International Environmental Modelling and Software Society (iEMSs)
7th Intl. Congress on Env. Modelling and Software, San Diego, California, USA
Daniel P. Ames, Nigel W. T. Quinn, Andrea E. Rizzoli (Eds.)
http://www.iemss.org/society/index.php/iemss-2014-proceedings

Web service and plug-in architecture for flexibility and openness of environmental data sharing platforms

S. Knox [a], **P. Meier** [a] **and J. Harou** [a]

[a] *School of Mechanical, Aerospace and Civil Engineering, The University of Manchester, Manchester, UK.*
(stephen.knox@manchester.ac.uk, philipp.meier@manchester.ac.uk, julien.harou@manchester.ac.uk)

Abstract: The sharing, comparison and maintenance of computer model input and result data for real-world projects is a challenge. In collaborative research projects, particularly when collaboration occurs between multiple institutions in different locations, the complications are greater still. The co-development of water resource network simulation and optimisation models is one field that faces this challenge. There is a need for ways to allow multiple users to build and apply models collaboratively. This paper argues that a web service architecture is an appropriate approach for multi-institution maintenance of resource network models. We describe the design of a prototype system that allows developers to collaboratively create, edit and share data associated with resource network models.

Keywords: data management, web services, collaboration, model development, model platform, open source

1 INTRODUCTION

Remote collaboration between and within resource management institutions is increasingly common. Working remotely has been shown to have no detrimental effect on the quality of work compared those working at the same site [Sonnenwald et al., 2003]. This decentralisation relies on an IT infrastructure capable of supporting the needs of the collaborators, specifically the ability to share, access and edit information in a controlled environment. Web services are a proven solution to many of the issues posed by cross-site work such as data access, scheduling and instant communication [Wang et al., 2010].

In resource system modelling such as water resources, there are several platforms that allow the management of models and their data to varying extents [Harou et al., 2010; Leavesley et al., 1996; Johnson et al., 1995]. So far, software environments for managing and sharing different modelling datasets remotely in collaboration with other model developers and users is limited. This paper argues such distributed capabilities can be achieved using web services in a plug-in architecture.

Web services are used to expose the functionality of an information system and make it available through standard web technologies [Alonso et al., 2004]. Web services provide a flexible, cross-platform approach to software development, and have been successfully applied to different areas of water research. This paper presents a web service approach to collaboratively managing both models and datasets. We refer to a *Model* as a piece of software which performs a simulation or optimisation.

The rest of this paper will be structured as follows: Section 2 discusses use of web services in water

resources. Section 3 describes model platforms, and how web services can be applied to allow collaboration between model developers. Section 4 presents a model platform design, for allowing multiple users to concurrently work on model data. Design, testing and limitations are discussed. Section 5 presents conclusions and future work.

2 WEB SERVICES IN WATER RESEARCH

Web services are a well established technology in environmental modelling [Buytaert et al., 2012; Horák et al., 2008; Goodall et al., 2011; Castronova et al., 2013]. Web services are flexible, platform independent and allow centralisation of services, making software releases and updates easier, allowing deployment on high performance clusters, and allowing users in different location to access the same information at the same time.

In water resources, the use of web services can be divided into two broad categories.

1. *Model-Based*: Models themselves are exposed as web services.
2. *Data-Based*: Data management and access is performed through web services.

Model-based web services provide a means for a single remote model to serve multiple requesting clients, removing the need to install model software locally and potentially providing greater computational resources for intensive calculations [Goodall et al., 2011; Gupta et al., 2012]. The standardisation of a model interface increases its appeal as it can be integrated with multiple Decision Support Systems.

Castranova et al, 2013 demonstrates model-based services by implementing a standardised way for models to be defined as web services using the Open Geo-Spacial Consortium (OGC) Web Processing Service (WPS) standard. Using the TOPMODEL hydrological model as an example, this work shows the benefits of using a service-oriented approach in integrated modelling but also demonstrates that there can be drawbacks in terms of response times and robustness.

Two types of data-based services exist: (1) Services like the CUAHSI hydrologic information system [Ames et al., 2009] which allow access to hydrological data encoded in a standard format, such as WaterML. (2) Services which manage the data required by multiple heterogeneous models [Horák et al., 2008; Mau and Roth, 2012]. Data-based services broaden the range of data available to model developers by standardising the data formats and making them available online. Our goal is to create a web service which will allow access, creation and maintenance of resource network data and provide a way for models to access this data directly.

3 MODEL PLATFORMS

Harou et al. [2010] describe a model platform as a generic software for managing resource networks (water, energy, transport etc) and their data. This platform stores data in one location and exports data for use by different models. Their proof-of-concept 'Hydro Platform' allowed users to draw a network with nodes and links, then assign attributes and data to these objects. The network could then be exported as model input files using a 'plug-in'.

Model platforms build on the idea of heterogeneous data collection and model access by integrating data from multiple sources and models into a single platform. Model developers can read in data from different sources and use this data to run models from the same user interface. A model platform does not in itself perform any modelling tasks or provide the ability to build models themselves. Instead, a model platform combines several different data sources (aided by tools to aggregate heterogeneous

data) with the ability to run or export data to models from the same graphical user interface. Access to these data and models is done through Apps, much like you would find in a web browser.

The proof-of-concept Hydro Platform was an integrated stand-alone system and did not support sharing data between users. This would require the export, then email (or some other transfer), then re-import of a dataset by another user. A better approach can be achieved through web services, whereby the data is stored remotely, with a user interface (UI) connecting via a web API. This way, multiple UI instances can connect to the same data source, allowing different users to work on the same network.

Two important factors need to be addressed in remote collaboration of this kind. Security and Access control. Security is a recognised concern for web services, particularly with respect to the abuse of services and prohibiting of unauthorised users [Goodall et al., 2011]. A collaborative model platform must address security issues, as well as allowing and enforcing management of valid users. Different types of users require different permissions, and users should have control over how and with whom their work is shared.

4 PROPOSED DESIGN

Here we describe work on a proposed model platform designed to allow the remote management of networks and data, using clients to connect with external models and data sources. This software is under development, and initial testing is under way.

The system, referred to as *Hydra Platform*, uses a client-server architecture where clients add and edit resource networks on the server. A network in Hydra Platform is a set of nodes and links representing a resource network, including their associated attributes and metadata. Networks are organised into projects which can be shared between users. A project can contain multiple networks, and multiple users can have access to the same project or individual networks within a project. Multiple sets of data can be associated with the same network through scenarios, each of which represent a different variation of the same network, for example 'dry year', 'wet year' etc.

The server hosts the data storage, processing and request handlers. Clients, are referred to as *Apps*, are independent software which provide external functionality such as importing and exporting data and running models. Apps connect to the server using a SOAP (Simple Object Access Protocol) web service. An App can request an entire a network using the `get_network` function call. The retrieved network can then be edited by the App as desired before calling `update_network` to save the changes.

The User Interface (UI) connects to the server through the same API as all clients, making it a special case App. For example, the UI can be used to manage the other Apps, making them act as plug-ins, much like those found in web browsers (see section 4.2).

By default, a server is installed locally with the UI, allowing the software platform to act as a stand-alone application. The UI can also be connected to a remote server where the user must log in. In this environment, projects, networks and datasets can be shared between users.

Figure 1 shows a simple architecture diagram of Hydra Platform. The server has three logical parts; the database, the DB library, which includes the ORM and the web server interface. Splitting the web server interface and the database library allows new server types to be deployed with ease and also allows back-end changes to the database to be transparent to the service implementation. Connecting to the server are Apps, which can be categorised into the UI, Model Apps and Input/Output Apps. Displayed here are a CSV import / export, a model app such as GAMS and import / export App to an external database. All Apps connect to the server through the same interface. The dashed line represents the fact that the UI can control other Apps, making them plug-ins to the UI.

The server is written in Python, using an ORM (Object Relational Mapping) built with SQLAlchemy.

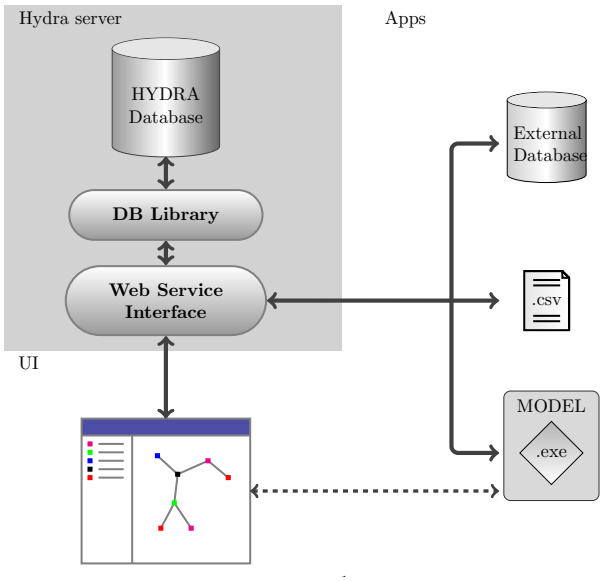

Figure 1. A client-server architecture, where multiple Apps of different kinds connect to the server through a web service API.

The ORM allows Hydra to be easily extended to use any database implementation (Oracle, SQLite, PostgreSQL) without affecting the API. To date, MySQL and SQLite have been used.

SOAP is used as the web service protocol, implemented with the Python-based Spyne RPC (Remote Procedure Call) toolkit. Spyne is used on the server as it is an actively developed, open source library which provides a comprehensive and robust toolkit for creating SOAP services. While SOAP has the drawback of being slow to parse large amounts of information (section 4.5), it has the ability to clearly define complex structures.

Hydra Platform is being developed as part of a jointly funded TSB project and is due for public release in early 2015. The server will be open-source and licensed as GPL, while client libraries will be LGPL. Apps developed against the server can be licensed independently.

4.1 Sharing and Security

To control how information is shared, and with whom, a tiered ownership system is employed. In an example work flow, a single user creates a network and assigns data values to the nodes and links. The creator (owner) can then choose to share the network with other users, specifying restrictions. Restrictions include 'Read Only', 'Read and Write', 'Read, Write and Share'. Users also have control over the sharing of individual datasets. Datasets are globally visible by default, but can be 'locked'. In this case only the creator, users with whom the dataset has been shared and administrators may view this dataset.

With the proposed remote server, every user must log in and new users must be added manually with a user name and password. Security is provided by standard web security techniques (login over ssl, for example). The server employs a 'Users, Roles, Permissions' hierarchy for user management. In this approach, several permissions are defined, reflecting individual pieces of functionality. Roles are then created, each with a subset of permissions. Users are then given one or more roles. This is a common approach, used, for example, by the Unix operating system.

4.2 Apps

An 'App' is an instance of a web service client. Apps can be written for any platform and in most programming languages. Several example Apps have already been created, including Import/Export to CSV and an Import/Export to the Generalized Algebraic Modelling System (GAMS) binary format. These Apps have been written in Python, making them cross-platform.

The User Interface is another example of an App, albeit a far more complex one than Import/Export. Other Apps can be downloaded, installed and run directly from the UI and can therefore be thought of as UI plug-ins despite them being independent programs. For an App to be used as a plug-in by the UI, it must be accompanied by an XML file, describing its functionality, required inputs and expected outputs. The format for these files are defined in an XSD. The UI interprets an App's 'Plug-in XML', displaying the appropriate buttons and inputs dynamically, and allows the App to be called directly by invoking an executable with the appropriate parameters. A Windows UI is being developed by partners in C#, demonstrating the cross-platform capabilities of a web-service architecture.

4.3 Database design

An important aspect of the server's design is hiding database implementation from end users as much as possible. Developers should only be interested in the functionality provided by the web service, not the specific details of the database. This allows the server and database more opportunity to be developed without affecting connecting Apps.

The proposed system is designed to accommodate a general network structure. The network topology is stored in Nodes and Links. Attributes and data (including meta data) are also stored. The supported data types are scalars, free form text, time series, arrays and equally spaced time series. To avoid duplication of data, datasets are stored independently to networks, allowing them to be referenced by several attributes at once. Users, roles and permissions are also accommodated.

Such a generic structure lends itself well to model platforms as it pushes the responsibility of application-specific network design to the Apps. This means that multiple Apps can use the same network in a variety of different ways. An advantage of this approach is that the proposed software can be applied to multiple areas of network analysis without modification (e.g. energy, transport, etc.).

4.4 Handling concurrency

Collaborative editing of documents on a single server requires that all updates are maintained in a consistent manner. As well as using database techniques such as versioning and transactions to keep data consistent, the server can handle multiple requests simultaneously.

For ease of development, the SOAP server has thus far been single threaded, so requests are queued up and processed sequentially. Should a request take a long time (over one second, for example) this can lead to poor responsiveness in a user interface. To accommodate multiple concurrent users, a multi-threaded solution using the CherryPy web platform [Hellegouarch, 2007] has been developed but has not been extensively tested yet. It is likely that we will integrate the server with a web platform such as Django [Forcier et al., 2008] before testing concurrency takes place. We are also investigating whether multiple single-threaded servers behind a load balancer [Chaczko et al., 2011] using mod_wsgi, for example, is favourable to a single multi-threaded server.

4.5 Testing and Performance

All server functions are covered by unit tests to ensure a stable development path. Load tests have been performed to gauge an upper bound of network size. The UI development has been used to test responses times, which in turn have led to optimisations. This iterative process will continue. Concurrency is also being tested to ensure that simultaneous requests can be handled efficiently.

Tests thus far have focussed on features and response times and have shown that the server can handle small requests adequately for a smooth UI experience. It is in data-heavy interactions, where a slow-down is observed. Figure 2 shows the client-side parsing time versus server-side processing time for the `add_network` call. Using synthetically generated networks of increasing size (each node and link has two attributes, one of which is a time series), it can be clearly seen that the parsing time of the SOAP package in SUDS is costly. The solid line shows the time taken to generate the outgoing SOAP request and parse the incoming response. The dashed line indicates how long the server takes to process the `add_network` request.

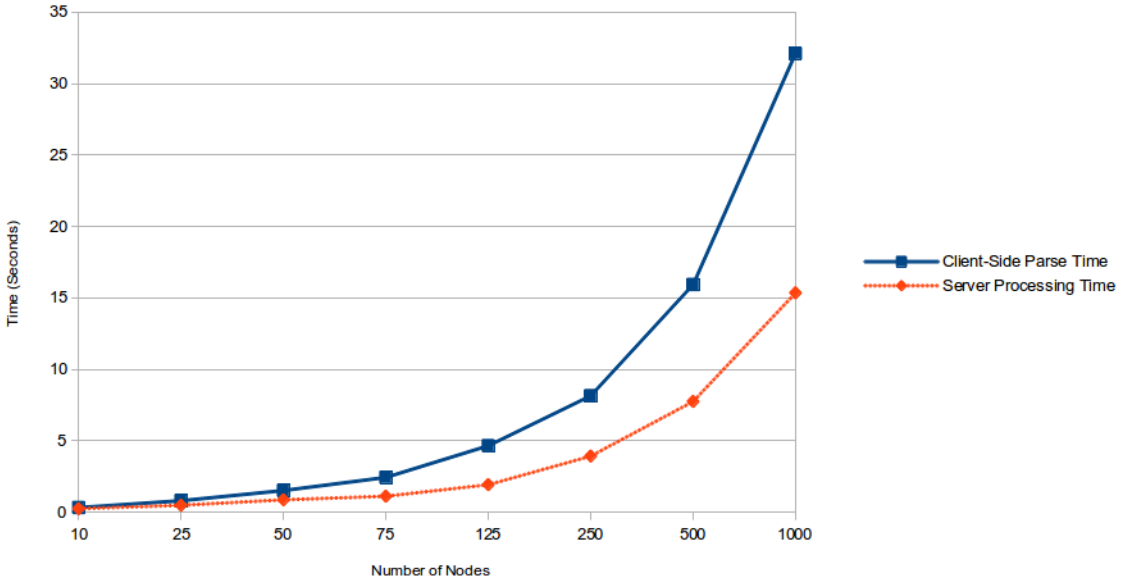

Figure 2. A comparison of the time it takes to perform the input and output parsing of an entire network, versus the time taken by the server to process this request. Tested using the most data intensive function call, `add_network`

Most testing was done with a large regional water capacity expansion optimisation model with 1150 nodes, 242 links and 22592 datasets, identical to the model described by Padula et al [Padula et al., 2013] but with data derived from public sources. This network gives a realistic use case for large network interactions. 35.72 seconds is spent by the server processing and saving the network during the `add_network` function call. 175.4 seconds is spent by the client parsing the request and responses to XML.[1] These results were obtained using the python SOAP library, SUDS. Other python client libraries have also been tested with similar results. While saving can be made on the server side, the most major impact on upload is the use of XML, which is required in SOAP.

A number of solutions to this issue are being considered, including using a JSON (JavaScript Object Notation) instead of XML. In an initial test using JSON, client-side parsing time of the sample network

[1] Tested on a Dell Latitude Laptop running Ubuntu with 16GB RAM, Intel quad-core i7 2.10Ghz processor

is reduced to 0.141 seconds. The modular design of Hydra makes this change simple on the server side, but Apps and unit tests will need to be updated to reflect this change before more significant testing can take place. One drawback of using JSON is that while it is faster to parse, it does not provide the same level of validation and type safety found in XML.

5 CONCLUSIONS

Model platforms allow model developers to manage data and access multiple models in a single integrated environment. We are implementing an open-source model platform for network-based models, built with a Service Oriented Architecture. We have shown that although limitations exist, Hydra Platform can allow model developers to jointly edit the same network, ensuring consistent comparisons between models. The App architecture allows computational models written in any programming language to be integrated.

A user interface is being implemented independently where multiple users can connect to a remote server and graphically draw, share, edit and save the same network. Using an App, each user sharing a network can export it to the data format required for their model. The results of an analysis can then be re-imported for visualisation and comparison. Ownership, permissions and security issues are being addressed.

Initial testing has begun for load management, concurrency, response time for user interface tasks and deployment on remote and local machines. Thus far, we have discovered that while large networks are handled well on the server, generating the requests can be slow using SOAP. Performance increases can be gained by replacing XML with JSON.

Future goals include testing server configurations in a realistic multi-user environment, and preparing for an early release to collaborators. Some known issues are being addressed by introducing a HTTP-based, multi-threaded web service interface. The modular design of the server means that this change can be performed while still maintaining the current SOAP server.

ACKNOWLEDGMENTS

The project is partially funded by the UK government through the Technology Strategy Board (TSB)

REFERENCES

Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2004). *Web services*. Springer.

Ames, D., Horsburgh, J., Goodall, J., Whiteaker, T., Tarboton, D., and Maidment, D. (2009). Introducing the open source cuahsi hydrologic information system desktop application (his desktop). *18th World IMACS Congress and MODSIM09 International Congress on Modelling and Simulation, Modelling and Simulation Society of Australia and New Zealand and International Association for Mathematics and Computers in Simulation*, pages 4353–4359.

Buytaert, W., Baez, S., Bustamante, M., and Dewulf, A. (2012). Web-based environmental simulation: bridging the gap between scientific modeling and decision-making. *Environmental science & technology*, 46(4):1971–1976.

Castronova, A. M., Goodall, J. L., and Elag, M. M. (2013). Models as web services using the open geospatial consortium (ogc) web processing service (wps) standard. *Environmental Modelling & Software*, 41(0):72 – 83.

Chaczko, Z., Mahadevan, V., Aslanzadeh, S., and Mcdermid, C. (2011). Availability and load balancing in cloud computing. In *International Conference on Computer and Software Modeling*, volume 14.

Forcier, J., Bissex, P., and Chun, W. (2008). *Python Web Development with Django*. Addison-Wesley Professional, 1 edition.

Goodall, J. L., Robinson, B. F., and Castronova, A. M. (2011). Modeling water resource systems using a service-oriented computing paradigm. *Environmental Modelling & Software*, 26(5):573 – 582.

Gupta, T., Jonesb, R., Bastinb, L., and Cornfordb, D. (2012). Integrating openmi and uncertweb: Managing uncertainty in openmi models. In *International Environmental Modelling and Software Society (iEMSs)*.

Harou, J. J., Pinte, D., Tilmant, A., Rosenberg, D. E., Rheinheimer, D. E., Hansen, K., Reed, P. M., Reynaud, A., Medellin-Azuara, J., Pulido-Velazquez, M., Matrosov, E., Padula, S., and Zhu, T. (2010). An open-source model platform for water management that links models to a generic user-interface and data-manager. In Swayne, D. A., Yang, W., Voinov, A. A., Rizzoli, A., and Filatova, T., editors, *2010 International Congress on Environmental Modelling and Software Modelling for Environments Sake, Fifth Biennial Meeting, Ottawa, Canada*. International Environmental Modelling and Software Society (iEMSs).

Hellegouarch, S. (2007). *CherryPy Essentials: Rapid Python Web Application Development Design, Develop, Test, and Deploy Your Python Web Applications Easily*. Packt Publishing.

Horák, J., Orlik, A., and Stromsky, J. (2008). Web services for distributed and interoperable hydro-information systems. *Hydrology and Earth System Sciences*, 12(2):635–644.

Johnson, W., Williams, Q., and Kirshen, P. (1995). Weap: A comprehensive and integrated model of supply and demand. In *Proceedings of the 1995 Georgia Water Resources Conference*. Georgia Institute of Technology.

Leavesley, G., Restrepo, P., Markstrom, S., Dixon, M., and Stannard, L. (1996). The modular modeling system (mms): Users manual. *US Geological Survey Open File Report*, 96(151):142.

Mau, P. and Roth, M. (2012). Lost in translation mediating between distributed environmental resources. In *Proceedings of 2012 International Congress on Environmental Modelling and Software*.

Padula, S., Harou, J., Papageorgiou, L., Ji, Y., Ahmad, M., and Hepworth, N. (2013). Least economic cost regional water supply planning optimising infrastructure investments and demand management for south east englands 17.6 million people. *Water Resources Management*, 27(15):5017–5044.

Sonnenwald, D. H., Whitton, M. C., and Maglaughlin, K. L. (2003). Evaluating a scientific collaboratory: Results of a controlled experiment. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 10(2):150–176.

Wang, L., Von Laszewski, G., Younge, A., He, X., Kunze, M., Tao, J., and Fu, C. (2010). Cloud computing: a perspective study. *New Generation Computing*, 28(2):137–146.