Environmental Modelling and Software 122 (2019) 104538

Contents lists available at ScienceDirect

Environmental Modelling and Software

journal homepage: www.elsevier.com/locate/envsoft

An open-source data manager for network models

Stephen Knox [a], James Tomlinson [a], Julien J. Harou [a,c,*], Philipp Meier [b], David E. Rosenberg [d], Jay R. Lund [e], David E. Rheinheimer [f]

[a] Department of Mechanical, Aerospace and Civil Engineering (MACE), University of Manchester, Manchester, UK
[b] Eawag, Department of Surface Waters - Research and Management, Kastanienbaum, Switzerland
[c] Department of Civil, Environmental and Geomatic Engineering, University College London, UK
[d] Department of Civil and Environmental Engineering and Utah Water Research Laboratory, Utah State University, 8200 Old Main Hill, Logan, UT 84322-8200, USA
[e] Department of Civil and Environmental Engineering, 3109 Ghausi Hall, University of California, One Shields Avenue, Davis, CA 95616, USA
[f] Department of Civil and Environmental Engineering, University of Massachusetts, 130 Natural Resources Road, Amherst, MA 01003, USA

ARTICLE INFO

Keywords:
Model platform
Web services
Python
Open source
Software framework
Network modelling

ABSTRACT

Developing simulation and optimisation models for resource networks like water or energy systems increasingly involves integrating multiple data sources and software. Connecting multiple models and managing data accessed by different groups of analysts is a software challenge. Many resource systems are represented in computer models as networks of nodes and links, driven by a range of objectives and rules. We present a data storage platform, written in Python, which exploits the commonality of network representations to store data for multiple model types within a single deployment. This open-source platform provides a common source of data to multiple models using consistent data formats, reducing likelihood of error compared to file based data management. When deployed as a web service, it allows data to be shared securely among authorised users over the internet, facilitating collaboration. A case study describes the hosting of a water utility planning model, with an accompanying worked example.

Software availability

Program title: Hydra Platform
Developer: Stephen Knox
Contact address: stephen.knox@manchester.ac.uk
Software Access: https://www.github.com/hydraplatform
Year first available: 2015
Hardware required: Windows 7, 10 (tested), Linux, MacOSX
Program language: Python
Program size: 116k
Availability and cost: open source
License: LGPL for hydra base, hydra server. MIT for clients.

1. Introduction

Recent advances in simulation and optimisation of environmental systems have relied on increasingly detailed models running many scenarios Herman et al. (2015). Cluster and cloud computing also has enabled larger and more complex models to be used for increasingly sophisticated decision-making analyses (Reed et al., 2013; Kasprzyk et al., 2013; Maier et al., 2014; Huskova et al., 2016; Eker and Kwakkel, 2018). In addition to increasing the data-intensive nature of environmental modelling, the sharing of data among collaborators and stakeholders has become common and will become increasingly important in future (Carr et al., 2012; Voinov et al., 2016; van Bruggen et al., 2019). However, translating data formats and managing data itself can be a barrier to effective and efficient modelling. This can result in model developers spending more time transforming data than with the actual modelling and developing system insights.

Multi-disciplinary and remote collaboration is now a common requirement for environmental modelling, and is often essential for environmental system modelling (Laniak et al., 2013). Manually managing multiple input and output files, and synchronising among collaborators is inefficient and error-prone, as it requires continuous communication to ensure all parties remain up-to-date. Asynchronous collaboration on the same data or model is cumbersome or impossible manually. Centralising data storage where users control access to data eliminates this issue. Providing a suite of data management functions, such as scenario management and history tracking, through a multi-user asynchronous system further reduces the likelihood of data becoming out of sync. Several approaches have been used in the last two decades to

* Corresponding author at: Department of Mechanical, Aerospace and Civil Engineering (MACE), University of Manchester, Manchester, UK.
E-mail address: julien.harou@manchester.ac.uk (J.J. Harou).

https://doi.org/10.1016/j.envsoft.2019.104538
Received 1 March 2019; Received in revised form 13 August 2019; Accepted 27 September 2019
Available online 30 September 2019

address the model data management problems. Below we present some approaches and identify some gaps in this area.

Many existing modelling platforms (e.g. commercially available decision-support systems for water and energy planning) use an all-in-one single-user design, where a software installed on a user's machine contains the user interface, data storage and models all bundled together (Johnson et al., 1995; Kuczera, 1997; Ltd, 2000; Sieber et al., 2005). Modern web technologies can now provide a similar experience, with several additional advantages. Some benefits of using a web framework over traditional desktop-based software include:

1. Upgrades to the database, code base and user interface (UI) can be done transparently across collaborators.
2. If the models are made accessible via a web Application Programming Interface (API) or User Interface (UI), features can be added and upgraded transparently, without the need to reinstall or reconfigure software.
3. Sharing data is a simple process, as it requires relaxing access permissions on existing data.
4. Having a centralised data management system with a standardised API ensures consistent data formats, allows for access control and allows model results to be more easily investigated and reproduced.
5. Scalability to large-scale model runs can be simplified, as the server can be upgraded or connected to a cloud service without user disruption.
6. Web user interfaces such as Google Docs, Google Maps, Lucidchart, etc. have become common and are considered by some to match desktop-based approaches for usability (Blau and Caspi, 2009). Having data accessible through the web allows users to choose how they work with the data, be it with such online tools, or with desktop solutions.

Use of web services for managing environmental data is well established (Buytaert et al., 2012), with projects focussing on deploying models through web services (Kralisch and Krause, 2006; Gregersen et al., 2007; Cetinkaya et al., 2008; Kralisch et al., 2012; Peckham et al., 2013) , standardisation of time series data (Ames et al., 2009), management of large quantities of data (Vitolo et al., 2015; Folk et al., 2011; Rew and Davis, 1990) , coupling of models at runtime (Gregersen et al., 2007), and management of data for specific model runs (Knox et al., 2014; Meier et al., 2014; Zander and Kralisch, 2011). There are few enforced general standards or requirements for environmental data, which means there is no single solution to data management. Many projects specialise in one area, be it serving large quantities of time series data (Ames et al., 2009), focussing on supporting water resource modelling (Chalh et al., 2015) or standardising model run-time data communication.

A focus of recent research has been facilitating model sharing and result publishing, to better ensure model reproducibility (Tarboton et al., 2013). By exposing a model platform through a web service, it is possible to make particular model runs and result sets public either by relaxing access permissions on data in the system or by exporting the data to public repositories such as HydroShare. Using this approach, transparency is ensured as the public data is exactly what is used by the model.

McKinney and Cai (2002) presented an object-oriented approach to data management of data for network-based models within a GIS (Geographic Information System) environment. This used a generalised node-link structure for networks, where model-specific subclasses could be created. This allowed the structure to be applied to multiple network types. Harou et al. (2010) describe a model platform as a generic software which separates models from data management and visualisation. Such software stores data in one location and exports data for use by different models. 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 data from different sources for use in multiple models. A Model Platform need not itself perform any modelling tasks or provide ability to build models. Instead, a model platform combines several data sources (aided by a plug-in architecture and tools to aggregate heterogeneous interfaces) with the ability to run or export data to various models from the same source.

Model platforms facilitate model integration through loose model coupling (Jiang et al., 2017; Goodall et al., 2013). Loose coupling allows models to communicate through a third-party to store intermediate data in a standardised way, eliminating the need for direct transfers between models, which can require custom, unscalable solutions (Kelly et al., 2013). This approach introduces an overhead, and is more suitable to model chaining (Meier et al., 2014) where models run in their entirety in a given order, in contrast to a solution like Pynsim or OpenMI (Knox et al., 2018b; Gregersen et al., 2007) which allows the models to be connected on a per-timestep basis.

A centralised data store relies on the structure of environmental modelling data being predictable enough to store data generally, and the system being flexible enough to deal with the idiosyncrasies and exceptions of individual systems and model requirements. In addition it relies on a common understanding of how data is represented. Many environmental management models use networks of nodes and links to represent systems, and use largely predictable data formats such as timeseries, arrays, scalars.

Despite advances described above, the ability to easily connect different models to the same data manager, share and collaborate on model runs, and make data public is still needed, with no current technologies having yet made these tasks easy. The Model Platform concept goes some way to achieving this by separating data management from the models themselves. In this paper, we add web services, which enable the collaborative aspects of model development and analysis.

We present a solution for storing network structures and associated datasets in a general way, with the addition of version management, multi-user sharing including data ownership control. The presented platform manages network-based data to support modelling tasks, such as uploading new inputs data, managing scenarios, and extracting results.

While data is stored in a general way, this is not a single public data repository. Rather, it is a software platform that can be used with specific models and does not enforce metadata rules to enable searching of datasets, for example. This platform supports a modeller who has compiled the necessary data from the appropriate sources, and built a network-based model. It allows the modeller to store, manage and share the inputs and outputs of their model with other users. Support for scenario analysis and data sharing gives the user the possibility to compare results and collaborate with other users. We call this system 'Hydra Platform'.

The rest of this paper is structured as follows: Section 2 describes the design of Hydra Platform. Section 3 outlines a case study which illustrates a real-world deployment of Hydra Platform. The appendix has a worked example of a water-energy simulation model and two further partially implemented case studies.

2. Design

Hydra Platform (referred to here-on as 'Hydra' for brevity) is a model platform, and aims to provide data support for network-based models. The software stores inputs and outputs for models which use a network structure including data, metadata and model topology. Models which use network structure include for example the simulation or optimisation of water, energy, transport, and trading systems. Networks are not the only way to represent a modelled domain. Some models, for example, are raster-based hydrologic models (Marsh et al., 2018) and would not be suitable for a network-based system.

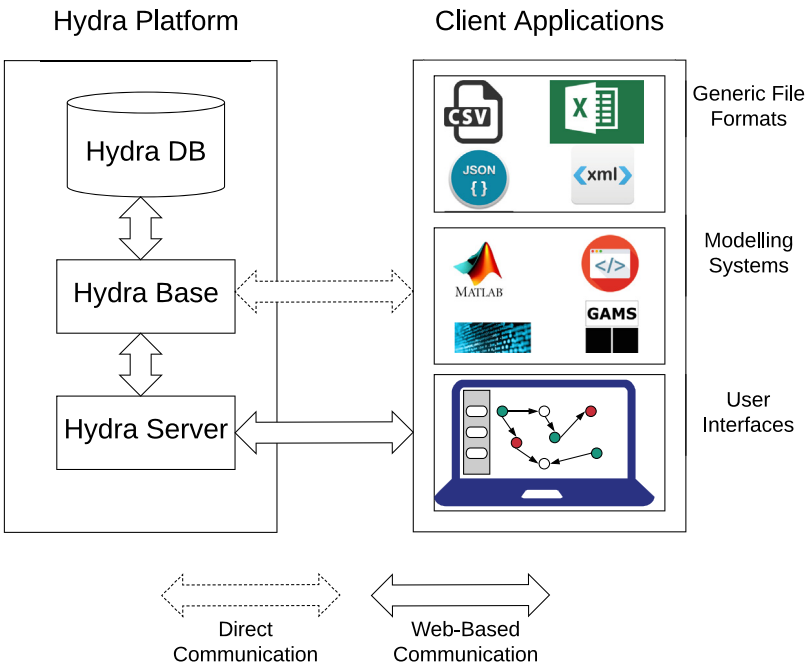

Fig. 1. Overview of the Hydra Architecture. Data storage and management (left) are separated from the application of that data (right). Data within Hydra can be used within three broad categories — importing and exporting to generic data formats such as JSON or Excel, translating to and from specific model instances, written in different languages such as GAMS or Matlab, and visualised and managed data using a UI.

2.1. Principle

Hydra is designed to provide a standardised means to store and access data for network-based modelling and to allow data from multiple models, model types and modelling disciplines to be stored and accessed from the same place.

To achieve this, Hydra has two main design principles:

- Decouple data storage and management from application logic, and
- Provide support for a wide range of applications.

To achieve the first, an architecture is required which contains minimal application logic, and is flexible enough to store different types of data. To achieve the second, a programming interface (API) is required which links to the data storage and provides logical and standardised protocols for accessing data. These are often competing requirements.

Supporting a wide range of applications requires a design which understands the scope of possible applications while maximising its utility within that scope. In the case of Hydra, this is network-based resource modelling. Within this broad application area, Hydra should be useable by any modeller from different disciplines equally — it should be designed from the outset to not favour any one. This principle relies on a generalised design, where application-specific content can be entered as data, rather than requiring changes to Hydra's source code or architecture.

2.2. Methodology

Hydra uses a high-level representation of networks to support storing a range of network types, and employs a 'templating' system, which mimics object-oriented programming by allowing users to define user-configurable network types, and then create instances of those types. Using this approach, a user can define types of nodes and links, with properties relevant to specific modelling problems. Instances of those nodes and link classes, along with their datasets are the inputs for modelling. A range of dataset formats are supported, along with ability to define custom data formats for non-standard values. The customizability of the system facilitates integration with a range of models, so long as they use a network (nodes and links) structure.

Hydra is a software library, written in Python, that provides an application programming interface (API). The API contains high-level operations to create and edit networks, manage scenarios, and share networks with other users. The user has no direct interaction with the database (i.e. they must use the API to perform queries on the database). This separation of the logical layer from the storage allows the database implementation to be upgraded or altered while data operations remain unchanged. A web API (an additional Python library) connects to the core Python library to allow remote data management. Using a web API allows multiple users to use and access the same data, limiting duplication in collaborative modelling. Remote access through a web API also allows software developers to build applications in any mainstream programming language. Separating database from logical operations, and providing a web API all serve to broaden the application areas in which Hydra can operate, from a simple desktop installation to a large-scale, cloud-enabled service. Fig. 1 gives an overview of the architecture.

2.3. Usage

Four use cases for interaction with Hydra include:

1. Importing data from a 3rd party file format, for example output of a model run.
2. Exporting data to a 3rd party file format, for example for use in a model run.
3. Running a model with data stored in Hydra. This involves exporting data to the appropriate file format, running a model instance, then importing results back to Hydra.
4. Using a User Interface, i.e, a graphical front-end so non-developers can interact with the data.

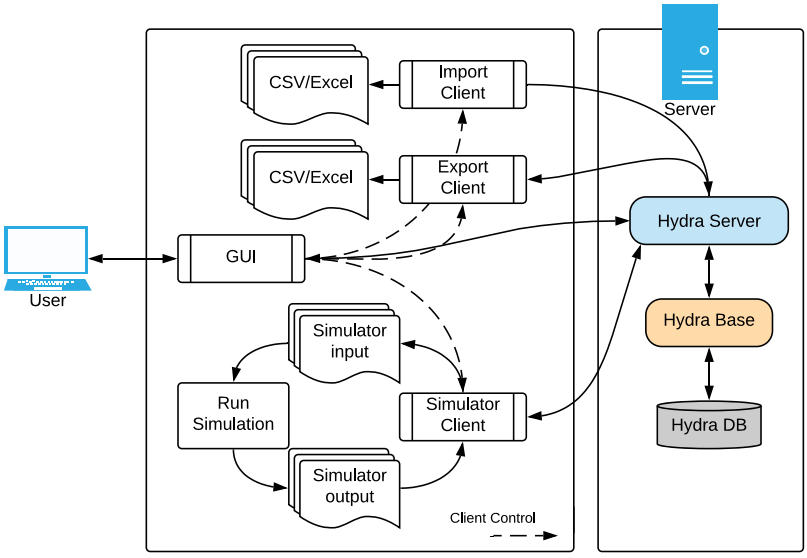

Fig. 2. Hydra's web architecture using example clients. The client makes remote procedure calls to hydra server to import and export data, to general file formats like Excel, to applied systems like a simulator, and to a GUI which visualises the network and data.

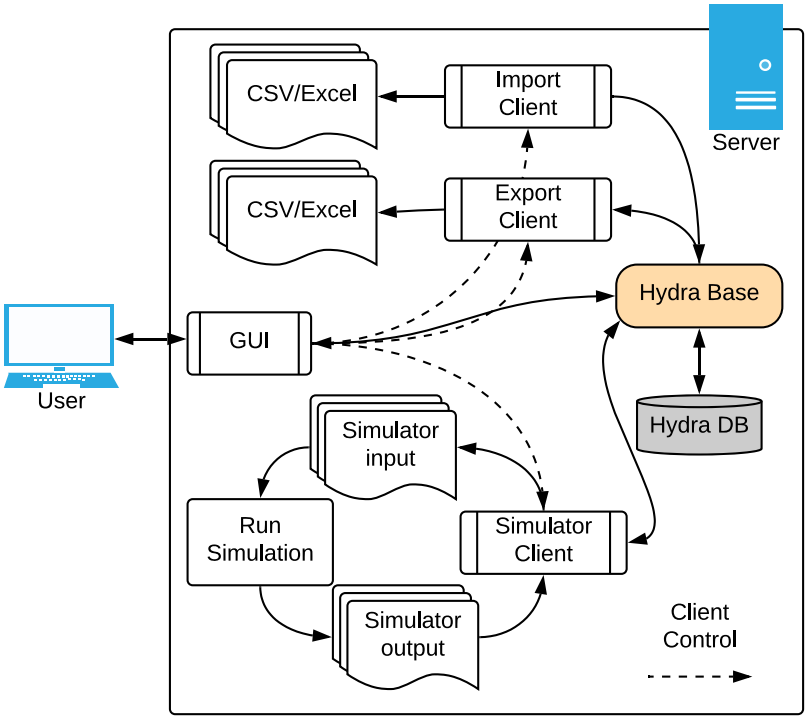

Fig. 3. Hydra's architecture when a server is not deployed. Clients communicate directly with Hydra Base on the same physical machine. This approach gives greater data storage efficiency, when internet access is unreliable, and/or where a multi-user environment is not needed. The clients require little or no alteration to work with this local deployment and with Hydra Server.

All use cases require software that interact with Hydra. A client application, or 'client' for brevity, is a software application that makes requests to an API. Clients are not part of Hydra, but independent software packages which provide functionality, such as the use cases described above.

This section describes how clients interact with Hydra to store and access data, the generalised data storage design and how it enables accessing diverse network types and collaborative features.

A client application requests data from or sends data to Hydra by interacting with the API. For example API functions `get_network` or `update_network` allow a client to retrieve and modify a network respectively. A client application may interact with the API to read,

write, update or delete data from Hydra. Users of Hydra can write their own custom client applications provided they interact with the API appropriately. Hydra has its own data format which the client API must use. This format closely reflects the database schema structure. The data structures can be explored in the examples on the Hydra Base GitHub repository Knox et al. (2018a), or in the JSON client application (Knox, 2018).

2.3.1. Data import/export

Data Import and Export (IO) relates to data in any format which can be stored in Hydra. This category includes data not necessarily linked to any one modelling system or method, but which can be communicated. These might include network topology contained in shapefiles, time series data such as that available from the CUAHSI HIS (Ames et al., 2009), and encoded as WaterML (Valentine et al., 2012), data stored in Microsoft Excel files, CSV files etc.

2.3.2. Model clients

Model clients allow communication with specific models or modelling frameworks, for example such as Matlab, GAMS (Rosenthal, 2008), Pyomo. These require input and output in specific data formats. A model client can also include the function to run the model itself, in which case the client algorithm would be:

1. Export the data to file 'input.modelformat'.
2. Run a model instance using 'input.modelformat'.
3. Once complete, import results from 'output.modelformat'.

The process of import and export are identical to a data import / export client. A user could use the export and import clients in isolation, running the model manually, but the 'run' client allows this step to be automated.

2.3.3. User interfaces

Hydra is a web server API and so does not have a graphical interface. A User Interface (UI) is a client application which allows a user to visually edit network topology, data, initiate model runs, share results etc.

2.4. Architecture

Hydra has two components: *Hydra Base* and *Hydra Server*. Hydra Base is a Python package (library), and performs all computational tasks such as database interaction and validation. Hydra Base can be installed as a standard Python package and used as such, with no requirement to act as a web server. Hydra Server is an additional Python package, which exposes the functions within Hydra Base as Remote Procedure Calls (RPCs). This separation of the base from the server allows for flexible deployment, with the same instance being accessible remotely, locally or both, depending on the requirement of the application. Figs. 2 and 3 demonstrate the different ways Hydra can be deployed, where clients applications interact through a web service or directly with the Python API respectively.

2.4.1. Hydra object types & terminology

The following terms refer to concepts within Hydra. This nomenclature is reflected in the database structure and API, and will be referred to throughout this paper. Capitalisation of these terms later in the paper indicates that they relate to this list.

- A **Project** is a container for Networks. A Project can contain multiple Networks and have multiple owners, each with configurable levels of access.
- A **Network** describes a topology and contains Nodes, Links and Groups.
- **Nodes** & **Links** form the topology of a network and can have data associated with them through attributes. These can have types, like 'Reservoir', or 'Water Diversion' for Nodes or 'River' or 'Pipe line' for Links.
- **Groups** are containers of Nodes, Links and other groups. These can be used to define interdependencies among Nodes & Links, or to construct a non-physical entity such as a governance hierarchy.
- An **Attribute** is defined by a name and dimension, for example: 'Water Flow, Volumetric Flow Rate'. To illustrate, consider two attributes: {Water Flow : Volumetric Flow Rate} and {Storage : Volume}. Node A might have a Water Flow associated with it while Node B might have both Water Flow and Storage.
- A **Dataset** is an individual data value. It is defined by a name, data type, unit, value and metadata. When associated with an Attribute, the unit of the Dataset must match the dimension of the Attribute it refers to.
- Hydra supports multiple datasets for the same attribute. This is done through **Scenarios** . A Network can contain multiple scenarios, each of which contains mappings between an Attribute and a Dataset.
- As Hydra can support data for multiple models, it uses a **Template** to define the types of Nodes, Links and Groups required for each model.
- A **Template Type** is defined by a name (e.g. 'Reservoir') and a set of Attributes (e.g. 'Storage', 'Max Head', etc.). This can be thought of as the equivalent of a class in Object Oriented Programming (OOP). A Node or Link within a Network will be an instance of a Template Type.

2.5. Scenarios & data

A Scenario in Hydra represents data associated with a Network. Scenarios allow attributes of Nodes, Links and Groups in a Network to have multiple Datasets associated to them at once.

Any Node, Link or Group attribute can be connected to a Dataset through a Scenario. A reference table in the database allows multiple attributes to be connected to the same Dataset to ensure Datasets are not duplicated.

Datasets are stored independently of Networks in Hydra. This allows the system to store and manage Datasets without the need for network topology. This is useful for example when there is a need to import time series from CUAHSI HIS (Ames et al., 2009) or other data without having a network. This also avoids the duplication of data.

2.5.1. Units

Hydra manages units and dimensions by providing a default set including length, area, volume density etc., with a set of default units for each dimension. New dimensions and units can be added on a per-user basis. Functions such as `add_unit` and `get_all_dimensions` allow units to be managed and searched. Units are not required for datasets, but it is encouraged to specify a unit. If a dataset has no dimension, the unit 'dimensionless' is used.

Hydra can convert units on Datasets. The `get_dataset` function call has an optional 'unit' argument which will return a dataset object with the requested dataset converted to the specified unit. An error is thrown if units are not of the same dimension.

2.6. Client interaction

Hydra is object-oriented and uses objects for network and data management, and this is reflected in the web API, which is also object-based. That is, the user sends/receives objects to/from the web server or the native Python library in the same way.

The Web API is a wrapper for the functions within Hydra Base. These functions have the same names and accept the same arguments, and pass these arguments into the equivalent function within Hydra

Base. The API is implemented as a JSON Remote Procedure Call protocol (RPC) using the Spyne Python library. Spyne allows for validation of argument types and definition of the object types which the API can accept. Spyne also allows the same API to be deployed for several protocols at once, for example SOAP and JSON. This provides flexibility for different client requirements (for example using SOAP (Simple Object Access Protocol) with C# .net) can automatically build classes and perform validation within the client. An example of how the web API wraps hydra functions can be found in Listing 2, Appendix A.4.

There are two ways in which clients can interact with Hydra: *local* or *remote*. Hydra provides client libraries for Python, Java and C#. The Python implementation is the most complete, and provides two classes which enable interactions: `JSONConnection` and `RemoteJSONConnection`. As the names imply, these classes allow interaction with Hydra using JSON-based data structures either directly with the Python library or using http requests. The data sent and received from each connection type and the function calls are identical, meaning the client code would require little or no alteration when using the different connection types. Listing 1 in Appendix A.4 presents an example of an interaction with Hydra using a remote connection with the Python client library.

2.7. Relational database

Hydra uses a relational database for data storage, with SQLite, Mysql and Postresql so far tested. The Python library SQLAlchemy is used for concurrency and transaction management, upgrades, queries, and allows support of multiple database types. SQLAlchemy is a popular open-source, well supported library, proven to support the security, concurrency and scalability requirements of large-scale web applications.

The summary database schema appears in Fig. 4 and the full version appears in Fig. 11, in the Appendix.

2.8. Templates

Hydra is a generalised data management platform which can support multiple types of networks concurrently. For example, multiple users might work on different projects, using different models, written in different languages, without knowledge of any other. Supporting this means allowing users to define custom Node, Link and Group types which relate to their model. For example, User A's model deals with 'Supply' and 'Demand' nodes, while User B deals with 'Reservoir', 'Desalination' and 'Treatment' nodes. Hydra achieves this with 'Templates'. A Template allows a user to create definitions of the Node & Link types required by their model, including the attributes which each type requires (see Figs. 5–7).

Templates allow Hydra to support multiple different types of Networks within the same system, supporting multiple types of model concurrently. In addition to defining the Node & Link types required by a model, Hydra supports data validation in the template API. These are implemented as 'restrictions' on the attributes of a type, and can include enumerations (e.g. 'the value must be 10, 20 or 30'), data ranges ('the value must be between 10 and 30'), date ranges (e.g. 'the time series must be between January 2018 and December 2018').

2.9. Collaboration and security

Hydra supports user and data management by:

1. Global management of Users, Roles and Permissions, and
2. Management of entities by their 'owner'.

Hydra uses a 'User-Role-Permission' structure to define what a user can and cannot do. This is done by assigning a user a Role, which has some permissions such as 'add_network', 'add_user'. In these examples, an attempt to call the 'add_network' or 'add_user' function by a user without these permissions will result in a permission error. Using this system, a user can be given general permissions across the whole platform, but may still be restricted on a case-by-case basis.

In addition to global permission management, Hydra supports the sharing and management of individual Projects, Networks, Scenarios and Datasets. The user who creates a Project, Network or Dataset is deemed its 'owner' and has control to delete, hide or share it. Here we describe how each can be managed:

1. Projects: A user can share a project with other users through the 'share_project' function call. There are three modes of sharing 'view', 'edit' and 'share'. These flags indicate whether the receiving user can view, edit or re-share the project. Once shared, the receiving user can see all networks within the shared project.
2. Networks: Like Projects, Networks can be shared with other users. In this case, the user will be able to see the Project containing the shared Network, but any unshared Networks,within the same Project will not be visible. The same 'view, edit, share' rules apply to the Network.
3. Datasets: Datasets are by default independent of any Network, Project or Scenario. Datasets are then linked to Networks through Scenarios, as explained in Section 2.5, to reduce duplication. An important aspect of data management is the ability to store confidential data. To this end, Hydra allows datasets to be 'hidden' such that only the owner can see them. The owner can then share these datasets with other users, while keeping them hidden from other users. Hidden dataset are stored in the same way to other dataset. Their access is controlled by the 'hidden' flag in the `tDataset` table which when set to 'Y' checks the `tDatasetOwner` table. This table is a whitelist of users who have access to this dataset.

Security is important in multi-user systems which convey data. Hydra employs well-established security protocols such as encrypting passwords using the Bcrypt hashing function. In Hydra, multiple users may share potentially sensitive data with other users. Hydra's security protocols are designed to ensure only users with the appropriate privileges may view and edit data. A series of unit tests (available on GitHub) were created to ensure sharing functions operate as expected, and these are tested automatically whenever changes are made to Hydra's code. Security features such as secure web connections (https) and altering Hydra's default password hashing key are the responsibility of each deployment.

3. Case study

Hydra is operational, and has been used in regional-scale modelling efforts within the UK. We present a case study to illustrate an actual deployment in an industrial setting, where a consultancy manages large numbers of model runs and scenario analyses for a utility company, and where efficient sharing of data was necessary. Hydra enabled efficient sharing of data, and simplified the modelling workflow.

A common modelling workflow in an industrial setting involves a water institution such as a water utility or river basin agency hiring a consultant to build a model to analyse, for instance, the levels of service impacts of a new infrastructure asset or management policy. The model used in this project is the 'Economic Balance of Supply and Demand (EBSD)' model (Padula et al., 2013), a least-cost water supply planning optimisation model used by UK water companies for infrastructure investment planning. This model is used to minimise investment costs over a 25 year planning horizon subject to maintaining supply–demand balance at a particular level of service. The problem is

Fig. 4. A simplified Hydra Platform database schema, with some tables removed for clarity. The different colours and styles of the lines distinguish the connection types. The full schema can be found in Fig. 11, in Appendix.

formulated as a mixed integer linear programming (MILP) with binary decision variables for each feasible investment option in each year. The solution is subject to constraints that ensure the supply–demand balance is satisfied in each 'Water Resource Zone' planning area and investment dependencies or exclusivities are enforced. In our case the EBSD model is written in GAMS (Rosenthal, 2008), a scripting language for mathematical programming optimisation models. The model requires a single text input file containing the input network topology and scenario data. For a large water utility with many feasible options, solving the MILP is computationally expensive — a run containing all possible options can take several hours.

As per regulatory requirements the water utility supplies both supply and demand data. This is generated independently prior to and during the modelling process. Supply data includes feasible infrastructure options including their location, size and outputs. Demand data is derived from, for example, population forecasts and analysis of historic per-capita consumption.

Prior projects with this model involved the utility company requesting the consultant to run the model (located on their PC), then waiting for the results to be returned via email in Microsoft Excel format. Reporting was done by the consultant using PowerPoint. This process proved slow and cumbersome, limiting the amount of iteration and refinement of the modelling, and ultimately reducing business intelligence on different investment strategies.

As regulatory requirements change, UK water utilities are expected to perform more analysis (model runs), with more input scenarios and sensitivity analysis. The traditional workflow described gave the utility little ability to perform such detailed analysis, as running models and accessing results was too time consuming. Hydra gave the utility more autonomy to perform model runs themselves, and for reporting and data analysis to be shared between consultant and utility, and potentially with regulators.

The original EBSD model was expanded to employ the Modelling to Generate Alternatives (MGA) technique (Chang et al., 1983; Brill

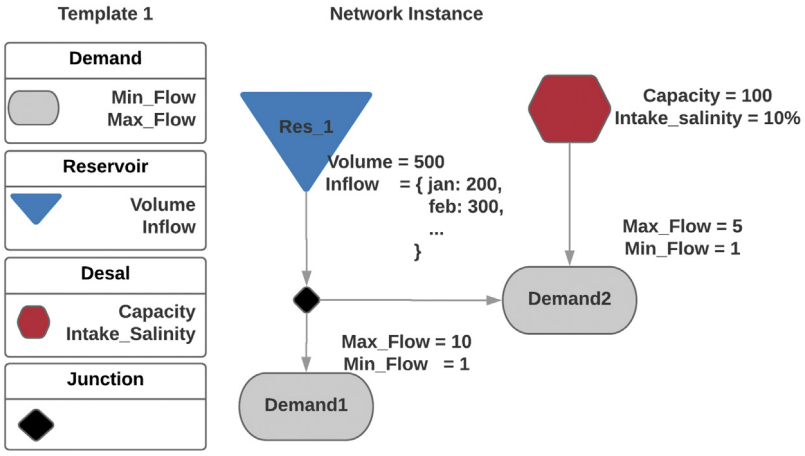

Fig. 5. Illustration of a template defining the Node types for a water resources model (left). Node types have attributes and a shape/colour. On the right is an instance of a Network which uses this template. Each Node instance has the appropriate Attributes, with a value.

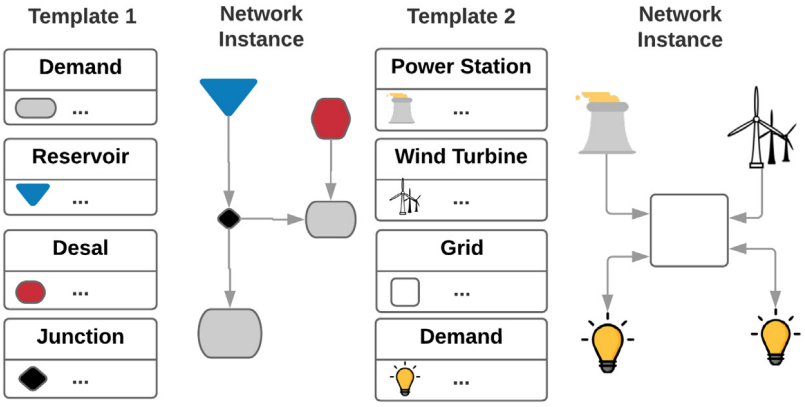

Fig. 6. Shows two networks stored in the same system (potentially the same Project), using different templates. As Hydra stores its data in a generalised way, it can support user-configurable network types from several different domains.

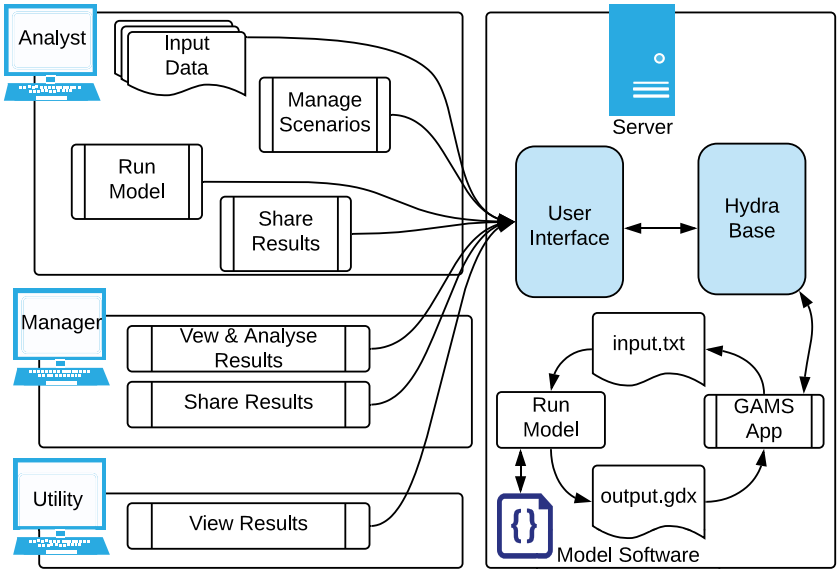

Fig. 7. An existing industrial deployment of Hydra. A water utility engages an analyst to input data, and perform model runs. Model input and output data are stored centrally, so the client has access at all times. Once an analysis is complete, the analyst shares results with her Manager, who checks the data before sharing with the client. Hydra resides on the server with a custom Web UI and the model. The model can be run from anywhere, and the results accessed easily.

et al., 1990) which relaxes the 'least cost' requirement of the model to return the n least cost alternatives within a specified threshold, for example within 2% of least cost. This enabled the water utility to explore a range of results which may perform better in criteria other than cost, such as environmental impact, resilience to drought etc. This additional requirement caused each model run to be significantly longer (ranging from the original 15 min run time to several hours). Running such models on a local workstation was infeasible, and for efficiency, multiple runs should be configured and deployed at the same time.

In addition to the MGA addition, the utility required that constraints be configurable, such as pre-requisites (must build x before y) and mutually exclusivities (can only build x or y but not both). To implement these constraints, Hydra's groups were used, where nodes were grouped by constraint type. The groups were then exported to the model as constraints using the client application.

The additional requirements of this project compared to prior EBSD projects were as follows:

- Make data and model accessible online to enable collaborative modelling.
- Allow the water utility to upload data and run scenarios independently.
- Configure the MGA model runs (e.g. the number of optimised configurations to save).
- Allow MGA results to be explored and downloaded for analysis.
- Allow multiple model runs to be launched concurrently.
- Make constraints user-configurable.

The software facilitated and enabled the following project roles:

1. The water utility seeking to run the model, and who must present results to the regulator.
2. An analyst either in the water company or a consultant who performs the actual model runs and analyses the results.
3. The developer of the model code, and specialist in EBSD and MGA. Provides support to the analyst in interpreting results and expanding the model.
4. The developer of the UI, Hydra and technical support. After initial development has little interaction with the project.

For this project Hydra was installed on an Amazon EC2 instance along with the database, GAMS client application and the GAMS model file. A User Interface for Hydra was deployed on an Amazon EC2 instance, connected to Hydra. Import and Export functions were developed in the UI to allow uploading and downloading of model data in the formats required by the water utility. The analyst received input data from multiple sources for existing and proposed infrastructure (the network topology), weather forecasts, cost information, and demand forecasts. A custom Excel client application compiled these sources into a single Hydra compatible Input JSON string, which was uploaded to Hydra using the 'add_network' function. Through the UI, the analyst ran the model, cloned the scenarios to tweak input data, such as interest rates or constraints, and re-ran the model, storing outputs in different scenarios. Once model runs were complete, the consultant was given access to the network using 'share_network'. The consultant and utility analysts then performed analyses on the model results before presenting them to the regulator.

By using Hydra in this instance, a transparent modelling workflow was enabled, with input data and modelling results available to both consultants and utility analysts via a web browser. Analysts could consolidate input and output data, and share results within a single platform, enabled by having a single shared place to select and refine data and manage model scenarios. The translation tool required for compatibility with GAMS is open-source (GPL3) and available on GitHub (Knox et al., 2011). Table 1 lists requirements for the project and the solution to those requirements provided either by Hydra directly or through web services generally.

The Hydra features highlighted in this case study are demonstrated in a worked example in Appendix A.1. The example illustrates how a model can be uploaded, run and shared through Hydra. The full code for this example is on GitHub. The worked example also demonstrates features used in the two supplementary case studies described in Appendices A.2 and A.3. These proposed case studies differ from this one by focussing on collaborative model development in a research setting and managing data privacy in an industrial setting respectively.

4. Discussion

This paper has presented Hydra Platform, also referred to as 'Hydra' for short, a data management software system which helps modellers collaboratively manage, share and analyse input and output data to network models. As models and modelling efforts expand in scope and importance within resource system decision-making processes, there is a growing need to share data and collaborate on models which require contributions from multiple people in different physical locations. The old paradigm of having the model on a user's local machine and using a file naming system, spreadsheets and email for data management is often too limiting to ensure reproducibility and consistency amongst collaborators.

Hydra addresses data management and sharing issues by using a web server which contains functions for storing network structures in a generic way. This allows the storage of multiple types of networks while remaining within the well-defined sphere of network simulation and optimisation. By focusing on network-based models, there is enough flexibility to support a cross-section of environmental modelling, while being focussed enough to provide a meaningful and enabling platform for these models.

The web server provides several features for creating and editing data, while supporting multiple degrees of user control. These range from limiting user roles ('User A cannot create networks') to a user being able to control access to specific aspects of their work ('User A can see my network, but not edit it'). Data within Hydra are by default only visible to the user who creates it, but a user can change this setting. By using the same data for model input, and using the same containers to store model outputs, there is consistency amongst users, which could improve understanding and reproducibility of results. The use of scenarios allows users to create multiple versions of a network, thus allowing different inputs (and associated outputs) to be compared and to avoid conflicts.

As multiple types of network can be stored in the same software system and shared among users, Hydra supports model integration. This can be achieved by linking two Networks together (for example, an energy Network and a water Network). Having run one model to completion, and stored the results, these results can be transferred to the other network, and used as input to a second model. This process is termed 'loose coupling', and is supported by Hydra with its 'attribute linking' process, which allows a node of one network to be connected to that of another. As outputs of one are immediately available as an input to the next, this approach reduce both the time needed to run an integrated model and errors in data transfer.

Our approach to facilitating this kind of model integration is use of a common generalised data store combined with translation tools. These tools create the appropriate input formats for each model and stores their outputs for further use or analysis. While this requires the researcher to translate data to and from the format of this centralised system, this enables them to make their tools available to others, reducing this burden on future modellers. Currently publicly available tools exist to translate network data to and from the following formats and modelling tools: GAMS, PYOMO, WaterML (via CUAHSI HIS), Microsoft Excel, CSV, and GIS Shapefiles.

To demonstrate that Hydra is a functioning system, we presented a case study of its use in an industrial setting. The case study describes an existing deployment of Hydra within a water utility. Hydra reduces

Table 1
Data management requirements of an industrial modelling project, and the solutions provided by Hydra and web services.

Requirement	Solution
Make data and model accessible online	Store data in Hydra, and give access via UI. In addition to enabling collaboration. This made model debugging, UI and data management upgrades fast, allowing for an efficient modelling workflow.
Upload data and run scenarios independently	Using Hydra, all actors could upload data and run models through a web UI. This resulted in an efficient and convenient data management solution.
Configure optimisation model formulation	By extending the GAMS client application and UI, the utility could configure which model formulation to run and how many results they wished to assess in a given MGA run.
Allow MGA results to be explored and downloaded for analysis.	Custom plotting and data input / output features in the UI supported this. Using a web UI allowed model runs to be monitored and results inspected and downloaded from anywhere.
Allow multiple model runs to be deployed concurrently and progress monitored.	Being deployed on the cloud (using AWS), the computing environment could be expanded run multiple instances of the GAMS client application concurrently, then scaled back as necessary.
Make constraints user configurable	Hydra's group system was used to create constraints, where a group could contain all the mutually exclusive options, for example.

reliance on model developers performing the model runs, and allows the water utility to run the model themselves. This case study illustrates how the model itself can be deployed remotely, connecting to Hydra, and managed through a custom user interface. Two additional proposed case studies are presented in the appendix, with example code showing how they can be achieved.

Early Hydra projects have pointed to needed future developments. These include lowering the barrier to entry for the use of Hydra by non-software developers, allowing Hydra to handle the storage of large datasets (multiple GB), and to improve the accessibility and quality (documentation, comments etc.) of existing client applications. As the number of deployments of Hydra increases, we will continue to develop its collaboration and security features, for example by allowing encryption of datasets by individual users to further ensure sensitive data is protected.

To encourage adoption of Hydra as a solution to data management needs, it has become clear that a UI would be helpful. We are building on the work presented here to create a generalised UI for managing networks and data. The UI will feature the ability to create Network types, and build Networks manually using these types. The UI will also feature the ability to run models, and activate client applications to integrate input and output of data. More than one UI using Hydra may emerge.

The data required to run some models, and the results produced by large models can become too large to store in Hydra — in the order of GB per dataset. In order to accommodate the potentially huge quantities of data, we are investigating using NoSQL approaches and big data techniques for accessing huge datasets. Hydra, then, will be used to store references to dataset locations, which reside externally.

As with the UI, a key to the adoption of Hydra is the availability of a range of pre-existing client applications where users can quickly get data into and out of the system with minimal effort. We intend to continue developing the existing client applications for GAMS, Matlab, Excel as well as documenting their development to reduce barriers to entry for prospective client developers.

Hydra does not contain the facility to directly make a dataset or network publicly available. This is an important improvement to support publications and to encourage scientific reproducibility. To this end, we are investigating the development of a connection to Hydro Share (Tarboton et al., 2013), an online repository of data and models for water resources and other fields.

All tools presented here are open-source and available on GitHub. We will continue developing documentation, examples, tests of Hydra and its applications with the goal of encouraging adoption.

5. Conclusions

The environmental systems models being developed are increasingly complex and there is an increasing need to manage data for such models to facilitate collaboration and sharing across users and locations, reduce errors, and promote scientific reproducibility and transparency.

We have presented Hydra Platform ('Hydra'), a web-based data management platform, specialising in storing network-based data. Some features of Hydra include:

- Open-source Python library with an extension enabling it to be deployed as a web service.
- Client libraries to simplify interaction with local and remote instances of Hydra.
- Networks of multiple types can be stored in the same instance of Hydra, allowing it to serve multiple model types concurrently, so long as all models use a node-link structure.
- Flexible data support: Built-in support for data types: data frames, timeseries, arrays, scalars and free text. Expandable to support other types.
- Client applications connect to Hydra to translate the data into the formats required by a modelling framework, and to send back results of model runs.
- Includes a user permission system and supports data ownership. This enables data confidentiality while providing the ability to share data amongst users.
- Multiple modellers can access the same inputs or model results, simplifying collaboration and minimising the likelihood of the errors common in traditional data sharing.

The case study described a successful deployment of Hydra in an industrial setting and highlighted some features of Hydra, such as its facility for sharing data between users, running models through client applications, and management of confidential data.

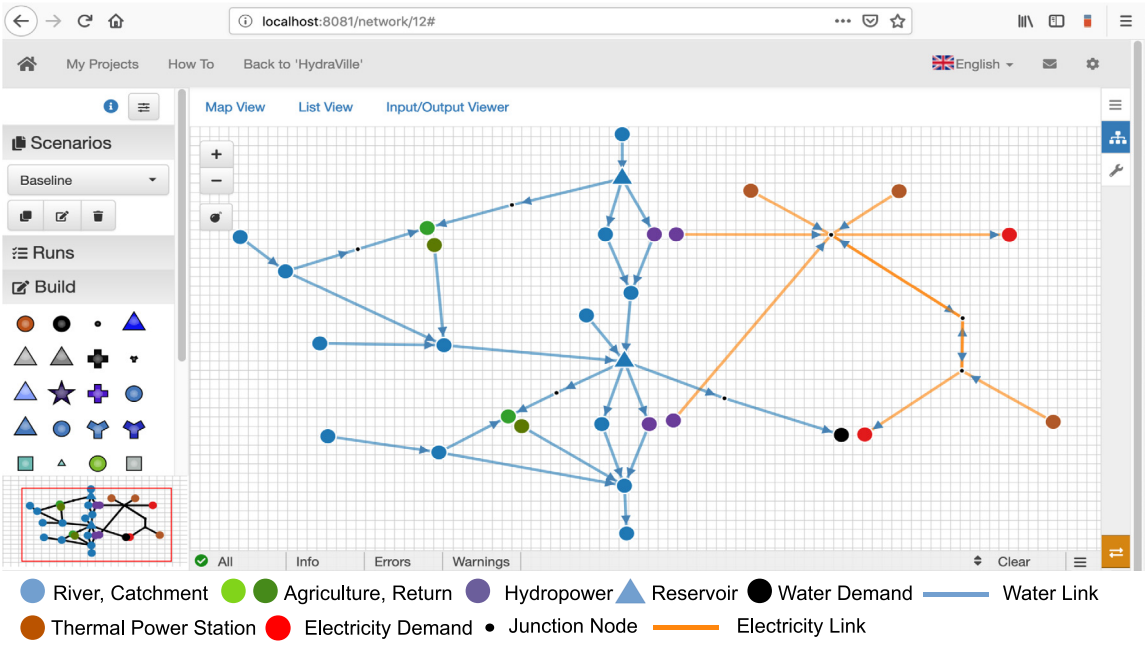

Fig. 8. Schematic of the water-energy system used in the worked example. The blue links illustrate the water system and orange links show the energy system. Hydropower nodes, represented in purple, join the systems. The model is shown in a web-based graphical interface, designed with Hydra as its data manager. This interface incorporates the flexibility of Hydra, such as its support for multiple model types, sharing between users and running different client applications. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

In future work we have proposed development of a user interface to simplify data interaction, an approach to managing 'big data', and a way to publish input data and model results.

Acknowledgements

Support for the software development was partially funded by the UK government through Innovate UK (File Reference number 101338). The UK Economic and Social Research Council (ESRC) provided support through the FutureDAMS project (ES/P011373/1). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors but not necessarily the funders.

Appendix

This appendix contains a worked example, which provides a compact demonstration of how the case study and two proposed case studies can be implemented with code. The worked example shows an application to an energy-water system simulator. The appendix also contains two further case studies which are partially designed and implemented showing further examples of Hydra's use which 1: supports collaborative development of a model across academic institutions (Appendix A.2) and 2: allows secure control of data sharing within a large industrial project (Appendix A.3). Finally, two small snippets of code are presented which demonstrate how to perform a basic hydra task of updating an existing network, and an illustration of the hydra server design.

A.1. Worked example

The following example shows a step-by-step process for creating and importing a network, running a model, sharing results while keeping input data private, and inspecting results visually. The demonstration involves simple scripts which perform actions within Hydra, such as creating projects, retrieving network details and so on. The reader is free and encouraged to amend these scripts. It is important to note that these scripts are for illustrative purposes only and as such are kept as simple as possible. They are not designed to provide a full, comprehensive example of how a client application should be built for Hydra. For example, almost no error checking is performed.

This example uses a modelling tool called 'Pywr', a python-based simulation system. A client application, called the 'hydra-pywr app' is used to load network data into Hydra, to run the model and to store results. To ensure maximum compatibility of this demo, the app is run through''pipenv', a virtual environment system used in Python.

The model in question simulates coupled water and energy systems and is applied to a small synthetic resource system network. The simulation model includes different water demands (irrigation, water supply and, hydropower), and energy demands. Energy demands are supplied by hydropower plants and conventional thermal power plants. A simple transmission system connects the different energy supplies to the energy demands. We note that this energy system model does not include any dispatch constraints. The model schematic is show in Fig. 8. It is shown in a web-based front-end interface for Hydra which is currently in development.

The formulation of the model itself is beyond the purposes of this example, and has been submitted for publication; data workflow is our focus here. A more detailed description of this model and the example models are available at the same location on GitHub.

The outputs of the code in this example have been shortened for clarity. The code contained in the scripts shown below can be found at:

https://github.com/UMWRG/hydra-pywr-demos.

```
# Step 1

Install Hydra.

There are multiple strategies for installing Hydra. The most common is using 'pypi' with
'pip install hydra-base'.

This example uses 'pipenv' as it helps with cross-compatibility and allows us to use avoid intruding on the
readers's local python installation.

   >>> pip install pipenv

   #Download all the correct dependencies from the Pipfile.
   #These contain hydra, pywr and the pywr app..

   #This ensures pywr builds with fewer dependencies.
   >>> export PYWR_BUILD_GLPK=true

   >>> pipenv install

   #Enter the virtual environment

   >>> pipenv shell

# Step 2
Create a project in Hydra. This will output a project ID to the terminal. The -u argument is the user,
in this case 'root'

   >>> python create_project.py -u root
   Project My New Project created with ID 2

# Step 3
In order to import a Pywr model to Hydra first a template must be registered with Hydra. The Pywr-Hydra
application includes functionality to register a template.

   >>> hydra-pywr template register

# Step 4
Using the pywr app, upload the network to Hydra. Choose one of the models to run. This uses a water-energy model
called 'hydraville':

   >>> hydra-pywr upload ../water-energy-embed/water-energy-embed.json --project=2
   Network 2 created

   >>> python get_network_details.py --network=2
   Name: 'Hydraville', 'ID': 2, 'Scenario ID': 2

# Step 5
Run the model

   >>> hydra-pywr run  --network-id=2 --scenario-id=2
   Model run successfully.

# Step 6
Create a second user

   >>> python create_user.py --user='user2' --password='secureME'
   User 2 created

# Step 7
Share the network with user 2, keeping 'costs' hidden.

   >>> python share_network.py --user='test' --hidden-attribute='cost'

# Step 8
Posing as the shared user, Inspect results (in this case, simulated_volume) with a simple graph. A timeseries
should appear for each of the nodes which have a simulated_volume attribute.

   >>> python plot_results.py --scenario=2 --attribute=simulated_volume ...
   ... -u user2 -p secureME

# Step 9
Posing as the shared user, Inspect results (in this case, simulated_volume) with a simple graph. Notice that
an error occurs, as user2 does not have permission to view the value of the cost. *This means that user 2 may be
unable to run the model, or the model may produce inconsistent results as the cost value will not be exported.*

   >>> python plot_results.py --scenario=2 --attribute=cost -u user2 -p secureME
   An error occurred retrieving scenario 48.
   Reason: Unable to view value for dataset cost
```

A.2. Proposed Case study 1

This proposed case study case was designed to manage the difficulties that can arise when working collaboratively on multiple models or multiple instances of input data.

Consider a model being developed by two researchers in different organisations. For example, two separate model formulations are developed of a water system, each model capable of being run with the same input data, but with differing outputs. This project requires keeping track of multiple evolving models and input data files.

To minimise the disorder of multiple researchers developing input data, and analysing output data of the different models, Hydra is used to store the data centrally, using the Project/Network structure for the separate model formulations, and its Scenario management capabilities for parameterising data.

In this application, the model is an optimisation-driven water resources simulation written in the Generic Algebraic Modelling System (GAMS), a scripting language for optimisation models. Different variations ('formulations') of the optimisation-drive simulation model are created and compared. The GAMS models require a single text input file containing the Network topology and Scenario data. The goal is to study the differences in results resulting from different formulations. The models share most data, but some parameters are particular to each optimisation model formulation.

To manage the inputs to each different model, and the many possible outputs, the two researchers, User A and User B deploy an instance of Hydra on an Amazon EC2 server, using a MySQL database and an Apache web server.

User A first puts the network and the baseline Scenario into an Excel file, in the format required by the Excel client application (Knox and Mohamed, 2018). She then uses an Excel client application to upload the file to Hydra. User A then shares the Network with user B, granting full edit access using the 'share_network' function. Both users can then clone the baseline Scenario, into, for example, 'Climate Change' within the Network using the 'clone_scenario' function. In this case, the input data is identical, but the Scenario can be used to store different outputs once the respective models have been run. This allows them to manage different instances of data relating to the same Network. Should they wish to work on fully separate Networks or even Projects, the 'clone_network' function could also be used.

To run the model, a GAMS client application is used which calls 'get_network' to retrieve the Network and Scenario in question. This is returned as a JSON string, which is then converted by the client into 'input.txt', ready to be imported in the GAMS code. Upon completion, the output — 'output.gdx' file is read by the client, converted back into the Hydra-friendly JSON format, and 'update_network' is called, with the new result values included.

Following the same procedure for each model results in three Scenarios stored against the Network, which can be compared, stored for later and shared with User B, which in turn can re-run the model with new input data. The translation tool required for compatibility with GAMS is open-source (GPL2) and available on GitHub (Knox et al., 2011).

A.3. Proposed Case study 2

Regional infrastructure planning increasingly encourages water companies to build regional (inter-utility) models to explore the use of common assets and transfers to cut costs and increase reliability (Matrosov et al., 2013; Zeff et al., 2014). Multiple data sources and need for transparency while maintaining confidentiality of cost data (a regulatory requirement) can become difficult without dedicated management of the data. This proposed case study presents an example of how Hydra could be used to improve data sharing and management in this project.

The stakeholders engage a consultant for modelling. An analyst within the consultancy runs the model. The model in this proposed case study is a single regional-scale simulation, written using the Python Water Resource Simulator (Pywr). This model accepts a single JSON input file, containing topology and scenario data.

Data on supplies, demands and costs is compiled by each water company. Each water company may view their own data, and overall results, but cost data from other utilities must be hidden from them (given the UK has regulated private utilities). Hydra can consolidate all the data so it can be input to the single regional model. The model run itself is done by a trusted third-party analyst who has permission to view all datasets, but each water company is only able to see its own (see Figs. 9 and 10).

To allow controlled data access to all parties, Hydra is deployed on an Amazon EC2 instance, which also hosts the Pywr software and a custom UI. This ensures the model can be run from any location, removing dependency on specific computers or institutions.

An analyst in the consultancy compiles the data from multiple sources into a JSON file, ensuring that sensitive datasets have a 'hidden' flag. This ensures nobody except the analyst can see the datasets. Using a JSON client application, the network is uploaded to the server. Next, the sensitive datasets are shared with the appropriate user groups, followed by the network being shared with all parties. When accessing the network, each stakeholder can see all non-hidden datasets, as well as the hidden datasets to which it has been granted access.

A.4. Examples

```
# Connects to Hydra.
import hydra_client import
    RemoteJSONConnection

#Create the connection
jc = RemoteJSONConnection(url='https://
    example.hydra.org.uk')
jc.connect(url='https://example.hydra.org.
    uk')
jc.login(username='root', password='
    i_am_root')

#Get the network
mynetwork = jc.get_network ( network_id =
    5 )

# Construct New Node
node = {'name': 'My New Node', 'type': '
    Reservoir'}

# Add Node to Network
mynetwork.nodes.append(node)

# Save Network
jc.update_network(mynetwork)
```

Listing 1: This describes how a user might retrieve and update a network. Hydra provides functions for managing smaller units (e.g. individual Nodes, Links, Datasets etc), but this illustrates a valid workflow. The full, operational code is available on GitHub (Knox, 2018)

Listing 1 shows pseudocode to create a RemoteJSONConnection, to enable a client to communicate with a remote installation of Hydra Server. Using a JSONConnection the client would attempt to connect to a local instance of Hydra Base. A client application developer can easily switch between a local and remote connection, with both connections

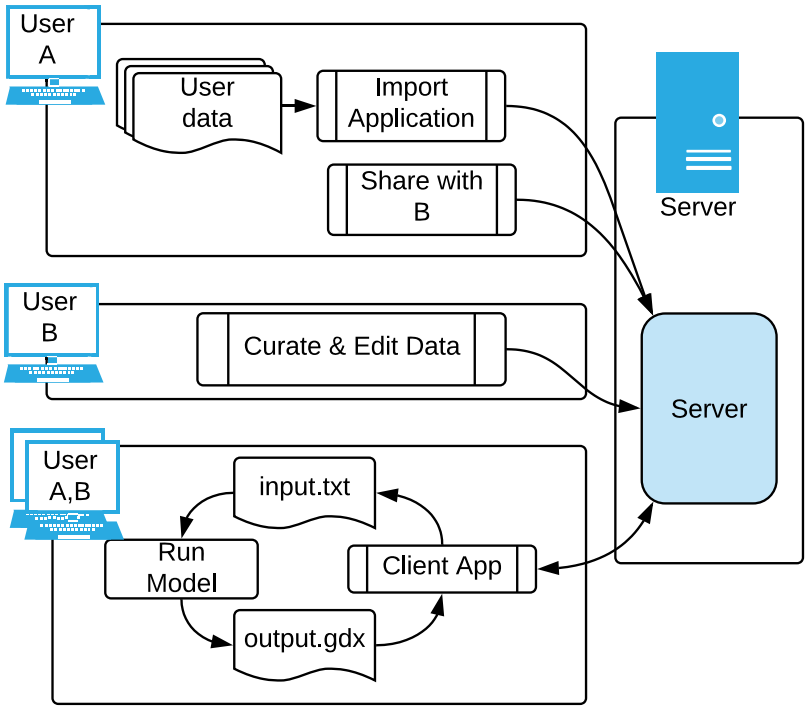

Fig. 9. Proposed Case Study 1: Two users collaboratively develop a model by interacting with a centralised server. User A imports the Network and shares it with User B. User B can then edit or validate the data. Both users can then run their model, with the results being stored centrally, eliminating any direct data transfer between users. Hydra's support for scenarios enables logical comparison and development of models For example, they can be compared with Hydra's compare_scenarios function.

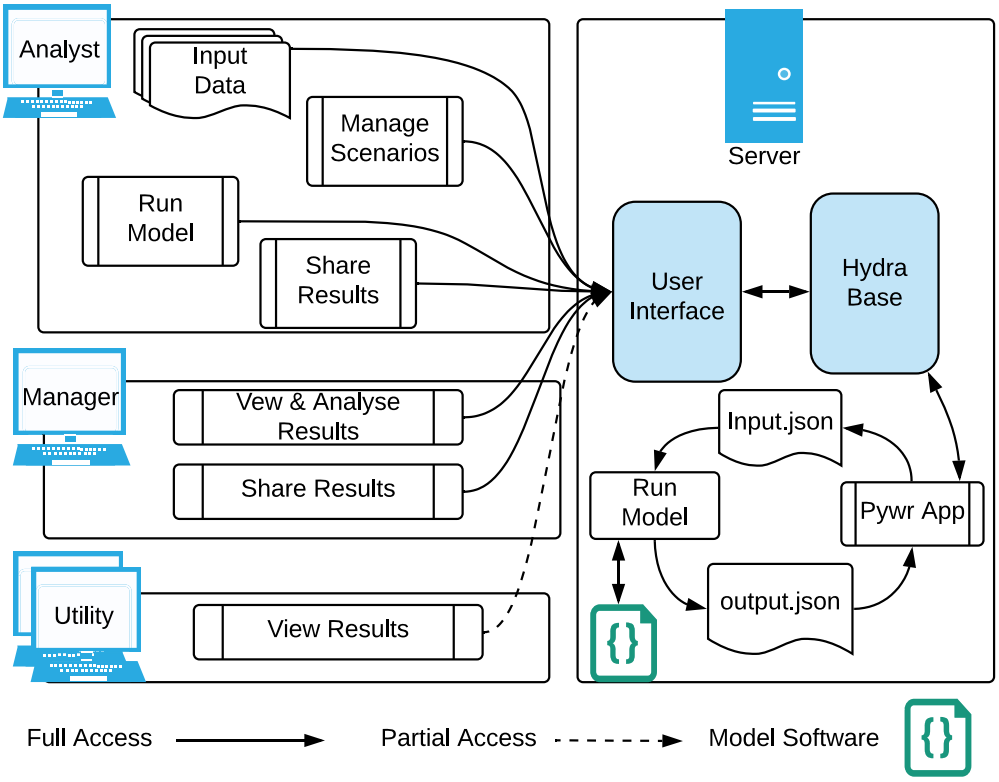

Fig. 10. Proposed Case Study 2: A proposed deployment of Hydra based on the experience of a large regional modelling project. Here, a regulatory agency would enforce strict control over who can see and edit certain datasets. A third-party analyst with permission to see all the data performs the data management and model runs. Each utility has access to the same Network (and therefore the model outputs), but with restricted access to some input data, based on the permission settings within Hydra.

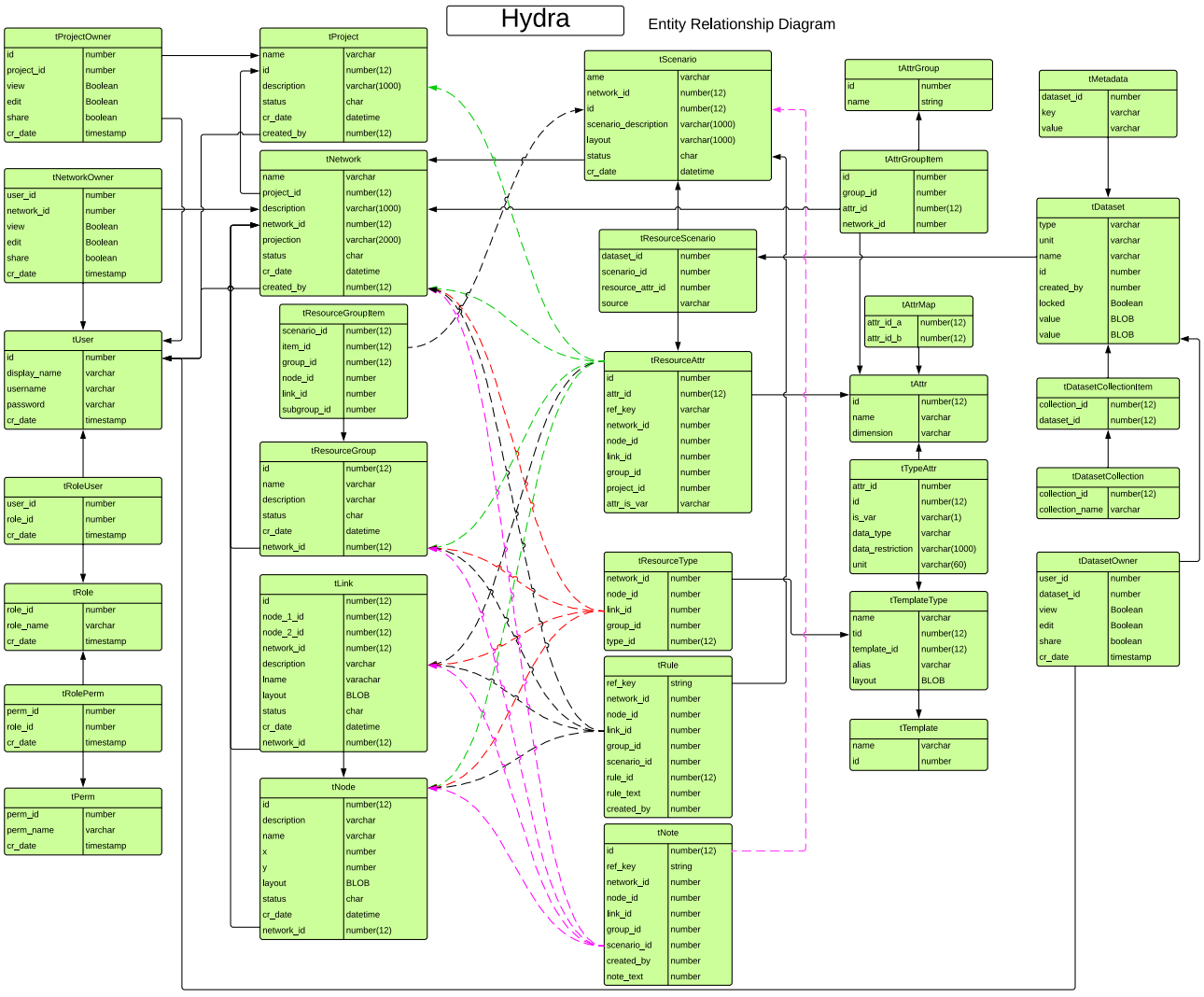

Fig. 11. The Hydra Platform database schema. The different colours and styles of the lines are to aid in distinguishing the connections. For the meaning of colours, refer to the simplified diagram, Fig. 4.

interacting in an identical fashion. This gives the developer flexibility for testing, or to allow the client to interact with different end points with little or no changes to the client-side code. The full, operational code is available on GitHub (Knox, 2018). Other examples can be found in the hydra-base repository, located on GitHub (Knox et al., 2018a)

```
import hydra_base as hb

@rpc(Integer, returns=Network) #define the
    input and output types
def get_network(network_id)
    return Network(hb.get_network(network_id)
        )
```

Listing 2: Pseudocode demonstrating a hydra server function. This illustrates that Hydra Server contains no logic and is used as an access portal for functions within Hydra Base.

Listing 2 demonstrates using pseudocode how the Web API works. The Spyne library exposes the 'get_network' function as a Remote Procedure Call (RPC), accepting an Integer as an argument. It then calls the 'get_network' function in Hydra Base, returning a 'Network' object. In this example, the Network object comes from a class defined by the web API which ensures the incoming and outgoing structures are valid. For every function in Hydra Base which should be exposed, there is an identical function in the server, with this workflow.

References

Ames, P.D., Horsburgh, J., Goodall, J., Whiteaker, T., Tarboton, D., Maidment, D., 2009. Introducing the open source CUAHSI hydrologic information system desktop application (HIS desktop). In: 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, pp. 4353–4359.

Blau, I., Caspi, A., 2009. What type of collaboration helps? Psychological ownership, perceived learning and outcome quality of collaboration using Google Docs. In: Proceedings of the Chais Conference on Instructional Technologies Research, vol. 12, pp. 48–55.

Brill, E.D., Flach, J.M., Hopkins, L.D., Ranjithan, S., 1990. Mga: a decision support system for complex, incompletely defined problems. IEEE Trans. Syst. Man Cybern. 20 (4), 745–757.

Buytaert, W., Baez, S., Bustamante, M., Dewulf, A., 2012. Web-based environmental simulation: Bridging the gap between scientific modeling and decision-making. Environ. Sci. Technol. 46 (4), 1971–1976, PMID: 22260091. http://dx.doi.org/10.1021/es2031278.

Carr, G., Blöschl, G., Loucks, D.P., 2012. Evaluating participation in water resource management: A review. Water Resour. Res. 48 (11), URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2011WR011662.

Cetinkaya, C., Fistikoglu, O., Fedra, K., Harmancioglu, N., 2008. Optimization methods applied for sustainable management of water-scarce basins. J. Hydroinform. 10.

Chalh, R., Bakkoury, Z., Technologies, C.D.O., 2015. Big data open platform for water resources management. URL https://ieeexplore.ieee.org/abstract/document/7336964/.

Chang, S.Y., Brill, E.D., Hopkins, L.D., 1983. Modeling to generate alternatives: A fuzzy approach. Fuzzy Sets Systems 9 (1), 137–151, URL http://www.sciencedirect.com/science/article/pii/S0165011483800141.

Eker, S., Kwakkel, J., 2018. Including robustness considerations in the search phase of many-objective robust decision making. Environ. Model. Softw. 105, 201–216, URL http://pure.iiasa.ac.at/id/eprint/15236/.

Folk, M., Heber, G., Koziol, Q., EDBT, t.E.P.P.o., 2011. An overview of the hdf5 technology suite and its applications. URL https://dl.acm.org/citation.cfm?id=1966900.

Goodall, L.J., Saint, D.K., Ercan, B.M., Briley, J.L., Murphy, S., You, H., DeLuca, C., Rood, B.R., 2013. Coupling climate and hydrological models: Interoperability through web services. Eviron. Modell. Softw. 46, 250–259, URL http://www.sciencedirect.com/science/article/pii/S136481521300090X.

Gregersen, J., Gijsbers, P., Westen, S., 2007. Openmi: Open modelling interface. J. Hydroinform. 9 (3), 175–191.

Harou, J.J., Pinte, D., Tilmant, A., Rosenberg, E.D., Rheinheimer, E.D., Hansen, K., Reed, M.P., Reynaud, A., Medellin-Azuara, J., Pulido-Velazquez, M., Matrosov, E., Padula, S., Zhu, T., 2010. An open-source model platform for water management that links models to a generic user-interface and data-manager. In: International Environmental Modelling and Software Society (iEMSs).

Herman, J.D., Reed, P.M., Zeff, H.B., Characklis, G.W., 2015. How should robustness be defined for water systems planning under change? J. Water Resour. Plann. Manag. 141 (10), 04015012, URL https://ascelibrary.org/doi/abs/10.1061/%28ASCE%29WR.1943-5452.0000509.

Huskova, I., Matrosov, E.S., Harou, J.J., Kasprzyk, J.R., Lambert, C., 2016. Screening robust water infrastructure investments and their trade-offs under global change: A London example. Global Environ. Change 41, 216–227. http://dx.doi.org/10.1016/j.gloenvcha.2016.10.007, URL http://www.sciencedirect.com/science/article/pii/S0959378016304216.

Jiang, P., Elag, M., Kumar, P., Peckham, D.S., Marini, L., Rui, L., 2017. A service-oriented architecture for coupling web service models using the basic model interface (BMI). Environ. Model. Softw. 92, 107–118.

Johnson, W., Williams, Q., 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.

Kasprzyk, J.R., Nataraj, S., Reed, P.M., Lempert, R.J., 2013. Many objective robust decision making for complex environmental systems undergoing change. Environ. Model. Softw. 42, 55–71, URL http://www.sciencedirect.com/science/article/pii/S1364815212003131.

Kelly, A.R., Jakeman, J.A., Barreteau, O., Borsuk, E.M., ElSawah, S., Hamilton, H.S., Henriksen, J.H., Kuikka, S., Maier, R.H., Rizzoli, E.A., Delden, v.H., Voinov, A.A., 2013. Selecting among five common modelling approaches for integrated environmental assessment and management. Eviron. Model. Softw. 47, 159–181, URL http://www.sciencedirect.com/science/article/pii/S1364815213001151.

Knox, S., 2018. JSON app. URL https://github.com/UMWRG/json-app.

Knox, S., Meier, P., Harou, J.J., 2014. Web service and plug-in architecture for flexibility and openness of environmental data sharing platforms. In: 7th Intl. Congress on Env. Modelling and Software, pp. 83–90. URL http://www.iemss.org/society/index.php/iemss-2014-proceedings.

Knox, S., Meier, P., Tomlinson, J., Slavin, P., Basolu, G., 2018a. Hydra base. URL https://github.com/hydraplatform.

Knox, S., Meier, P., Yoon, J., Harou, J., 2018b. A python framework for multi-agent simulation of networked resource systems. Environ. Model. Softw. 103, 16–28.

Knox, S., Mohamed, K., 2018. Excel app. URL https://github.com/UMWRG/ExcelApp.

Knox, S., Mohamed, K., Slavin, P., 2011. GAMS App. URL https://github.com/umwrg/gamsapp.

Kralisch, S., Böhm, B., Böhm, C., Busch, C., Fink, M., 2012. ILMS–a software platform for integrated environmental management. URL http://iemss.logismi.co/xmlui/handle/iemss/12373.

Kralisch, S., Krause, P., 2006. JAMS–A framework for natural resource model development and application. URL https://scholarsarchive.byu.edu/iemssconference/2006/all/9/.

Kuczera, G., 1997. Wathnet: Generalised Water Supply Headworks Simulation using Network Linear Programming. Department of Civil, Surveying and Environmental Engineering, University of Newcastle, Australia.

Laniak, F.G., Olchin, G., Goodall, J., Voinov, A., Hill, M., Glynn, P., Whelan, G., Geller, G., Quinn, N., Blind, M., Peckham, S., Reaney, S., Gaber, N., Kennedy, R., Hughes, A., 2013. Integrated environmental modeling: A vision and roadmap for the future. Eviron. Model. Softw. 39, 3–23, URL http://www.sciencedirect.com/science/article/pii/S1364815212002381.

Ltd, O.S.S., 2000. Aquator. URL http://www.oxscisoft.com/.

Maier, H., Kapelan, Z., Kasprzyk, J., Kollat, J., Matott, L., Cunha, M., Dandy, G., Gibbs, M., Keedwell, E., Marchi, A., Ostfeld, A., Savic, D., Solomatine, D., Vrugt, J., Zecchin, A., Minsker, B., Barbour, E., Kuczera, G., Pasha, F., Castelletti, A., Giuliani, M., Reed, P., 2014. Evolutionary algorithms and other metaheuristics in water resources: current status, research challenges and future directions. Environ. Model. Softw. 62, 271–299, URL http://www.sciencedirect.com/science/article/pii/S1364815214002679.

Marsh, B.C., Spiteri, J.R., Geosciences, J.P.C., 2018. Multi-objective unstructured triangular mesh generation for use in hydrological and land surface models. URL https://www.sciencedirect.com/science/article/pii/S0098300417309676.

Matrosov, E.S., Padula, S., Harou, J.J., 2013. Selecting portfolios of water supply and demand management strategies under uncertainty—Contrasting economic optimisation and 'robust decision making' approaches. Water Resour. Manag. 27 (4), 1123–1148. http://dx.doi.org/10.1007/s11269-012-0118-x.

McKinney, D.C., Cai, X., 2002. Linking GIS and water resources management models: an object-oriented method. Environ. Model. Softw. 17 (5), 413–425, URL http://www.sciencedirect.com/science/article/pii/S1364815202000154.

Meier, P., Knox, S., Harou, J.J., 2014. Linking water resource network models to an open data management platform. In: 7th Intl. Congress on Env. Modelling and Software, pp. 463–469. URL http://www.iemss.org/society/index.php/iemss-2014-proceedings.

Padula, S., Harou, J., Papageorgiou, L., Ji, Y., Ahmad, M., Hepworth, N., 2013. Least economic cost regional water supply planning ? optimising infrastructure investments and demand management for south east England?s 17.6 million people. Water Resour. Manag. 27 (15), 5017–5044. http://dx.doi.org/10.1007/s11269-013-0437-6.

Peckham, D.S., Hutton, H.E.W., Norris, B., 2013. A component-based approach to integrated modeling in the geosciences: the design of CSDMS. Comput. Geosci. 53, 3–12. http://dx.doi.org/10.1016/j.cageo.2012.04.002, URL http://www.sciencedirect.com/science/article/pii/S0098300412001252.

Reed, P., Hadka, D., Herman, J., Kasprzyk, J., Kollat, J., 2013. Evolutionary multiobjective optimization in water resources: The past, present, and future. Adv. Water Resour. 51, 438–456. http://dx.doi.org/10.1016/j.advwatres.2012.01.005, URL http://www.sciencedirect.com/science/article/pii/S0309170812000073.

Rew, R., Davis, G., 1990. Netcdf: an interface for scientific data access. IEEE Comput. Graph. Appl. 10 (4), 76–82.

Rosenthal, R.E., 2008. GAMS – A user's guide. URL http://www.un.org/en/development/desa/policy/mdg_workshops/eclac_training_mdgs/brookeetal2008_gamsusersguide.pdf.

Sieber, J., Swartz, C., Huber-Lee, H.A., 2005. Water Evaluation and Planning System (WEAP): User Guide. Stockholm Environment Institute, Boston.

Tarboton, G.D., Idaszak, R., Horsburgh, S.J., Ames, D., Goodall, L.J., Band, E.L., Merwade, V., Couch, A., Arrigo, J., Hooper, P.R., 2013. Hydroshare: an online, collaborative environment for the sharing of hydrologic data and models. In: AGU Fall Meeting Abstracts, vol. 1.

Valentine, D., Taylor, P., Zaslavsky, I., 2012. Waterml, an information standard for the ex- change of in-situ hydrological observations. In: EGU General Assembly Conference Abstracts, vol. 14. p. 13275.

van Bruggen, A., Nikolic, I., Kwakkel, J., 2019. Modeling with stakeholders for transformative change. Sustainability 11 (3), 825.

Vitolo, C., Elkhatib, Y., Reusser, D., Software, C.M.I., 2015. Web technologies for environmental big data. URL https://www.sciencedirect.com/science/article/pii/S1364815214002965.

Voinov, A., Kolagani, N., McCall, M.K., Glynn, P.D., Kragt, M.E., Ostermann, F.O., Pierce, S.A., Ramu, P., 2016. Modelling with stakeholders – Next generation. Environ. Model. Softw. 77, 196–220, URL http://www.sciencedirect.com/science/article/pii/S1364815215301055.

Zander, F., Kralisch, S., 2011. Environmental data management with the river basin information system. URL http://www.iwrms.uni-jena.de/fileadmin/Geoinformatik/projekte/RBIS/web/zander.pdf.

Zeff, H.B., Kasprzyk, J.R., Herman, J.D., Reed, P.M., Characklis, G.W., 2014. Navigating financial and supply reliability tradeoffs in regional drought management portfolios. Water Resour. Res. 50 (6), 4906–4923, URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2013WR015126.