A WEB-SERVICE FRAMEWORK FOR THE WATER EVALUATION AND PLANNING SYSTEM

Mostafa D. Fard

Arizona Center for Integrative Modeling & Simulation
School of Computing, Informatics, and Decision Systems Engineering
Tempe, AZ, USA
smd.fard@asu.edu

Hessam S. Sarjoughian

Arizona Center for Integrative Modeling & Simulation
School of Computing, Informatics, and Decision Systems Engineering
Tempe, AZ, USA
sarjoughian@asu.edu

ABSTRACT

For systems-of-systems, the use of different modeling methods is important not only because each system can be described more correctly, but also benefit from tools that are in use by different communities. Using an opaque or closed source-code tool with others, however, is challenging. Consequently, to facilitate the development of simulations for systems-of-systems it is useful to cast closed-source models in a flexible component-based framework. Using this concept, a RESTful web service framework is developed for the Water Evaluation And Planning (WEAP) software system. The WEAP RESTful framework has a suite of model components for all the model entities defined in the WEAP system. An example water system model shows the computational cost for the web-service framework is negligible. Casting the model entities to components can play a key role in using the WEAP system with other modeling frameworks useful for simulating the complexities of the Food-Energy-Water systems.

Keywords: Component-based modeling, MDA, RESTful services, systems-of-systems, WEAP.

1 INTRODUCTION

Practitioners and researchers contemplate questions of interest on a part of a system. In this approach, a significant part of a system-of-systems is modeled in detail while all other parts that affect or affected by it are simplified (or sometime entirely excluded). This is attractive as the overall system complexity and scale is greatly constrained through replacing the dynamics of some systems as inputs and outputs. For example, in modeling a water system that uses solar energy, the amount of available photovoltaic energy is modeled as some piecewise input regime. A key consequence is that such an input is non-functional. In contrast, a reactive constituent model produces outputs in part based on consuming inputs dynamically. A reactive photovoltaic model that supplies energy to the water system subject to water demand fluctuation supports understanding of a water system that cannot be achieved through input data. The need for component-based modeling and simulation becomes evident for understanding of the interactions (nexus) among different parts of complex systems such as the Food-Energy-Water (FEW) systems (Hoff 2011). Simulation developed using different component-based modeling approaches is a key in detailing different kinds of behaviors belonging to different parts of a system-of-systems.

Some of the existing and popular modeling and simulation frameworks appear to be component-based since non-componentized models are displayed as components in tools. A popular tool for study of water supply and demand is the Water Evaluation and Planning (WEAP) system (Sieber and Purkey 2005). In WEAP, a

model of a water system is defined as a project that has node and link elements with associated experiment scenarios. The inputs and outputs for the node and link models are global variables. The WEAP system supports generic scripting to manipulate models. In particular, inputs external to a WEAP model (nodes and links) can be read via scripts and similarly outputs external to the models can be written via scripts. Models that have shared variables and supporting scripts for interaction with other models lack the flexibility afforded by component-based modeling and simulation frameworks. A particular consequence of modeling frameworks such as WEAP is the difficulty of using it systematically with other frameworks. A desirable modeling framework should simplify and promote combing models that are developed in different frameworks or tools. In such a modeling framework each model entity is a standalone component having its inputs and outputs and functions encapsulated and thus not shared with any other model. In a component-based modeling framework, models are clients and servers that can independently acted on. In a direct way, componentization of tools such as WEAP can further their use in modeling and simulation system-of-systems including the class of *Food-Energy-Water* (FEW) systems (Hoff 2011).

In this paper, we detail a component-based framework we have developed for the WEAP system using web services. This framework provides *component proxies* for the model entities defined in the WEAP system. The proxy component models have counterparts in the WEAP system. A suite of Ecore models are developed using the Model-Driven Architecture design approach. We use the Ecore modeling combined with the RESTful framework to design and implement a web-service framework for the WEAP system. The proposed WEAP RESTful framework supports automatically extracts and replication of any water system model (e.g., demand and supply nodes and transmission link) developed in the WEAP system. This web-service framework enables development and simulation of multiple water systems as supported by the WEAP system.

2 BACKGROUND

In this section the Water Evaluation and Planning (WEAP) system and the RESTful framework are briefly described. The former provides an overview of the model conceptualization and development. The WEAP overview is from the perspective of being used with other modeling and simulation frameworks. The latter highlights the choice of technology as a suitable platform for design and implementation of the web-service framework for the WEAP system.

2.1 Water Evaluation and Planning Software System

A modeling, simulation, and evaluation framework for studying water supply and demand scenarios is the WEAP system (Sieber and Purkey 2005). Analysts can develop abstractions for water resources, transportation, and usage mapped to some given geographical space. This system is targeted for water policy decision making (availability and usage) for a water system network.

As a modeling, simulation and analysis tool, WEAP consists of the Schematic, Data, Results, and Scenario Explorer sections (see Figure 4(a)) (WEAP21). The Schematic has a number of *entities*. It allows selecting and assigning the entities of a water system in a geospatial map. The entities are river, diversion, reservoir, groundwater, desalination plant, wastewater treatment plant, demand site, transmission link, catchment, return flow, and flow requirement. The *Data* serves to parametrize the inputs and equations in terms of supply resources, demand sites, hydrology, and water quality with assumptions. The Data portion allows defining the details of the entities in the Schematic view of the system model. The entities in the Schematic view are categorized in the Data view. The *Results* section is for choosing outputs to be extracted and viewed. The *Scenario Explorer* section can be used to define experiments that can be observed during execution or stored for post processing.

The WEAP system is a closed-source software tool. It has pre-defined entities for supply resources (rivers, groundwater, reservoirs, desalination plants), facilities (withdrawal, transmission, and wastewater treatment), demand sites, pollution, and ecosystem. These entities are quasi-component models that have

pre-defined inputs and outputs. As components, their inputs and outputs are known, but their mass-balance equations are hidden. The variables for these models appear to be shared among them. The WEAP development team can make changes to existing entities or add new ones. The logical and schematic models with their data are tightly interwoven to experimentations, and results. Only the input and output variables for the models are accessible outside of the WEAP system. Water system models are not hierarchical.

A model for a water systems in the WEAP system is represented as a graph of entities (e.g., a Demand Site) where entities can receive inputs and produce outputs. An input data set can be set internally or read externally. Similarly, an output data set can be accessed internally (graphs and tables) or externally written. The model can be simulated for a finite number of rounds (e.g., a round can be a day) for some given input and output data sets. Once a simulation starts for some given input, it must process all its given input. It should be noted that the rounds within one simulation are invisible to any other simulation even though the input and output data sets are available. Interrupting a simulation midstream and resuming it is not allowed. In this respect, WEAP models are not reactive since input from any external simulation model is not possible. The input and output data of a WEAP model can be used to combined with some other models.

2.2 RESTful Web Services

Web Service is a language independent communication method for software systems. There are two main protocols for web services, SOAP and REST. The SOAP (Simple Object Access Protocol) service is a standard communication protocol specification for XML-based message exchange. It uses different transport protocols, such as HTTP (Hypertext Transfer Protocol) and SMTP (Simple Mail Transfer Protocol). This standard was developed as an alternative to CORBA (Common Object Request Broker Architecture) or DCOM (Distributed Component Object Model). It can sometimes be slower than middleware technologies due to its verbose XML format. With SOAP, the client does not choose to interact directly with a resource, but instead calls a service to access the various external or remote objects and resources. Higher security and reliability, fewer number of errors, asynchronous requests, distributed computing, and support from other standards (e.g., WSDL) can be considered as the main reasons for using the SOAP standard.

The REST (REpresentational State Transfer) is an architectural style for creating web services. The client-server communication, stateless operation, uniform interface, and resource caching are the underlying principles for the REST architecture (Fielding and Taylor 2000). Unlike SOAP, REST is not required to use XML, but instead can return XML, JSON, YAML or any other format depending on the client needs. With REST, a client locates a resource on the server to either update that resource, delete it or retrieve some information about it. RESTful is the implementation of the REST architecture that uses HTTP as its underlying communication method (GET, PUT, POST, and DELETE). Greater scalability, compatibility, performance, simplicity, point-to-point communication, and limited bandwidth can be considered the defining aspects of the RESTful services.

3 RELATED WORK

A variety of modeling frameworks have been developed in recent years based on the concept of components (Zeigler and Sarjoughian 2017). A model of a system can be created as a collection of interacting components implemented in object-based or object-oriented programming languages. In contrast, modeling approaches based on data structures and function are developed based on procedural programming languages. The WEAP system belongs to the later class modeling approaches. Considering simulation studies of Food-Energy-Water systems, other tools and framework such as PRMS (Markstrom et al. 2015) and GSFLOW (Markstrom et al. 2008) and WEF (Daher and Mohtar 2015) are available. PRMS (Precipitation Runoff Modeling System) is a deterministic, physical process based modeling system developed to evaluate the response of various combinations of climate and land use on streamflow and general watershed hydrology. GSFLOW (Ground-water and Surface-water FLOW) is developed to simulate coupled ground-water and surface-water resources. The WEF-Nexus Tool 2.0 defines a holistic

model that has food, energy and water parts. Considering modeling and simulation of Food-Energy-Water systems, it is impractical to use WEF and others like it since they do not support component-based modeling and thus difficult to be integrated with other tools. Understanding the nexus of FEW systems (White et al. 2017) can benefit from modeling interactions among subsystems using the heterogeneous model composability approach (Sarjoughian 2006).

The Extensible Modeling and Simulation Framework (XMSF) is defined as a composable set of standards, profiles and recommended practices for web-based modeling & simulation (Brutzman et al. 2002). XMSF (Pullen et al. 2005) supports the migration of legacy components into web enabled components reusable for distributed heterogeneous simulation applications. XMSF is based on SOAP and XML (Brutzman et al. 2002), whereas our WEAP web-service system benefits from the RESTful web services and the Ecore modeling frameworks.

4 A WEB SERVICE FRAMEWORK FOR THE WEAP SOFTWARE SYSTEM

Selecting a protocol for a framework depends on functional and nonfunctional requirements. For example, if the framework is about integrating two simple information systems, the right choice will be RESTful. However, if systems are complex and they should have additional security levels, the better choice can be SOAP (Tihomirovs and Grabis 2016). Many developers found SOAP cumbersome and hard to use. Especially working with SOAP in JavaScript programming language requires extensive code development for very simple tasks because the required XML structure must be created repeatedly. Thus, the lightweight, fast and scalable RESTful framework is used in this research.

4.1 WEAP Model Components

As discussed before, we are construing a frame around the WEAP system (this is called componentized WEAP). It represents the entities that are included in the WEAP system with their data. Figure 1 shows the model components and their data in the WEAP RESTful framework. The main class *WEAP* represents the WEAP system has an array of projects associated with geographic areas in. Each *Project* has its own configuration properties (name, startYear, endYear, timeStepPerYear, etc.) and three references to the *Scenario*, *Link* and *Node* classes. The *Node* and *Link* are two main abstract classes which shown in gray color in Figure 1. These are general classes and will be specialized in the following subsections. The *WEAP*, *Project,* and *Scenario* are concrete classes and shown in yellow color in Figure 1. These correspond to the actual elements in the WEAP system. These are used for instantiating a specific domain model in the WEAP RESTful framework. In the rest of the paper, abstract and concrete classes are defined to serve the same purposes. In a general view, the WEAP system is a black box simulation model which has an structure defined by a modeler. The behaviors for the specialized node and link entities (e.g., Reservoir and Transmission) are predefined in the WEAP system. Every scenario provides data to be simulated. The simulated results can be observed as graphs and tabulated data. Every project has at least one scenario; the *Current Accounts* which provides a snapshot of actual water demand, pollution loads, resources and supplies for the system (Sieber and Purkey 2005). It is actually used to define the initial data for the inputs at the start of a simulation. Even though a scenario has a hierarchical structure in the WEAP system, this is not important in the RESTful WEAP framework since each data for a scenario is independent from the data from other scenarios. The *name* is a key property in *Project*, *Scenario* and *Node* classes (see Figure 1). At least two nodes and one link are required to have a meaningful model (the reason for the cardinality constraint for the *nodes* and *links* composite relations in Figure 1).

The WEAP RESTful framework has a complete set of model components that cover all the entities in the Schematic view and the entities in the Data section of the WEAP system. The model components are categorized to the *Node* and *Link* types according to their respective properties (see Figure 1). Each *Link* has one source node and one target node.

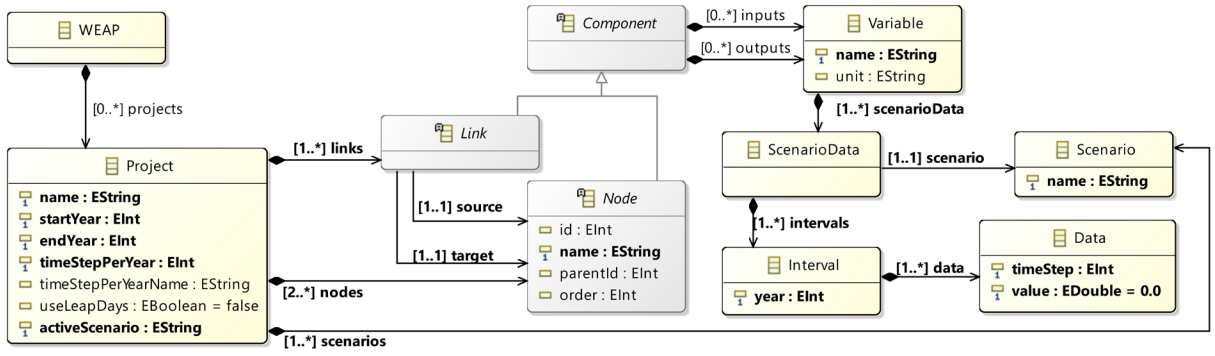


Figure 1: Ecore specification for model components and their data in the WEAP RESTful framework.

All model components (nodes and links) have some input and output variables which are presented by composition relations from *Component* class to *Variable* class in Figure 1. Each model component has its own predefined variables. The WEAP system allows users to define their own variables and equations to refine and/or adapt the pre-defined constraints and conditions defined for a given model (Sieber and Purkey 2005). For each *Variable* (which has a unique *name* and a *unit*), one or more intervals are defined per scenario and each *Interval* can have many *Data* values (indicates a value for a specific time step of a year). References defines a one-to-many constraint given the default Current Accounts scenario defined in the WEAP system. From a higher abstract perspective, in Figure 1, the *Node* and *Link* classes with their input and output variables, define the structure of a model. The *Variable*, *ScenarioData*, *Scenario*, *Interval* and *Data* classes with their relations, define the overall input data and output result for a model in the WEAP RESTful framework that mirrors those defined in the WEAP system.

As shown in Figure 2, there are different types of node. At a high level of abstraction, a node is either simple or complex. A set of simple nodes are inherited from the abstract class *Node*. A set of complex nodes are inherited from the abstract class *Flow*. The flow nodes are an ordered collection of simple nodes (called reach points) that can be connect to other nodes using links. As the result, a specific variable of a flow can have different values in different reach points. Composite relations from *Flow* abstract class to other classes in Figure 2 indicate the nodes which can be placed on the *Flow* components (on the *River* and *Diversion*), except *Reservoir* that can be a separate single node, as well. A *Diversion* can start from the *DiversionOutflowPoint* on another flow (a distributary from another river or diversion). Also a *River* or a *Diversion* can end in a *TributaryInflowPoint* on another flow (i.e., a tributary turns into a river or a diversion).

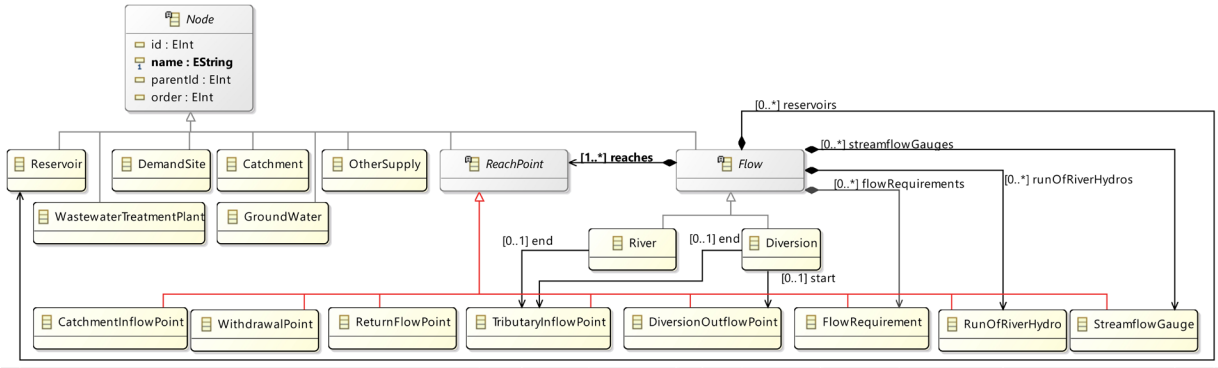
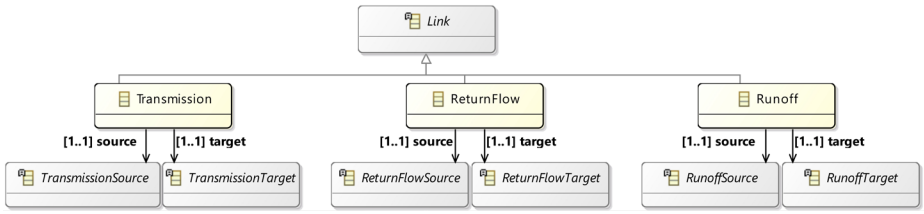
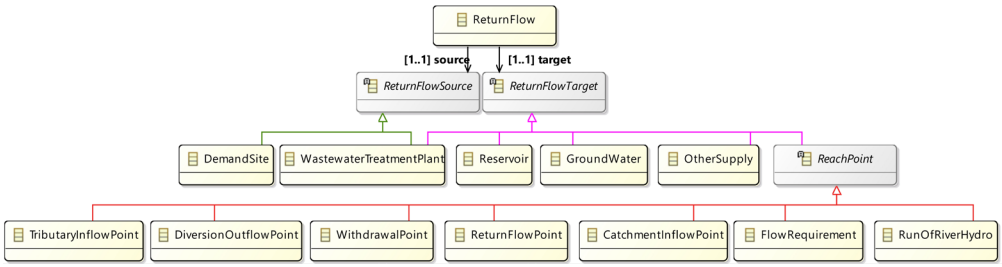


Figure 2: Ecore specification for the node components in the WEAP RESTful Framework.

Figure 3(a) shows three types of links (*Transmission*, *ReturnFlow* and *Runoff*). Each links has a source node and a target node. There are restrictions for the source and target nodes for different type of links (Sieber and Purkey 2005). For example, allowable source and target nodes for *ReturnFlow* are shown in Figure 3(b). As can be seen, a *Wastewater Treatment Plant* can be the source or target of a *Return Flow*, but a *Reservoir* just can be the target node (*Return Flow* cannot start from a *Reservoir*). Consider that link is inherited from *Component* class (see Figure 1), so *Transmission*, *Return Flow* and *Runoff* have their own input and output variables and data. The specifications for the other link types (*Transmission* and *Runoff*) are not shown due to space limitation.



(a)



(b)

Figure 3: Ecore specifications for link components in the WEAP RESTful framework. (a) Type of Links (b) allowable source and target nodes for the ReturnFlow link.

4.2 WEAP RESTful Framework

Two basic concepts in the RESTful API are *Resource* and *URL* (Uniform Resource Locator) (Fielding and Taylor 2000). The Resource is an object or representation of a thing that has some data associated with it. The URL is a path through which a resource can be located and some actions can be performed on it. The URL in a RESTful API should only contain resources (nouns), not actions or verbs. In the WEAP RESTful framework, the model components are the actual resources, so we need to make a well-defined structure for the URL to operate on them. The JSON format is used to communicate data structure in the framework. The required WEAP RESTful APIs to communicate with the WEAP application software are divided to four categories; *Project*, *Node*, *Link* and *Flow*. To define URLs, constants are written in *UpperCase* style (starting all concatenated words with upper case); parameters start with colons and written in *lowerCase* style (starting the first word with lower case and the rest concatenated words with upper case); query parameters to apply to some filters on returned data written after the question mark by *Key=Value* and *lowerCase* style for the *Key*.

Table 1 lists the URL patterns for the four APIs defined above. Each open and close pair bracket means empty string or the pattern inside the bracket can be placed in the URL. Also the pipe symbol means OR for the values inside the brackets. There is a mapping between the classes in Figure 1 and Figure 2 and the URL patterns in Table 1. All URLs start with constant `/Water` which refers to the *WEAP* class in Figure 1. For example using "`/Water`" URL returns the information of all projects (an array of class *Project*) using the composite relation from the *WEAP* class to the *Project* class in Figure 1. The second part for the URLs must be the name of a project if it needs to be selected. When a specific project is selected, there is

access to the configuration information of the selected project in the WEAP system, all nodes, links and scenarios (references from the *Project* class to the *Node*, *Link* and *Scenario* classes in Figure 1). For example, the URL "`/Water/demo/Nodes`" will return the information about all nodes in the *demo* project.

Table 1: URL Pattern for different type of APIs.

Category	URL Patterns	Component Types
Project	`/Water[/:projectName[/Nodes | /Links | /Scenarios]]`	-
Node	`/Water/:projectName/ComponentType[/:componentName[/VariableType[/:variableName[?scenarioName=S&startYear=N&endYear=N&startTimeStep=N&endTimeStep=N]]]]`	DemandSite, Reservoir, GroundWater, …
Link	`/Water/:projectName/ComponentType[/:sourceName/:targetName[/VariableType[/:variableName[?scenarioName=S&startYear=N&endYear=N&startTimeStep=N&endTimeStep=N]]]]`	Transmission, Runoff, ReturnFlow
Flow	`/Water/:projectName/ComponentType[/:componentName[/PropertyName[/:subNodeName]][/VariableType[/:variableName[?scenarioName=S&startYear=N&endYear=N&startTimeStep=N&endTimeStep=N]]]]`	River, Diversion

For the Node, Link or Flow categories in Table 1, the constant ComponentType in the URL patterns must be replaced by one of the items in its corresponding Component Types column. The key of the component (name of the entity for Node or Flow, and the name of the source and target nodes for Link) must be used to select that. For example in URL "`/Water/demo/DemandSite/phoenix`" we have access to the data for the *phoenix* demand site in the *demo* project. VariableType in the URL patterns must be replaced by the *Inputs* or *Outputs* term (see the *inputs* and *outputs* parameters in *Node* and *Link* classes in Figure 1). A variable can be specified by its name, and query parameters can be used to filter the returned data. As discussed before, the Flow component is an ordered collection of simple sub-nodes. Thus, in the Flow URLs, the *PropertyName* can be replaced with the *RunOfRiverHydros*, *StreamflowGauges*, *FlowRequirements*, *Reaches* or *Reservoirs* (the reference properties in *River* or *Diversion* components in Figure 2) to access to a specific collection of sub-nodes, and use *:subNodeName* in the URL to select one specific node.

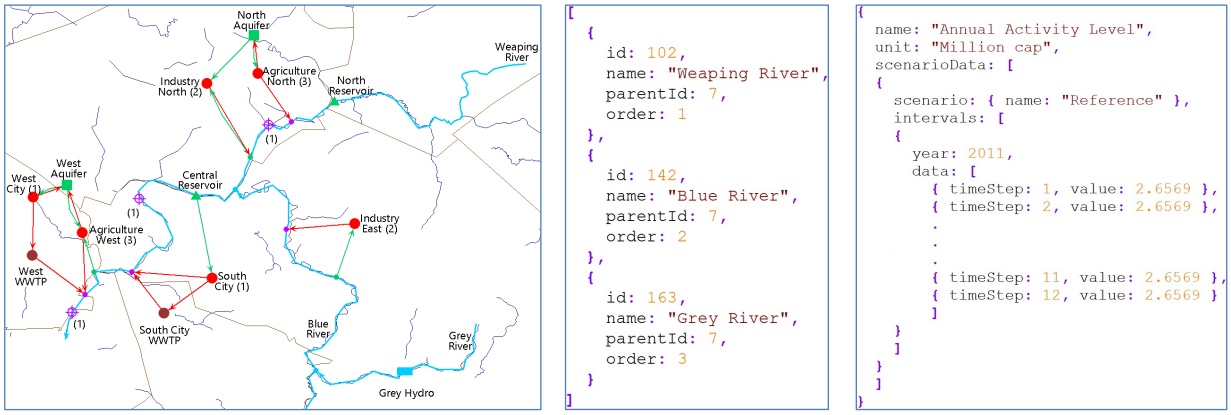
It is important to note that the functionality of the WEAP RESTful framework is restricted to the available WEAP-APIs (Automating WEAP). It means, there is not any WEAP-API to create a new component (Demand Site, River and etc.) from the outside of the WEAP system, but the existing information of a model can be fetched. So, the structure of a model (projects, nodes and links in Figure 1) must first be created in the WEAP system. Then, input or output for the existing components and their data (Variable in Figure 1) can be manipulated via the WEAP RESTful framework.

The "*Weaping River Basin*" project is one of the default projects in the WEAP system. It runs the simulation from 2010 to 2020 with 12 time steps per year. Figure 4(a) shows the Schematic view of this project in the WEAP system. Figure 4(b) is the result of calling an API to get the rivers in this project. Figure 4(c) shows the result of calling an API to get the data for the "*Activity Level*" variable of the "*West City*" demand site for the "*Reference*" scenario in *2011*.

4.3 Design and Implementation

Figure 5 presents the architecture of the WEAP RESTful framework. The WEAP system is located at the lowest level (*Proprietary Software*) of the framework. The *Componentized WEAP* layer allows access to the WEAP system by the *Service* layer and thus the *Web API* layer. The *Data Access* and *Components* packages provide simplified access to the projects in the WEAP system. These packages have classes that correspond to the model entities in the WEAP system. The *Service* centralizes external access to data and functions, in addition to hiding internal implementation. This layer has equivalent classes for the WEAP system entities. Each service can have access to many repositories in the *Data Access* layer. The *Web API*

is the highest layer supporting communication with client applications. It contains the web server and many controllers for handling the API requests. It also routes each request to the correct service and return well-formed data in JSON format (using the Domain Models defined in Figure 1) as the response.



(a) (b) (c)

Figure 4: (a) Schematic View "*Weaping River Basin*" project in the WEAP system . (b) result of calling the URL="`/Water/Weaping%20River%20Basin/River`". (c) result of calling the URL="`/Water/Weaping%20River%20Basin/DemandSite/West%20City/Annual%20Activity%20Level?scenarioName=Reference&startYear=2011&endYear=2011`".

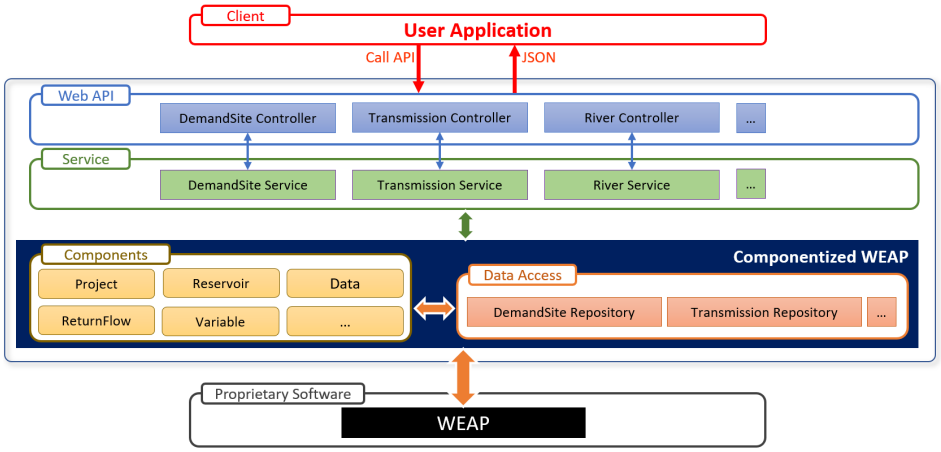


Figure 5: The WEAP RESTful Framework Architecture.

Figure 6 is a sequence diagram showing response to a fetching data request (API calling) by a client application. The calling API will parse and route to the correct controller and operations in a Controller object. From this step, we will work on a specific component (e.g., River, DemandSite, and ReturnFlow). Each controller object invokes a method from a Service object. In this framework, the *Factory* design pattern is used to create components; step 1.3.1 (corresponds to the WEAP's components) and their sub-domain models in step 1.3.1.1.1.3 in Figure 6. Also, each service class invoke method/methods from a Data Access object. A request to create component will send to a Factory object via a Service object and then a request will be sent to fetch data from a target repository. An instance of the WEAP system will instantiate at this step via Winax (JavaScript package that will explain later). In steps 1.3.1.1.1.2 and 1.3.1.1.1.2.1, the actual data are extracted from the WEAP system by calling the WEAP-APIs (Automating WEAP), then in the loop section, the data will searched for the relevant domain models. The return data is added to the

created component in step 1.3.1.1. Then the component with its data is returned to the Controller object. Finally, the controller object returns the data to the client application.

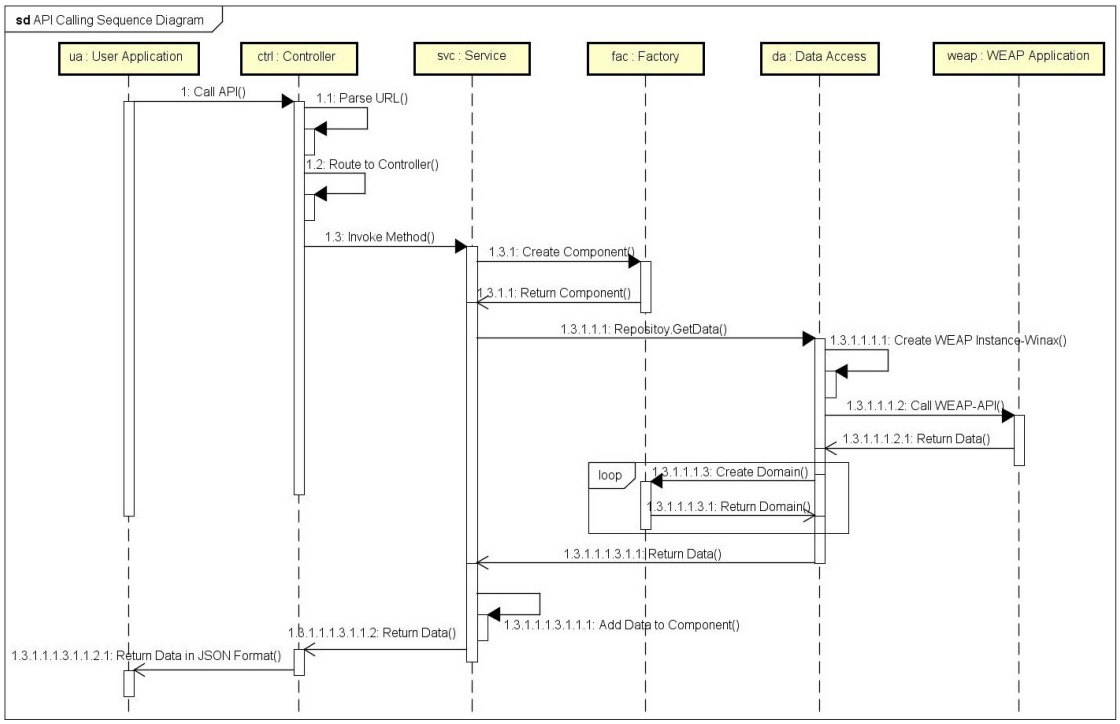


Figure 6: API calling sequence diagram.

Figure 7 shows the class diagram for the *Data Access* layer. The *INodeEntity*, *ILinkEntity* and *IFlowEntity* interfaces (corresponding to the three categories of model components) define the signatures of the methods which must be implemented. Each interface has its own abstract class for implementing the base methods. The three *AbstractNodeEntity*, *AbstractLinkEntity* and *AbstractFlowEntity* abstract classes inherit from the *AbstractEntity* class which defines *Component Type*. The node, link and flow classes are inherited from their respective abstract classes. Interfaces are defined for other classes (e.g., *DemandSite*, *Catchment*, and *River*) that are not shown in Figure 7.

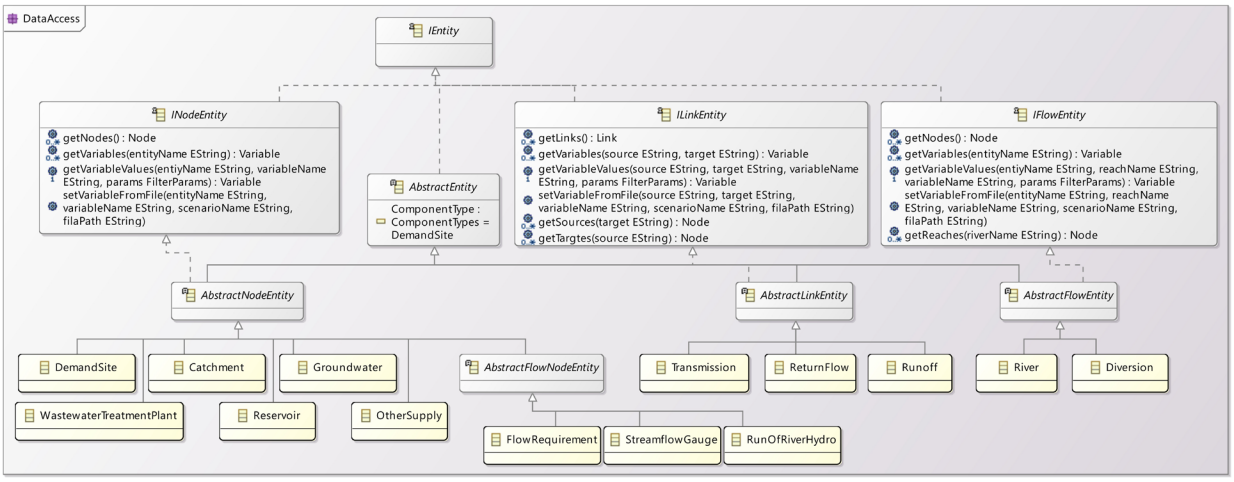


Figure 7: Data Access Layer class diagram.

5 PERFORMANCE EFFICIENCY OF THE WEAP RESTFUL SIMULATION

As described before, the WEAP system is a close-source software. However, the Automating WEAP API supports the script languages VB-Script, JScript, and Python to manipulate and execute models developed in the WEAP system. To compare the WEAP system computational time, the *Weaping River Basin* example (see Figure 4(a)) is simulated in two experimentation settings. In one, the WEAP system is executed using an algorithm implemented in JScript. In another, the WEAP RESTful framework is used to execute the example model. Figure 8 shows the different stages in running the WEAP system using the JScript and the WEAP RESTful framework. As can be seen, the first two stages are the same in the two experimentation settings. The initialization includes defining configuration properties and the values for input variables (via direct setting or reading from a file). This stage is followed by a non-interruptible execution of the simulation from the start year to the end year. The results of the simulation can be saved in the Excel or CSV formats in one experimentation setting and in the JSON format in the other. The framework incur overhead due to the componentization stage (see the loop part in the sequence diagram shown in Figure 6). The amount of the return data (i.e., execution of the domain model) affects the execution time of simulations. The *Weaping River Basin* model is executed in different configurations. The model includes 3 Rivers, 2 Reservoirs, 2 Groundwater, 6 Demand Sites, 8 Transmission Links, 2 Wastewater Treatment Plants, 12 Return Flows, 1 Run of River Hydro, and 3 Flow Requirements.

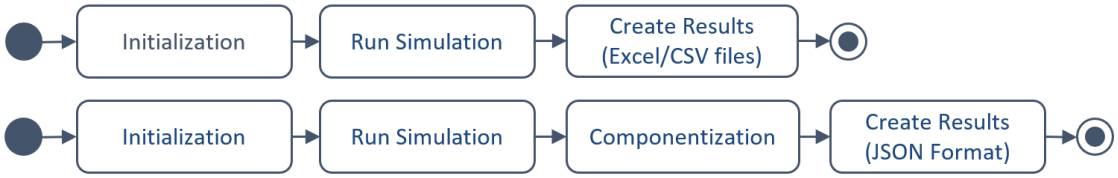


Figure 8: Execution stages of the WEAP system via the JScript and the WEAP RESTful framework.

A personal computer with 20 GB RAM and Core i5 Intel CPU on Windows10 64 bits is used for running all the experiments shown in Figure 1. The execution times for different time intervals (from 2000 to 2030 with 5 year interval) in monthly and daily time steps (12 and 365 steps per year) are collected for the two experimentation settings. All execution times are in seconds and averaged over 10 replications.

Figure 1 shows that the most important difference between the JScript and the Framework is in for *Create Result* computation time (*Total Execution Time* is the sum of *Calculation Time* and *Create Result Time*). In both the script and framework, the *Calculation Time* is the time of running simulation via WEAP system. The small differences between the calculation time for the same configuration in the JScript and the framework can be contributed to the available system-resources during the simulation and the creation of the WEAP instance (using ActiveXObject for the JScript vs. using the Winax package for the framework). In some cases the JScript executes faster. The total execution times for the JScript and the framework are shown in Figure 1. The largest overhead for the framework is around 8 percent and in most cases the overhead is 3 to 4 percent. The experiments above show that the WEAP RESTful framework has negligible overhead computational cost.

The WEAP RESTful framework is developed using NodeJS and Typescript. The NodeJS is used for implementing the server side application; an Event-Driven, Non-Blocking I/O Model and Open Source C++ framework (Cantelon et al. 2013). The NodeJS is built on V8 (the engine written by google) and add some features to handle the JavaScript server-side programming. Typescript is a superset of JavaScript built by Microsoft in 2012. It is open-source and support strongly-typed programming (in comparison to JavaScript which is duck-type). It is cross-platform and have capability to define Module and Namespace, in addition to defining Generic, Interface and Abstract classes.

The main packages which have been used in the framework are *TS-Node 7.0.1* (Typescript-Node) to use Typescript in the NodeJS server side application; *Express 4.16.3* to build web application and APIs;

Routing-Controller 0.7.7 to create structured, declarative and beautifully organized class-based controllers; *Body-Parser 1.18.3* to parse the body of incoming request to web-server; and *Winax 1.0.14* to define *ActiveXObject* in NodeJS (create WEAP instance in server side application).

Table 2: Performance evaluations (measured in seconds) for the JScript and RESTful framework.

start	end	Monthly (12 time-steps per year)							Daily (365 time-steps per year)						
		Script			Framework				Script			Framework			
		Calculation Time	Create Result Time	Total Execution Time	Calculation Time	Create Result Time	Total Execution Time	Execution Time Ratio (Framework/Script)	Calculation Time	Create Result Time	Total Execution Time	Calculation Time	Create Result Time	Total Execution Time	Execution Time Ratio (Framework/Script)
2000	2000	1.2	0	1.2	1.2	0.1	1.3	**1.08**	3.1	0	3.1	3.2	0.1	3.3	**1.06**
	2004	3.3	0	3.3	3.3	0.1	3.4	**1.03**	39.4	0	39.4	38.5	0.5	39	**0.99**
	2009	5.4	0	5.4	5.4	0.2	5.6	**1.03**	84.3	0.1	84.5	83.9	3.3	87.2	**1.03**
	2014	7.3	0	7.3	7.4	0.1	7.5	**1.03**	128.6	0.2	128.8	129	4.8	133.8	**1.04**
	2019	9.5	0	9.5	9.7	0.1	9.8	**1.03**	175.1	0.2	175.3	176	6.6	182.6	**1.04**
	2024	11.4	0	11.4	11.7	0.1	11.8	**1.03**	222.6	0.3	222.9	221.7	8.3	230	**1.03**
	2029	13.4	0	13.4	13.7	0.1	13.8	**1.03**	268.4	0.3	268.7	269.7	10	279.7	**1.04**

6 CONCLUSIONS

Systems-of-systems can be simulated at higher fidelity using models that are more accurate and have more details, preferably with theoretical groundings. It is also useful for modeling and simulation frameworks and tools to be acceptable to its user communities, for example from the standpoint of the user needs such as ease of developing models and experiments. Tools such as WEAP are used by application domain researchers and practitioners. A tool such as the WEAP system provides the basic means for subjecting its internalized entities to inputs and observing outputs externally. To enable this kind of tool, it is helpful to componentize its entities. Using the concept, a RESTful framework is developed for the WEAP system. This framework has the model components for all the model entities that are included in the WEAP system. The model components replicated externally to the WEAP system allow using the WEAP system as a component-based tool supported within the RESTful framework. The WEAP RESTful framework is planned to be used with modeling and simulation tools of food and energy systems and thus supporting studies of Food- Energy-Water systems.

ACKNOWLEDGMENT

This research is funded under the NSF grant #CNS-1639227. We acknowledge fruitful discussions with our collaborators on the Food-Energy-Water project.

REFERENCES

Automating WEAP (API). http://www.weap21.org/WebHelp/API.htm. Accessed Jan. 6, 2019.

Brutzman, D., K. Morse, J. M. Pullen, and M. Zyda. 2002. “Extensible Modeling and Simulation Framework (XMSF): Challenges for Web-based modeling and simulation”. Monterey, CA: Naval Postgraduate School.

Cantelon, M., M. Harter, T. J. Holowaychuk, and N. Rajlich. 2013. *Node.js in Action*. 1st ed. Greenwich, CT, USA: Manning Publications Co.

Daher, B. T., and R. H. Mohtar. 2015. "Water–energy–food (WEF) Nexus Tool 2.0: guiding integrative resource planning and decision-making". Water International 40.5-6:, pp. 748-771.

Fielding, R. T., R. N. Taylor. 2000. *Architectural styles and the design of network-based software architectures*, vol. 7. USA: University of California, Irvine.

Hoff, H. 2011. "Understanding the Nexus, Background Paper for the Bonn 2011 Conference: The Water, Energy and Food Security Nexus". Stockholm Environment Institute (SEI), Stockholm, Sweden.

Markstrom, S. L., R. G. Niswonger, R. S. Regan, D. E. Prudic, and P. M. Barlow. 2008. "GSFLOW - Coupled Ground-water and Surface-water FLOW model based on the integration of PRMS and MODFLOW-2005": U.S. Geological Survey Techniques and Methods 6-D1, 240 p.

Markstrom, S. L., R. S. Regan, L. E. Hay, R. J. Viger, R. M. T. Webb, R. A. Payn, and J. H. LaFontaine. 2015. "PRMS-IV, the precipitation-runoff modeling system, version 4". US Geological Survey Techniques and Methods.

Pullen, J.M., R. Brunton, D.P. Brutzman, D. Drake, M.R. Hieb, K.L. Morse, A. Tolk. 2005. "Using web services to integrate heterogeneous simulations in a grid environment". Future Generation Computer Systems, 21(1): pp. 97-106.

Sarjoughian, H. S. 2006. "Model Composability". In *Proceedings of the 2006 Winter Simulation Conference*, pp. 149-158.

Sieber, J., and D. Purkey. 2005. "Water Evaluation and Planning System (WEAP): User Guide". Stockholm Environment Institute, Boston.

Tihomirovs, J., and J. Grabis. 2016. "Comparison of soap and rest based web services using software evaluation metrics," Information Technology and Management Science, vol. 19(1), pp. 92–97.

WEAP21, Water Evaluation And Planning. http://www.weap21.org. Accessed Jan. 6, 2019.

White, D., J. Jones, R. Maciejewski, R. Aggarwal, G. Mascaro. 2017. "Stakeholder Analysis for the Food-Energy-Water Nexus in Phoenix, Arizona: Implications for Nexus Governance". Sustainability 9(12): 2204.

Zeigler, B. P., and H. S. Sarjoughian. 2017. *Guide to Modeling and Simulation of Systems of Systems*. 2nd edition, Springer Science.

AUTHOR BIOGRAPHIES

MOSTAFA D. FARD is a Ph.D. student in Computer Science program in the School of Computing, Informatics, and Decision Systems Engineering (CIDSE) at Arizona State University, Tempe, AZ, USA. He can be contacted at smd.fard@asu.edu.

HESSAM S. SARJOUGHIAN is an Associate Professor of Computer Science and Computer Engineering in the School of Computing, Informatics, and Decision Systems Engineering (CIDSE) at Arizona State University, and the co-director of the Arizona Center for Integrative Modeling & Simulation (ACIMS), Tempe, AZ, USA. His research interests include model theory, poly-formalism modeling, collaborative modeling, simulation for complexity science, and M&S frameworks/tools. He can be contacted at sarjoughian@asu.edu.