Proceedings of the 2021 Winter Simulation Conference
S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo and M. Loper, eds.

A RESTFUL PERSISTENT DEVS-BASED INTERACTION MODEL FOR THE COMPONENTIZED WEAP AND LEAP RESTFUL FRAMEWORKS

Mostafa D. Fard
Hessam S. Sarjoughian

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

ABSTRACT

Modeling the interactions between separate models contributes to building flexible hybrid simulation frameworks. In earlier work, an Algorithmic Interaction Model was proposed and developed to integrate the componentized WEAP and LEAP RESTful frameworks for modeling and simulating water and energy systems. However, this approach does not separate modeling and simulation protocol from each other. It also does not support flexible, structured model hierarchies. To overcome these limitations, the parallel DEVS formalism is used to develop an Interaction Model for use with the DEVS-Suite simulator. The resulting DEVS Interaction Model (DEVS-IM) is supported with a RESTful framework and MongoDB for storing the interaction models for the coupled water and energy models. The DEVS-IM, grounded in system theory and component-based modeling, offers strong support for the model reusability, flexibility, and maintainability traits essential for developing realistic simulations of coupled energy and water systems.

1 INTRODUCTION

The Food-Energy-Water Nexus (FEW-Nexus) has emerged as necessary for their better use, production, and management (Hoff 2011). The "nexus" refers to the relationships among food, energy, and water systems. The "incorporation" and "cross-linking" defining the nexus are crucial for understanding and managing the food, energy, and water systems as a whole system. Improving our understanding of these systems as a whole system requires modeling the degree to which each system depends on and affects the others. For example, water needs to irrigate land for food production and coal power plants for energy generation. Energy requires for agricultural practices and to transport and treat water. Agricultural activities can generate energy through biofuels but also affect water quality. Knowledge of the linkages, synergies, and conflicts is needed to develop policies for the sustainability of collective water, energy, and food systems.

Modeling the FEW-Nexus is a challenging task that requires extensive data on specific study areas (Zhang et al. 2019; Fard et al. 2020). Domain experts can participate in collective work using existing tools and better use of previously acquired knowledge and experience. The Water Evaluation and Planning (WEAP 2021) and Low Emissions Analysis Platform (LEAP 2021) tools are used to model, simulate, and evaluate individual and combined water and energy systems. The advantage of using such tools and frameworks reduces the effort and resources needed for simulation studies.

The composition of models is considered essential in developing complex systems and simulation models capable of expressing a system's structure and behavior (Sarjoughian 2006). Systems are integrated from homogenous or heterogeneous sub-systems. Each sub-system can be considered as an independent system or component. Considering the water, energy, and food systems as separate models/components and

coupling them via an interaction model leads to modularity and valuable flexibility for modeling, simulating, and evaluating the FEW and its nexus (Fard and Sarjoughian 2020b; Fard et al. 2020).

Addressing the need to integrate tools for understanding and assessing the FEW-Nexus, this article is presenting a new modeling and simulation design based on the DEVS formalism for coupling models developed in the WEAP and LEAP systems. Even though these tools are internally linked, defining interactions between water and energy models is limited in terms of flexibly defining choices of data to be communicated, time resolution, and control with support for distributed simulation. The contributions of this paper are (i) a DEVS-based Interaction Model that conforms to the Knowledge Interchange Broker modeling approach, (ii) a persistent model base for supporting a family of DEVS-IM models in MongoDB, (iii) the DEVS-IM MongoDB aimed at automated partial code generation for the DEVS-Suite simulator, (iv) a RESTful architecture that supports (i) and (ii).

2 BACKGROUND

In this section, details pertaining to the scope of this paper are briefly presented. The emphasis is on model composability. The interaction model specification and the database model repository presented in this paper are intended to be agnostics to simulation platforms. There are a variety of approaches, frameworks, and tools for multiple simulators to interoperate (Wang et al. 2011). In this work, the DEVS Interaction Model is developed as a RESTful framework. This supports the DEVS-IM and the Componentized WEAP and LEAP to execute as separate RESTful simulators.

2.1 Componentized-WEAP & Componentized-LEAP Frameworks

The Componentized-WEAP (C-WEAP) and Componentized-LEAP (C-LEAP) RESTful frameworks are componentized proxies of the WEAP and LEAP systems (Fard and Sarjoughian 2020a). Each system becomes a component-based tool supported within a RESTful framework. These frameworks have model components for all entities, variables, and data defined in the WEAP and LEAP systems. The returned data of different APIs is in JSON format (correspond to the defined domain models for the components and variables). The C-WEAP and C-LEAP frameworks present the same schema for a project, its scenarios, components, and the input/output variables belong to each component. The executions of the water and energy model are supported by the WEAP and LEAP systems.

2.2 The Algorithmic Interaction Model

As a first design for coupling the C-WEAP and C-LEAP models, an Algorithmic Interaction Model (Algorithmic-IM) is developed based on the Knowledge Interchange Broker (KIB) approach (Fard and Sarjoughian 2020b). The interactions between the water and energy models are defined as separate models. The input and output relationships between the composed models are defined as a set of hierarchically structured components (i.e., interaction model). The components, as modules and transformations, prescribe time-based data mappings subject to a controlled execution regime. Thus, the water and energy models do not have direct knowledge of each other. However, this interaction model does not separate domain-specific model specification from its execution protocol, a fundamental principle of the Knowledge Interchange Broker modeling approach for model composability.

Figure 1 presents a portion of the class diagram for the Algorithmic-IM. The gray and yellow classes are abstract and concrete classes, respectively. The interaction model has a set of *Modules*, each composed of one or more *Transformations*. Two different kinds of input and output ports are defined, module's ports and transformation's ports. The module's ports support receiving/sending data from/to water and energy models via function calls. These ports allow the modules to communicate to specific entities and variables in the C-WEAP and C-LEAP models (defined in the WEAP and LEAP systems). Three types of coupling are supported between the module and/or transformation ports; modules' inputs to transformations' inputs, transformations' outputs to transformations' inputs, and transformations' outputs to modules' outputs. The data communicates unidirectionally via defined couplings. Each transformation can process data values on

its input ports and send data values on its output ports. The structure of the received or sent data in each module and transformation must be defined, and it must implement the `IMessage` interface.

The Algorithmic-IM has a cyclic, discrete time-step, and synchronous execution protocol for concurrent and bidirectional data mappings between the water and energy models. The Algorithmic-IM has a round-based execution regime consisting of six steps. 1) executing the connected systems to the module's input ports, 2) invoking data from the systems via module input ports, 3) sending the received data to the transformation's input ports, 4) executing the transformations, 5) sending the processed data to the module's output ports/other transformation's input ports; finally, 6) sending the output data (collected in the module's output ports) to the external systems. Consequently, connected systems (i.e., WEAP and LEAP) can execute independently and simultaneously by executing the interaction model (Fard and Sarjoughian 2020b). The Algorithmic-IM communicates with the C-WEAP and C-LEAP frameworks using their designated RESTful APIs. Using the signature of the C-WEAP and C-LEAP framework's APIs, every module's port constructs the relevant URL of the componentized water and/or energy models. The composition of the water and energy models via the Algorithmic-IM is well-formed in structure and behavior. The execution performance of the Algorithmic-IM was tested for a real example (the Phoenix Active Management Area) with an acceptable result (around 4 percent overhead) in comparison to the internal linkage between the WEAP and LEAP systems. Nevertheless, the Algorithmic-IM design has some limitations and constraints, which are tried to be eliminated in a new design based on the parallel Discrete Event System Specification (DEVS) formalism.

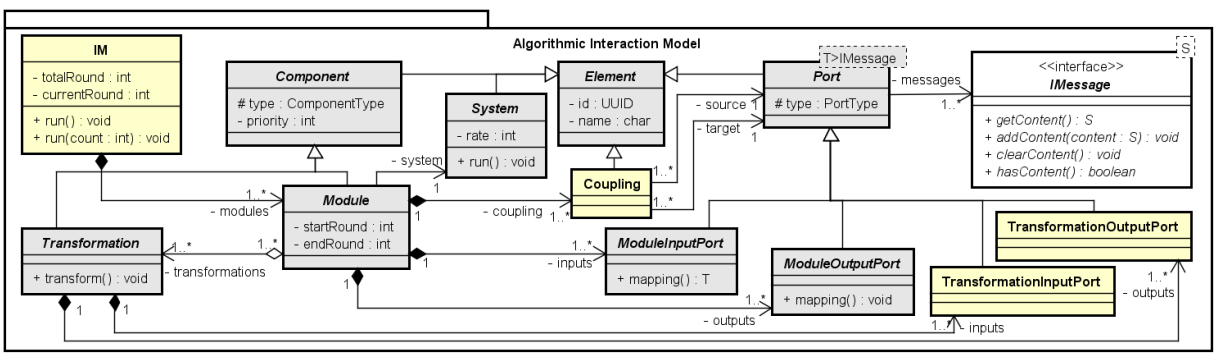


Figure 1: Modeling specification class diagram for the Algorithmic Interaction Model.

3 RELATED WORK

The research on the FEW-Nexus has seen a significant increase in both the number of studies and the scientific community's ability to assess water, energy, and food linkages at multiple resolutions. Thirty-five methods, tools, and frameworks are reviewed in detail related to the Water-Energy Nexus based on the geographical scale and the nexus scope (Dai et al. 2018). Some frameworks, such as CLEWS (Howells et al. 2013) and NexSym (Martinez-Hernandez et al. 2017), are modular. Some other frameworks and tools, such as the WEF Nexus Tool 2.0 (Daher and Mohtar 2015) and WEAP-LEAP (Sieber et al. 2005), are not founded on component-based modeling principles.

Modeling the interactions between heterogeneous model types has been specified using the DEVS formalism (Mayer and Sarjoughian 2016). In this case, parallel DEVS is used to specify and implement the KIB as an Interaction Model for hybrid Composable Cellular Automata and Geographical Resources Analysis Support System (GRASS) models. The Geo-KIB is developed as an extension of the KIB to support spatiotemporal data mappings (Boyd and Sarjoughian 2020). It is designed and implemented using modular object-oriented design principles and techniques. The Geo-KIB has a set of spatiotemporal data mapping functions to regulate the interaction between two I/O modular models. One simulation is designated as input and the other as output for the Geo-KIB. Each mapping can have some inputs (discrete-event and discrete-time inputs) and some outputs (active and passive outputs) ports to be connected to the simulations. This

type of interaction modeling is used to compose distinct spatiotemporal agent-based models where human food consumption and food growth bidirectionally interact with one another. It is also used for composing discrete spatiotemporal agent-based and GRASS models.

The Service-Oriented Architectures (SOA) has been widely used as the communication mechanism to achieve interoperability and composability for simulations in a transparent, open, and scalable way (Tsai et al. 2006). Among these is the web-based DEVS distributed framework was defined in DEVS/SOA architecture (Mittal et al. 2009). In another work, the RESTful Interoperability Simulation Environment (RISE) supports interoperating heterogeneous simulation models and tools regardless of their underlying technology or algorithms (Al-Zoubi and Wainer 2010).

Today's most common database implementation is based on the relational model, which uses SQL as its query language. For example, the CoSMoS modeling and simulation tool stores and generates partial code for families of parallel DEVS models as relational databases (Sarjoughian and Elamvazhuthi 2009). Also, simulation software such as AnyLogic uses relational databases to read input data and write simulation output. Not Only SQL (NoSQL) databases are receiving popularity for their capability in dealing with a large amount of complex data in various structures (Parker et al. 2013). MongoDB (MongoDB Documentation 2021), a NoSQL database, has dynamic schema and stores data as BSON documents (binary encoded JSON-like objects). This database is used as the model base for the DEVS interaction models.

4 DEVS-BASED PERSISTENCE INTERACTION MODEL

It is advantageous to use a formal modeling method instead of an algorithm (presented in section 2.2) to model and simulate the interactions between the nexus of the water-energy system. A component-based, hierarchical modeling approach that aligns with system thinking helps with the development, reuse, and maintainability of interaction models. The parallel DEVS formalism (Chow and Zeigler 1994) is selected for designing the Interaction Model due to its strong modularity, hierarchy, and support for discrete-time state transitions with inputs and outputs used in C-WEAP and C-LEAP frameworks. Furthermore, it is important to use established modeling and simulation engines. The parallel DEVS models can be developed, simulated, tested, and debugged using the DEVS-Suite simulator (ACIMS 2021; McLaughlin and Sarjoughian 2020). Together, the DEVS formalism and the DEVS-Suite simulator provide a solid advancement to the interaction model's algorithmic approach and implementation. Additionally, this simulator's component-based Model-Façade-View-Control design naturally aligns with RESTful architecture. The RESTful architecture of the DEVS-Suite simulator is key for integration with the RESTful C-WEAP and C-LEAP tools.

4.1 Internal Structure Specification

The specification of the interaction model using the parallel DEVS formalism leads to rigorous, systematic model development. Figure 2 illustrates a portion of the class diagram for a new Interaction Model design based on the parallel DEVS formalism (called DEVS-IM). This interaction model satisfies two needs. One is to model the interactions among the water and entity models. Another is for the highest level DEVS coupled model to communicate with other simulators. The `modeling` package, in Figure 2, highlights the DEVS's core modeling engine. Two main classes in the `component` package are `IMAtomicModel` and `IMCoupledModel`. The `componentType` attribute in these two classes can be IM, INPUT_CONNECTOR, OUTPUT_CONNECTOR, PROCESS, TASK, and LOGIC. Each atomic or coupled inherited class sets the `componentType` value in its constructor. The required functionalities for the components are defined in their corresponding interfaces. An `OutputConnector` class may select an `InputConnector` class (the `input` association relation between the classes), depends on its functionality. The `Project` class, in the `im` package, is inherited from the `Entity` class and can have multiple interaction models (see the composition relation between the `Project` and `IM` classes). The `Project`, `IM`, `Process`, and `Task` are concrete classes, and the `Logic`, `InputConnector`, and `OutputConnector` abstract classes must be specialized.

The composition of the WEAP and LEAP models can have one or more interaction model components for a given project. Because the `IM` (or `Process`) class inherits from the `IMCoupledModel` class, the *IM* (or

Process) component can have many *Process*, *Task*, and *Logic* components as its sub-components. Each *IM* component can have multiple *InputConnector* and *OutputConnector* components. These input and output components, defined using DEVS atomic models, can be coupled with any DEVS model as well as communicating (via function calls) with the C-WEAP and C-LEAP frameworks. The *Process* component cannot have any *InputConnector* or *OutputConnector* component as its sub-components. The DEVS coupled *IM* component does not have any external input and external output ports. The `init()` method in the `IM` class defines the initialization for the interaction model.

The *Task* component in the DEVS-IM design serves the purpose of the *Transformation* component in the Algorithmic-IM. The main difference between these two components is the capability to set the time advance value in the *Task* component. Also, a modeler can define other new components (derived from the `IMAtomicModel` class) which have different behavior compared to the *Task* component. The specification of the *Task* component is defined in Listing 1. The *Active* and *Passive* are defined for the values of the `phase` attribute defined in the `IMAtomicModel` class. The input and output port names and values are defined by the modeler. A queue is used to store the input values (messages) received on the input ports when the model is in *Active* phase. The value of *ActiveTime* is read and changed using the `getActiveTime()` and `setActiveTime(…)` methods in the `Task` class (see Figure 2). The `activate()` method changes the initial phase to *Active*. The output function corresponds to the `perform()` method. The queue size is checked in the internal transition function to define the next state. The specifications for the remaining functions are straightforward. Some useful predefined components (e.g., Queue, Stack, Random Generator, Periodic Generator, etc.) can be inherited from the `Task` class to have simpler and faster modeling.

In addition, the *Logic* component is an atomic model with a specific behavior that does not increase the simulation clock. It can be used to define some logical operations. For example, the *Junction* component to send data on the input/s to all output ports, the *Choice* component to send the input data to one of the outputs based on some condition, and the *Synchronization* component to sync the inputs and send them on outputs. The formal specification of these components is not explained due to the space limitation of the paper. No explicit concept of the logic component was used in the Algorithmic-IM. Like the *Task* component, the *Logic* components can be defined by the user.

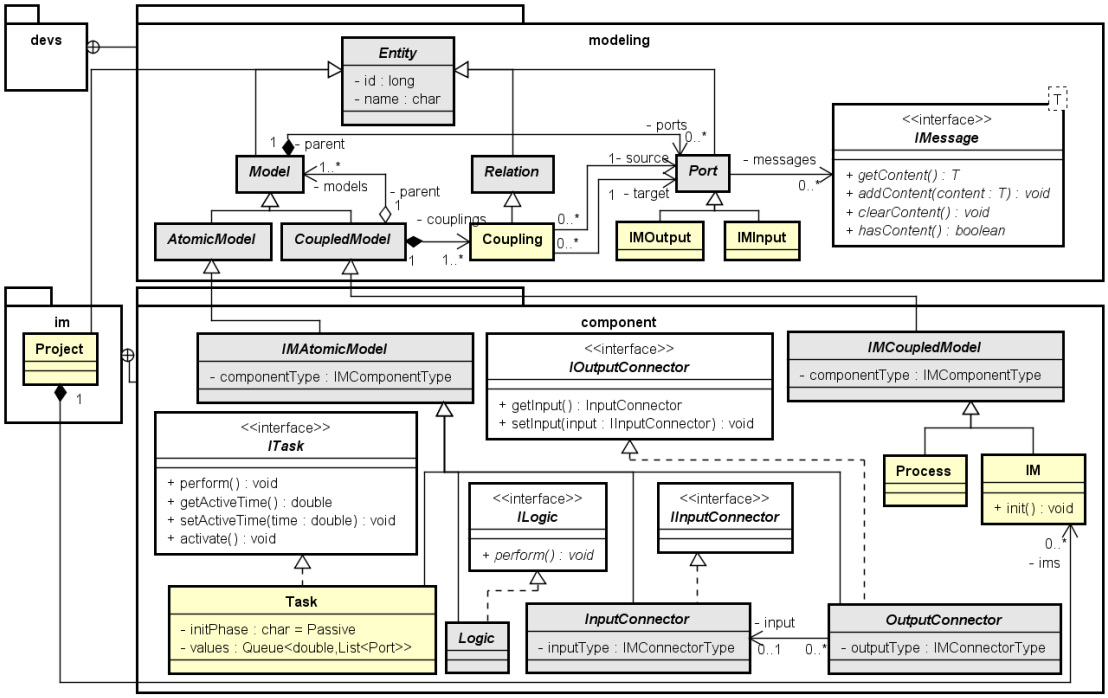


Figure 2: The class diagram for the DEVS-Based Interaction Model.

4.2 External Structure Specification

Component-based modeling is an approach for simulating water and energy resource systems. A model comprises a set of components, each with some defined modeling objectives. The system package in Figure 3 includes three concrete classes to represent the external systems (i.e., the WEAP and LEAP tools) in a hierarchical component-based manner. Multiple Systems can be defined in each project (see the composite relation from the Project class to the System class in Figure 3). All classes have the id (as the key attribute) and unique name attributes. The ISystem and IFunction interfaces defined the method signatures to be implemented in the System and Function concrete classes. The init() and run() methods in the ISystem are used for initialization (running at the beginning of the interaction model simulation execution) and execute the external simulation system, respectively. A hierarchy of components can be defined in a system, and each component can have many functions. A Function can have one parameter of type Object. In the case of having multiple input parameters for a function, they must be wrapped in an object/class. The desired objective of the Function must be implemented in the exec(…) method. The getResult() method returns an IMessage as the result of the execution. In the WEAP and LEAP systems, the functions of each component must call the RESTful APIs defined by the C-WEAP and C-LEAP frameworks. In the Algorithmic-IM, all these steps are generalized and handled inside the mapping() methods of the module's ports (see Figure 1). The Algorithmic-IM did not have any consideration for the details of the external system's models. For the DEVS-IM, an interface of the external system must be defined in the interaction model.

Listing 1: Parallel DEVS Task Specification

$$M_{Task} =< X^b, Y^b, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta >$$

$$IPorts = \{\ldots\}, OPorts = \{\ldots\}$$

$$\mathrm{X}^b = \{(p,v) \mid p \in IPorts,\ v \subset values\}, \mathrm{Y}^b = \{(p,v) \mid p \in OPorts,\ v \subset values\}$$

$$S = phase \times \sigma \times Queue, Phase = \{\text{“}Passive\text{”}, \text{“}Active\text{”}\}, \sigma = \Re_0^{+\infty}, Queue =< \Re_0^{+\infty}, (p,v) >$$

$$s_0 = \begin{cases} (\text{"}Passive\text{"}, Infinity, \emptyset) & initPhase == \text{"}Passive\text{"} \\ (\text{"}Active\text{"}, ActiveTime, \emptyset) & otherwise \end{cases}$$

$$\delta_{ext}(s,e,x^b) = \begin{cases} (\text{“}Active\text{”},\ ActiveTime, Queue.add(\sigma + e, (p,v)) & phase == \text{"}Passive\text{"} \\ (\text{"}Active\ \text{"}, \sigma - e, Queue.add(\sigma + e, (p,v)) & otherwise \end{cases}$$

$$\delta_{int}(s) = \begin{cases} (\text{“}Active\text{”},\ ActiveTime, Quque) & Queue.size > 1 \\ (\text{“}Passive\text{”},\ Infinity, Queue) & otherwise \end{cases}$$

$$\delta_{con}(s, ta(s), x) = \delta_{ext}(\delta_{int}(s), 0, x)$$

$$\lambda(\text{“}Active\text{”}, \sigma, Queue.head) = perform()$$

$$ta(phase, \sigma, Queue) = \Re_0^{+\infty}$$

As mentioned before, the interaction model connectors are atomic models from the DEVS viewpoint. Simultaneously, they are connectors to the external systems from the interaction model standpoint. Some predefined input and output connectors (with specific behavior) are defined in the current DEVS-IM design (the `predefinedComponent` package in Figure 3). The `InputConnector` and `OutputConnector` abstract classes in Figure 3 (the DEVS-IM design) have the same role as the `ModuleInputPort` and `ModuleOutputPort` abstract classes in Figure 1 (the Algorithmic-IM design). In the module's ports of the Algorithmic-IM, the `mapping()` functions defined the port behavior. However, in the DEVS-IM design, the behavior defines using the DEVS functions. From the DEVS specification, all the interaction model connectors have one input port (named *"in"*) and can have one output port (named *"out"*).

The *TransientInput* component allows the interaction model to use discrete-event inputs, which trigger a reaction in the model when an input is received. The `TransientInput` class has a `write(…)` method which can be called by the external system or another interaction model to inject data to the interaction model in a discrete-event manner. The `write(…)` method invokes the external transition function of the atomic model. The incoming data to the connector (on the *"in"* input port of the connector) is transiently sent (via *"out"* output port of the connector) to the internal interaction model's components (which are connected to the input connector).

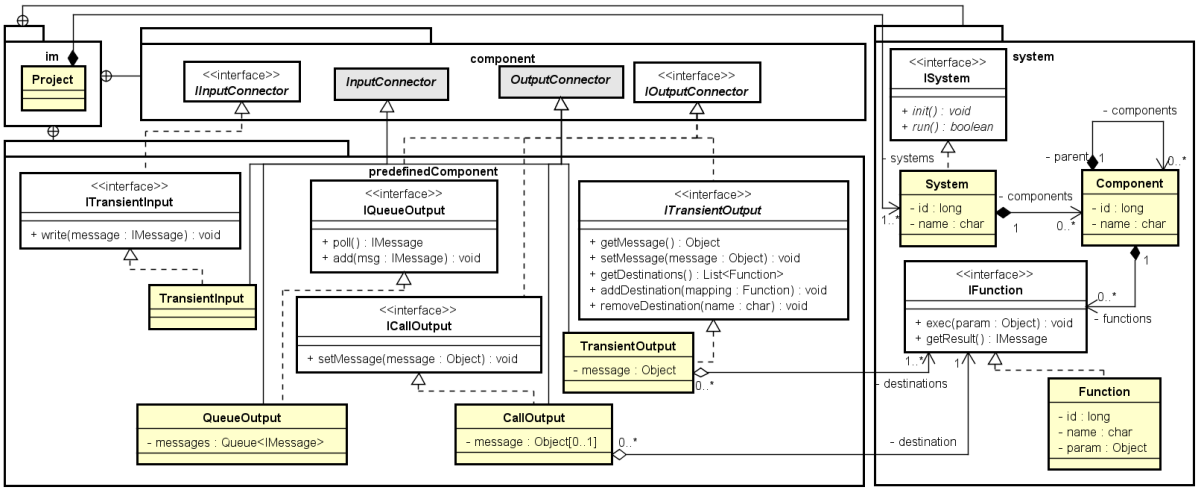


Figure 3: DEVS-IM input and output specification for the external system.

Three output connectors are defined in the DEVS-IM with different behaviors. The *TransientOutput* connector immediately sends data to the external systems. The composite relation from the `TransientOutput` class to the `IFunction` interface defines a list of destinations to send the data (using the `exec(…)` method of the selected function). Data can be sent to multiple destinations. The message to send is set using the `setMessage(…)` method. For example, the data for the input variables of the WEAP or LEAP models can be set using the *TransientOutput* connector (via C-WEAP and C-LEAP APIs). The *CallOutput* connector uses to call a specific function from an external system and return the result to a *TransientInput* connector. The association relation from the `CallOutput` class to the `IFunction` interface defines a destination to invoke. If the invoked function needs a parameter, it can be defined using the `setMessage(…)` method. For example, the result of the output variables of the WEAP or LEAP models can be read using the *CallOutput* connector. The *QueueOutput* component uses to queue some data to be read by the external systems. It has `add(…)` and `poll()` methods to add a message to the end of the queue and read the head message from the queue, respectively. The first method invokes by receiving a message on the input and the second method invokes by an external system. From the DEVS atomic perspective, the *TransientOutput* and *QueueOutput* connector do not have any output ports.

Figure 4 illustrates a manually drawn DEVS-IM component diagram. In the top layer diagram, there are *InputConnector*, *OutputConnector*, *Process*, *Task*, and *Junction* components with couplings among them. The connectors named "out1", "out2", and "out3" of the "IM" interaction model are *TransientOutput*, *QueueOutput*, and *CallOutput*, respectively. Suppose the input connector "in3" of the "IM" component is selected in the output connector "out3" for the `input` attribute (see Figure 2). In the equivalent DEVS model, all components are presented as atomic and coupled models. There are eight atomic models and one coupled model in the "*IM*" component (the "*Process 1*" coupled model has sub-components, as well).

As described before, all components in the DEVS-IM design are derived from the atomic or coupled DEVS models. Thus, the interaction model is a parallel DEVS model (with some specific atomic models as the connectors to communicate with the outside world). Consequently, the DEVS simulation protocol is used to simulate the interaction model. To simulate the DEVS models, a hierarchy of simulator objects that mirrors the hierarchical tree structure of the DEVS model is used. There is a DEVS simulator corresponding to each atomic model and a DEVS coordinator corresponding to each coupled model. A root coordinator oversees controlling the executions of all atomic and coupled simulators. The simulators and coordinators are responsible for the correct simulation of coupled models. A key advantage of using a well-defined simulation protocol is that it allows a simulator to execute models independent of their specific behaviors.

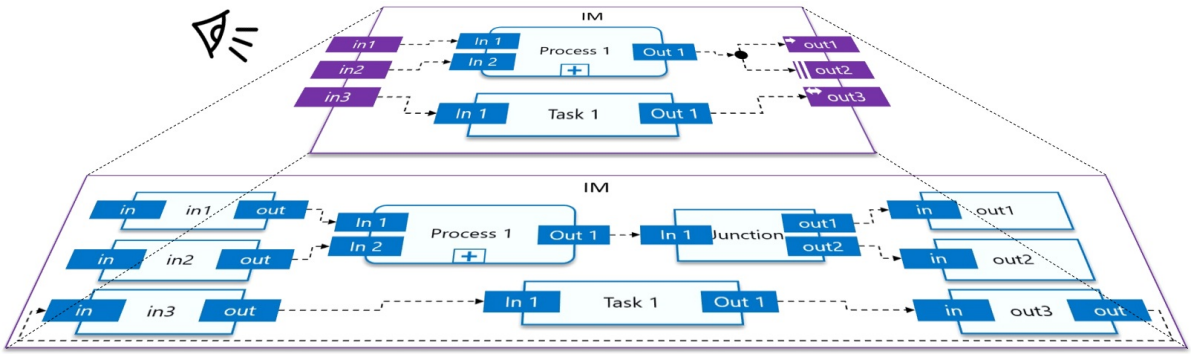


Figure 4: An example Interaction Model viewed as a DEVS model.

4.3 Database Specification

Figure 5 illustrates the database schema in the DEVS-IM design. There are four collections with `id` as the primary key. The *pk*, *fk*, and *dk* show the primary key, foreign key, and destination key in a collection or relation, respectively. Mandatory values are indicated by the star. The `projects` collection stores *Project* components (see Figure 2). All atomic and coupled models (i.e., all hierarchy of *IM*, *Logic*, *Task*, *Process*, and *Connector* components), input and output ports, and couplings defined in an interaction model (see Figure 2) are stored in the `models` collection. The data of the external system interfaces (see Figure 3) are stored in the `systems` and `components` collections. The `ports` and `couplings` in the `models` collection and the `functions` in the `components` collection are defined using one-to-many relationship with embedded documents. The hierarchy structure for the models and components is defined using one-to-one relationship with document references (using `parentId` field). The rest relationships are defined using one-to-many relationship with document references.

The stored data in the database (specifically the `models` collection) will be used to generate the skeleton of a complete project in the DEVS-Suite simulator to define the behavior for the atomic models in the Java programming language. The predefined behavior of the *Logic* and *Connector* components define during the code generation (see section 4.2). Thus, the modeler can define the behavior for the *Task* components and the functions of the components (for the external system interfaces). The test, debug, and visualization features of the DEVS-Suite simulator can also be used to validate the interaction model's correctness.

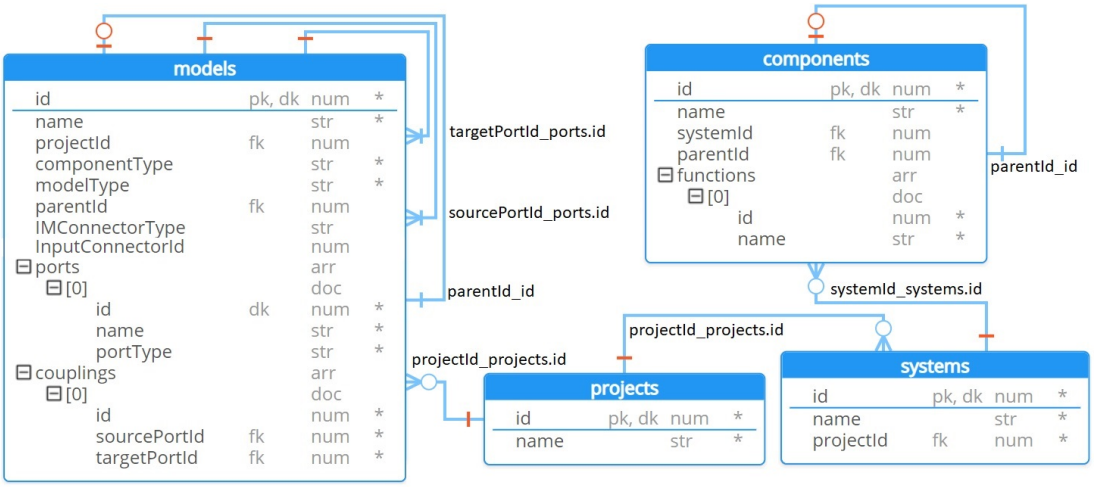


Figure 5: Database schema of the DEVS-Based Interaction Model.

5 INTERACTION MODEL RESTFUL FRAMEWORK

The layered architecture of the DEVS-IM RESTful framework is shown in Figure 6. The backend section is written in the Java Spring Boot framework. The database and the external systems (the C-WEAP and C-LEAP frameworks) are placed at the bottom layer. The Data Access layer is responsible for storing/retrieving data to/from the database, and it is the only layer that has access to the database. The interfaces of the external systems must be defined in the External System Interface part. It manages the required components of the external systems and the communication mechanism (e.g., calling RESTful APIs from the C-WEAP and C-LEAP frameworks). The business logic and information processes are handled in the Service layer. All communications below the Service layer are based on the Domain models (see Figure 2).

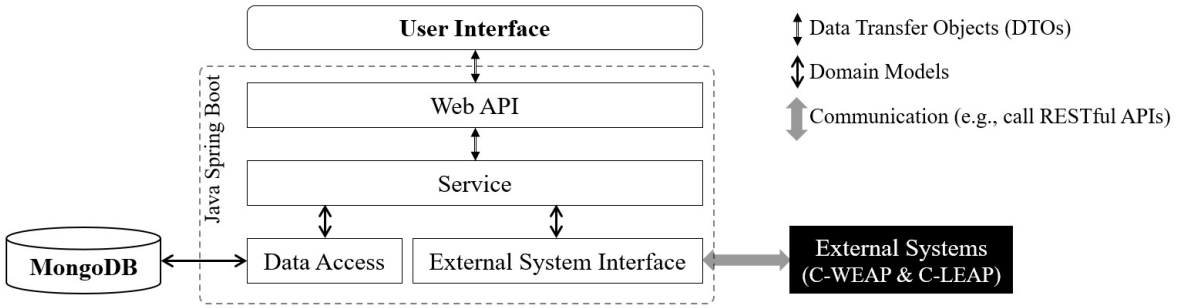
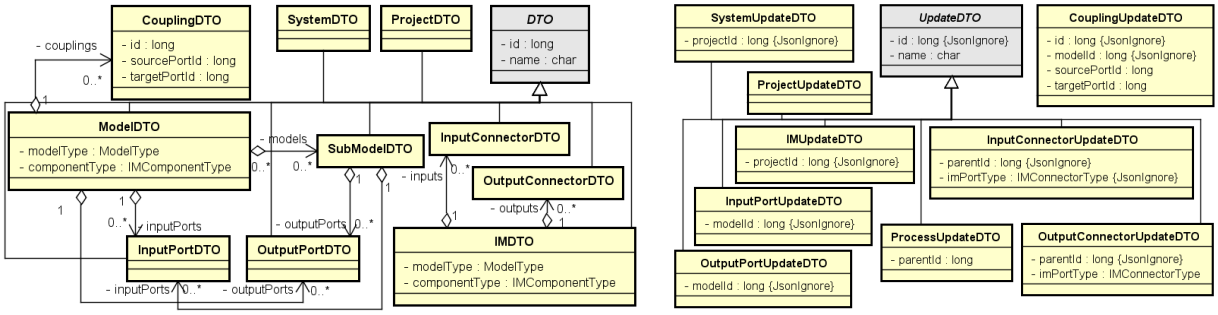


Figure 6: The DEVS-Based Interaction Model layer architecture.

Communications in the Service layer are based on the Data Transfer Objects (DTOs). Figure 7 (a) and (b) present the DTOs to retrieve data (the response of the GET requests) and insert/update data (the body of the POST or PUT requests), respectively. The delete operation handles by setting the id of a component as a URL parameter. The Web API layer contains the webserver and controller parts for handling various client API requests. The data needed for the RESTful framework is in JSON format. The *{JsonIgnore}* constraint for some attributes in Figure 7(b) indicates that these attributes are hidden for the User Interface. These attributes are used to communicate data between Web API and Service layers (see Figure 6).

The standard naming design of the REST architecture is used to define the DEVS-IM RESTful APIs to retrieve, insert, update, and delete different entities in the DEVS-IM RESTful framework using the HTTP GET, POST, PUT and DELETE methods. For example, calling the URL "`/Projects/1/IMs`" with the POST method and the body of the request include the `IMUpdateDTO` object (see Figure 7(b)), inserts a new *IM* component (if the name property is unique in the current project) and returns the inserted value (an `IMDTO` class in Figure 7(a)). Or, calling the URL "`/Models/1`" with the GET method (given the model id is valid and it is the id of an *IM* component) returns an interaction model with all connectors, sub-components with their input and output ports, and the couplings (the `ModelDTO` class in Figure 7(a)).



(a) (b)

Figure 7: A portion of the class diagram for the DTOs in the DEVS-Based Interaction Model. (a) To retrieve data. (b) To insert or update data.

A portion of the class diagram for the Service and the Data Access layers of the DEVS-IM is shown in Figure 8. Each service class in the `Service` package has some association relation with other service classes, and one association relation with its correspond repository in the `Data Access` package (just association from `ModelService` and `IMService` classes to the `ModelRepository` and `IMRepository` classes are shown in the diagram). The interfaces define the required methods which the corresponding classes must implement. Other details of the classes in the Services and Data Access packages are omitted for brevity.

A sequence diagram scenario for a client inserting an IM model is shown in Figure 9. The incoming message 1 by the `ui` object is processed by the `imc` object. The `id` of the project sets in step 2. In step 3, a message is invoked on the `ims` object to insert a new interaction model. The `ims` object checks some validation in step 4 (i.e., the name attribute cannot ne null or empty, having a valid id for the project, and prevent duplicate names for the interaction model). Then, the DTO object maps to a Domain object, and a new valid sequential unique id is set. The `ims` object invokes message 7 on the `imr` object for inserting the interaction model. The `imr` object invokes message 8 on the `db` object (i.e., MongoDB database) and returns the inserted data to the database. Finally, the `im` object (the Domain object) maps to the IMDTO object and returns as the result in step 9. The result returns to the `imc` and `ui` objects, consequently.

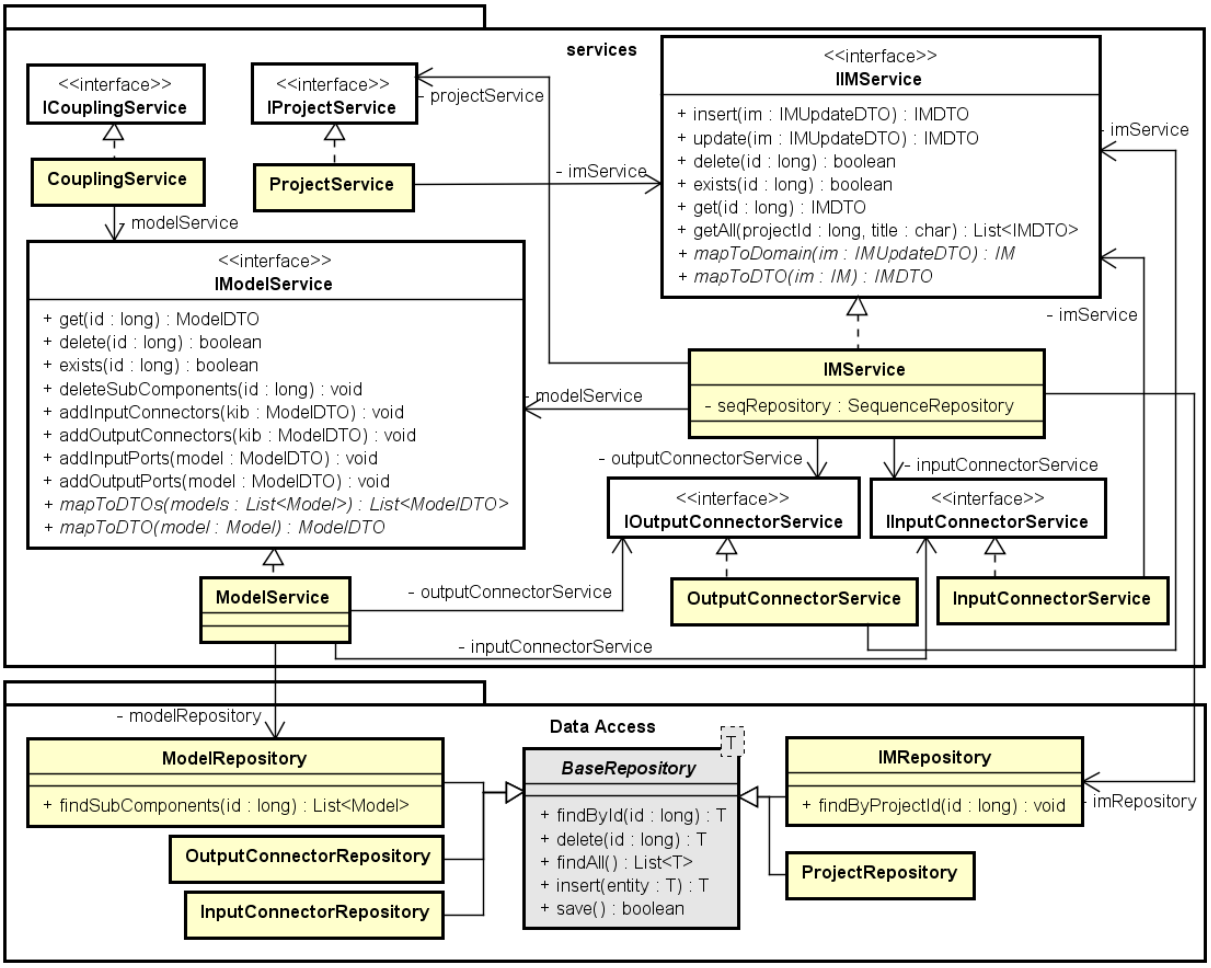


Figure 8: A partial class diagram for the Service and Data Access layers.

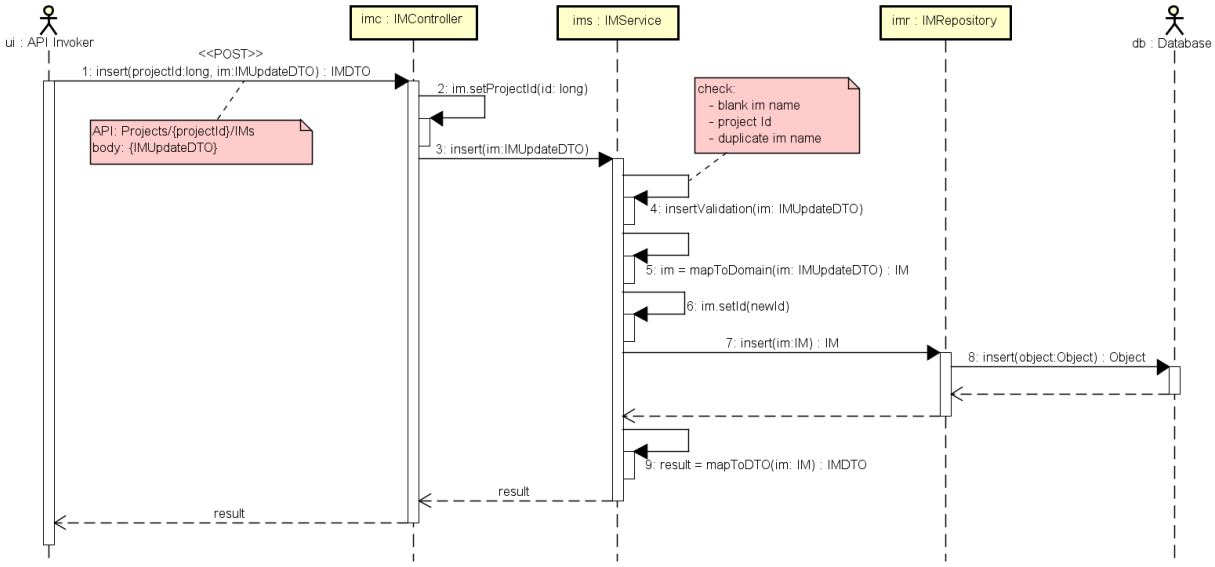


Figure 9: A sequence diagram to insert an IM model via the DEVS-Based Interaction Model.

6 CONCLUSION

The DEVS-Based Interaction Model (DEVS-IM) design is presented in this research for composing C-WEAP and C-LEAP systems. The interaction model supports the flexible and rigorous specification, implementation, and execution of the water and energy systems nexus using the DEVS-Suite simulator. The DEVS-IM, unlike the Algorithmic-IM, separates model specification and its execution. This is crucial for simplifying model development, a key challenge for simulating complex food-energy-water dynamical systems. Higher degrees of model reusability, flexibility, and maintainability can be achieved in comparison to the Algorithmic-IM, and more generally, other existing approaches and frameworks in use for developing Water-Energy simulations. The DEVS-IM, unlike Algorithmic-IM, is supported with MongoDB and used in the RESTful framework. Consequently, the correctness of model specification and execution in the new interaction model design is grounded on the parallel DEVS modeling. The DEVS-Suite simulator's support, including verification and black-box unit testing, is important in developing and simulating models. Since the C-WEAP, C-LEAP, and DEVS-IM are RESTful frameworks, it is simpler to integrate them. Future work includes evaluating the computational efficiency of the DEVS-IM and developing a visual modeling environment for creating and simulating hybrid water, energy, and interaction models.

ACKNOWLEDGMENT

This research is supported under the NSF Grant #CNS-1639227, "INFEWS/T2: Flexible Model Compositions and Visual Representations for Planning and Policy Decisions at the sub-regional level of the Food-Energy-Water nexus".

REFERENCES

ACIMS 2021. DEVS-Suite Simulator. https://acims.asu.edu/software/devs-suite, accessed 10th April 2021.

Al-Zoubi, K., and K. Wainer. 2010. "RISE: REST-ing Heterogeneous Simulations Interoperability". *In Proceeding of the 2010 Winter Simulation Conference*, edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, 2968-2980. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Boyd, W. A., and Hessam S. Sarjoughian. 2020. "Composition of Geographic-Based Component Simulation Models". *In Proceeding of the 2020 Winter Simulation Conference*, edited by K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, 2257-2268. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Chow, A. C. H., and B. P. Zeigler. 1994. "Parallel DEVS: A Parallel, Hierarchical, Modular Modeling Formalism". *In Proceeding of the 1994 Winter Simulation Conference*, edited by J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila. 716-722. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

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): 748-771.

Dai, J., S. Wu, G. Han, J. Weinberg, X. Xie, X. Wu, X. Song, B. Jia, W. Xue, and Q. Yang. 2018. "Water-Energy Nexus: A Review of Methods and Tools for Macro-assessment". *Applied Energy* 210: 393-408.

Fard, M. D., and H. S. Sarjoughian. 2020a. "A RESTful Framework Design for Componentizing the Water Evaluation and Planning (WEAP) System". *Simulation Modeling Practice and Theory* 106:102199.

Fard, M. D., and H. S. Sarjoughian. 2020b. "Coupling WEAP and LEAP Models Using Interaction Modeling". *In Proceedings of the 2020 Spring Simulation Conference*, edited by K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, 1-12. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Fard, M. D., H. S. Sarjoughian. I. Mahmood, A. Mounir, X. Guan, and G. Mascaro. 2020. "Modeling the Water-Energy Nexus for the Phoenix Active Management Area". *In Proceeding of the 2020 Winter Simulation Conference*, edited by K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, 2317-2328. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Hoff, H. 2011. "Understanding the Nexus: Background Paper for the Bonn2011 Nexus Conference". The Water, Energy and Food Security Nexus.

Howells, M., S. Hermann, M. Welsch, M. Bazilian, R. Segerström, T. Alfstad, D. Gielen, H. Rogner, G. Fischer, H. Van Velthuizen, and D. Wiberg. 2013. "Integrated Analysis of Climate Change, Land-Use, Energy and Water Strategies". Nature Climate Change 3: 621-626.

LEAP 2021. "Long-range Energy and Planning System". Stockholm Environment Institute. https://www.energycommunity.org, accessed 10th April 2021.

Martinez-Hernandez, E., M. Leach, and A. Yang. 2017. "Understanding Water-Energy-Food and Ecosystem Interactions Using The Nexus Simulation Tool NexSym". *Applied Energy* 206: 1009-1021.

Mayer, G. R., and H. S. Sarjoughian. 2016. "Building a Hybrid DEVS and GRASS Model Using a Composable Cellular Automaton". *International Journal of Modeling, Simulation, and Scientific Computing* 7 (1): 1-31.

McLaughlin, M. B., and H. S. Sarjoughian. 2020. "DEVS-Scripting: A Black-Box Test Frame for DEVS Models". *In Proceeding of the 2020 Winter Simulation Conference*, edited by K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, 2317-2328. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Mittal, S., J. L. Risco-Martín, and B. P. Zeigler. 2009. "DEVS/SOA: A Cross-Platform Framework for Net-Centric Modeling and Simulation in DEVS Unified Process". *Simulation* 85(7): 419-450.

MongoDB 2021. "MongoDB Documentation". https://docs.mongodb.com, accessed 10th April 2021.

Parker, Z., S. Poe, and S. V. Vrbsky. 2013. "Comparing NoSQL MongoDB to an SQL DB". *ACM Southeast Conference.* 1-6.

Sarjoughian, H. S. 2006. "Model Composability". *In Proceeding of the 2006 Winter Simulation Conference*, edited by L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 149-158. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Sarjoughian, H. S., and V. Elamvazhuthi. 2009. "CoSMoS: A Visual Environment for Component-Based Modeling, Experimental Design, and Simulation". In *Proceedings of the International Conference on Simulation Tools and Techniques*, March 2nd-6th, Rome, Italy, 1-9.

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

Tsai, W. T., C. Fan, Y. Chen, and R. Paul. 2006. "DDSOS: A dynamic distributed service-oriented simulation framework". *Annual Simulation Symposium (ANSS'06)* 8.

Wang, W., Wang, W., Zhu, Y. and Li, Q., 2011. "Service-oriented simulation framework: An overview and unifying methodology". *Simulation* 87 (3): 221-252.

WEAP 2021. "Water Evaluation And Planning System". http://www.weap21.org/WebHelp/index.html, accessed 10th April 2021.

Zhang, P., L. Zhang, Y. Chang, M. Xu, Y. Hao, S. Liang, G. Liu, Z. Yang, and C. Wang. 2019. "Food-Energy-Water (FEW) Nexus for Urban Sustainability: A Comprehensive Review". *Resources, Conservation and Recycling* 142: 215-224.

AUTHOR BIOGRAPHIES

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.

MOSTAFA D. FARD is a Ph.D. student in the 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.