Environmental Modelling & Software 103 (2018) 16–28

Contents lists available at ScienceDirect

Environmental Modelling & Software

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

A python framework for multi-agent simulation of networked resource systems

Stephen Knox [a], Philipp Meier [b], Jim Yoon [c], Julien J. Harou [a, d, *]

[a] School of Mechanical, Aerospace and Civil Engineering (MACE), The University of Manchester, Manchester, UK
[b] Eawag, Department of Surface Waters - Research and Management, Kastanienbaum, Switzerland
[c] Stanford University, Palo Alto, CA, USA
[d] Department of Civil, Environmental and Geomatic Engineering, University College London, London, UK

ARTICLE INFO

Article history:
Received 23 November 2017
Accepted 17 January 2018
Available online 15 February 2018

Keywords:
Modelling framework
Simulation
Python
Open source

ABSTRACT

Modelling managed resource systems can involve the integration of multiple software modules into a single codebase. These modules are often written by non-software specialists, using heterogeneous terminologies and modelling approaches. One approach to model integration is to use a central structure to which each external module connects. This common interface acts as an agreed mode of communication for all contributors. We propose the Python Network Simulation (Pynsim) Framework, an open-source library for building simulation models of networked systems. Pynsim's central structure is a network, but it also supports non-physical entities like organisational hierarchies. We present two case studies using Pynsim which demonstrate how its use can lead to flexible and maintainable simulation models. First is a multi-agent model simulating the hydrologic and human components of Jordan's water system. The second uses a multi-objective evolutionary algorithm to identify the best locations for new run-of-river power plants in Switzerland.

Software availability

Program title: Pynsim
Developer: Stephen Knox
Contact address: stephen.knox@manchester.ac.uk
Software Access: https://www.github.com/umwrg/pynsim
Year first available: 2015
Hardware required: Windows 7, 10 (tested), Linux, MacOSX
Program language: Python
Program size: 116k
Availability and cost: open source
License: GPL3

1. Introduction

The use of simulation to model managed environmental systems is well established (Buytaert et al., 2012; Loucks et al., 2005; Robinson, 2014; Sánchez, 2007; Thorp and Bronson, 2013), as is the need to integrate models from multiple disciplines in order to represent the complexities and interdependencies of environmental systems (Burroughs, 2007; Castilla-Rho et al., 2015; Knapen et al., 2013). Integrated modelling aims to combine models from multiple different disciplines such as ecology & sociology (Daloğlu et al., 2014), water resources & economics (Harou et al., 2009) and water resources & sociology (Barthel et al., 2008).

In addition to the integration of multiple disciplines, some models aim to enable decision making of individual actors within the modelled system. The dynamic interaction and communication between actors within the domain can be modelled using an agent-based approach (Schreinemachers and Berger, 2011). Agent-based modelling is an established methodology to resource modelling (Castilla-Rho et al., 2015; Kelly et al., 2013; Tesfatsion et al., 2017), and several toolkits and languages are available (Hiebeler et al., 1994; Luke et al., 2003; Collier et al., 2003; Tisue and Wilensky, 2004). Multi-Agent Based Simulation (MABS) is a widely used technique, with several examples of cross-disciplinary model integration (Ghazi et al., 2014; Bosse et al., 2013; Daloğlu et al., 2014).

A common approach to model integration is *component-based*

* Corresponding author. School of Mechanical, Aerospace and Civil Engineering (MACE), The University of Manchester, Manchester, UK
E-mail addresses: stephen.knox@manchester.ac.uk (S. Knox), julien.harou@manchester.ac.uk (J.J. Harou).

https://doi.org/10.1016/j.envsoft.2018.01.019

modelling, in which processes within an integrated model are represented by pluggable model components. These integrated models are known as Integrated Environmental Models or IEMs. Component-based models are often implemented as an Environmental Modelling Framework, which are standards or software systems used to build and integrate models around a single coherent structure. EMFs provide abstractions for defining the input and output of models, and the domains on which the models operate.

EMFs differ in the domain they apply to, ranging from the specific (Hill et al., 2004) to the general (David et al., 2013; Bernholdt et al., 2002), and in the degree to which they require the model to be altered (their invasiveness) (Lloyd et al., 2011; Dozier et al., 2016). The decision to integrate a model into an EMF therefore relies on how much the model itself must be modified in order to allow integration and on the value of its inclusion.

The increasing integration of models and the complexity of the software required to support advances in integrated modelling is hampered by the ability of model developers to create sustainable code-bases which are usable beyond the scope of an individual project (David et al., 2013). Development of such code-bases is often difficult to justify as academic projects tend to focus on scientific accuracy and expediency over software sustainability and scalability.

In their vision and roadmap for the future of IEM, Laniak et al. (2013) describe the need for the environmental modelling community to make software development and sharing more prominent, in addition to considering their work to be a part of a larger ecosystem.

One step towards this goal is to encourage the use of existing, open-source technologies which allow model developers to build components, and to publish them through well-established software repositories. This can only be achieved by building a structure for model development and integration which is attractive and simple enough to encourage uptake and involvement from the wider community.

Component-based modelling frameworks rely on a consistent underlying structure to which all components integrate. Some approaches use standards, where each model uses the framework to explicitly define the input and output parameters required to run the model (Gregersen et al., 2007). Others provide a common structure to which all models must comply; for example most water resource management modelling problems can be abstracted to networks of nodes and links (Letcher et al., 2007). Recent work on network modelling software generalises network structures, allowing them to be applied to more than one domain, like water resources, energy and transport and any other domain in which network structures are used (Harou et al., 2010; Knox et al., 2014; Meier et al., 2014). Such an abstraction pushes the responsibility of domain-specific representation and functionality to the model developers themselves, and in doing so can support models spanning multiple disciplines. A generic network representation could allow, for example, multiple networks stored within the same system to be combined by linking models – e.g. the output of a water resource model is fed into the input of an energy management model by specifying a commonality between two different networks (Lee et al., 2007), such as representing a hydroelectric dam which serves as both a water supply reservoir in a water model and a production source in the energy model.

Using a network structure to tie multiple models together also allows for the support of multi-agent behaviour, where the model components can not only act on the network as a whole, but where each node or combination of nodes within the network can execute code independently at each time-step.

The work presented in this paper is referred to as the Python Network Simulation (Pynsim) Framework, a Python package containing abstract classes which are designed to be extended using an object-oriented structure. Pynsim adopts a modular design, allowing multiple users to 'plug in' their code and uses a central network structure to provide a common interface, where each module interacts with the network, not directly with each other. It allows simulations to be performed, where multiple sub-models adhere to a common representation of a network of nodes and links. Rather than imposing any particular standard for model communication, Pynsim provides a basic structure upon which network-based simulations can be built, laying the groundwork for components of those simulations to be released as public Python libraries.

Pynsim is an object-oriented framework written in Python and attempts to build on the design of existing modelling frameworks. It aims to facilitate model integration, agent-based modelling and the use of a 'component-based' design where components can be added and removed with ease. In addition to incorporating a component-based model integration through the use of 'pluggable' modules, it also supports agent-based modelling by allowing each network element (node, link or institution) to execute code individually at run time. An institution allows Pynsim to represent organisational and other non-physical hierarchies by acting as a container for nodes, links and other institutions.

The contribution of this work is a component-based simulation framework designed specifically for networks, including representation of decision-making hierarchies and support for multi-agent modelling, and which is accessible as a standard Python library. Two case-studies demonstrate how Pynsim's features and associated modelling approach aided development and led to flexible and maintainable human-environment system simulation models.

This paper is structured as follows: Section 2 describes related work in integrated and agent-based modelling. Section 3 identifies features of related frameworks and then introduces Pynsim and its unique functionality. Implementation details of Pynsim are then presented in Section 4. Two case studies are presented in Section 5. Finally a discussion and concluding remarks are presented in Sections 6 and 7.

2. Related work

2.1. Modelling frameworks

Environmental Modelling Frameworks (EMFs) support modular development of integrated models through provision of libraries of core modules and reusable tools for common tasks, such as unit conversion, language interoperability, data manipulation, analysis and visualisation (Argent et al., 2006).

The Object Modelling System (OMS) is an open-source EMF which maintains the design principles of the earlier Modular Modelling System (MMS) (David et al., 2013; Leavesley et al., 2007). OMS has a facility to build simulations, allowing definition of time steps, parameters and models for environmental modelling. OMS is Java-based and offers a lightweight, non-invasive (the models themselves change very little, if at all) framework for integrating models together. This is done by using code annotation to describe the required model parameters and a simple scripting language to connect the models. OMS focusses on reducing the amount of code required to integrate models.

The Open Modelling Interface (OpenMI) (Gregersen et al., 2007) is a standard for the exchange of data between models at runtime (Castronova et al., 2013b). OpenMI is implemented as a set of object-oriented interface classes and supports the integration of models, GUIs and data sources, where each one can be developed independently, and then integrated as 'linkable components'

through use of these interfaces. Legacy models can be integrated to OpenMI through a 'wrapper' which runs the model code inside an OpenMI compliant linkable component. This minimises the changes required to the model code itself.

In addition to the standard itself, OpenMI provides a Software Development Kit (SDK) for Java and C# to simplify the creation of components and the OmiEd desktop application for graphically linking models. The OpenMI framework has been used in several research projects internationally for environmental modelling (Knapen et al., 2013; Goodall et al., 2011, 2013). In 2014, it was adopted as an official standard by the Open Geospatial Consortium (OGC).

The Community Surface Dynamics Modelling System (CSDMS) is a modelling framework comprising of two parts: The Basic Modelling Interface (BMI) and the Component Modelling Interface (CMI) (Peckham et al., 2013). BMI provides an object-oriented programming interface similar to OpenMI, which allows a model to 'plug into' the CMI framework. The CMI provides the ability to match models written in different programming languages, perform unit conversion, and provides a GUI to manage components. To be BMI-compliant, the model developer must implement three functions from a BMI main interface: `initialize` to set up the component, `update` where the main functionality is implemented, and which updates the state variables at each time step and `finalize`, which tears down the model, for instance deallocating memory or saving results. BMI is the basis of the lighter-weight EMELI (Experimental Modelling Environment for Linking and Interoperability), a modelling framework written in Python that was designed to allow the re-use of component models by using a standardised interface. Most recently EMELI has been extended as a web service, allowing it to integrate BMI-enabled web services, thereby making them language-agnostic. (Peckham, 2014; Jiang et al., 2017). Projects aimed at linking BMI to OpenMI are under way in an effort to create a more cohesive offering of integrated model frameworks (Goodall and Peckman, 2016).

A web-based approach is also used by the DMIF (Distributed Model Integration Framework) which connects models deployed on different hardware and software platforms using web services (Belete et al., 2017). The technical interoperability within DMIF was tested by connecting GAMS and NetLogo, the former an optimisation-based language and the latter an agent-based simulator. This shows a clear path of development towards combining agent-based simulations with external optimisation techniques, a capability offered in Pynsim.

2.2. Multi-agent modelling

Multi-Agent modelling is a popular approach to environmental modelling as it allows complex questions to be broken down into atomic parts, where the interaction of sub-components of a system are modelled independently. Bousquet et al. (1999) argue that the theory of multi-agent systems not only offers pertinent and powerful formalization tools, but also provides a better framework for computer simulations. Many different definitions of agent-based modelling exist and have been discussed extensively (Thiele et al., 2011; Castilla-Rho et al., 2015; Davidsson, 2000). We refer to an agent as something which 'takes input from the environment and produces actions as outputs that affect the environment but with incomplete information at the agent level and without a global control mechanism' (Wooldridge, 2009).

Several agent-based languages and toolkits are available and used in environmental modelling. These include NetLogo (Tisue and Wilensky, 2004), Repast (Galán et al., 2009), Jade (Pipattanasomporn et al., 2009), Agent Factory and nxsim, to name a few. Of these, NetLogo, Repast and Cormas have been used within environmental modelling, and are described here. Nxsim, being Python based, is also described.

NetLogo (Tisue and Wilensky, 2004) has been used for modelling social and environmental interactions in groundwater systems (Castilla-Rho et al., 2015). NetLogo is a generalised simulation environment, allowing agent-based simulations in several domains. NetLogo supports both grid-based and network-based agent connections. In addition to having a user interface, it provides a built-in scripting language for defining agent behaviour. For integrating with external models and for more advanced developers NetLogo provides extension libraries for integrating programming languages such as Java, R and MATLAB.

Repast is a suite of generic open-source simulation platforms, written in Java which has been used to model water management systems (Galán et al., 2009). Repast is aimed at both novice developers through its ReLogo platform, but can be used to build scalable, distributed simulations using Repast Symphony. Repast is a Java API, providing a suite of tools for running simulations and developed using a philosophy of "abstraction of simulation infrastructure, extensibility, and 'good enough' performance" (Collier, 2003). One of the approaches employed by Repast to simplify initial development is to incorporate Python scripting. Being an API, Repast passes the responsibility of code organisation to the developer, rather than prescribing a particular structure.

Cormas (Common-Pool Resources and Multi-Agent Systems) is a multi-agent platform, written in Smalltalk (Bousquet et al., 1998). Originally designed for modelling resource management, Cormas has been applied to several other areas using multi-agent simulation. Through its user interface, an environment of agents can be created, where each agent is either a subclass of a spatial agent, with geographic information but with limited ability to communicate, or a communicating agent with no spatial properties, but which can send messages to other communicating agents. These agents can then be programmed as necessary. This structure reflects the design presented by Bousquet et al. (1999), which describes the need for 'institutions' which can represent abstract hierarchies as well as spatial agents reflecting physical entities. Cormas has been applied to multiple environmental multi-agent systems such as simulating irrigation systems (Barreteau and Bousquet, 2000), river catchments (Becu et al., 2003) and social/ecological systems (Saqalli et al., 2010).

Nxsim is a Python library designed for network based agent-based modelling. Nxsim is designed to support large networks of a single type of agent. Nxsim is open source and based on the well known networkx (Hagbert et al., 2004) library to represent topology and on SimPy (Lünsdorf and Scherfke, 2013) for discrete time simulation.

3. Pynsim

A common theme with recent developments in model frameworks is the goal to abstract the complexity of the software infrastructure, with the assumption that many model developers have limited skills in software development and architecture (Müller, 2010; Tisue and Wilensky, 2004). By contrast, scripting has become a common feature of modelling, particularly the use of Python (Marta-Almeida et al., 2011; Thorp and Bronson, 2013).

As resource models become increasingly sophisticated (Castronova et al., 2013a; Ames et al., 2009), a flexible approach is becoming more relevant, where through a well-defined software architecture, modellers can define their own modules, write their own code and rapidly build resource network models. To keep up with modern technologies, the approach should allow for the use of components (Buahin and Horsburgh, 2015; Whelan et al., 2014) which can be interchanged and be able to represent both the physical and non-physical aspects of a domain.

Given an appropriate structure, scripting can become more accessible to modellers and the steep initial learning curve is outweighed by the flexibility gained and the potential of benefiting from a wider community of software developers. The HydroShare platform demonstrates that such a software development community already exists within the water resources community (Tarboton et al., 2013; Morsy et al., 2017).

Several technologies and solutions exist for integrated environmental modelling, summarised in Kelly et al. (2013), but many solutions tend to focus on one specific problem area or domain, even in a cross-disciplinary context. For example the agent-based framework EMLab (Chappin et al., 2017) focusses on energy and climate specifically. By using a simple basic structure capable of being extended, a more general framework could be used, widening the number of applications and models which can be integrated.

Pynsim (pronounced 'pinsim') is a modelling framework designed to allow building networked resource system simulators in a structured way. Pynsim uses an object-oriented approach to define the network, its properties and simulation behaviour. This approach allows the structure to be general and modular, meaning Pynsim can be applied to many areas of network modelling, and allows multiple behaviour-defining codes to be integrated into a single simulation.

Pynsim is a modelling framework in that it supports the integration of multiple modules acting on a central data structure, a network, where the network is effectively the means of communication between modules.

Pynsim takes several design cues from existing modelling frameworks (detailed in section 2) such as adopting object-oriented classes for structure, lightweight design, and a component-based architecture, but narrows the focus of application to the simulation of managed resource system networks. Pynsim does not attempt to formalise attribute matching, unit conversion or provide cross-language interoperability.

In designing Pynsim, best practices and design philosophies from related works led to a set of requirements for the new modelling framework. The framework should support the simulation of a network of agents. It should be generalised to cater for multiple domains and should allow model integration using object-oriented classes as the semantic glue. It should also provide a means to add and remove algorithms and processes from the simulation with ease. This framework should be a standard software library, therefore depending on external systems for visualisation, data analysis and management.

Pynsim's novelty is that it is a generic network simulation framework, written in Python, capable of supporting multi-agent modelling and representing the physical and hierarchical aspects of network-based systems.

The use of Python has several benefits:

- Python is already a recognised tool in resource modelling (Marta-Almeida et al., 2011; Fienen and Plant, 2015; Thorp and Bronson, 2013)
- Python scripts can be made OpenMI and BMI compliant, allowing broad integration into an established environmental modelling ecosystem (Bulatewicz et al., 2013).
- Python offers numerical and scientific libraries such as Pandas, NumPy and SciPy.
- In Pyomo, Python has a native optimisation language ready-made, allowing tightly-coupled model integration.
- Extensions to core libraries can be published and downloaded through the Python Package Index (PyPI).

Pynsim supports component-based modelling (Whelan et al., 2014; Buahin and Horsburgh, 2015), by virtue of its object-oriented structure and through its ability to configure and manage multiple engines (see Section 4). Additionally Pynsim is suitable for agent-based modelling (Wooldridge, 2009; Castilla-Rho et al., 2015) as each component of a network is capable of running its own decision making code. Pynsim does not impose a component-based or agent-based approach. Instead it provides the framework in which one or both of these approaches can be adopted. The Pynsim codebase is lightweight in that it has no dependencies on external libraries and does not present a significant computational overhead (see Section 6).

Definitions. Pynsim uses the following terminology to describe its elements. These mirror the class names used within the Pynsim code.

Module: A file or folder containing python definitions and statements.

Object: An instance of a class (as defined within a Module), from object-oriented design principles.

Network Component Type: The blueprint of a network component (node, link or the network itself) with a unique set of properties to represent a structural element in the network.

Network Component: An instance of a component type. Computation can take place within a component. In Pynsim, a component refers to the constituent parts of a network. A component is an object.

Engine: A piece of software that performs calculations i.e. a sub-model. An engine calls an algorithm or a collection of algorithms that simulates or optimises a process. Multiple engines can exist within a generic model and provide a means of controlling the sequence of processes.

Generic Model: A collection of defined component types and engines.

Model: An instance of a generic model, applied to a particular system. This includes the network topology but without data.

Simulation: A run or instance of a model with a particular set of model parameters, boundary and initial conditions. Also termed a *model run*.

3.1. Design

With Pynsim, a developer first creates a generic model; defining component types and engines, and their behaviour. Next a Model is created by defining instances of these component types. After applying input data to these components, the model can be run – a simulation.

Pynsim uses an object-oriented design, providing a set of abstract classes which must be extended to build a simulator (see Fig. 1). Building a model in Pynsim requires first creating subclasses of the base network components, which we call building a *Generic Model*.

These classes, illustrated in Fig. 1 are:

`Network, Node, Link, Institution` (a grouping of nodes and links) `Engines and a Simulator`. The abstract classes provided by Pynsim are designed to be extended using inheritance, a fundamental feature of Object Oriented Programming (OOP).

Inheritance is a mechanism for establishing *'is a'* relationships between objects, whereby an object is based on a higher-level object. In this way the more specific object (sub-class) exhibits the same behaviour and contains the same properties as that of the higher-level object (super-class), but with additional properties and behaviour.

The classes created for a generic model are what make it specific to a domain. In one domain a subclass of Node class may represent a power station, while in another, the class may represent a junction in a transport network or a group of farms in an agricultural region. The same principle applies to links, institutions and networks.

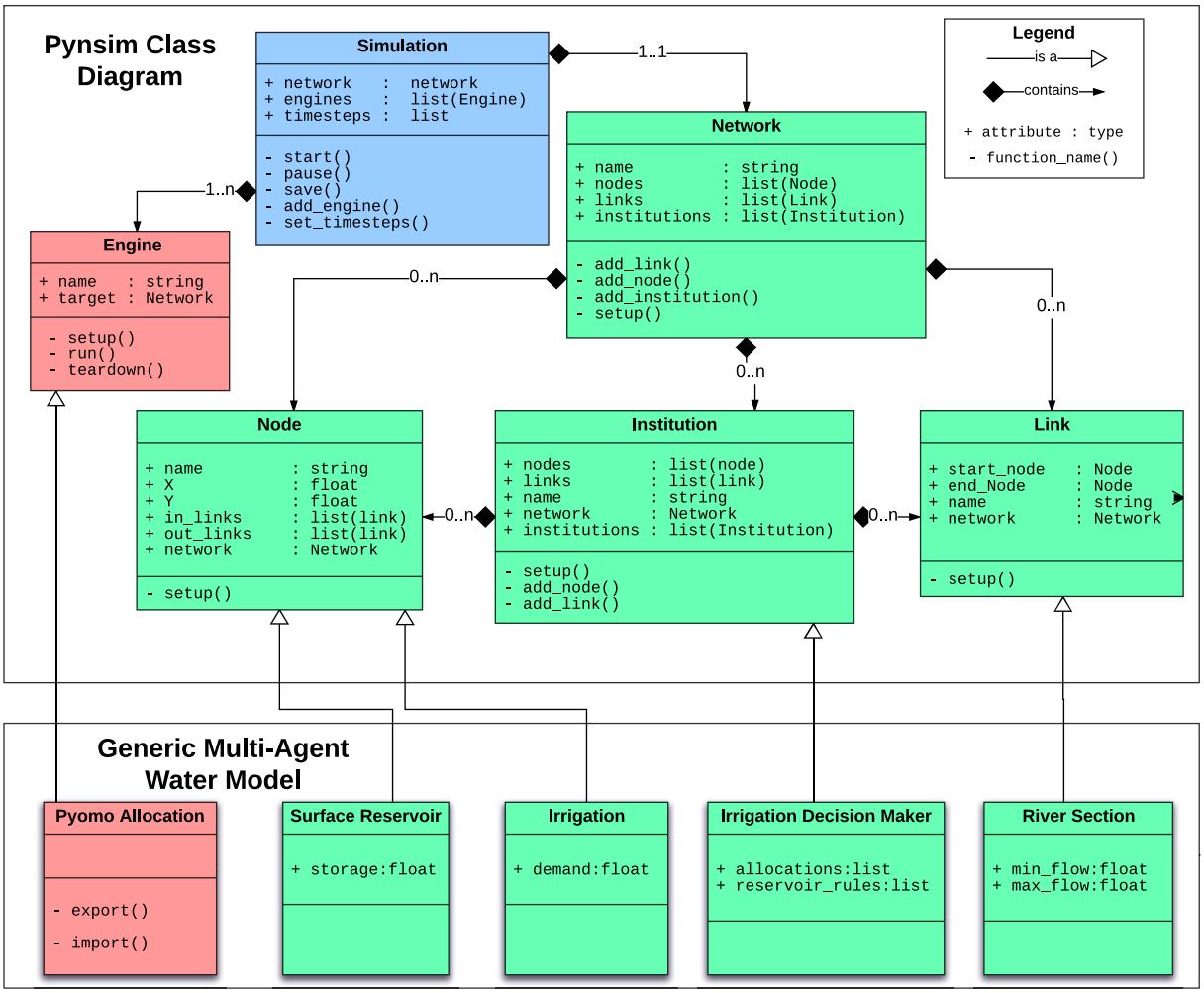

Fig. 1. A UML (Unified Modelling Language) diagram, describing the base components of Pynsim, and the extensions to these components, which make up a generic model. The classes in green are 'components', which make up the structure of a Pynsim network. The Engine class, is represented in pink, and can accept any component as a 'target' upon which to perform actions. The Simulator class acts as a container for the network and engines, defines time periods, and controls the model run's sequencing. By subclassing engines and components, a generic model is created. The bottom section of this image illustrates a generic model for a simple water resources system, using a single engine linked to a Pyomo optimisation (Hart et al., 2011). A similar example can be found in Pynsim's online documentation. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

One or more `Engines` must be defined by extending the Engine abstract class. Engines contain the generic model's logic, implemented either by activating an external sub-model or by implementing the logic directly in Python. It is the model developer's responsibility to write linking code to external processes, and therefore gives the developer control over the level of invasiveness required for implementing modifications. Engines are executed at each time step in a specific order, defining an explicit sequence of processes.

The last step in building the model is creating instances of the network components and engines. This is done in a Python script, adding the network and engines to the simulator, defining the model run's time-steps, assigning data to the network components, and then starting the simulation.

Pynsim handles the progression of time as a series of discrete time steps. The discretisation of time is defined before a simulation is run. As Pynsim iterates over the list of time steps, a pre-defined sequence is executed. The sequence of a Pynsim simulation is illustrated in Fig. 2 and occurs as follows:

1. Simulation starts
2. Simulation begins iterating over the user-defined time steps.
3. For each time step, t, each network component (node, link and institution) in the network sets itself up in preparation for being used in the engines. This is the *setup phase.*
4. At time step t, one or more *engines* perform an action on the network or part of the network.

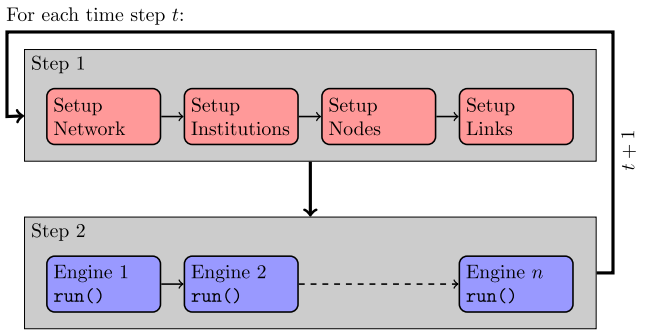

Fig. 2. The flow of operation of Pynsim. At each time step, the network goes through a setup phase, followed by the engines running in sequence.

5. At the end of time step t the network state is saved and the simulation continues.

The setup phase decouples the extraction of data for a given time-step from operations performed on the network. This makes engine code less complex as it only deals with the network at a single time step on each iteration.

3.2. Network components

Components comprise the topological structure of the network. They include the network itself, nodes, links and institutions. A component has some baseline properties like `name`, `description` and `component_type` but also supports user-defined attributes which represent the properties relevant to the model, and whose changes can be automatically recorded as the simulation progresses.

Pynsim components contain a `setup` function. At each time-step of the simulation, this function is called for every component individually.

The setup function has two roles: The first role is to allow each component to retrieve and set property values for the current time-step. This may involve querying external databases or other forms of exogenous data structures. Data calculated during previous time-steps may also be retrieved during setup. All the components in the network then set these properties locally.

Having set parameters locally, the second role of the setup function is to implement some more intelligent and autonomous decision making behaviour based on its current or past state. This is where a component can take the role of an 'agent', as the setup code is run on each component independently.

3.2.1. Nodes and links

Nodes and links provide the basic structure of a network. A node can represent any physical or abstract entity, provided it has a name and a coordinate. A link is defined by a name, a start node and an end node, and therefore cannot exist without nodes. Nodes and links act in the same way in that they are both 'set up' at the beginning of each time step and have a set of user-defined properties which an engine can access.

3.2.2. Institution

Institutions represent some grouping of other network components. This grouping can represent a physical similarity: 'all farms in a district' or a more abstract grouping, such as 'all power stations owned by owner X'. An institution can contain multiple nodes, links and other institutions. In addition to creating logical groupings for physical components, institutions can be used to implement hierarchical decision making, such as national and regional government policy. Being a network component like nodes and links, institutions are also 'set up' at each time step.

3.2.3. Network

The network is the container for all the nodes, links and institutions. The network class contains several functions to manage its sub-components such as functions for adding and removing nodes and links. In addition, a network has basic utility functions for visualising its topology and graphing properties over time. A model is associated with exactly one network (system topology).

3.2.4. Properties

In Pynsim, a list of `properties` are defined for each component type. A property is a 'state' (simulation) or 'decision' (optimisation) variable, as opposed to temporary or transient variables used during calculations or fixed parameters that do not change over the course of the model run. A property often stores a model result or a value required by another component in the current time-step or by itself in a future time-step.

At each time-step in a Pynsim simulation, the properties of an individual network component can be set either during the setup phase or by an engine. The types of properties that the components have are defined in their subclass. Pynsim provides a standardised way of defining properties, making it easy to distinguish a property pertinent to the model from other temporary variables or fixed parameters.

The primary reason for defining properties explicitly is that it allows Pynsim to track changes to these properties over time. The value of each property is stored in an internal history structure at the end of each time step. When the simulation is complete, the history of each property can be interrogated, allowing for in-depth post-processing. History can also be interrogated by components mid-simulation, allowing them to make decisions based on historical data. Pynsim provides a visualisation function, in which the changes to a property can be plotted over time.

3.3. Engines

An engine is a piece of software that performs calculations (Gregersen et al., 2007). Pynsim models must contain at least one engine, which performs an operation on a *target*. A target can be the network itself, as well as an institution in the network or an individual node. In Pynsim, an engine's operations can be performed fully within its own functions, using an external model, or by calling functions on components within the target.

Engines can be initialised at the start of the simulation, run at each time step, and finalised at the simulation's end, reflecting the 'intialise/update/finalise' design seen in other component-based architectures.

Pynsim supports multiple engines running sequentially. This allows the same network to be used in several sub-models, guaranteeing consistency of data and allowing sub-models focussing on different aspects of the problem area to be integrated into the same system.

The operation of the engine itself does not need to be written in Python. Code written in external languages may require the developer to convert the network into input files for the external model. It is here that more specialised frameworks for integration of sub-models could be used. When the external code is run, the results could be parsed in Python and saved. Using this approach, any new or legacy code can be integrated with Pynsim provided the developer writes the input and output parsing in the engine class. No automatic standardisation of units or attribute names is performed by Pynsim. It is the responsibility of the engine developers to ensure consistency of data and attribute names.

3.4. Simulator

The Pynsim simulator controls the model runs and acts as a container for the network and engines. The time steps for the model run are specified in the simulator. When the simulation starts, the simulator initialises each engine by calling its `initialise` function. It then iterates over these time steps, setting up the network (calling the `setup` function of each component), then running each engine in turn (calling the `run` function of each engine) for each time step. Finally, each engine is finalized by calling its `teardown` function.

4. Implementation

Pynsim provides a suite of abstract classes, with a simple code

structure, all of which can be (but don't necessarily need to be) extended.

There are three modules providing the base classes and functionality: *engine, component* and *simulator*. The *component* module contains the base classes for the network topology; `Network, Node, Link, and Institution`. These classes must be extended to define the elements of the network. The engine module contains the abstract class `Engine`. This class must be extended for the simulator to provide functionality. The simulator module holds the `Simulator` class. It is here that a simulation is controlled. The Simulator class contains simple features such as `start, add_engine, add_network` etc. A Simulator class must be instantiated, and the network and engine must be loaded into the Simulator. The time steps for the simulation must also be defined explicitly.

4.1. Time steps

The only rule for time steps is that they must be specified as an iterator. There is no restriction on what format the time step itself takes. Time steps are a simulator property, so must be set on the simulator object. By default time steps is an empty list, so if no time steps are specified before the simulation is started, Pynsim will return an error.

Valid time steps might be:

```
['mon','tue','wed','thu','fri']
```

Or

```
['2018/01/01', '2018/02/01', '2018/03/01', '2018/04/01']
```

When the simulation is started, it iterates over the time steps list, calling the setup functions and then running the engines for each time step. The network and each engine has access to the current time step, as well as the time step index. The time step index is the current numerical point in the time step list. For example, in the daily time steps above, 'wed' has a time step index of 2. By accessing the time step index, engines can be defined to perform certain processes at only specific time steps (e.g. an annual process that is only run at every 12 time steps for a monthly model).

4.2. Properties

Exogenous data can be set as a regular attribute on a component, but Pynsim also provides a means of allowing the model to define and then keep track of properties which may require analysis once the simulation has been completed. This is done using the `_properties` attribute. The underscore postfix is a python convention used to indicate an attribute which is private to an object and not meant to be accessed externally. `_properties` must be defined as an attribute of every component and is a Python dictionary. The keys of this dictionary are the component's properties, and the values are its defaults. Upon creation of each component, the contents of this dictionary are converted into object attributes, defaulted to the value specified in the dictionary.

For example, a class defining a reservoir might have a `water_level` property:

```
from pynsim import Node

class Reservoir(Node):
    _properties = {
        'water_level' : 100
    }
```

When instantiated, the reservoir object will have a water_level attribute. From the example above, an object `myRes, myRes.water_level` will yield 100.

This approach is taken so Pynsim can track certain properties while ignoring others. At the end of each time-step, every property which was defined in the '_properties' dictionary is recorded in an internal history structure. Every network component has a history property (`_history`). This is a Python dictionary, keyed on property name. The values of the dictionary are implemented as lists, which is appended at the end of each time-step with the current value of that property. By the end of the simulation, the change to every one of these properties has been stored and indexed by time-step. This allows for analysis of data changes throughout the simulation.

4.3. Engines

An engine is implemented as an abstract Python class with three functions: `initialise, run, teardown`. When implementing an engine, a developer must subclass this abstract class and implement at least the run function. This function is called during every time-step of the simulation. The initialise and teardown functions are called at the beginning and end of the simulation respectively. This approach matches the design pattern commonly used in object-oriented component modelling interfaces like BMI, OpenMI and OMS. The contents of an engine's functions are at the discretion of the developer, with the initialisation step typically used for loading external data, the run function used to propagate a sub-model in time, and teardown used for storing or parsing result data.

When instantiating an engine, a `target` must be specified. This represents the network or component of the network upon which the engine will perform its calculations. A target must be a subclass of the `component` type (i.e. Network, Node, Link, Institution).

The `run` function can be used in two ways: to perform a function which can be written natively in Python or to run an external program. In the latter case, the target must be converted into an input for the external program and the result of this program must be interpreted, with the resulting data set back on the target.

Engines are added to a simulator using the 'add_engine()' function, which appends the engine object to an iterator (`simulator.engines`) within the simulation. When the simulation begins, it iterates over this list, calling `engine.run()` on each engine, allowing for an explicit sequencing of model processes through the ordering of the engines list.

4.4. Data and result analysis

Pynsim provides several high-level features for data analysis, including drawing the topology of the network, analysing results data and identifying areas of poor performance in terms of run time. We briefly detail these features here.

Pynsim records the changes to the properties of components at each time step. By calling a component's `get_history(my_property)` function, the value of a specified property can be retrieved for each time step.

A simple suite of visualisation tools are provided to allow a user to graph changes to a property over time. This is implemented in the `network.plot(my\_property)` function and takes a property name as an argument. This inbuilt data analysis functionality is designed to provide quick, high-level information to developers which may be extended as necessary, rather than as an in-depth analysis tool like that presented by (White et al., 2016) and (Jin et al., 2017). No assumption is made about what kind of analysis a user might require so more detailed analysis is left to the developer to perform.

Visualisations are produced using matplotlib, but Pynsim does not rely on matplotlib being installed to run, only when the relevant graphical functions are called. This maintains Pynsim's lightweight but readily-expandable functionality, while availing of existing Python libraries.

In addition to data analysis, Pynsim provides analyses on computational performance. The time taken to run each engine and each setup function can be recorded by setting the `record_time` argument of the simulator to True. Timings can be visualised by calling the `simulator.plot_timing()` function. This function provides a graph of the cumulative time taken for the setup function of every resource in the network and for each engine. For more detailed performance analysis, the timings of each node, link and institutions can also be graphed individually using the network object's `plot_timing()` function. Likewise, if the simulator contains multiple engines, the performance of each can be plotted using the simulator's `plot_engine_timing()` function. These tools allow a developer to identify bottlenecks quickly without having to use external profiling tools. As with the data analysis, the philosophy of Pynsim is to provide high-level analysis tools to identify obvious irregularities, but detailed analysis is left to the developer.

5. Case studies

This section presents two case studies in which Pynsim has been used. They highlight different aspects of Pynsim, demonstrating its use in multiple applications.

5.1. The Jordan water project

The Jordan Water Project (JWP) is a collaborative, interdisciplinary research effort focused on the development of a multi-agent hydroeconomic model for the evaluation of policy interventions in Jordan, which ranks as one of the most vulnerable countries in the world in terms of freshwater supply (Padowski et al., 2015; Rajsekhar and Gorelick, 2017). The goal of the project is to enhance Jordan's long-term water security in the face of climate and socioeconomic change. The central feature of the JWP is an integrated model which couples spatially explicit representation of hydrologic systems with multi-agent representation of water allocation and use at both institutional and consumer levels of human decision-making. Through Pynsim's component and engine architecture, the model adopts a modular structure, with individual modules developed and calibrated independently and subsequently linked through the identification of important interactions and feedbacks in the system. The overarching model structure is simulation-based (i.e., non-optimization), though subsystem level optimization algorithms are adopted for specific modules.

The project team includes researchers from a variety of disciplines (hydrology, water resource systems analysis, economics, geography), each responsible for development of specific modules. Given the interdisciplinary makeup of the team and the complexity of the model structure, the JWP serves as an example for utilising the Pynsim framework in enhancing complex human-natural model development. While an in-depth analysis of the model is beyond the scope of this paper, we illustrate some modules of the Jordan Water Project that highlight the use and value of Pynsim, focussing on a cross-section of the physical, social and institutional aspects of the system.

The *groundwater system* is represented through a set of approximately 200 groundwater nodes. Each node uses a Pynsim property which tracks the groundwater lift from the water table to the ground surface. A response function is written using a Pynsim engine, which consolidates pumping information from the human agents, then calculates the response of the groundwater system at every groundwater node using a response matrix (Maddock, 1972). These engines use external drawdown tables, pre-processed from hundreds of MODFLOW simulations, illustrating how a Pynsim engine is linked to an external source to perform calculations or access information.

At each time period, *households*, implemented as Pynsim nodes, request and purchase water from multiple water sources (institutional allocators and private water tankers) in response to prices and availability of the various sources and in accordance with a unique demand curve (Klassert et al., 2015). Households make initial demand calculations independently based on their current circumstance, executed during the setup phase of the time step. The amount of water purchased, the price of purchase, and the consumer surplus of every household node is tracked for all model time periods using Pynsim's properties. These are later used for model validation and analysis.

Private tankers act as an informal water market between farmers (who own private groundwater wells) and urban households. The private tanker water market is implemented as an institution, and uses a market algorithm that matches willing farm sellers with urban buyers, factoring in transportation cost based upon distance between any two farm and urban nodes. The tanker market uses the flexibility of the Pynsim institution class to model non-spatial abstractions such as a water market applied to resource networks.

Farms, implemented as Pynsim nodes, attempt to maximize profit under a set of physical and behavioural constraints, many of which are imposed by other Pynsim components in the system. Farms use the dynamic groundwater lift stored in the groundwater nodes as input to their optimisation calculation, which is performed through an engine that calls an external GAMS model run. Here the support for agent-based decision making integrated with external optimization solvers is demonstrated, with each farm needing to dynamically decide on whether to sell its water or use it for irrigation.

Organizational-level allocator agents are implemented as Pynsim institution classes. In contrast to the water user agents which are implemented as spatial nodes, the institutional agents are entities that contain a subset of the nodes and links in the system and make decisions over them. In the Jordan model, these institutions represent real world agencies which allocate water from their supply nodes to their water user nodes. Each institution uses a linear programming formulation that minimizes water deficit (the difference between allocated water volume and baseline demand) in determining current and projected water allocations to the water user nodes. Pynsim's ability to represent non-physical hierarchies makes possible the deployment of such institutional agents.

The simulation is run for a 10-year historical period and a 50-year future period for a diverse set of scenario and intervention combinations. Pynsim tracks component properties of interest (e.g. groundwater head levels at each groundwater node, water purchases at each water user node, etc.) over time, which are used for system performance evaluation. Pynsim provides a common framework for the development of a complex integrated model, streamlining software architecture design and easing module compatibility, particularly in a collaborative, interdisciplinary model development environment. The framework provides a common architecture for development, while also allowing for a high degree of flexibility for component extension, as demonstrated by the ability to use Pynsim components to simulate a wide range of processes, from physical hydrologic flow to an abstracted private water tanker market. The component-based structure of Pynsim allowed the code-base to be broken up into manageable parts and for each developer to build and test their engine in

isolation. This allowed new engines or updates to be integrated in a structured and controlled way.

For the Jordan Water Project, the framework serves as a means to standardize model conceptualization and development among a group of researchers from a variety of disciplines each approaching environmental modelling from a unique disciplinary and methodological perspective.

5.2. Investment planning for run-of-river power plants

With an increasing electricity demand and the decision to phase out nuclear power plants, the Swiss energy strategy plans for an increase in hydropower production. The already high degree of exploited potential shifts the focus of hydropower expansion to small run-of-river power plants. As riverine ecosystems are already exposed to multiple stressors affecting ecosystem functioning, the impact of disrupted network connectivity within a river network should be included in the planning process (Altermatt, 2013).

This case study presents a tool to evaluate Pareto-optimal configurations of run-of-river power plants considering their effect on river network connectivity. The positioning of run-of-river hydropower plants within a river network is formulated as a multi-objective problem and solved with a multi-objective evolutionary algorithm (MOEA). Different objectives are considered, such as total electricity production, investment cost, flow deficit and network connectivity, each of which is implemented as a separate engine.

Pareto-optimal extension options are derived by linking a simulation model to the MOEA, in this case BorgMOEA (Hadka and Reed, 2013), searching for optimal decision variable sets based on multiple objective values calculated by engines. This requires the evaluation of different potential configurations, running the simulation multiple times.

Before evaluating one specific configuration, the network topology needs to be defined based on decision variables provided by the optimisation algorithm (See Fig. 3). This demonstrates the ability of Pynsim to build a network topology through code, based on exogenous data sources. To save computation time, those parts of the network that do not change during the optimisation process (i.e. the river itself), are generated beforehand. This allows reuse of the basic network structure including data without the need for instantiating a new network object for each simulation run. Parameter values provided by the optimisation algorithm can be set on an existing network, thanks to the object-oriented design of Pynsim.

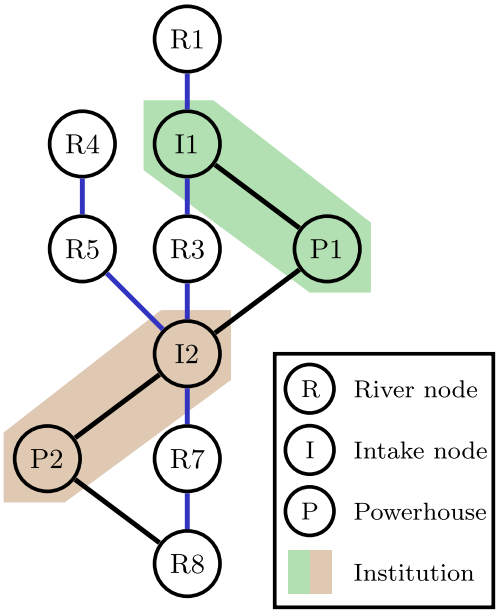

Fig. 3. Schematic of a simple river network including two run-of-river hydropower plants. A hydropower operator which operates the water intake, power house and outlet forms an institution.

In order to represent the structures of a power plant as a whole, the water intake, the powerhouse and corresponding pipes are combined as institutions. The investment cost is calculated at the institutional level. Institutions provide an intermediate hierarchy level well suited for representing institutional units in a model.

Multiple engines are used. Before each objective is calculated, a flow routing engine sets the discharges on each link in the whole network.

The use of an evolutionary optimisation algorithm imposes the need for a fast simulation model. Pynsim supports fast simulations by leaving the model developer in control over what code is executed. Pynsim's use of engines allows for enough granularity to include or exclude certain calculations with little effort. This makes developing the model easier and less error-prone.

6. Discussion

Pynsim was applied successfully to two case studies with different requirements, and the lessons learned from these applications and in the continued development of Pynsim have highlighted a number of benefits and limitations.

For the Jordan Water Project, Pynsim provided a common framework for the development of a complex integrated model, serving as a foundation to standardize model conceptualization and development among a group of researchers from a variety of disciplines (hydrology, systems analysis, economics, geography), each approaching environmental modelling from a unique disciplinary and methodological perspective. This foundation is based on Pynsim's object-oriented design and provision of a common network through which sub-modules can interact. Pynsim's support for institutions, hierarchical groupings of nodes and links, were used to represent governance structures and their decision making processes. Additionally, the deployment of engines, institutions and agent nodes allowed for the use of optimisation in addition to rule-based agent decision-making processes. Pynsim's property history structures were relied on extensively for both processing of results as well as in agent's decision-making processes. Finally, Pynsim's distinction of tracked properties for each agent simplified data management by providing a system to focus on variables of interest.

In the Swiss power plant project, the benefit of Pynsim's object-oriented structure is demonstrated. In this case, the modeller was an experienced Python developer, so Pynsim's online documentation and availability through pypi were factors in choosing it for this project. These are often considered a necessity for uptake of Python libraries. Pynsim's institution class is used for the Swiss model to provide grouping for a set of interdependent nodes rather than representing an organisational entity as in the Jordan project. Linking to a Multi-Objective Evolutionary Algorithm (MOEA) is achieved using one of Python's freely available libraries. Pynsim's object-oriented design is used because optimising the location of new hydropower plants required a network configuration which could be updated easily through code.

The primary limitations of Pynsim can be broadly split into two categories: performance (how Pynsim's overhead affects run times) and its level of formalisation and standardization.

Being written in Python, an interpreted scripting language, Pynsim has an inherent drawback in performance compared to compiled languages such as C or C++. Our experience is that the

benefits of ease-of-use and accessibility outweighs potential limitations of computational performance. The two case-studies demonstrated that computational efficiency of Pynsim engines have the most impact on model run durations. The inherent drawback therefore is not necessarily in Pynsim but in the engine and component setup code, which is the responsibility of the developer to streamline. Fig. 4 shows the overhead of Pynsim as a function of the number of time steps and the number of components in the network. In these test cases, a single empty engine is used with increasing numbers of nodes containing empty setup functions and an increasing number of time-steps. The linear increase indicates stable performance.

While the general network structure and flexible approach to integration makes Pynsim applicable to several areas, one could argue that Pynsim is *too* flexible, affording model developers too much freedom. The responsibility for non-Python experts to build an entire Python-based structure for a simulator may be infeasible for some developers. We have endeavoured for Pynsim to strike an effective balance between flexibility and structure, as demonstrated in the Jordan Water Project. Pynsim comes with extensive online documentation and includes several examples, but even with this support we are aware that some model developers will be apprehensive to build models from scratch in Python. This is where we anticipate that the model development community can come together to build Pynsim extensions for their areas of expertise, with libraries of component types and standardised ways of importing and exporting data to common external formats.

Pynsim is under continual development, with ongoing feedback of application developers taken into account. Some notable areas of planned Pynsim development are integration with existing model development platforms, improvement of documentation, and formalisation of how add-on packages can be used and published.

Improving efficiency is a constant goal, such as streamlining storage of the history of component properties – (e.g. only saving the values when they change to reduce memory usage). In addition, the ability to easily run multiple engines in parallel would allow developers to maximise their computing resources. The online documentation would benefit from more examples, including those of greater technical complexity. This is an area which needs continuous improvement. Finally, compliance is a goal, with efforts currently underway to make engines compatible with the BMI standard. While out of scope of current Pynsim develpment, the availability of Python libraries make BMI integration feasible, with OpenMI compliance a longer term goal, when support for OpenMI in Python becomes more mature. OpenMI compliance would give model developers within Pynsim access to an existing community of models in addition to working towards a common future vision where model frameworks are more interoperable (Laniak et al., 2013).

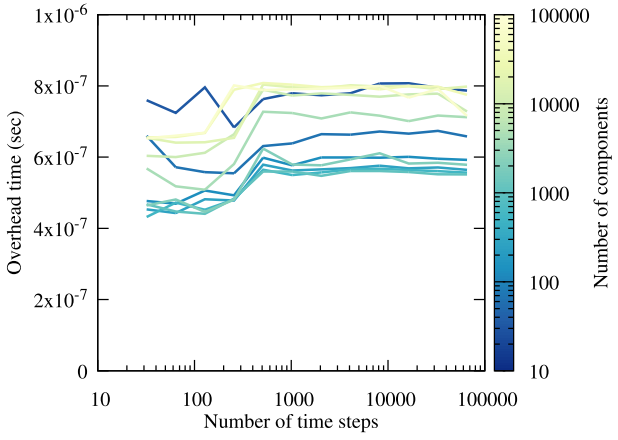

Fig. 4. Total overhead time of Pynsim as a function of the number of components and the number of time steps. The small variation in time across the spectrum of time steps indicates stable computational performance as time steps increase.

7. Conclusions

As many models of environmental resource systems benefit from an integrated and multidisciplinary multi-actor representation of the modelled domain, integrating specialised models at runtime is increasingly useful. There are multiple ways of performing model integration, from adhering to static input/output standards to directly developing components around a single software structure – 'component-based modelling'. Achieving a sustainable, reusable model can be a challenge, as most model developers do not have a formal background in software development and require a framework which is initially accessible, but which can be extended to support sophisticated scientific aims and methods. The Jordan Water Project is an example of this situation.

We presented Pynsim, a Python library designed to aid in the rapid development of customised networked resource system simulators. Pynsim uses a network structure as the common data representation and employs a modular design where external sub-models, or 'Engines', interact with the network instead of directly with each other. Engines take the data they need from the network, perform a calculation either directly in Python or by calling an external model or service, and then save results back on the network's components. It is within these network components that input data and results are expected to be stored. As a simulation progresses, Pynsim records changes to properties of the nodes and links, allowing for post-processing of data and analysis. Using the network as the central data storage structure ensures a common means of communication between engines.

A setup function is called on each component individually at each time-step, and each component object has access to current and historical simulation data. This approach allows components to act as independent agents within the simulation.

In addition to nodes and links, a Pynsim network can contain institutions; groupings of nodes and links, allowing Pynsim to support representation of non-physical hierarchies such as government institutions or social groups. The JWP used institutions to represent organisational hierarchies, while the Swiss model used institutions to represent an interdependent set of nodes.

The practical application of Pynsim is demonstrated by two case-studies, the details of which are summarised in Table 1. The Jordan Water Project (JWP) benefitted from Pynsim's component-based architecture as multiple sub-models were integrated into a single multi-agent model of Jordan's water resources system. The JWP was developed by a diverse group of scientists, none of whom had any formal prior software development experience. Using Pynsim, a modular code base was produced which streamlined updates, testing and integration. Several external data sources and models were used by this model, all connected using Pynsim's Engines. Pynsim's support for agents was used for independent decision making of some nodes, while its institutions allowed the representation of hierarchical decision processes.

Pynsim's flexibility is demonstrated by the second case study in which the network's topology is dynamically created from an external source and then altered while searching for the optimal placement of new power plants on an existing river network. This project used a multi-objective evolutionary algorithm, illustrating how Pynsim can be connected to an external processing source. Through institutions, groupings of interconnecting nodes could be referred to as one entity.

Table 1
The two case studies have different requirements, structurally and technically. This table illustrates the use of Pysnim in these projects, summarising how its features are used. Pynsim's internal structures are network-based, but these case-studies use them differently, one loading a large static network on initialisation, while the other alters the network's topology. Both use institutions, one to represent governance hierarchies, while the other puts interdependent nodes into groupings to represent a single entity. Both studies use engines, with the Jordan case study relying on them to break up the code structure, so each of its developers can work independently. Engines in both studies connect to external models to perform calculations. The Jordan case study is an agent-based model, using Pynsim to execute code within each node and institution during each time-step.

Pynsim Feature	Jordan Case Study	Swiss Case Study
Object-oriented design	Provided a common structure and interface to support the integration of modules developed independently by a multi-disciplinary team; Provided an efficient software framework to organize 1000s of nodes and agents.	Allowed for the flexible modification of network topology within an heuristic search MOEA framework.
Python based	Connection to GAMS using its built-in Python API and native integration with Pyomo for mathematical programming, along with other scientific programming packages (NumPy, SciPy)	Relies heavily on NumPy and Pandas.
Network Based	Supported implementation of a range of node types, including hydrologic water supply and human water use nodes.	Simulation is run repeatedly, where the model dynamically changes the network topology before running each simulation. The same network object is used throughout.
Institutions	Supported the implementation of government agencies as non-spatial model entities.	Allowed for the grouping of interdependent nodes to simplify engine algorithms.
Engines	Allowed for the detailed sequencing of processes among 1000s of agents and for the modelling complex hydrologic processes (e.g., groundwater response).	Deploys different algorithms and approaches by inclusion or exclusion of engines.

Pynsim relies on model developers embracing Python as the language linking their models to data and other models. Opportunely, Python is becoming more common in water resources and environmental modelling and so this approach may be attractive to model developers seeking to integrate multiple models.

Acknowledgments

This work was conducted as part of the Belmont Forum water security theme, funded in the UK by the Natural Environment Research Council (NERC) under grant NE/ L009285/2 to The University of Manchester and grant NE/ L009285/1 to University College London, and in the US by the National Science Foundation under grant GEO/OAD-1342869 to Stanford University.

Appendix A. Supplementary data

Supplementary data related to this article can be found at https://doi.org/10.1016/j.envsoft.2018.01.019.

References

Altermatt, F., 2013. Diversity in riverine metacommunities: a network perspective. Aquat. Ecol. 47 (3), 365–377.

Ames, 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.

Argent, R., Voinov, A., Maxwell, T., Cuddy, S., Rahman, J., Seaton, S., Vertessy, R., Braddock, R., 2006. Comparing modelling frameworks – a workshop approach. Environ. Modell. Softw 21 (7), 895–910. http://www.sciencedirect.com/science/article/pii/S1364815205000915.

Barreteau, O., Bousquet, F., 2000. Shadoc: a multi-agent model to tackle viability of irrigated systems. Ann. Oper. Res. 94 (1), 139–162.

Barthel, R., Janisch, S., Schwarz, N., Trifkovic, A., Nickel, D., Schulz, C., Mauser, W., 2008. An integrated modelling framework for simulating regional-scale actor responses to global change in the water domain. Environ. Modell. Softw 23 (9), 1095–1121. http://www.sciencedirect.com/science/article/pii/S1364815208000224.

Becu, N., Perez, P., Walker, A., Barreteau, O., Page, C., 2003. Agent based simulation of a small catchment water management in northern Thailand: description of the CATCHSCAPE model. Ecol. Model. 170 (2–3), 319–331. http://www.sciencedirect.com/science/article/pii/S0304380003002369.

Belete, G.F., Voinov, A., Morales, J., AUG 2017. Designing the distributed model integration framework - DMIF. Environ. Model. Software 94, 112–126.

Bernholdt, D.E., Elwasif, W.R., Kohl, J.A., Epperly, T.G., 2002. A component architecture for high-performance computing. In: Proceedings of the Workshop on Performance Optimization via High-level Languages and Libraries (POHLL-02). New York, NY, USA.

Bosse, T., Blom, H., Stroeve, S., Sharpanskykh, A., Nov 2013. An integrated multi-agent model for modelling hazards within air traffic management. In: Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2013 IEEE/WIC/ACM International Joint Conferences on, vol. 2, pp. 179–186.

Bousquet, F., Bakam, I., Proton, H., Le Page, C., 1998. Cormas: Common-pool Resources and Multi-agent Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 826–837. https://doi.org/10.1007/3-540-64574-8_469.

Bousquet, F., Barreteau, O., Le Page, C., Mullon, C., Weber, J., 1999. An environmental modelling approach: the use of multi-agent simulations. Advances in environmental and ecological modelling 113, 122.

Buahin, C.A., Horsburgh, J.S., 2015. Evaluating the simulation times and mass balance errors of component-based models: an application of OpenMI 2.0 to an urban stormwater system. Environ. Modell. Softw 72, 92–109. http://www.sciencedirect.com/science/article/pii/S1364815215300086.

Bulatewicz, T., Allen, A., Peterson, J., Staggenborg, S., Welch, S., Steward, D., 2013. The simple script wrapper for OpenMI: enabling interdisciplinary modeling studies. Environ. Modell. Softw. 39, 283–294 thematic Issue on the Future of Integrated Modeling Science and Technology. http://www.sciencedirect.com/science/article/pii/S1364815212002046.

Burroughs, W.J., 2007. Climate Change: a Multidisciplinary Approach. Cambridge University Press.

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.

Castilla-Rho, J., Mariethoz, G., Rojas, R., Andersen, M., Kelly, B., 2015. An agent-based platform for simulating complex human-aquifer interactions in managed groundwater systems. Environ. Modell. Softw 73, 305–323. http://www.sciencedirect.com/science/article/pii/S136481521530044X.

Castronova, A.M., Goodall, J.L., Elag, M.M., 2013a. Models as web services using the open geospatial Consortium (OGC) web processing service (WPS) standard. Environ. Modell. Softw 41 (0), 72–83. http://www.sciencedirect.com/science/article/pii/S1364815212002812.

Castronova, A.M., Goodall, J.L., Ercan, M.B., 2013b. Integrated modeling within a hydrologic information system: an OpenMI based approach. Environ. Modell. Softw 39, 263–273.

Chappin, E.J.L., de Vries, L.J., Richstein, J.C., Bhagwat, P., Iychettira, K., Khan, S., OCT 2017. Simulating climate and energy policy with agent-based modelling: the Energy Modelling Laboratory (EMLab). Environ. Model. Software 96, 421–431.

Collier, N., 2003. Repast: an Extensible Framework for Agent Simulation, vol. 36. The University of Chicagos Social Science Research, pp. 371–375.

Collier, N., Howe, T., North, M., 2003. Onward and upward: the transition to Repast 2.0. In: Proceedings of the First Annual North American Association for Computational Social and Organizational Science Conference, pp. 241–268.

Daloğlu, I., Nassauer, J.I., Riolo, R., Scavia, D., 2014. An integrated social and ecological modeling framework - impacts of agricultural conservation practices on water quality. Ecol. Soc. 19, 12.

David, O.,I.I.,J.A., Lloyd, W., Green, T., Rojas, K., Leavesley, G., Ahuja, L., 2013. A software engineering perspective on environmental modeling framework design: the object modeling system. Environ. Modell. Softw. 39, 201–213 thematic Issue on the Future of Integrated Modeling Science and Technology. http://www.sciencedirect.com/science/article/pii/S1364815212000886.

Davidsson, P., 2000. Multi-agent-based simulation. In: Second International Workshop, MABS 2000.

Dozier, A.Q., David, O., Arabi, M., Lloyd, W., Zhang, Y., JUN 2016. A minimally

invasive model data passing interface for integrating legacy environmental system models. Environ. Model. Software 80, 265–280.
Fienen, M.N., Plant, N.G., 2015. A cross-validation package driving Netica with Python. Environ. Modell. Softw. 63, 14–23. http://www.sciencedirect.com/science/article/pii/S1364815214002606.
Galán, J.M., López-Paredes, A., del Olmo, R., 2009. An agent-based model for domestic water management in valladolid metropolitan area. Water Resour. Res. 45 (5), W05401.
Ghazi, S., Khadir, T., Dugdale, J., 2014. Highlights of practical applications of heterogeneous multi-agent systems. The PAAMS collection: PAAMS 2014 international workshops, Salamanca, Spain, June 4-6, 2014. In: Proceedings. Springer International Publishing, Cham, Ch. Multi-agent Based Simulation of Environmental Pollution Issues: a Review, pp. 13–21. https://doi.org/10.1007/978-3-319-07767-3_2.
Goodall, J., Peckman, S., 2016. Interoperability between the Basic Modeling Interface (BMI) and the Open Modeling Interface (OpenMI): a step toward building the earth system bridge for modeling framework interoperability. In: 8th International Congress on Environmental Modelling and Software.
Goodall, J.L., Robinson, B.F., Castronova, A.M., 2011. Modeling water resource systems using a service-oriented computing paradigm. Environ. Modell. Softw 26 (5), 573–582. http://www.sciencedirect.com/science/article/pii/S1364815210003178.
Goodall, J.L., Saint, K.D., Ercan, M.B., Briley, L.J., Murphy, S., You, H., DeLuca, C., Rood, R.B., 2013. Coupling climate and hydrological models: interoperability through web services. Environ. Modell. Softw 46, 250–259. http://www.sciencedirect.com/science/article/pii/S136481521300090X.
Gregersen, J., Gijsbers, P., Westen, S., 2007. OpenMI: open modelling interface. J. Hydroinf. 9 (3), 175–191.
Hadka, D., Reed, P., 2013. Borg: an auto-adaptive many-objective evolutionary computing framework. Evol. Comput. 21 (2), 231–259.
Hagbert, E., Schult, D., Swart, P., 2004. Networkx: Python Software for Complex Networks. https://github.com/networkx/.
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., 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., Filatova, T. (Eds.), 2010 International Congress on Environmental Modelling and Software Modelling for Environment's Sake, Fifth Biennial Meeting, Ottawa, Canada. International Environmental Modelling and Software Society (IEMSs).
Harou, J.J., Pulido-Velazquez, M., Rosenberg, D.E., Medelln-Azuara, J., Lund, J.R., Howitt, R.E., 2009. Hydro-economic models: concepts, design, applications, and future prospects. J. Hydrol. 375 (34), 627–643. http://www.sciencedirect.com/science/article/pii/S0022169409003588.
Hart, W.E., Watson, J.-P., Woodruff, D.L., 2011. Pyomo: modeling and solving mathematical programs in Python. Mathematical Programming Computation 3 (3), 219–260. https://doi.org/10.1007/s12532-011-0026-8.
Hiebeler, D., et al., 1994. The swarm simulation system and individual-based modeling. In: Proceedings of Decision Support 2001: Advanced Technology for Natural Resource Management. Santa Fe Institute.
Hill, C., DeLuca, C., Suarez, M., Da Silva, A., et al., 2004. The architecture of the earth system modeling framework. Comput. Sci. Eng. 6 (1), 18–28.
Jiang, P., Elag, M., Kumar, P., Peckham, S.D., Marini, L., Rui, L., JUN 2017. A service-oriented architecture for coupling web service models using the Basic Model Interface (BMI). Environ. Model. Software 92, 107–118.
Jin, X., Robinson, K., Lee, A., Polhill, J.G., Pritchard, C., Parker, D.C., OCT 2017. A prototype cloud-based reproducible data analysis and visualization platform for outputs of agent-based models. Environ. Model. Software 96, 172–180.
Kelly, R.A., Jakeman, A.J., Barreteau, O., Borsuk, M.E., ElSawah, S., Hamilton, S.H., Henriksen, H.J., Kuikka, S., Maier, H.R., Rizzoli, A.E., van Delden, H., Voinov, A.A., 2013. Selecting among five common modelling approaches for integrated environmental assessment and management. Environ. Modell. Softw 47, 159–181. http://www.sciencedirect.com/science/article/pii/S1364815213001151.
Klassert, C., Sigel, K., Gawel, E., Klauer, B., 2015. Modeling residential water consumption in amman: the role of intermittency, storage, and pricing for piped and tanker water. Water 7 (7), 3643–3670.
Knapen, R., Janssen, S., Roosenschoon, O., Verweij, P., de Winter, W., Uiterwijk, M., Wien, J.-E., 2013. Evaluating OpenMI as a model integration platform across disciplines. Environ. Modell. Softw 39, 274–282 thematic Issue on the Future of Integrated Modeling Science and Technology. http://www.sciencedirect.com/science/article/pii/S1364815212001946.
Knox, S., Meier, P., Harou, J.J., 2014. Web service and plug-in architecture for flexibility and openness of environmental data sharing platforms. In: Ames, D.P., Quinn, N.W.T., Rizzoli, A.E. (Eds.), 7th Intl. Congress on Env. Modelling and Software. International Environmental Modelling and Software Society (IEMSs), San Diego, California, USA, pp. 83–90. In: http://www.iemss.org/society/index.php/iemss-2014-proceedings.
Laniak, G.F., 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. Environ. Modell. Softw. 39, 3–23 thematic Issue on the Future of Integrated Modeling Science and Technology. http://www.sciencedirect.com/science/article/pii/S1364815212002381.
Leavesley, G.H., Markstrom, S.L., Viger, R.J., Hay, L.E., 2007. Hydrological modelling in arid and semi-arid areas. In: Ch. The Modular Modeling System (MMS): a Toolbox for Water-and Environmental-resources Management. Cambridge University Press, p. 87.
Lee, E., Mitchell, J., Wallace, W., Nov 2007. Restoration of services in interdependent infrastructure systems: a network flows approach. IEEE Trans. Syst. Man Cybern. C Appl. Rev. 37 (6), 1303–1317.
Letcher, R.A., Croke, B.F.W., Jakeman, A.J., May 2007. Integrated assessment modelling for water resource allocation and management: a generalised conceptual framework. Environ. Model. Softw. 22 (5), 733–742. https://doi.org/10.1016/j.envsoft.2005.12.014.
Lloyd, W., David, O.,I.I.,J.A., Rojas, K., Carlson, J., Leavesley, G., Krause, P., Green, T., Ahuja, L., 2011. Environmental modeling framework invasiveness: analysis and implications. Environ. Modell. Softw. 26 (10), 1240–1250. http://www.sciencedirect.com/science/article/pii/S1364815211000867.
Loucks, D.P., Van Beek, E., Stedinger, J.R., Dijkman, J.P., Villars, M.T., 2005. Water Resources Systems Planning and Management: an Introduction to Methods, Models and Applications. UNESCO, Paris.
Luke, S., Balan, G.C., Panait, L., Cioffi-Revilla, C., Paus, S., 2003. Mason: a java multi-agent simulation library. In: Proceedings of Agent 2003 Conference on Challenges in Social Simulation, vol. 9.
Lünsdorf, O., Scherfke, S., 2013. Event Discrete Simulation for Python. https://simpy.readthedocs.io/en/latest/index.html.
Maddock, T., 1972. Algebraic technological function from a simulation model. Water Resour. Res. 8 (1), 129–134.
Marta-Almeida, M., Ruiz-Villarreal, M., Otero, P., Cobas, M., Peliz, A., Nolasco, R., Cirano, M., Pereira, J., 2011. OOF$_e$: a Python engine for automating regional and coastal ocean forecasts. Environ. Modell. Softw. 26 (5), 680–682. http://www.sciencedirect.com/science/article/pii/S1364815210003221.
Meier, P., Knox, S., Harou, J.J., 2014. Linking water resource network models to an open data management platform. In: Ames, D.P., Quinn, N.W.T., Rizzoli, A.E. (Eds.), 7th Intl. Congress on Env. Modelling and Software. International Environmental Modelling and Software Society (IEMSs), San Diego, California, USA, pp. 463–469. In: http://www.iemss.org/society/index.php/iemss-2014-proceedings.
Morsy, M.M., Goodall, J.L., Castronova, A.M., Dash, P., Merwade, V., Sadler, J.M., Rajib, M.A., Horsburgh, J.S., Tarboton, D.G., JUL 2017. Design of a metadata framework for environmental models with an example hydrologic application in HydroShare. Environ. Model. Software 93, 13–28.
Müller, J., 2010. A framework for integrated modeling using a knowledge-driven approach. In: Proceedings of the International Congress on Environmental Modelling and Software, Ottawa, Canada.
Padowski, J.C., Gorelick, S.M., Thompson, B.H., Rozelle, S., Fendorf, S., 2015. Assessment of human-natural system characteristics influencing global freshwater supply vulnerability. Environ. Res. Lett. 10 (10), 104014. http://stacks.iop.org/1748-9326/10/i=10/a=104014.
Peckham, S.D., 2014. Emeli 1.0: an experimental smart modeling framework for automatic coupling of self-describing models. In: International Conference on Hydroinformatics.
Peckham, S.D., Hutton, E.W., Norris, B., 2013. A component-based approach to integrated modeling in the geosciences: the design of CSDMS. Comput. Geosci. 53, 3–12 modeling for Environmental Change. http://www.sciencedirect.com/science/article/pii/S0098300412001252.
Pipattanasomporn, M., Feroze, H., Rahman, S., 2009. Multi-agent systems in a distributed smart grid: design and implementation. In: Power Systems Conference and Exposition, 2009. PSCE'09. IEEE/PES. IEEE, pp. 1–8.
Rajsekhar, D., Gorelick, S.M., 2017. Increasing drought in Jordan: climate change and cascading syrian land-use impacts on reducing transboundary flow. Science Advances 3 (8). http://advances.sciencemag.org/content/3/8/e1700581.
Robinson, S., 2014. Simulation: the Practice of Model Development and Use. Palgrave Macmillan.
Sánchez, P.J., 2007. Fundamentals of simulation modeling. In: Proceedings of the 39th Conference on Winter Simulation: 40 Years! the Best Is yet to Come. IEEE Press, pp. 54–62.
Saqalli, M., Gérard, B., Bielders, C., Defourny, P., 2010. Testing the impact of social forces on the evolution of sahelian farming systems: a combined agent-based modeling and anthropological approach. Ecol. Model. 221 (22), 2714–2727.
Schreinemachers, P., Berger, T., 2011. An agent-based simulation model of humanenvironment interactions in agricultural systems. Environ. Modell. Softw 26 (7), 845–859. http://www.sciencedirect.com/science/article/pii/S1364815211000314.
Tarboton, D., Idaszak, R., Horsburgh, J., Ames, D., Goodall, J., Band, L., Merwade, V., Couch, A., Arrigo, J., Hooper, R., et al., 2013. Hydroshare: an online, collaborative environment for the sharing of hydrologic data and models. In: AGU Fall Meeting Abstracts, vol. 1, p. 1510.
Tesfatsion, L., Rehmann, C.R., Cardoso, D.S., Jie, Y., Gutowski, W.J., MAR 2017. An agent-based platform for the study of watersheds as coupled natural and human systems. Environ. Model. Software 89, 40–60.
Thiele, J., Kurth, W., Grimm, V., 2011. Agent-and individual-based modelling with netlogo: introduction and new netlogo extensions. Die Grüne Reihe 22, 68–101.

Thorp, K., Bronson, K., 2013. A model-independent open-source geospatial tool for managing point-based environmental model simulations at multiple spatial locations. Eviron. Modell. Softw. 50, 25–36. http://www.sciencedirect.com/science/article/pii/S136481521300193X.

Tisue, S., Wilensky, U., 2004. Netlogo: a simple environment for modeling complexity. In: International Conference on Complex Systems. Boston, MA, pp. 16–21.

Whelan, G., Kim, K., Pelton, M.A., Castleton, K.J., Laniak, G.F., Wolfe, K., Parmar, R., Babendreier, J., Galvin, M., 2014. Design of a component-based integrated environmental modeling framework. Eviron. Modell. Softw 55, 1–24. http://www.sciencedirect.com/science/article/pii/S1364815214000267.

White, J.T., Fienen, M.N., Doherty, J.E., NOV 2016. A python framework for environmental model uncertainty analysis. Environ. Model. Software 85, 217–228.

Wooldridge, M., 2009. An Introduction to MultiAgent Systems, second ed. Wiley Publishing.