Environmental Modelling & Software 94 (2017) 112–126

Contents lists available at ScienceDirect

Environmental Modelling & Software

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

# Designing the Distributed Model Integration Framework – DMIF

Getachew F. Belete*, Alexey Voinov, Javier Morales

*University of Twente, ITC, 7500 AE Enschede, The Netherlands*

ARTICLE INFO

*Article history:*
Received 22 November 2016
Received in revised form
28 March 2017
Accepted 2 April 2017
Available online 14 April 2017

*Keywords:*
Integrated modeling
Web services
Wrapping
Service oriented architecture
Semantic mediation

ABSTRACT

We describe and discuss the design and prototype of the Distributed Model Integration Framework (DMIF) that links models deployed on different hardware and software platforms. We used distributed computing and service-oriented development approaches to address the different aspects of interoperability. Reusable web service wrappers were developed for technical interoperability models created in NetLogo and GAMS modeling languages. We investigated automated semantic mapping of text-based input-output data and attribute names of components using word overlap semantic matching algorithms and using an openly available lexical database. We also incorporated automated unit conversion in semantic mediation by using openly available ontologies. DMIF helps to avoid significant amount of reinvention by framework developers, and opens up the modeling process for many stakeholders who are not prepared to deal with the technical difficulties associated with installing, configuring, and running various models. As a proof of concept, we implemented our design to integrate several climate-energy-economy models.



**Software/data availability**

*DMIF*

| | |
|---|---|
| Software Developer | Getachew F. Belete |
| Address | University of Twente, ITC, 7500 AE Enschede, Netherlands |
| Tel | +31-684-838-692 |
| E-mail | getfeleke@gmail.com |
| First available | 2015 |
| Hardware requirements | 1 GHz CPU 512 MB RAM |
| Software requirements | Windows 7 Or Newer |
| Availability | Open source |
| Cost | Free |
| Program language | C# and Java |
| Program size | 250 MB |
| Software Access | http://owsgip.itc.utwente.nl/projects/complex/index.php/2-uncategorised/46-model-integration-framework |

## 1. Introduction

Models are simplifications of reality and are developed with the objective to understand a concept or system, to analyze what its future states and trends may look like, and if possible to come up with appropriate management decisions and mitigation or adaptation strategies. Thousands of computer models have been developed. However, complex problems such as climate mitigation require interdisciplinary knowledge and data from many domains including climate, hydrology, energy, economy, land use, behavioral sciences, etc. The complex and interrelated nature of such real-world problems requires holistic system-of-systems thinking (Laniak et al., 2013). In this case it is not practical, perhaps impossible, to construct a single model that could simulate such complex processes (Gijsbers and Gregersen, 2005). Integration of models and tools may be a solution (Stoorvogel, 1995). It also reuses existing models and it is faster and less expensive than reengineering legacy systems (Madni and Sievers, 2014).

During integration, we should understand that component models can be developed using different assumptions and semantics, different methodologies, tools and techniques, may operate at different temporal and spatial scales, may have different levels of complexity, etc. Integration of models assumes linking such heterogeneous models together into an operational model chain (Knapen et al., 2013), or rather a network with loops and feedbacks, where one model down the chain can also feed input back into a model above. This requires addressing interoperability at technical, semantic, and dataset levels (Belete et al., 2017). Based

* Corresponding author.
*E-mail addresses:* getfeleke@gmail.com (G.F. Belete), aavoinov@gmail.com (A. Voinov), j.morales@utwente.nl (J. Morales).

http://dx.doi.org/10.1016/j.envsoft.2017.04.003
1364-8152/

on this we define a model integration framework as a set of software libraries, classes, and components that enable one to manage technical, semantic, and dataset aspects of interoperability.

Integration of models requires mediation that goes beyond merging information and data that use different schemas. Computer-based models contain sophisticated knowledge statements, which may be represented in different ways. Due to this, integration of models requires understanding of the different contexts of the models involved. It also requires mechanisms to modify the incoming information so that it fits to the assumptions, conditions (rules), and processes in the data-receiving model. Best practice indicates that this can be achieved by providing dedicated components or modules that handle context-based interpretation and semantic mediation. For example, such a mediator component is known as the Knowledge Manager in SEAMLESS (Athanasiadis and Janssen, 2008) or the Semantic Discovery Broker in eHabitat web processing service (Dubois et al., 2013). One of the objectives of modeling is to provide information to decision makers. Despite the large number of available models, decision makers lack easy access to models to evaluate alternative scenarios (Booth et al., 2011). Models that do not require installation of special software, and that do not need special training, are more accessible but rare. With the advance of web technology, presenting models and integration frameworks on the web, i.e. when a model can run in a normal web browser requiring no additional software installation, is becoming more popular since it can significantly increase model accessibility and sharing. However, the volume of data involved, complexity of the software platform required, computational demand for model execution are identified as barriers that prevent sharing and reusing models over the web (Brooking and Hunter, 2013).

Although availing models through web pages is useful, usage of models will still be constrained by the way the web page presents the model. For example, users may want to directly access model output and display it as part of another application. This leads to the idea of providing a "Model as a Service" (Geller and Turner, 2007; Geller and Melton, 2008; Roman et al., 2009). Presenting models as web services has the benefits of making models and their outputs more accessible, easier for model comparison, scalable, and implementable using a variety of approaches (Nativi et al., 2013; Peckham and Goodall, 2013). A web service developed using one programming language can be accessed and consumed by applications using a number of other programming languages, i.e. without requiring intermediary language interoperability tools. The Interoperable nature of services gives the opportunity to integrate legacy models by presenting them as web services (Goodall et al., 2013; Granell et al., 2010).

Currently, one of the main challenges facing the integrated environmental modeling community is lack of interoperability across independently built systems (Goodall et al., 2011; Laniak et al., 2013). This means that besides improving accessibility of models and data, we also need a mechanism for integration across disciplines and models developed using different platforms. There is an increasing need for methodology that enables to build a system-of-systems (Butterfield et al., 2008) by connecting independent component systems and making them interoperable.

The context in which a certain model is used can vary significantly. Besides, a model can be linked to a number of models in different ways. Users have their own integration requirements. Integration scenarios identified and designed by a certain group of modelers or developers may not satisfy integration requirements of the whole user community. Even one particular user can come up with a number of integration requirements. On the other hand, only a small subset of end users may have the programming skills needed to modify the source code of integration frameworks in accordance to their requirements. Due to this, there is a need for tools in which users can select certain models and link them without the need for additional design, coding, debugging, etc.

In this paper, we present the design methodology of a Distributed Model Integration Framework (DMIF). We present the approach used to convert heterogeneous and independently built models into plug-and-play components, and the mechanisms used to automate some of semantic mediation tasks. In addition, we present a case study in semantic mediation using semantic matching algorithms and lexical databases. We also introduce interfaces that enable runtime access and integration of web service based models without the need of additional coding. We also discuss the limitations of such approach. After all, models are always built for a purpose and there is no guarantee that when we reuse a model we will be using it in the same way as intended by the original model construction. This is how so called 'integronsters' (Voinov and Shugart, 2013) are created, which we certainly want to avoid. User mediation and pre-integration assessment are the only ways that we can safeguard ourselves from unintended misuse of models. We explore how integration interfaces and semantic mediation can assist in such user guided model evaluation.

The remainder of the paper is organized as follows: Section 2 presents the design criteria of model integration frameworks. Section 3 describes the architecture of DMIF. Section 4 provides information on how we provided for technical interoperability by presenting models as web services. Section 5 describes the methodology for building the semantic mediation module of model integration frameworks. This section also presents algorithms for semantic matching of text-based input-output data and for searching of components using attribute names of components. Section 6 introduces runtime access and integration of web service based models at the GUI level. Section 7 discusses additional issues that we should consider in developing integration frameworks, and we give our conclusions in Section 8.

## 2. Design criteria for DMIF

Our aim is to provide a verifiable design of a model integration framework that helps to link multidisciplinary heterogeneous models distributed over various software and hardware platforms in a meaningful way. Design should follow user requirements, and it should be verifiable against those user requirements. There are many factors that a model integration framework should consider (Belete and Voinov, 2014, 2016; Belete et al., 2014, 2017). There is no ideal integration technique, which best suites all kinds of model integration requirements. However, as a guideline we know that an integration framework should consider the following design criteria. It should:

- support models developed using different programming languages,
- include models hosted on different hardware and software platforms,
- access models located anywhere on the Internet,
- keep independently developed models autonomous,
- avoid reinventing whenever reuse of resources is possible,
- provide functionalities as reusable components,
- be extensible without disturbing the existing system,
- be accessible on the web.

Given these criteria, our design focuses on interoperability of heterogeneous models. To realize this we need to establish a few well-known dependencies (Rosen et al., 2008) among such models that enable them to exchange data and to collaborate.

## 3. The DMIF framework

### 3.1. Background

To meet model integration requirements, we considered Best Practices of System Integration, Enterprise Application Integration, Distributed Systems, and Software Design Patterns (Erl, 2008a; Gamma et al., 1994; Tanenbaum and Van Steen, 2007). We observed that web services have a good potential to transform independently built models into plug-and-play components. This is because web services are platform independent, self-contained, interoperable applications that are accessible over the web. Moreover, any piece of existing code can be transformed into a network available service (Papazoglou, 2008). Besides, those independently built service-based models can be linked into a system-of-systems by using Service-Oriented Approach (SOA) of software development. SOA is a design approach in which services are linked together based on the following principles: participating services are expressed using standardized metadata, services are loosely coupled, reusable, autonomous, distributed across different platforms, discoverable, and composable regardless of their size (Erl et al., 2009). Due to the nature of SOA, it inherently addresses such integration issues as loose coupling, interoperability, and platform independence (Erl, 2008b).

### 3.2. Architecture of the framework

The Distributed Model Integration Framework (DMIF) is designed with the aim to handle linking of various heterogeneous computer-based models. A prototype of DMIF can be accessed from the COMPLEX[1] project model repository page. In this section we start describing DMIF by presenting its architecture (Fig. 1), which defines the logical organization of the system in terms of software components (Fowler, 2003). Fig. 1 depicts the high-level view of the integration framework. Besides, software architecture demonstrates how concepts in the problem domain are handled by the solution domain (Butterfield et al., 2008).

The architecture of DMIF follows a layered structure based on principles of separation of concerns and distributed computing. Following the separation of concerns principle (Laplante, 2007), the system is a composition of a group of logically related elements. As a result, it relieves models from managing integration related tasks, for example, interoperability and data conversion. Following the distributed computing paradigm (Tanenbaum and Van Steen, 2007), a system is a collection of subsystems deployed on different heterogeneous platforms that communicate with each other by exchanging messages (Papazoglou, 2008). Distributed systems hide the fact that the resources and processes are distributed across multiple computers, and the system presents itself as if it were hosted only in a single computer. Due to this, models can be anywhere on the Internet (Nativi et al., 2013).

In the context of model integration frameworks, distributed computing paradigm provides the following benefits:

- Models can be developed using different programming languages and can be deployed in different operating systems,
- Participating models are kept autonomous,
- Models can run concurrently on different machines using multithreading techniques – which has significant performance gain when linking model with 'longer' execution times (Wainer et al., 2008),
- Distributed systems are scalable, more resources, such as, models, databases, files can be added later on.

As shown in Fig. 1, DMIF consists of three layers. Integration begins from the *User Interface layer*. This layer is used to select the models that will be integrated; to specify the execution workflow, representing the correct order in which the models have to be connected; and, to provide all required input data for the execution of the workflow. The user inputs are converted into appropriate data structure and then passed to the Integration layer. Based on the specified workflow, the *Technical Integration Module* is responsible for accessing the models and ontologies that reside in the *Resource layer*, and which can be located anywhere on the Internet. Translation and interpretation of different data represented by participating models is done using the *Semantic Mediation Module*. Workflows can be stored as part of the *Integration Rule* for future use. The same applies for and semantic mapping and translation since they can be stored into the existing *Semantic Mapping*. The *Resource Layer* represents a number of available models in the system including their corresponding wrappers. A model may read its inputs from a file, a database, or directly from its user interface. Wrapper web services associated with the models make them web-enabled and allow the Integration layer to interoperate with them by exchanging messages. However, using web services as wrappers has some disadvantages since they use verbose text-based protocol to exchange messages, which means larger message sizes than for example binary protocols. Besides, web service based approach has challenges due to limited reliability of the communication between services over the Internet. Detailed discussion of different parts of the architecture is presented in the following sections.

## 4. Wrapping and technical interoperability

### 4.1. Enabling technical interoperability

Technical integration of models requires automating data exchange between models and making them jointly executable. As Knapen et al. (2013) indicate that technical integration of models can be achieved in five different approaches: soft linking, using scripts, proprietary monolithic, proprietary loose linking, and with open standards. As mentioned earlier, the strategy we used to build the model integration framework is W3C open Web Service standards.[2] We assume that all participating models and tools can be presented as web services. Then the interaction between models is managed by the technical integration module using SOA principles. The technical integration module is the core for the whole integrated system, since it is responsible for establishing the connection between models, for routing, and messaging. It is the composition engine (Alonso et al., 2004) of service-based models.

Presenting models as web services will make them componentized. Here a software component is defined as an independent unit of functionality, with a well-defined interface, and which internal states are not externally observable (Clemens et al., 1998). Componentization has the benefits of reusability, interoperability, limited dependency on other components, and dynamic linking capability during runtime (Peckham et al., 2013; Rizzoli et al., 2008). Since we are integrating already existing models, we prefer to avoid rewriting them when presenting them as web services. We turn models into web services by wrapping them, and transforming them into plug-and-play autonomous components. A

[1] DMIF - http://owsgip.itc.utwente.nl/projects/complex/index.php/2-uncategorised/46-model-integration-framework.

[2] http://www.w3.org/standards/webofservices/.

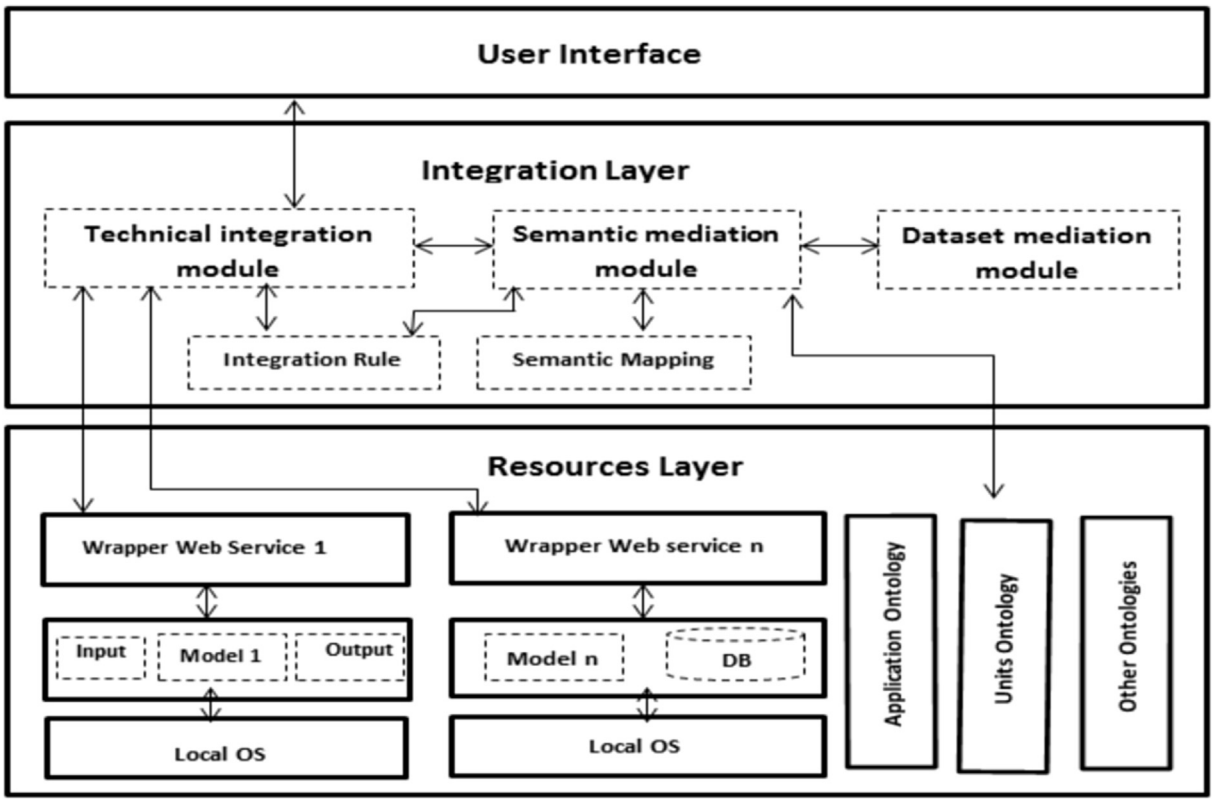


**Fig. 1.** Architecture of the distributed model integration framework - DMIF. The models together with their input-output files/databases and ontologies are based on the principle of reuse of available resources. Wrapper web services are developed as means to link models to the integration layer.

wrapper is an interface that encapsulates a model and provides access to the model, its functions and data, in a required way. For example, in the Community Surface Dynamics Modeling System (CSDMS), the BMI[3] wrapper is a set of functions named initialize, update, finalize, run_model, get_attribute, etc. that provide information about the model and run it.

Depending on the ease of implementation, wrapper web services can be developed by using different programming languages. As a proof of concept for our design approach, we implemented wrappers for a case study that links three models from the COMPLEX[4] project. The first model is GCAM,[5] a dynamic-recursive partial equilibrium model that represents the economy, the energy sector and land use. The model is developed using C++ and it uses a number of XML based configuration files. The second model is EXIOMOD[6]- a macroeconomic model developed using GAMS[7] programming language. The third model is NIROO (Niamir and Filatova, 2015), an agent based model that focuses on households' energy use and potential behavioral changes, and developed using NetLogo.[8] The objective of linking the three models is to build a system-of-systems that could simulate the effects of implementation of United Nations Framework Convention on Climate Change (UNFCCC[9]) policy scenarios. The UNFCCC policy scenarios are implemented in GCAM and the effect can be analyzed on EXIOMOD and NIROO (Belete et al., in revision). However, the three models cannot interoperate naturally. To create interoperability between these three models we developed C#-based web service wrappers for the GCAM and EXIOMOD models. The wrapper for EXIOMOD was built on top of the GAMS.NET API. This API manipulates GAMS based models from C#. Similarly, a Java-based web service wrapper was built for NIROO model on top of NetLogo-Java library (NetLogo.jar). The challenge here is that models are developed independently, they are not designed to interoperate with other models (Nativi et al., 2013), and they do not have knowledge about other member models (Butterfield et al., 2008). Yet for integration purposes, we have to come up with an interface that will help a model to act as a plug-and-play component.

By choosing the web service approach we offer the ultimate flexibility for using the models. As long as they are wrapped as web services they become available for further integration. Web service based wrapping addresses our main design goals: it supports models located anywhere on the Internet, developed using different programming languages, hosted on different hardware and software platforms, and it keeps independently developed models autonomous. The amount of work in developing wrappers depends on how much we need to adapt the existing model. If the model is built using a modular approach, developing wrappers may be straightforward; but if a complex model is implemented as a monolithic function, we may also need to modify the model code. Still, from the software development viewpoint, as long as the wrapper produces a web service, we do not need to impose any additional requirements on its functionality.

What still remains problematic is the descriptive part of the model and how it will be delivered by the web service. In fact, in addition to wrapping the model code to expose it as a web service, we need to put extra effort in properly documenting and exposing

[3] BMI - http://csdms.colorado.edu/wiki/BMI_Description.
[4] COMPLEX - http://owsgip.itc.utwente.nl/projects/complex/.
[5] GCAM - http://www.globalchange.umd.edu/gcam/.
[6] EXIOMOD - http://owsgip.itc.utwente.nl/projects/complex/index.php/2-uncategorised/23-exiomod.
[7] GAMS - https://www.gams.com/.
[8] NetLogo - https://ccl.northwestern.edu/netlogo/.
[9] UNFCCC - http://unfccc.int/meetings/copenhagen_dec_2009/items/5264.php.

the model. Machine-readable description of web services, the WSDL[10] file, is designed to provide metadata information about the service. The contents of the WSDL file mainly focuses on technical issues such as data type definitions, set of operations or functions provided by the service, binding information, etc. However, the WSDL can also be used to provide semantic information about models that will be used during integration. For example, WSDL has optional Document element aimed for human readable documentation. This element can be used to incorporate standardized model metadata that will facilitate searching and integration of component models.

### 4.2. Web service orchestration

In DMIF, models do not interact with each other directly. Instead, the technical integration module handles the interaction between models. Service orchestration (Erl, 2008a; Papazoglou, 2008) is one of the main tasks of the technical integration module. It serves as a mediator (Gamma et al., 1994) that encapsulates the interaction between models. Based on the design principle of separation of concern, models are not expected to maintain information about 'where' other models are hosted and to support a protocol for 'how to' communicate with them. If we require models to maintain such information, then:

- a model should maintain a list of other models that could be linked to it,
- whenever a new model joins the framework, existing models should incorporate new information,
- when there is change in location or some modification in the interface of a certain model these changes should be reflected in the data maintained by existing models.

All this results in information duplication and hinders scalability.

In our approach, only the technical integration module maintains reference information about participating models. By using this information, the technical integration module can access both local and remotely hosted models. It also coordinates the communication between models so that the system gives a sense of a single aggregate service. Addition or removal of a component does not affect other existing components and this makes the framework extensible.

The workflow of orchestration of services varies depending on the type and number of models involved (Yang et al., 2012; Zhao et al., 2012). This is because communication between models could be unidirectional, bidirectional or iterative. As a result, each case of integration will have its own workflow. However, as shown in Fig. 1, in all cases integration starts at the user interface level. The user interface passes user inputs together with the sequence of execution to the technical integration module. The technical integration module then executes the models and receives their output. It also handles interaction with the semantic mediation and dataset conversion modules.

This design approach allows users to select the components they prefer to couple at the GUI level and this makes them responsible for the final decisions made about candidate models and their linkage mechanisms. User defined coupling presents significant challenges since users are free to couple any available component models, which could possibly give birth to 'integronsters' (Voinov and Shugart, 2013), i.e., products that are valid in terms of component coupling but useless as models. To avoid this, during design time, based on the logical and contextual information of participating components we can identify input-output data exchanges between them, and make sure that the accompanying semantic mediation and dataset conversion functionalities are in place. We can document all such validated coupling potentials in a file and use them as integration rules (Belete et al., 2014). During run-time, the technical integration module will validate the user-defined coupling against the integration rule before proceeding to service orchestration. If the proposed coupling is not available in the integration rule, the system will alert about the possibility of an 'integronster' but the user can override the suggestion and continue. On the other hand, this checking can become automated once adequate model documentation standards become accepted and implemented. Only when wrapped models contain sufficient semantic metadata to allow the system to determine whether the composition results make sense, the integration process will allow some level of automation. Until then the user will have to be involved for the final checking and quality assurance of the proposed integration schemes.

The other major issue we need to consider during orchestration is optimization of performance of the integrated model. In reality, many processes that are represented by integrated models happen in parallel. However, it is common practice to represent such parallel processes using sequential execution of simulation models. For better performance of simulation, our design approach supports parallel execution of models using the multi-threading technique (Tanenbaum and Van Steen, 2007) – i.e. the technical integration module calls the participating services as different threads of a single process but each of those services will run in parallel on their hosting machines. Currently, almost all modern computers have multi-core CPUs, and the parallel execution approach gives the opportunity for effective utilization of available processing resources (David et al., 2013). In distributed systems, the multi-threading technique has a significant effect in the performance of the system since subsystems can be executed in parallel on different computers. Especially if each of the processes are taking longer execution times, multi-threading will have rewards in performance (Wainer et al., 2008).

The other point in relation to performance optimization is the size of data to be exchanged, which could be dependent on the way we develop wrappers. A model may return several variables as output. In designing wrapper services, we can include either all these variables or only some of them as the output of the wrapped model. If the wrapper includes only the required variables for the data exchange then we can minimize performance overhead due to exchange of large size of data over the network. However this at the same time can restrict the application of wrappers making them coupling specific: when a model is coupled with yet another model a new wrapper will be needed if additional model output is to be exposed.

## 5. Semantic mediation

### 5.1. What is involved in semantic mediation

Semantic mediation is a means to link knowledge represented in different ways. In integrating models, semantic mediation is done based on the contextual information of participating models, and metadata information of models is the main source of contextual information. Depending on the extent of the semantic mediation automation, it requires providing a certain level of intelligent translation of concepts. This may include gathering and analyzing information based on the contexts involved. The challenges in semantic mediation are:

[10] WSDL - https://www.w3.org/TR/wsdl.

- How to minimize hard coding in semantic mediation?
- How to provide standardized semantic mediation?
- How to accommodate the vast semantic difference among different models?

In semantic mediation, hard coding or hard wiring should be minimized (Uschold, 2003) since hard-coded semantic mediation is fragile whenever there is change in representation of model attributes, data structure, and data itself. Hard coding can be minimized by standardizing the way we represent concepts and build structures that maintain the semantic relations between different concepts. Depending on the specific context, the level of detail in which a concept is represented varies. Based on this, such artifacts as Glossary, Lexicon, and Ontology have been used to organize concepts (Buccella et al., 2009; De Nicola et al., 2009). Among these an ontology contains more organized information since it includes a hierarchy of concepts, relationships between concepts, and cardinality, which are not present in a glossary. Ontology preserves the semantics of the represented concept and this helps to eliminate the need for custom developed code for semantic mediation (Papazoglou, 2008). Since ontologies are based on a set of formal expressions upon which a computer can reason (Janssen, 2009), they also enable higher level of reuse (Wang et al., 2002). In general, ontologies together with semantic mapping and translation techniques are crucial to minimize hard coding in semantic mediation (Li et al., 2011; Uschold, 2003). If a multi-disciplinary ontology is available, the metadata of participating models can be mapped and translated using semantic processing techniques. However, building ontology to address semantic interoperability across multi-disciplinary models is not an easy task and requires concerted efforts of many people (Argent, 2004). Constructing a multi-disciplinary full-fledged ontology requires huge collaborative efforts among different disciplines. Again, we find a need for some model documentation standards and metadata for the provision of reusable and standardized data conversion functionalities.

In DMIF, we classified the semantic mediation task into two categories:

1) semantic matching, which includes translating text data and mapping of variable names and,
2) unit conversion.

### 5.2. Semantic matching in integration of models

We developed a semantic matching module for text data and variable names used in models. For both cases, first, we used the word overlap matching (Canhasi, 2013) or token-based matching (Cohen et al., 2003) algorithms that does not consider hierarchical semantic relationships between concepts, and then we used a lexicon database based matching algorithm that considers semantic relationships between concepts.

#### 5.2.1. The semantic matching algorithms

Suppose we are linking model 1 with model 2. Our aim is to match a list of available text-based input data/variable names in the second model, with a list of text-based output data/variable names from the first model. While in many models we still see variable names that are quite semantically meaningless, there is a trend towards choosing informative naming conventions (Peckham, 2014). First, let us consider the word overlap matching algorithm, which has two phases. The first phase is direct text matching. The algorithm tries to find a text that matches exactly, and if it finds a match then it will assign matching index value as 1, which means the two texts match 100%. The next phase is to do token-based matching for those texts that did not match directly. In this phase, the algorithm will try to find a number of matching words in the two texts under consideration, and it does the matching by using the following steps:

- split (tokenize) the first text into words,
- split (tokenize) the second text into words,
- count the number of matching words between the two texts,
- compute matching index

To compute the matching index, we use *Jaccard similarity coefficient* (Achananuparp et al., 2008; Sarawagi and Kirpal, 2004), since it measures the size of intersection between two given texts, it is suitable to represent the word level similarity measurement (Niwattanakul et al., 2013). The Jaccard similarity coefficient (J) for texts A and B is computed as:

$$J(A,\ B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad 0 \le J(A,\ B) \le 1$$

If $A$ and $B$ are both empty then $J(A,\ B) = 1$

Where the notation ∩ stands for intersection and ∪ is union between sets A and B.

However, the word overlap algorithm does not completely satisfy our needs, because it is common that different modelers can represent the same concept using different words and phrases, which have similarity in meaning. For example, suppose we want to check the similarity of 'car' and 'automobile' using the word overlap algorithm. The algorithm returns matching index of zero. Nevertheless, semantic mediation requires a matching algorithm that considers relations between texts based on meaning of words.

Due to this, we developed the second semantic matching algorithm that uses a lexicon database that considers hierarchical semantic relation between concepts. We used an openly available lexicon database called WordNet[11] that consists of over a hundred thousand words with meanings and a complex architecture of a network of words. In the database nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms called synsets, each expressing a distinct concept. The main relation among words in WordNet is 'being synonymous', e.g. car and automobile are synonymous. WordNet computes similarity between two words using the following steps:

- find shortest path length (in links/edges) between the synsets containing the compared words w1 and w2,
- find the depth of the first common subsumer (or superset) of both synsets,
- compute the similarity index based on the path length and depth information.

Like the word overlap algorithm, the similarity index value for WordNet based algorithm ranges between zero and one. For example, if we compute similarity between 'car' and 'automobile' it returns one.

In general, applying the two semantic matching algorithms has challenges. The first challenge is that it is difficult to do the matching for acronyms and abbreviations that are commonly used by the modeling community. The second challenge is to decide where to put the margin for similarity index values after which we can consider matching as 'valid'. We observed that, in some cases matching index value of 0.8 may not be good enough to consider

[11] WordNet - https://wordnet.princeton.edu/.

the matching as 'valid' and in some other cases an index value of 0.3 might be enough. For better results, setting the margins requires the decision of the human user. As test cases, we applied these two semantic matching algorithms in matching text data and variable names of models.

### 5.2.2. Semantic matching of text data

**Case study 1:** The first case study focuses on matching text-based input-output data to link an Airports information web service[12] with a Global Weather[13] model that is available as a web service. This example is a case of the integration of data as a service with a model as service. The objective of the linking is to generate an enhanced list of airports in a country, which includes current weather information such as *visibility, sky conditions, pressure, temperature, relative humidity, and dew point*. This can be done since the Airports web service provides a list of airports in a country, and the Global Weather web service provides weather information of a given city in a country. Name of cities are text data to be passed from one web service to the next web service. Consider, for example, the case of 'Netherlands', as being an input text data given to the two web services. Table 1 shows that the two web services represent the names of cities in different ways.

Then, given the list of names of cities, we can handle the semantic mediation by direct mapping between the names. The problem with this approach is that we are not controlling these web services while the owners can modify the content of the data anytime. Therefore, whenever the content of data is modified we have to modify the mapping. Due to this hard-coded semantic mapping is fragile. The solution is to develop an automatic semantic matching algorithm that manages the semantic linking between the text values.

The result of word overlap matching (Fig. 2) shows that only 25% of the names match using direct text matching, and 43.75% of the names can be matched using word overlap matching technique. In this case, we observed that matching results (Table 2) with index $\geq 0.25$ can be considered as 'valid' for our purpose. On the other hand, when we consider the results of matching using WordNet, we get matching for ENSCHEDE TWENTE, SCHIPOL, and UTRECHT SOESTERBRG, which is certainly wrong and was not identified using word overlap matching algorithm due to the difference in spelling. However, such results as 1) BERGEN OP ZOOM WONSDRECHT match with De Kooy with index = 0.625 and 2) ROTTERDAM match with Eindhoven with index = 0.899999 are not 'valid' for our purpose. In terms of semantics, the matching could be correct since they are cities in the same country but it does not solve our problem. This means, 12.5% of the matching output is misleading. This shows that for matching short text based input-output data, usage of a lexicon database may not necessarily improve the results of the semantic mediation. This could be mainly because a short text provides very limited contextual information.

### 5.2.3. Semantic matching of variable names of models

**Case study 2**: The second case study was to match variable names of components in the CSDMS model repository. The objective is: given a component (data-receiving component) that requires external input data find other possible components (data-providing components) that could provide input data for it. The CSDMS uses standard names (Peckham, 2014) to document component attributes, and we consider only models which are documented following the standard names. As a first step, we put all input variable names of components in one category and output variable names in another category. Then, we apply both word overlap matching and WordNet based matching algorithms to find components that could provide data to a given component. In this case, we applied the WordNet based algorithm to investigate to what extent standardization will minimize the need for complex semantic processing.

**Table 1**
Text data for semantic matching.

| List of airports in a country (source airports web service) | List of cities in a country (source weather web service) |
|---|---|
| AMSTERDAM SCHIPHOL | Amsterdam Airport Schiphol |
| BERGEN OP ZOOM WONSDRECHT | AntillesFlamingo Airport, Bonaire |
| BREDA GILZE RIJN | AntillesHato Airport, Curacao |
| DEN HELDER DE KOOY | AntillesJuliana Airport, Saint Maarten |
| EINDHOVEN | AntillesRoosevelt Airport Saint Eustatius |
| ENSCHEDE TWENTE | Groningen Airport Eelde |
| GRONINGEN EELDE | Leeuwarden |
| LEEUWARDEN | Maastricht Airport Zuid Limburg |
| LEIDEN VALKENBURG | Rotterdam Airport Zestienhoven |
| MAASTRICHT | De Bilt |
| ROTTERDAM | Deelen |
| SCHIPOL | De Kooy |
| THE HAGUE | Eindhoven |
| UDEN VOLKEL | Gilze-Rijen |
| UTRECHT SOESTERBRG | Soesterberg |
| WOENSDRECHT AB | Twenthe |
| | Valkenburg |
| | Volkel |
| | Vlieland |
| | Woensdrecht |

The results of the semantic matching (Table 3) show that for standardized variable names the usage of lexicon-based semantic matching does not provide better results than the simplistic direct text matching, which clearly indicates the benefits of standardized names when searching for relevant components. In this specific case, we observed that, although lexicon-based semantic matching index values are close to 1 the texts we are comparing represent different concepts. For example, variables channel_inflow_end_water__discharge and sea_water_surface_wave__period return matching index of 0.904. These matching values are misleading to perform automated mediation.

We also considered searching for data-providing components over the full metadata of the components instead of doing the semantic matching within processed data, i.e. within a specific list of input and output variable names. Note that the CSDMS provides an API that returns standardized metadata of components in JSON[14] (JavaScript Object Notation) format. The metadata information from the APIs consists of data beyond the list of input-output variable names of components. Due to this, we need a functionality to extract the list of input and output variable names from the full metadata of the component. To realize the search for components on the full metadata we used the following algorithm:

- Read the whole metadata of the data-receiving component,
- Read the whole metadata of all other components in the repository,
- Extract input variable names of the data-receiving component from its metadata information,
- Extract the list of output variable names of all components from the metadata information,
- Do semantic matching,

[12] Airports information web service - http://www.webservicex.net/airport.asmx.
[13] Weather web service - http://www.webservicex.net/globalweather.asmx.
[14] JSON - http://www.json.org/.

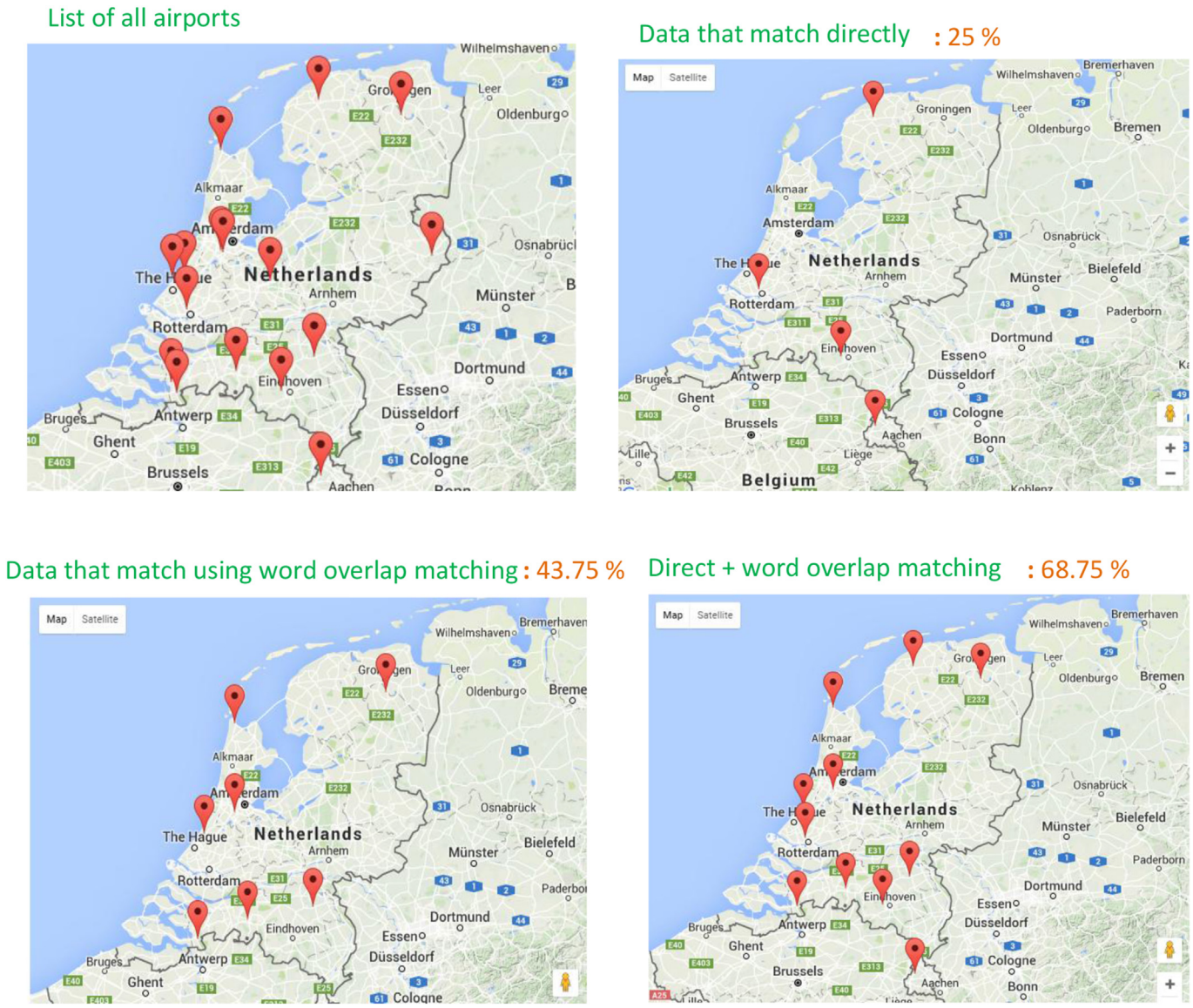


**Fig. 2.** Screen shot of output of direct semantic matching for Netherlands.

**Table 2**
Results of semantic matching using word overlaps algorithm and using WordNet.

| Airport name | Word overlaps matching | | WordNet based matching | |
|---|---|---|---|---|
| | Matching city | Matching index | Matching city | Matching index |
| AMSTERDAM SCHIPHOL | Amsterdam Airport Schiphol | 0.66666666 | Amsterdam Airport Schiphol | 0.863999 |
| BERGEN OP ZOOM WONSDRECHT | – | 0 | De Kooy | 0.625 |
| BREDA GILZE RIJN | Gilze-Rijen | 0.25 | Gilze-Rijen | 0.759999 |
| DEN HELDER DE KOOY | De Kooy | 0.5 | De Kooy | 0.966666 |
| EINDHOVEN | Eindhoven | 1 | Eindhoven | 1 |
| ENSCHEDE TWENTE | – | 0 | Twenthe | 0.6833336 |
| GRONINGEN EELDE | Groningen Airport Eelde | 0.66666666 | Groningen Airport Eelde | 0.8000000 |
| LEEUWARDEN | Leeuwarden | 1 | Leeuwarden | 1 |
| LEIDEN VALKENBURG | Valkenburg | 0.5 | Valkenburg | 0.7333333 |
| MAASTRICHT | Maastricht Airport Zuid Limburg | 0.25 | Maastricht Airport Zuid Limburg | 0.5 |
| ROTTERDAM | Rotterdam Airport Zestienhoven | 0.33333333 | Eindhoven | 0.8999999 |
| SCHIPOL | – | 0 | Amsterdam Airport Schiphol | 0.5124999 |
| THE HAGUE | – | 0 | AntillesHato Airport, Curacao | 0.2119999 |
| UDEN VOLKEL | Volkel | 0.5 | Volkel | 0.7233332 |
| UTRECHT SOESTERBRG | – | 0 | Soesterberg | 0.6599999 |
| WOENSDRECHT AB | Woensdrecht | 0.5 | Woensdrecht | 0.6666666 |

**Table 3**
Semantic matching in the CSDMS repository for standardized model variable names (a) using direct matching algorithm, (b) using WordNet based matching algorithm.

(a)

| Data-receiving component -> Avulsion. Requires input data for variables: | Using Direct matching algorithm | | |
|---|---|---|---|
| | Data-providing component | Output variable name | Matching index |
| channel_inflow_end_bed_load_sediment__mass_flow_rate | river | channel_inflow_end_bed_load_sediment__mass_flow_rate | 1 |
| channel_inflow_end_water__discharge | river | channel_inflow_end_water__discharge | 1 |
| surface__elevation | cem | surface__elevation | 1 |
| | child | surface__elevation | 1 |
| | sedflux2d | surface__elevation | 1 |

(b)

| Data-receiving component –> Avulsion. Requires input data for variables: | Using WordNet based matching algorithm | | |
|---|---|---|---|
| | Data-providing component | Output variable name | Matching index |
| channel_inflow_end_bed_load_sediment__mass_flow_rate | child | bed_load__mass_flow_rate | 0.9228572 |
| | river | channel_inflow_end_bed_load_sediment__mass_flow_rate | 1 |
| channel_inflow_end_water__discharge | child | channel_water__discharge | 0.90875 |
| | river | channel_inflow_end_water__discharge | 1 |
| | waves | sea_water_surface_wave__period | 0.904 |
| | windwaves | sea_surface_water_wave__period | 0.904 |
| surface__elevation | cem | surface__elevation | 1 |
| | child | surface__elevation | 1 |
| | sedflux2d | bedrock_surface__elevation | 0.924 |
| | sedflux2d | surface__elevation | 1 |

- List components (together with the corresponding output variable names) that can possibly provide data to the data-receiving component.

The result of searching for components from the whole metadata (Fig. 3) shows that using the semantic matching algorithms we can search for other components that could provide data for a given component. In this case, we also see that applying the lexicon database does not improve the matching of standardized variable names, which confirms the 'Law of the Semantic Web' by Uschold (2003) – "the more agreement there is, the less it is necessary to have machine-processable semantics".

In general, the results of the semantic matching algorithms indicate that although we do not fully automate the semantic mediation process we can still provide useful information that could facilitate user guided semantic mediation.

### 5.3. Unit conversion in integrating models

Unit conversion is another semantic mediation task that we considered in DMIF. As part of our design criteria - avoid reinventing whenever reuse of resources is possible – we found that QUDT[15] or Quantities, Units, Dimensions and Data Types Ontologies is a good resource for unit conversion in semantic mediation. As indicated in its website, QUDT, developed by TopQuadrant[16] and NASA,[17] has the objective of improving the quality of software interfaces, web services, and data interoperability by providing consistent terms and constructs in defining attributes of datasets and messages. Besides, QUDT ontology is freely available under Creative Commons Attribution-Share Alike 3.0 United States License.[18] The QUDT ontology is available in both OWL and TURTLE formats (Yu, 2011) and these standardized data structures help to build generic semantic mediation functionalities.

The Units ontology is organized into categories of units, for example, SIUnit, SIDerivedUnit, DerivedUnit, NotUsedWithSIUnit, NonSIUnit. These categories are further organized into types, such as AngelUnit, ForceUnit, CurrencyUnit, MolarEnergyUnit, TimeUnit, etc. Besides, each unit has a number of attributes that will help to perform semantic translations. For example, the unit kilometerPerHour has attributes: Label = 'Kilometer per Hour', Abbreviation = 'km/hr', Symbol = 'km/hr', Type = 'LinearVelocityUnit', ConversionMultiplier = '0.2777777777777778e0'. This means that, to convert *km/hr* to the base unit *m/s* we should multiply it by the conversion multiplier.

For a given unit, the semantic mediation module queries the ontology and provides a list of possible matching units into which a given unit can be translated. Consider Fig. 4 for graphical demonstration of how we handled unit conversion using QUDT ontology. From the list of available units, assume that the first model is using kilometerPerHour to express a variable. Then, the semantic mediation module will search in QUDT for the Type attribute of the selected unit; in this specific case, kilometerPerHour belongs to the type LinearVelocityUnit. By using the fetched value of Type attribute (i.e. LinearVelocityUnit), the semantic mediation module will infer and provide a list of available LinearVelocityUnit type of units, for example, CentimeterPerSecond, FootPerHour, Knot, MeterPerSecond. Let us assume that the second model uses FootPerMinute to express linear velocity, which has a conversion multiplier of 0.00508e0. Then the semantic mediation module will automatically derive the conversionMultiplier from km/hr to ft/min by dividing the conversionMultiplier of the first model by that of the second model.

The semantic module uses dotNetRDF[19] API to query the QUDT ontology. The API is an Open Source. Net Library[20] that enables to work with RDF, SPARQL and the Semantic Web. The API provides full support for SPARQL 1.1 Query and Update together with inferencing capability. Due to these features, it is possible to extend the

[15] QUDT - http://www.qudt.org/.
[16] http://www.topquadrant.com/.
[17] http://www.nasa.gov/.
[18] https://creativecommons.org/licenses/by-sa/3.0/us/.
[19] http://dotnetrdf.org/.
[20] http://www.w3.org/2001/sw/wiki/DotNetRDF.

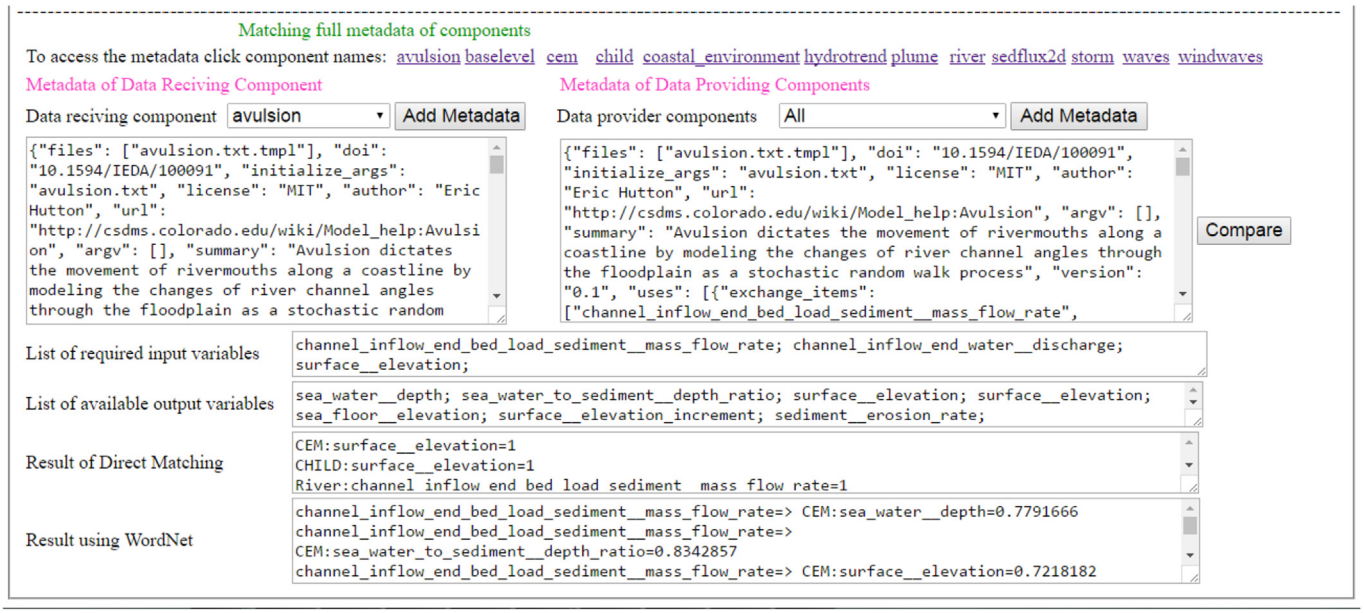


**Fig. 3.** Screen shot of the user interface for searching for components that could provide data to a certain component from the CSDMS standardized metadata of components.

semantic mediation module by adding a number of SPARQL queries.

## 6. Runtime integration of models

We define runtime integration as an integration method in which users can select and integrate models using a graphical user interface, on the fly, when needed. This can be achieved by following a two-step process: first, the models should be made available based on some standard, and second, there should be a 'generic' user interface that can link models and run simulations. Note that the 'generic' interface is built by assuming the defined standard. Based on this, we developed a prototype of a 'generic' interface that provides: 1) Runtime access interface (Fig. 5) - to run stand-alone web service based models, and 2) Runtime integration interface (Fig. 6) - to integrate web service based models without the need for additional design, coding, debugging, etc. To run stand-alone web service based models, the user needs to enter a URL of the service description (WSDL[21]) in the provided text box, then click the 'Get Available Methods' button. The system will list available functions of the web service under the 'Methods' tree-view control. If we select one of the listed functions, the corresponding required input variables will be displayed under 'Variables' control. Following this, we can select each of the input variables and set the required inputs to the model. In general, by using this interface, we can access different web service based models regardless of their underlying software and hardware platforms, and their location. The models may return simple or complex datasets and the output will be displayed in the user interface. For example, we accessed models from UV Index and Alert forecast service on EPA,[22] Global weather prediction services,[23] U.S. neighborhood level demographic services,[24] etc.

The runtime integration interface has the following main parts (Fig. 6):

1) Provide the URL of the WSDL of web services to be linked. Note that the participating services can be models or data sets organized in various ways, for example, sensor observation data (Regueiro et al., 2015). Individual web services can have one or more functions and the user has to select which functions are to be involved in the integration. Besides, the user can also provide input data or scenarios if there is any input of that kind that the model requires.
2) Access to a data conversion service if needed. If the output of one service can serve directly as input for the second service, then the user will skip this part. However, if data conversion is required and if the data conversion functionality is available as web service, then the user should provide the URL of the data conversion service. If data conversion service is not available, skilled users can build data conversion services and wider user community can reuse it.
3) The integration builder. In this part, the user should specify the workflow and input-output data exchange pattern between the services. The integration builder has two list or combo boxes that will be automatically populated when the user performs the operations described in steps (1) and (2). The first combo box is called *Output functions*, and lists the functions that produce output. It maintains the concatenated value of service name with semicolon followed by the method name. For example, a service named servicex with a method called methodx will be listed as servicex:methodx. If servicex has also an additional method called methody then this method will be listed as servicex:methody. The second combo box, called *Input variables*, maintains a list of input variables together with the corresponding method and service names. It contains a concatenated value of the service name, method name and input variable names. For example, if methodx of servicex has input variables varn and varm, then it will be presented as two entries servicex:methodx:varn and servicex:methodx:varm. The user can define the data exchange pattern between the services by matching the values listed in the combo boxes.

To see how a user can link models in the GUI, consider the following example. Assume that we are integrating model M1 with

[21] WSDL - https://www.w3.org/TR/wsdl.
[22] EPA UV alert service - https://www.epa.gov/enviro/web-services#hourlyzip.
[23] http://www.webservicex.net/ws/default.aspx.
[24] http://ws.cdyne.com/DemographixWS/DemographixQuery.asmx.

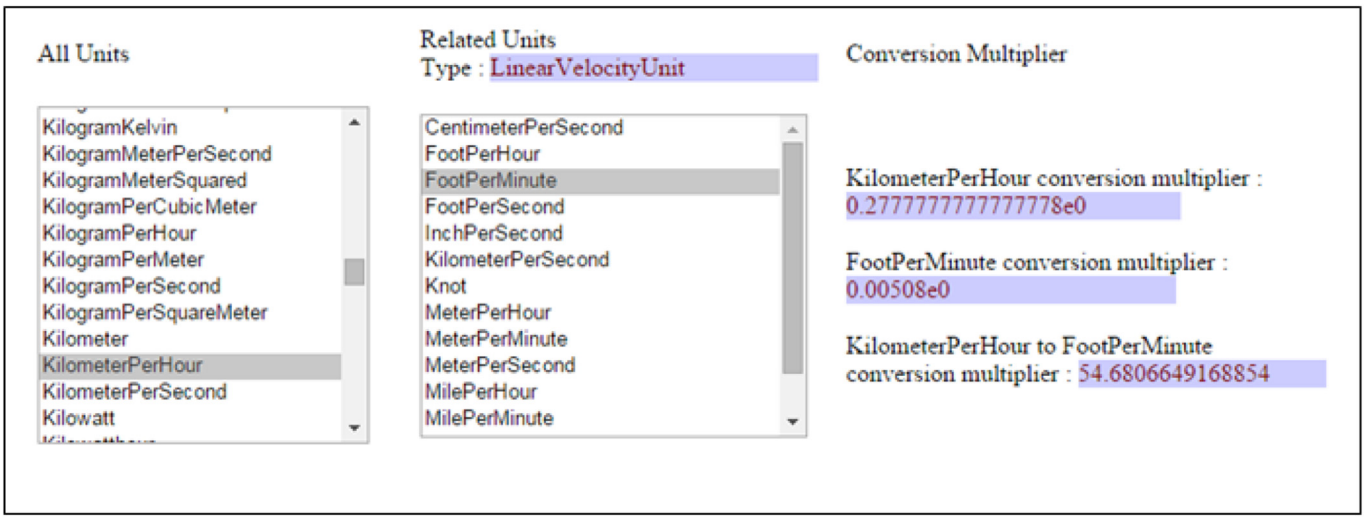


**Fig. 4.** Graphic representation of semantic mediation in converting units.

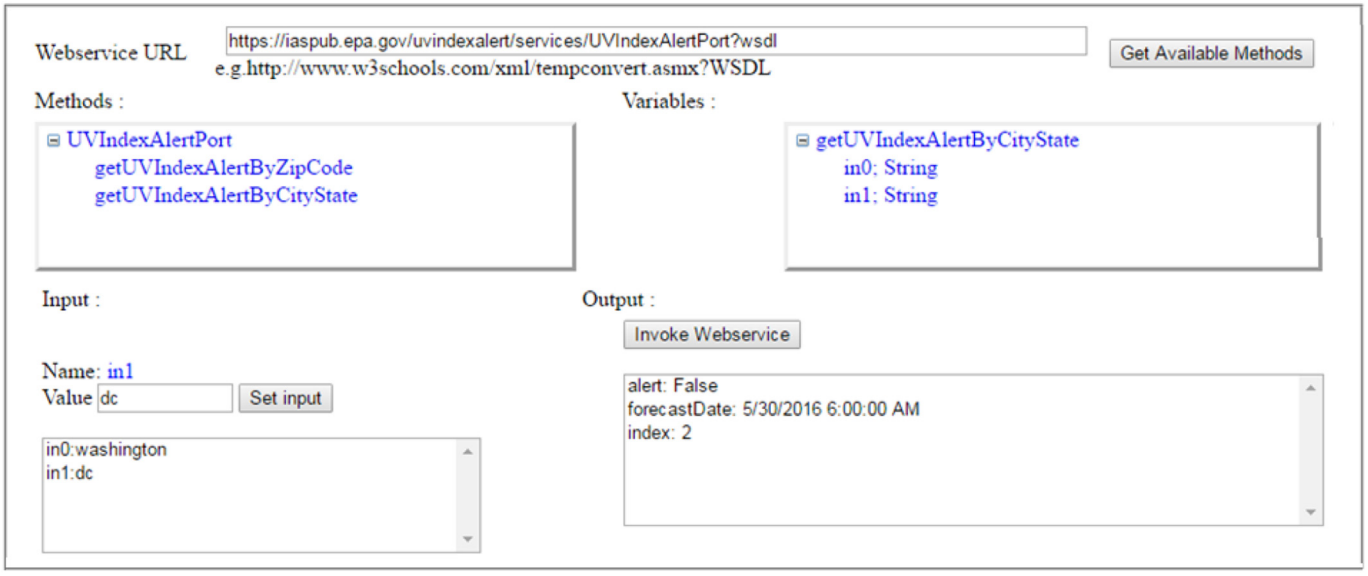


**Fig. 5.** User interface for runtime access of web service based models, the case of EPA's UV alert web service.

model M2 to simulate a certain scenario. For example, M1 is predator population dynamics model and M2 is the prey population dynamics model, so that the integration of M1 and M2 will produce the classical predator-prey model. Suppose M1 has a function named function1 with input variables var1 and var2, and after execution it produces the value of var1 (i.e. var1′) as an output. Similarly, M2 has input variables var3 and var4, and after execution it produces the value of var4 (i.e. var4′) as an output. Besides, suppose that var1 and var3 represent the same variable, except it is named differently, and similarly for var2 and var4. As we see in Fig. 7(a), when the integrated model runs for more than one cycle the outputs of M1 will be used as input for M1 and M2; and similarly the output of M2 will be used as input for M1 and M2. The user can define the linking by selecting the data-providing item from Output functions combo box and data-receiving item from the Input variables combo box and then click the 'Link' button, and this will generate data exchange commands as follows:

M1: function1 => M1: function1:var1
M1: function1 => M2: function2:var3
M2: function2 => M2: function2:var4
M2: function2 => M1: function1:var2

The listing represents data exchange pattern shown in Fig. 7(a). The first line in the expression means that 'output produced from function1 of model M1 will be used as input by variable var1 of function1 of model M1'. Similarly, the second line says that 'output produced from function1 of model M1 will be used as input by variable var3 of function2 of model M2'. Once the linking is defined, the user can set the number of cycles the models should run, and then run the models. In every simulation cycle, the system executes all the linkages defined by the user.

However, in most of the cases during integration the output of one model may require a conversion to the appropriate format before it is passed to the next model. Consider Fig. 7(b), where we include a data conversion web service, D, which has at least two

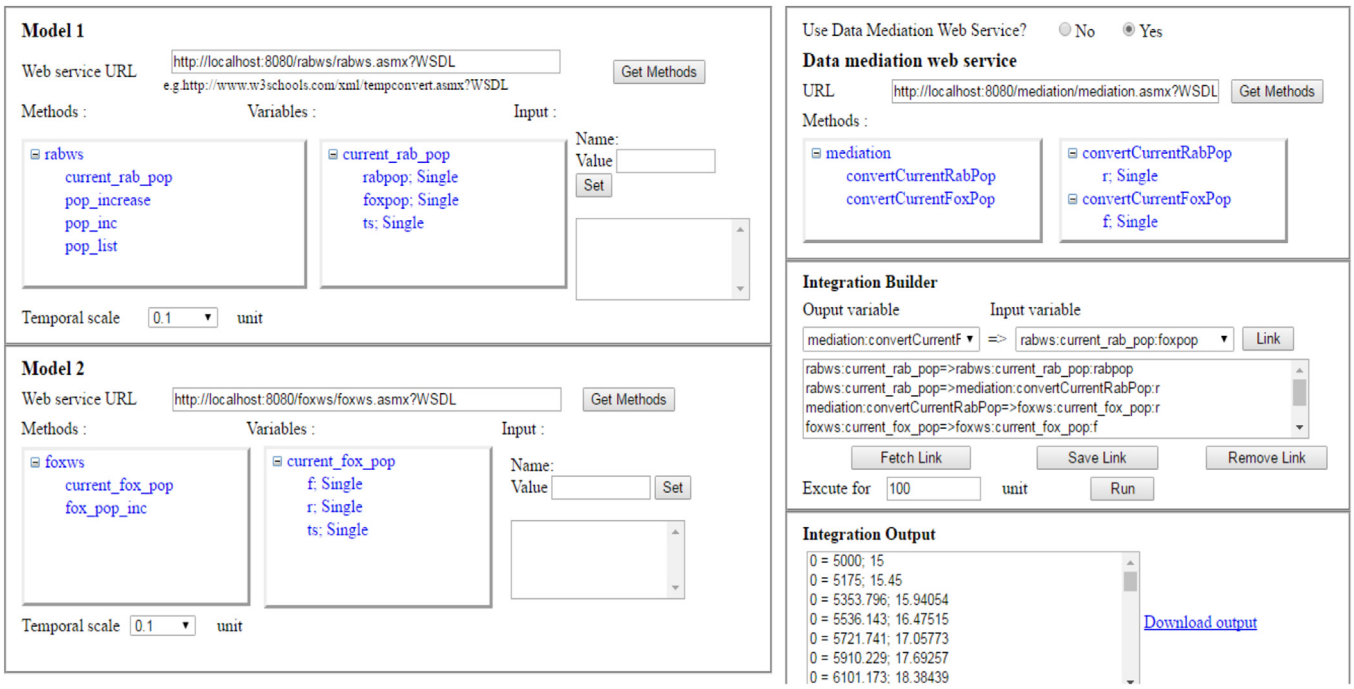


**Fig. 6.** User interface for runtime linking of web service based models.

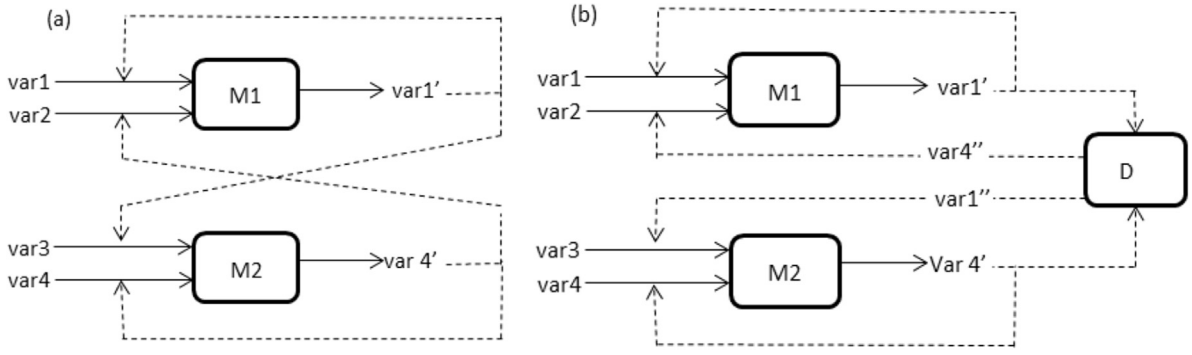


**Fig. 7.** Variable mapping in integrating model M1 with model M2. M1 takes inputs var1 and var2 and produces output var1′. M2 takes inputs var3 and var4 and produces output var4′. (a) Var1 and Var3 are the same except they are named differently. The same applies for var2 and var4 and the models are linked directly; (b) Data conversion is required. That is model D has a function that converts var1′ to var1″ and another function that converts var4′ to var4″.

functions, let us say function3 and function4. The purpose of function3 is to take output data from M1, and convert it so that it can be passed to M2. Similarly, function4 takes output from M2, converts it and so that it can be sent to M1. The user can define such kind of data exchange pattern from the user interface as follows:

M1: function1 => M1: function1:var1
M1: function1 => D: function3:var1
D: function3 => M2: function2:var3
M2: function2 => M2: function2:var4
M2: function2 => D: function4:var4
D: function4 => M1: function1:var2

In addition, the user can set the temporal scales for each model and the duration for which the integrated system will run. Based on these inputs all data exchange commands defined by the user will be executed. This kind of interfaces can resolve:

- access right issues since we do not need to acquire models and web service based models can be hosted on the owners' servers,
- the issues related to deployment and configuration of models on end users' computers,
- the issues related to the requirements of writing the code that links the participating models.

In addition, it will favor accessibility and reusability of models, and collaboration among modelers. The interface can be improved further by providing alternative data visualization features like maps, tables, and graphs. The usability will be certainly much improved if the user can select the list of output variables and the format in which the output will be displayed.

## 7. Discussion

Making existing independently developed models interoperable can be difficult (Geller and Turner, 2007; Nativi et al., 2013). To contribute towards a solution to this global challenge, first, we have set our design goals for a model integration framework and then we treated interoperability at different levels.

The main contribution of our work is that we designed a

distributed model integration framework that could link multi-disciplinary heterogeneous models possibly deployed on different hardware and software platforms. Given the URL of a web service based model, it can be incorporated into the DMIF integration framework regardless of its platform and location. As a result, a model available in a UNIX environment and another model operating under Windows can interoperate by using our design approach. A model developed with stakeholder participation using the user-friendliness of the NetLogo interface can be connected to some sophisticated climatic or economic models. Designing and developing applications that can integrate and run models on the web is a way forward for integrated environmental modeling (Laniak et al., 2013) and we consider our effort as a step in that direction.

In developing wrappers for models, we observed different challenges that require further careful consideration. Many of legacy models are developed as one 'main' function that read its inputs from a file, manipulates several variables, writes its outputs to file, and returns nothing. In such a case, the wrapper should invoke the 'main' function, but the list of input-output variables that the wrapper should implement varies depending on the specific context of the models involved in the coupling. There is also a possibility of the need to develop more than one wrapper for a single model or component. For example, in our case study we observed that, if we let the wrapper to return all model output then it would have its effect on system performance. Due to this, for example (in our case study that is described in section 4.1) in developing a wrapper for GCAM, even though GCAM produces several outputs, we designed it to return only two variables. This helped us to improve system performance. However, if we want to link GCAM with another model that requires access to other output variables of GCAM, then it may be better to develop a new wrapper function/method than modifying the existing one. In fact, in developing another wrapper for a model, we can reuse significant amount of code and we need to change only the variables that should be included as the model output.

The other identified challenge we observed in relation to wrapping and synchronizing the communication of models is error and exception handling. Error and exception can arise from the wrappers, mediation module, or from the underlying models. Error and exception that arise from the mediation module can be handled using the usual error and exception handling strategies. However, error and exception generated by the underlying models vary depending on the implementations of the models. Sometimes errors generated by a model may seem to originate from the wrapper. Unless we do not have detailed documentation of the models, it is very difficult to list the possible kinds of errors and exceptions that could be generated. We may have access to the source code of the models but it could be very difficult to list errors and exceptions, especially run-time errors. Which kind of errors and exceptions can we tolerate? Which are not tolerable? There is no general rule to manage error and exception generated by models, but we have to decide based on individual cases and this makes the way towards generic integration framework more difficult.

With the advances in web technology, presenting models as web services is becoming more prevailing (Castronova et al., 2013; Geller and Melton, 2008; Goodall et al., 2011). The efforts by GEO Model Web Initiative (Nativi et al., 2013) and Apps for the environment[25] by the US Environmental Protection Agency are also exemplary initiatives towards developing models as services. Nevertheless, discovery and integration of service-based models will be challenging unless the naming of service properties is unambiguously defined (Nativi et al., 2013). The CSDMS Standard Names (Peckham, 2014) is an exemplary step towards standardizing names for semantic mediation. However, providing generic semantic mediation requires going beyond standardizing names, it requires a full-fledged ontology. Again the challenge is to build such an ontology to address semantic interoperability across multidisciplinary models, which is not a simple task.

We demonstrated that the semantic mediation in unit conversion among models can be managed using an openly available ontology. Following this, we were able to avoid the need for custom developed code to convert, say, between SI and derived units. However, models can represent a quantity in a number of different ways, for example, concatenated units that are not included in the ontology. There are other freely available ontologies, such as the Semantic Web for Earth and Environmental Terminology - SWEET[26] that currently contains around 6000 concepts in 200 separate ontologies. These could broaden the use of ontology beyond unit conversion.

To minimize the custom coding and hard coding in semantic mediation, metadata of model interface such as input-output variable names, units, scales, etc. should be connected to the ontology (Papazoglou, 2008). To realize this we have to use either the terminologies of the ontology in wrapper model interfaces, or we have to incorporate model interface metadata into an existing ontology. The first option requires understanding and consensus among model developers. However, the second option can be done relatively easily without seeking much collaboration among modeling communities. In this case we are constructing light weight ontology, which could be poor on axioms but sufficient for the definition of attributes and their relationships (Kiryakov et al., 2004). It requires only designing the appropriate structure to accommodate model interface ontology in the existing ontology, in our specific case to QUDT ontology. For example, a model interface has input and output variables; and each of those input and output variables has a name and unit attributes. We can use semantic relationships like 'is-a', 'equivalent', etc. to establish links between names of variables, which are found in different models. Similar relationships can be established between the units used by the model and the corresponding available units in QUDT ontology. Once we define the structure to accommodate model interface ontology then we can register attributes of participating models in the ontology. This can make variable mapping and unit conversion between models fully automatic. But this is not going to happen until model documentation standards will be elaborated and implemented. Here, the seven requirements for semantic annotation by Uren et al. (2006) can be useful in documenting model interface metadata.

The use of distributed modeling approach enabled us to construct runtime access and integration tools. We showed that the technical aspect of integration could be standardized and developed as a 'generic' utility. The availability of such tools helps to focus on providing automatic data mediation functionalities so that users may not need to build data mediation services. Basically, data mediation requires to identify WHAT, WHERE, and WHEN of data in the context of each model (Moore and Tindall, 2005). The WHAT is a quantity description of the data represented by the variable; WHERE is about location information; and WHEN describes temporal information about data. Sometimes the WHERE and WHEN can be optional depending on the context of use of the data. For example, if both model 1 and

[25] http://www.epa.gov/mygreenapps/.

[26] https://sweet.jpl.nasa.gov/.

model 2 refer to the same spatial location and temporal instance then the WHERE and WHEN may be considered as optional when performing data conversion. Due to this, unlike in technical integration, it is difficult to develop a 'generic' module that will handle the data mediation of all incoming models, until appropriate model documentation standards are in place and are adopted by the modeling community. Automatic data mediation functionalities can act on the provided data based on the specific context of the participating models. This may require building artificially intelligent agents that fit such purposes. This requires further enhancement of the semantic mediation features and embedding learning capabilities into the data conversion module. We hope that this will be the future direction of model integration frameworks.

## 8. Conclusion

In the year 2007, The Ecological Forecasting Program at NASA has set the vision for 5–10 years, calling it "The Model Web Concept" (Geller and Turner, 2007). One of its aspirations was to have distributed, networked, and interoperating models, datasets and sensors. In this paper, we present the design of a modular, distributed, scalable, and web-enabled model integration framework that enables linking of a wide variety of models. We also illustrate some of our ideas by implementing a prototype application and by performing a pilot case study. This is hardly a complete solution to all model integration challenges. Our aim is to demonstrate how heterogeneous models can be linked, regardless of their nature and their location.

By offering models as web services and providing an interface that works in a regular browser we make models more accessible and expect them to be more widely used both on their own as stand-alone simulation tools and in integration with other models, serving various needs of decision support. The fact that to run a model we no longer need to download and install it, and setup all the input-output interfaces, very much simplifies its use and creates opportunities for a wider participatory use of models for learning and decision making (Voinov et al., 2016). This also delivers much needed functionality that can make models available in mobile devices and can facilitate co-learning and co-management with large numbers of stakeholders becoming involved in the process. However, further integration and meaningful use of models will become possible only when appropriate model documentation standards will become available and implemented. Standards that can convey the contextual information of the participating models can help consistent interpretation of the represented concepts. This will also facilitate the provision of reusable and standardized data conversion functionalities by grouping available data formats and by developing algorithms that serve specific contexts. Until then we are always running the risk of misrepresenting the ideas, assumptions, conventions, and intentions that are part of any model definition, and as result misusing models. The involvement of experts in the process of model integration will still be essential for some final checks and balances when deciding how models can be coupled and run.

## 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.

## References

Achananuparp, P., Hu, X., Shen, X., 2008. The Evaluation of Sentence Similarity Measures, Data Warehousing and Knowledge Discovery. Springer, pp. 305–316.
Alonso, G., Casati, F., Kuno, H., Machiraju, V., 2004. Web Services: Concepts, Architectures and Applications. Springer.
Argent, R.M., 2004. An overview of model integration for environmental applications—components, frameworks and semantics. Environ. Model. Softw. 19, 219–234.
Athanasiadis, I.N., Janssen, S., 2008. Semantic mediation for environmental model components integration. Inf. Technol. Environ. Eng. 1.
Belete, G.F., Voinov, A., 2014. Integration of Models for Low Carbon Economy. 7th International Congress on Environmental Modelling and Software (iEMSs), San Diego, California, USA.
Belete, G., Voinov, A., Holst, N., 2014. An Architecture for Integration of Multidisciplinary Models. 7th International Congress on Environmental Modelling and Software (iEMSs), San Diego, California, USA.
Belete, G.F., Voinov, A., 2016. Exploring temporal and functional synchronization in integrating models: a sensitivity analysis. Comput. Geosci. 90, 162–171.
Belete, G.F., Voinov, A., Bulavskaya, T., Niamir, L., Dhavala, K., Arto, I., Moghayer, S., Filatova, T. Web service based approach to linking heterogeneous climate-energy-economy models for climate change mitigation analysis. Int. J. Energy. (Manuscript in revision).
Belete, G.F., Voinov, A., Laniak, G.F., 2017. An overview of the model integration process: from pre-integration assessment to testing. Environ. Model. Softw. 87, 49–63.
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.
Buccella, A., Cechich, A., Fillottrani, P., 2009. Ontology-driven geographic information integration: a survey of current approaches. Comput. Geosci. 35, 710–723.
Butterfield, M.L., Pearlman, J.S., Vickroy, S.C., 2008. A system-of-systems engineering GEOSS: architectural approach. Syst. J. IEEE 2, 321–332.
Canhasi, E., 2013. Measuring the Sentence Level Similarity.
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.
Clemens, S., Dominik, G., Stephan, M., 1998. Component Software: beyond Object-oriented Programming (Addison-Wesley New York).
Cohen, W., Ravikumar, P., Fienberg, S., 2003. A Comparison of String Metrics for Matching Names and Records. Kdd workshop on data cleaning and object consolidation, pp. 73–78.
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.
De Nicola, A., Missikoff, M., Navigli, R., 2009. A software engineering approach to ontology building. Inf. Syst. 34, 258–275.
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.
Erl, T., Karmarkar, A., Walmsley, P., Haas, H., Yalcinalp, L.U., Liu, K., Orchard, D., Tost, A., Pasley, J., 2009. Web Service Contract Design and Versioning for SOA. Prentice Hall.
Fowler, M., 2003. Who Needs an Architect. IEEE SOFTWARE.
Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1994. Design Patterns: Elements of Reusable Object-oriented Software. Pearson Education.
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. Geoscience and Remote Sensing Symposium, 2007. IGARSS 2007. IEEE International. IEEE, pp. 2469–2472.
Gijsbers, P., Gregersen, J., 2005. OpenMI: a Glue for Model Integration. MODSIM 2005 International Congress on Modelling and Simulation, pp. 648–654.
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.
Janssen, S.J., 2009. Managing the Hydra in Integration: Developing an Integrated Assessment Tool for Agricultural Systems. Wageningen Universiteit (Wageningen University).
Kiryakov, A., Popov, B., Terziev, I., Manov, D., Ognyanoff, D., 2004. Semantic annotation, indexing, and retrieval. Web Semant. Sci. Serv. Agents World Wide Web

2, 49–79.
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.
Laplante, P.A., 2007. What Every Engineer Should Know about Software Engineering. CRC Press.
Li, W., Yang, C., Nebert, D., Raskin, R., Houser, P., Wu, H., Li, Z., 2011. Semantic-based web service discovery and chaining for building an Arctic spatial data infrastructure. Comput. Geosci. 37, 1752–1762.
Madni, A.M., Sievers, M., 2014. Systems integration: key perspectives, experiences, and challenges. Syst. Eng. 17, 37–51.
Moore, R.V., Tindall, C.I., 2005. An overview of the open modelling interface and environment (the OpenMI). Environ. Sci. Policy 8, 279–286.
Nativi, S., Mazzetti, P., Geller, G.N., 2013. Environmental model access and interoperability: the GEO Model Web initiative. Environ. Model. Softw. 39, 214–228.
Niamir, L., Filatova, T., 2015. Linking Agent-based Energy Market with Computable General Equilibrium Model: an Integrated Approach to Climate-economy-energy System. 20th WEHIA Conference, Sophia Antipolis, France. http://dx.doi.org/10.13140/RG.2.1.3242.0961, 21–23 May.
Niwattanakul, S., Singthongchai, J., Naenudorn, E., Wanapu, S., 2013. Using of Jaccard coefficient for keywords similarity. In: Proceedings of the International MultiConference of Engineers and Computer Scientists, p. 6.
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., Goodall, J.L., 2013. Driving plug-and-play models with data from web services: a demonstration of interoperability between CSDMS and CUAHSI-HIS. Comput. Geosci. 53, 154–161.
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.
Regueiro, M.A., Viqueira, J.R., Taboada, J.A., Cotos, J.M., 2015. Virtual integration of sensor observation data. Comput. Geosci. 81, 12–19.
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). AGILE Workshop: Grid Technologies for Geospatial Applications, Hannover, Germany.
Rosen, M., Lublinsky, B., Smith, K.T., Balcer, M.J., 2008. Applied SOA: Service-oriented Architecture and Design Strategies. John Wiley & Sons.
Sarawagi, S., Kirpal, A., 2004. Efficient set joins on similarity predicates. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data. ACM, pp. 743–754.
Stoorvogel, J.J., 1995. Integration of computer-based models and tools to evaluate alternative land-use scenarios as part of an agricultural systems analysis. Agric. Syst. 49, 353–367.
Tanenbaum, A.S., Van Steen, M., 2007. Distributed Systems. Prentice-Hall.
Uren, V., Cimiano, P., Iria, J., Handschuh, S., Vargas-Vera, M., Motta, E., Ciravegna, F., 2006. Semantic annotation for knowledge management: requirements and a survey of the state of the art. Web Semant. Sci. Serv. agents World Wide Web 4, 14–28.
Uschold, M., 2003. Where are the semantics in the semantic web? Ai Mag. 24, 25.
Voinov, A., Kolagani, N., McCall, M.K., Glynn, P.D., Kragt, M.E., Ostermann, F.O., Pierce, S.A., Ramu, P., 2016. Modelling with stakeholders–next generation. Environ. Model. Softw. 77, 196–220.
Voinov, A., Shugart, H.H., 2013. 'Integronsters', integral and integrated modeling. Environ. Model. Softw. 39, 149–158.
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.
Yang, C., Chen, N., Di, L., 2012. RESTFul based heterogeneous Geoprocessing workflow interoperation for Sensor Web Service. Comput. Geosci. 47, 102–110.
Yu, L., 2011. A Developer's Guide to the Semantic Web. Springer Science & Business Media.
Zhao, P., Di, L., Yu, G., 2012. Building asynchronous geospatial processing workflows with web services. Comput. Geosci. 39, 34–41.