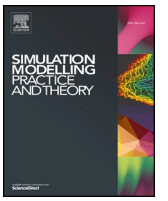
Simulation Modelling Practice and Theory 106 (2021) 102199

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

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



A RESTful framework design for componentizing the water evaluation and planning (WEAP) system

Mostafa D. Fard, Hessam S. Sarjoughian*

Arizona State University, Arizona Center for Integrative Modeling & Simulation, School of Computing, Informatics, and Decision Systems Engineering, Tempe, AZ 85281, United States

ARTICLE INFO

Keywords:
Computational modeling
System integration
Systems modeling
Web services
WEAP

ABSTRACT

The Water Evaluation and Planning (WEAP) system, a modeling and simulation tool, offers certain advantages for studying water systems. These systems are modeled as a collection of supply, demand, and transmission entities. Domain experts can construct whole water system models from template model parts (entities) with built-in constraints for combining them. The parameterized entities are simulated with the aid of textual and visual experiment set-ups. A set of scripts are provided for reading and writing data in entities. Lack of access to the source code of these model entities makes using integrating them other kinds of models more difficult. This is a crucial restriction since modeling water systems increasingly need to be used with the models, for example, energy and food systems to improve understanding and predictions of water supply and demand policies. Given the significance of combining water system models with other types of models, this paper proposes developing a Componentized-WEAP (C-WEAP) by embedding the WEAP system in a RESTful framework. The water system entities are mapped to meta-components using the Ecore modeling methodology. Each meta-component is paired with its concrete counterpart in the WEAP system. A complete set of meta-components for the entities is developed. An existing water system model is developed in the C-WEAP RESTful framework. The simulation of this water system demonstrates the computational cost of the proposed framework is negligible. The developed C-WEAP RESTful framework promotes integrating it with other software systems.

1. Introduction

It is common practice for modelers to contemplate questions of interest by examining systems in terms of their parts and relationships. This approach allows some parts of a system-of-systems to be modeled in detail while all other parts that affect or affected by it to be simple or even excluded. This is attractive as system complexity and scale can be significantly constrained by replacing the dynamics of a system as inputs and outputs. For example, in modeling a water system that uses solar energy, the amount of available photovoltaic energy can be modeled as piecewise input regimes. A key consequence of this choice is that the input regime is non-functional. In contrast, a reactive model produces outputs in part based on consuming inputs dynamically from other models. This photovoltaic model supplies energy to the water system subject to water demand fluctuation can lead to a better understanding of a water system that cannot be achieved through input data alone. From this vantage point, the need for component-based modeling and simulation is evident for understanding the interactions (nexus) among different parts, for example, Food-Energy-Water (FEW)

* Corresponding author.
E-mail address: Hessam.Sarjoughian@asu.edu (H.S. Sarjoughian).

https://doi.org/10.1016/j.simpat.2020.102199
Received 9 April 2020; Received in revised form 4 September 2020; Accepted 21 September 2020
Available online 26 September 2020
1569-190X/

systems [1]. Simulations developed using component-based modeling approaches are important in detailing different kinds of behaviors belonging to different parts of a system-of-systems.

Some existing and popular modeling and simulation tools and frameworks appear to be component-based even though they are not. An example is the Water Evaluation and Planning (WEAP) system for studying water supply and demand [2]. In this framework, models of water systems can be defined via a set of node and link entities assigned to a geographic region. Mass-balanced equations are defined using shared variables (inputs and outputs for the node and link models). The WEAP system supports some scripting languages for manipulating and executing the entities and their data. Approaches that use shared variables amongst models lack sufficient flexibility afforded by component-based modeling frameworks. Consequently, it would be challenging to use the WEAP system with component-based modeling and simulation environments such as DEVS-Suite [3]. A desirable modeling framework should aid combing models developed in different frameworks and their tools. In a component-based modeling framework, each model is a standalone component having its inputs and outputs and functions encapsulated and thus promote modularity, which is a key enabler for synthesizing hierarchical models [4]. In component-based modeling frameworks, models can act on one another. Componentization of the tools such as WEAP and Long-range Energy and Planning (LEAP) system [5] can further their use for system-of-systems modeling, including the class of FEW systems [6].

The overarching research question is on integrating the WEAP system (a legacy modeling and simulation system) with other modeling and simulation tools/frameworks to model and simulate systems of systems. This need requires exposing the entities of water models in the WEAP system to be represented as external logical components to be combined with models of other systems, including energy and food systems. The preliminary work on this research described a preliminary design and implementation using the idea of software components for the WEAP system [6]. The WEAP system was briefly described, along with its role in modeling and simulating the Food-Energy-Water nexus. In this work, the basic design with a prototype of the WEAP RESTful framework was briefly described. The performance efficiency of the RESTful framework compared with the JScript implementation was also presented.

This paper's contributions include new insight into the development of the Componentized-WEAP (C-WEAP) RESTful framework through providing the underlying details and formulation of the WEAP entities as proxy component models using the Model Driven Architecture (MDA) approach and the UML diagrams (moving between different levels of abstraction). These models are defined using the Ecore meta-modeling approach, where every proxy model component corresponds to every WEAP entity. The Ecore is a meta-level design abstraction that maps to concrete-level design abstraction following the MDA approach. The Ecore, in turn, is mapped to the UML diagrams, and then employed to design the RESTful services for the proposed C-WEAP framework. An analytical formulation is developed to show the relationship between the computational scalability of the C-WEAP framework relative to that of the WEAP system. More related works are considered and discussed. Finally, detailed performance analysis shows the computational cost of the RESTful framework to be negligible when compared with the benefits of WEAP componentization.

Section II of the paper describes in detail the WEAP system from the vantage point of component-based modeling. Section III describes selected related works. Also, some approaches for water modeling are described briefly. Section IV describes the C-WEAP RESTful framework architecture at different abstract levels (Ecore, UML diagram, and RESTful service). Section V presents the performance efficiency and analysis of the C-WEAP RESTful framework. Finally, the paper ends up with a conclusion in section VI.

2. Background

The WEAP system capabilities are described with the focus on its use with other frameworks and tools. The RESTful framework, as the underlying framework for the development of the C-WEAP, is briefly described. Together they are aimed at providing details for the remaining sections.

2.1. Water evaluation and planning system

The WEAP system is a framework for modeling, simulating, and evaluating water systems [2]. Models are defined as a network of water supply and demand entities that are connected via transportation entities. A WEAP model defines the allocation of water from different sources through preferences and mass balance constraints. The WEAP system provides a set of entities and procedures to study and find solutions to the problems faced by decision-makers. Using a scenario-based approach, each study area has natural watersheds, reservoirs, streams, and canals that serve to supply demands by a variety of users, including households, industry, and agriculture [7]. The WEAP tool is widely used in the world for water allocation and water management [8–12].

The development of a WEAP model includes several steps [13]. A study is defined to have a time frame, a spatial boundary (see Fig. 1), system entities, and configuration. The Current Accounts, which can be used for calibration of a model, provides a snapshot of actual water demand, pollution loads, resources and supplies for the system. Scenarios are defined using the Current Accounts and allow one to explore the impact of alternative assumptions or policies on future water availability and use. Finally, the scenarios are evaluated regarding water sufficiency, costs and benefits, compatibility with environmental targets, and sensitivity to uncertainty in key variables.

As a tightly integrated modeling, simulation, and analysis tool, the WEAP tool consists of five views/parts to model different aspects of a water system. The structure of the model must be defined in the Schematic view. A water model in the WEAP system is a graph of node and link entities shown in the Schematic view (the solid red box in Fig. 2(a)). The predefined node and link entities can be used to construct the nodes and links assigned to a geospatial map (the dotted blue box in Fig. 2(a)). The entity types are the river, diversion, reservoir, groundwater, desalination plant, demand site, catchment, wastewater treatment plant, runoff, transmission link,

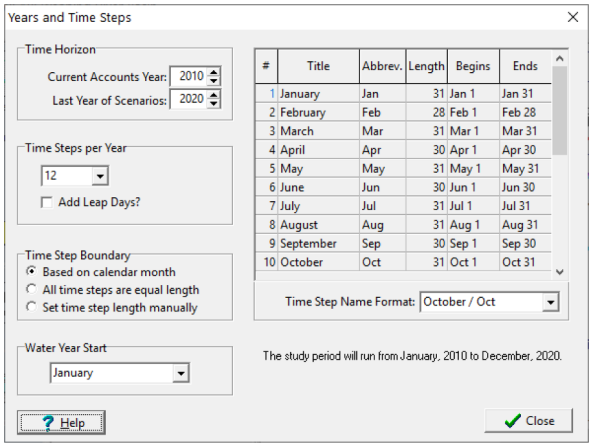


Fig. 1. General configuration in the WEAP system.

return flow, run of river hydro, flow requirement, and streamflow gage. Some entities are presented as nodes (like demand site, groundwater, etc.), and some are presented as links between two nodes (like transmission, return flow, etc.). The primary information about a node, like *name* must be set in the Schematic view. Also, *status* (active or deactivate) and *priority* can be set for nodes and links, as well. In Fig. 2(a), the number between parenthesis next to each entity type (River, Demand Site, etc.) shows its quantity. The number between parenthesis for each entity in the geospatial map indicates its priority relative to other entities during simulation executions.

The defined entities in a Schematic view have predefined inputs and outputs (called data and result variables in the WEAP system). The next part in the WEAP system is Data view which serves to parameterize the inputs and equations for the entities in a tree structure which has the Supply and Resources, Demand Sites and Catchments, Hydrology, Water Quality, and Key Assumptions categories (the solid red box in Fig. 2(b)). The Data view allows a modeler to create variables and relationships, enter assumptions and projections using mathematical expressions, the time-series wizard, and link to external files (e.g., CSV or Excel data file). The input variables of an entity are separated in different categories according to their usage. For example, Fig. 2(b) (the dotted blue box) shows the *West City* demand site has seven categories (*Water Use, Loss and Reuse, Demand Management, Water Quality, Cost, Priority*, and *Advanced*), and *Annual Activity Level, Annual Water Use Rate, Monthly Variation*, and *Consumption* variables belong to the *Water Use* category.

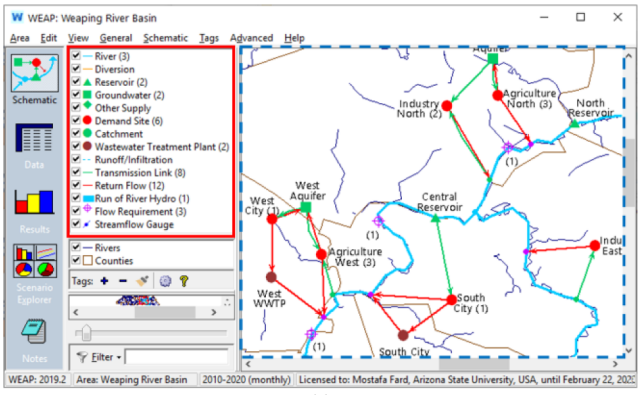
The Results view is for choosing the outputs of the simulation to be extracted and viewed in charts, tables, and on the Schematic view (see Fig. 2(c)). Also, different entities, scenarios, years, and units can be used as plots displaying variable values for time-steps. The data can be filtered for a detailed and flexible display of the model input and output data values for time-step trajectories. The Scenario Explorer view can be used to select experiments to be observed and stored for post-processing. It provides the facility to observe the changes in the selected outputs by changing inputs. The Notes view provides a place to add the documentation for each entity.

From the modeling perspective, the structure of the model (entities and their connections) must be defined in the Schematic view. Then, the data will be injected into the inputs of the model via the Data view. Finally, after the simulation execution, the output data of the model is observable via the Result view. The experiments for a model in the WEAP system are designed via scenarios and general configuration (see Fig. 1). The *"Current Accounts"* scenario is considered as the initialization for the model, and other scenarios can be defined to observe the effects of changing assumptions or policies. The Scenario Explorer and Notes views of the WEAP system do not have any impact on the structure or behavior of a model.

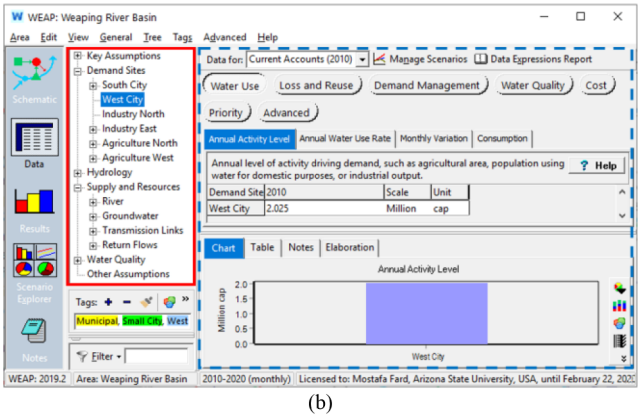
From the simulation perspective, the WEAP system is based on Discrete-Time System Specification [14], and it has an uninterruptible execution. All input data must be ready before the start of every simulation. All output data will be accessible after a whole simulation execution period is completed (from the start year to the end year, according to the general configuration). Interrupting the WEAP simulation midstream and resuming it is not allowed. In this respect, WEAP models are not reactive since they cannot have input from any external simulation model while being executed.

The WEAP system has a set of predefined entities for common supply resources such as reservoirs, facilities such as transmission links, and demand sites such as cities. New entities can be added by the WEAP development team due to the WEAP system being proprietary. Even though entities with their input and output variables are known, their mass-balance equations cannot be discovered from outside. The variables for the entities appear to be shared within the WEAP system. A water model's logical and schematic parts are tightly interwoven to the scenarios, general configuration, and results.

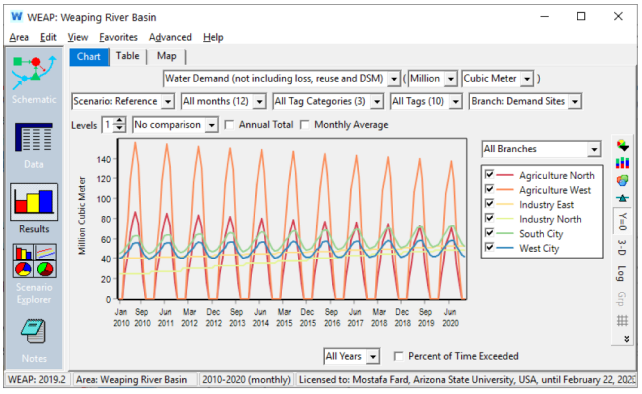
Fig. 3(a) presents the data schema related to an entity from the outside perspective for a prototypical model and scenario. The data schema in Fig. 3(a) has three axes – the y-axis is used for variable names, the z-axis is used for the years of a simulation, and the x-axis is used for yearly time granularity. Each entity (e.g., a demand site) has several input and output variables. A variable can have annual time granularity like variable v_2 shown in Fig. 3(a) which has one value per year, or finer time granularity (for example, monthly, weekly, daily, etc.) like variables v_1 shown in Fig. 3(a). The time granularity for variables which do not have annual time-



(a)

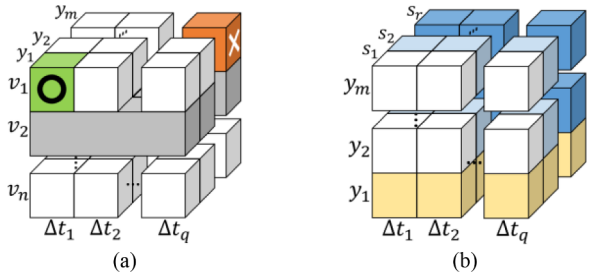


(b)



(c)

Fig. 2. The WEAP system views (a) Schematic view. (b) Data view. (c) Result view.



(a) (b)

Fig. 3. WEAP's component data schema. (a) data schema for different variables of a component related to a scenario. (b) data schema for a specific variable related to different scenarios.

steps must be defined in the General Configuration (*Time Step per Year* section in Fig. 1). Each cube in Fig. 3 has a float-type data value according to its defined time granularity. Years and time-steps in each year have ascending order. For example, the green (which has a circle sign) and orange (which has a cross sign) cubes in Fig. 3(a) present the first and last timestamp's values for the variable v_1 in a simulation scenario. Considering Fig. 3(a), there are $m \times q$ timestamp's values (number of years × number of time-steps per year) for variable v_1. The variable v_2 for the same simulation experiment has m timestamp's values for years y_1, y_2, ..., y_m, each having one time-step.

Fig. 3(b) shows the data schema for one variable related to multiple scenarios. Considering any variable in Fig. 3(a), the y-axis is used for the years of a simulation, the z-axis is used for the number of simulation scenarios, and the x-axis is used for yearly time granularity. For example, suppose variable v_1 is selected from Fig. 3(a) and expanded for multiple scenarios. The result will be Fig. 3(b) presenting the data cube values for scenarios $s_1, \cdots, s_r$ for years y_1, ..., y_m each divided to q equal time-steps. It is important to note that the values in the lowest horizontal level (the yellow cubes) are the same for all scenarios because they represent the values for *Current Accounts* scenario. Other scenarios affect the values of a variable after the start year (start year + 1) to the end year.

The WEAP system uses mixed-integer linear programming (MILP) to optimize the satisfaction of requirements for the demand sites, reservoir filling, user-specified instream flows, and hydropower entities subject to demand priorities, supply preferences, mass balance and other constraints [13]. The WEAP system supports LPSolve, XA, and Gurobi MILP solvers. The LPSolve is open source and included in the WEAP system. XA and Gurobi are commercial products [15]. For very large models, the commercial solvers can perform much faster. The simulation results can vary slightly depending on the selected solver.

The WEAP system Application Programming Interfaces (APIs) support the VB-Script, JavaScript, Perl, and Python languages using the standard COM Automation Server. The APIs allow reading and writing external data values for the variables shown in Fig. 3. For any framework to use the WEAP system, it must conform to the provided.

2.2. RESTful web services

Web Services refer to independent software-centric components communicating amongst each other [16,17]. There are two main web-service protocols. One is the Simple Object Access Protocol (SOAP) [18] and the other is the REpresentational State Transfer (REST) [19]. The former is an XML-based standard communication specification over a particular protocol such as HTTP and SMTP. The latter is a web-based architectural style with flexible message formats such as XML and JSON. The REST supports specific client-server communication, stateless operation, uniform interface, and resource caching. From the client perspective, the SOAP is based on the operation/method, whereas the REST is based on the resource. The RESTful framework is an implementation of the REST architecture based on the HTTP protocol [20]. Asynchronous requests, higher security and reliability, and error reduction are the main reasons for choosing the SOAP standard. Greater scalability, compatibility, performance, and simplicity are the common reasons for choosing the REST standard.

Web services such as the Extensible Modeling and Simulation Framework (XMSF) [21] are defined as an integrable set of standards, profiles, and recommended practices for web-based modeling & simulation [22]. The XMSF supports the migration of legacy components into web-enabled components for distributed heterogeneous simulation applications. It is based on SOAP and XML [21], whereas the proposed WEAP web-service system benefits from the RESTful web services and the Ecore modeling framework.

3. Related work

There exist a wide variety of many tools for modeling and simulating water systems, serving purposes ranging from natural hydrological processes to engineered distribution networks. Such software tools are developed by representing water systems as data sets with functions, objects, and services. Legacy and object-oriented software systems can be encapsulated as services in Service Oriented Architecture (SOA) paradigm. Various approaches have been proposed for transforming legacy software systems to be integrable with other software systems [23]. One of these approaches is “wrapping” where any proprietary legacy software system with input/output API (e.g., WEAP system) can be encapsulated inside other software systems [24,25]. Individual functions in the legacy software are wrapped into web services. New components are designed according to the code segments that perform a service or data modification. Each new component is given a Web Services Description Language (WSDL) interface and targeted for either SOAP or RESTful framework. This research follows the rationale and the general approach of transforming a legacy system to flexible service-oriented software frameworks in addition to component-based modeling and simulation [4].

Services and service compositions with resource management supporting load balancing and context storage are proposed for legacy modeling and simulation tools [26]. This approach is applied to a legacy solid multibody simulation. This tool consists of five Fortran programs which are wrapped individually in Java-based web-services. The Message Passing Interface (MPI) [27] concept is used to define the communication between process (or communication) logic and domain logic with workflows/service compositions. The result improves the existing legacy application and speeds up the overall simulation execution by automating manual tasks and parallelizing web-services to control, orchestrate, and visualize simulation experiments. The C-WEAP RESTful framework shares the wrapping legacy software applications; however, its primary goal is to support structured integration with other modeling and simulation tools and generally scientific applications.

An objected-oriented modeling framework without using the SOA paradigm is proposed for simulating watershed flow and sediment processes at the catchment scale based on fine-grained components [28]. This work is based on the systems-of-systems concept to build new water-related models quickly and effectively by de/composing models in the manner of plug and play of user-

defined components. The objects from decomposition are encapsulated into corresponding components using dependency injection, a technique for achieving separation of concerns, to define the relationships and coupling of different components.

A web-based platform has been introduced to support efficient multivariate visualization of environmental data from sensor observation networks [29]. Data are collected based on the SOAP framework and XML format from different locations/resources for creating various visualizations and analysis using JSON. This platform uses a caching system (collected from distributed sensors via web-services) to store data in databases (PostgreSQL for local storing and Hadoop-based OpenTSDB for distributed storing) to increase data access efficiency. Also, a data cube model is established to reshape heterogeneous data and support unified data operations.

Considering simulation studies of integrated food, energy, and water (FEW) systems, frameworks, and tools such as Precipitation Runoff Modeling System (PRMS) [30] and Ground-water and Surface-water FLOW (GSFLOW) [31] and WEF-Nexus Tool 2.0 [32] have been developed. These tools are not based on component-based modeling principles and service-oriented computing. The PRMS is a deterministic, distributed-parameter, physical process-based modeling system developed to evaluate the impacts of various combinations of climate and land use on surface-water runoff, sediment yields, and general basin hydrology. The GSFLOW is designed to simulate coupled ground-water and surface-water systems. The WEF Nexus Tool 2.0 is a scenario-based tool for guiding resource allocation at the country level for a given level of food self-sufficiency and a set of technologies, land uses, and resource availabilities. These approaches and tools, unlike those briefly described above, can provide limited capabilities needed for integrating food, energy, and water models developed in different simulation tools [33].

Agent-based modeling is also proposed and used for water resource management system. Agents have their own goals and behaviors and can adapt and modify their behaviors [34]. In water resources systems modeling, agents can be individual ground and surface water users, water polluters, various infrastructural elements, cities, or policymakers on different levels [35]. A framework to model and simulate water supply and demand for urban households has been developed based on the Agent-Based Modeling and System Dynamic modeling [36]. This is an approach to model water management at a micro (short-term) and macro (long-term) abstraction levels. Another approach for simulating urban water resource management uses a multiagent Q-learning-based allocation agent-based algorithm (with adaptive reward value function to improve the performance) [37]. This algorithm supports allocating water resources efficiently among stakeholders. An agent-based framework has also been developed to simulate the behavioral characteristics of urban water users while accounting for their social interactions [38]. A model has interactions between agents as well as agents and environments. The focus of this framework is to help study and evaluate the impact of different climate and government policies. Although agent-based frameworks are inherently grounded in the concepts of components, they are not as flexible and scalable as service-oriented frameworks and thus can be challenging to be loosely integrated with other tools.

The WEAP system is used as the primary tool for modeling and simulating water management under different socioeconomic and climate scenarios for regions including South Africa, China, Greece, Benin, and Pakistan [8–12]. This suggests that the WEAP system's componentization can help combine these and other WEAP water simulations with separate simulations for agriculture, climate, and energy systems. Given the related works highlighted above, the C-WEAP is a tool built using the RESTful SOA architecture, and the data cube structure allows integrating it with tools for simulating energy and food systems [39].

4. A web service framework for the WEAP system

According to the constraint for using the JavaScript language (to invoke Automating the use of WEAP APIs [40]) and the difficulties of using XML-based protocols (extensive code development to create XML structure) [41], the RESTful framework is used to implement the web-service framework for the WEAP system. This framework's lightweight, fast, and scalability traits are the basis for its use for the development of the C-WEAP RESTful framework.

4.1. Components of the WEAP model

To componentize the WEAP system, the entities that are included in the WEAP system with their data are mapped to components using the Ecore Modeling Framework (EMF) [42,43]. The Schematic, Data, and Result views of the WEAP system are designed according to the Ecore meta-modeling [43]. The Ecore meta-model is used to model the WEAP entities at an abstract view without specifying their functions. At this abstraction level, the data structure of different WEAP sections, entities, variables, and the relationship among these parts are modeled. The specification in Fig. 4 is defined using the EClass, EAttribute, EDataType, and EReference elements of the Ecore meta-model diagrams. A UML diagram is presented in Section V to describe the actual implementation of the C-WEAP RESTful framework for the componentization of the WEAP system at the Data Access layer of the RESTful framework architecture (see Section IV.C). It is important to note that the WEAP's APIs expose the scope and functionality of the componentized entities defined for the WEAP framework [44]. For example, it is not possible to add new variables for any entity via WEAP's API, thus adding these variables must be achieved within the WEAP system (see Fig. 2(b)).

In Fig. 4, the WEAP class has an array of projects, each associated with a geographic area. Each project has its own configuration (*name, startYear, endYear, timeStepPerYear*, etc.), which are mapped to the properties in Fig. 1. The Node and Link are two abstract classes that are detailed in the following subsections. The WEAP, Project, Version, and Scenario are concrete classes. These correspond to the entities in the WEAP system which instantiate a specific domain model in the WEAP RESTful framework. The remaining abstract and concrete classes are useful for the design to be simple, yet flexible.

A simulation model in the WEAP system has a structure defined by a modeler. The behaviors for the specialized node and link entities are predefined. The date specified in a scenario is needed to simulate some aspects of a water system. Every project has at

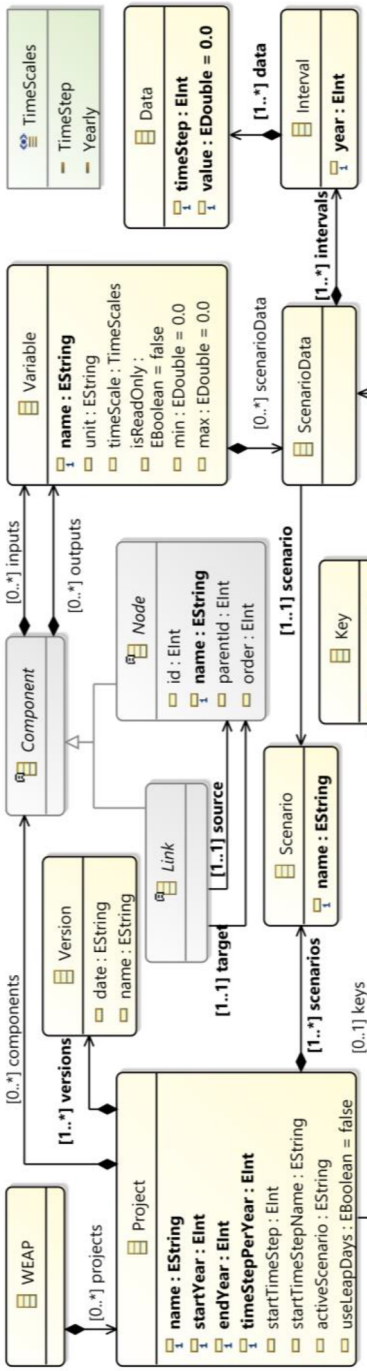


Fig. 4. Ecore specification for model components and their data in the WEAP system.

least one scenario, the *Current Accounts,* which provides a snapshot of actual water demand, pollution loads, resources, and supplies for the system [2]. It is used to define the initial data for the inputs at the start of a simulation. The hierarchical structure of scenario data in the WEAP system does not have any functional role. Each data for a scenario is independent of any other scenario data (see our perspective about the data in Fig. 3). Also, every project has at least one version (refer to the create time of the project). The *name* property is the key attribute in the Project, Scenario, and Node classes (see Fig. 4). The key for the Version is the concatenation of its properties (*date* and *name*). The C-WEAP RESTful framework has a complete set of model components that cover all the entities and variables in the Schematic, Data, and Result views of the WEAP system (see Fig. 2). The derived model components do not add any operations to those provided for the WEAP entities. The model components are categorized into the Node and Link types according to their respective properties (see Fig. 4). Each Link has one source node and one target node.

All model components have some input and output variables (see Fig. 4). The WEAP system has some predefined entity variables and equations. New variables and equations may be added by users to predefined entities as needed. For each Variable, one or more Intervals are defined per scenario, and each Interval can have many Data values (a value represents a specific time-step of a year). The Variable class has a unique *name* property as the key with *unit, timeScale, isReadOnly*, min, and max properties. A variable can have just one value per year if the *timeScale* property sets to Yearly, or it can have multiple values (the number of values should be equal to the *timeStepPerYear* property in Project class) if *timeScale* sets to TimeStep. The value of an input variable cannot change if the *isReadOnly* property is set to True. The properties min and max place constraints on the acceptable values for a variable. From a higher abstraction view shown in Fig. 4, the Node and Link classes with their input and output variables define the structure of a model.

The C-WEAP RESTful framework has the same schema (a generic view) for all the WEAP entity types (e.g., Demand Site, Catchment, and Transmission Link) and their variables. For example, a catchment can have different set of input variables based on its selected simulation method (such as Rainfall Runoff, Irrigation Demand Only, and MABIA), and the C-WEAP framework presents a set of input variables (see Fig. 4) for this entity at a high abstraction level. Thus, two catchments in a project can have different sets of input variables. The Variable, ScenarioData, Scenario, Interval, and Data classes with their relations define the overall input data and output result for a model in the C-WEAP RESTful framework that mirrors those defined in the WEAP system.

As shown in Fig. 5, different nodes (correspond to the entities in Fig. 2(a)) are inherited from Node, Flow, or ReachPoint abstract classes. The River and Diversion nodes have some sub-nodes, which are an ordered collection of reach point nodes. Consequently, a variable of a flow component can have different values in its reach points. The WEAP system has three link entities (Transmission Link, Return Flow, and Runoff). Each link starts from a node and ends at another node with some constraint for the source and target nodes based on the link type [2]. The allowable source and target nodes for the Transmission Link, Return Flow, and Runoff are shown in Fig. 6. The Transmission, Return Flow, and Runoff links have their own input and output variables and data (see Fig. 4).

4.2. Mapping componentized-WEAP components to the RESTful framework

The model components of the WEAP system are the actual resources in the C-WEAP RESTful framework, so a well-defined structure for the URL needs to be present to operate on the resources. The data needed for the RESTful framework is in JSON format. The RESTful API categories are Project, Version, Node, Link, and Flow. The subset of the APIs listed in Appendix A is used in developing the C-WEAP RESTful framework. In the pattern of the URLs, constants are written in *PascalCase* style; parameters start with colons and written in *camelCase* style; query parameters (to apply to some filters on returned data) written after the question mark by *Key* = *Value* (*camelCase* style for the *Key* part). The retrieve, insert, update, and delete operations for each URL are supported with the HTTP GET, POST, PUT and DELETE methods, respectively.

The URL patterns for six API types are shown in Table 1(a). The pattern inside each open and close pair bracket is optional. The appropriate types are presented in Table 1(b). There is a mapping between the URL patterns in Table 1 and the Ecore specifications in Fig. 4, 5 and 6. All URLs start with constant /Water, which refers to the WEAP class shown in Fig. 4. For example, calling "/Water" returns the name of all projects (an array of string) correspond to the composite relation from the WEAP class to the Project class in Fig. 4. When a project is selected using the:projectName parameter, the project configuration information can be read or changed depending on the URL's method. Finally, a model (e.g., demo) can be executed using the URL "/Water/demo/Run". The Version saves the project in different timestamps. The list of versions can be retrieved using the URL "/Water/demo/Version". The project reverts to a specific version (e.g., 20,200,820-test) using the URL "/Water/demo/20,200,820-test/Revert".

In the patterns for the Node, Link, or Flow categories in Table 1(a), the constant types (bold types in the URLs) must be replaced by one of the values in its corresponding type in Table 1(b). The name of a Node or Flow, and the names of the source and target nodes of a Link are used to select a component. For example, the URL "/Water/demo/DemandSite/phoenix" returns the *phoenix* demand site's data of the *demo* project. The VariableType in the URL patterns must be replaced by the *"Inputs"* or *"Outputs"* (refer to the Data or Result variable in the WEAP system). The data of a variable can be retrieved by mentioning the name of the variable and the intended scenario. The expression of a variable can be retrieved by adding the Expression constant in the URL. Query parameters can be used to filter the returned data (the years and time-steps). In the Flow URLs, the subNodeType must be replaced with the corresponding value in Table 1(b) (the reference properties in Flow abstract class in Fig. 5) to access to a specific collection of sub-nodes, and then use:*subNodeName* to select one.

The "*Weaping River Basin"* project is one of the default projects in the WEAP system, with 12 time-steps per year from 2010 to 2020 [7]. Fig. 1 shows the Schematic View of this project in the WEAP system. Fig. 7(a) shows the result of calling an API to get the rivers in the *Weaping River Basic* model. The returned data is an array that contains three River objects. Fig. 7(b) shows the result of calling an API to get the data for the *Annual Activity Level* variable of the *West City* demand site for the *Reference* scenario between

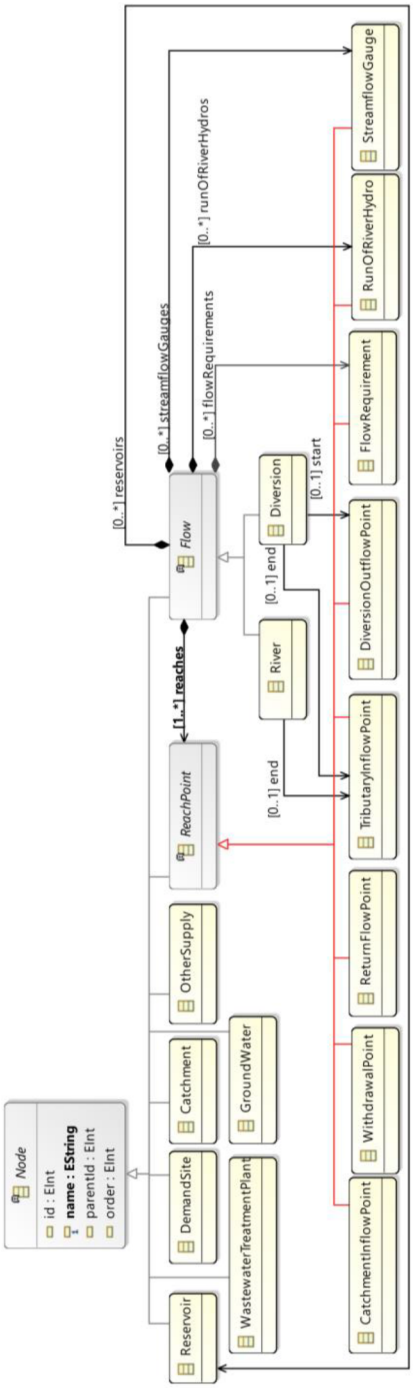


Fig. 5. Ecore specification for the node components in the WEAP RESTful Framework.

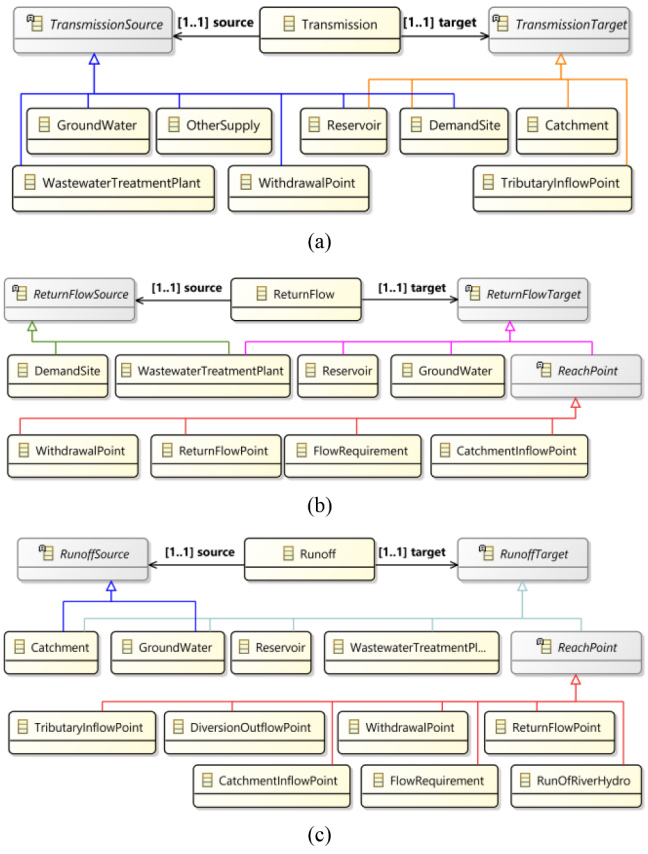


Fig. 6. Ecore specifications for allowable source and target nodes for the (a) Transmission Link, (b) ReturnFlow link, and (c)Runoff.

Table 1
URL pattern for different types of APIs.

(a) Category	URL PATTERN
Project	/Water[/:projectName[/R*un*]]
Version	/Water/:projectName/Versions[/:versionName/Revert]
Key	/Water/:projectName/Keys[/:KeyName/:scenarioName[/E*xpression*]]
Node	/Water/:projectName/**NodeType**[/:nodeName[/**VariableType**[/:variableName/:scenarioName[/E*xpression*] [?startYear = *N*&endYear = *N*&startTimeStep = *N*&endTimeStep = *N*]]]]
Link	/Water/:projectName/**LinkType**[/:sourceName/:targetName[/**VariableType**[/:variableName/:scenarioName[/E*xpression*] [?startYear = *N*&endYear = *N*&startTimeStep = *N*&endTimeStep = *N*]]]]
Flow	/Water/:projectName/**FlowType**[/:flowName[/**subNodeType**[/:*subNodeName*]][/**VariableType**[/:variableName/:scenarioName [/E*xpression*][?&startYear = *N*&endYear = *N*&startTimeStep = *N*&endTimeStep = *N*]]]]

(b) Type	Values
NoteType	Catchments, DemandSites, Groundwaters, Reservoirs, OtherSupplies, WastewaterTreatments
LinkType	Transmissions, Runoffs, ReturnFlows
FlowType	Rivers, Diversions
VariableType	Inputs, Outputs
subNodeType	Reaches, Reservoirs, RunOfRiverHydros, StreamflowGauges, FlowRequirements

2010 and *2012*. The returned data is an array of Interval objects. The variable in this example has a *Yearly* time-scale (see TimeScales enumeration type in Fig. 4), so the result has one instance of Data class (one pair of *timeStep* and *value*). Using a variable with the *TimeStep* time scale will return 12 (due to the *timeStepsPerYear* value of the project) instances of Data class for each year. Tree structures for the Node and Interval shown in Fig. 7 are generated by the C-WEAP RESTful framework.

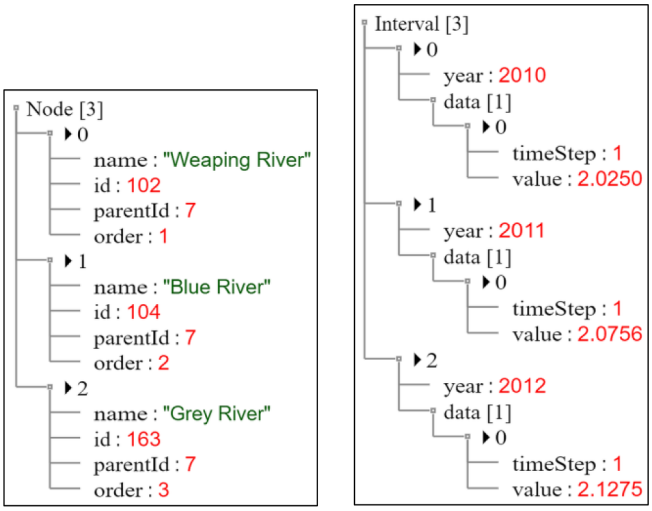


Fig. 7. (a) result of calling the URL= "/Water/Weaping%20River%20Basin /River". (b) result of calling the URL= "/Water/Weaping%20River %20Bas in/DemandSite/West%20City/Annual%20Activity%20Level/Reference?startYear=2010&endYear=2012".

4.3. Design and implementation

The layered architecture of the C-WEAP RESTful framework is shown in Fig. 8. The dotted area indicates the server-side layers of this framework. The WEAP system and the File System are placed at the framework's bottom layer. The externalized WEAP model components and their configurations are stored in a CSV file format (see Section IV.B). The Data Access layer is responsible for ensuring the consistency of the componentized models and their configurations at all times with its WEAP system counterpart. Only this layer has direct access to the File System and to the WEAP system via its APIs [13]. It can communicate with any model that exists in the bottom layer (i.e., for creating and/or executing WEAP models).

All communications between the Data Access and the Controller layers are managed by the layer consisting of the Data Transfer Object (see Fig. 4, 5 and 6) and Service parts. The Service part is responsible for communicating and processing information about the WEAP entities (contained in the bottom layer) via the componentized Data Access layer to the Controller layer. Every C-WEAP model (which is identical to the WEAP system model) can be manipulated by a client application in an independent fashion. The Controller layer contains the web-server and controller parts (not shown in Fig. 8) for handling various client API requests.

A class diagram for the Data Access layer is shown in Fig. 9. This is a realization of the Ecore specification defined in Fig. 4. The WEAPDao class can retrieve a project with a given name, a list of all existing WEAP projects, or execute a project with a given name. Considering the ProjectDao class, it has a WEAP object and the name of a project belonging to it. An Update operation can modify some information about the project. The get operation returns a project with a given name. The ProjectDao class has the operations needed to manipulate all the entities of a WEAP model. One operation is to find the list of all scenarios, and another operation is for selecting a scenario. The generic getNodes and getLinks operations are defined to return a list of all the nodes and links of a project. The getNode operation finds a node with a given name. The getLink operation finds a link with a given pair of source and target nodes. The NodeDao and LinkDao classes are inherited from ComponentDao abstract class. The constructors for these generic classes can find the entries shown in Table 1. An input or output of a component is an instance of the VariableDao class. Values for the input and output ports for a given scenario are defined using the ValueDao class (see Fig. 4). Other details of the classes in the DataAccessLayer package are omitted for brevity.

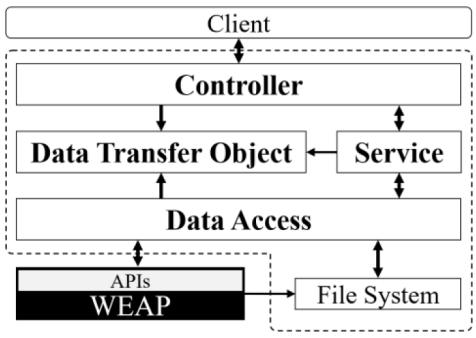


Fig. 8. The componentized-WEAP RESTful framework layer architecture.

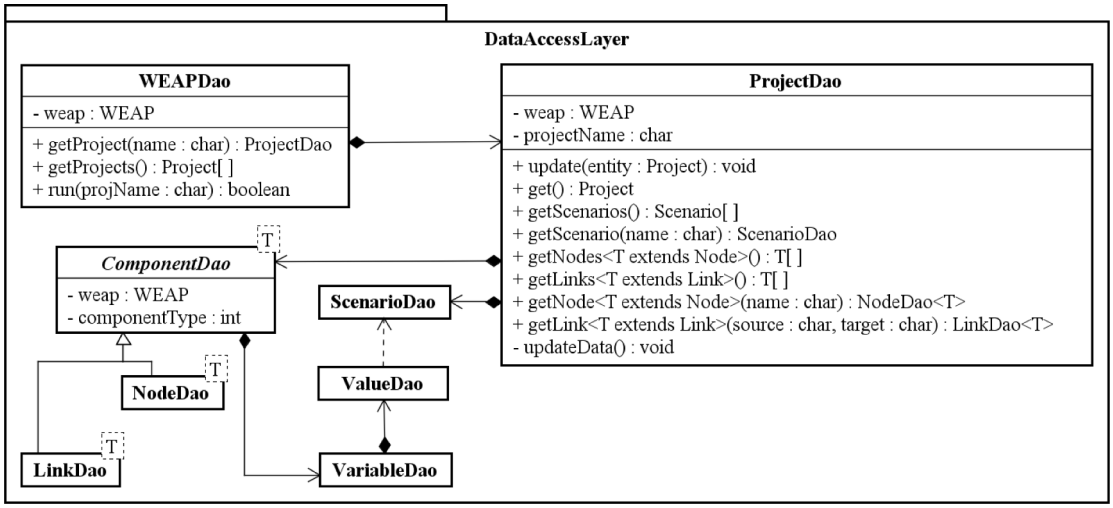


Fig. 9. A class diagram for the Data Access layer of the Componentized-WEAP framework.

A sequence diagram scenario for a client fetching the rivers of the *Weaping River Basin* project is shown in Fig. 11. This specification is devised to show a normal (positive) sequence of messages among a select set of objects instantiated from the classes shown in Fig. 9. At the end of this scenario, the three rivers in the *Weaping River Basin* project are identified. The incoming message 1 (RESTful API request) by the cl object is processed by the ctrl object. Subsequently, in steps 2–4, the svc, wDao, and weap objects are created. In step 5, the ctrl object parses an incoming request to extract some parameter of interest (e.g., a project name). Then, message 6 is invoked on the svc object to find all existing rivers.

The svc object invokes message 7 on the wDao object. A pDao object is created and then returned to the svc object. The svc object invokes message 9 on the pDao object for identifying the rivers in the project. The pDao object invokes message 10 on the weap object which in turn finds the data for the rivers [44]. In the loop section, the riv objects are created in step 11 using the returned array in step 10 (see Fig. 5). The properties of the riv object for each branch is updated in step 12. Finally, the list of the created rivers is returned to the svc, ctrl, and cl objects. This scenario depicts a complete cycle starting from a client application to the WEAP system and ending at the client (see Fig. 8).

4.4. Componentized-WEAP file system

Time-series functions for the variables in the WEAP system do not allow specifying time-step values for all years of the simulation (external CSV/Excel files must be used). Fig. 10 shows the file system structure used by the C-WEAP RESTful framework to store the CSV files and use them with the entities in the WEAP system. The *Workspace* folder is located next to the executable C-WEAP file. Two *inputs.csv* and *outputs.csv* files, under ProjectName folder, are used to configure the *Min, Max*, and *TimeScale* properties for the variables of the WEAP's entities. The data for variables are stored in CSV files under the Data folder. The required folder names, such as project name and component type, are retrieved from the invoked URLs. The *ReadFromFile* function (defined in the WEAP system) is used to make reference to the CSV files [13]. The folder structure shown in Fig. 10 prevents any conflict of the data for different projects, components, variables, and scenarios.

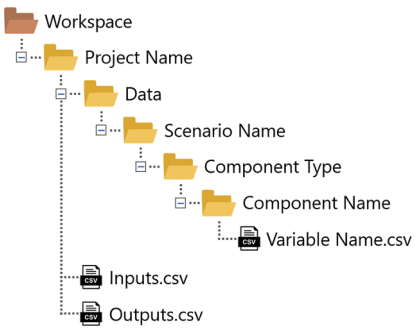


Fig. 10. The Componentized-WEAP Framework file system.

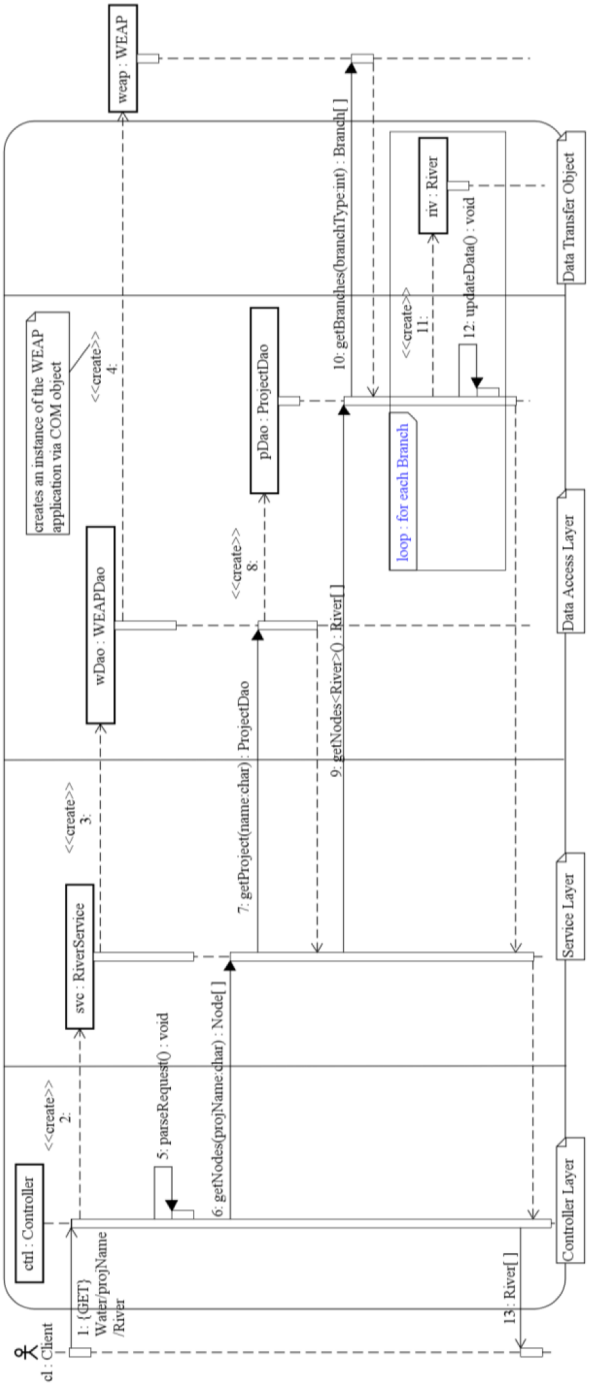


Fig. 11. Sequence diagram to get all rivers of a project via the Componentized-WEAP.

5. Performance evaluation of the componentized-WEAP framework

Using a MILP Solver (LPSolve, XA, or Gurobi) for the WEAP system, or using any other calculation methods for the entities (e.g., Catchment and Demand Site) do not affect the execution of the entities (i.e., the C-WEAP framework does not affect the execution time for any WEAP model). The execution time of the componentized WEAP is higher than the standalone execution of the WEAP system due to the computation time of the RESTful framework. Changes to any project configuration and scenarios do not change the WEAP entities and their structure (see Fig. 4).

As described before, the WEAP system is closed-source software. The Automating WEAP APIs [40] support the VB-Script, JScript, and Python languages to manipulate and execute WEAP models. The C-WEAP system's time efficiency is evaluated against a JScript algorithm (i.e., non-componentized) for the *Weaping River Basin* example (see Fig. 1). In both approaches, the simulated experiments have identical set-up (i.e., properties and the values for the input variables are configured). The executions of these simulations are not interrupted and carried out by the APIs. The C-WEAP RESTful framework requires additional steps for identifying WEAP elements, constructing WEAP components, and de/constructing data (see Section IV).

The elements of the *Weaping River Basin* model are 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. The model is configured for daily and monthly time-steps (12 and 365 steps per year). The efficiency of the C-WEAP RESTful framework relative to its proprietary counterpart for each model configuration is compared for a 30-year period (e.g., 2000–2029) with 5-year intervals. Execution times are measured in seconds and averaged over 10 replications. An isolated personal computer with 20 GB RAM and Core i5 Intel CPU on Windows10 64 bits is used for running all the experiments. The execution times for different time intervals (from 2000 to 2030 with 5 years 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.

5.1. Simulation experiments and comparison

Fig. 12 shows the performance evaluations for the above WEAP script and C-WEAP RESTful framework simulation experiments. There are small differences between the componentized and no-componentized models. These differences are due to the available system-resources and the creation of the WEAP instance using ActiveX and Winax. The execution times for the Script and C-WEAP RESTful framework are shown in Fig. 12(a) and (b) for monthly and daily time-steps, respectively. Table 2 and 3 present the minimum, average, maximum, difference, and average data execution times for monthly and daily scenarios. The *Min, Max*, and *Dif* data show expected variations in the execution times of the Script and RESTful framework.

Fig. 12(c) shows the overhead of using the C-WEAP RESTful framework for daily scenarios for a 30-year simulation period with data collected every 5 years. For each year, the times are the time belongs to the componentization. For example, just 10 s of 279.9 s (see Table 3) for a 30-year simulation scenario belongs to the componentization, and the rest is the execution time of the WEAP system. For this configuration, the computation times in Fig. 12(c) is the maximum overhead of the componentization for the *Weaping River Basin* model. The execution times reduce as less data is retrieved (see Fig. 3). The execution overhead changes linearly from 0.1 to 10 s for one-year to 30-year simulation. This trend has a direct relation to the number of time-steps per each year of the simulation period. For example, the extra computation time for the monthly time-step for the 30-year simulation ($30 \times 12 = 360$ timestamps) is almost the same for the daily time-step for a one-year simulation period ($1 \times 365 = 365$ timestamps).

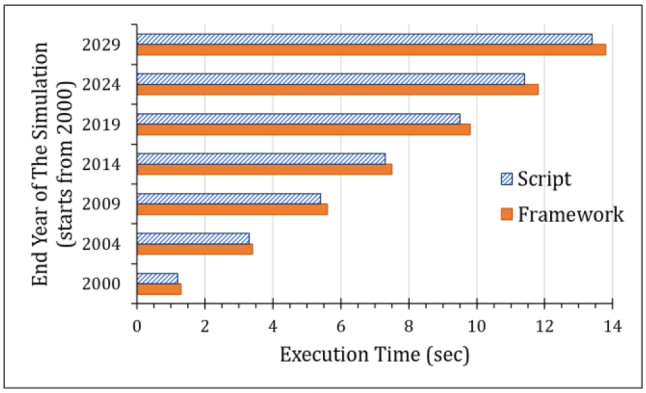
The impact of componentizing the WEAP on total time for simulation studies is negligible. Fig. 12(d) presents the overhead percentage in using the C-WEAP RESTful framework for daily and monthly time-steps (the ratio of the Framework average execution time to the Script average execution time in Table 2 and 3) for the *Weaping River Basin* model. The largest overhead at $\sim$8% is for the first simulation period with the daily time-step. This overhead can be attributed to the model and simulation initialization. For the subsequent simulation periods, the maximum overhead monthly and daily time-steps ranges between 3% and 4%.

5.2. Performance analysis

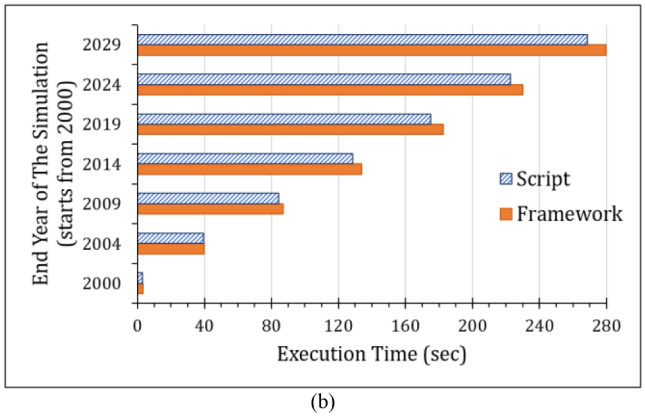
The impact of the WEAP system componentization on the simulation execution time is observed to have a direct relation to the number of timestamps while the scale of a model does not. The overhead of the C-WEAP RESTful framework (*CW*) function is defined as

$$CW = CI + CWE + DS \times TS \tag{1}$$

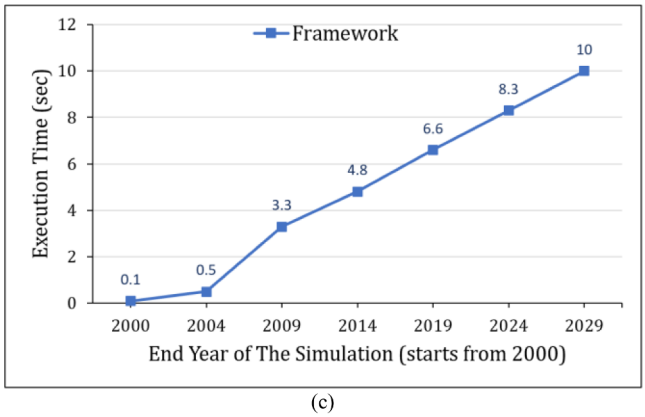
where *CI* is for component identification time, *CWE* is for C-WEAP RESTful framework execution time, and *TS* is the number of timestamps for the duration ($(EndYear - StartYear) \times \#TimeStepPerYear$) of simulation experiment and *DS* is for the data de/construction time. The *CI* factor has a generic implementation (see Section IV.A), so it has a constant value for a model. The *CWE* and *DS* factors have constant values for a given model (see Section IV.C). According to these factors, below a threshold value for the number of timestamps (e.g., arround 400 timesteps for the *Weaping River Basin* simulation), the CI and CWE play the dominant role in total



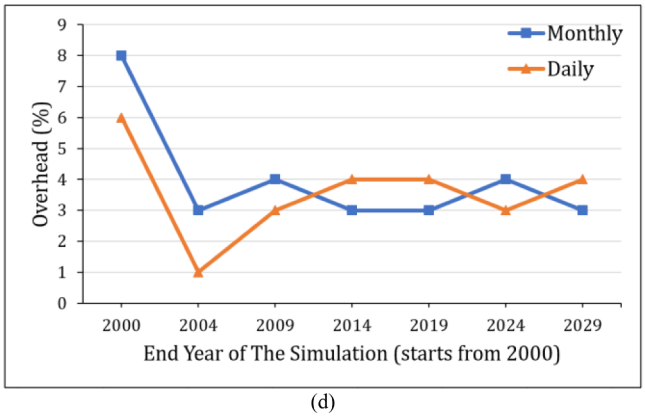
(a)



(b)



(c)



(d)

(caption on next page)

Fig. 12. The C-WEAP RESTful framework performance vs. WEAP script evaluation. (a) Total execution times (monthly timesteps). (b) Total execution times (daily timesteps). (c) C-WEAP RESTful framework overhead (daily timesteps). (d) C-WEAP RESTful framework overhead in comparison to WEAP script.

simulation execution time; otherwise, the TS is the main contributor to total simulation time. For the above simulation experiments, the overhead for the C-WEAP RESTful framework executing on a monthly time-step was 0.1 s for all time intervals. However, it increased for daily time-steps as shown in Fig. 12(c). The execution time ratio of the C-WEAP RESTful framework over the Script WEAP system is defined as

$$Ratio = \frac{Init + WE + CW}{Init + WE} \tag{2}$$

where the *Init* is for the time period required for initializing the WEAP system, and the *WE* is for the time period needed to execute a model in the WEAP system. The *Init* and *WE* factors belong to the WEAP system and are directly related to a model's scale. Thus, the ratio of using the C-WEAP RESTful framework vs. WEAP script will decrease as the scale of the model increases. Table 4 presents the execution times of a more complex *Weaping River Basin* model (the number of entities is 3-times of the previous experiment) for one-year, 15-year, and 30-year simulation period (for Monthly and Daily timesteps). The table shows in these configurations that the componentization's overhead decreases by increasing the scale of the model, having longer execution time for the WEAP system (WE), and constant C-WEAP RESTful framework execution time overhead (CW).

Table 2
The minimum, maximum, and average execution times (monthly timesteps) using the C-WEAP RESTful framework and the WEAP system script.

End Year	Script				Framework			
	Min	Max	Dif	Ave	Min	Max	Dif	Ave
2000	1.1	1.3	0.2	1.2	1.2	1.4	0.2	1.3
2004	3.1	3.5	0.4	3.3	3.3	3.6	0.3	3.4
2009	5.3	5.5	0.2	5.4	5.5	5.7	0.2	5.6
2014	7.2	7.3	0.1	7.3	7.4	7.6	0.2	7.5
2019	9.4	9.8	0.4	9.5	9.6	10	0.4	9.8
2024	11.3	11.5	0.2	11.4	11.7	11.9	0.2	11.8
2029	13.2	13.8	0.6	13.4	13.8	14	0.2	13.8

Table 3
The minimum, maximum, and average execution times (daily timesteps) using the C-WEAP RESTful framework and the WEAP system script.

End Year	Script				Framework			
	Min	Max	Dif	Ave	Min	Max	Dif	Ave
2000	3	3.3	0.3	3.1	3.3	3.4	0.1	3.3
2004	39.1	40.1	1	39.5	39.1	40.2	1.1	39.9
2009	84.2	85	0.8	84.5	87	87.6	0.6	87.2
2014	128.3	130.2	1.9	128.8	133.3	134.1	0.8	133.8
2019	174.1	175.5	1.4	175.3	181.7	184.6	2.9	182.6
2024	222.4	223.5	1.1	222.9	229	230.9	1.9	230
2029	268.4	269.2	0.8	268.7	278	280.4	2.4	279.7

Table 4
The minimum, maximum, and average execution times of the complex Weaping River Basin model using the C-WEAP RESTful framework and the WEAP system script.

End Year		Script			Framework				Ratio(%)
	Min	Max	Dif	Ave	Min	Max	Dif	Ave	
Monthly	2000	1.9	2.2	0.3	2	1.9	2.4	0.5	2.2	1.1
	2014	12.7	14.4	1.7	13.3	13.3	14.7	1.4	13.8	1.04
	2029	23.9	25.4	1.5	24.5	24.1	25.7	1.6	24.8	1.01
Daily	2000	5.8	6.6	0.8	6	6	7.1	1.1	6.3	1.05
	2014	228.9	238.5	10.4	232.4	228.2	238.5	10.3	232.2	1
	2029	455.5	460.3	4.8	458.2	463.4	472	8.6	468.6	1.02

5.3. Componentized-WEAP framework software

The C-WEAP is a web-service framework that uses the NodeJS [45] and Typescript frameworks for implementing the server-side application. Typescript is an open-source framework and the superset of JavaScript [46], which has some added and facilitated features (strongly typed programming, module and namespace, generic, interface, and abstraction). The current C-WEAP implementation requires using the version 2019.2 of the commercial WEAP system [13]. SEI can publish new versions for the WEAP system, but thus far, the changes are UI related. The WEAP system has numerous APIs, but the C-WEAP framework uses a portion of them (see Appendix A). Changes to the C-WEAP RESTful framework are not anticipated, as the models have remained unchanged for several years. Furthermore, making changes to the optimization solvers do not affect the C-WEAP RESTful framework. Thus, if there are no changes in the APIs that can have side-effects on the Entities, Project, and Scenario, they do not cause making changes to the RESTful framework. However, changes to the C-WEAP framework is expected as the RESTful framework, and its enabling APIs are expected to evolve in the future.

Due to the use of the WEAP APIs listed in Appendix A, it is necessary to have a WEAP system license to use the C-WEAP framework. The executable version of the C-WEAP framework and a User-Guide are available [47]. The main packages which have been used to develop the C-WEAP framework are *TS-Node 8.10.2* (Typescript-Node) to use Typescript in the NodeJS server-side application; *Express 4.17.1* to build a web application and APIs; *Routing-Controller 0.8.1* to create structured, declarative and beautifully organized class-based controllers; *Body-Parser 1.19.0* to parse the body of the incoming request to web-server, and *Winax 1.20.0* to define *ActiveXObject* in NodeJS (create WEAP instance in server-side application), and some additional packages like *class-transformer, class-validator*, and *reflect-metadata*.

5.4. Applicability

The described design and implementation of the developed framework is applicable to those that have characteristics similar to the WEAP system. For example, the C-WEAP framework can be reused in a straightforward fashion for the LEAP system. The LEAP system, also developed by SEI, is a software tool for energy policy analysis and climate change mitigation assessment. It is an integrated scenario-based modeling tool that can be used to track energy consumption, production, and resource extraction in all sectors of an economy. The LEAP system shares the WEAP approach to model development, execution, and evaluation. The architecture of the C-WEAP framework can be reused for the LEAP system.

The RESTful APIs, Service, and Data Access parts of the Componentized-LEAP framework are the same as those developed for the C-WEAP framework. A Data Transfer Object design is developed according to the LEAP model structure (i.e., entities and their relationships). The execution time step is defined as an integer value and is subject to a different constraint as compared to the WEAP system.

6. Conclusions

The WEAP system is appealing to domain experts from the standpoint of ease of use for rapid model development. This paper provides detail for defining the WEAP components as proxies for WEAP entities using meta-modeling and Model Driven Architecture. The WEAP entities, input and output variables, and their data are represented using the Ecore meta-modeling approach, where each proxy model component corresponds to a WEAP entity. The Ecore presents a well-defined componentized specification for the WEAP legacy modeling and simulation system. These components are used in a flexible service-oriented framework. The outcome is the Componentized-WEAP (C-WEAP) RESTful framework. The C-WEAP framework helps to consider a set of component models instead of thinking about a group of shared variables (belong to different entities) that are used in mass-balanced equations. Also, the REST APIs ease the use of the WEAP system in modern computing platforms, including its integration with other tools to model and simulate more complex systems, like the Food-Energy systems. Every model entity developed in the WEAP system is automatically extracted and included as a componentized model in the C-WEAP RESTful framework.

The componentization of the WEAP system supports a higher degree of control for manipulating and simulating the water entity models. For example, the C-WEAP framework can help simplify the design of simulation experiments and optimization studies that can be difficult using the scripting languages supported in the WEAP system. The RESTful framework with the WEAP componentization can lend itself to better support the development of customized visualization tools. The realization of the C-WEAP RESTful framework can be adopted for similar kinds of systems (e.g., LEAP system) and simplify integration with other web-services.

Acknowledgment

This research is funded by the National Science Foundation under Grant #CNS-1639227, “INFEWS/T2: Flexible Model Compositions and Visual Representations for Planning and Policy Decisions at the Sub-regional level of food-energy-water nexus”. We are also grateful to four anonymous referees for their critiques and constructive suggestions.

Appendix A

Table A1

Table A1
The WEAP's APIs used in the Componentized-WEAP framework.

#	API	Return Object	Category
1	WEAP.ActiveArea	Area	WEAP
2	WEAP.ActiveArea.Name	String	
3	WEAP.WaterYearStart	Integer	
4	WEAP.ActiveScenario	Scenario	
5	WEAP.BaseYear	Integer	
6	WEAP.EndYear	Integer	
7	WEAP.TimeStepName(Id)	String	
8	WEAP.NumTimeSteps	Integer	
9	WEAP.View	String	
10	WEAP.Calculate(LastYear, LastTimestep, AlwaysCalculate)	–	
11	WEAP.ResultValue(BranchName:VariableName, Year, TimeStep,ScenarioName)	Double	
12	WEAP.Areas(Id)	Area[]	Area
13	WEAP.Areas.Count	Integer	
14	WEAP.Versions.Count	Integer	Version
15	WEAP.Versions(Name/Id)	Version	
16	WEAP.Versions.Exist(VersionName)	Boolean	
17	WEAP.SaveVersion(VersionName)	–	
18	WEAP.Versions(VersionName).Revert()	–	
19	WEAP.Scenarios(Id)	Scenario[]	Scenario
20	WEAP.Scenarios.Exists(ScenarioName)	Boolean	
21	WEAP.Scenarios.Add(ScenarioName)	–	
22	WEAP.Scenarios(ScenarioName).Delete()	–	
23	WEAP.Branch(BranchName)	Branch	Branch
24	WEAP.BranchExists(BranchName)	Boolean	
25	WEAP.Branch(BranchName).Children	Branch[]	
26	WEAP.Branch(BranchName).Variables	Variable[]	
27	WEAP.Branch(BranchName).Variables.Exists(VariableName)	Boolean	

References

[1] H. Hoff, "Understanding the nexus: Background paper for the Bonn2011 Conference," 2011.
[2] J. Sieber, C. Swartz, A.H. Huber-Lee, Water Evaluation and Planning System (WEAP): User Guide, Stockholm Environment Institute, Boston, 2005.
[3] ACIMS, "DEVS-Suite simulator," 2020. URL: https://acims.asu.edu/software/devs-suite/.
[4] B.P. Zeigler, H.S. Sarjoughian, Guide to modeling and simulation of systems of systems, Springer Sci. (2017).
[5] SEI, "Long-range energy alternatives planning (LEAP) system," 2020. URL: https://www.energycommunity.org.
[6] M.D. Fard, H.S. Sarjoughian, A web-service framework for the water evaluation and planning system, Spring Simul. Conf. (Spring Sim) (2019) 1–12.
[7] D. Yates, J. Sieber, D. Purkey, A. Huber-Lee, WEAP21—a demand-, priority-, and preference-driven water planning model: part 1: model characteristics, Water Int. 30 4 (2005) 487–500.
[8] H. Lévite, H. Sally, J. Cour, Testing water demand management scenarios in a water-stressed basin in South Africa: application of the WEAP model, Phys. Chem. Earth, Parts A/B/C 28 (2003) 779–786.
[9] J. Gao, P. Christensen, W. Li, Application of the WEAP model in strategic environmental assessment: experiences from a case study in an arid/semi-arid area in China, J. Environ. Manag. 198 (2017) 363–371.
[10] A. Psomas, Y. Panagopoulos, D. Konsta, M. Mimikou, Designing water efficiency measures in a catchment in Greece using WEAP and SWAT models, Proc. Eng. 162 (2016) 269–276.
[11] B. Höllermann, S. Giertz, B. Diekkrüger, Benin 2025—balancing future water availability and demand using the WEAP 'water evaluation and planning'system, Water Resour. Manag. 24 (2010) 3591–3613.
[12] A. Amin, J. Iqbal, A.A. Asghar, L. R, Analysis of current and future water demands in the upper Indus basin under IPCC climate and socio-economic scenarios using a hydro-economic WEAP model, Water 10 5 (2018) 537.
[13] SEI, "WEAP: water evaluation and planning system," 2020. URL: http://www.weap21.org/WebHelp/index.html.
[14] B.P. Zeigler, H. Praehofer, T.G. Kim, Theory of Modeling and Simulation, Academic press, 2000.
[15] Gurobi, "Gurobi optimization, " 2019. URL: https://www.gurobi.com/.
[16] D. Fensel, H. Lausen, A. Polleres, J.D. Bruijn, M. Stollberg, D. Roman, J. Domingue, Enabling semantic web services: the web service modeling ontology, Springer Sci. Bus. Media (2006).
[17] Y. Kun, W.A.N.G. Xiao-Ling, Z.H.O.U. Ao-Ying, Underlying techniques for web services: a survey, J. Softw. (2004).
[18] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S. Thatte, D. Winer, Simple object access protocol (SOAP) 1.1, World Wide Web Consort. (W3C) (2001).
[19] R.T. Fielding, R.N. Taylor, Architectural styles and the design of network-based software architectures, Doctoral Dissertation, University of California, Irvine, 2000.
[20] L. Richardson, S. Ruby, RESTful Web Services, O'Reilly Media, 2008.
[21] D. Brutzman, M. Zyda, J.M. Pullen, K.L. Morse, Extensible modeling and simulation framework (XMSF): challenges for web-based modeling and simulation, Findings and Recommendations Report of the XMSF Technical Challenges Workshop and Strategic Opportunities Symposium, 2002.
[22] M. Sonntag, S. Hotta, D. Karastoyanova, D. Molnar, S. Schmauder, Using services and service compositions to enable the distributed execution of legacy simulation applications, In European Conference on a Service-Based Internet, Berlin, Heidelberg, Springer, 2011.

[23] J. Bisbal, D. Lawless, B. Wu, J. Grimson, Legacy information systems: issues and directions, IEEE Softw. (1999) 103–111.
[24] H. Sneed, Integrating legacy software into a service oriented architecture, Software Maintenance and Reengineering, 2006.
[25] H. Sneed, Wrapping legacy software for reuse in a SOA, Multikonferenz Wirtschaftsinformatik (2006) 345–360.
[26] J.M. Pullen, R. Brunton, D. Brutzman, D. Drake, M. Hieb, K.L. Morse, A. Tolk, Using web services to integrate heterogeneous simulations in a grid environment, Fut. Gener. Comput. Syst. 21 1 (2005) 97–106.
[27] MPI, "MPI forum, " 2019. URL: https://www.mpi-forum.org/.
[28] C. Cai Zhang, F. Qin, X. Wang Zhang, J. Zhu, Y. Xin Zhang, H. Wang, An object-oriented framework for modeling watershed flow and sediment process based on fine-grained components, Arab. J. Geosci. 12 (20) (2019) 620.
[29] W. Li, S. Wu, M. Song, X. Zhou, A scalable cyberinfrastructure solution to support big data management and multivariate visualization of time-series sensor observation data, Earth Sci. Inf. 9 4 (2016) 449–464.
[30] S.L. Markstrom, R.S. Regan, L.E. Hay, R.J. Viger, R.M. Webb, R.A. Payn, J.H. LaFontaine, PRMS-IV, the precipitation-runoff modeling system, version 4, US Geol. Surv. Tech. Methods (2015) 6–B7.
[31] S.L. Markstrom, R.G. Niswonger, R.S. Regan, D.E. Prudic, P.M. Barlow, GSFLOW-coupled ground-water and surface-water FLOW model based on the integration of the precipitation-runoff modeling system (PRMS) and the modular ground-water flow model (MODFLOW-2005), US Geol. Surv. Tech. Methods 6 (2008) 240.
[32] B.T. Daher, R.H. Mohtar, Water–energy–food (WEF) Nexus tool 2.0: guiding integrative resource planning and decision-making, Water Int. 40 5-6 (2015) 748–771 Vols.
[33] H.S. Sarjoughian, Model composability, In Proceedings of the 2006 Winter Simulation Conference, 2006, pp. 149–158.
[34] M. Akbari, Models for management of water conflicts: a case study of the San-Joaquin watershed, California, Colorado State University, Colorado, US, 2012.
[35] V.V. Nikolic, S.P. Simonovic, Multi-method modeling framework for support of integrated water resources management, Environ. Process. 2 3 (2015) 461–483.
[36] M.S.Q. Alvi, I. Mahmood, F. Javed, A.W. Malik, H.S. Sarjoughian, Dynamic behavioural modeling, simulation and analysis of household water consumption in an urban area: a hybrid approach, 2018 Winter Simulation Conference (WSC), 2018, pp. 2411–2422.
[37] J. Ni, M. Liu, L. Ren, S.X. Yang, A multiagent Q-learning-based optimal allocation approach for urban water resource management system, IEEE Trans. Autom. Sci. Eng. 11 1 (2013) 204–214.
[38] P. Darbandsari, R. Kerachian, S. Malakpour-Estalaki, An Agent-based behavioral simulation model for residential water demand management: the case-study of Tehran, Iran, Simul. Modell. Pract. Theo. 78 (2017) 51–72.
[39] M.D. Fard, H.S. Sarjoughian, Coupling WEAP and LEAP models using interaction modeling, SpringSim Conference, Fairfax, VA, USA, 2020.
[40] SEI, "Water Evaluation and Planning (WEAP) system", 2020. URL: http://www.weap21.org/.
[41] J. Tihomirovs, J. Grabis, Comparison of soap and rest based web services using software evaluation metrics, Inf. Technol. Manag. Sci. 19 1 (2016) 92–97.
[42] F. Budinsky, D. Steinberg, R. Ellersick, T.J. Grose, E. Merks, Eclipse modeling framework: a developer's guide, Addison-Wesley Prof. (2004).
[43] D. Steinberg, B. Frank, E. Merks, M. Paternostro, EMF: eclipse modeling framework, Pearson Edu. (2008).
[44] SEI, "Automating WEAP (API)," 2020. URL: http://www.weap21.org/WebHelp/API.htm.
[45] M. Cantelon, M. Harter, T.J. Holowaychuk, N. Rajlich, Node.js in Action, Greenwich, Manning, 2014.
[46] L. Wittgenstein, The Big Typescript: TS 213, John Wiley & Sons, 2012.
[47] ACIMS, "User-guide for componentized-WEAP RESTful framework," 2020. URL: https://acims.asu.edu/software/C-WEAP.

Mostafa D. Fard received the B.S. degree in software engineering from Shamsipour Technical and Vocational College, Tehran, Iran in 2010 and M.S. degrees in software engineering from University of Tehran, Tehran, Iran in 2014. He is currently pursuing his Ph.D. degree in computer science at Arizona State University, Tempe, AZ, USA.

Hessam S. Sarjoughian is currently an Associate Professor of computer science and computer engineering with the School of Computing, Informatics, and Decision Systems Engineering (CIDSE), Arizona State University (ASU), Tempe, AZ, USA, and the Co-Director of the Ari-zona Center for Integrative Modeling and Simulation (ACIMS). His-research interests include model theory, poly-formalism modeling, collaborative modeling, simulation for complexity science, and M&S frameworks/tools. He is the Director of the ASU Online Master of Engineering in Modeling and Simulation Program.