it is instantiated and queried for a list of the tools it

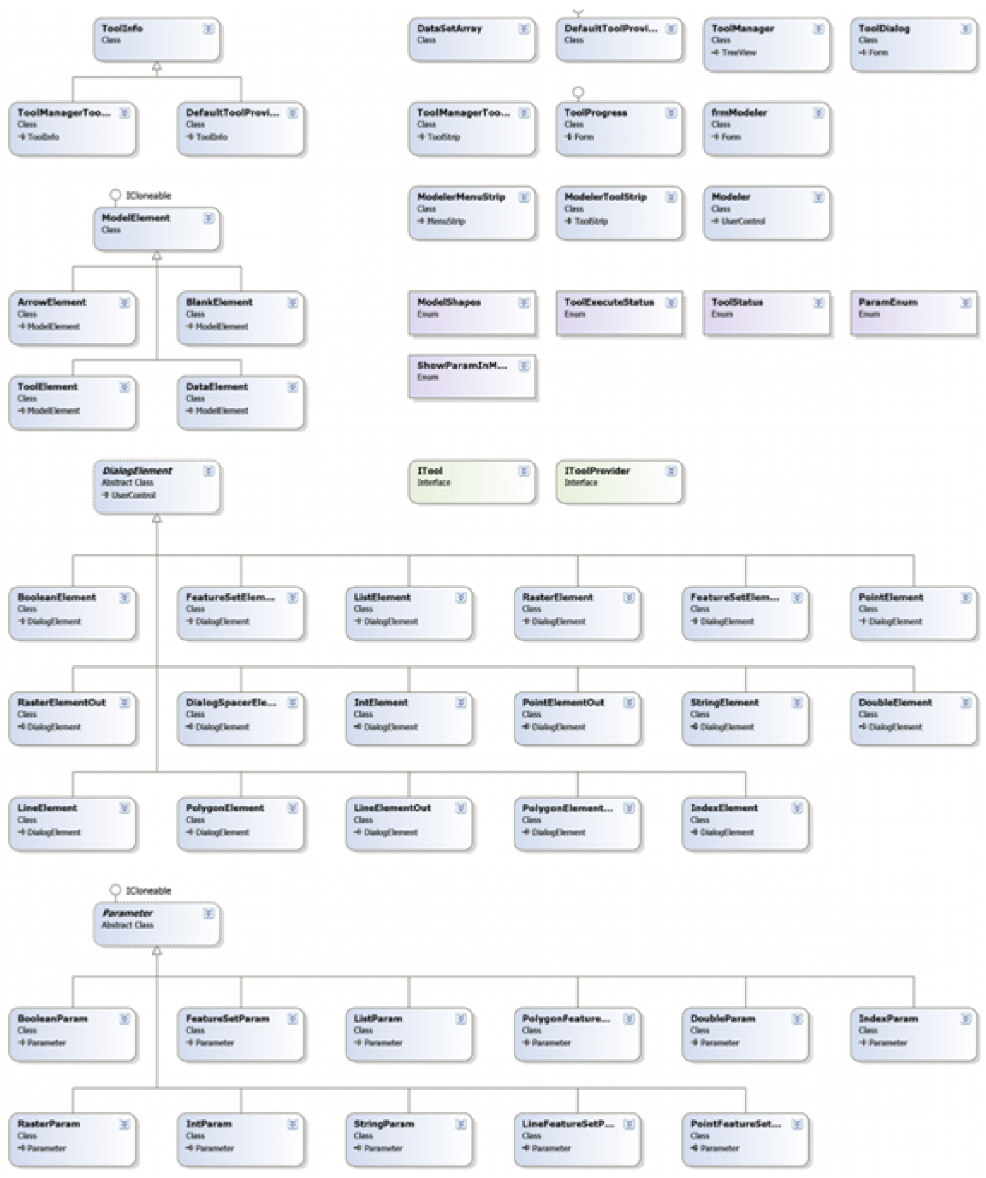


Figure 2: The ToolManager and Modeler class diagram

is capable of providing. This allows tool providers to create tools from a wide variety of sources. A default tool provider is included in the ToolManager. This tool provider scans specified folders for assemblies which implement the ITool interface directly. Loading of parameter definitions and their user interface is also done the same way with the ToolManager looking for assemblies that implement the IParameterProvider interface, and a default provider which scans for IParameter implementing assemblies.

The ITool and IToolProvider Interfaces

The goal of the ITool interface is to remove the burden of creating a user interface and maintaining tool interoperability from the tool developer. Developers designing tools need only implement the ITool interface when designing their tool, and the ToolManager generates the graphical user interface automatically for them when the tool is instantiated. The ITool interface contains several properties which are read by the ToolManager when the tool is first detected and instantiated.

The Name, UniqueName, Category and Version properties on the ITool interface are used by the ToolManager to identify the tool and are require for a tool to be loaded, if any of these fields are missing the tool will not be added to the toolbox. HelpText, HelpImage, HelpURL, Author, Icon and ToolTip are used to populate related parts of the graphical user interface on behalf on the developer. These are optional and ignored if they return null.

The InputParameters and OutputParameters properties return arrays of type IParameter, which are used by the ToolManager and Modeler to execute tools, and populate user interface dialogs with appropriate graphical components. When a tool is instantiated for execution, the initialize method is called first, allowing the tool to populate its InputParameters array with blank parameters. Then, as the user modifies inputs in the array, the ParameterChanged method is called allowing tool developers to create inputs which are dependent on changes to other inputs. Finally when the tool is ready to be run the Execute method is called. The result of this execution is stored in the OutputParameters array and can be transmitted to the next tool by reference or saved to disk as needed. Figure 4 displays the methods and properties of the ITool interface.

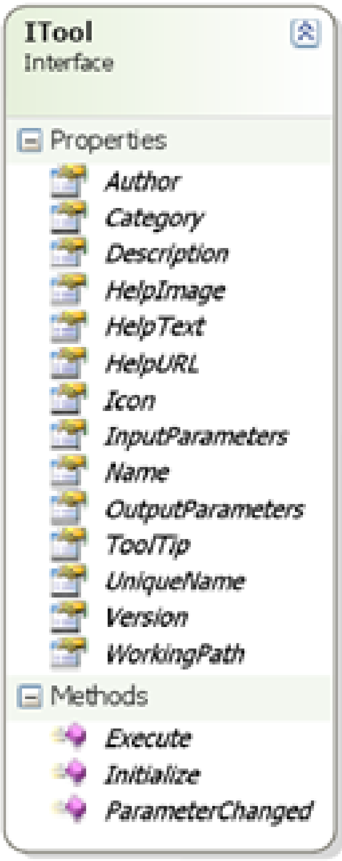
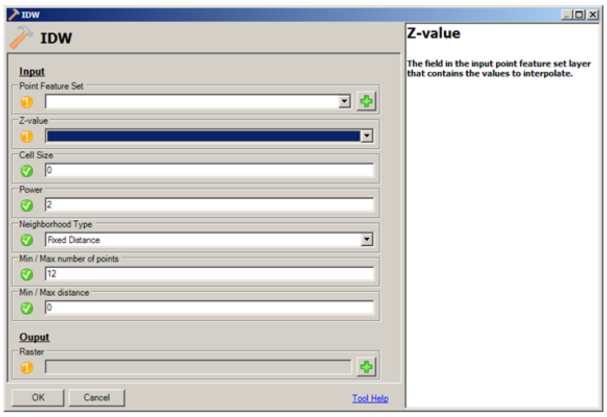


Figure 4: The methods and properties of the ITool interface

Figure 5 illustrates the form that is automatically generated when the Inverse Distance Weighting tool is created. Note the help text on the right is automatically displayed when the user highlights a particular input parameter. Status lights on the left side of the parameter field display the parameters' validity. The ITool interface contains all the information necessary for running and displaying a tool.

The IToolProvider interface allows tools to be generated in a wide variety of ways. While the default ToolProvider searches folders for assemblies that contain ITools, there are many other ways that tools could be generated. For example, a ToolProvider could connect to a Web Processing Service, get a list of available tools, and then generate a corresponding set of ITools which would then be in charge of instantiating for the ToolManager.


IDW

Input
Point Feature Set
Z-value
Cell Size 0
Power 2
Neighborhood Type Fixed Distance
Min / Max number of points 12
Min / Max distance 0

Ouput
Raster

Z-value

The field in the input point feature set layer that contains the values to interpolate.

Figure 5: The Inverse Distance Weighting tool dialog running in Window

Tools can also be generated by the Modeler. This is done by saving the model which includes several tools to a XML file which are then recognized by ToolManager as a stand alone Tool. These tools, when called to be executed by the ToolManager will create a new instance of the Modeler, load the saved model and execute it seamlessly as if it were a single tool. This new tool will run as long as each of the Tools used to create the model are available to the ToolManager at execution time.

The IParameter Interfaces

Parameters are the input and output of a tool and need to be defined so that they can have an appropriate visual representation. For example, a numerical parameter should allow for a minimum and maximum value to be specified in order to limit the users' input to a certain range. It should also be capable of specifying a default value and be represented on the tool dialog by a text box that will only accept numerical values. This can be accomplished by creating a parameter object that specifies these constraints and contains a control object that represents how the parameter should be represented in the tool dialog.

The IParameter interface consists of several properties which are used by the ToolManager and Modeler to identify the parameter type. The DefaultSpecified property is used to determine if a tool developer has specified a value to be used as default. The HelpImage and HelpText properties are used to populate the help area on the right hand side of the automatically generated tool dialog. ModelName is used by the modeler to store a unique identifier for a particular instance of a tool. Name is used to identify the parameter input in the tooldialog. The ParamType property returns a string which is used to identify the parameter's type and determine if it is compatible with another tool's input. ParamVisible is used on input parameters to determine if they will be graphically displayed in the Modeler. Finally the Value property contains the actual parameters value.

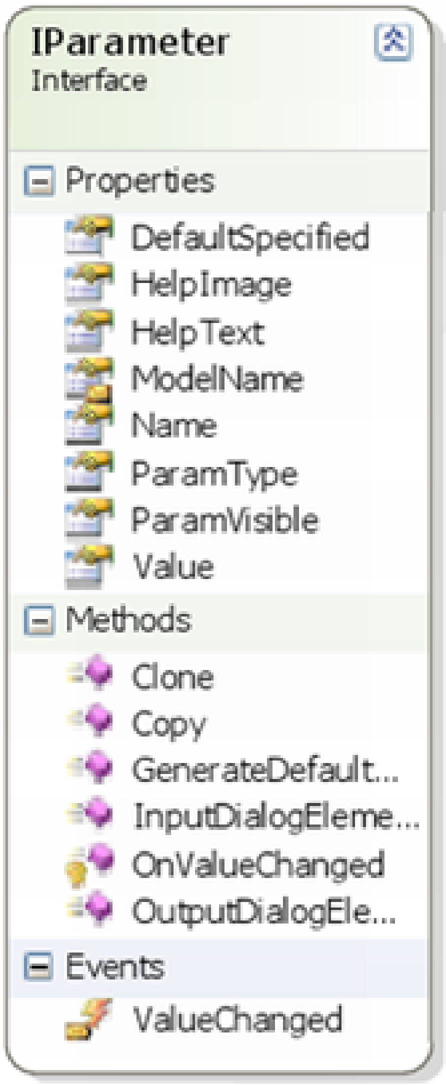


Figure 6: The properties, methods and events of the IParameter interface

The clone and copy methods are inherited from the ICloneable class and are used for creating temporary instances of the Parameter when editing values so that if the user cancels without saving his changes they will not affect the underlying objects. GenerateDefaultOutput populates a parameter with a basic value such that the model can be run. InputDialogElement and OutputDialogElement return an instance of the appropriate graphical representation of the parameter for inputs and outputs respectively. OnValueChanged is called when then parameters value have been modified and fires the ValueChanged event, which is then used to notify the IParameter's parent ITool that its value has been modified by calling the ParameterChanged method. Figure 6 displays the IParameter Interface including its properties, methods and events.

Figure 7: The IParameter base element as it appears in the Microsoft Visual Studio designer. The base component is never seen in the modeler

Figure 8: The List Parameter input element as it appears in the Microsoft Visual Studio designer

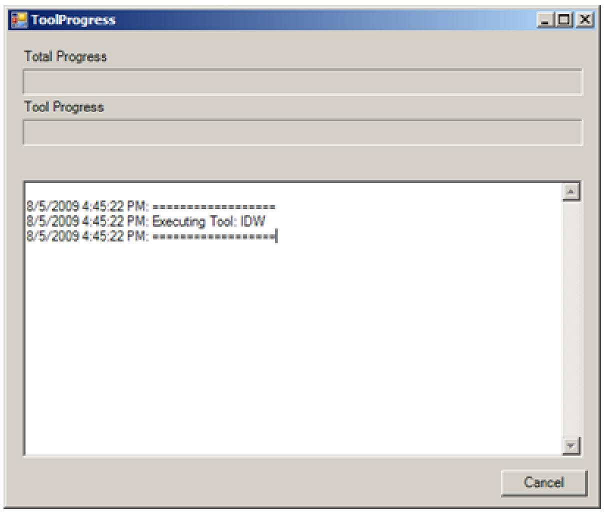
Figure 7 displays the IParameter base graphical user interface, which all parameters must return when the InputDialogElement or OutputDialogElement are called. Figure 8 displays the List Parameter component which implements the IParameter interface. IParameters are responsible for generating two graphical components, one for input and one for output parameter configurations. This ensures that parameters act differently for inputs and outputs.

Tool and Model Execution

There are two different ways that a tool can be executed, either from the ToolManager or from the Modeler. To execute tools from the ToolManager, a user double clicks the tool's name to create a tool dialog populated with the tool's inputs and outputs. Next they populate these fields and then press the ok button. The ToolManager, then calls the Execute method on the tool and displays a progress dialog box.

The modeler executes tools in a similar way. The user drags and drops tools from the ToolManager into the Modeler and links them together by dragging link lines. Internally the Modeler creates association between the relevant input and output parameters. The user can then modify a tool's parameters by double clicking on the tool's graphical representation to access the corresponding tool dialog. Once the model is configured the user clicks the Modeler's execute button which begins the model's execution.

Once a tool is called to be executed, either from the ToolManager or integrated into a model, a background thread is started to carry out the tool's execution. In the modeler, tools that are not ready to be executed because they depend on other tools are queued while tools that are ready to be executed are assigned to a thread and executed. Queued tools are then reviewed as executing tools complete. A separated thread is used to ensure that the tool progress dialog remains responsive to user activity. Messages from the background thread are relayed to the foreground progress dialog thread to allow progress indicators to be updated by the tool. Figure 9 displays the progress indicator form running a tool. In the event of user cancellation, tools are responsible for cleanly exiting.



```
8/5/2009 4:45:22 PM: ===================
8/5/2009 4:45:22 PM: Executing Tool: IDW
8/5/2009 4:45:22 PM: ===================
```

Figure 9: Progress indicator dialog

Modeler Architecture Overview

The Modeler and ToolManager are intentionaly modular, and any single component can be replaced with another, which satisfies the interface requirements of that component. Not all components are necessary for the entire environment to work. If, for example, the Modeler was not included in a project because it was not needed, the ToolManager could be included by itself as a visual component or as a instantiated object invisible to the end user. This high level of interchangability ensures that the components of the modeling environment can be used to meet the widest ranges of developer needs.

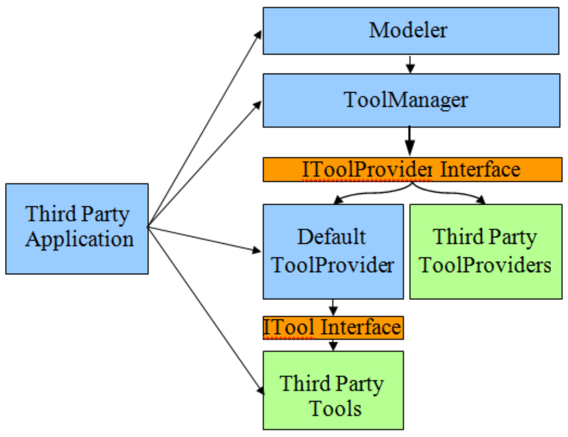


Figure 10: MapWindow Modeler Architecture

Figure 10 displays the overall architechture of the entire MapWindow Modeling project. At the highest level is the Modeler which can be used graphically and programmatically to load, link and execute tools. The Modeler requests instances of tools from the ToolManager which is capable of creating instances of tools and passing them to the Modeler or executing them directly. The ToolManager can load tools using the IToolProvider interface from the Default ToolProvider or from Third Party ToolProviders. The Default ToolProvider can load tools from assemblies which implement the ITool interface directly. Finally third party applications can use any of the MapWindow Modeler components in their own code or they can link directly to Tool or ToolProviders assemblies.

Modeling Environment Comparison and Case Study

The MapWindow Modeler user interface works much like the user interface of the ArcGIS ModelBuilder and gvSIG Sextante Modeler. All three environments present a graphical user interface that allows the user to use the mouse to drag model components from a list into the modeling area where it is represented by a square or circle. In all three environments double clicking on a model element such as a tool or data item, opens a dialog box with options relating to the element. The forms have slightly different appearance but the same functionality. All three environments have help text available to guide the user when configuring a model's elements.

A common task that is often performed when dealing with flood data sets is the delineation of watersheds from raster elevation data. One of the first steps of this process is the generation of raster stream data (8). The initial digital elevation model (DEM) data often contains artifacts referred to as pits; these pits can interfere with the stream delineation process and are usually eliminated by using a pit filling algorithm.

Next the flow direction of the elevation data is calculated. Flow direction calculates the direction in which a drop of water entering a cell would flow. A flow accumulation layer is created next, that calculates the total number of cells which flow into every cell of a raster. Finally the flow accumulation data is reclassified into a Boolean mask where 1 represents cells that have sufficient flow accumulation to be considered a stream and 0 for cells that do not.

This model was created in the ESRI ArcGIS ModelBuilder, the gvSIG Sextante Modeler and in the MapWindow Modeler. Figure 11 illustrates the model as it appears in the ArcGIS ModelBuilder modeling environment (top), in the Sextante Modeler (middle) and in the MapWindow GIS Modeler (bottom). Note that because of differences in the way the processes work internally, they may produce different outputs. Also note that in some cases different tools had to be used because the exact same tools do not exist in all of the environments. For example the Reclassify tool was used in the ArcGIS model and Sextante model while the raster threshold tool was used in the MapWindow model. In some cases a single tool in one environment is capable of producing the same result as two or more tools in another, such as the Flow Accumulation tool in the Sextante tool which produces the same result as the D8 and Flow path tools in the MapWindow model. Also note that the Sextante model does not show the intermediate or output data in its model only the input data and model processes.

Figure 12 shows the final delineated stream data as created by the ArcGIS ModelBuilder and displayed in the ArcMap (top), as created by the Sextante Modeler and displayed in gvSIG (middle) and as created by the MapWindow Modeler and displayed in MapWindow GIS (bottom). Note that the DEM appears differently in MapWindow GIS as its default elevation symbolizer uses a hill-shading technique.

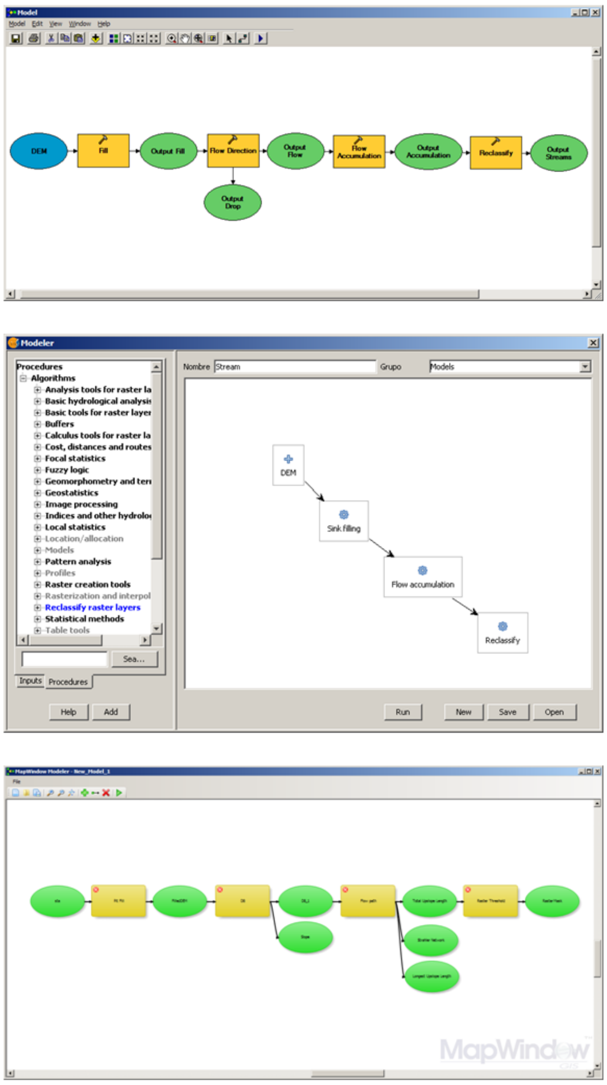


Figure 11: The stream delineation model as it appears in: (top) the ArcGIS ModelBuilder, (middle) the Sextante Modeler, and (bottom) the MapWindow Modeler

Figure 13 shows the results of the three models when overlaid together above the original DEM, note that only small differences exist between the three models, while the general trend of the stream is preserved between them. These slight discrepancies in the various models' results are to be expected given that they each implement slightly different algorithms to achieve their result. The run time of the ArcGIS Modeler was 14 minutes 35 seconds, the run time of the Sextante Modeler was 24 minutes 9 seconds while the run time of the MapWindow Modeler was 22 minutes 33 seconds. The process which took the longest time in all three cases was the pit fill algorithm. It is also responsible for the large discrepancies in times between the three models.

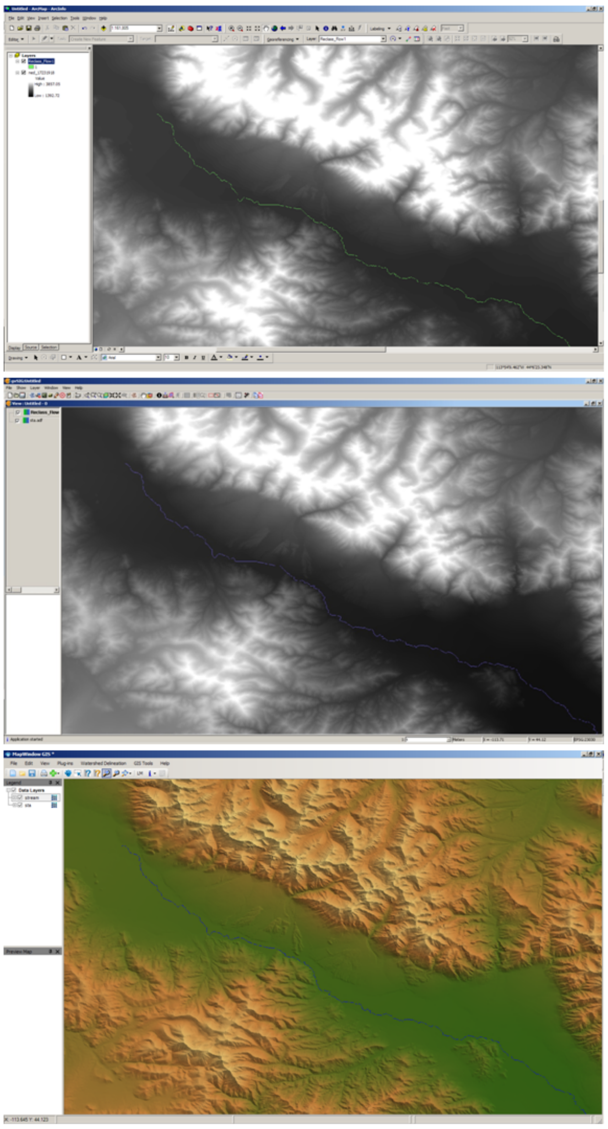


Figure 12: Final stream delineation and original DEM as display in: (top) ArcMAP and produced with ArcGIS ModelBuilder, (middle) gvSIG and produced with Sextante Modeler, (bottom) MapWindow GIS and produced with MapWindow Modeler

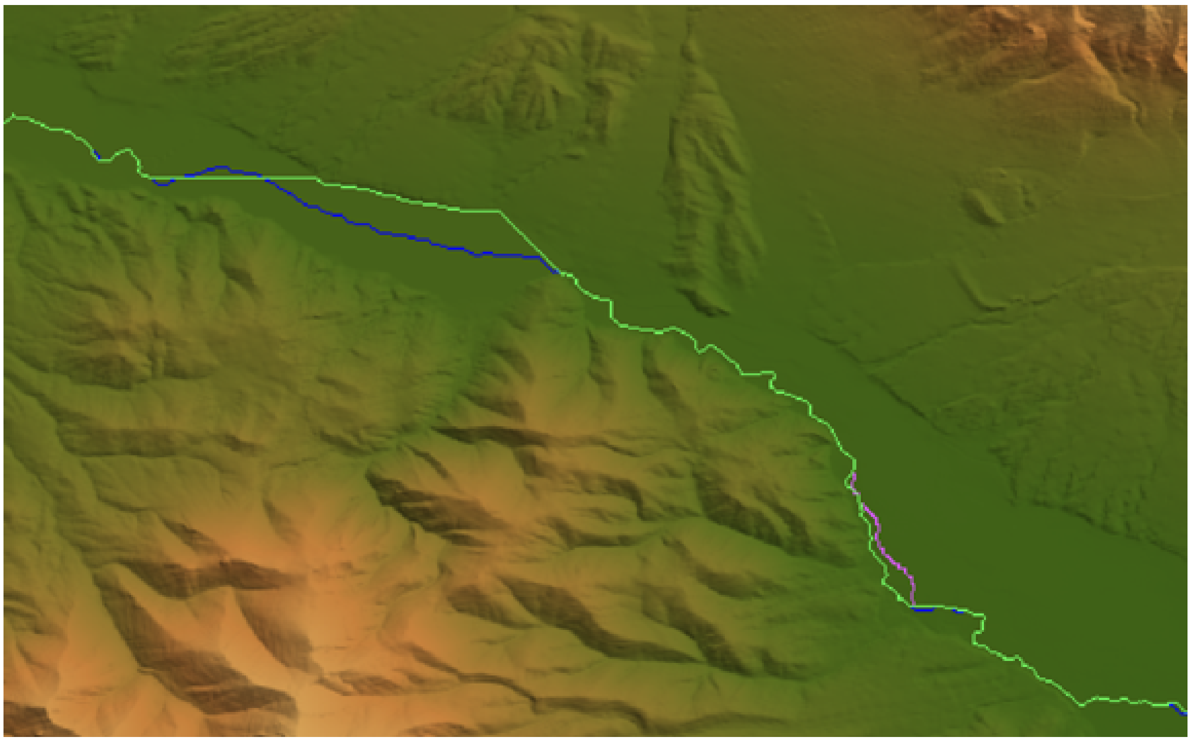


Figure 13: The results of the three models overlaid above the original. Green is the top layer and is the result of the ArcGIS ModelBuilder, Purple the result of the Sextante Modeler and blue the result of the MapWindow Modeler.

Discussion and Conclusions

The MapWindow GIS Modeler is a versatile modeling environment, which can handle many different data types. It executes models in similar time to other GIS modeling environments and faster than other open source ones. Due to its modular and extensible architecture it can use tools of many different designs. The design flexibility not only allows tools to function in a wide variety of different ways, but it allows tools and their associated parameters to be generated from any number of sources. Its ease of use for end users and developers, as well as its integration with MapWindow GIS 6 and MapWindow GIS 4, ensures that the widest range of users will have access to the program. By building on the successful design of previous generations of MapWindow GIS, the MapWindow Modeler benefits from all of the development expertise, keeping the designs that were the most effective while eliminating some of the more constrictive problems. It is one more tool available to both developers looking to create new modeling tools and end users wishing to create models with such tools.

Brian Marchionni & Daniel Ames,
Idaho State University
marcbria@isu.edu
dan.ames@isu.edu

Bibliography

[1] D. Ames. *MapWinGIS Reference Manual: A function guide for the free MapWindow GIS ActiveX map component*. Lulu.com, Morrisville, North Carolina, 2007.

[2] D. Ames, C. Michaelis, and T. Dunsford. Introducing the mapwindow gis project. *The Journal of the Open Source Geospatial Foundation*, 2:13–16, 2007.

[3] S. Greenberg. Toolkits and interface creativity. *Multimedia Tools and Applications*, 32(2):139–159, 2007.

[4] J. B. Gregersen, P. J. A. Gijsbers, and S. J. P. Westen. Openmi: Open modelling interface. *Journal of Hydroinformatics*, 9(3):175–191, 2007.

[5] L. Liquori and A. Spiwack. Extending feathertrait java with interfaces. *Theoretical Computer Science*, 398(1-3):243–260, 2008.

[6] T. Maxwell. A paris-model approach to modular simulation. *Environmental Modelling & Software*, 14(6):511–517, 1999.

[7] V. Olaya and J. C. Gimenez. *SEXTANTE: a gvSIG-based platform for geographical analysis*. Free and Open Source Software for Geospatial, Victoria, Canada, 2007.

[8] K. L. Verdin and J. P. Verdin. A topological system for delineation and codification of the earth's river basins. *Journal of Hydrology*, 218(1-2):1–12, 1999.

[9] Y. Xie and D. G. Brown. Simulation in spatial analysis and modeling. *Computers, Environment and Urban Systems*, 31(3):229–231, 2007.

Imprint

Editor in Chief:
Tyler Mitchell - tmitchell AT osgeo.org

Assistant Editor:
Landon Blake

Section Editors & Review Team:
Eli Adam
Daniel Ames
Dr. Franz-Josef Behr
Jason Fournier
Dimitris Kotzinos
Scott Mitchell
Barry Rowlingson
Jorge Sanz
Micha Silver
Dr. Rafal Wawer
Zachary Woolard

Acknowledgements
Daniel Holt, LaTeX magic & layout support
Various reviewers & writers

The *OSGeo Journal* is a publication of the *OSGeo Foundation*. The base of this journal, the LaTeX 2_ε style source has been kindly provided by the GRASS and R News editorial boards.

Journal online: http://www.osgeo.org/journal

OSGeo Homepage: http://www.osgeo.org

Mail contact through OSGeo, PO Box 4844, Williams Lake, British Columbia, Canada, V2G 2V8

ISSN 1994-1897

Imprint

Editor in Chief:
Tyler Mitchell - tmitchell AT osgeo.org

Assistant Editor:
Landon Blake

Section Editors & Review Team:
Eli Adam
Daniel Ames
Dr. Franz-Josef Behr
Jason Fournier
Dimitris Kotzinos
Scott Mitchell
Barry Rowlingson
Jorge Sanz
Micha Silver
Dr. Rafal Wawer
Zachary Woolard

Acknowledgements
Daniel Holt, LaTeX magic & layout support
Various reviewers & writers

The *OSGeo Journal* is a publication of the *OSGeo Foundation*. The base of this journal, the LaTeX 2ε style source has been kindly provided by the GRASS and R News editorial boards.

Journal online: http://www.osgeo.org/journal

OSGeo Homepage: http://www.osgeo.org

Mail contact through OSGeo, PO Box 4844, Williams Lake, British Columbia, Canada, V2G 2V8

9 771994 189002 00

ISSN 1994-1897

OSGeo

Your Open Source Compass

osgeo.org