ADVANCED THEORY AND SIMULATIONS
www.advtheorysimul.com

A Web-Based Architecture to Operationalize Geospatial Simulation Environments

Gabriel Wainer and Bruno St-Aubin**

Large scale geospatial simulation projects require multidisciplinary efforts by actors with highly variable skills and domains of expertise. Subject matter experts, modelers, developers, analysts, and decision makers must collaborate closely to model a real-world system, simulate it, analyze its results and disseminate them. Simulation environments, tailored to business scenarios, can provide the necessary support to facilitate their collaboration throughout the simulation lifecycle. Commercial modeling and simulation software can provide an environment to facilitate simulation studies for users but, they tend to be narrowly scoped. This research focuses on the different categories of users and introduces four business processes that carry those users across the simulation lifecycle. These concepts are translated into an architecture that facilitates the operationalization of geospatial simulation environments using modeling and simulation as a service and Discrete Event Systems Specification.

1. Introduction

Geospatial simulation is inherently complex and multidisciplinary,[1,2] It requires subject matter expertise about the system under study (economy, biology, logistics, etc.), knowledge of simulation theory and an understanding of geospatial concepts (coordinate systems, spatial analysis, topology, etc.). Across the lifecycle of a simulation project (from knowledge acquisition to the operationalization of a model), different skills are required. Beyond trivial systems, it is unlikely that a single participant possesses all the skills required to conduct a simulation project by themselves. Therefore, multi-disciplinary collaboration is commonplace when developing, validating, and operationalizing such simulation models.

Consider for example, a demography model meant to study the effects of socio-economic policy changes on a given municipality. Depending on the policies to evaluate, understanding the system may require urban planners, engineers, policy makers, economists, etc. Abstracting the city into a model requires that they communicate with a simulation expert that can translate their knowledge into a model. Once simulated, data scientists and analysts must extract patterns and meaning from the results of a simulation experiment and then pass their analyses along to the policy makers in a position to make decisions. Other scenarios would unfold similarly. Studying patient pathways in healthcare for example, would require knowledge of the healthcare system so perhaps doctors, nurses or hospital administrators could be involved as subject matter experts (SME) alongside modelers, programmers, and analysts.

A typical solution to facilitate and support users is to build single use simulators that cater to very specific problems. Since end users are well known, the software can be tailored to their skills and knowledge base. Examples of such approach would include EnergyPlus (a well-established engine for energy simulation in buildings, which includes HVAC, lighting, etc.),[3,4] OpenFlows FLOOD (a geospatial simulation software to understand and mitigate flood risks[5]), DIALux[6] or Radiance[7] (for planning indoor and outdoor lighting in design). These software provide extensive sets of domain-specific parameters that can be used to conduct simulation on a system. They also tend to adopt language that is familiar to their intended user's domain of expertise. The software relies on black box models that are highly coupled with the simulator, enough so that models become indistinguishable from the simulator. Therefore, they are difficult to reuse outside of the application domain they were intended for. This leads to the proliferation of single use simulators that cannot be used easily in the context of multidisciplinary projects. Indeed, the integration of single use simulators to compose larger scoped simulations is a complex problem that has led to co-simulation research, a subfield of M&S that consists of the theories and techniques to enable global simulation of a coupled system via the composition of simulators.[8]

Geospatial simulation software exhibit the same challenges: their models are often ad hoc and tightly coupled to the software, making it difficult to share, reuse, validate or combine them with other models.[9] They are generally siloed within highly specialized subfields (as discussed above, or others, such as

G. Wainer, B. St-Aubin
Dept. of Systems and Computer Engineering
Carleton University
Ottawa, ON Canada
E-mail: gwainer@sce.carleton.ca; brunostaubin@cmail.carleton.ca

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/adts.202400144

DOI: 10.1002/adts.202400144

flood management[10–12] forest fire response[13–15] land use[16–18] or transportation,[19,20]).

In this research we propose a different approach: using a formal specification as the core foundation of modeling and simulation activities. We integrate the Discrete Event Systems Specification (DEVS) into an architecture that supports multi-disciplinary simulation studies. DEVS is a modular and hierarchical simulation method that can be used as a common denominator for any other simulation formalism.[21] Any models that follow the specification can be coupled to compose models that represent larger, more complex systems. It is therefore possible for modelers to assemble multi-disciplinary libraries of models to be reused in simulation applications. Despite its advantages, DEVS has only seen limited usage in industrial and commercial scenarios. Besides notable exceptions such as the MS4Me software,[22] very few DEVS simulators can be considered to be consumer off the shelf (COTS) software, ready to be used by non-expert users.

As noted in,[23] it is unlikely that a simulation approach will be taken seriously by industry and the general public unless it can be packaged as a COTS software. In the rest of the article, we show our main contribution of this research: an architecture that contains the components necessary to easily prepare ready-to-use simulation environments tailored to the end user's business requirements. The architecture relies on a clear definition of roles and responsibilities to cater to a series of business processes for modelers, subject matter experts (SME), web developers and end users. The research addresses long standing simulation challenges identified by the community. Notably, we provide new means to democratize the use of DEVS modeling and simulation by lowering the barrier to entry, providing tools for collaboration and a data-centric approach to simulation modeling. We show how the proposed method is well suited to data-heavy, multi-disciplinary processes like geospatial simulation. We also discuss how it also preserves key features of DEVS (genericity, modularity, flexibility, etc.) and encourages users to follow best practices in model documentation which fosters model reusability and improves model discoverability.

2. Background and Related Work

Beyond the practical challenges faced by modelers discussed in the introduction, there are various issues that warrant further study and that contribute to the lack of adoption for generic simulation methodologies:

- The need to lessen the modeling efforts through model composability.[24,25]
- Adequate tools and environments to support collaboration over the entire simulation lifecycle, including experiment management, results analysis, feedback process, etc.[26,27]
- Democratization of M&S should capitalize on the fact that technical skills are now prevalent in all research fields. Researchers require adequate tools to empower them to collaborate across multiple disciplines.[25–27]
- Modeling and simulation as a service (MSaaS) is deemed necessary for increased accessibility to simulation in decision-making processes.[25,26,28]

As discussed in,[26] complexity, size and quality of M&S projects are limited by the methods and tools that we have, and the authors propose the use of *Web Simulation Science (WSS)*, or *Modeling and Simulation Ecosystems* to create theories, methods and technologies needed to realize large scale simulation projects.[23] proposes the use of *hypermodeling* principles,[29] which allow models to be interconnected, integrated and linked to heterogeneous documents. Although WSS has not gained much traction in the M&S research community, related topics have generated higher volumes of literature which are coherent with a WSS approach[30–32] Research on model and simulation as a service or simulation visualization could also be considered related to the concept[33–36]

Component based modeling (CBM) has been put forth to facilitate the role of the modeler in the simulation lifecycle. CBM decomposes complex systems into self-contained, reusable building blocks. For instance, in plant engineering piping and instrumentation diagrams (P&ID) are used to automatically derive simulation models.[37] Examples of P&ID to automate simulation modelling are abundant[38–41] It is used in many other field such as construction where CYCLONE (Cyclic Operations Network)[42] is a popular method to simulate job site processes.

Another common way to support modelers is through visual programming languages. CoSMoS for example, is one such tool.[43] CoSMoS users can manually connect boxes representing individual models to compose a larger coupled model in Java. DesignDEVS[44] also allows modelers to choose from predefined models, configure them through a user interface (UI) and assemble them. Other examples include[45] or CD++ Builder.[46] These tools specifically target the modeling activities and only offer limited capacity at supporting the remainder of the lifecycle.

Simulation as a service has been proposed to support the simulation execution step. *RESTful Interoperability Simulation Environment* (RISE)[47] exposes a simulator as an easily accessible web service. By offloading the simulation to one or more servers, it lowered the technological barrier to entry for simulation. However, it does not offer any support to the simulation lifecycle beyond the remote execution of a simulation.

Research on post-simulation activities such as visualization and analysis of simulation results is more uncommon.[48] wrote that "visualization offers one of the most promising means to convey information from a simulation model to decision-makers in a meaningful way". Macal also remarked a clear gap in this subfield of simulation.[49] In,[50] authors highlighted the importance and role of visualization for simulation by writing that "ultimately, the acceptance of complexity models within mainstream science and society will depend on the results that are produced visually". But often, research on visualization occurs in the context of specific application domains. In[51] CLINSim was used to model queues in hospitals using specific icons to show different actors, colors to show their states, vector graphics for rooms, etc. CAPSIS is an open-source software designed to model and simulate forest growth modeling.[52] Likewise,[53] built a framework to follow microbial contamination of produce across supply chains. Numerous other examples of domain specific visualization tools that strongly coupled to models or simulators exist.

2.1. Discrete Event System Specification

DEVS is a well-established technique for efficiently modeling real-life systems. It is derived from systems theory and was first described by Bernard Zeigler in 1976.[24] It provides a discrete event-based method to abstract systems into models that can be used for experimentation in cases where it is impractical or impossible to experiment on the real system. A complete description of the formalism can be found in.[24]

By its nature, DEVS has all the characteristics required to support a data-centric, component-based modeling approach like the one used in the architecture we propose. It supports hierarchical and modular model development, favoring the reusability of models. Through composition, any model can be reused as a component to represent larger and more complex systems. It has been demonstrated that the DEVS formalism can be used as a common denominator for any formal method of modeling, whether discrete or continuous.[21] But this generic nature comes at a cost: increased complexity in the modeling process. Coding DEVS models often relies on a simulator specific framework, which means that models prepared for one simulator are not compatible with another simulator even if both are based on the same programming language.

A common problem is that modelers often do not adequately separate experiments from models. Experiment parameters are hard coded in the models which makes it difficult to reuse them. There is also the possibility of namespace clashes when trying to reuse models. Attempts have been made to decouple model and simulator through small languages. Cell-DEVS models in CD++ is one such example.[25] If many simulators were to adopt these small languages, then models could be reused easily across simulators but, this has not been the case due to significant limitations. Otherwise, standardization of models has been attempted only with mitigated success.[28]

2.2. Simulation in Geographic Information Systems (GIS)

Geographic Information Science (GIScience) is concerned with acquisition, management, processing, analysis, representation, and storage of geographic data. Geographic Information Systems (GIS) are software tools designed to carry out these tasks. GIS have become standard tools used in an increasingly large array of applications; urban logistics, land use planning, emergency responses, natural resources inventory, tracking epidemics, socio-economic analysis, etc. They can include simulation capacity,[54] however, their simulation potential is limited; they offer few models that cannot be extended, and the performance is constrained by the computer's hardware.[55]

Adoption of GIS as a geostatistical tool occurred rapidly and the proliferation of geospatial data followed suit. Big data has become commonplace in the field and has led to a number of challenges, which can be summarized as follows:[56]

- Volume: The large size of the geospatial data that can sometimes exceed the capabilities of current computer systems.
- Velocity: The varied and sometimes high frequency with which geospatial datasets are generated and updated can exceed the capacity of current computer systems.
- Variety: The heterogeneous nature of geospatial data. The data originates from different sources, has different levels of accuracy, is stored in different formats and can be more or less structured.
- Veracity: The unreliability of data in terms of precision, accuracy, or other aspects of uncertainty.

Although this can be somewhat mitigated using certain strategies (generalization or down-sampling for example), GIS data often originates from heterogeneous sources, another challenge to geospatial modelling. Data standards are uncommon. Datasets on a common topic may contain different attributes or the value domain for an attribute may be different. For example, the city of Toronto, Montreal and Ottawa each describe bicycle paths differently.[70–72] There have been efforts to standardize various aspects of GIS and the data it uses, but most standards are focused on semantics, file formats, and web services[57] and in practice are rarely applied. Some of these challenges can be overcome through robust GIS ecosystems and tools.[58]

Democratization of GIS occurred naturally over time. Although their evolution was originally driven by business requirements of users, they have reached a level of maturity where they now generate business opportunities for users. The web-based architecture for DEVS simulation we propose in this paper is inspired from GIS environments and architectures. Like GIS, we believe that a robust but flexible architecture is a necessary step towards the democratization of simulation.

3. Users and Gaps in the Simulation Lifecycle

Abundant research on the simulation lifecycle exists.[59] Most follow a similar pattern where the emphasis is on the steps that lead to the simulation itself rather than those that follow. Some research exclusively focusses on pre-simulation steps. For example, Sargent in[60] presents a bidirectional iterative cycle of three steps to represent the modeling process: problem entity, conceptual model, computerized model. The author puts data validity at the heart of the process, each step is subject to a specific type of validation: conceptual model validation, computerized model validation and operational validation. Some lifecycles proposed such as the nine step version proposed by Benjamin et al.[61] and derived by Loper[62] are straightforward and focus mostly on the pre-simulation steps. In[63] the authors present a version of the lifecycle that slightly modifies the nomenclature of Law's process and emphasizes the importance of the data collection step. To frame our architecture, situate users in it and identify gaps, we propose a simplified version of the lifecycle, illustrated in **Figure 1**.

The lifecycle includes 3 pre-simulation steps. The initial *system knowledge* step consists of gathering information on the real-world system to model. This can be achieved through data or through discussions with SMEs. This step covers problem formulation, data acquisition and conceptual model definition. It can lead to several artefacts such as data files, a collection of scientific papers, diagrams, etc. The *model* step consists of building the model, testing (pilot simulation runs) and validating it. The typical artefact generated by this step is the source code for the simulation model. The *simulation* step is the execution of the simulation using the model previously built. It includes the

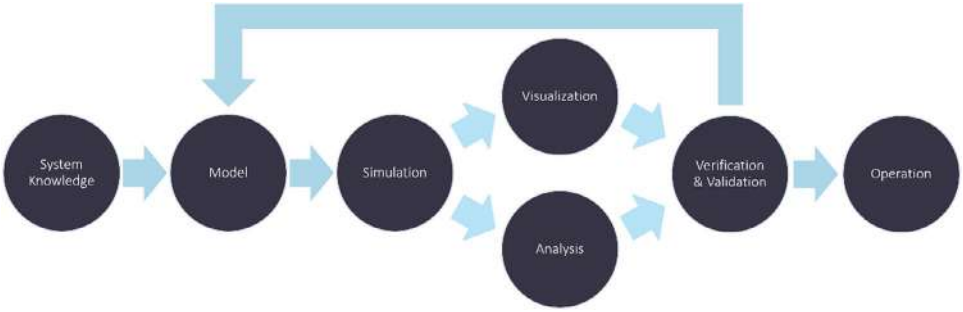

Figure 1. Simplified simulation lifecycle.

preparation of experiments and leads to a set of simulation results in the form of log files containing the simulation trace.

Post-simulation includes 4 steps. The *visualization* and *analysis* steps, when included in simulation lifecycles, are generally presented as a single step. In our case, we decided to separate them since we consider that each requires different skillsets to accomplish and therefore involve different users. *Visualization* is exclusively the process of preparing a visual representation of the simulation trace through whatever platform is appropriate for the model. For cellular automata models, this could be a lattice of cells, for spatial models, possibly a map. *Analysis*, on the other hand, is the act of extracting meaning from the simulation outputs, often through statistics or data science related processes. Both steps can lead to various simulation artefacts such as data tables, diagrams, animations of the simulation trace, analytical notes, etc. *Verification and validation* serve as a decision point on whether the model requires an additional refinement iteration or whether more experiments should be conducted. Finally, the *operation* step is where simulation artefacts are put into operation. This may be achieved in different ways, some simpler than others. In this research, we focus on the dissemination of simulation artefacts through a web-based simulation application that leverages any simulation lifecycle artefact previously mentioned.

3.1. Users in the Simulation Lifecycle

Large scale multi-disciplinary simulation projects unavoidably involve actors with various roles and backgrounds. Although it is clear in the literature that authors assume that multiple stakeholders are involved in a simulation project, we have found no clearly accepted categorization of their personas. However, in[59] Balci identifies four types of stakeholders: the M&S developer, the project manager, the organization (client) and the community of interest. Unfortunately, the categories of users are ill-defined, and it is unclear what their roles are in the lifecycle. Therefore, to frame the lifecycle we propose, we review and summarize the categories of users we originally described in (**Table 1**).[64]

In the lifecycle we propose, the SMEs occupy a central role; they provide the real-world knowledge upon which the model is built and against which it is analyzed and validated. A SME could be an epidemiologist for disease spread models, an urban planner for traffic simulation, an economist for socio-economic policy planning, etc. SMEs are specialists in their field, they are not cross trained in simulation theory or web services technology. They must transfer their knowledge of the real-world system to the modeler who translates it into a model ready to be simulated. Therefore, a modeler must be skilled in the programming language of the simulator and its modeling framework. To operationalize a model, web developers prepare web environments tailored to the needs of their end users. This can be as simple as a web application that displays interactive charts and tables for sets of pre-run simulations or as complex as a web application that allows end users to experiment with a model or even build their own models. The nature and complexity of a simulation environment depends on the use case and business requirements it is tailored to. Web developers require a different subset of software development skills than modelers: web development, networking, graphic design, etc. Finally, the end users in the

Table 1. Summarized categories of users from.[64]

Stakeholder	Role	Skills	Requirements
Model developer	Build model Run simulation	Simulation formalism Limited coding skills	Assisted modelling Debugging tools Organized models Documented models
Subject matter expert	Provide System knowledge Analysis Verification, validation	Data science and analysis Subject matter expertise	Standardized simulation outputs Parse simulation outputs
Web developer	Visualization Dissemination	Software engineering	Tools to abstract the complexity of the simulation
Decision-maker	Decision	Organizational knowledge	Intuitively presented simulation results (reports, infographics, etc.)

Table 2. A subset of Grand Challenges identified over the past decade.

Gap	Author	Description	Steps of the lifecycle
A general simulation environment (web simulation science)	Khan,[25] Khan,[27] Taylor[26]	An environment to support simulation related activities for multi-disciplinary users.	All steps
Model discoverability and composability	Morse and Tolk,[25] Morse[27]	Allow users to find previously built models, explore them, and reuse them to build larger scale models.	Modeling
M&S as a service	Tolk,[25] Tolk[27]	Lower the technological barrier to entry for simulation by offloading simulation to web services.	Modeling, Simulation
Replicability	Yilmaz,[25] Yilmaz[27]	Facilitate simulation experiment reproduction through various means.	Modeling, Verification and Validation
Democratization	Khan,[25] Zander and Mosterman,[25] Zander,[27] Page[26]	Lower the barrier to entry for simulation by providing tools to all types of simulation users	Analysis, Visualization, Decision

lifecycle operate the dissemination application. Although they do not possess any specific technical skills, they can be expected to understand the business domain within which the dissemination application is operated since it is tailored to their requirements.

It should be noted that these categories of stakeholders are meant to be a guide for the division of labor within the simulation lifecycle in a larger organization. They are not immutable. In many cases, there will also be overlap between them; one stakeholder may play multiple roles. For example, the modeler may also be responsible for developing a web application for dissemination since there is some overlap in the skillsets required for both steps. In certain fields, for example those where SMEs are also data scientists, it may be conceivable for an SME to learn how to develop a model using a specific simulator framework. For small simulation projects, it is entirely possible that a single person could assume all the roles.

3.2. Gaps in the Simulation Lifecycle

Across the simulation lifecycle, there are multiple pressure points that users must face. Many of these gaps were captured through panels on the grand challenges of simulation that have occurred over the last decade.[25–27] In the following table, we summarize and describe the gaps for which we propose mitigation strategies further in this paper and we identify the steps of lifecycle and users that they affect (**Table 2**).

We believe that many of these challenges can be addressed through the definition of a collaboration process between actors of the simulation lifecycle and an environment that supports simulation activities for them. The architecture can spawn environments that increase model discoverability through web services and favor the replication of experiments through proper documentation of models with a metadata specification. It can also contribute to the democratization of simulation by exposing simulation resources as a service which lowers the barrier to entry.

4. An Architecture to Support the Simulation Lifecycle

In this chapter, we present a web-based architecture that enables users to build and publish geospatial simulation environments tailored to their business needs while considering the skills and expertise of different actors. We focus on geospatial use cases since the data sources used in this field lend themselves well to data-centric modeling and can lead to applications in diverse fields. The architecture relies on geoprocessing workflows for model composition, interoperable simulation artefacts and a toolbox to enable the visualization of simulation results. We first present a conceptual overview of the architecture.

4.1. Architecture Overview

The architecture allows non-expert users to prepare simulation environments that are customized according to domain-specific requirements. It is a flexible architecture that provides several capabilities that can be employed partially or in their integrality to operationalize geospatial simulation models. It serves diverse purposes: model documentation, automated transition between experiment and model, visualization support, etc. Any of these capabilities can be integrated into environments spawned by the architecture. The figure below describes the web-based architecture for geospatial simulation at a conceptual level and summarizes the roles that different users assume to prepare custom simulation environments for end users (**Figure 2**).

The architecture relies on a central repository that holds artefacts generated as users collaborate across the business processes defined further in this section. SMEs possess knowledge of the application domain at the center of a simulation project. They provide the real-world knowledge upon which models are built. They are also responsible for analyzing results and preparing visualizations as well as creating Model Composition Workflows (MCW) that can be called to generate models from geospatial data sources. Model metadata files, MCWs, analyses and visualizations are stored in the central repository and can be reused at any other point by other users. The modeler uses the knowledge provided by the SME to build a conceptual model, implements it, and adds it to the library of models, also contained in the central repository. Any model can be used in MCWs once they have been added to the library of models.

At this point in the process, web developers, in collaboration with end users, can create an environment for simulation-based experimentation. End users must first provide the requirements

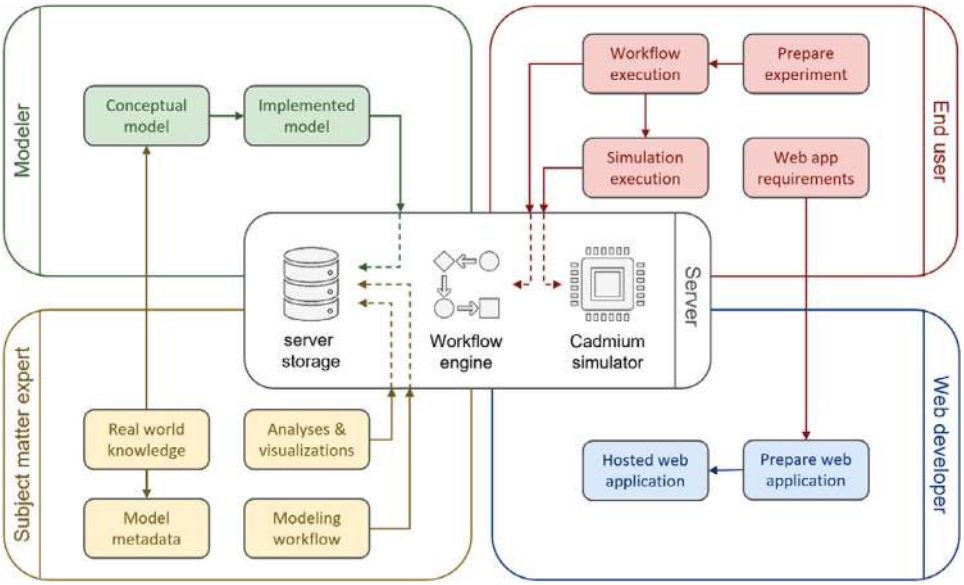

Figure 2. Overview of roles and responsibilities in the architecture proposed.

for their business case (which workflows they will use, the data sources required, the look and feel of their application, etc.) Web developers can then build an application that will fulfill those requirements and act as a simulation experimentation environment for end users. With the help of a front-end library of tools, web developers do not require any simulation expertise to build visualization platforms. The environment built this way can reuse any simulation artefact in the central repository. End users can leverage any of the resources to conduct experiments, analyses, visualizations, and disseminate them.

4.2. Interoperable Simulation Artefacts

The integration of the architecture's components requires interoperable simulation artefacts that "glue" them together. To achieve this, we propose specifications for two key simulation artefacts: model metadata and simulation results. Well-specified, machine-readable artefacts can be used to integrate components seamlessly. In an ideal scenario, this would rely on an interoperability standard for DEVS simulation artefacts but, to the best of our knowledge, there are no such standards. There has been no concerted effort by the DEVS community or the wider simulation research community to design standards to document DEVS models through metadata or to properly structure simulation results.

4.2.1. Metadata Specification for DEVS Simulation Models

The metadata specification we propose is derived from the Dublin Core Metadata Initiative (DCMI) with additional elements to describe DEVS models. Appendix 1 describes each element of the specification in more detail. For each element, we provide its label, a description and indicate whether they are mandatory (M), optional (O) and repeatable (R). A more detailed version of the specification with examples is available.[65] This specification has been the subject of a separate publication.[66] For the sake of brevity, we only discuss the elements introduced to support DEVS models in this section. These elements are meant to document the structure and behavior of DEVS atomic or coupled models with the goal of providing a general, high-level understanding of the model.

The *time* metadata element documents the time representation used for the model. In DEVS, models can use different time representations. For example, one model could consider one unit of time to be a day while another would consider it to be a second. In these cases, models may simply be incompatible, or the simulator may require a time conversion when it determines the next event to occur in a simulation. In any case, this element allows a system to warn users trying to couple models with different time representations.

The *behavior* element provides a high-level description of the model's behavior or in other words, a description of its internal, external, output and time advance functions. This is mainly meant to support model discovery. The *state* element provides information about the variables that constitute the state of a model. It contains repeatable *variable* elements for each state variable of the model. Each variable must contain a *name* to label the variable and optionally, a high-level *description* to provide context to users. It also includes a *message* identifier that references the type of the message that is logged when the state is output. Many DEVS simulators output the state of a model at certain steps of the simulation process. The messages logged are often not sufficient by themselves to be automatically interpreted in post-simulation applications. To mitigate this, the specification includes an element that captures all the information required to interpret them further in this section. Therefore, the message element here is a unique identifier that refers to a single message definition documented at the end of this section.

A *subcomponent* is a model instance that composes a coupled model. Therefore, only coupled models should document this

```
{
   identifier: vehicle_output
   fields: [{
      name: velocity,
      description: the vehicle's current velocity
      type: numerical
      uom: kmh
   }, {
      name: position,
      description: the vehicle's current position
      fields: [{
         name: latitude,
         description: latitude of the vehicle
         type: numerical
         uom: degrees
      },{
         name: longitude,
         description: longitude of the vehicle
         type: numerical
         uom: degrees
      }]
   }]
}
```

Figure 3. A simulation message defined according to the metadata specification.

element. Each subcomponent consists of a value that identifies the instance and a *model* element that identifies the model type associated to the instance. For example, a processor coupled model could have 4 instances of a core model where each one is identified by a letter. Coupled models should also provide a *coupling* element that documents the internal and external couplings of the model. Each coupling identifies the origin model (*from model*), origin port (*from port*), the destination model (*to model*) and the destination port (*to port*) involved. The repeatable *port* element documents a model's ports through which messages are output. Each port specifies a *name* to label it, a *type* (whether it meant for input or output) and is associated to the *message* type used when it outputs and logs a message.

The final element in this section is one of the most useful. It documents each type of *message* that is output by a model, either when its state is logged or when a message is output through a port. It provides a way for post-simulation applications to visually represent a simulation trace in post-simulation applications. It also contains the needed context for users to correctly understand the contents of messages. One model may output many different messages either when the simulator logs the state of a model or when a model outputs a message through a port. Each message has a unique *identifier* so that it can be referenced by both the state and port elements of a model. Each message also specifies one or more *field* elements where each *field* describes a value from the message. Each *field* has a *name* used to label the value and a *description* to provide contextual information to users. Each field also has a series of optional qualifiers for the value. *Type* indicates whether the variable associated to the value is nominal, numerical or ordinal. This can be used when preparing color scales when visualizing a simulation trace. *Uom* informs of the unit of measure associated to the *field*; this provides additional context for interpretation. Units of measures can be considered when validating coupling between models. Finally, a *field* element can also contain nested *field* elements. This is used to represent messages with a more complex data structure. For example, a vehicle model that outputs a message containing its position and travel velocity could be documented using JavaScript object notation (JSON) as shown in the figure below. Note that the latitude and longitude fields are nested within the position field (**Figure 3**).

Table 3. Example assumptions made by simulation tools when interpreting log files.

Assumption level	Example assumption when reading log files
Simulator	• For CD++, parse only log messages beginning with "Message Y" • For Cadmium, read message data between characters "<" and ">"
Model	• Display a road network on a map to show vehicle models long roads. • Hardcode pictograms to represent specific models like a hospital or a school • Hardcode "speed" and "acceleration" labels for the state variables of a "car" model
Experiment	• Hardcode labels related to the experiment (e.g., growth rate in a demography experiment)

4.2.2. Decoupled Simulation Results

Simulation log files are generated by simulators as they simulate a model. Most DEVS simulators log output messages emitted after an internal transition by the output function. Some simulators, like Cadmium, also output state messages when the simulation time step advances. The message format varies significantly by simulator. Since there are no common format, post-simulation tools to visualize and analyze simulation results are not interoperable. They are built in an ad hoc manner and rely on assumptions made about the simulator, the model or even the experiment. The table below provides examples of such assumptions (**Table 3**).

The format we propose to store simulation results is simple and its implementation is minimalist to reduce the size of log files as much as possible. When considered alongside model metadata, it provides the information required to reconstruct simulation traces and interpret them in visualization applications. The figure below illustrates the integration between the metadata specification and the results specification. The classes introduced in the diagram are generic, and they encapsulate operations that can be used with a variety of data types according to the needs of the simulation engine and the models developed (**Figure 4**).

The results format consists of a list of sequential time frames where each *frame* holds a *time* value. *Frames* act as containers for messages. A *message* can be a *model message* in which case it contains the *data* for the state of a model and is associated to a model *subcomponent*. The message can be reconstructed using the *message type* associated to the state of the subcomponent's model type. A message can also be a *port message* in which case it contains the data for an output message and is associated to a model *port*. In this case, the message is reconstructed using the *message type* associated to the port. Using the integrated specification, a message can be stored in its minimal expression. To reference models and ports, we use their positional index in the metadata

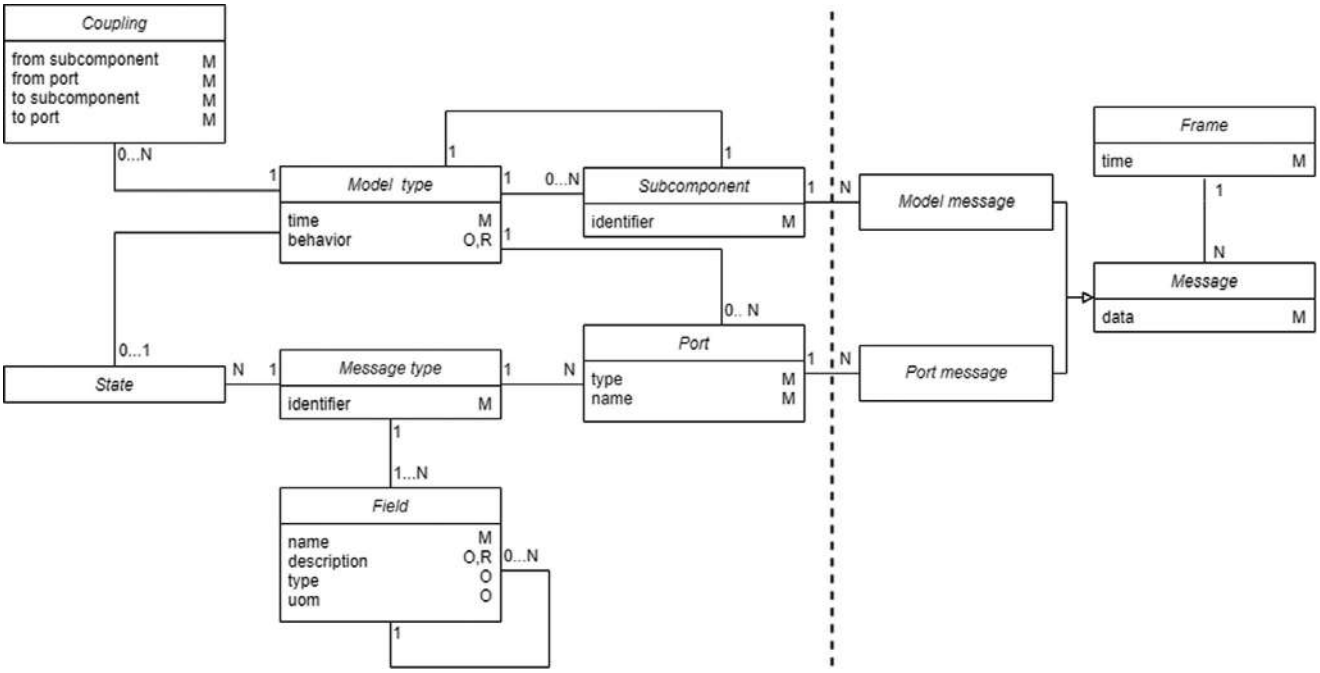

Figure 4. The integrated metadata (left) and results (right) specification (O stands for optional, M for mandatory and R for repeatable).

specification, and we use the message metadata to reconstruct the message. Consider for example, the sample simulation results below (**Figure 5**).

The first line contains a single numeric value that indicates a new time frame. Each message that follows, until another line with a single digit is reached (or the end of file), is part of that time frame. In this example, the 8th time frame of the simulation contains 4 messages. A message containing a single numeric value to the left of the semi-colon is a model state message. In this case, the value represents the positional index of a subcomponent in the coupled model. A message containing two numeric values to the left of a semi-colon is a port output message. In this case, the first value again identifies the subcomponent by its position while the second value represent the position of the port in the port list of the subcomponent's model type. Values to the right of the semi-colon contain the message data which must be injected into the message type associated to either the state or the port. A complete description of the reconstruction process is beyond the scope of this paper and will be the subject of a future paper.

4.2.3. Business Processes of the Simulation Lifecycle

The integrated architecture described in 4.1 has components to support each user category in conducting multi-disciplinary simulation studies. Its loosely coupled components can be partially or integrally implemented to support some or all the business processes involved in the simulation lifecycle. A business process (BP) is a coordinated chain of activities intended to produce a result, or a repeating cycle that reaches a goal.[67] A single BP can involve interaction between different categories of users and systems in an organization. BPs divide the work involved in a simulation project into units that can be addressed by experts with different skillsets. In this section we decompose the simulation lifecycle in four main BPs to provide a practical foundation to build DEVS simulation environments.

```
8                          → new time frame
617;9                      → model state message
618;9                      → model state message
619;36,4061,36,3989        → port output message
620;40,5972,90,5842        → port output message
```

Figure 5. Sample simulation results following the specification.

4.2.4. Building New Model Components

Figure 6 introduces a BP for collaboration between the modeler and the SME. It covers the first two steps of the lifecycle and aims at preparing model components, documenting them, and publishing them to be reused in subsequent steps. We use the term *model component* to designate models that can be assembled to represent the system under study. We need to use this BP only when the model components required to represent the real-world system do not already exist in the library of models.

The first activity of this BP consists of a SME transferring their real-world knowledge about the system under study to a modeler who will use it to prepare a conceptual model of a system component. Although there are numerous definitions of what a conceptual model is [65] and there is no consensus on definitions, we focus on what can be acted upon by the modeler in the subsequent activity. Therefore, in this research the conceptual model must describe the possible states of the system, its behavior, the assumptions it requires, its limitations, etc. With this information, the Modeler can implement the model, which

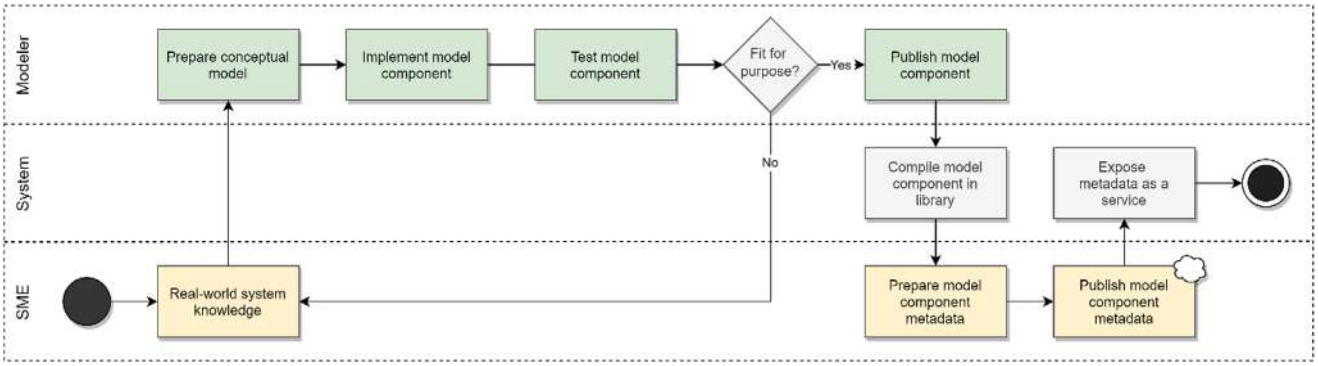

Figure 6. A business process to prepare model components and document them.

involves defining its behavior and structure using the corresponding modeling method, formalism or software framework. In our case we are mainly interested in DEVS models, and therefore, a modeler could use a DEVS modeler to define the structure of the coupled models, and the behavior of the atomic models (in the implementation we introduce in[64,66] we use the Cadmium simulator and its simulation framework). Once a model is complete, the Modeler and SME should evaluate whether it is fit for purpose through functional tests (i.e., simulating the atomic model with test input data and verifying the output). If it is not fit, then it should be conceptually revaluated and rebuilt.

Once a model is deemed to be satisfactory, the modeler publishes it to a library of models, a collection of properly documented, compiled and reusable components available in the system. To do so, Modelers may need to integrate the new model to the library manually (which might involve coding the model in a programming language, adding the code to the library, recompiling it, etc. A code versioning system with continuous integration could be used to automate the deployment of the library of models). Once this is accomplished, the SME should document the model trough metadata and publish the document to the server. At this point the component is exposed publicly by the environment, it becomes discoverable and ready to be reused in the model composition business process we present next.

4.2.5. Model Composition as a Service with Domain Specific Workflows

The second BP, presented in **Figure 7** further supports the modeling steps of the lifecycle by allowing users to build simulation models through domain-specific model composition workflows (MCW). A MCW is a series of steps that maps model components onto real-world data and couples them together using relationships extracted from the data. This approach is important in domains where the data is typically rich and abundant. For instance, in the field of GIS, geospatial data is generally organized as tables that represent conceptual layers on which model components can be mapped. Layers also contain topological and other spatial relationships that can be used to derive model couplings. Similarly, Business Information Models (BIM), commonly used by architects, provide a rich source of data that can be used to compose models. In these fields, workflow approaches to process data are commonplace. The ESRI software suite uses the *ModelBuilder* tool while QGIS, uses the *graphical modeler*. Both tools use a visual programming approach to let users assemble workflows that can extract, transform, and load data. For architectural models, Autodesk Revit provides a similar capability of workflow-based 3D design.

This BP is entirely accomplished by the SME using the architecture. The first step is for a SME to manually prepare a MCW that defines how models will be composed from domain-specific

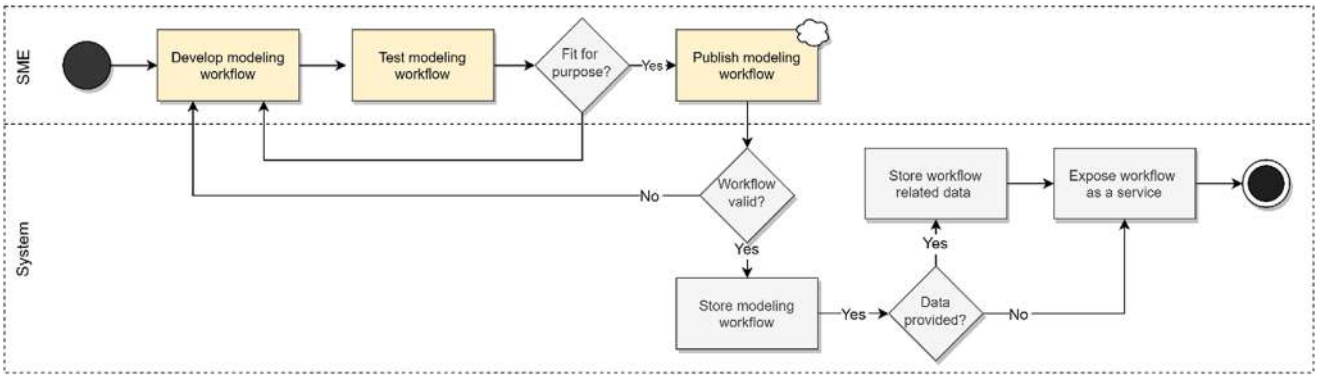

Figure 7. SMEs prepare a model composition workflow hosted as a service.

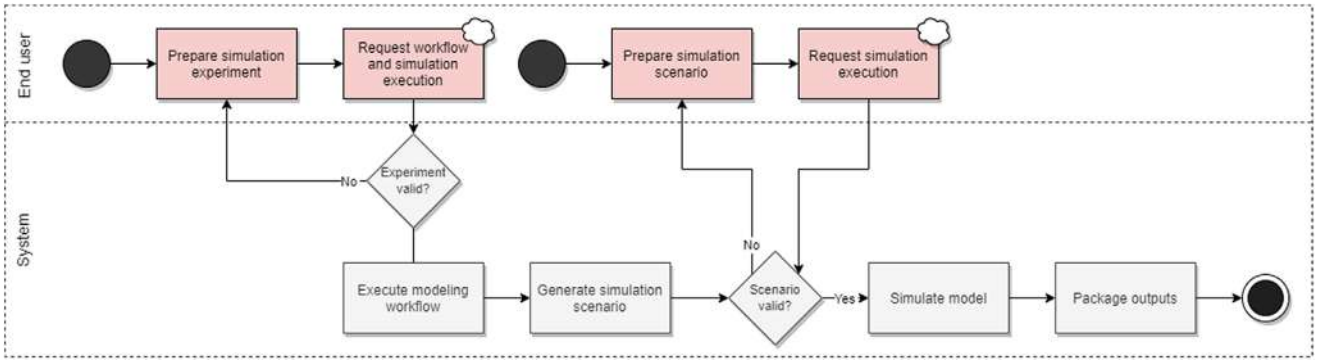

Figure 8. A business process to simulate a user defined simulation scenario.

data. For GIS, a MCW is a sequence of computational geometry and spatial analysis operations (concretely, a JSON or XML file that documents the sequence of operations, input and output data sources, parameters needed, etc.) For example, if users require an intersection between two spatial data layers, then the workflow should include a "spatial intersection" operation with parameters indicating each layer involved and the output layer generated. Similarly, if a spatial buffer is required, then the MCW should include a "buffer" operation and indicate the source layer that will be buffered as well as a distance radius. A MCW file can be prepared manually, or visual programming could be used (i.e., a tool similar to ESRI ModelBuilder).

MCWs generate simulation scenarios used in the subsequent BP to run a simulation. The workflow should provide sufficient parameterization options for users to conduct the experiments required by their business case. Once a workflow is ready, the SME can publish it as a service. Before hosting it as a service, the system validates the workflow. A workflow is valid if its syntax is correct, if the data sources it requires were provided and if the couplings it will attempt to create are valid. To validate couplings, the system uses the model metadata previously published and verifies that input and output ports involved in couplings are compatible. It can verify for example, whether the units of measure of both ports match, whether all model time representations are coherent, if spatial coverages between models match, etc.

Domain specific MCWs provide a way for non-simulation experts to build complex models using a language that is familiar to them. In that sense this BP contributes to democratization of the field. It also favors composability and reusability since it is specifically designed to reuse model components and assemble them in a larger-scale model. Since the workflows are published as services, it would be easy to integrate them into more elaborate contexts such as geospatial optimization studies. Workflows can be as simple or as complex as a use cases requires.

4.2.6. Simulating a Composed Model

The third BP provides Simulation as a Service capabilities. This BP focuses on the third step of the simulation lifecycle described in section 3. Because of the prior work accomplished in collaboration by the modeler and the subject matter expert, this workflow can be conducted by the end user. Indeed, it relies on the models and the workflow that were published to the environment to facilitate the simulation process. Executing a simulation is only a matter of providing the experiment parameters (**Figure 8**).

There are two entry points to this process, and both involve the preparation of a simulation scenario file. A simulation scenario describes a model, its subcomponents, and the couplings between them. With a scenario, the simulator can instantiate a model, simulate it and generate results. The first entry point assumes that users are building a model with a MCW as described in the previous section. The second entry point assumes that users are manually creating the simulation scenario. This is meant to favor loose coupling of the BPs; a user is not required to go through all the BPs to execute a simulation, they can simply simulate a model by directly providing a scenario.

From the first entry point, end users must first prepare a simulation experiment. The experiment is the set of parameters required by the workflow. The end user then requests a workflow execution with the experiment and the system validates that the parameters provided are correct for the selected workflow. The system then executes the workflow that generates the simulation scenario from the experiment. Starting from the second entry point, users must manually prepare a simulation scenario then request a simulation execution. Regardless of the entry point used, the system validates the scenario provided. Various aspects of model couplings can be validated. For example, the system can verify that the couplings involve ports that exist on the models, that messages sent and received use the same units of measure, that time representations are compatible, etc. After validation, the simulation scenario is ready, and the system can simulate it, generate results, package them as outputs and return them to the end user.

This BP provides modeling and simulation as a service and therefore addresses the democratization gap in the lifecycle since it lowers the barrier to entry for users that wish to simulate a model. Indeed, users do not require powerful hardware since the computation is offloaded to the server. In addition, it also offers an easy way to execute simulation through a web API that can be integrated to any other system.

4.2.7. Model and Results Operationalization

The BP in Figure 6 is a collaboration between the web developer, SME and the end users. It covers all the steps of the simulation lifecycle that occur after the simulation step. Its goal is the

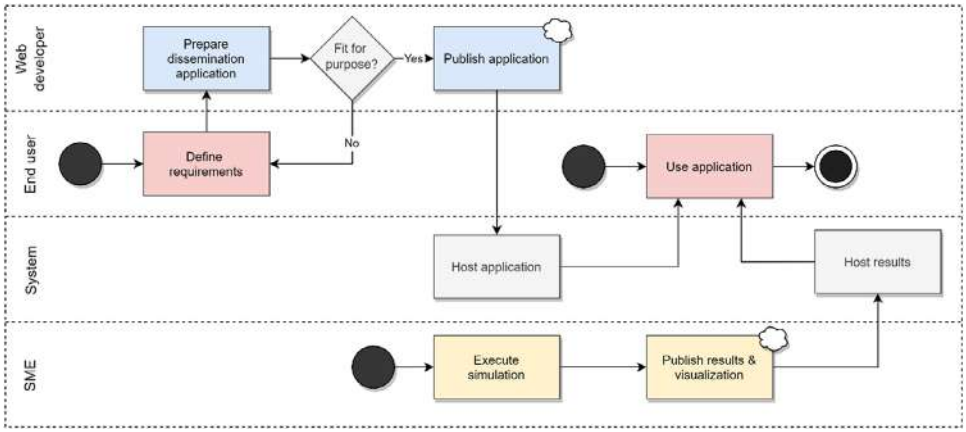

Figure 9. A business process for the preparation of a web-based simulation platform.

operationalization of a simulation model and results. Operationalization can simply mean the presentation of simulation results as tables or charts. In more complex cases, operationalization can be an interactive web application that allows users to conduct their own experiments using an intuitive, browser-based user interface. The complexity depends on the goals the end-user seeks to accomplish (**Figure 9**).

This BP has multiple entry points. The first one involves building a web application for operationalization. End users must first elicit their requirements. This can involve preparing user interface mockups, describing user stories, identifying analytics to be shown to users, clarifying the interactions with the simulation data, etc. This provides guidance for web developers to build a first iteration of the platform. Once the platform is deemed fit for purpose through testing by the end user, it can be published to a server (in our case, a DEVS web server can also serve as a hosting platform). The second entry point provides a sub process for the SME to publish content to be used by an already existing application. The SME can execute simulations and publish their results on the web environment. Once published, the results are available to the platform which can display them to the end user.

Providing a simple and intuitive platform to allow users to interact and analyze simulation results is a step towards democratization of the field. Indeed, it allows non-expert users to understand simulations and even perhaps conduct their own if such is the purpose of the dissemination platform.

5. Architecture Implementation

In practice, there are many ways in which the architecture can be operationalized. Each component of the architecture must be implemented, and several technologies selected to do so, each with their own advantages and disadvantages. In this chapter, we discuss implementation considerations for the architecture and present the choices we made to implement a prototype version of the architecture used in the case study presented in the next section.

5.1. Implementation Overview

Different technologies can be used to implement each component of the architecture. For example, there are several engines that can execute geoprocessing workflows, some are open-sourced, some are proprietary and custom user solutions could even be used. Similarly, there are many possible strategies to implement the library of models: traditional relational database management systems, NoSQL databases or even a non-database approach such as the one we used in our implementation. Each choice of technology has advantages and disadvantages that must be balanced according to the context within which the architecture is implemented. Many factors can be considered: financial considerations (open versus proprietary), familiarity with specific programming languages, the type of front-end platform used to access the environment (desktop, mobile, browser), familiarity with specific web development frameworks (React, vue.js, D3, etc.), the complexity of the spatial analysis operations required, performance required (redundancy, high availability, backups, etc.), and others.

When prototyping the architecture for this research, we tested different technology variations. Considering the academic context in which this research occurred, we opted for open-sourced technologies due to limited availability of funding and resources. We first implemented the architecture using Cadmium for simulation, Spring Boot for services, QGIS for workflows and SQL Server for the library of models. We successfully implemented a prototype using these technologies but observed that a relational database solution to document models was too rigid as we were developing the concept. Data maintenance and evolution of the data model was time-consuming and error prone. Therefore, we built a second version where we replaced the relational database management system by a non-database solution that relies on the file system. **Figure 10** below illustrates the implementation.

5.2. Back-End Library of Models and Simulator

5.2.1. Simulation Artefact Management

This implementation adopts a database-less approach to organize and manage simulation artefacts. It simply uses the server's file system to store artefacts. When installing the web services package, a file system location must be provided. At this location, the system will create folders for each branch of the services that allow users to upload files (*models* to store model metadata files,

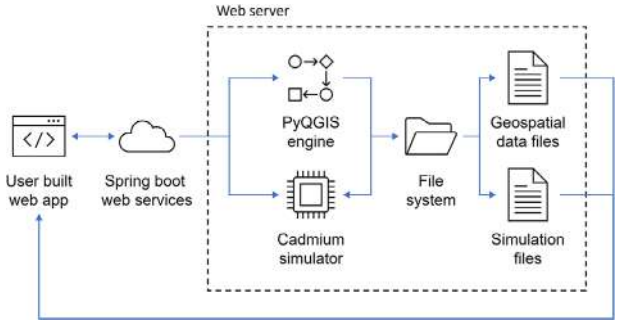

Figure 10. Second version of the implemented architecture.

workflows to store workflow files and, *visualizations* to store visualization related files). The system will also create index JSON files for each folder. Index files list the contents of each folder with a summary description and a creation date. They are used to facilitate discovery. Finally, the system uses a *scratch* folder for the purpose of holding temporary files as different operations are executed.

When users upload resource(s) to the environment the system stores it in the corresponding folder. In some cases, users must upload multiple files for a resource. Workflows, for example, must be published alongside any data sources they require. Therefore, the system creates a sub-folder to hold the files. Subfolders are labeled using a UUID and the identifier is entered in the corresponding index file. The system also creates a REST end point for each resource that allows users to retrieve them, delete them or update them. When a resource is removed from the environment or updated, the index file is updated. The figure below illustrates the folder structure used (**Figure 11**).

5.2.2. Geoprocessing Workflow Engine

The architecture allows users to define and publish model composition workflows (MCW). A MCW consists of a series of sequential spatial analysis tasks that compose simulation models using a CBM approach. MCWs process geospatial data, associate atomic models to records and assemble them in coupled models according to their spatial relationships or other characteristics. In our implementation, users define workflows as JSON files which are then interpreted by a workflow engine we built using PyQGIS, the library behind the QGIS software. A conceptual representation of a workflow file is described below.

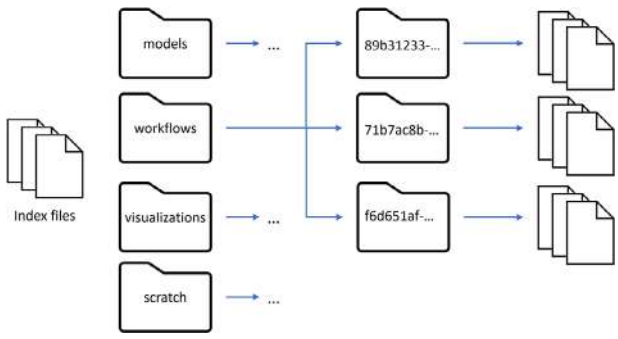

Figure 11. Simulation artefact storage structure.

A workflow contains 3 sections. The *inputs* section specifies the geospatial data layers and non-geospatial attribute tables that are used by the workflow. The *outputs* section indicates which geospatial layers should be output by the workflow in addition to the simulation scenario file. Output data are overlaid on the map upon visualization and serve as a medium on which simulation results can be attached. For example, a workflow that builds a simulation scenario for road traffic could output the road network layer as a contextual layer and color road segments according to the model's state. The *tasks* section contains a list of parameterized spatial analyses and other computational geometry tasks. Each task must identify the spatial analysis operation by name, specify its input data source(s), output locations and any other parameters required by the operation. It is also possible for users to provide parameters that override those contained in the workflow. This mechanism provides more flexibility for users to conduct simulation experiments (**Figure 12**).

When executing tasks, the engine can handle task results in different ways. It can store them on the file system so that they can be reused later for visualization. It can also hold them in memory so that any subsequent tasks in the workflow can reuse them. It can also hold them temporarily in memory so that only the next task can reuse them. This provides some control over memory management for workflow designers. Each task can also have a set of parameters that can be either static or dynamic. Static parameters are invariable parameters that are defined in the workflow file. Dynamic parameters can be provided in several ways and their content is determined at runtime.

Finally, a workflow must also specify how the atomic model instances and relationships will be mapped onto data. They must specify which tasks results the models will be mapped to, the type of model that will be used, the field used to retrieve a unique identifier for the model instance and a series of fields that will be used to initialize model instances. Relation sets are used to establish the couplings between the models. A relation set specifies a join operation on two task results, one for the origin and the other for the destination. Each set of joined records becomes a coupling. A set of relations must also indicate the output and input port that will constitute the coupling.

Workflows are processed in 4 phases: request reception, workflow initialization, task execution and scenario preparation. The first phase begins when a user requests a workflow execution by sending a request to the appropriate REST endpoint. Once the request is received, the engine retrieves the workflow by its UUID. The workflow is loaded and the engine proceeds to load the geospatial data layers that the workflow requires in memory, as specified in the workflow definition file. The second phase consists of executing spatial analysis or computational geometry tasks sequentially. First, a task is instantiated and parameterized using the configuration contained in the workflow file. Then, the engine injects data layers required by the task. This can be a previous result that was explicitly stored or, a data layer that was published alongside the workflow. If the task supports user parameters and the user has provided them, they will be injected into the task parameterization. The final step of parameterization is the injection of previous results if required. Once executed the workflow will output the task results if requested otherwise, it moves on to the next task until none remain. The final phase of a workflow consists of preparing the

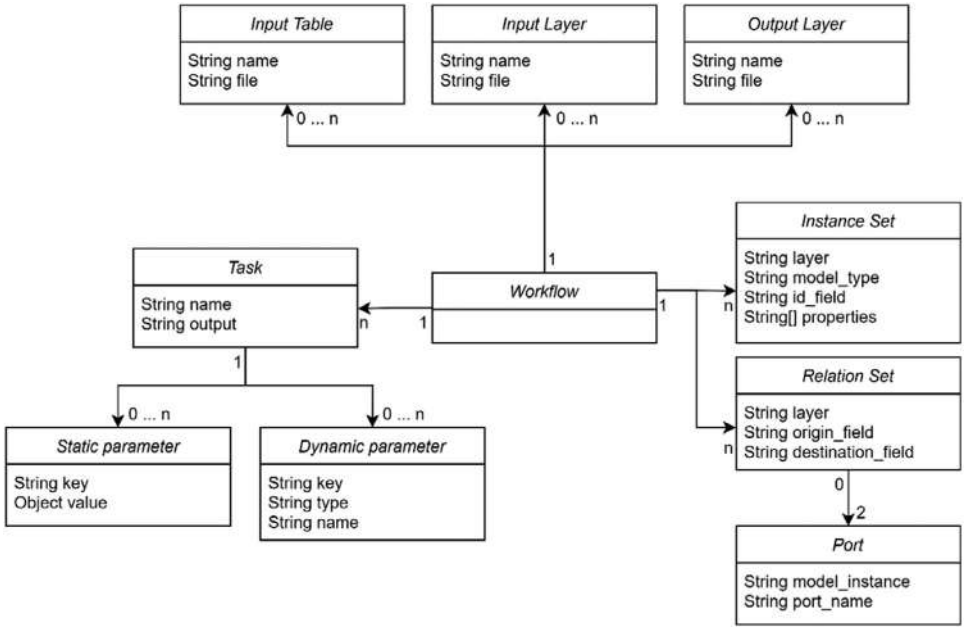

Figure 12. Conceptual representation of a workflow definition.

scenario file itself. To do this, the engine first writes results to disk as specified by the workflow. Then, it creates the sets of instances and relationships which indicate how model instances are created and coupled. Instance sets and relationship sets are written to a scenario file which completes the workflow execution.

5.2.3. Component-Based Simulation in Cadmium

We modified the Cadmium simulator and its framework so that it can build complete simulation models from a scenario file. The specification for a simulation scenario is described in **Figure 13** below. The data structure represents the main coupled model to simulate, sometimes referred to as the top model. A coupled model contains a list of components and a list of couplings that connect models together. Components can be another coupled model or a sourced component and must specify the model type they represents. A sourced component tells the simulator how to create instances of a model type from a data source which can be direct or external. Direct source components use data included in the scenario file as a list of JSON initialization objects used to instantiate model components. Externally sourced components are more complex and not used in the context of this work. Couplings are simple: each coupling indicates the origin and

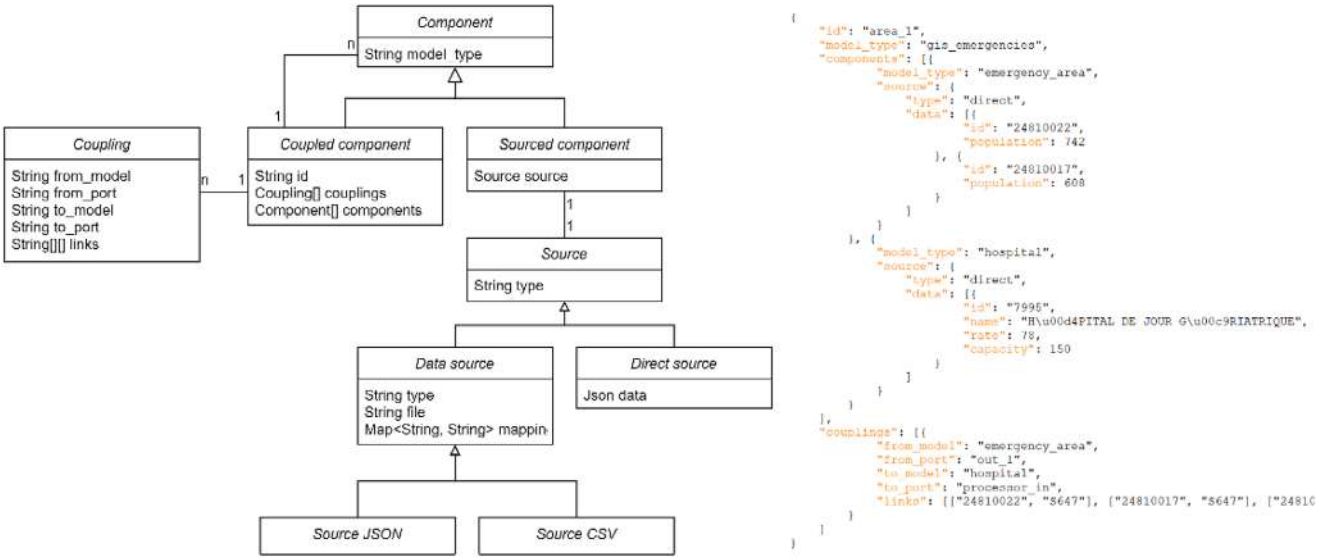

Figure 13. Conceptual data model of a simulation scenario (left) and a scenario file excerpt (right).

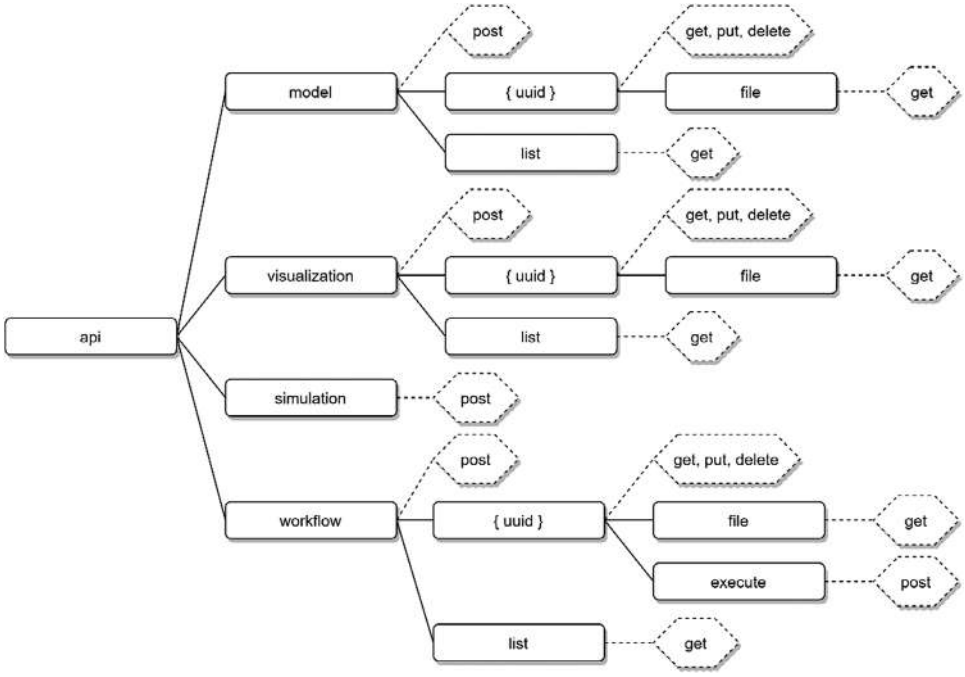

Figure 14. Route diagram for the simulation lifecycle management web API.

destination models and ports as well as pairs of identifiers for each origin and destination component.

Each model type used in a scenario must be coded following the Cadmium pattern (i.e., models must inherit from the appropriate base classes and implement a series of properties and methods representing the different elements of a DEVS model such as state, internal transition, external transition, etc.) Each model must also be registered with the simulator and added to a library of models. This involves assigning a string unique identifier to the model class so that it can be recuperated automatically. Once a model is tested and integrated into a local version of the library, adding it to the server-side library is simple, it is a matter of uploading the code and compiling it. Although straightforward, it is a manual process and therefore, error prone. For this reason, we are exploring a process where the library would be automatically updated and compiled from an online repository such as GitHub.

To process a scenario file into a model ready to simulate, we implemented a factory, a well-known software development pattern. The factory relies on a hashmap that associates model unique identifiers to class definitions. At runtime, the factory retrieves a model definition using the string identifier provided and creates multiple instances of it as specified in the scenario file. Once the model is fully built, the system can simulate it which generates a set of simulation results. The results are then packaged as an archive and sent back to the user or published for visualization.

5.3. Middleware Services

The central component of the web architecture for geospatial simulation is a REST based service architecture that enables the BPs detailed in section 4. It provides the services required to "glue" the four BPs together. The architecture illustrated in **Figure 14** was implemented using Spring Boot, a popular Java framework that facilitates the development of web services through dependency injection, among other features. The services, once developed, were published on an Apache Tomcat server hosted on the Compute Canada cloud provided by the Digital Research Alliance of Canada. Each branch of the web API roughly corresponds to one of the four BPs. The *model* branch allows users to document models using metadata (section 4.3.1), the *workflow* branch allows users to publish MCWs (section 4.3.2), the *simulation* branch provides simulation as a service capability (section 4.3.3) and the *visualization* branch is used to publish simulation results and visualization configurations (section 4.3.4). Each branch provides a way for users to create resources on the server and expose them publicly for discovery and consumption.

The *model*, *workflow* and *visualization* branches display a common behavior. Users can publish (create) new resources by sending a POST request to the appropriate endpoint with the correct payload. The payload typically consists of the file itself and additional descriptive data for the file (author, date created, etc.) Whenever a file is published, the system generates a UUID to identify it, stores it on the server's file system and updates an index that lists all files of that type. Users can also read, update, or delete a resource by respectively sending a GET, PUT or DELETE request to the REST endpoint of a resource identified by its UUID (i.e., *api/{branch}/{uuid}*) and providing the required payload. For each of these branches, users can also send a GET request to the *file* endpoint of a resource to download the file. For example, sending a GET request to the *api/workflow/{uuid}* endpoint will return the summary descriptive data for the corresponding workflow definition file while sending a GET request to the *api/workflow/{uuid}/file* endpoint will download the workflow

file itself. Sending a GET request to the *list* endpoint of a branch, for example *api/model/list* will display the contents of the index file for that branch. It will list all the models available in the model repository.

5.4. Front-End Tools for Dissemination

5.4.1. Front-End Library for Simulation-Based Web Applications

In the field of simulation, dissemination of models and results is often limited to the publication of analytical charts, tables, statistics, or other static analyses issued from the simulation results. Advanced data visualization applications allow end-users to interact with simulation models or results in a manner that is tailored to their specific needs. The web services in the architecture provide access to the backend resources that these applications require. However, non-expert end users do not interact directly with web services. They require a front-end tailored to facilitate interaction. Building customized front-end applications is time and effort consuming. To mitigate this, the architecture includes a library of front-end tools that abstracts some of the complexity involved in interacting with the architecture.

The library contains several useful data structures. The *configuration* package provides several classes to hold map visualization configuration, to handle playback options, layout and size for the visualization, style to be applied to models, etc. It contains functions to facilitate access to each configuration parameter. This data structure will be explained in the following section. The *simulation* package contains classes that hold the simulation results and the model metadata as discussed in section 4.2. These provide functions to access the structural elements of a model (components, ports, couplings, etc.) Optimized access to model components is important when animating the simulation trace since these functions are called often enough to impact on performance. There are also functions to control the flow of a simulation animation which emits events when moving forward or backward. This allows other components to listen to changes in the state of the *simulation* and react accordingly. The visualization engine for example rely on these events to redraw itself when the simulation state changes. Finally, the *metadata* structure contains all elements of the metadata specification.

The library also contains various utility classes to help users accomplish different tasks when building web applications. The tools package contains a series of static utility classes provided for the convenience of developers. *Dom* provides document object model (DOM) manipulation functions, *Net* has functions used to send web requests to the web services of the architecture and *Zip* is used to manipulate compressed files on the front-end. The UI package contains simple user interface elements such as a *box-input* element that allows users to upload files, a *color picker* element to let users choose colors used to draw simulation results and a *popup* element to show dialog windows to users. There are also more complex widgets such as the *palette* widget which allows users to assign styles to Cell-DEVS visualizations, the *settings* widget used to configure visualization options and the *linker* widget used to associate diagram to structural elements of a model. There are also non-GUI elements such as the *chunk reader* which can be used to read a simulation log file by chunks and the *parser* which reads simulation results.

The front-end tools also contain three main visualization components that can be used to visualize different types of DEVS simulations. The *diagram* class is used to visualize standard DEVS models using SVG to display and animate messages as they travel through models and ports (right-most image in **Figure 15** shows an example). The *grid* class is used to visualize Cell-DEVS models using a canvas HTML element which behaves in a manner similar to bitmap images (left-most image in Figure 15 shows an example). Each cell has a red, green, blue value assigned according to the state of the cell using a color classification schema. The *gis* class is used to draw a regular DEVS model on a map. Due to its central role in the architecture, we discuss it in greater detail in the next section.

5.4.2. Map Based Visualization

Map-based visualization can be used to disseminate geospatial simulation results. Web developers can develop interactive maps that display the results of geospatial simulations executed through the architecture. In this implementation, we use the OpenLayers API to draw the map. It is a long-standing, well-established open-sourced library for the development of interactive maps on the web. Other libraries could have been used, for example, MapBox GL js, ArcGIS js API, or Leaflet but we settled on OpenLayers due to it being open-sourced and our familiarity with it. Using the library, all that is required is to create an instance of the *gis* class and initialize it by providing a visualization configuration that indicates the geospatial data files to display, the simulation results to associate to them and, the map styles to use. A conceptual representation of the visualization configuration file is shown in the appendix.

A visualization configuration must first specify a *basemap* which is shown underneath the vector data and usually contains hydrography, basic road network, land use information, etc. It must also specify a *center* and *zoom* level to set the initial extent of the map. It also contains several data *layers* to represent the geospatial features in the simulation model (for example, roads, buildings, administrative areas, etc.) Each *layer* has the following fields:

- A unique *id* so the layer can be uniquely referenced in an application.
- A geometric primitive *type* (point, line polygon) used to determine the appropriate style object used to draw the layer.
- A *file* path to the geojson file containing the geometries for the layer. This file is often generated by the model composition workflows.
- A list of attribute *fields* that will be loaded from the geojson file and included with the layer in the application.
- A *label* used to name the layer in a user-friendly manner in the application.
- A *join* field used to link the geospatial layer and the simulation results.
- Each layer is also associated to a default style object used to draw the layer when it is not associated with simulation results.

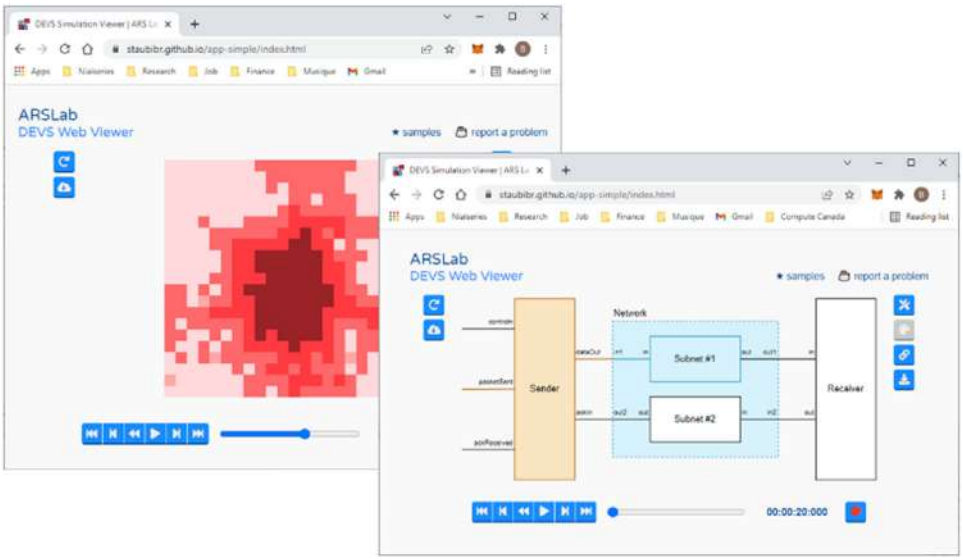

Figure 15. A Cell-DEVS grid visualization (left) and a DEVS diagram visualization (right).

Similarly, the visualization can contain multiple *variables*. Each *variable* represents a value from the model state messages. A *variable* has a *name* used for labeling purposes and must be associated to the *layer* representing the corresponding model and a *style* to be used when drawing the *variable* on the map. Our development library currently supports several styling options for geometric primitives: points, lines, and polygons. The table below summarizes the options (**Table 4**).

A good way to intuitively display changes to a model's state to vary visual variables (size, stroke, fill, etc.) as the value of a model's state variable changes. Each model state variable can be displayed using dynamic visual variables. Users can specify how to vary the *scale*, *radius*, *stroke* or *fill* of a geometry's symbol according to the state of a model. To do this, they can use the *bucket scale*, *bucket radius*, *bucket stroke* and *bucket fill* classes. Each of these style option allows users to define how to vary the style according to the model state variable. They can use equivalent classification where the spread of the values for the variable is divided in equal classes. They can also use quantile classification which divides the spread of values in classes with the same number of values in each. The number of classes is defined either by the number of colors provided (for *fill* and *stroke*) or the *classes* value (for *radius* and *scale*). In the case of *stroke* and *fill* styles, the fill colors and stroke widths for each class can be provided as lists of color codes and width values, respectively. For *radius* and *scale*, minimum and maximum values are provided, and the system derives the values for each class to be used.

6. Data Driven Geospatial Simulation Environments

In this section, we discuss a case study where the environment described in the previous sections was used to build a web-based simulation application. This is an artificial scenario that is meant

Table 4. Styling options by geometric primitive.

style	geometries	property	example
scale	point	size	
radius	point	size	
stroke	point, line polygon	width	
stroke	point, line polygon	color	
fill	point, polygon	color	

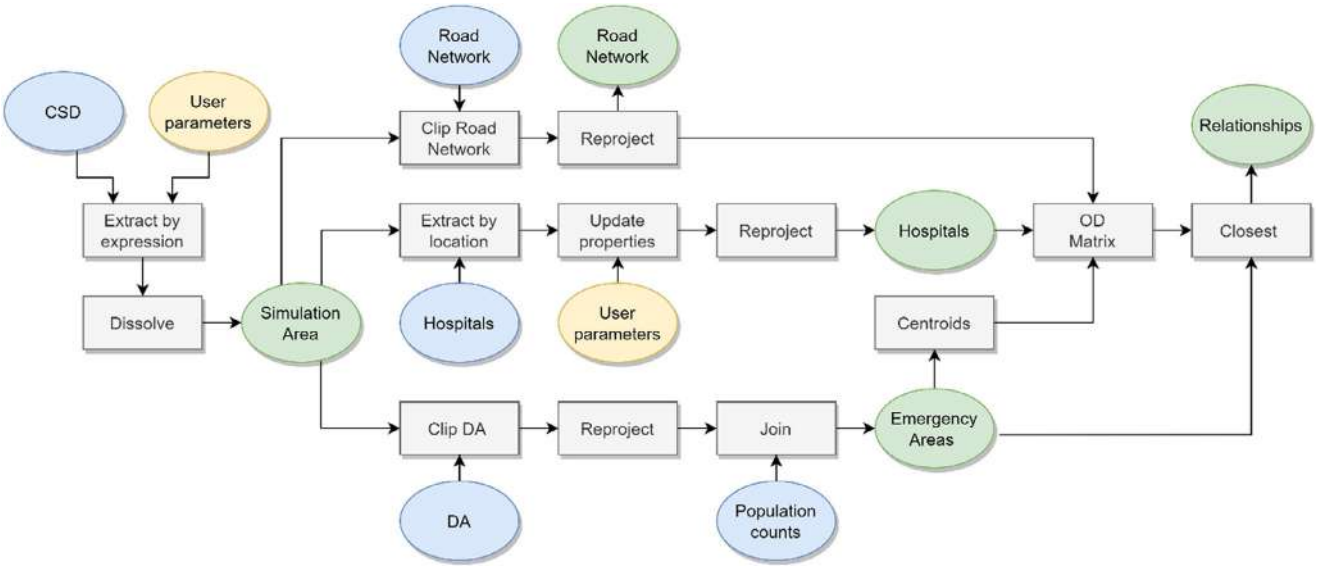

Figure 16. Workflow used in the case study (blue: data inputs, yellow: user parameters, green: outputs, grey: operations).

to highlight how each component of the environment could be employed in an organization to allow SMEs to conduct their own simulation experiments and disseminate the results. The scenario focuses on the management of medical emergencies by geographic areas. It allows end users to create their own simulation models and visualize results through an intuitive, map-based user interface. The general concept of the scenario is to prepare a model that will generate and process emergencies according to the spatial characteristics of the region selected by the end user. A more complete video of this use case scenario is available.[68]

The scenario relies on two atomic model components: an *emergency area* model generates emergencies that are mapped to geographic areas and a *hospital* model that processes the emergencies. The number of emergencies generated by each *area* is based on a simple formula that considers a base emergency rate and the area's population count to determine a maximum number of emergencies that can occur. The number of emergencies emitted every 24 hours by an *area* is randomized between one and the maximum value calculated. Emergencies are sent to the first nearest hospital. If the hospital is under capacity, it will accept the emergencies. Otherwise, it will reject them, and the *area* will send it to the next closest *hospital*. It will repeat this once more if needed and, if it is still rejected, will register it as a failed emergency (i.e., death). *Hospitals* process emergencies daily at a rate defined by their parameters. This model can be considered as a "toy model". Preparing a truly representative emergency management model is beyond the scope of this work. Following the business process presented earlier, the two models were coded for the Cadmium simulator, and uploaded to the environment along with their metadata.

We then prepared the model composition workflow shown in **Figure 16**. It consists of several spatial analysis and attribute data operations. It uses several data sources from Statistics Canada to prepare a model: Canadian census subdivisions (CSD) and dissemination areas (DA), the Open Database of Healthcare Facilities (ODHF), the National Road Network (NRN) and, the Census Profile 2016. The workflow is parameterized to accept a list of CSD unique identifiers (a CSD corresponds roughly to a municipality in more population dense areas) and a list of hospital parameters that overwrite their default capacity and rate values. This allows users to designate a simulation area and test different scenarios for hospitals.

The first step of the workflow is to dissolve (merge) the selected CSDs into a single polygon that represents the simulation area. The DA, ODHF and NRN datasets are then constrained to the simulation area polygon through a clip or extract by location operation. For each DA, the workflow joins the population counts from the Census Profile. Next the workflow overrides ODHF hospital attributes for each set of user-defined hospital attributes provided. Centroids for DAs are computed and used to calculate an origin-destination road distance matrix with each ODHF hospital in the simulation area based on the NRN. This requires calculating the driving distance along the road network between each permutation of DA centroids and hospital points. This is the bottleneck of the workflow since a shortest distance algorithm, a notoriously computationally expensive operation, must be executed for each pair of points. Using the resulting origin-destination matrix, the workflow can determine the three closest hospitals for each DA. The unique identifier for each closest hospital is stored on each DA as separate fields (i.e., three fields are added to the DA layer, *hospital_1*, *hospital_2* and *hospital_3*). Finally, the workflow outputs the DA, hospital and NRN data then constructs the scenario for the simulator. The scenario dictates that *emergency area* and *hospital* type models should be mapped onto the resulting DAs and ODHF hospitals, respectively. It also specifies that each *emergency area* should be connected to the *hospitals* identified using the origin-destination matrix.

The workflow was then published to be subsequently used as a service. Then, we built a web application using the front-end toolbox that allows users to define their own experiments using the model and generate different. Users can select a simulation area and override hospital attributes by interacting with

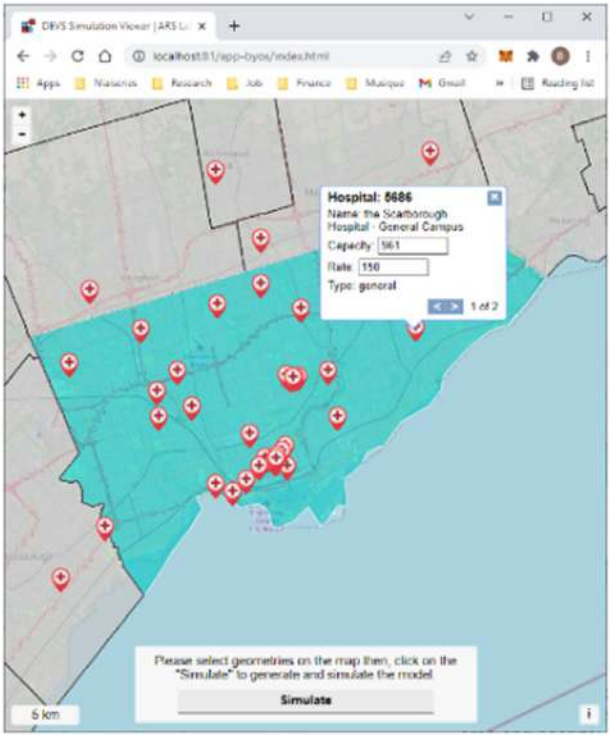

Figure 17. Users select a simulation area and set experiment parameters.

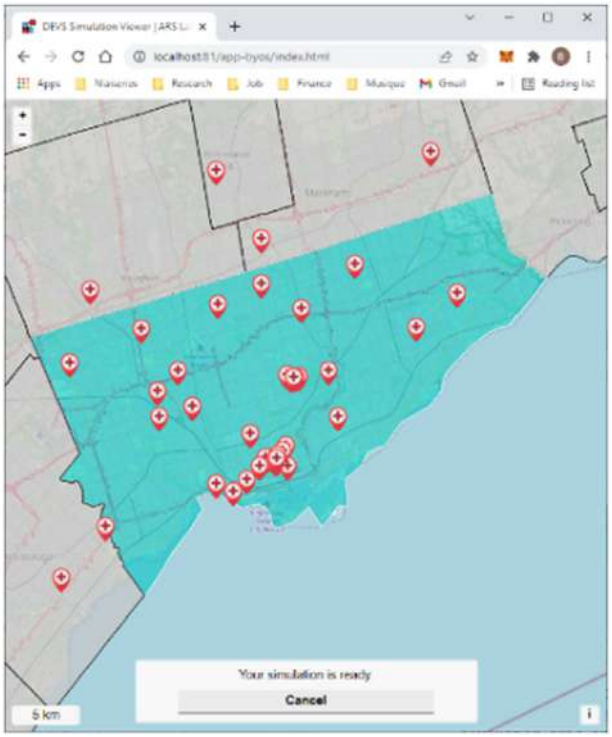

Figure 18. The experiment is simulated on the server.

a map. To do so, they must click on CSDs to select them and hospitals to modify their values. In **Figure** 17, a user has selected Toronto as a simulation area and overrode the parameters for the Scarborough Centenary Hospital. Once the user has defined an experiment, they click the simulate button which first calls the execute endpoint for the published workflow service (./api/workflow/{uuid}/execute). Once executed, the application calls the simulation endpoint (./api/simulation) and sends the scenario file resulting from the workflow execution. After the simulation is complete, the system publishes a visualization to the server by calling the visualization endpoint (./api/visualization) and providing the simulation and workflow outputs. Once this process is complete, the application shows the visualization as shown in **Figures** 18 and 19.

7. Conclusion and Future Work

The current modeling and simulation ecosystem, particularly with regard to the DEVS methodology, lacks a comprehensive framework to properly support non-expert users in complex simulation projects. The four web-based BP and the implemented simulation environment introduced in this paper support the simulation lifecycle. The first BP allows users to prepare models and document them thoroughly using a model metadata specification. The second allows users remotely simulate models by providing simulation scenarios. The third provides the tools required for users to prepare visualization platforms and publish visualizations for their simulation results. The fourth one is an extension of the second. It allows users to automatically build spatial simulation models through model composition workflows that map model components on geospatial data using geometric and topological characteristics. The architecture, through its web services, can act as the "glue" between the different steps of the simulation lifecycle. It also provides a way to decompose the lifecycle into parts that can be tackled by individuals with different skillsets. We limit the role of modelers and web developers to a few very specific activities in the BPs. The SME assumes most

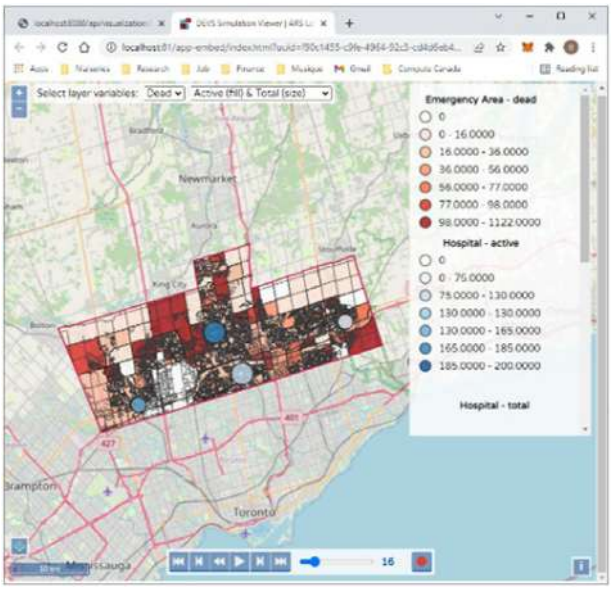

Figure 19. A visualization of the resulting simulation.

of the work composing models, documenting them, conducting experiments, preparing visualizations, and analyzing results.

Further research is required to study specific aspects of the architecture to further reduce the burden on the modelers, analysts, and developers or to improve user experience. For example, determining better mechanisms to transfer simulation results from the back end to the front end and researching better ways of processing that data on the front-end. Web workers are now well supported in modern browsers and enable parallel threads in browsers. These could be used to efficiently prepare the simulation data for rendering as it arrives from the back end. WebAssembly is a new type of code that can be run in web browsers. It is a low-level assembly-like language that runs at near-native performance. These tools could improve user experience by reducing transfer and processing delays. It would also allow the visualization of much larger simulation results. Another interesting research avenue is to evaluate the usefulness of simulation analytics as a service to easily extract insights from simulation results. A suite of services could be designed to query results and compile statistics that are ready to visualize as analytics such as conventional charts (line, bar, pie, etc.) or more elaborate ones (Sankey diagrams, sunburst, heatmaps, etc.)

DEVS based simulation, for all its theoretical advantages (modularity and hierarchy), remains marginally used in the simulation industry due to its complexity. Domain specific, single use simulators remain more popular largely because they are tailored to their domain of application. This provides avenues to simplify their usage, often by tightly coupling simulator and model or by leaning heavily on parametric models. On the other hand, it makes it difficult to integrate in multi-disciplinary models and therefore, condemns them to remain in their disciplinary silo. Through a better simulation environment, it is possible to reduce the barrier to entry for DEVS so that it can become a common formalism to implement multi-disciplinary models in a collaborative manner.

Conflict of Interest

The authors declare no conflict of interest.

Data Availability Statement

Research data are not shared.

Keywords

geosimulation, geospatial simulation, simulation environment, simulation lifecycle

Received: February 7, 2024
Revised: May 17, 2024
Published online: June 19, 2024

[1] J. O. Henriksen, *Journal of Simulation* **2008**, *2*, 3.
[2] M. Batty, P. M. Torrens, *J Geog* **2001**.
[3] D. B. Crawley, L. K. Lawrie, F. C. Winkelmann, W. F. Buhl, Y. J. Huang, C. O. Pedersen, R. K. Strand, R. J. Liesen, D. E. Fisher, M. J. Witte, J. Glazer, *Energy Build.* **2001**, *33*, 319.
[4] EnergyPlus, EnergyPlus | EnergyPlus **2021**, https://energyplus.net/ (accessed April 23, 2022).
[5] *OpenFlows FLOOD Modeling Software, Bentley Systems Incorporated*, **2022**, https://www.bentley.com/en/products/product-line/hydraulics-and-hydrology-software/openflows-flood (accessed: April 2022).
[6] DIALux, DIALux - DIAL **2021**, https://www.dial.de/en/dialux/ (accessed March 5, 2021).
[7] D. Fuller, A. McNeil, Radiance — Radsite **2017**, https://www.radiance-online.org/ (accessed: March 2021).
[8] C. Gomes, C. Thule, D. Broman, P. G. Larsen, H. Vangheluwe, *ACM Comput. Surv.* **2018**, *51*, 1.
[9] A. Crooks, C. Castle, M. Batty, *Comput Environ Urban Syst* **2008**, *32*, 417.
[10] X. Meng, M. Zhang, J. Wen, S. Du, H. Xu, L. Wang, Y. Yang, *Sustain* **2019**, *11*.
[11] L. Liu, Y. Liu, X. Wang, D. Yu, K. Liu, H. Huang, G. Hu, *Nat. Hazards Earth Syst. Sci.* **2015**, *15*, 381.
[12] S. Jin, Y. Yan, X. Jiang, *IOP Conf. Ser. Earth Environ. Sci.* **2017**, 100.
[13] I. Kaur, A. Mentrelli, F. Bosseur, J. B. Filippi, G. Pagnini, *Commun Nonlinear Sci Numer Simul* **2016**, *39*, 300.
[14] O. Jellouli, A. Bernoussi, M. Mâatouk, M. Amharref, *Math. Comput. Model. Dyn. Syst.* **2016**, *22*, 493.
[15] J. B. Filippi, F. Bosseur, C. Mari, C. Lac, *Atmosphere* **2018**, *9*, 218.
[16] X. Zhang, *China, J. Geogr. Inf. Syst. China. J. Geogr. Inf. Syst.* **2016**, *8*, 317.
[17] R. Wang, Y. Murayama, *ISPRS Int. J. Geo-Information.* **2017**, *6*, 150.
[18] M. Batty, Y. Xie, Z. Sun, *Comput Environ Urban Syst* **1999**, *23*, 205.
[19] Y. Huang, A. Verbraeck, M. D. Seck, *Simulation* **2015**, *91*, 1027.
[20] M. Zhao, X. Yao, J. Sun, S. Zhang, J. Bai, *IEEE trans Intell Transp Syst* **2019**, *20*, 323.
[21] H. L. M. Vangheluwe, in *CACSD. IEEE Int. Symp. Comput. Control Syst. Des.*, **2000**, (Cat. No.00TH8537), IEEE, USA, pp. 129–134.
[22] RTSync Corp., Better Predictions, Smarter Decisions | RTSync **2021**, http://www.rtsync.com/pages/products/ms4me.html (accessed: April 2022).
[23] S. J. E. Taylor, in Proc. 2011 Winter Simul. Conf., IEEE, **2011**, pp. 2904–2908.
[24] B. P. Zeigler, H. Praehofer, T. G. Kim, *Theory of Modeling and Simulation*, 2nd ed., Elsevier, San Diego, CA, USA, **2000**.
[25] G. A. Wainer, *Discrete-Event Modeling and Simulation : A Practitioner's Approach*, Taylor & Francis, Boca Raton, FL **2009**.
[26] H. L. Vangheluwe, J. De Lara, P. J. Mosterman, *AI, Simul. Plan. High Auton. Syst.* **2002**, 9.
[27] G. A. Wainer, N. Giambiasi, *Discret Event Dyn Syst* **2002**, *12*, 135.
[28] G. Wainer, K. Al-Zoubi, D. Hill, S. Mittal, J. Martín, H. Sarjoughian, L. Touraille, M. Traoré, B. Zeigler, in *Discret. Model. Simul.*, CRC Press, USA, **2010**, pp. 393–425.
[29] R. M. Fujimoto, *ACM Trans. Model. Comput. Simul.* **2016**, *26*, 1.
[30] S. J. E. Taylor, A. Khan, K. L. Morse, A. Tolk, L. Yilmaz, J. Zander, P. J. Mosterman, *Simulation* **2015**, *91*, 648.
[31] S. J. E. Taylor, R. M. Fujimoto, E. H. Page, P. A. Fishwick, A. M. Uhrmacher, G. A. Wainer, in Proc. Title Proc. 2012 Winter Simul. Conf., IEEE, **2012**: pp. 1–15.
[32] S. J. E. Taylor, A. Khan, K. L. Morse, A. Tolk, L. Yilmaz, J. Zander, in Proc. Symp. Theory Model. Simulation, **2013**, pp. 1–8.
[33] L. Yilmaz, S. J. E. Taylor, R. M. Fujimoto, F. Darema, *Winter Simul. Conf. Savannah, Georg.* **2014**, *2014*, 2797.
[34] M. F. Goodchild, *Annu Rev Environ Resour* **2003**, *28*, 493.
[35] C. R. Dietrich, G. N. Newsam, *Water Resour Res* **1993**, *29*, 2861.
[36] M. Zapatero, R. Castro, G. A. Wainer, M. Houssein, Proc. 2011 Winter Simul. Conf **2011**, pp. 997–1009.
[37] M. F. Goodchild, *Dialogues Hum Geogr* **2013**, *3*, 280.

[38] A. C. Robinson, U. Demšar, A. B. Moore, A. Buckley, B. Jiang, K. Field, M.-J. Kraak, S. P. Camboim, C. R. Sluter, *Int. J. Cartogr.* **2017**, *3*, 32.
[39] S. Li, S. Dragicevic, F. A. Castro, M. Sester, S. Winter, A. Coltekin, C. Pettit, B. Jiang, J. Haworth, A. Stein, T. Cheng, *ISPRS J Photogramm Remote Sens* **2016**, *115*, 119.
[40] D. Laney, *META Delta* **2001**, *949*, 4.
[41] S. Sutharan, *Eval Rev* **2014**, *41*, 4.
[42] A. Sawhney, S. M. AbouRizk, D. W. Halpin, *Can. J. Civ. Eng.* **1998**, *25*, 16.
[43] H. S. Sarjoughian, V. Elamvazhuthi, Second Int. ICST Conf. Simul. Tools Tech, ICST, USA, **2009**, pp. 1–9.
[44] R. Goldstein, S. Breslav, A. Khan, in Proc. 2016 Spring Simul. Multiconference – TMS/DEVS Symp. Theory Model. Simulation, TMS/DEVS, USA, **2016**, *2016*.
[45] F. Bergero, E. Kofman, *Simulation* **2011**, *87*, 113.
[46] M. Bonaventura, G. A. Wainer, R. Castro, *Simulation* **2013**, *89*, 4.
[47] K. Al-Zoubi, G. Wainer, *J Parallel Distrib Comput* **2013**, *73*, 580.
[48] R. Goldstein, S. Breslav, A. Khan, *Simulation* **2018**, *94*, 301.
[49] C. M. Macal, *Simulation* **2001**, *77*, 90.
[50] A. J. Collins, D. K. Ball, J. Romberger, *MODSIM World* **2014**, 1.
[51] J. Kuljis, R. J. Paul, C. Chen, *Simulation* **2001**, *77*, 141.
[52] S. Dufour-Kowalski, B. Courbaud, P. Dreyfus, C. Meredieu, F. de Coligny, *Ann For Sci* **2012**, *69*, 221.
[53] C. Zoellner, M. A. Al-Mamun, Y. Grohn, P. Jackson, R. Worobo, *Appl. Environ. Microbiol.* **2018**, *84*, e00813.
[54] https://pro.arcgis.com/en/pro-app/2.8/help/analysis/geoprocessing/modelbuilder/what-is-modelbuilder-.htm (accessed: April 2022).
[55] QGIS, https://docs.qgis.org/3.16/en/docs/user_manual/processing/modeler.html (accessed: April 2022).
[56] Natural Resources Canada, Geospatial Standards and Operational Policies, **2019**, https://www.nrcan.gc.ca/earth-sciences/geomatics/canadas-spatial-data-infrastructure/8902 (accessed: July 2022).
[57] Open Geospatial Consortium, OGC Standards | OGC **2022**, https://www.ogc.org/docs/is (accessed: February 2022).
[58] ESRI, ArcGIS Web AppBuilder **2022**, https://developers.arcgis.com/web-appbuilder/guide/xt-welcome.htm (accessed: July 2022).
[59] O. Balci, *Simulation* **2012**, *88*, 870.
[60] R. G. Sargent, Proc. 2010 Winter Simul. Conf., IEEE, USA, **2010**, pp. 166–183.
[61] P. Benjamin, M. Patki, R. Mayer, in Proc. 2006 Winter Simul. Conf., IEEE, **2006**, pp. 1151–1159.
[62] M. L. Loper, *Modeling and Simulation in the Systems Engineering Life Cycle*, Springer London, London, **2015**.
[63] J. Byrne, P. Byrne, D. Carvalho e Ferreira, A. M. Ivers, in Proc. Winter Simul. Conf., IEEE, USA, **2014**, *2014*, pp. 2738–2749.
[64] B. St-Aubin, J. Menard, G. Wainer, in Proc. 2021 Annu. Model. Simul. Conf. ANNSIM, IEEE, USA, **2021**, *2021*, pp. 1–12.
[65] Full metadata documentation: https://staubibr-stable.github.io/doc-meta.
[66] B. St-Aubin, G. Wainer, in IEEE/ACM 26th Int. Symp. Distrib. Simul. Real Time Appl, IEEE, Alès, France **2022**, pp. 160–163.
[67] A. Pourshahid, D. Amyot, L. Peyton, S. Ghanavati, P. Chen, M. Weiss, A. J. Forster, *Electron. Commer. Res.* **2009**, *9*, 269.
[68] Use case scenario video: https://youtu.be/VAtNItlEVh0.