A Domain Specific Language to Simplify the Creation of Large Scale Federated Model Sets

Zachary T. Reinhart, Sunil Suram, Kenneth M. Bryden

A Domain Specific Language to Simplify the Creation of Large Scale Federated Model Sets

Zachary T. Reinhart[1], Sunil Suram[1], Kenneth M. Bryden[1*]

[1]Iowa State University, Ames, IA USA
{zrein, sunils, kmbryden}@iastate.edu

Abstract. This paper presents an attempt to address the challenge of modeling complex systems in which people, energy, and the environment meet. This challenge is met by developing a simple domain specific language for buiding systems models in a federated modeling environment. The language and its support infrastructure are designed for simplicity and ease of use. This language is demonstrated using a thermodynamic model of a biomass cookstove for the developing world as an example, and the use of the tools described in this paper to further extend that cookstove model into an end-to-end design tool for cookstoves and other energy systems for the developing world is discussed.

Keywords: federated modeling, integrated modeling, complex systems

1 Introduction

Sustainability is the challenge of the future. Understanding systems in which people, energy, and the environment intersect will be critical in meeting this challenge. Modeling these large-scale complex systems will be key in gaining this understanding. However, creating these complex, integrated models is challenging. Traditionally, models for each of these domains have been built independently by researchers working in different, often unconnected fields. Integrating these disparate models to create a holistic systems model and that accurately represents these systems in which people, energy, and the environment intersect is difficult, time consuming, and expensive with conventional approaches. One approach to this model integration challenge is the development of federated model sets [1].

2 Background

Integrated modeling has largely been developed by the environmental modeling community as an answer to the challenge of modeling large, complex systems. The goal of integrated modeling is to take a variety of components, be they mathematical models, databases, or other data sources, and combine them together into an integrated systems model [2]. A model integration framework is often utilized to achieve this integration. Various open- and closed-source model integration frameworks have been created,

each with slightly different goals. For example, SCIRun [3] and OpenDX [4] have a visualization focus, while others focus on providing a component-based approach to model integration, such as the Object Modeling System (OMS) [5], The Invisible Modeling Environment (TIME) [6], and the Community Surface Dynamics Modeling System (CSDMS) [7]. Others, such as VE-Suite [8], seek to be general purpose model integration tools. Closed source packages such as Matlab™, Simulink™, [9] and Aspen Plus™ [10] can be considered as model integration tools, though they tend to be less flexible and are often focused on a specific domain, such as process plant simulation in the case of Aspen Plus.

The drawback of most integrated modeling frameworks is that they require significant effort in project management and software development to create the desired integrated model, and in some cases they lack the ability to utilize and manage large-scale, high-fidelity models needed for detailed analysis. Typically, full access to the source code of all constituent models is required, and the system builder or team must develop a global ontology that connects all models. This approach tends to produce a well-integrated product, but the challenges of management, software development, and global ontology development can become unwieldy when modeling large systems with myriad component models.

There have been attempts to address this shortcoming. The CSDMS implements a component-based programming model that is intended to minimize invasive changes to component models. This is achieved through a pair of software interfaces, the Basic Model Interface (BMI) and the Component Model Interface (CMI) [7, 11-12]. The BMI is the interface to the component model, and the CMI is the corresponding interface that connects to the integration framework. Once the code of a component model has been modified to implement the BMI, it can be registered as a usable component model within the CSDMS framework [7, 12]. While this approach was originally created for large-scale geological modeling, Antonelli, Bryden, and LeSar [13] have adapted the BMI to link a molecular dynamics simulation with a lattice Boltzmann solver to form a concurrent multi-scale model of fluid flow, demonstrating the applicability of the BMI concept to other types of models and systems.

Another approach is the development of federated model sets [1]. Federated model sets have been envisioned to take advantage of cloud-based independent models and to enable rapid construction of complex system models from these cloud-based independent modeling components. Specifically, in a federated model set, component models are implemented as independent information services with self-documenting interfaces. Information transfer between models is brokered, allowing for the creation of peer-to-peer ontologies instead of a global ontology for the integrated model. Component models can maintain a high level of autonomy and independence while still being members of a larger system model. This independence streamlines component and system model development and project management through a clear separation of roles.

In a federated modeling system, there are three primary user roles: the component modeler, the system builder, and the end user. The component modeler provides the constituent component models that can be utilized to build systems models. The system builder selects component models and composes a systems model. The end user utilizes the systems model to answer questions or gain insight into the modeled system. While

these roles may overlap, they can be totally independent. The component modeler need not coordinate development with the system builder; they need only to provide their model in the required format with proper library entries and message contracts for inclusion in the federated modeling system. System builders may choose freely from these component models to construct systems models.

Suram, MacCarty, and Bryden [14] have implemented a proof-of-concept federated model set based on a heat-transfer model of a small, shielded-fire cookstove for the developing world developed by MacCarty and Bryden [15]. This model was decomposed into seven independent information services, each implemented as a stateless web microservice [14,16]. These microservices fulfill requests for computation; each request must contain all information necessary to fulfill the request, and after the results of the computation are returned, all state is discarded. As currently implemented by Suram, MacCarty, and Bryden [14], a federated modeling system has the following characteristics.

- Models as stateless microservices
- Message broker for communication between microservices and infrastructure services
- A programmable federation management system (FMS)
- Simple message contracts
- Web-based front-end server

These characteristics can be leveraged to create a system model that is accessible as a dynamic web service, but to do so the system builder must write code in Java that coordinates with the FMS to route data through the selected system of models. Any translation that is necessary between adjacent models must be handled manually by the system builder. Creating, modifying, and utilizing a federated modeling system as it exists now requires the system builder to be an expert in cloud computing and app development. However, by creating a domain specific language (DSL) and supporting infrastructure, we can substantially simplify the process of creating a federated model set, making it approachable for many more users.

A DSL is a custom programming language designed for a specific problem domain, built using the idioms and concepts of that domain [17]. The advantages of DSLs over general purpose languages include increased productivity and more readable, maintainable, and reusable code, among others. One of the chief advantages of a DSL for complex system modeling is that it enables clear communication with domain experts [17, 18]. Since a DSL natively uses the concepts and idioms of the domain in question, code in that language is easily read, understood, and written by domain experts even with limited programming experience [17,18]. DSLs strive for fluent translation between mental models and code. This is a significant advantage when building models of complex systems, and is our primary motivation in designing a DSL for federated modeling.

3 Language Discussion

To create a working federated model set, four pieces of information are necessary:

Input blocks:

```
constants <system name>
 <constant 1>
 <constant 2>
 .
 .
 <constant n>
end constants
```

```
inputs <system name>
 <input 1>
 <input 2>
 .
 .
 <input n>
end inputs
```

Output block:

```
outputs <system name>
 <output 1>
 <output 2>
 .
 .
 <output n>
end outputs
```

System block:

```
system <system name> from library <library address>
 <model 1 name>
 <model 2 name>
 .
 .
 <model n name>
end system
```

Fig. 1. Brief syntax for code blocks.

- A list of constituent models,
- A list of connections between models,
- A list of input parameters,
- A list of desired output variables.

The desired component models and system level inputs and outputs are usually straightforward to define. The connections between models are generally much more complex to define fully. However, many of these connections can be determined automatically by the DSL tools by leveraging the self-describing interfaces of the constituent models along with supporting infrastructure. Therefore, the focus of the DSL is to allow the system builder to specify the constituent models, inputs, and outputs in a straightforward manner, with less emphasis on fully defining detailed connections between models. By removing this burden of complexity, the syntax of the language can be made readable and intuitive, even to those users who have limited programming experience. To that end, we have laid out a simple syntax that consists of three types of text blocks: input blocks, output blocks, and system blocks, illustrated in Fig. 1. These blocks may appear in any order in a system script. The input and output blocks do not need to be placed ahead of the system block, for example. The syntax avoids the use of brackets, braces, and other symbols whose purpose may not be obvious to new users.

There are two types of input blocks, user inputs and constants. User inputs are passed in by the end users of a model. Since a federated model set is implemented as a web service, inputs will be uploaded to a specific web location. Constants are specified by the system builder, either in the code they write or in a file. The file location may be specified as a valid uniform resource locator (URL). Output blocks simply specify the

output variables of interest. These outputs will be available for download at a specified URL upon completion of the requested modeling job. The system block contains a list of the models that constitute the system, along with the address of the model library from which these models are drawn. In many cases, these models can just be listed by their unique names as looked up in the component library. They do not need to be listed in any defined order. In cases where it is necessary for the user to define some of the connections between models, the following syntax is used:

```
model_a, model_b -> model_d
```

where `model_a` and `model_b` represent the names of two models that are providing inputs to a third model, named `model_d`. This syntax could be extended to allow the system builder to specify specific variables from source and destination models if needed.

To maintain this level of simplicity, supporting software infrastructure within the federated modeling environment is required. The critical enabling infrastructure services are the model library services and the message contract database. The DSL interpreter will rely on these services to determine possible workflows for solving the specified models [19]. These services and their use will be discussed in the next section.

4 Support Infrastructure Discussion

As mentioned above, the model library service and the message contract database are the key infrastructure services for enabling the simplicity of our DSL. The model library service is straightforward. It is a database that stores information about each component model available in a federated modeling system. Each record in the database corresponds to one component model, and will have information such as a unique name, a network address for the model microservice, a human-readable description, and other relevant data and metadata. Among this data is a full list of required and optional inputs and outputs in the form of message contract references.

All data transfer between models in a federated model set is governed by a system of "snappers" [1] or message contracts [14]. Taken together, these message contracts and their corresponding connections on component models form an extensible system of complex composite data types that enables peer-to-peer ontologies to be negotiated between adjacent communicating models [1]. A message contract takes the form of a document or database record that contains the following fields:

- A globally unique identifier (GUID),
- A list of variables, including name, data type, and description for each,
- A human-readable name,
- A brief description.

This information is used as the basis for a mediated negotiation between two models, with the FMS serving as the mediator. First, the FMS checks that the output of the source model and the input of the destination model are referencing the same message contract GUID. This ensures a basic shared vocabulary of variable names and data

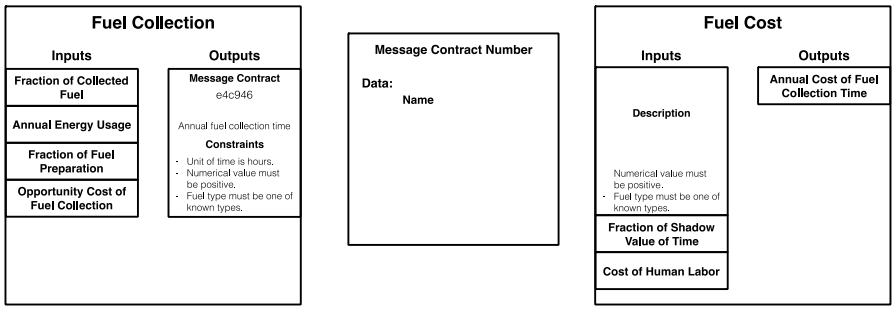

Fig. 2. Illustration of model input/output and matching message contract.

types. The interfaces of the models also incorporate questions and answers, or constraints, that are used to further negotiate data exchange. At minimum, these constraints should define the units for all variables. Simple, scalar unit conversions will be performed by the FMS. In addition, other optional constraints can be imposed by component modelers, such as valid input and output ranges, ranges of accuracy or stability, invalid inputs due to discontinuities, etc. If acceptable answers are found to all required question/answer pairs between two models, a peer-to-peer ontology is defined and the models may communicate freely.

This process is most easily understood by examining an example. Consider the following two component models drawn from MacCarty and Bryden's [20] village energy model, which is being modularized for use in a federated modeling system. The model presented in [20] is an integrated model of the comprehensive energy use of a small African village composed of smaller component models. We will select two models that are used internally, the fuel collection model and the fuel cost model. The fuel collection model takes several inputs and provides an output of the number of annual hours of a given fuel type. The fuel cost model takes the annual hours of a given fuel type as an input, among others, and produces the annual cost of fuel collection time for a given fuel as an output.

We define a message contract to facilitate this connection. The contract is assigned a GUID by which it will be referenced. It defines a data interchange containing two variables: a floating-point variable named hours containing the time quantity, and a string variable named type containing the fuel type. Each model would reference the GUID and have a set of constraints, which would be the unit for time, hours, a constraint that values be positive, and the requirement that the fuel be of a known type. The two models and their message contract are illustrated in Fig. 2.

Once the model library and the message contract system are in place, the DSL tools can utilize the data within both systems to infer connections within model sets and calculate computational workflows to solve model sets. The inputs and outputs of a given component model together with the referenced message contracts allow the system to create a list of possible connections that are valid. That is, for any given input or output of a model, a list of compatible other models can be created. In some cases, there will be a unique set of compatible connections between the component models listed in the

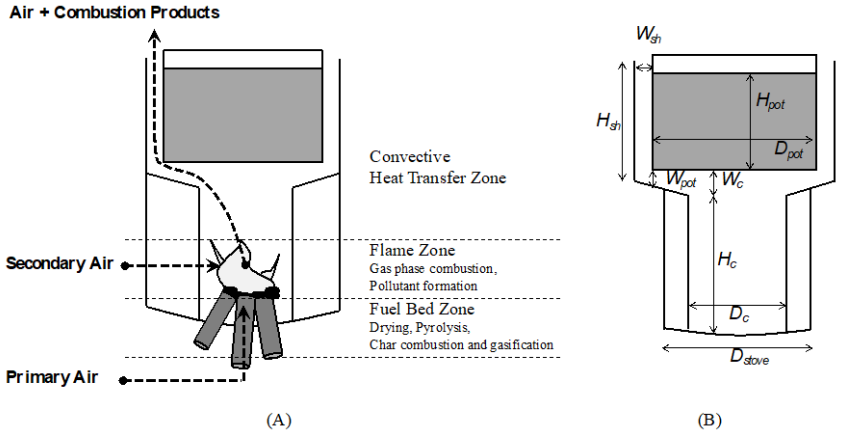

Fig. 3. (A) Biomass cookstove with modeling zones. Adapted from [25]. (B) Geometric stove design parameters. Adapted from [15].

DSL script. In this case, the system builder will not have to explicitly specify any connections between models. When there are multiple valid sets of connections between specified component models, the DSL tools will provide detailed feedback on the points of ambiguity to the system builder. The system builder can then use the syntax defined in the previous section to select one among the possible valid connections where the ambiguity exists. In this way, the DSL tools and the system builder form a partnership, with the system builder offering clarifying information when necessary instead of having to be concerned with the details of every connection between the cloud-based component model microservices in a model set. In the next section, we will explore how all this will work with a real system by applying these tools to the cookstove model used by Suram, MacCarty, and Bryden in [14].

5 Example and Discussion

5.1 Cookstove Model Example

Most of the energy requirements for families in the developing world are currently met by three stone fires [21, 22]. However, three stone fires are inefficient, dangerous, and have significant negative environmental impact [23, 24]. Improve biomass cookstoves have been developed to address the shortcomings of the three stone fire. Such a cookstove typically consists of a combustion chamber, a grate to elevate the fuel for better airflow, and geometric features to direct the flow of hot gases from the fire around the pot, improving heat transfer [15]. Fig. 3 shows a representation of this type of cookstove. Improved biomass cookstoves are key components of energy interventions in the developing world, but there is a dearth of modeling and design tools for such

Table 1. Constituent models of cookstove federated model set, after [14].

Model Name	Description
Mesh	Initializes geometry and allocates variables
Bed	Calculates rate of burning, production of fuel moisture and combustion products, and fuelbed and exit gas
Flame	Calculates rate of burning volatiles and exit gas temperature from the flame zone
Heat Transfer	Calculates exit temperatures through each control
Flow	Calculates velocity and pressure drop through each control volume in the flow path.
L2 norm	Calculates L2 norm between mass flow rates
Convergence	Sets a flag if the system of models has converged

cookstoves. MacCarty and Bryden's model [15] seeks to address this shortage, providing a tool for designers to experiment with materials and geometry while receiving immediate feedback on the thermal efficiency of their notional stove.

This model breaks the cookstove into three coupled zones: the fuel bed zone, the flame zone, and the heat transfer zone, as shown in Fig. 3(A). Solid phase combustion is modeled in the fuel bed zone, gas phase combustion in the flame zone, and heat transfer to the pot and the environment is modeled in the heat transfer zone. Buoyancy driven fluid flow is modeled throughout the coupled zones [15]. This model has been implemented in [14] as seven component model microservices, which are summarized in Table 1.

Originally these models were federated manually with custom Java code. To leverage our proposed domain specific language, model library entries would be created for each model, along with an appropriate system of message contracts. For brevity's sake, the creation of the various message contracts will not be discussed in depth here. The strategy is to create a single message contract for each interacting pair of models. The connections in this system are simple, so only a handful of message contracts are necessary. As discussed earlier, the creation of the model library entries and message contracts is the responsibility of the component model builder, not the system builder. The system builder should find all those things in place and validated for any component model that is listed as ready to use.

With the library entries and message contracts in place, we can now represent this system in the domain specific language. Since message contracts were constructed specifically for the connections necessary in this system, there is only one valid set of possible connections, and the system builder will not need to explicitly specify any connections. Thus, the code representing the system model becomes:

```
inputs stove_model
   H_sh
   W_sh
   W_pot
   W_c
```

```
    H_c
    D_c
    D_stove
    D_pot
    H_pot
end inputs

constants stove_model
    http://10.10.10.10/stovemodel/constants.json
end constants

outputs stove_model
    stove_efficiency
end outputs

system stove_model from library stove_library
    stove_mesh
    bed_model
    flame_model
    heat_model
    flow_model
    l2_norm
    convergence_check
end system
```

In the above code, the inputs block contains the specified input variables. These are geometric parameters for the stove. Briefly, they are the height of the pot shield, H_sh; the width between the pot shield and the pot, W_sh; the width of the gap between the pot and stove at the corner of the pot, W_pot; the distance from the bottom of the pot to the top of the combustion chamber, W_c; the height of the combustion chamber, H_c; the inner and outer diameters of the combustion chamber, D_c and D_stove; the diameter of the pot, D_pot; and the height of the pot, H_pot. These correspond to the geometric parameters show in Fig. 3 (B). The constants block contains a URL that points to a data file in JavaScript Object Notation (JSON) format. This file lists all necessary constants to run the model. The outputs block specifies the output of interest, thermal efficiency. The system block simply lists the seven component models shown in Table 1 that make up the cookstove model.

In a federated modeling environment, this would be the only code necessary to create the cookstove model as a web service. When run, the system would check the code, instantiate model microservices as necessary, create appropriate URLs for inputs and outputs, and provide those addresses along with status messages to the end user. The set of input and output URLs form an application programming interface (API) to the systems model, as used in web app development. An intuitive graphical user interface for modifying geometric stove parameters and visualizing the resulting performance of the stove could be built around this API.

5.2 Extensibility Discussion

The above model allows users to design cookstoves based on thermal efficiency. However, cookstoves for the developing world are intended to improve quality of life through better health outcomes and reduced environmental impact. The current cookstove model does not account for any of these factors, but it could form part of a systems design tool that does. By utilizing models from domain experts in energy systems for the developing world and sustainable agricultural and integrating these models with the cookstove model, a design tool that provides feedback in terms of health outcomes and environmental impact could be created.

MacCarty and Bryden [20] have identified the following factors as significant in prediction the adoption and impact of any energy device in the developing world:

- Desirability – perceived quality and aesthetic benefit
- Disruption – ability of stove to fit existing cooking patterns
- Convenience – ease of use and amount of attention required
- Safety – safety of the device for the user and family

have developed an integrated systems model [20] that accounts for these factors encompasses all the common energy needs in rural villages in the developing world. This model takes as input a set of energy components such as cookstoves, lighting equipment, water heating systems, etc., and performs a Monte Carlo simulation to predict the outcomes for a village, including energy access, environmental effects, health impact, cost, quality of life.

Of major concern when considering any energy intervention in the developing world is environmental impact. While this impact occurs in many forms, including CO_2 and black carbon emissions, deforestation [20] and its impact on local agriculture should also be considered. Muth and Bryden [26] have developed an integrated systems model to predict sustainable levels of agricultural residue removal. This integrated model incorporates several existing models for soil erosion and agricultural databases to create a tool that can predict the sustainability of a given farming practice at a high resolution anywhere in the United States. This model is becoming the standard for modeling sustainable agricultural processes in the United States, and it could be easily adapted for other places in the world.

By modularizing both the village energy model and the sustainable agricultural residue model and implementing them in a federated modeling environment, it becomes simple to modify them and integrate them into a new design model. The thermal efficiency from the cookstove model would become an input to the village energy model, and biomass remove rates from village energy model would become inputs into the sustainable agriculture model. The sustainable agriculture model would be modified with the appropriate databases to predict soil erosion and conditions in the developing world.

This would yield an end-to-end design tool that could predict health, environmental, agricultural, and energy outcomes for the developing world based on material and geometric design changes to the cookstove being designed. It would also be straightforward to extend this design tool with physical models of other energy devices being

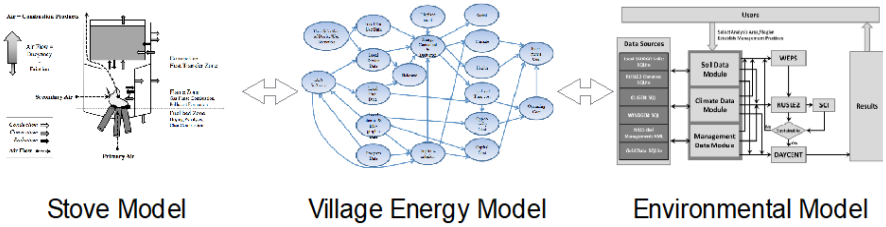

Fig. 4. Model integration example

considered for the developing world, giving designers, aid workers, and policy makers a powerful tool to predict the real-world impact of holistic, village-wide energy systems design in a way that has not been possible before. A conceptual representation of this is shown in Fig. 4.

Since each component and subcomponent would exist as an independent web microservice, each model could be maintained and updated by domain experts. The domain experts in each field could retain ownership of their models, even as those models become constituent parts of the integrated end-to-end stove design model. Further, the stove design model will automatically benefit from improvements in the performance, capability, and accuracy of the underlying constituent models.

6 Conclusion and Future Work

This paper presents a proposed domain specific language and supporting infrastructure for rapidly and fluently creating systems models in a federated modeling environment. This not only simplifies model development and team management, but also enables more informed decision making based on rapidly developed end-to-end models of complex systems in which people, energy, and the environment meet. Further research is needed to demonstrate the capabilities envisioned here and integrate the set of environmental and other models needed to create the stove design tool as presented.

References

1. Bryden KM (2014) A Proposed Approach to the Development of Federated Model Sets. In: Ames DP, Quinn NWT, Rizzoli AE (eds) International Environmental Modeling and Society 7th Intl. Congress on Env. Modeling and Software. San Diego, CA
2. Arnold TR (2013) Procedural knowledge for integrated modelling: Towards the Modelling Playground. Environ Model Softw 39:135–148. doi: 10.1016/j.envsoft.2012.04.015
3. Scientific Computing and Imaging Institute: SCIRun. http://www.scirun.org.
4. OpenDX. http://www.opendx.org
5. David O, Ascough JC, Lloyd W, Green TR, Rojas KW, Leavesley GH, Ahuja LR (2013) A software engineering perspective on environmental modeling framework design: The Object Modeling System. Environ Model Softw 39:201–213. doi: 10.1016/j.envsoft.2012.03.006

6. Rahman JM, Seaton SP, Perraud J-M, Hotham H, Verrelli DI, Coleman JR (2003) It's TIME for a new environmental modelling framework. In: MODSIM 2004 International Congress on Modelling and Simulation. Modeling and Simulation Society of Australia and New Zealand, pp 1727–1732
7. Peckham SD, Hutton EWH, Norris B (2013) A component-based approach to integrated modeling in the geosciences: The design of CSDMS. Comput Geosci 53:3–12. doi: 10.1016/j.cageo.2012.04.002
8. McCorkle DS, Bryden KM (2007) Using the Semantic Web technologies in virtual engineering tools to create extensible interfaces. Virtual Real 11:253–260. doi: 10.1007/s10055-007-0077-3
9. Mathworks. http://www.mathworks.com
10. AspenTech. http://www.aspentech.com
11. Community Surface Dynamics Modeling System Basic Model Interface (BMI). https://csdms.colorado.edu/wiki/BMI_Description
12. Community Surface Dynamics Modeling System Component Model Interface (CMI). https://csdms.colorado.edu/wiki/CMI_Description
13. Antonelli PE, Bryden KM, LeSar R (2016) A model-to-model interface for concurrent multiscale simulations. Comput Mater Sci 123:244–251. doi: 10.1016/j.commatsci.2016.06.031
14. Suram S, MacCarty NA, Bryden KM (2016) A Distributed Systems Approach to Engineering Modeling. To be submitted to: Advances in Engineering Software
15. MacCarty NA, Bryden KM (2016) A generalized heat-transfer model for shielded-fire household cookstoves. Energy Sustain Dev 33:96–107. doi: 10.1016/j.esd.2016.03.003
16. Thönes J (2015) Microservices. IEEE Softw. 32:113–116.
17. van Deursen A, Klint P, Visser J (2000) Domain-specific languages: an annotated bibliography. ACM Sigplan Not. doi: 10.1145/352029.352035
18. Fowler M (2011) Domain-specific languages / Martin Fowler. Upper Saddle River, NJ: Addison-Wesley, Upper Saddle River, NJ
19. McNunn GS, Bryden KM (2013) A Proposed Implementation of Tarjan's Algorithm for Scheduling the Solution Sequence of Systems of Federated Models. Procedia Comput Sci 20:223–228.
20. MacCarty NA, Bryden KM (2016) An integrated systems model for energy services in rural developing communities. Energy 113:536–557. doi: 10.1016/j.energy.2016.06.145
21. International Energy Agency (2010) Energy Poverty: how to make modern energy access universal? Special Early Excerpt of the World Energy Outlook 2010 for the UN General Assembly on the Millennium Development Goals. Paris.
22. Johnson NG, Bryden KM (2012) Energy supply and use in a rural West African village. Energy 43:283–292. doi: 10.1016/j.energy.2012.04.028
23. Lim SS, Vos T, Flaxman AD, Danaei G, Shibuya K, Adair-Rohani H, AlMazroa MA, Amann M, Anderson HR, Andrews KG, others (2013) A comparative risk assessment of burden of disease and injury attributable to 67 risk factors and risk factor clusters in 21 regions, 1990--2010: a systematic analysis for the Global Burden of Disease Study 2010. Lancet 380:2224–2260.
24. Bond TC, Sun H (2005) Can Reducing Black Carbon Emissions Counteract Global Warming? Environ Sci Technol 39:5921–5926. doi: 10.1021/es0480421
25. MacCarty NA, Bryden KM (2015) Modeling of household biomass cookstoves: A review. Energy Sustain Dev 26:1–13.
26. Muth DJ, Bryden KM (2013) An integrated model for assessment of sustainable agricultural residue removal limits for bioenergy systems. Environ Model Softw 39:50–69. doi: 10.1016/j.envsoft.2012.04.006