Environmental Modelling & Software 87 (2017) 49–63

Contents lists available at ScienceDirect

Environmental Modelling & Software

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

An overview of the model integration process: From pre-integration assessment to testing

Getachew F. Belete [a, *], Alexey Voinov [a], Gerard F. Laniak [b]

[a] University of Twente, ITC, 7500 AE Enschede, Netherlands
[b] US Environmental Protection Agency, Office of Research and Development, USA

ARTICLE INFO

Article history:
Received 15 March 2016
Received in revised form
30 August 2016
Accepted 30 October 2016
Available online 8 November 2016

Keywords:
Integrated modeling
Interoperability
Interfaces
Wrapping
Components
Web services

ABSTRACT

Integration of models requires linking models which can be developed using different tools, methodologies, and assumptions. We performed a literature review with the aim of improving our understanding of model integration process, and also presenting better strategies for building integrated modeling systems. We identified five different phases to characterize integration process: pre-integration assessment, preparation of models for integration, orchestration of models during simulation, data interoperability, and testing. Commonly, there is little reuse of existing frameworks beyond the development teams and not much sharing of science components across frameworks. We believe this must change to enable researchers and assessors to form complex workflows that leverage the current environmental science available. In this paper, we characterize the model integration process and compare integration practices of different groups. We highlight key strategies, features, standards, and practices that can be employed by developers to increase reuse and interoperability of science software components and systems.

Contents

* Corresponding author.
E-mail addresses: getfeleke@gmail.com (G.F. Belete), aavoinov@gmail.com (A. Voinov), laniak.gerry@epa.gov (G.F. Laniak).

http://dx.doi.org/10.1016/j.envsoft.2016.10.013
1364-8152/

1. Introduction

A model is a simplified abstraction of reality (Schmolke et al., 2010; Voinov, 2008). To answer questions related to environmental problems, scientists and policy-makers, may need to address complex issues at regional, continental, and even global scales (Harris, 2002; Verburg et al., 2008). Developing all-inclusive models is difficult, however, and we must integrate individual models that represent specific domains (Gregersen et al., 2007). Integration of models is becoming more important due to an increased desire to understand, investigate, and mitigate human impact on the environment (Laniak et al., 2013; Voinov and Cerco, 2010). The challenge is making standalone models discoverable, reusable, and interoperable (Goodall et al., 2011). Here, discoverability means the availability of meta information of a resource so it can be effectively searched for, located and understood to ensure proper use (Erl, T., 2008b). Reusability is how existing software can, in whole or in part, be used to develop new software (Frakes and Kang, 2005), as well as the degree to which an asset can be used in more than one system or in building other assets (ISO/IEC, 2011). Reuse also implies a decision process that includes evaluating software for its ability to serve a new purpose. And, finally, interoperability is the ability of a system or a product to work with other systems or products without special effort by the user (Chang and Lee, 2004).

In this paper a component may represent an elemental physical/chemical/biological/social process (e.g., infiltration or advection) or a logical integration of elemental processes (e.g., a watershed model). A system may be a model (an integrated set of elemental processes) or modeling system. Note that a model can refer to a component or a system. Finally, when we refer to an integrated modeling system we are including both the collection of science components and the framework software that facilitates their orchestration. These definitions apply throughout the paper.

Data exchange and data manipulation are fundamental to integrated modeling systems (Argent, 2004; Leimbach and Jaeger, 2005), and are often constrained by technical and conceptual challenges. Technically, individual models are designed to serve as stand-alone components that serve unique purposes and goals. Conceptual challenges include resolving the different ways modelers and science domains represent data and knowledge. Interoperability of models can also be provided at different levels. At the technical level, models should be able to 'talk to each other' which requires automating data exchange, making models jointly executable, and ensuring repeatability and reproducibility of model chain configuration and processing (Knapen et al., 2013). At the semantic level, models should 'understand each other' by identifying and, if possible, bridging semantic differences in an automated manner. After semantic mediation, the dataset should be syntactically interoperable, i.e., data produced by one model are converted and formatted to serve as input for a receiving model.

We performed a literature review to improve our understanding of the model integration process and also to collect and present better strategies for building integrated modeling systems. At first, our review was limited to integrated climate change mitigation analysis systems. Due to the commonality of integration techniques with other environmental domains, however, we broadened the scope to include integration of models in general. Our search used key phrases of 'integration of models', 'integrated assessment study' and 'integrated modeling frameworks,' and we considered 38 articles (see Table 1), of which 18 focused on specific integration modeling systems (aka frameworks).

To provide structure and context for the paper, we look to model development process which includes requirements analysis, design, implementation, and testing phases (David et al., 2013; Moore and Tindall, 2005; Whelan et al., 2014), with these steps conducted in an iterative manner (Grimm et al., 2014; Jakeman et al., 2006). Model integration process follows a similar sequence and is also iterative (Holzworth and Huth, 2011; Holzworth et al., 2014), with a small portion of integration realized first, and more complex issues and functionalities incorporated through iterations. Keeping all this in mind, we divided model integration process into phases of pre-integration assessment, preparation of models for integration, orchestration of participating models during simulation, data interoperability, and testing (Fig. 1).

This process forms the basis for our review. As we conducted the review, it became clear that three topics deserve special attention due to their significant role: interface standards for interoperability (within the preparation of models for integration phase); performance in integrating models (within the orchestration phase); and discovery, accessibility, and ease of use which are process-wide considerations. A summary of this structure, along with brief descriptions of categories of methods employed, examples of existing modeling systems, and references are presented in Table 1.

We note that:

- Most papers emphasize only certain phases of model integration.
- Comparing all selected frameworks with all aspects of the integration process is challenging due to limitations of available information. We believe, however, that, collectively, the full range of methods is represented.
- The set of model integration frameworks in Table 1 represents a robust, albeit incomplete, representation of existing frameworks.

Findings related to phases of integration: pre-integration assessment, preparation of models for integration, model orchestration, data interoperability, and testing are described in Sections 2 through 5. To avoid repetition of content, we treat preparation of models for integration and orchestration of participating models during simulation together in Section 3. Discoverability, accessibility, and ease of use of integrated systems are addressed in Section 6. Section 7 presents discussion as well as conclusions, and finally recommendations for increasing efficiency and effectiveness of the model integration process are presented in Section 8.

Table 1
Elements or aspects of model integration practices.

Elements or aspects of model integration	Different approaches to implement elements	Description and example frameworks	References
Pre-integration assessment	–	How selection of models and system conceptualization is done. Examples in: FRAMES, OpenMI, SEAMLESS	Butterfield et al. (2008); Janssen et al. (2009); Janssen et al. (2011); Van Ittersum et al. (2008); Whelan et al. (2014)
Preparing models for integration	Component based	Models are presented as components for interoperability. Examples in: CSDMS, ESMF, FRAMES, OpenMI, SEAMLESS	DeLuca et al. (2012); Janssen et al. (2011); Peckham et al. (2013); Van Ittersum et al. (2008); Whelan et al. (2014);
	Web service based	Models are presented as web services for interoperability. Examples in: AWARE, eHabitat, GEO Model Web	Dubois et al. (2013); Goodall et al. (2011) Granell et al. (2010); Geller and Melton (2008); Geller and Turner (2007); Nativi et al. (2013); Roman et al. (2009);
	Hybrid	Some models are presented as components and others as web services.	Castronova et al. (2013); Goodall et al. (2013);
	Tailor-made	Custom-made techniques for interoperability. Examples in: CMP, TIME	Moore et al. (2007); Rahman et al. (2003);
Interface standards for interoperability	–	Model interface standards designed by the modeling community. Examples in: CSDMS, ESMF, OpenMI, OGC WPS.	da Silva et al. (2003); DeLuca et al. (2012); Goodall et al. (2011); Hill et al. (2004); Moore and Tindall (2005); Peckham (2014); Schut and Whiteside (2007);
Orchestration of interaction of models during simulation	Component based	Orchestration of models is done using component-based development. Examples in: CSDMS, FRAMES, OpenMI, SEAMLESS	Gregersen et al. (2007); Janssen et al. (2011); Van Ittersum et al. (2008); Peckham et al. (2013); Whelan et al. (2014);
	SOA based	SOA principles are used for orchestrating models. Examples in: AWARE, eHabitat, GEO Model Web.	Dubois et al. (2013); Butterfield et al. (2008); Goodall et al. (2011); Granell et al. (2010); Nativi et al. (2013);
	Hybrid	Both component-based and SOA-based approaches are used.	Castronova et al. (2013); Goodall et al. (2013);
	Tailor-made	An ad hoc collection of techniques for orchestration. Examples in: APSIM, CMP, OMS,	David et al. (2013); Holst (2013); Moore et al. (2007); Keating et al. (2003);
Performance in integrating models	–	How performance optimization is treated by integration frameworks. Examples in: ESMF, OMS	David et al. (2013); Hill et al. (2004); Knapen et al. (2013).
Data interoperability	Hard-coding	Hard-coding is used for data interoperability. Examples in: CMP, eHabitat	Dubois et al. (2013); Moore et al. (2007);
	Framework specific annotations	Application-specific annotations are used for data interoperability. Examples in: OMS, TIME	David et al. (2013); Rahman et al. (2003);
	Controlled vocabulary and ontology	Reusable vocabulary and ontology is used for data interoperability. Examples in: FRAMES, SEAMLESS	Argent (2004); Athanasiadis and Janssen (2008); Geller and Turner (2007); Nativi et al. (2013); Rizzoli et al. (2008); Van Ittersum et al. (2008); Whelan et al. (2014);
Testing	Unit test, integration test, automated testing, test bench	How frameworks treat testing. Examples in: APSIM, FRAMES, ModCom, OpenMI, SIAT, TIME	Bertolino (2007); Bruggeman and Bolding (2014); Hillyer et al. (2003); Holzworth et al. (2014); Holzworth et al. (2015); Luo (2001); Moore and Tindall (2005); Rahman et al. (2003); Verweij et al. (2010); Whelan et al. (2014);
Discovery, accessibility, and ease of use	–	Efforts to improve discovery, accessibility, and ease of use of models. Examples in: BioMA, eHabitat, InsightMaker, iPlant, ModCom, SEEK, Systo,	Brooking and Hunter (2013); da Silva et al. (2003); DeLuca et al. (2012); Goecks et al. (2010); Goff et al. (2011); Hillyer et al. (2003); Wolstencroft et al. (2015);

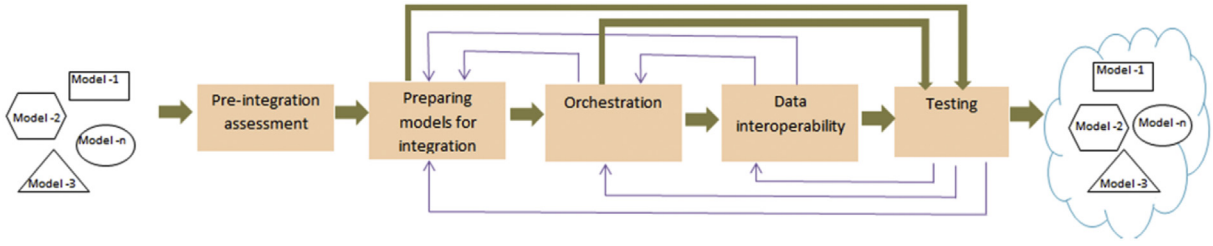

Fig. 1. Iterative relationship among model integration phases.

2. Pre-integration assessment

Pre-integration assessment combines science-based requirements of an environmental assessment with computer technologies to create an initial integrated software system workflow design. The assessment may be related to research, environmental analysis, or decision/policy support. Pre-integration represents a key aspect of model integration since it establishes the roles of and relationship between scientist (performing an environmental assessment, for example) and software engineer (designing and building the computer-based modeling system). These complementary roles have often been performed by the same person, but

as science problems and required modeling systems have grown more complex, these roles necessarily required a relationship between experts in the domain science and software engineering.

On the science side, pre-integration assessment includes a problem statement; articulation of the purpose and goals of the science-based modeling effort; conceptualization of the relevant system (including major components and interactions); characterization of scenarios and use cases; analysis of alternatives; design of a science-based approach; and a statement of any project resource constraints and solution criteria. Collectively, this information represents the requirements that must be satisfied by either an existing or new software system.

Ramamoorthy et al. (1992) point out that the main cost of integration is the effort required to detect and resolve inconsistencies in the coupled system, a cost that can be reduced if inconsistencies are detected and resolved prior to integration – that is, during the process of transforming the science design to a software design. To aid transformation, the software engineer may utilize a set of tools recommended by the System-of-Systems Engineering (SoSE) approach (Butterfield et al., 2008), including: context views that define relationships, dependencies, and interactions between the system and its environment; use cases that list interactions between the system and the user; activity diagrams which show the flow of activities while using the system; and diagrams that provide a high-level description of system behavior.

In reviewing existing frameworks, we found only a few scientific papers that discuss how pre-integration assessment was performed. In SEAMLESS (Janssen et al., 2011; Van Ittersum et al., 2008), pre-integration was performed through discussions that involved scientists, assessors, decision support specialists. The discussions focused first on defining scenarios and indicators (which would be tested by the integrated system), and then on analyzing system requirements. An assessment project ontology (Janssen et al., 2009) was developed to create a common understanding for people involved in the project. The discussions produced the overall architecture of the integrated system, specifications of components, and data to be exchanged between them.

For FRAMES, Whelan et al. (2014) intended to accommodate multiple modeling domains and assessment scenarios (workflows) by designing the framework to be independent of any specific set of models. They suggested that developing a framework capable of constructing and orchestrating integrated modeling systems required detailed analysis with respect to 19 aspects of design and implementation that they have identified. In summary, these requirements target metadata definition, standardized vocabularies, component connectivity, data-transfer compatibility, system architecture and functionality, and model accessibility.

The OpenMI project provides a formal standard for data exchange among models, including specification of an application program interface that describes how models should request, and respond to requests for, data. Various frameworks, including the OpenMI software development kit, implement this standard to achieve model interoperability. OpenMI suggests that model-linking can begin only after identifying data that can potentially be exchanged, defining how the data will be exchanged, and schematizing deployment and running of the linked computation. Developers who want to use the OpenMI framework should ensure that components use the same platform (for example, do not mix Java and .Net components); that the same OpenMI version is implemented; and that input data not provided by other components will be provided from external data sources. This shows that pre-integration assessment must consider different levels of details in meeting integration goals.

The connection between integration frameworks and science domains cannot and should not be ignored, although there are various levels of 'affinity'. For example, SEAMLESS was entirely driven by agricultural science. FRAMES or OpenMI were intended to be domain-independent, however, they were initiated by specific science questions and domains. FRAMES is rooted in chemical fate and exposure modeling, but OpenMI is largely driven by hydrological applications. Similarly, ESMF had its origins in Earth systems modeling, while CSDMS was instigated by surface dynamics modeling. Such affinity to particular science domains is revealed by model components in the respective model repositories, in the types of applications where the frameworks have been tested, and also in the equations and methods that the frameworks typically handle.

From the literature review, we identified the following questions for facilitating pre-integration assessment:

(1) Why are we integrating? What are the objectives?
(2) What system is to be simulated? What scenarios are to be simulated using the integrated system?
(3) What indicator variables are to be tested using the integrated system? What are the options for calculation of the indicator values?
(4) Are the required modeling components from the same science domain? Are the semantics compatible? Are there frameworks that cover the relevant science domain, or you will need to fit into a 'foreign' framework or develop both the new framework and the models within this framework?
(5) How much data and information will the models exchange, such as one variable vs. hundreds of variables, how often, over what spatial grid and resolution? What processes require feedback loops?
(6) Is it important (or necessary) to keep participating components independent of one another, or will linking be established by creating a permanent dependency of one component upon another, effectively merging multiple components into a single one?
(7) What are the implications of programming languages, operating systems, and computer platforms used? Will language interoperability tools be required? Will individual components remain on their respective platforms (e.g., development environment or operating system) or they will they be migrated to a common platform in the integrated system?
(8) What programming skills are required and are they available? How much new software will be required to link models?
(9) How will system input data (i.e., data about the physical/chemical/biological/social etc. system being modeled) be organized and distributed among components?
(10) Are data unit conversion functions needed?
(11) Is dataset translation, aggregation/disaggregation required? If so, are the tools available or must they be created?
(12) Are there specific performance requirements to satisfy (e.g., parallel computation)?

Consideration of the above-mentioned conceptual and technical questions is the basis for an initial design that will enable to evaluate feasibility of the integration and resource requirements. The result of the pre-integration assessment will be either to proceed with integration, not consider integration, or obtain additional information to make a better decision. If the decision is to continue with integration, the next questions are “How should the models be prepared for integration?”, and “How should model orchestration be conducted?” These topics are addressed in the next section.

3. Technical integration of models: preparing models for integration and orchestration of participating models during simulation

Technical integration requires specific software strategies that prepare modeling components for linked execution within a framework designed to execute the integrated modeling system. Because the literature does not provide sufficient detail, we will not attempt a technical framework-to-framework comparison. We introduce concepts of model preparation and system orchestration in Sections 3.1 and 3.2; discuss model preparation strategies and requirements employed by existing frameworks in Section 3.3; and discuss two key issues related to model integration and orchestration, interface standards for interoperability and framework performance optimization, in Sections 3.4 and 3.5.

3.1. Preparing models for integration

From a software engineering perspective, preparing a component for integration depends on its native software environment (language and OS), manner in which inputs and outputs are organized (e.g., using files, databases, etc.), and integration approach used by the target system (framework). Consider a practical example of a model that produces climate change scenarios up to year 2100. The simulation start year is constant, but the end year is a user input. The model's output, as originally designed, is accessible only after the final time step. If we want to link this model with another so that data is exchanged at the end of each time step, we must make certain modifications. Modifications include preparing a function that returns the values of the required variables at the end of each time step, and a function that receives values of selected variables from the model to be linked. In achieving this linkage, we need to consider that the two models are developed using different languages, are deployed on different platforms, treat space and time differently, etc. The process of preparing models for integration can thus go through some combination of steps, as summarized in Fig. 2. And if we want to use the model independently of the development environment, or need to protect intellectual property rights of developers, we may have to deliver a model in binary format as DLL or EXE file rather than source code (Moore et al., 2007).

Model modifications will sometimes be needed to enable automated input-output data exchange that conforms to a specific framework design. In such situations, small existing (legacy) models can be reprogrammed, or we may directly modify the model interface to conform to the framework data exchange protocol (Fig. 3). For larger and more complex models, however, rewriting is time-consuming and error-prone (Peckham et al., 2013). To avoid rewriting, it is common to develop a communication wrapper (Van Ittersum et al., 2008) which is a thin layer of code on top of the native component that manages data exchanges between the native component and other components or the system. Wrappers could be developed to make a model language-interoperable; make a model accessible over a network; implement agreed-upon names for functions and variables; provide a custom interface with required input-output attributes; or address a combination of these objectives. Depending on the integration framework, a model wrapper can be developed following industry standards or “local” standards; some of these are discussed in Section 3.4. Generally, the work required to develop wrappers depends on how much we want to adapt the existing model interface.

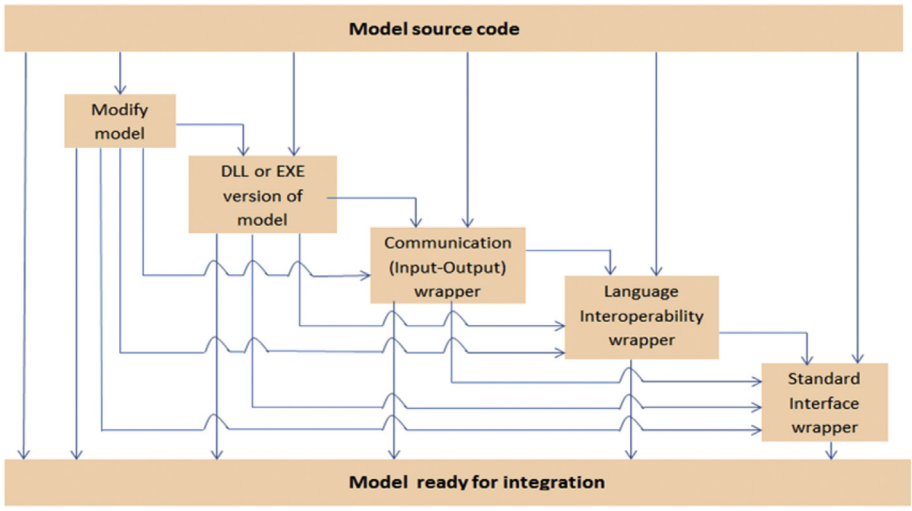

Fig. 2. Alternative paths in preparing a model for integration.

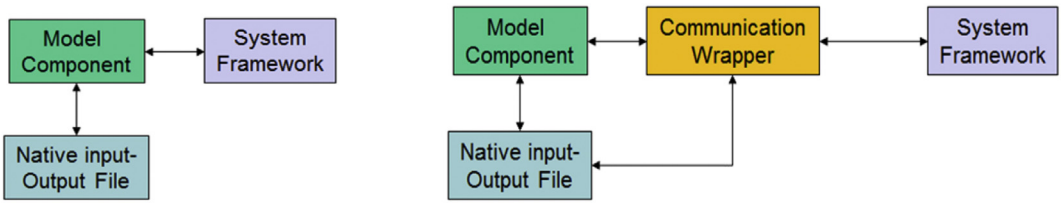

Fig. 3. Common strategies to implement framework data exchange protocol on existing models are to modify the model interface directly to conform to the framework data exchange protocol or develop a wrapper as a separate component.

Table 2
Brief comparison of technical integration methods commonly used by integration frameworks.

	Component-based	SOA-based	Hybrid of component-based and SOA	Custom-made
Primary characteristics	Reusability of components and system level functionalities.	Loosely coupled systems; distributed services.	Models as components and as web services.	An ad hoc collection of methods/techniques is used.
Advantages	Plug & play; separation of concern; minimizes context switching time of components.	Plug & play; location independence; platform independence; scalable.	Enables linkage to different frameworks; manage heterogeneity; reuse existing frameworks.	Developers can use ad hoc techniques for orchestration.
Disadvantages	Developing reusable orchestration engine is challenging.	May require heavy data exchange, and high availability.	Requires parsing data between components and web services.	May not support new version of models; lack of extensibility.
Examples	CSDMS, OpenMI, FRAMES, SEAMLESS	AWARE, eHabitat	Castronova et al. (2013), Goodall et al. (2013)	APSIM, CMP

3.2. Orchestration of participating models during simulation

A complementary part of technical integration is orchestrating models during simulation which requires enabling and managing interactions between components (Madni and Sievers, 2014). The orchestration task includes: identifying components to be included; establishing linkages and overall workflow to be implemented; and managing execution of the workflow (initializing components, synchronizing execution, facilitating data exchange, error management, and system file management). Furthermore, the operational nature of the data exchange within an integrated system may be sequential (feed forward) or iterative (feedback). In feed forward systems, one component completes its execution and produces data for subsequent consumption. In feedback systems, components require the exchange of intermediate data values at certain time steps.

3.3. Commonly used approaches for technical integration of models

The actual methods employed for technical interoperability of models vary among development groups. From our literature review, we categorized them as component-based, web service-based, a hybrid of component and web service-based, and tailor-made (custom-made) techniques. These methods can be applied to both legacy and new science components. A brief comparison of the methods, which are further described in the following sections, is presented in Table 2.

3.3.1. Component-based approach for technical integration of models

The component-based approach is the design and construction of computer-based systems using reusable software components which are independent executable entities accessible only through its interface (Debayan, 2011). One main difference of component-based development from non-component-based is the emphasis on reusability of components (Crnkovic et al., 2006). This approach recognizes different levels of reusability: code-level, module-level, library-level, design pattern-level, and framework-level (Rekaby and Osama, 2012).

The component-based development lifecycle involves two main phases, component production and component integration or utilization (Rekaby and Osama, 2012). In the component-based approach, components have standardized calling interfaces (DeLuca et al., 2012) or 'sufficient commonality' (Debayan, 2011) which enables developers to formulate predictable architectural patterns that manage interactions between them. As a result, the workflow-managing functionality can itself be developed as a reusable framework-level component.

The component-based approach is one of the most commonly used techniques to prepare models for integration and manage simulations. For example, the Earth System Modeling Framework (ESMF) uses a component-based approach to integrate models (DeLuca et al., 2012); ESMF is based on the principle that complicated applications are broken into smaller components with standard calling interfaces. A model component that implements the ESMF standard interface can communicate with the ESMF shell and inter-operate with other models. Similarly, the CSDMS (Peckham et al., 2013) requires models to be presented as components to join the integration framework. Component-based programming is chosen so components can be added or replaced at runtime, and to ensure that components written with different programming languages and hosted on remote locations can join the framework. The CSDMS framework requires models to implement the two-level component-interface standard provided. The CSDMS uses functionalities of the common component architecture (CCA) framework[1] to coordinate communication between components developed using different frameworks (Peckham et al., 2013). FRAMES (Whelan et al., 2014), a feed forward modeling framework, employs the component-based approach and incorporates data dictionaries for data exchange. Wrappers are written for each component to read and write data to the dictionaries. The framework then manages transfer of data between components during runtime through an inter-component communication API. The API consists of routines that read model inputs, write model outputs, handle language interoperability, manage unit conversions, and provide error handling.

Existing component-based frameworks mostly employ a local data exchange standard. The exception involves the OpenMI standard that provides plug-and-play model access by implementing a component-based software development paradigm. The OpenMI strategy is to access model input-output directly at run-time and not use files for data exchange. OpenMI-compliant components can exchange data without any supervising authority if the components have information about each other (Gregersen et al., 2007). If, however, the communication involves iterations – that is, feedbacks and looping over components – a separate controller component is required; the controller contains an iteration algorithm and a link to individual components involved in the iteration. The OpenMI standard has been implemented in some framework designs such as SEAMLESS-IF (Janssen et al., 2011; Van Ittersum et al., 2008), SIAT (Verweij et al., 2010), NITROEUROPE (Sutton et al., 2009), EVOLTREE,[2] and FluidEarth.[3]

The component-based approach has a number of advantages for

[1] CCA - http://www.cca-forum.org/overview/.
[2] EVOLTREE - http://www.evoltree.eu/.
[3] FluidEarth - http://fluidearth.net/default.aspx.

the modeling domain, including:

- Components can be added or deleted from a simulation at runtime (Armstrong et al., 1999; Peckham et al., 2013);
- It is suitable for rapid execution of workflows because when an application (in this case, the workflow manager) calls a component, the component can run within the process space occupied by the calling application, and it can share memory and processor time with the calling application. This makes the communication between the calling application and the component very fast;
- It enables models to be independent and, thus, reusable entities;
- It enables creation of both tightly- and loosely-coupled systems; and
- It facilitates sharing of models and possibly collaboration among modelers or workgroups who share components.

Although the component-based approach has many advantages, it also has many challenges:

(1) Communication between components developed using different programming languages requires language interoperability tool (Peckham et al., 2013). Some systems avoid this by requiring all components to be written in the same language, but some use a language interoperability tool– for example, the CSDMS framework uses Babel[4] a language interoperability tool that manages communication between components written in C, C++, Java, Python, or FORTRAN;
(2) Communication between components hosted on different computers is commonly done via tight-coupling. For example, both CSDMS and FRAMES use Remote Method Invocation (RMI) for remote communications. RMI enables an object hosted on one computer to invoke an object hosted on another. RMI has disadvantages, however, since it operates on the object level (i.e., components are treated as objects rather than independent applications), and it creates tight-coupling between components. RMI is also slow and can be used only with Java, making it programming language-dependent (Tanenbaum and Van Steen, 2007);
(3) Due to the many unique implementations of data exchange protocols, the component-based approach has resulted in interoperability that is typically constrained to a specific framework, versus a common standard (as proposed by OpenMI) that would facilitate global or universal reuse and interoperability.

3.3.2. *Web services and service-oriented approach for technical integration of models*

Web services are web application components that can be accessed via the World Wide Web. Unlike the component-based approach, web services can interact independently of the underlying platform. They eliminate interoperability complications related to programming language, operating system, and computer platform (Castronova et al., 2013; Dubois et al., 2013; Goodall et al., 2011). Independently-built heterogeneous services can be linked to build integrated modeling systems using a Service-Oriented Architecture (SOA) design approach. SOA is based on design principles: standardized service contract or description; service loose-coupling; service abstraction (hiding unnecessary information); service reusability; service autonomy (keeping the service as an independent unit); service statelessness (minimizing the time in which the service will stay stateful); service discoverability (use meta information that accurately describe the service); and service composability (design services that will be composed with other services) (Erl, 2008b).

A main factor that differentiates the service-oriented approach from the component-based approach is that service-orientation implies distributed components will interoperate over a network (Huhns and Singh, 2005; Goodall et al., 2011). SOA enables building distributed applications by composing or putting together services while keeping them autonomous, regardless of their heterogeneity (Pessoa et al., 2008). In SOA, unlike a component-based approach, a langauge interoperability tool is not required for communication of heterogeneous services. Services may also run on different operating systems, and can encapsulate legacy systems. Configuration of services into integrated modeling systems can occur dynamically (Erl, 2008a; Papazoglou, 2008) and is scalable (Nativi et al., 2013).

Model developers use web services for various reasons, and the service-based approach is becoming increasingly popular. For example, the Group on Earth Observation (GEO) Model Web initiative (Nativi et al., 2013) suggests that presenting models as a service will increase interoperability and facilitate interaction of models, databases, and websites. They use a SOA-based approach to orchestrate distributed components, introduce a broker component to facilitate discovery of services, intermediate (i.e., inter-component) services to adapt output content and format of one model to an appropriate format before it is passed to the next, and handle asynchronous communication between remote servers. Similarly, the Model Web for Ecological models (Geller and Melton, 2008; Geller and Turner, 2007) suggests that an open-ended network of interoperable models and databases that is maintained, operated, and served independently can be realized with web services. The Model as Service approach (Roman et al., 2009), which is an extension of the Model Web initiative, notes that web services can migrate a stand-alone model into a service on the Web.

The AWARE[5] integration framework (Granell et al., 2010) uses the open geospatial services standard to make environmental models interoperable. They selected SOA because services can be published, discovered, aggregated, invoked, and reused independent of the underlying technology used to build them; service standards define how to interact with the available services in a uniform manner; the interoperability of web service provides the opportunity to present legacy models as services; and SOA enables creation of value-added functionality by linking existing services. Web services can be delivered to end users for visual service-chaining by presenting them via Cloud computing, and Grid computing technologies, and service-oriented infrastructures. Goodall et al. (2011) recommends a service-oriented computing approach to orchestrate models in simulation since SOA views integrated modeling systems as an interconnected collection of distributed components. Similarly, eHabitat (Dubois et al., 2013) uses SOA with brokers to handle service composition. A discovery broker, access broker, and semantic discovery broker mediate communication between services and hide the heterogeneity of underlying services.

The web service approach is used mainly to enable cross-language interoperability and link components that reside on different operating systems. Its main disadvantages are that request and replay messages are based on XML, which is larger in size than binary protocol; and performance on the Internet varies from time to time and place to place. It may require high availability.

[4] https://computation.llnl.gov/casc/components/index.html#page=home.

[5] http://www.aware-eu.info/.

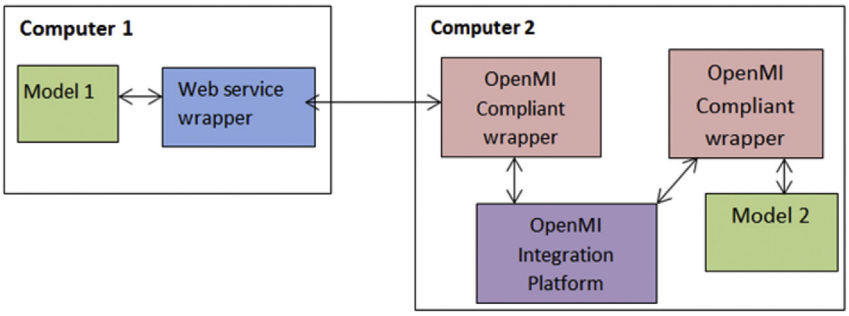

Fig. 4. Two-level-wrapping for cross-platform interoperability: Web service and OpenMI-compliant component wrapper.

3.3.3. Hybrid of component-based and web service-based approach for technical integration

Constructing an integrated modeling system using a specific software architecture or framework may not be optimal since a mix approaches may be more useful. Some modelers use a hybrid of component-based and service-oriented approaches to manage technical integration of models. Consider the following two cases where both used two-level wrapping to access a model hosted on a remote machine, as shown in Fig. 4. Castronova et al. (2013) used an OGC WPS wrapper and an OpenMI component wrapper to prepare the models for integration. Implementing the WPS on a model transforms it into a web service which can be accessed remotely via Internet. The linking of models is handled through the OpenMI-based HydroModeler environment, however, which requires wrapping models with the OpenMI interface standard. They developed an OpenMI compliant-wrapper that invokes WPS-based services.

Likewise, Goodall et al. (2013) used the hybrid approach to enable coupling across modeling frameworks. They linked two existing modeling frameworks – ESMF and OpenMI – using a service-oriented architectural approach. A climate model developed within ESMF and hosted on a high performance computing cluster was made available as a web service, and an OpenMI-based client that accesses the service was also built. The Soil and Water Assessment Tool (SWAT), a hydrological model, was hosted on a Windows-based personal computer and made OpenMI compliant. Communication between the two OpenMI-compliant components was organized using the OpenMI configuration editor which handled the transfer and translation of the exchanged items. From these two examples we can see that the service-oriented approach overcame heterogeneity due to computer architecture and programming language differences and the component-based framework was utilized to re-use the existing OpenMI integration tool.

3.3.4. Tailor-made (custom-made) techniques for technical integration of models

Some integration frameworks use interoperability techniques that are unique and neither component-based nor web service-based. We use the term 'tailor-made' or 'custom-made' to group those frameworks into one category. In the custom-made approach, developers use an ad hoc collection of methods/techniques for technical interoperability. They use framework-specific 'principles' and this could hinder the continuity and extensibility of such frameworks. Each is quite different and has its own advantages and disadvantages.

In the Common Modeling Protocol (CMP), models are presented, for technical interoperability, as an executable file or dynamic link library (Moore et al., 2007). No further specification is required on the structure of the model interface. Simulation specification is a hierarchical arrangement of components (i.e., a system is a tree of subsystems) using the Simulation Description Markup Language (SDML). It uses message-passing system to coordinate communication between components. To run a simulation, the user must define the configuration of components and initial values of state variables in an XML document that conforms to the SDML schema. Similarly, the Agricultural Production System Simulator (APSIM) (Keating et al., 2003) modules are self-contained and a central message-passing system is used for inter-module communication. Every simulation in APSIM must be represented as an XML document. Users employ the GUI to specify a simulation configuration by selecting preferences from a set of toolboxes. On the other hand, the TIME modeling framework (Rahman et al., 2003) requires models to be developed as a class using any.NET language such as C#, Visual J#, Visual Basic .NET, or Fortran 95.NET because cross-language interoperability[6] between models is handled by the Common Language Runtime[7] environment of the .NET framework.

Domain Specific Languages (DSL) (Fowler, 2010), programming languages developed to meet specific needs of a domain, also manage orchestration of models in simulation (Holst and Belete, 2015). OMS3 provides a domain-specific language named 'Simulation DSL' to configure simulations (David et al., 2013). The DSL is developed independently of the framework and functions as a layer on top of participating components. Similarly, the Universal Simulator for Ecological Models — UniSim — (Holst, 2013) uses DSL to define orchestration of components in a simulation.

In the custom-made approach, we observe that frameworks do not impose explicit model interface standards for interoperability, which could imply they require custom code to access specific models. Incorporating new versions of models could also require modification of the custom code. Custom-made approaches thus have the disadvantages of lacking flexibility, maintainability, and extensibility.

3.4. Interface standards and interoperability

Wrapping of models is commonly done for technical interoperability. If the wrapper interface is developed using a set of standard names for functions and variables, however, semantic interpretation of their attributes will be uniform and software can be included to test the correctness of data exchanges. Interface standards can therefore serve both technical and semantic interoperability. Currently, a standard does not imply universal awareness, acceptance and use, but rather that a framework designer has either specified or adopted a standard for use; thus, different interface standards are available. Commonly interface standards have mandatory and optional parts. They can be applied in

[6] https://msdn.microsoft.com/en-us/library/vstudio/a2c7tshk(v=vs.100).aspx.
[7] https://msdn.microsoft.com/en-us/library/8bs2ecf4(v=vs.110).aspx.

Table 3
Some interface standards used by the modeling community.

Name of interface standards	Description	Some of its features	Reference
OpenMI standard	A collection of programming interfaces for components.	It consists of a list of function names or method signatures called *initialize*, *update*, *inputItems*, *status*, etc. that enable the model being wrapped to request data from other models and respond to requests for data.	Moore and Tindall (2005)
CSDMS standard	Developed to simplify conversion of an existing model to a reusable, plug-and-play model component.	It has two levels of specification: (1) BMI[a] developed to provide model metadata to the next level and (2) CMI[b] communicates with BMI functions as well as with Service Components and the CSDMS Framework	https://csdms.colorado.edu
OGC WPS	Specifications on how inputs and outputs of geospatial services are handled.	It has three mandatory operations that should be implemented by all services: *GetCapabilities*, *DescribeProcess*, and *Execute*.	Schut and Whiteside (2007)
ESMF standard	Developed to have standard interface for model components.	To make a model ESMF-compliant, it should include methods named *initialize*, *run*, and *finalize*.	da Silva et al. (2003); DeLuca et al. (2012); Hill et al. (2004)
Web service standard for the water domain	Developed to present models from water resources domain as web services.	It combines semantic features like where, what, and when of the exchanged data from the OpenMI with web service features of the OGC WPS standards.	Goodall et al. (2011)

[a] BMI - http://csdms.colorado.edu/wiki/BMI_Description.
[b] CMI - http://csdms.colorado.edu/wiki/CMI_Description.

developing new models or in re-factoring of existing models. A brief comparison of existing interface standards is provided in Table 3.

3.5. Performance optimization and related issues in orchestrating simulations

Performance of computer simulation is an important concern for all modeling and integrated modeling, in particular. In this context, performance optimization is the process of tuning a software system to execute rapidly. Performance can be affected by the design of the software itself, the operating system, middleware, hardware or communication networks (Woodside et al., 2007). Here we discuss two performance optimization concepts applied by developers of integration frameworks: high performance computing (HPC) and parallel processing (multi-threading).

HPC is a cluster computing environment with a number of nodes containing both processing power and memory (Gavrilovska et al., 2007). The benefit of HPC is that data which are too large to process by one server will be distributed over a number of nodes. The HPC platform manages the distribution and processing. HPC has been used by modeling frameworks to simulate complex workflows such as by ESMF for climate simulations (Hill et al., 2004). Linking the integrated system with the HPC platform is an added value.

In designing integration, orchestration of models can be treated as a serial or parallel process. In reality, many natural processes represented by integrated models can happen in parallel, but such parallel processes are commonly represented using sequential execution of simulation models. In fact, almost all modern computers currently have multi-core CPUs, and the parallel execution approach provides effective utilization of available processing resources (David et al., 2013). Parallel execution of processes can be achieved using multithreading techniques (Tanenbaum and Van Steen, 2007). This also improves performance in non-HPC environments because threads in a process share the same memory area which reduces the time spent for process context-swapping. For example, OMS3 uses a multithreading technique for parallel processing of components in one machine (David et al., 2013). On the other hand OpenMI's control flow is pull-driven – that is, the succeeding component requests information from the preceding component in a chain. A component handles only one request at a time before acting upon another request; however, the called process can handle the request by using multiple threads or by a parallel computing sessions, depending on the developer's preference. In OpenMI, functionalities such as parallel, cloud or high performance computing are yet to be developed (Knapen et al., 2013).

For non-HPC environments, integration frameworks can increase performance by applying multithreading techniques over distributed computers. Multithreading has a significant effect on system performance since subsystems can be executed in parallel on different computers (Tanenbaum and Van Steen, 2007). And if each process takes longer to execute (compared to the time to exchange data), multithreading will generate particular rewards in performance gains (Wainer et al., 2008).

4. Data interoperability

4.1. Issues on data interoperability

Data interoperability is the ability to correctly interpret data that crosses system or organizational boundaries (Renner, 2001). At the core of integrated modeling simulations is the exchange of data between workflow components. Implicit in our previous discussion of component preparation and workflow orchestration aspects is the need to ensure that data being transferred between components is: semantically interoperable – that is, has unambiguous meaning and is correctly mapped and, if necessary, translated; and structurally interoperable – that is, the dataset is formatted in the required form. Semantic interoperability issues for data are the largest hurdles in model integration (Argent, 2004). Differences in entity naming, scales used to represent space and time, and ways we represent a concept in relationship to others are the most common causes of semantic heterogeneity among models.

It is often necessary to perform additional computations on data as it is transferred from one model to another. Data mediation requires identification of the WHAT, WHERE, and WHEN of data in the context of each model (Moore and Tindall, 2005). WHAT is the name, definition, and quantitative description of the data represented by the variable, including factors such as units, allowable minimum/maximums, etc. WHERE is location information and spatial resolution. WHEN describes temporal information and

Table 4
Brief comparison of data interoperability techniques used by integration frameworks.

	Hard-coding	In-house developed standard and annotations	Controlled vocabulary and an ontology
Primary characteristics	Uses an explicit rather than a symbolic name.	Use metadata for mediation.	Use vocabulary or ontology for mediation.
Advantages	Easier to implement.	Flexible and extensible.	Flexible, extensible, and accommodate change.
Disadvantages	Lacks extensibility and flexibility.	Usage is limited to small group.	Difficult to construct the vocabulary or ontology.
Examples	CMP	TIME, OMS3	SEAMLESS, FRAMES

resolution. While space and time are obvious contexts, other "dimensions" must be considered as well: for example, differences between how components categorize land use, or human/ecological species demographics, chemicals, etc. must be resolved. Thus, unlike technical integration of modules, developing a 'generic' module that will handle data mediation among different models is quite difficult.

A related issue is, which component will perform the data mediation? An independent module be dedicated to this purpose, or will the participant models themselves mediate the data. In most integration frameworks, a dedicated module for data interoperability is developed, but participating models can also be responsible. For example, in OpenMI, the producing (source) component is responsible for identifying and extracting the requested data, performing appropriate conversions, and determining semantic mapping to variables from the receiving component (Moore and Tindall, 2005). It is assumed that data conversion "can be done in the most optimal way by the source component" (Gregersen et al., 2007). This indicates that, if the source component can be linked to five components, it should contain implementations of dataset conversion for each of them. Such architecture has two problems: a duplication of effort by implementing dataset conversion functions for each model and a lack of manageability when the number of components grows because the method may not be extensible.

Frameworks generally vary in the degree to which they codify the metadata and conversions related to data exchange. Data interoperability is achieved either by hard-coding, using an in-house developed standard and annotations, or via a controlled vocabulary and ontology, as shown in Table 4 (Athanasiadis and Janssen, 2008; David et al., 2013; Rahman et al., 2003). In the following subsections we discuss data interoperability techniques used in the frameworks we reviewed.

4.2. Hard-coding for data interoperability

Hard-coding uses an explicit rather than symbolic name for something that will likely be changed[8]. In the context of data interoperability, this means that mapping and transfer of variables and related metadata from one component to another is achieved by direct connection between components. One ramification is that the mapping and transfer code must be generated for every combination of components that may share the data, resulting in duplication of code and high maintenance costs. Hard-coding is the most frequently used technique for data mediation because it is relatively easy to implement, but its implementation varies among frameworks. For example, in CMP (Moore et al., 2007) designers stress that adopting a set of conventions on component interfaces to address semantic and dataset interoperability is difficult. To reconcile inconsistency of data representation between components, they suggest using the component that translates the meaning of attribute names, quantities, and units of one component to another, or writing code in one component about the other components and adapting its interface at run-time.

[8] Hard-coding - http://whatis.techtarget.com/definition/hardcode.

4.3. Framework specific annotations for data interoperability

An annotation is a form of metadata which could be a comment or an explanation that is attached to text, image, or data (Yu, 2011). The second method of data interoperability is using application of specific annotations. Data mediation operations are performed based on information fetched from the annotations. A major advantage is that annotations for a component are reusable. Its disadvantages include that it is framework-dependent and that information presented in the annotation is dependent on how the developer defined the annotation.

Consider that TIME (Rahman et al., 2003) uses .NET metadata tags for classifying and documenting different attributes of models such as input-output, minimum and maximum allowed values, and units. OMS3 (David et al., 2013) similarly provides metadata information of classes, fields, and methods as annotations which contain information about component execution, unit and range constraints of data, and documentation. We consider the extension of the OpenMI standard in this category. The basic definition of the standard does not provide conventions for naming quantities and data operations (Gregersen et al., 2007). Data is expressed only in terms of data types, such as double, integer or string. However, the extension of the standard provides protocols (i.e., IBaseExchangeItem interface) to define how quantities, space, and time are described in the specific context of the data. This information can be fetched by calling functions, which provide metadata about the component such as *ValueDefinition*, *SpatialDefinition*, and *TimeSet*.

4.4. Controlled vocabulary and ontology for data interoperability

A controlled vocabulary is a list of "preferred terms" and definitions. When those terms are organized using a layer of interrelationships, axioms, and classes the result is ontology. As we progress toward more technical integration and automated data exchange, the need for standardized data interoperability via shared terminology and ontology becomes more important (Geller and Turner, 2007; Villa et al., 2009).

For example, the CSDMS framework handles semantic mediation by using Standard Names (Peckham, 2014) in the BMI and CMI interfaces. Semantic information about components can be found by accessing Model Information Functions of the BMI interface such as grid type and time steps. Dataset conversion functions are designed as reusable service components and can be accessed by CMI functions. Standard Names standardize not only variable names but also 'model assumption names,' based on a set of naming conventions. The naming conventions are based on 'object name + quantity name' and assumption names can be associated with variable names using an XML-based ⟨assume⟩ tag. Modelers are thus expected to use existing standard names in developing new models and contribute to identification and documentation of

new standard names. Another example is the FRAMES (Whelan et al., 2014) integrated modeling framework which uses Dictionary files (DICs) for standardized data exchange between models. DICs consist of a list of standard names for variables and related metadata that component-models are expected to write to and read from. For dataset mediation, FRAMES has a service component that brokers data movement into and out of DICs. The mediation component performs quality checks on the data and executes any necessary unit conversions.

The goal of standardizing names is an ontology for automatic semantic mediation (Papazoglou, 2008; Peckham, 2014) which preserves semantics of the data and eliminates the need for custom-developed code for semantic mediation (Papazoglou, 2008). An ontology enables higher levels of reuse (Wang et al., 2002) and also enables framework-independent representation of knowledge on data structures (Rizzoli et al., 2008). Building ontology to address semantic interoperability across multi-disciplinary models is not a task for a few individuals (Argent, 2004) – constructing a multi-disciplinary ontology requires huge collaborative efforts by different disciplines, as in Janssen et al. (2009).

Our literature review indicates ontologies are not commonly included in model integration frameworks. The GEO Model Web initiative (Nativi et al., 2013) suggests that Semantic Web technologies (RDF,[9] OWL,[10] SKOS,[11] SPARQL[12]) can create understanding between models since they provide machine-understandable semantics; it also suggests that automatic semantic mediation can be realized only if the naming of service properties is unambiguously defined. Standardized model metadata is a means to achieving semantic interoperability.

Although four models are linked In SEAMLESS, ontology was built to avoid the hard-coding of semantic mediation. The ontology enables semantic mediation among models, indicators and data sources (Janssen et al., 2009), and also makes mediation process extensible to incorporate new models. The SEAMLESS ontology and its content is stored in a knowledge base, and model output is written to a database; the database and the ontology are linked by a relational database schema. The framework has a dedicated Knowledge Manager (KM) module to maintain links between components, manipulate data sources, and facilitate data exchange (Athanasiadis and Janssen, 2008). The KM is based on an ontology specifying the data exchange among participating models. Inter-component communication occurs via underlying databases and is dependent on the KM. The KM operates as a mediator by receiving data from models and mapping them to appropriate database tables.

5. Testing integration

Testing verifies or validates whether a system satisfies certain specified requirements and improves the integrated system's quality. Integration of models is error-prone since it requires both scientific and programming efforts (Bruggeman and Bolding, 2014). Even though individual components are tested independently and may perform correctly, integration output can still produce unexpected results due to different assumptions made during integration (Voinov and Cerco, 2010). Testing remains difficult regardless of the integration strategy used (Ramamoorthy et al., 1992). As we have seen, there are many potential sources of error in designing and implementing software to facilitate data-exchange within integrated modeling systems, including matching of variables, conversion of units, use of metadata to perform semantic operations, and translation of space and time dimensions. Identifying and reporting specific sources of error (debugging) in integrated systems is also a challenge.

Integration of complex data and models is likely to propagate uncertainty throughout the model chain (Dubois et al., 2013) and uncertainty tends to increase as more models are chained together (Geller and Turner, 2007). Quantifying and communicating uncertainty in a model chain is challenging but crucial if we want to ensure the integration output will be accepted by the larger user community. In addition, time-stepping and numeric integration methods selected for linked components have their own effect on error and uncertainty in integration output (Belete and Voinov, 2016).

From the perspective of software engineering, testing occurs at different stages of development including unit testing, integration testing, system testing, and acceptance testing (Luo, 2001). Unit testing is the lowest level and is performed to verify correct component function. Integration testing assesses whether interfaces between different components of a system are communicating correctly. System testing attempts to check end-to-end functioning of the system, and acceptance testing is done to deliver the system to end users. We must also consider emergent properties that appear when we link components into integrated systems (Ammann and Offutt, 2008). Best practices indicate software testing can be improved by characterizing the Why, How, How Much, What, Where, and When (Bertolino, 2007):

- Why – Are we looking for errors or we are trying to improve the usability of the user interface?
- How – How do we select test samples?
- How Much – How big should the sample be that we will use for testing?
- What – Are we going to perform unit test, component/subsystem test, or integration test?
- Where – Are we going to test the simulated environment or the final context?
- When – At which stage of the product life cycle will we perform the testing?

Testing does not always get the appropriate emphasis due to time or cost constraints For instance, Holzworth et al. (2015) points that the agricultural modeling community fails to give appropriate attention to testing. However, some frameworks have applied various testing approaches. While the naming was different, they provided certain testing functionalities to evaluate the system or its components. For example, FRAMES provided a “testbed” with error archiving functions (Whelan et al., 2014), from which users can closely examine system errors. Similarly, TIME provided a generic “test bench” (Rahman et al., 2003) in which model-component developers can test and calibrate a model. In ModCom (Hillyer et al., 2003) the framework distribution incorporates a “test suit” that covers all functionalities of the framework, and which is mainly aimed at debugging the framework libraries. In APSIM (Holzworth et al., 2014), automated testing is incorporated into the system to improve stability and robustness; whenever a modified version of code is submitted, automated testing runs the test suit.

When integrating models, a test plan should be part of development. The test plan should clearly outline the scope of testing and incorporate test cases, the list of functionalities to be tested, and by whom and when the testing will be performed. Depending on the context, testing can be done by developers, software testers, or end users. OpenMI testing, for example, was done in two phases – first by the design team and then by external experts (Moore and Tindall,

[9] Resource Description Framework - https://www.w3.org/RDF/.
[10] Web Ontology Language - http://www.w3.org/TR/owl-features/.
[11] Simple Knowledge Organization System - http://www.w3.org/2004/02/skos/.
[12] SPARQL Query Language for RDF - http://www.w3.org/TR/rdf-sparql-query/.

2005). Testing was performed by eight universities and companies with the aim of migrating and linking different models with the provided tools and guidelines (Gregersen et al., 2007). Testing of integration can be done with bottom-up or top-down approaches (Myers et al., 2011). Bottom-up testing begins with testing system components at the lowest level and progresses to higher-level combinations of components. In top-down, testing begins with the highest-level components and continues to individual components at lower levels. Once the testing strategy is selected, testing procedures can be automated which has advantages of saving time and repeatability. The challenge is that it may not be possible to automate everything we would like to test, and automated testing cannot always replace manual testing (Berner et al., 2005). This requires us to respond to questions like: How do we scientifically know that it works and that we did not create a false positive? For example, SIAT's developers used automated unit testing for code but not for the GUI (Verweij et al., 2010), the reason being that they found automated unit testing technology for GUIs is immature.

6. Discovery, accessibility and ease of use of integrated systems

Discoverability and accessibility of a model is significant in model integration. Despite the large number of modeling components, users generally lack tools to locate them. And if discovered, the next challenge is to gain operational access and evaluate their relevance to the user's problem (Booth et al., 2011). Finally, ease of use is important when deciding about model incorporation into a user's integrated system. For example, ESMF has been developed with a goal of enhancing ease of use, portability, and reuse in coupling climate and weather prediction models (da Silva et al., 2003; DeLuca et al., 2012). One of design goal of ModCom (Hillyer et al., 2003) is to provide a visual tool that enables model linkage and execution. Similarly, the TIME modeling framework (Rahman et al., 2003) includes a 'visually rich user interface' and reusable data handling and visualization components for integration of models. In the SEAMLESS-IF integration framework, the graphical user interface is designed to assist users in performing integrated assessment for alternative policy options (Van Ittersum et al., 2008). The GUI offers easy access and display of system functionalities and outputs.

Preparing software for easy use requires knowing the needs of target user groups of the framework. For example, the Biophysical Model Applications (BioMA[13]) framework was developed to rapidly bridge model development from prototypes to operational applications with re-usable components. In this case, target users are both application users and advanced users. Ease of use features for application users relate to running and viewing available model simulations, while ease of use features for advanced users include creating and developing models.

With the advance of web technology, presenting model interfaces and integration frameworks in web pages has become more popular since it significantly increases model discoverability, accessibility and usability. Web-based user interfaces that target specific communities – e.g. scientists, policy makers, and the public – improve access to data and models and facilitate model reuse. "Apps for the environment"[14] developed by the U.S. Environmental Protection Agency is a good example. The introduction of a browser-based Web Modeling Tool[15] (WMT) by CSDMS to replace the older desktop-based Component Modeling Tool (CMT) also points in this direction. Improving the WMT's ease of use is a focus of current development. Likewise, Systo,[16] a web-based model whose technical design is based on the desktop-based Simile,[17] InsightMaker[18] and STELLA,[19] is a successful effort to make a model more accessible and easier to use.

Providing models via the web is not without challenges. As Brooking and Hunter (2013) point out, these challenges include managing large volumes of data, complexity of the required software platform, and computational demand for model execution are barriers that limit sharing and re-use of models over the web. Similarly, Nativi et al. (2013) point out developing a flexible architecture to link distributed components, minimizing interoperability agreements, optimizing performance, and securing availability in the long term are challenges in presenting models on the web.

Presenting models using web pages also creates a challenge since model usage is constrained by how the web page presents it. For example, users may want to display model output as part of another application, which leads to providing a "model as a service", making it available as a web service (Geller and Melton, 2008; Geller and Turner, 2007; Roman et al., 2009). This approach makes models and their output more accessible, more comparable, more scalable, and can be implemented using a variety of approaches (Nativi et al., 2013). One example of such a platform is eHabitat (Dubois et al., 2013) which is part of the European Union Digital Observatory for Protected Areas (DOPA[20]) project. In eHabitat, models and data sources are presented as a pool of OGC WPS services; using a web-based interface, users can select and assemble them like Lego blocks. Another example is the iPlant cyberinfrastructure (Goff et al., 2011) which provides several biology-related applications as a web service API so bioinformatics experts and software developers can embed them in their tools.

Some modeling frameworks use purposely-designed semantic technology features to improve discoverability of models and related resources. For example, Galaxy (Goecks et al., 2010) – a computational framework for life sciences – uses tagging (labeling) to describe resources. During simulation, the system automatically tracks input datasets, tools used, parameter values, output datasets, and a series of analysis steps that it stores as history. The user can add annotations about analysis steps. The authors report that tagging supports reproducibility of experiments. Similarly, the system biology data and model platform - SEEK (Wolstencroft et al., 2015) – provides a web-based environment for day-to-day collaboration and public access. It uses the Resource Description Framework (RDF) to store metadata of resources that facilitate data and model exploration. All of these efforts indicate that incorporation of purposely-designed semantic technology features can improve discoverability of models beyond simple search operations.

7. Discussion and conclusions

Our goal was to identify existing strategies and recommend additional ones that facilitate efficient reuse and interoperability of science-software components to enable cost-effective construction and execution of integrated multi-disciplinary modeling systems. We organized the review around the model integration process that includes pre-integration assessment, preparation of models for

[13] BioMA - http://bioma.jrc.ec.europa.eu/index.htm.
[14] Apps for the environment - http://www.epa.gov/mygreenapps/.
[15] CSDMS WMT - http://csdms.colorado.edu/wiki/WMT_information.
[16] Systo - http://www.systo.org/.
[17] Simile - http://www.simulistics.com/.
[18] InsightMaker - https://insightmaker.com/.
[19] STELLA - http://www.iseesystems.com/softwares/Education/StellaSoftware.aspx.
[20] The Digital Observatory for Protected Areas - http://dopa.jrc.ec.europa.eu/.

integration, orchestration, data interoperability, and testing. We highlight various methods employed by developers to achieve each process step. The following observations and conclusions emerged from this review. We expand this in the final section with recommendations intended to foster a community-wide conversation to improve implementation of the overall process, and design and deployment of modeling components in particular.

1) There is wide variation in strategies for implementing and documenting model integration across the development community. The process is driven by a science-based need to explore processes, problems, and solutions related to the environment that uses software-based tools to express the knowledge and execute modeling workflows. Science-based needs determine the requirements for the integrated software system. The variation in strategies is largely related to computer-based technologies (hardware and software) rather than differences in science-based requirements.
2) Frameworks that facilitate construction and orchestration of modeling workflows are complex, requiring significant resources to build and a steep learning curve to understand and utilize them for those outside the development and targeted user groups. We found that the literature describing the overall process and frameworks, in particular, varies widely in aspects of the integration process and frameworks that are discussed and in the level of detail in those discussions.
3) There is wide variation in strategies for orchestrating execution of workflows within the frameworks, and preparing modeling components for assimilation into the frameworks. With few exceptions, current practice is to design and implement integrated modeling systems to satisfy requirements of individual projects and development/user group preferences. This strategy has resulted in a large number of frameworks, but little reuse of the modeling components (and, thus, the embedded science) among the frameworks.
4) Data interoperability involves interpreting and converting data produced by one component to satisfy content and format requirements of components that will consume it. Again, there is wide variation in how this is implemented. Methods include explicit and hard-coded mapping of variables, use of metadata that describes data attributes (units, min/max range), use of controlled vocabularies and ontologies to ensure semantic consistency, and intermediate components designed to manipulate data (translate, interpolate, aggregate, disaggregate) to resolve dimensional conflicts (spatial/temporal grid conflicts). Such variation in component design creates significant barriers to interoperability and sharing of science, expressed as software components, among frameworks.
5) Testing of integrated modeling systems verifies operational integrity of the system design as implemented, i.e., the system correctly formulates and processes the overall workflow from system inputs, to component-level data exchanges, to generation of results. Testing is particularly challenging in integrated systems because of the increased potential for error and error propagation. The reviewed literature paid relatively little attention to details on this topic. It appears that formal testing (with documentation) is often limited due to resource constraints and some developers conduct and document testing from the component-to the system-level. Systems with limited testing are generally operated by the developers only, while systems with more formal testing are easier to transfer to users. This lack of documentation of testing limits acceptance of the system by developers outside the original team.
6) Discoverability, accessibility and ease of use are fundamental requirements for users. For existing systems, discoverability is typically limited to local knowledge within a group, the literature, and Google searches. As stated above, the literature often does not provide sufficient detail to determine important characteristics of the software required to make a decision about pursuing its reuse. On a positive note, we see a movement toward the web as a platform for deployment of modeling components which appears motivated by an emerging desire to address interoperability by making models available as a service, thus eliminating issues of incompatible computer languages and operating systems. As a secondary benefit, model discoverability is also enhanced with web-based deployment.

Our overall conclusion is that integrated modeling can be significantly enhanced if the global community of developers specifically focuses on cross-framework reuse and interoperability in design and implementation strategies. This will take us beyond sharing science knowledge via literature and move us toward sharing this knowledge more efficiently and cost-effectively in computer-based software form. Based on our review, we conclude that this is not only possible but that it is already occurring, although in an ad hoc rather than an organized manner. In the next section we make several recommendations to increase awareness of these possibilities and focus on specific efforts.

8. Recommendations

Using the perspective gained from this review we organized a list of recommendations for moving forward as a global community in developing integrated modeling systems. Note that the recommendations do not include details such as specific standards that should be utilized or developed, but rather where standards apply and what they should include (i.e., the science content). We chose this approach despite the fact that specific standards already exist (e.g., OpenMI, OGS WPS) because we believe it is more important to come to a community-wide awareness of the benefits of reuse and interoperability and to increase participation in them. Our recommendations reflect the software engineering concept of "separation of concerns" and the science-based concept of ontologies. Separation of concerns focuses on developing software components that perform a specific, limited set of functionalities. As more functionality is packaged in a single piece of software, its flexibility and use (and reuse) becomes more and more constrained in an integrated context. Ontologies are a structured way to represent knowledge about a domain; they define and describe concepts (things) included within a knowledge domain, as well as their related properties and relationships.

We begin by identifying three principal elements of integrated modeling systems: user interface, model integration infrastructure and tools (i.e., frameworks), and science-based components. We recommend properties of each and the communication-based relationship between them. Each element can be further separated into lower-order elements, properties, and relationships.

8.1. User interface

The user interface should be "thin" and focus on specific problems or classes of problems that give users the ability to select or construct a scientific workflow involving data, modeling, and tool components relevant to the user-specified problem; and visualize and process results of workflow execution. Workflows should accommodate at least two levels of complexity: first, a straightforward linking of multiple data, model, and tool components to enable both feed forward and feedback interoperability; and second, it should accommodate special "system"-level functionality such as data acquisition and harmonization tools, ability to perform

system-wide sensitivity and uncertainty analysis, Monte Carlo simulation, and analytical tools for synthesizing intermediate and endpoint data processing. These workflows define the requirements for design of the model integration framework and its components.

Users should have access to a repository of components and sufficient metadata about them to determine which ones are relevant to a user-stated problem and are interoperable. Metadata about model components (aka meta-models) should be standardized to ensure machine-readability and processing. The repository may be local (to an organization or a development group, for example) but, ideally, would be linked to a global collection of socio-environmental science software components.

Output of the user interface should be a standalone (and, ideally, standards-based) description of the workflow to be executed. The interface should also provide access to tools that consume and process results of the integrated modeling execution.

8.2. Model integration

The model integration framework implements workflow produced via the user interface. Important elements of the integration process, as gleaned from the review, include execution management, semantic mediation, dataset conversion, and error handling. Integration tools or complete frameworks may be designed to optimize model system execution on high performance computers, parallelization of execution (e.g., leveraging the cloud), and web-based integration.

8.3. Science components

Science components should be designed and implemented in a framework-independent manner. We encourage individual scientists to publish software in a way that can be discovered, understood, and integrated within any user interface and model integration system. Essential features associated with the design of components to meet these high-level requirements include:

a. An explicit API. At the highest level, it enables initializing, running, and closing functions. At a secondary level, it describes and enables programmatic access to science knowledge and functionality expressed in the component.
b. Metadata for input/output/internal variables that describe at least the:
   i. Meaning of variables, including assumptions made about variables and processes
   ii. Measure/units unambiguously
   iii. Valid range of values, if appropriate
c. In the context of a) and b), the content should be described using a controlled vocabulary and, eventually, a full ontology to describe relevant concepts and relationships.
d. Software is open source and available via public access version control systems such as gitHUB.

To develop the global repository, components must be published with science content and functionality to enable discovery, understanding, and integration. Their publication must include metadata that fully describe inputs, outputs, and internal workings of the component (aka black box). This will require controlled vocabularies and an ontological framework for describing data and modeling science. To achieve this, the modeling community should reinforce collaboration on developing and improving standards which can be used across frameworks. Adoption of the OpenMI Standard v2.0 as an OGC standard is an example of one such effort.

Disclaimer

The views expressed in this article are those of the author(s) and do not necessarily represent the views or policies of the U.S. Environmental Protection Agency.

Acknowledgment

G. F. Belete and A. Voinov were supported by COMPLEX – Knowledge Based Climate Mitigation Systems for a Low Carbon Economy Project, EU-FP7-308601. We are grateful to two anonymous reviewers for very useful comments. The authors thank Fran Rauschenberg of the US EPA for her valuable contributions.

References

Ammann, P., Offutt, J., 2008. Introduction to Software Testing. Cambridge University Press.
Argent, R.M., 2004. An overview of model integration for environmental applications—components, frameworks and semantics. Environ. Model. Softw. 19, 219–234.
Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L., Parker, S., Smolinski, B., 1999. Toward a common component architecture for high-performance scientific computing. In: High performance distributed computing, 1999. Proceedings. The Eighth International Symposium on. IEEE, pp. 115–124.
Athanasiadis, I.N., Janssen, S., 2008. Semantic mediation for environmental model components integration. Inf. Technol. Environ. Eng. 1.
Belete, G.F., Voinov, A., 2016. Exploring temporal and functional synchronization in integrating models: a sensitivity analysis. Comput. Geosci. 90, 162–171.
Berner, S., Weber, R., Keller, R.K., 2005. Observations and lessons learned from automated testing. In: Proceedings of the 27th International Conference on Software Engineering. ACM, pp. 571–579.
Bertolino, A., 2007. Software testing research: achievements, challenges, dreams, 2007 future of software engineering. IEEE Comput. Soc. 85–103.
Booth, N.L., Everman, E.J., Kuo, I.L., Sprague, L., Murphy, L., 2011. A web-based decision support system for assessing regional water-quality conditions and management Actions1. JAWRA J. Am. Water Resour. Assoc. 47, 1136–1150.
Brooking, C., Hunter, J., 2013. Providing online access to hydrological model simulations through interactive geospatial animations. Environ. Model. Softw. 43, 163–168.
Bruggeman, J., Bolding, K., 2014. A general framework for aquatic biogeochemical models. Environ. Model. Softw. 61, 249–265.
Butterfield, M.L., Pearlman, J.S., Vickroy, S.C., 2008. A system-of-systems engineering GEOSS: architectural approach. Syst. J. IEEE 2, 321–332.
Castronova, A.M., Goodall, J.L., Elag, M.M., 2013. Models as web services using the open geospatial consortium (ogc) web processing service (wps) standard. Environ. Model. Softw. 41, 72–83.
Chang, H., Lee, K., 2004. Applying web services and design patterns to modeling and simulating real-world systems. In: International Conference on AI, Simulation, and Planning in High Autonomy Systems. Springer, pp. 351–359.
Crnkovic, I., Chaudron, M., Larsson, S., 2006. Component-based development process and component lifecycle. In: Software Engineering Advances, International Conference on. IEEE, 44–44.
da Silva, A., DeLuca, C., Balaji, V., Hill, C., Anderson, J., Boville, B., Collins, N., Craig, T., Cruz, C., Flanigan, D., 2003. The Earth system modeling framework. In: 3rd NASA Earth Science Technology Conference.
David, O., Ascough, J., Lloyd, W., Green, T., Rojas, K., Leavesley, G., Ahuja, L., 2013. A software engineering perspective on environmental modeling framework design: the object modeling system. Environ. Model. Softw. 39, 201–213.
Debayan, B., 2011. Component Based Development-application in Software Engineering. Indian Statistical Institute.
DeLuca, C., Theurich, G., Balaji, V., 2012. The Earth System Modeling Framework, Earth System Modelling-volume 3. Springer, pp. 43–54.
Dubois, G., Schulz, M., Skøien, J., Bastin, L., Peedell, S., 2013. eHabitat, a multi-purpose web processing service for ecological modeling. Environ. Model. Softw. 41, 123–133.
Erl, T., 2008a. SOA Design Patterns. Pearson Education.
Erl, T., 2008b. Soa: Principles of Service Design. Prentice Hall Upper Saddle River.
Fowler, M., 2010. Domain-specific Languages. Pearson Education.
Frakes, W.B., Kang, K., 2005. Software reuse research: status and future. IEEE Trans. Softw. Eng. 529–536.
Gavrilovska, A., Kumar, S., Raj, H., Schwan, K., Gupta, V., Nathuji, R., Niranjan, R., Ranadive, A., Saraiya, P., 2007. High-performance hypervisor architectures: virtualization in hpc systems. In: Workshop on System-level Virtualization for HPC (HPCVirt). Citeseer.
Geller, G.N., Melton, F., 2008. Looking forward: applying an ecological model web to assess impacts of climate change. Biodiversity 9, 79–83.
Geller, G.N., Turner, W., 2007. The model web: a concept for ecological forecasting. In: Geoscience and Remote Sensing Symposium, 2007. IGARSS 2007. IEEE

International. IEEE, pp. 2469–2472.
Goecks, J., Nekrutenko, A., Taylor, J., 2010. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol. 11, 1.
Goff, S.A., Vaughn, M., McKay, S., Lyons, E., Stapleton, A.E., Gessler, D., Matasci, N., Wang, L., Hanlon, M., Lenards, A., 2011. The iPlant collaborative: cyberinfrastructure for plant biology. Front. Plant Sci. 2, 34.
Goodall, J.L., Robinson, B.F., Castronova, A.M., 2011. Modeling water resource systems using a service-oriented computing paradigm. Environ. Model. Softw. 26, 573–582.
Goodall, J.L., Saint, K.D., Ercan, M.B., Briley, L.J., Murphy, S., You, H., DeLuca, C., Rood, R.B., 2013. Coupling climate and hydrological models: interoperability through web services. Environ. Model. Softw. 46, 250–259.
Granell, C., Díaz, L., Gould, M., 2010. Service-oriented applications for environmental models: reusable geospatial services. Environ. Model. Softw. 25, 182–198.
Gregersen, J., Gijsbers, P., Westen, S., 2007. OpenMI: open modelling interface. J. Hydroinform. 9, 175–191.
Grimm, V., Augusiak, J., Focks, A., Frank, B.M., Gabsi, F., Johnston, A.S., Liu, C., Martin, B.T., Meli, M., Radchuk, V., 2014. Towards better modelling and decision support: documenting model development, testing, and analysis using TRACE. Ecol. Model. 280, 129–139.
Harris, G., 2002. Integrated assessment and modelling: an essential way of doing science. Environ. Model. Softw. 17, 201–207.
Hill, C., DeLuca, C., Suarez, M., Da Silva, A., 2004. The architecture of the earth system modeling framework. Comput. Sci. Eng. 6, 18–28.
Hillyer, C., Bolte, J., van Evert, F., Lamaker, A., 2003. The ModCom modular simulation system. Eur. J. Agron. 18, 333–343.
Holst, N., 2013. A universal simulator for ecological models. Ecol. Inf. 13, 70–76.
Holst, N., Belete, G.F., 2015. Domain-specific languages for ecological modelling. Ecol. Inf. 27, 26–38.
Holzworth, D.P., Huth, N.I., 2011. Simple software processes and tests improve the reliability and usefulness of a model. Environ. Model. Softw. 26, 510–516.
Holzworth, D.P., Huth, N.I., Zurcher, E.J., Herrmann, N.I., McLean, G., Chenu, K., van Oosterom, E.J., Snow, V., Murphy, C., Moore, A.D., 2014. APSIM–evolution towards a new generation of agricultural systems simulation. Environ. Model. Softw. 62, 327–350.
Holzworth, D.P., Snow, V., Janssen, S., Athanasiadis, I.N., Donatelli, M., Hoogenboom, G., White, J.W., Thorburn, P., 2015. Agricultural production systems modelling and software: current status and future prospects. Environ. Model. Softw. 72, 276–286.
Huhns, M.N., Singh, M.P., 2005. Service-oriented computing: key concepts and principles. Internet Computing, IEEE 9, 75–81.
ISO/IEC (International Organization for Standardization/International Electrotechnical Commission), 2011. Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models (IEC))ISO/IEC 25010:2011.
Jakeman, A.J., Letcher, R.A., Norton, J.P., 2006. Ten iterative steps in development and evaluation of environmental models. Environ. Model. Softw. 21, 602–614.
Janssen, S., Athanasiadis, I.N., Bezlepkina, I., Knapen, R., Li, H., Domínguez, I.P., Rizzoli, A.E., van Ittersum, M.K., 2011. Linking models for assessing agricultural land use change. Comput. Electron. Agric. 76, 148–160.
Janssen, S., Ewert, F., Li, H., Athanasiadis, I.N., Wien, J., Thérond, O., Knapen, M., Bezlepkina, I., Alkan-Olsson, J., Rizzoli, A.E., 2009. Defining assessment projects and scenarios for policy support: use of ontology in integrated assessment and modelling. Environ. Model. Softw. 24, 1491–1500.
Keating, B.A., Carberry, P.S., Hammer, G.L., Probert, M.E., Robertson, M.J., Holzworth, D., Huth, N.I., Hargreaves, J.N., Meinke, H., Hochman, Z., 2003. An overview of APSIM, a model designed for farming systems simulation. Eur. J. Agron. 18, 267–288.
Knapen, R., Janssen, S., Roosenschoon, O., Verweij, P., De Winter, W., Uiterwijk, M., Wien, J.-E., 2013. Evaluating OpenMI as a model integration platform across disciplines. Environ. Model. Softw. 39, 274–282.
Laniak, G.F., Olchin, G., Goodall, J., Voinov, A., Hill, M., Glynn, P., Whelan, G., Geller, G., Quinn, N., Blind, M., 2013. Integrated environmental modeling: a vision and roadmap for the future. Environ. Model. Softw. 39, 3–23.
Leimbach, M., Jaeger, C., 2005. A modular approach to integrated assessment modeling. Environ. Model. Assess. 9, 207–220.
Luo, L., 2001. Software Testing Techniques. PA 15232, 19. Institute for software research international Carnegie mellon university Pittsburgh.
Madni, A.M., Sievers, M., 2014. Systems integration: key perspectives, experiences, and challenges. Syst. Eng. 17, 37–51.
Moore, A., Holzworth, D., Herrmann, N., Huth, N., Robertson, M., 2007. The Common Modelling Protocol: a hierarchical framework for simulation of agricultural and environmental systems. Agric. Syst. 95, 37–48.
Moore, R.V., Tindall, C.I., 2005. An overview of the open modelling interface and environment (the OpenMI). Environ. Sci. Policy 8, 279–286.
Myers, G.J., Sandler, C., Badgett, T., 2011. The Art of Software Testing. John Wiley & Sons.
Nativi, S., Mazzetti, P., Geller, G.N., 2013. Environmental model access and interoperability: the GEO Model Web initiative. Environ. Model. Softw. 39, 214–228.
Papazoglou, M., 2008. Web Services & SOA: Principles and Technology. Pearson Education.
Peckham, S., 2014. The CSDMS Standard Names: Cross-domain Naming Conventions for Describing Process Models, Data Sets and Their Associated Variables. International Environmental Modelling and Software Society (iEMSs), San Diego, California, USA.
Peckham, S.D., Hutton, E.W., Norris, B., 2013. A component-based approach to integrated modeling in the geosciences: the design of CSDMS. Comput. Geosci. 53, 3–12.
Pessoa, R.M., Silva, E., van Sinderen, M., Quartel, D.A., Pires, L.F., 2008. Enterprise interoperability with SOA: a survey of service composition approaches. In: Enterprise Distributed Object Computing Conference Workshops, 2008 12th. IEEE, pp. 238–251.
Rahman, J., Seaton, S., Perraud, J., Hotham, H., Verrelli, D., Coleman, J., 2003. It's TIME for a new environmental modelling framework, MODSIM 2003 International Congress on Modelling and Simulation. In: Modelling and Simulation Society of Australia and New Zealand Inc. Townsville, pp. 1727–1732.
Ramamoorthy, C., Chandra, C., Kim, H., Shim, Y., Vij, V., 1992. Systems integration: problems and approaches, Systems Integration, 1992. ICSI'92. In: Proceedings of the Second International Conference on. IEEE, pp. 522–529.
Rekaby, A., Osama, A., 2012. Introducing integrated component-based development (ICBD) lifecycle and model. Int. J. Softw. Eng. Appl. 3, 87.
Renner, S., 2001. A Community of Interest Approach to Data Interoperability. Federal database colloquium, p. 2.
Rizzoli, A.E., Donatelli, M., Athanasiadis, I.N., Villa, F., Huber, D., 2008. Semantic links in integrated modelling frameworks. Math. Comput. Simul. 78, 412–423.
Roman, D., Schade, S., Berre, A., Bodsberg, N.R., Langlois, J., 2009. Model as a service (MaaS). In: AGILE Workshop: Grid Technologies for Geospatial Applications, Hannover, Germany.
Schmolke, A., Thorbek, P., DeAngelis, D.L., Grimm, V., 2010. Ecological models supporting environmental decision making: a strategy for the future. Trends Ecol. Evol. 25, 479–486.
Schut, P., Whiteside, A., 2007. OpenGIS Web Processing Service. OGC Project Document.
Sutton, M.A., Reis, S., Bahl, K.B., 2009. Reactive nitrogen in agroecosystems: integration with greenhouse gas interactions. Agric. Ecosyst. Environ. 133, 135–138.
Tanenbaum, A.S., Van Steen, M., 2007. Distributed Systems. Prentice-Hall.
Van Ittersum, M.K., Ewert, F., Heckelei, T., Wery, J., Olsson, J.A., Andersen, E., Bezlepkina, I., Brouwer, F., Donatelli, M., Flichman, G., 2008. Integrated assessment of agricultural systems–A component-based framework for the European Union (SEAMLESS). Agric. Syst. 96, 150–165.
Verburg, P.H., Eickhout, B., van Meijl, H., 2008. A multi-scale, multi-model approach for analyzing the future dynamics of European land use. Ann. Regional Sci. 42, 57–77.
Verweij, P., Knapen, M., De Winter, W., Wien, J., Te Roller, J., Sieber, S., Jansen, J., 2010. An IT perspective on integrated environmental modelling: the SIAT case. Ecol. Model. 221, 2167–2176.
Villa, F., Athanasiadis, I.N., Rizzoli, A.E., 2009. Modelling with knowledge: a review of emerging semantic approaches to environmental modelling. Environ. Model. Softw. 24, 577–587.
Voinov, A., Cerco, C., 2010. Model integration and the role of data. Environ. Model. Softw. 25, 965–969.
Voinov, A.A., 2008. Systems Science and Modeling for Ecological Economics. Academic Press.
Wainer, G.A., Madhoun, R., Al-Zoubi, K., 2008. Distributed simulation of DEVS and Cell-DEVS models in CD++ using Web-Services. Simul. Model. Pract. Theory 16, 1266–1292.
Wang, X., Chan, C.W., Hamilton, H.J., 2002. Design of knowledge-based systems with the ontology-domain-system approach. In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering. ACM, pp. 233–236.
Whelan, G., Kim, K., Pelton, M.A., Castleton, K.J., Laniak, G.F., Wolfe, K., Parmar, R., Babendreier, J., Galvin, M., 2014. Design of a component-based integrated environmental modeling framework. Environ. Model. Softw. 55, 1–24.
Woodside, M., Franks, G., Petriu, D.C., 2007. The Future of Software Performance Engineering, Future of Software Engineering, 2007. FOSE'07. IEEE, pp. 171–187.
Wolstencroft, K., Owen, S., Krebs, O., Nguyen, Q., Stanford, N.J., Golebiewski, M., Weidemann, A., Bittkowski, M., An, L., Shockley, D., 2015. SEEK: a systems biology data and model management platform. BMC Syst. Biol. 9, 33.
Yu, L., 2011. A Developer's Guide to the Semantic Web. Springer Science & Business Media.