Environmental Modelling and Software 144 (2021) 105164

Contents lists available at ScienceDirect

Environmental Modelling and Software

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

Interoperability engine design for model sharing and reuse among OpenMI, BMI and OpenGMS-IS model standards

Fengyuan Zhang [a,e,f], Min Chen [a,e,f,*], Albert J. Kettner [b], Daniel P. Ames [c], Quillon Harpham [d], Songshan Yue [a,e,f], Yongning Wen [a,e,f], Guonian Lü [a,e,f]

[a] Key Laboratory of Virtual Geographic Environment (Ministry of Education of PR China), Nanjing Normal University, Nanjing, 210023, China
[b] Community Surface Dynamics Modelling System (CSDMS), Institute of Arctic and Alpine Research (INSTAAR), University of Colorado, Boulder, CO, USA
[c] Department of Civil and Environmental Engineering, Brigham Young University, Provo, UT, USA
[d] HR Wallingford, Oxfordshire, UK
[e] Jiangsu Center for Collaborative Innovation in Geographical Information Resource Development and Application, Nanjing, 210023, China
[f] State Key Laboratory Cultivation Base of Geographical Environment Evolution (Jiangsu Province), Nanjing, 210023, China

ARTICLE INFO

Keywords:
Model interoperation
Model standards
Interoperability engine
OpenMI
BMI
OpenGMS-IS

ABSTRACT

Owing to the heterogeneity of geo-analysis models, many scholars and researchers have designed and promulgated standards in an attempt to address this. However, models based on different standards still cannot be shared and reused easily among different model frameworks. For example, models based on the OpenMI, BMI and OpenGMS-IS standards have heterogeneous development styles and formats, so they cannot interoperate. This article analyses the challenges faced when sharing and reusing models across different standards and provides a solution for model interoperation among them. By mapping fields, converting functions, and reorganizing components, our "interoperability engine" allows models that use one standard to be operated within a framework that supports a different standard. This article discusses the developed interoperability method and provides case studies (using e.g. SWMM, FDS, and the Permamodel Frost Number component) to successfully demonstrate model interoperation.

1. Introduction

Modeling and simulation are important analytical methods in geographical and environmental research, and various geoanalysis models have been developed and used for different domains, such as hydrological, ecological, geological, public healthy, and coastal models (Serreze, 2011; Granell et al., 2013; Laniak et al., 2013; Chen et al., 2015; Tóth et al., 2016; Belete et al., 2017a; Conde-Cid et al., 2019; Nourani et al., 2019; Sun et al., 2019; Wang et al., 2019; Shi and Lin, 2020; Baig et al., 2020). Models are useful tools for the simulation of dynamic phenomena and processes, analysing global/regional geo-hypotheses, and supporting decision/policy making (Chen et al., 2013, 2020, 2021; Lin et al., 2013a, 2013b; Lin and Chen, 2015; Lü et al., 2019; Belete et al., 2017b; Ma et al., 2021). With the development of geographical and environmental research, the reuse of such models can help users replicate studies or reduce time to further develop a model. Thus, existing models can help users integrate these models for geographical simulations and hence geographical and environmental sciences can accelerate by sharing and reusing models thereby reducing repetitive re-work.

Different researchers and domain communities use similar techniques to share models for reuse, by using open-source codes, by sharing model executables, or by making models sharable through web applications. However, the lack of universal model standards reduces model reuse between different domain communities and, in some cases, even within a domain community. To address this problem, an increasing number of model standards that rely on sets of different best practice techniques have been designed for model sharing. Owing to their effectiveness and accessibility, component-based models and service-oriented standards are popular in model standard design (Goodall et al., 2011; Whelan et al., 2014). Both of these have advantages and therefore, many platforms or groups implement these techniques at various levels. The Open Modeling Interface (OpenMI), Community Surface Dynamics Modeling System (CSDMS), and Open Geographic

* Corresponding author. Key Laboratory of Virtual Geographic Environment (Ministry of Education of PR China), Nanjing Normal University, Nanjing, 210023, China.
E-mail address: chenmin0902@163.com (M. Chen).

https://doi.org/10.1016/j.envsoft.2021.105164
Accepted 5 August 2021
Available online 13 August 2021

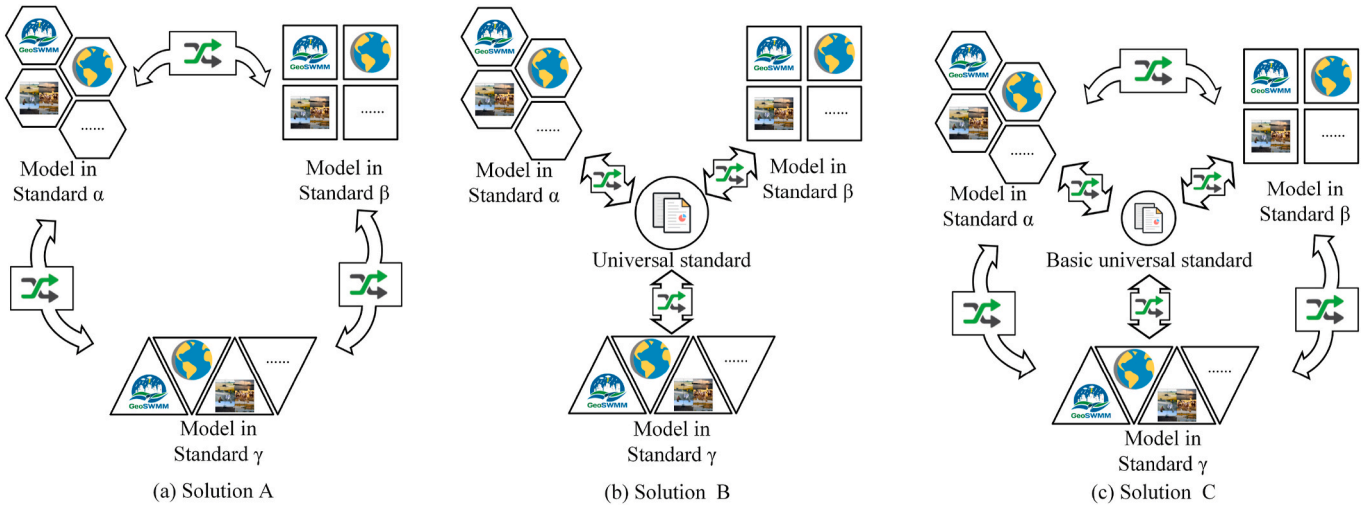

Fig. 1. Visual representation of three conceptual solutions of model interoperability between different model standards.

Modeling and Simulation (OpenGMS) have their own respective standards and software platforms, which have broad applications for model sharing within their user community but currently this does not happen across communities at any scale. OpenMI has a set of standard interfaces for model sharing through the use of components that are linked together to form a composition (Gregersen et al., 2007; Harpham et al., 2019). The basic model interface (BMI) is an open-source library specification for plug-and-play components in the PyMT framework, designed by CSDMS (Peckham et al., 2013; Jiang et al., 2017). OpenGMS is an open platform for model sharing and reuse (Chen et al., 2020) that has the interface standards OpenGMS Interface Set (OpenGMS-IS), which consists of a model encapsulation interface, model description interface, and sim-task operation interface, for model sharing by web services (Zhang et al., 2020b).

Each of these different model standards has been demonstrated to operate successfully by case studies and applications in geographical and environmental modeling. OpenMI has been employed in many case studies in hydrological modeling, including hydrological modeling systems and model integration (Bulatewicz et al., 2010; Castronova et al., 2013; Shrestha et al., 2013). It is an open-source software standard that enables model coupling that can be applied to a wide variety of models (Knapen et al., 2013). It has been implemented in a variety of languages including C#, C++, Java and Matlab for both commercial models and those used for research, OpenMI can also be applied to web services that support web processing to help users share and reuse models (Zhang et al., 2020a). CSDMS offers its community a platform through which open-source models can be easily listed and discovered through detailed model descriptions. To date, a subset of the more than 380 listed open-source models and tools in the model repository of CSDMS have been componentized following BMI standards (Hutton et al., 2020). BMI enables the ability to couple models, making it possible to simulate complex geographical or environmental processes (Drost et al., 2020). OpenGMS presents many models in OpenGMS portal libraries that are wrapped by OpenGMS-IS, such as the Soil and Water Assessment Tool (SWAT), Storm Water Management Model (SWMM), Weather Research and Forecasting Model (WRF) and more. With the help of OpenGMS-IS, these models can be published as web services for sharing and reuse (Zhang et al., 2019, 2020b). This enables geographical simulation applications for model users, such as modeling online systems, model integration, and collaboration (Wang et al., 2018; Xiao et al., 2019; Chen et al., 2019; Yue et al., 2020).

When models and applications are based on one specific standard (e.g. OpenMI, BMI, or OpenGMS-IS), these models and applications can be shared and reused with each other. However, OpenMI, BMI, and OpenGMS-IS based models and applications have heterogeneous development styles and formats for sharing and are, therefore, not interoperable out-of-the-box.

Notwithstanding any subtle differences of approach and application, a number of structural technical aspects prevent this automatic interoperability. First, these standards have different description fields and strategies to expose these fields. Model description information provides users with detailed information about the models themselves. These description fields for the different standards are heterogeneous, so the same information could be expressed in different fields or be communicated in a different manner. For example, for certain standards, the model name field may be marked as 'component name'. Such minor differences could mislead the user, at the least presenting a barrier for model application and making the model incompatible with other platforms and standards. Each standard may also follow different description strategies. Fields in some standards are defined in a corresponding description file, whereas other standards provide such information through an application programming interface (API).

Second, different standards have different invoking methods that cannot interoperate with each other directly. Even if the same functions are used, the parameters can be different. For example, BMI has a function named *update* for the next simulation step, whereas OpenMI invokes the function *getValue* to obtain the model simulation for the next time step. Both BMI and OpenMI are component-based standards, but OpenGMS-IS is a service-oriented standard for model sharing. Meanwhile, different standards have heterogeneous formats for data input/output (I/O). For example, BMI uses an in-memory data stream for data I/O, and this data is communicated between model components following standard names that define variables. In contrast, model services in OpenGMS use file transport for data exchange, and the data format can follow universal data exchange (UDX) for data mapping and conversion (Yue et al., 2015).

Third, file organizations and file dependencies are different for each standard. Different standards follow different development styles (such as component-based and service-oriented styles) and use different programming languages (such as C#, Python, JAVA). A file used in one standard can be challenging to be reused for another standard, which hinders the model invoking across different standards.

These three structural technical challenges make the base implementations of these different standards incompatible, which restricts the reuse and integration of models between frameworks that use these different standards. This research presents a solution for model interoperation among standards developed by CSDMS, OpenMI and OpenGMS. The standards presented in this study are specifically developed for sharing and integration of models. Such a design is intended to help users reuse a model compatible with one standard that

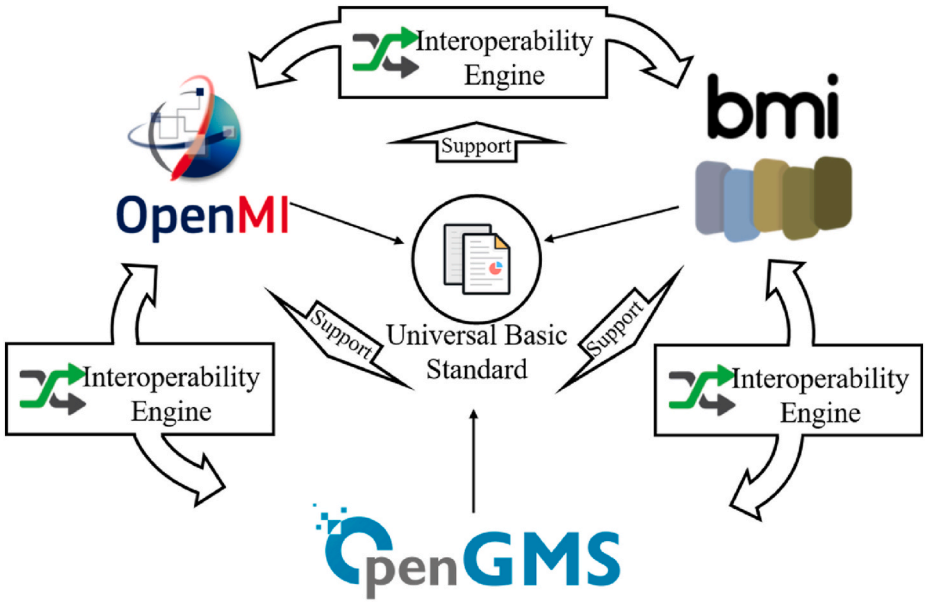

Fig. 2. The engine design among the models based on OpenMI, BMI, and OpenGMS-IS.

is not supported by another, and thereby enrich the model repository of systems or applications based the collection of compatible standards. For example, with the help of the interopability engine, the system based on OpenGMS can use the models based on BMI or OpenMI. Thus, this study would benefit users that prefer to use a particular model that contains a specific standard to be used in a framework that uses a different model standard.

Due to the heterogeneity of these standards, this research describes an interoperability engine to reuse models that are based on different standards. The interoperability engine contains three modules: (1) the field mapping module, (2) the functions conversion module and (3) component reorganization module, which can help users rewrap standardized models, and reuse them in different standards. The rewrapped model would keep the original fields and functions while following new standard. Proof of concept of model interoperations between models with different standards is provided through case studies using the models SWMM, Permamodel Frost Number, and FDS.

The paper is organized as follows: Section 2 introduces the potential solutions for the model interoperation. Section 3 presents the design of the interoperability engine, including field mapping, function conversion, and component reorganization. Section 4 introduces the implementation of the interoperability engine. Section 5 introduces the case studies of these model interoperation. Section 6 discusses the advantages and limitations of this research. Section 7 presents the conclusions and potential directions for future work.

2. Potential model interoperation solutions

This section describes the engine designs to enable model interoperability between models of different standards. Three solutions are considered for engine design. Fig. 1 presents the three conceptual solutions for engine development: Solutions A, B and C. The different shapes describe models that are based on three different standards (Standards α, β, and γ). The interoperability engines that make the models that are based on different standards interoperable are indicated by lines with an arrow at each end.

Solution A aims to build a set of engines to connect the models with each other. As shown in Fig. 1 (a), every model following a standard should have an interoperability engine that makes a connection possible with models following a different standard. In this solution, if a model with a new standard is included, the engines that make all former models interoperable should each be adjusted. This entails significant work for each new model added to the system.

Solution B describes a universal standard that includes all fields and functions, among other standards, and operates as a transfer station that connects all the standards. As shown in Fig. 1 (b), models of each standard are interoperable by using the universal standard. If a model is included from an additional standard, then this model must be made compatible with the universal standard.

Solution C also makes use of a universal standard. However, solution C differs from solution B in that this universal standard only remains part of the core and makes only use of rudimentary fields and functions that are essential in supporting interoperability. The approach is extendable to unique fields and functions and supports the interoperability between them. As shown in Fig. 1(c), the basic universal standard is less elaborate than that of solution B and has other interoperability engines to ensure connections. Additionally, the engine between two models that have different standards should only be developed when necessary.

Solutions A and B both have advantages and limitations. When dealing with a variety of model standards with solution A, the engine that ensure interoperability will be elaborate and require many communication operations to handle data transfer between all of the models. As model standardization is more detailed, so the engine development work and complexity will increase. Without universal standards, it becomes difficult to add new standards with this approach. Solution B is easier to implement. New standards adapt to the universal standard to become interoperable with any standard that is compatible with the universal standard. However, this assumes the presence of a satisfactory universal standard with sufficient coverage across other standards and with a degree of future-proofing to other standards that may come along. This is clearly difficult to achieve for all fields and methods and may require a degree of foresight. Solution C achieves a balance between Solutions A and B. It maintains universal standards only for the basic fields and methods but still has an engine for interoperation between models that have different standards. There is therefore a measure of the strengths and weaknesses of Solutions A and B and it also requires an assessment of the aspects necessary for inclusion in the universal standard – the 'basic fields and methods'.

3. Engine design

Based on Solution C (Fig. 1), this study has designed and developed

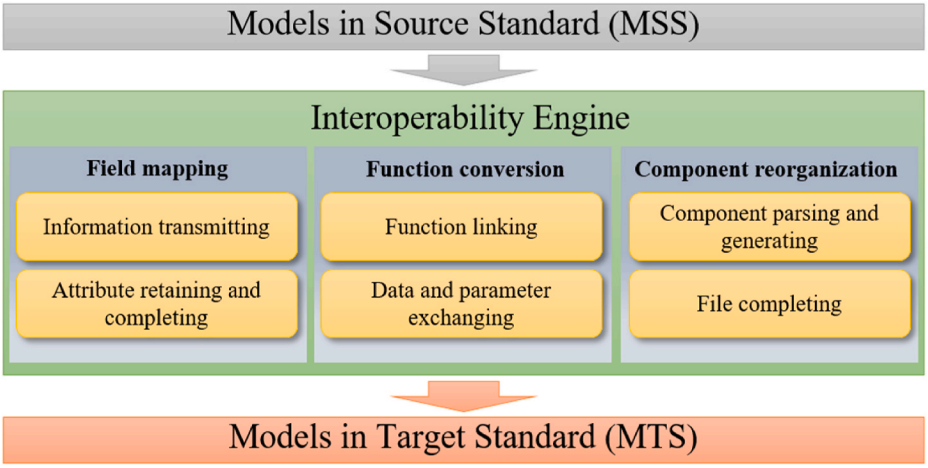

Fig. 3. The framework of the interoperability engine.

an interoperability engine for models that are based on OpenMI, BMI, and OpenGMS-IS (Fig. 2). First, a basic universal standard is developed that can help users interoperate models based on the other standards. The basic universal standard contains tables for description fields and functions, which can support the interoperability engine design between the models that are based on different standards. Then, the operations between every two standards are bidirectional, so that there are two interoperation functions developed between each pair of standards. For example, the engine should two interoperation functions between OpenMI and OpenGMS: convert OpenMI based models to OpenGMS and the reversed one. The model that is initiating a call to another model is named the "model in source standards" (MSS), while the model that has been called is named the "model in target standards" (MTS).

As shown in Fig. 3, the model interoperability engine is designed to convert MSS to MTS. The engine design consists of three modules: field mapping, function conversion, and component reorganization. The field mapping module maps the description fields between the models in different standards. The description fields of different standards are heterogeneous, so fields in one standard may not exist in other standards. Therefore, field mapping has two functions: information transmitting, and attribute retaining and completing. Information transmitting aims to transmit the matching information from MSS to MTS. Attribute retaining and completing is designed to preserve the fields and retain missing fields in the MTS from the MSS. The function conversion module maps functions between the MSS and MTS to enable interfaces from the MTS to invoke and interact with the MSS. This module consists of function linking and data and parameter exchanging. Function linking is used to link related functions together between every two standards to allow the MTS to invoke or interoperate with the MSS. Data and parameter exchanging is designed to convert the input/output data/parameter formats or content between every two standards. Due to the heterogeneity of the model components, component reorganization reorders the file to reformat the model from the MSS to the MTS via component parsing and generating and file completing. Component parsing and generating is applied to parse the MSS and generate the components of the MTS. After parsing, file completion can supply necessary files in the MTS that allow the model to be reused with the target standard.

3.1. Field mapping

Different standards have different fields to describe a model. Some are the same or similar, such as model name, brief description, inputs, and outputs, but some are different. A number of fields with the same meaning can be matched together between the MSS and MTS (such as *ModelName/Title* and *ModelDes/Info*). Some fields cannot be matched, as some cannot be found in either the MTS or the MSS. Moreover, the methods used to obtain these fields are also different. For some models, the fields can be obtained from a file, while for other models the information of fields can be obtained from the API. Therefore, the heterogeneity of a model description will lead to misunderstanding and as such, create a barrier for the reuse of models in different standards.

Table 1
Common fields table for model interoperation.

Common Fields	Description
Name	The name of the model or model component
Description	Brief description about the model
Inputs	The input items in the model
Outputs	The output items in the model
InputsCount	The count of input items
OutputsCount	The count of output items
InputType	The type of input items
OutputType	The type of output items

To address these problems, this research designs a common field table for basic field mapping. The lookup table is presented in Table 1. The lookup table contains the basic fields for the three standards, which makes it possible for models to map to different description fields with other standards' fields or APIs to obtain the corresponding information of the fields. As shown in Fig. 3, the field *ModelName*, *ModelDes*, *Request*, and *Response* in the MSS are mapped to *Name*, *Description*, *Inputs*, and *Outputs* in the common fields table. Then, these fields can be mapped to the MTS as *Title*, *Info*, *Inputs*, and *Outputs*. For the fields that differ between the MSS and MTS, the method is designed to create supplement documents and preserved documents: supplement documents are put in place to complete missing fields from the MSS to the MTS; preserving documents can be used to retain useful fields missing in the MTS that may be used in future. As shown in Fig. 4, the field *keywords* and *categories* in the MSS are missing in the target standard, and these fields can be retained in the preserved documents in the MTS. Meanwhile, the field *platform* and *dimensions* cannot be found in the source standard, so these can be completed by the supplement document. The fields in preserving documents may not be used in the MTS, but they can be useful when the MTS or the applications for MTS are upgraded.

3.2. Function conversion

The functions in these standards show strong heterogeneity and data in the I/O of related functions also need conversion for interoperability purposes. First, the functions for invoking differ between standards. For example, in BMI, the function *update* can be used to perform a time step; in OpenMI, the equivalent function is *GetValues*. Even if the functions

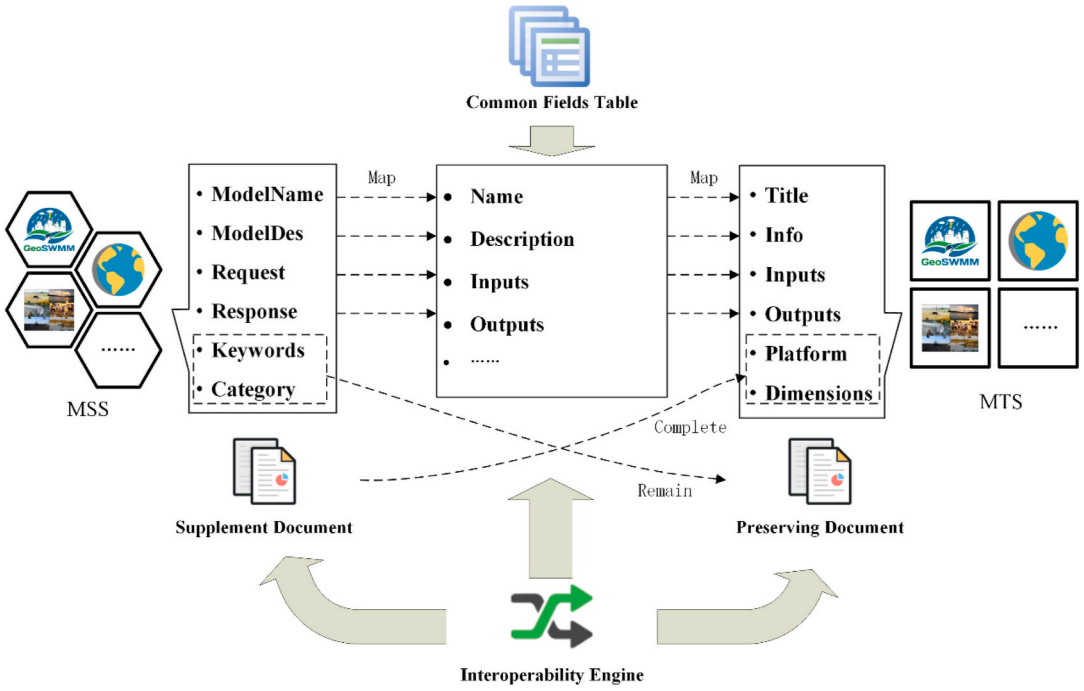

Fig. 4. Fields mapping design in the interoperability engine.

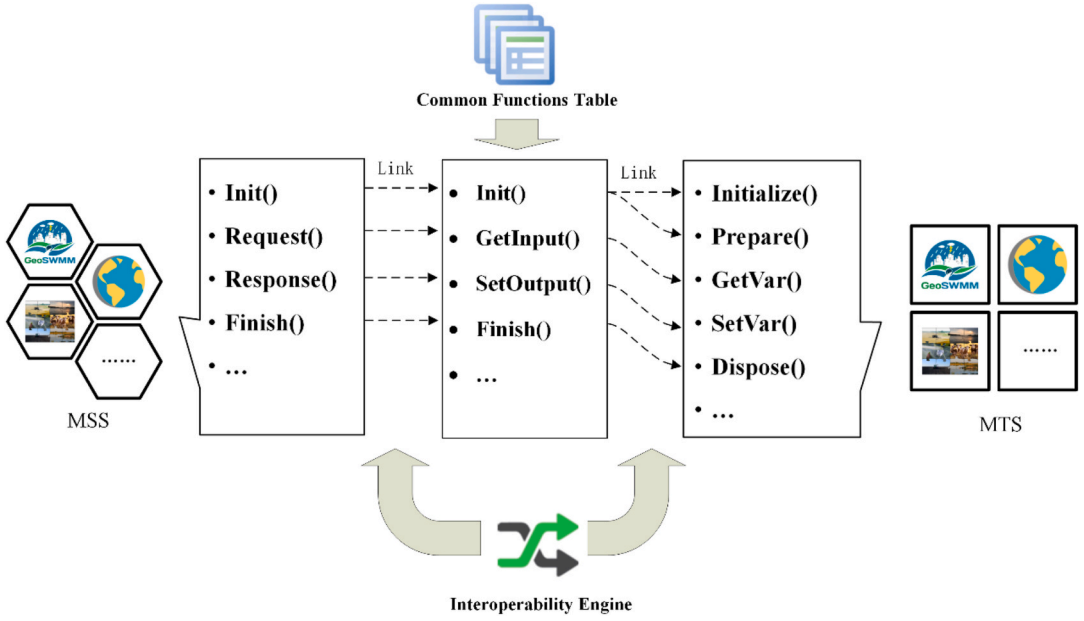

Fig. 5. Function linking design in the interoperability engine.

can be converted, the logic in the corresponding functions may differ. The *update* function in BMI (python) does not set any parameters, but *GetValues* in OpenMI can set *time* and *linkID* and obtain the corresponding result. Furthermore, the data that need to be transported in functions can also differ, as different standards have diverse requirements for data formats and content. For example, the OpenGMS model service can be inputted with raw data files or streams or UDX model-based data; models based on BMI should be inputted with data formatted as an array, and each data point should be associated with one standard name (Yue et al., 2015; Jiang et al., 2017).

To address these issues a function conversion module has been designed that consists of linking functions and data exchange. First,

Table 2
Common functions table for model interoperation.

Common Functions	Description
Init()	Initialize the model
NextStep()	Execute the model or push the model into the next step
Finish()	Finish the model and release the related resource
GetInput()	Get the input data
SetInput()	Set the input data
GetOutput()	Get the output data
SetOutput()	Set the output data

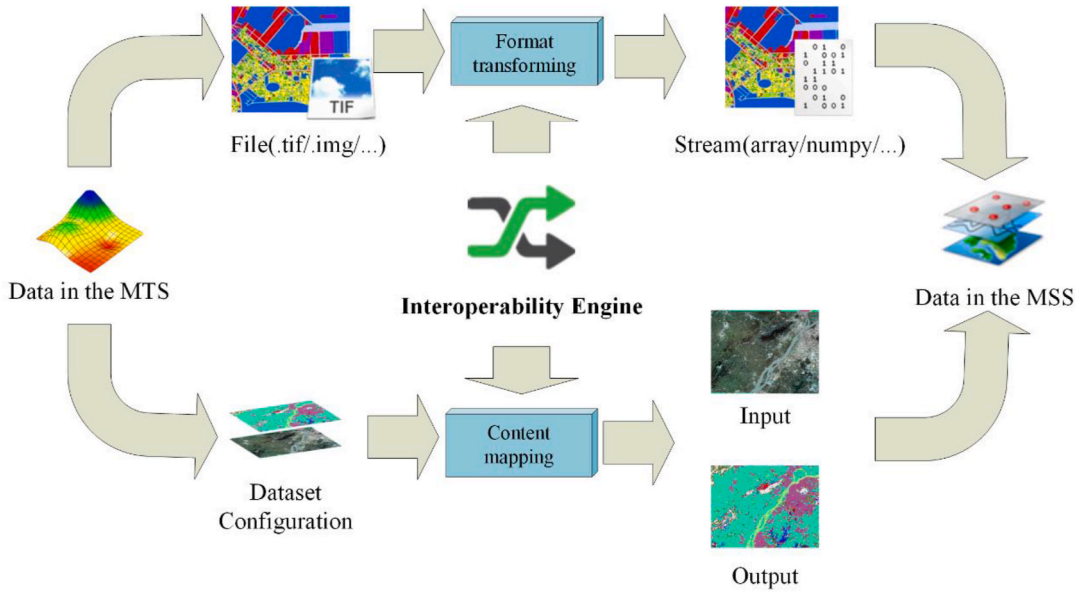

Fig. 6. Data exchanging in the interoperability engine.

function linking, as shown in Fig. 5, has a common functions table (Table 2) for reference to link functions between standards. Different from the common fields table, the function linking in the common functions table may experience cases in which one function can be linked with multiple other functions during different usage situations. The reason for this problem is that the standards are used by different

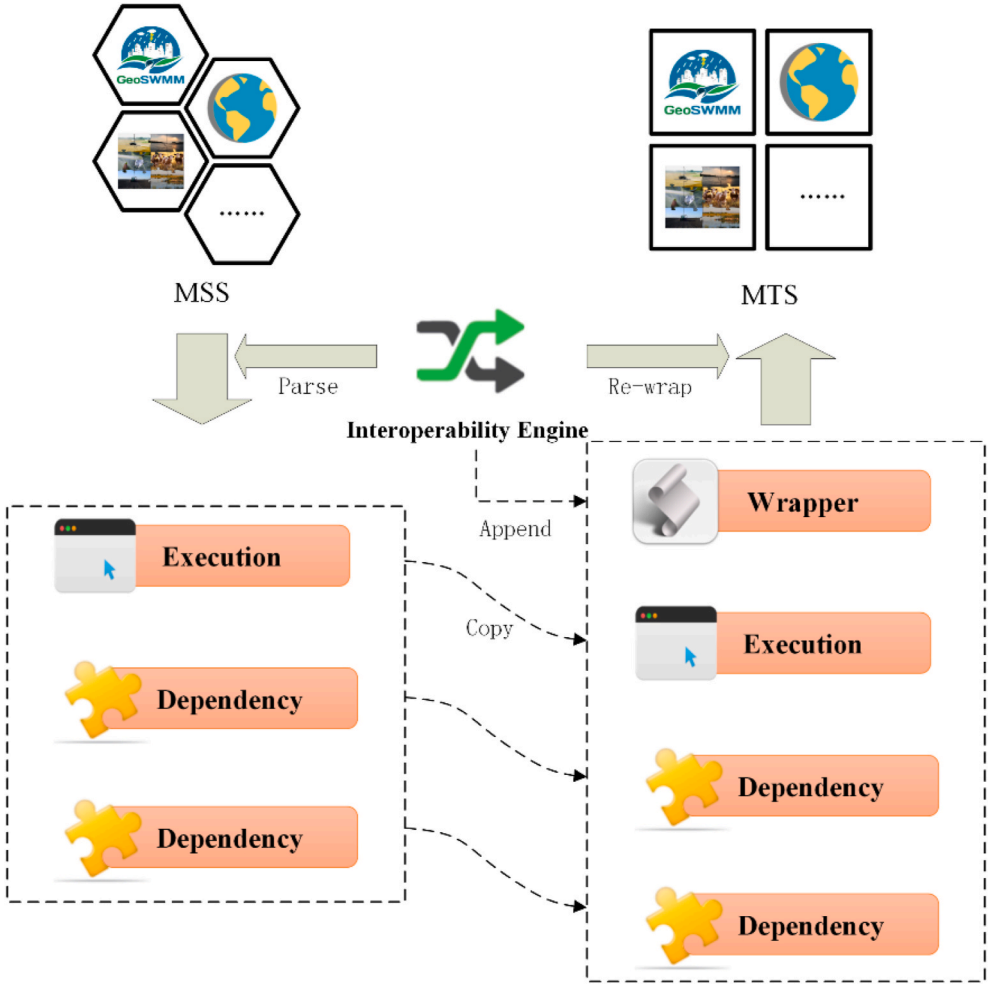

Fig. 7. File reorganization in the interoperability engine.

Table 3
Sub-engines and functions in the interoperability engine.

Sub-engine name	Function	Description
OpenMI-OpenGMS Engine	convertOpenGMS2OpenMI	Convert the OpenGMS service to the OpenMI component
	convertOpenMI2OpenGMS	Convert the OpenMI component to the OpenGMS service
BMI-OpenGMS Engine	convertOpenGMS2BMI	Convert the OpenGMS service to the BMI component
	convertBMI2OpenGMS	Convert the BMI component to the OpenGMS service

roles: model wrappers and model users. Model wrappers would provide the model resource in specific standard, and model users play the role of "model customers" for the simulations. In the standards for components (such as BMI and OpenMI), the functions can be the same for wrappers and users; however, in the standards for services (such as OpenGMS), the functions are different. Therefore, in the table, a function may be linked by two functions in different situations or a combination of two or more functions. For example, as shown in Fig. 4, the function *Init()* in the common functions table can be linked to *Init()* in the MSS and *Initialize()* and *Prepare()* in the MTS.

Second, data exchange, as shown in Fig. 6, can help users convert the data/parameter format or map the data/parameter content from MTS to MSS. The heterogeneity of data consists of format and content differences. Therefore, the interoperability engine has the functions of format transformation and content mapping. Format transformation can help users convert the data format between standards. For example, the input data in OpenGMS models can be a file (such as TIFF, IMG, CSV, etc.), and the input for models based on BMI should be provided as arrays: if OpenGMS wants to interoperate with a BMI model, it should convert the file into an array. Content mapping aims to reorganize the data content to fit the data to the target standard. For example, the I/O data in OpenMI may be configured in the arguments together but do not distinguish input and output, which is necessary for other standards. Therefore, the interoperability engine should supply a function to map the data in the MTS to input and output data to the MSS.

3.3. Component reorganization

A model following a specific standard often has a fixed set of dependency files such as dynamic link library (DLL), shared library (SO), or python module (PY) files to support invoking. However, the file organization between MSS and MTS is more heterogeneous. For example, the model organization between the models wrapped by BMI and OpenGMS is different. The former needs the file that supports BMI component, and the later needs OpenGMS related files for service generating. Therefore, the files in MSS are reorganized such that they follow the target standard when the MTS needs to interoperate the MSS.

The component reorganization module in the engine has functions for component parsing and generating, and file completing. First, the engine parses the MSS to perform the operation for file completion. Owing to the differences of a model when applied to different standards, the components of models have different development styles to represent a model. Some are plug-and-play components, others are web services. There, the engine for the different standards have diverse strategies for parsing. After file completion, with following the target standard, the model is rewrapped as MTS by the engine. Then, as shown in Fig. 7, to ensure that the MSS can be invoked, all the files from the MSS should be retained. Meanwhile, the MSS should append some other files to support the model reusability for the target standard. These files include a wrapper file that includes the dependency files for target standard. The wrapper file that is appended in the MTS can help the model interoperate with the MSS solve functions.

4. Implement

To validate the design in this research, an interoperability engine was constructed based on the aforementioned design principles. The engine between the MSS and the MTS is bidirectional, which means that there should be at least two functions between every two standards. Even if models follow the same standard, they could have different development styles with different methods for interoperation. Two sub-engines are developed among OpenMI, BMI, and OpenGMS-IS. Owing to the different development styles, these engines have different methods to fit different kinds of models to the source standard.

This study discusses several functions for the interoperation among these standards. The collection of engines is shown in Table 3. There are four functions for the two pairs of standards, two functions each. The engine is developed in python, and the engine only contains the interoperation of OpenMI/OpenGMS and BMI/OpenGMS. Owing to the programming hetereogenity of different standards, some parts of these engines would use other languages to meet the requirement of standards, such as the component of OpenMI, which uses C#. The rules for field mapping and function conversation can be referred in Supplementary Information, Appendix A and B.

4.1. OpenGMS-OpenMI

The function OpenGMS-OpenMI is designed to convert the OpenGMS model service to the OpenMI component. As shown in Fig. 8, at first, this engine parses the APIs of the OpenGMS service, and maps the attributes

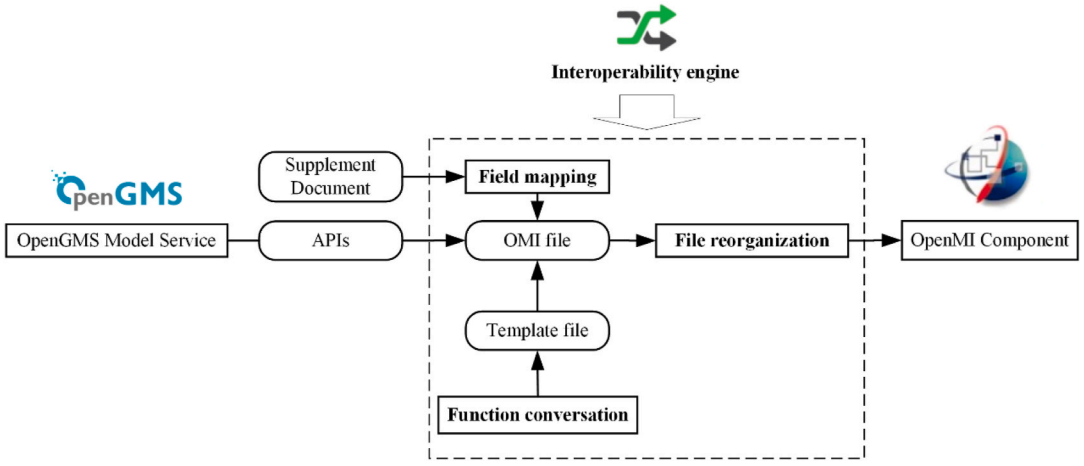

Fig. 8. A function description of an OpenGMS model service to an OpenMI component.

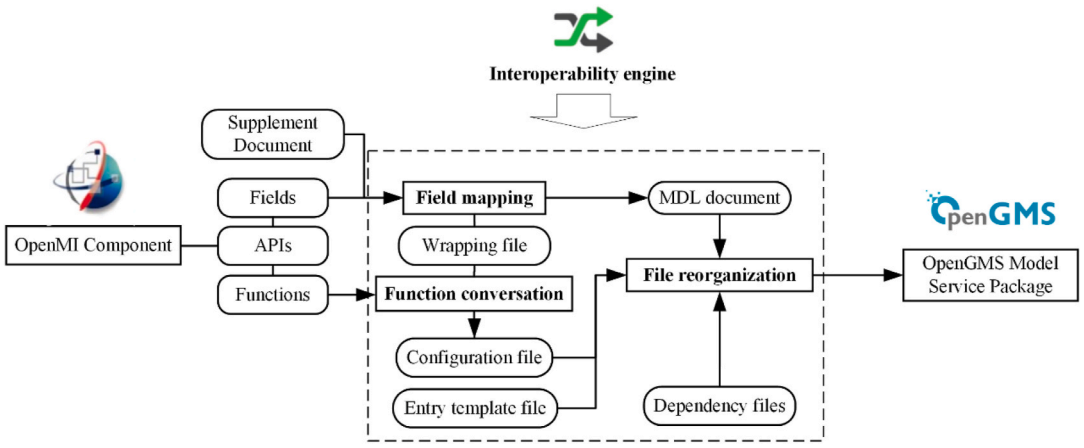

Fig. 9. A function description of converting an OpenMI component to an OpenGMS model service.

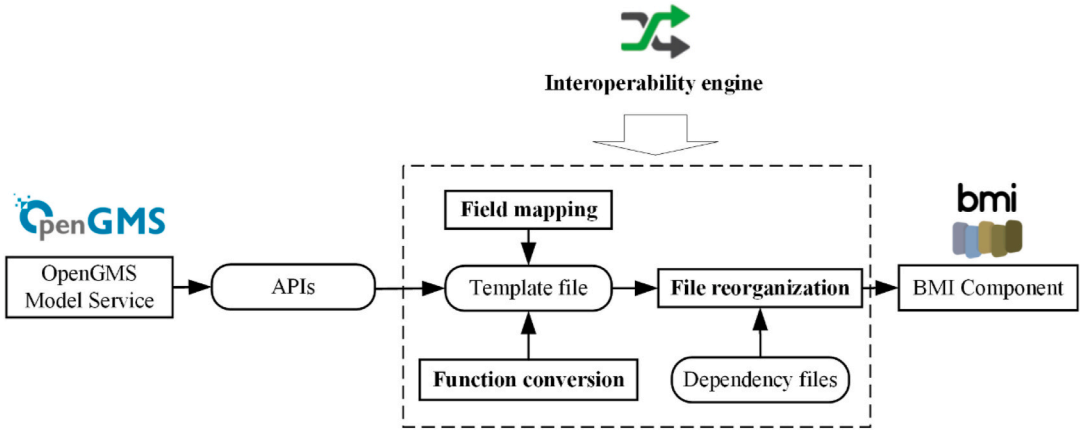

Fig. 10. A function description of converting an OpenGMS model service to a BMI component.

in model services to the OMI file by a model service address. All the input and output data in the OpenGMS model service are mapped as arguments in the OMF file. The missing fields are supplied by the supplement document, such as component name and data. Then, the engine generates the template file for function conversion between OpenGMS service and OpenMI component. Finally, the engine copies related resource files and dependency libraries to the component.

As models in OpenGMS are web services, the engine would not read the model attributes directly. It would link the related API to the interface in OpenMI, so the attributes displayed in the converted OpenMI component are dynamic. In this case, the OpenMI components use C#, so the engine for OpenMI-OpenGMS-IS is developed in C# and python. The python is used for API and C# is used for the template file.

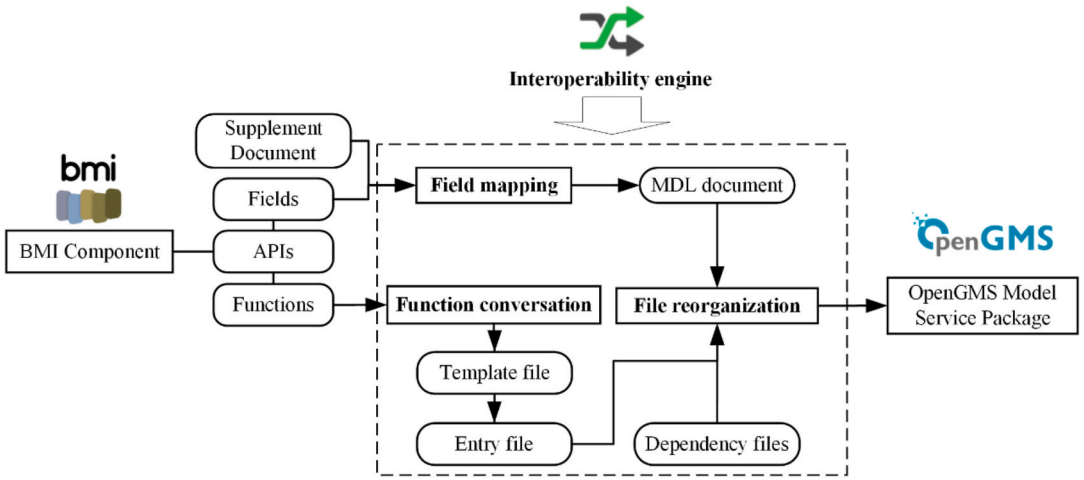

Fig. 11. A function description of converting a BMI component to an OpenGMS model service.

Table 4
Geo-analysis models in case studies to validate the model interoperation.

Model	Description	Source	Reference
Permamodel Frost Number	A model component in Permamodel to calculate frost number	BMI	Overeem et al. (2018); Nelson and Outcalt (1987)
SWMM	A dynamic rainfall-runoff simulation model	OpenMI	Rossman (2010)
Fire Dynamic Simulators (FDS)	Indoor fire disaster simulation model	OpenGMS	McGrattan et al. (2013)

4.2. OpenMI-OpenGMS

The function OpenMI-OpenGMS is designed to wrap the OpenMI component to an OpenGMS service. As shown in Fig. 9, first, with the help of OpenGMS SDK, the engine parses the APIs of the OpenMI component to fields and functions. The fields parsed from the APIs and the supplied fields in supplement document are mapped in a MDL document. The functions in OpenMI component are then converted to the OpenGMS wrapping interface, to a wrapping file, which is generated by a configuration file and an entry template file. The configuration file is used to indicate the input data and output data. And the entry template file can be converting the corresponding wrapping file of the OpenGMS model service package. Then, with the help of the MDL document, the wrapping file and the related dependency files, the engine reorganizes the component as OpenGMS model service package, which can be deployed as an OpenGMS model service.

4.3. OpenGMS-BMI

The function OpenGMS-BMI is designed to convert OpenGMS model services to BMI components. As shown in Fig. 10, the engine parses the APIs of OpenGMS model services. Then, the engine records the basic fields of the component (including service's IP, port and id) and converts related functions. Similar to the function OpenGMS-OpenMI, the fields of the converted BMI component are dynamic. Then the engine generates a BMI component by a template file. The template file has established rules for function conversion. Finally, the engine appends the BMI necessary files with dependency files and generates a BMI component. In the BMI component, the data I/O in BMI is mapped as an array stream and each I/O has a standard name. Therefore, in the conversion of data, each I/O would be formatted as data files and have a specific parameter name.

4.4. BMI-OpenGMS

The function BMI-OpenGMS is designed to convert BMI components to the OpenGMS model service. As shown in Fig. 11, first, the engine parses the BMI component, and map the fields to the MDL document. Any missing fields are added by the supplement document, like e.g. running environment information. Then, the engine converts the functions to the OpenGMS wrapping interface by using the template file, and generate a wrapping file. The data I/O in OpenGMS are transferred by files and the data I/O in BMI are transferred by a stream. Therefore, the engine would generate a temporary file for data I/O. Finally, the engine reorganizes the file as an OpenGMS model service package by the MDL

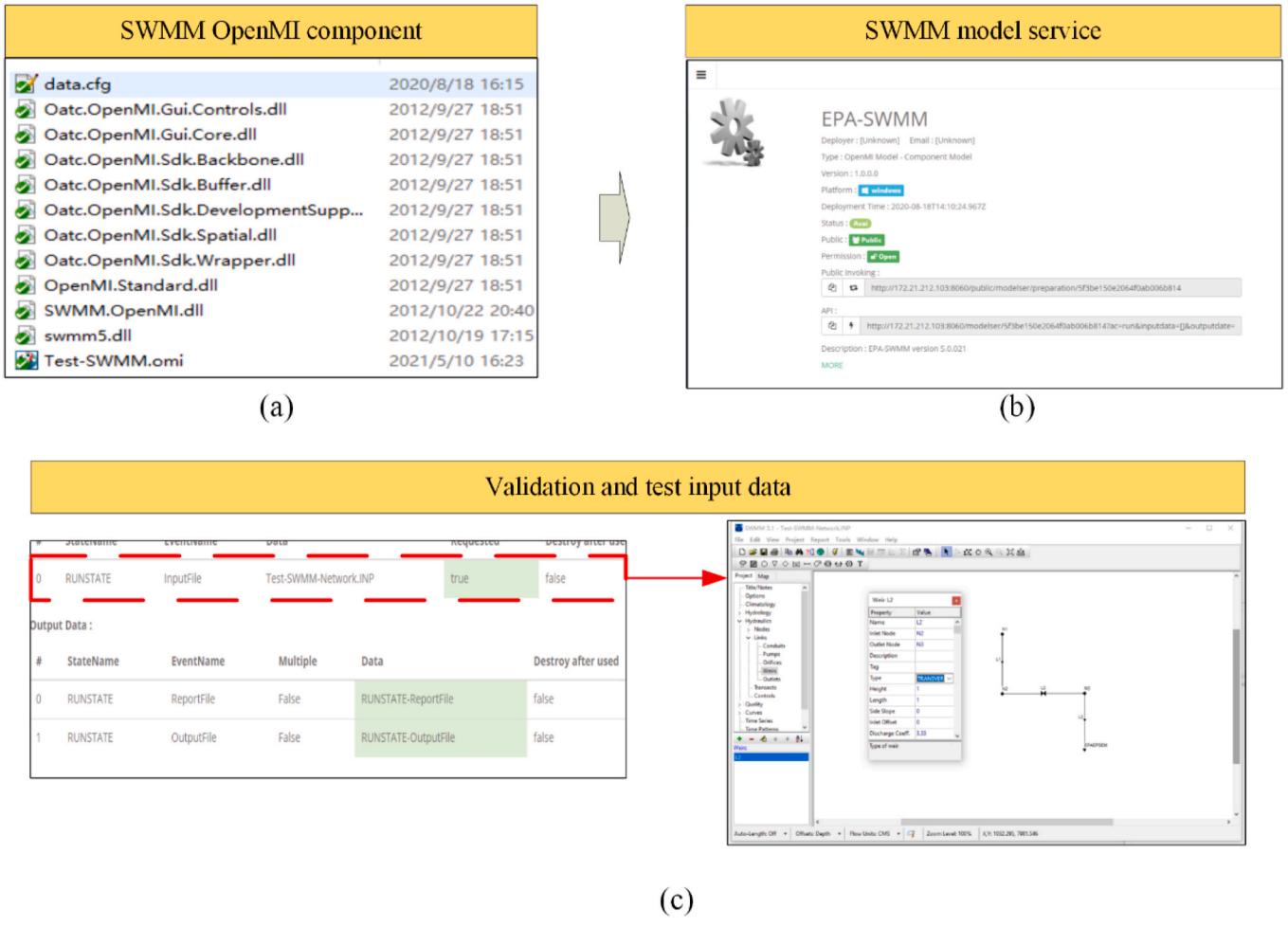

Fig. 12. Case study of OpenGMS-IS interoperating SWMM based on OpenMI.

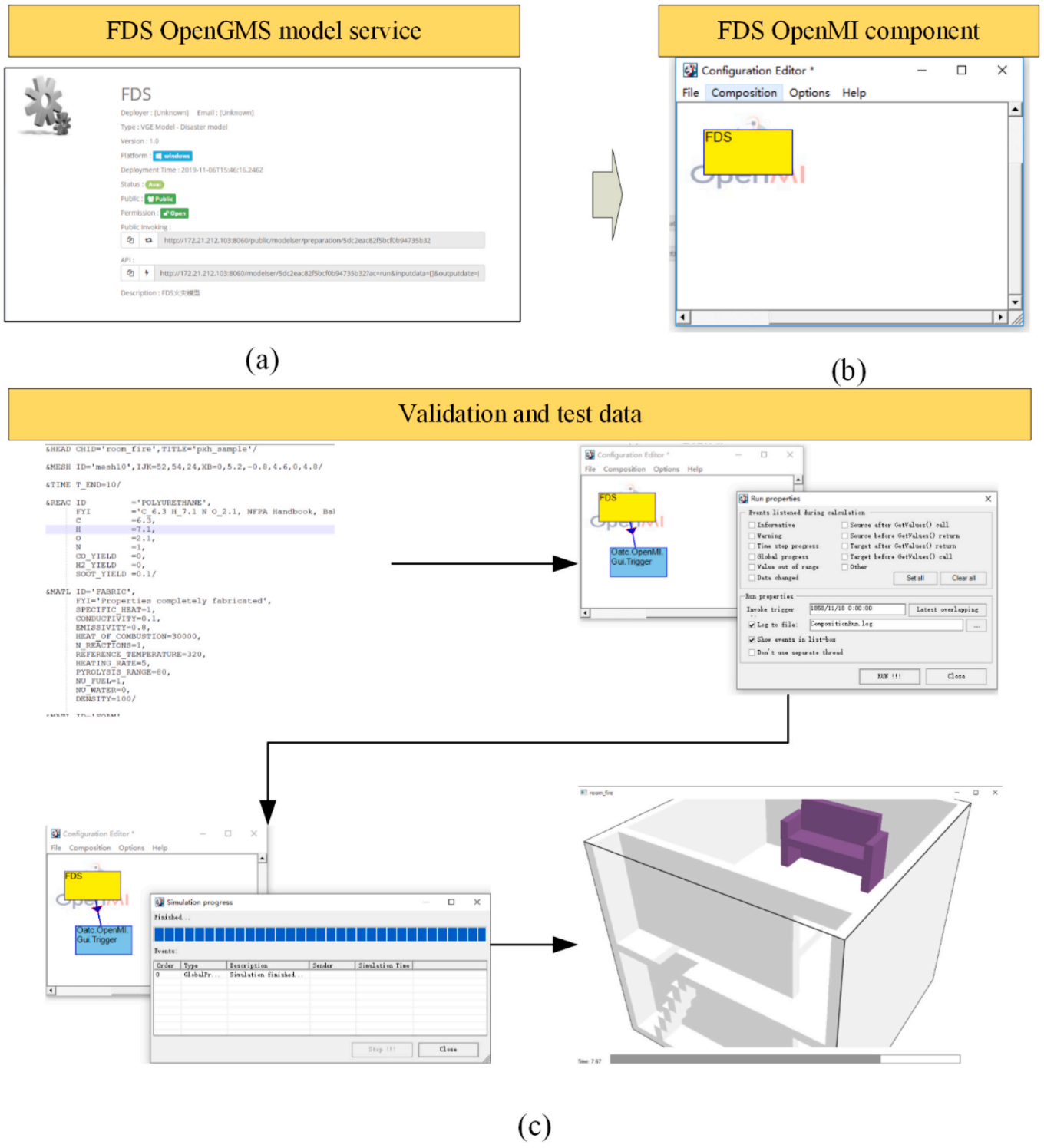

Fig. 13. Case study of OpenMI interoperating FDS based on OpenGMS model service.

document file and the wrapping file.

5. Case studies

For this study we used different geo-analysis models to validate the effectiveness of developed interoperability engines. The models are shown in Table 4. The Permamodel Frost Number is a model component developed by Nelson and Outcalt to calculate "Frost Number" in permafrost, which is a dimensionless ratio based on freezing and thawing degree days in the year. SWMM model is a model developed by U.S. Environmental Protection Agency (EPA) to simulate rainfall-runoff for an urban area. Fire Dynamic Simulators (FDS) is a model developed by U.S. National Institute of Standards and Technology (NIST) to simulate indoor fire disasters.

As shown in Figs. 12 and 13, we use SWMM and FDS model services to validate the interoperability engine between OpenGMS and OpenMI for this study. As shown in Fig. 12(a) and (b), the interoperability engine interoperates SWMM OpenMI component as model service. Then, as

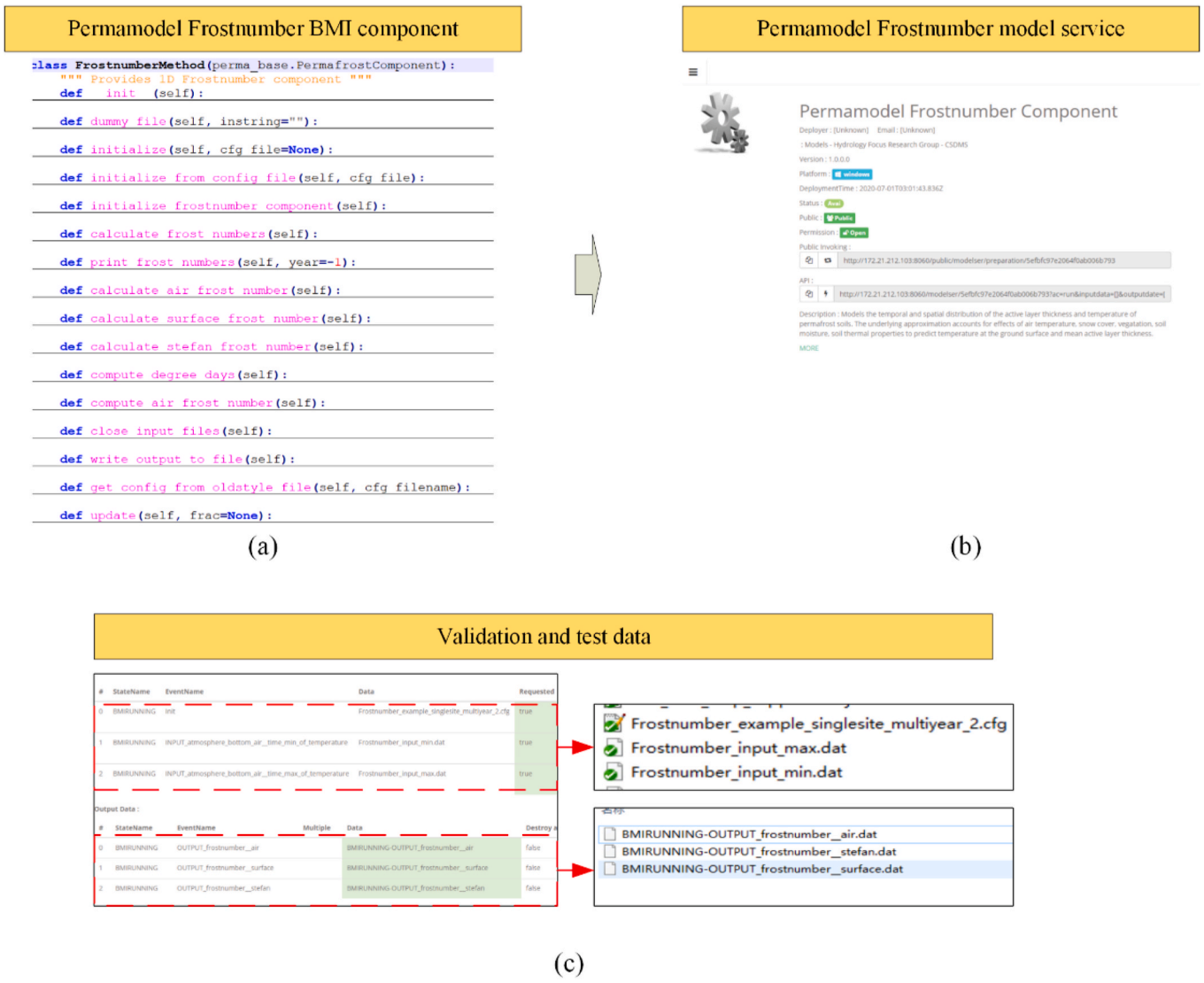

Fig. 14. Case study of OpenGMS-IS interoperating Permamodel Forestnumber based on BMI.

shown in Fig. 12(c), we use the network test data (*.inp) to invoke the SWMM OpenGMS service and obtain the result. As shown in Fig. 13(a) and (b), the interoperability engine interoperates FDS model service as an OpenMI component. Then, as shown Fig. 13(c), we use the test data (*.fds) to invoke the FDS model service by OpenMI.

As shown in Figs. 14 and 15, we use the Permamodel and FDS model services to validate the interoperability engine between OpenGMS and OpenMI for this study. As shown in Fig. 14(a) and (b), the interoperability engine interoperates the Permamodel Frost Number component as a model service. Then, as shown in Fig. 15(c), we use the test data to invoke Permamodel Forestnumber OpenGMS service and obtain the result. As shown in Fig. 13(a) and (b), the interoperability engine interoperate FDS model service as a BMI component. Then, as shown Fig. 13(c), we use the test data (*.fds) to invoke the FDS model service by BMI.

6. Discussion

6.1. Standards selection

OpenMI, BMI, and OpenGMS-IS are supported by different communities or groups, but all have a focus toward developing or implementing models. Although when developing models, these communities use different technological styles and usage logics, they have broad applications in different domains, such as hydrology, land, atmosphere, etc. OpenMI has a component-based style and pays more attention to model integration in simulations. Models based on BMI are supported by the CSDMS community. They also use component-based models, where components written in different languages can be wrapped. OpenGMS-IS is the interface standard applied in OpenGMS and it aims to share and reuse models in an open web environment. So, models with OpenGMS-IS are services on the web. With wide application and different developing styles of these standards in the simulation for different domains, they can be good paradigms for model interoperation among standards.

In addition to the heterogeneities of these standards, they also share common ground. For example, each standard has a model description for introduction and a statement for the model's input/output. So, the common tables (including common fields table and common functions table) can be generated for interoperation among the different standards. However, the tables are only applied among OpenMI, BMI, and OpenGMS-IS. Therefore, if any new standards are introduced, the tables can be reconsidered and reorganized, such as via field or function appending.

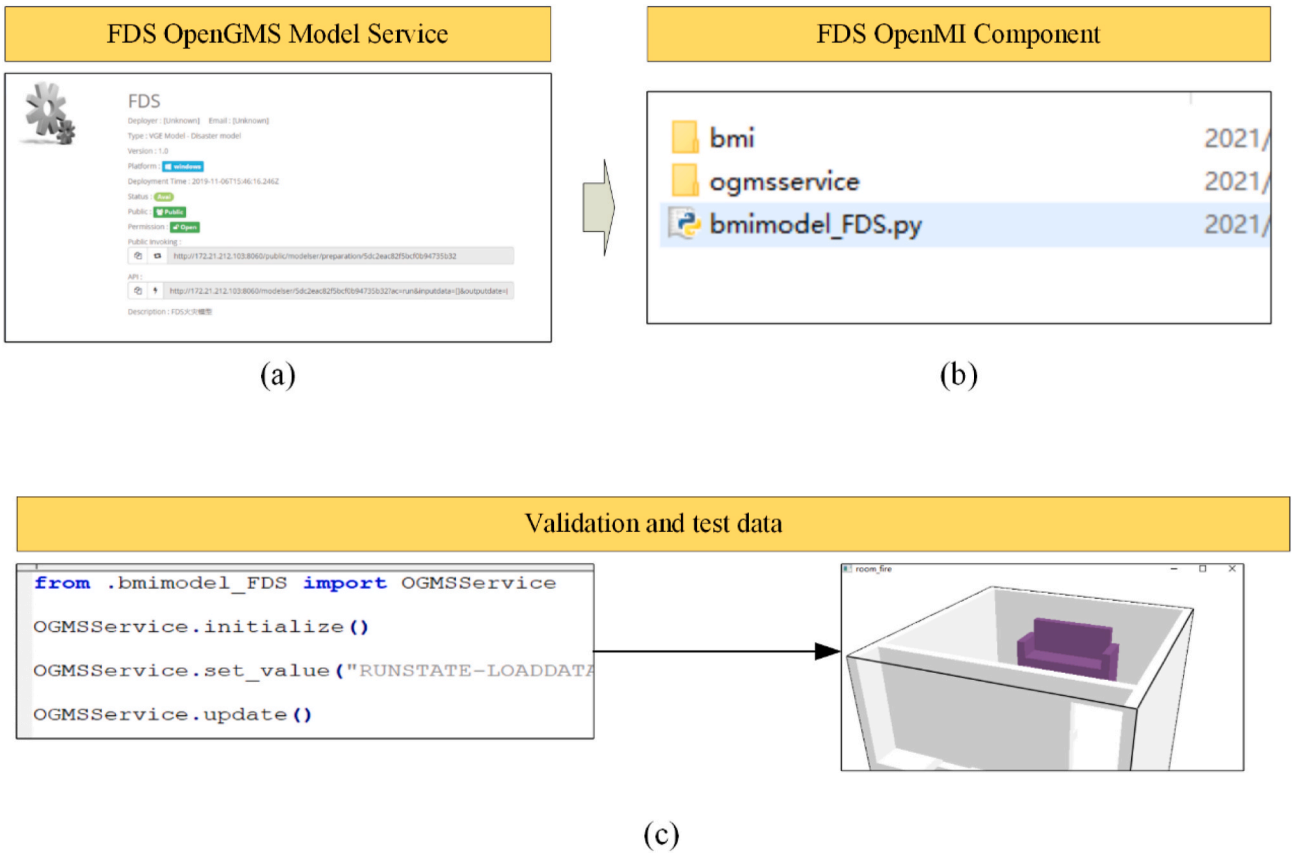

Fig. 15. Case study of BMI interoperating Permamodel Forestnumber based on OpenGMS model service.

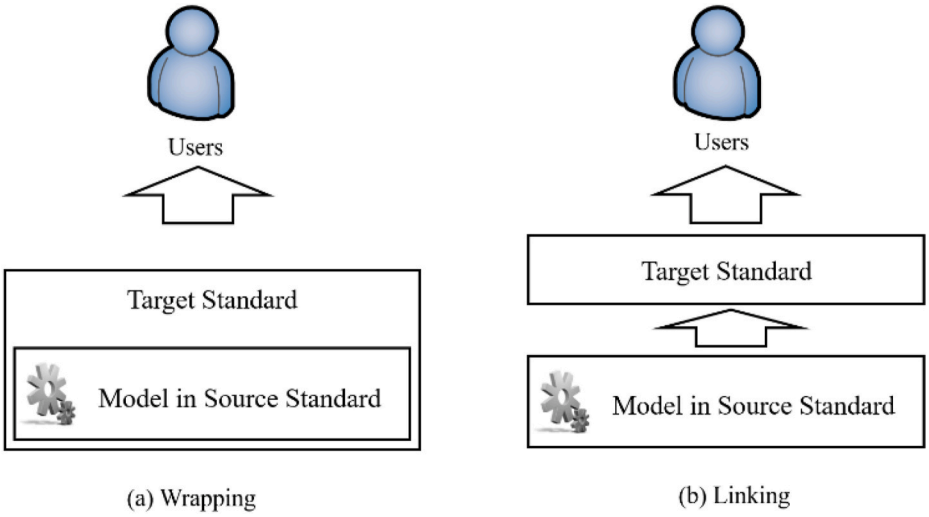

Fig. 16. Two approaches to design interoperability engines.

6.2. Wrapping or linking

Owing to different developing styles, the interoperability engine follows different approaches for the model interoperation among different standards. As shown in Fig. 16, there are two approaches to design interoperability engines: wrapping and linking. Wrapping, as shown in Fig. 13(a), means that the engine wraps the model with completed dependency files as model components in a new standard, such as OpenMI or BMI component to OpenGMS model service. In contrast, as shown in Fig. 13(b), linking means that the engine utilizes MTS to link the MSS on the web, and in the model in MTS, it would not have - or just partly have - model files in the source standard, such as OpenGMS model service to an OpenMI or BMI component. Compared with linking, wrapping does not change the original model and its components, and a change in the raw MSS would not influence the model in the MTS. Thus, wrapping is more suitable for component-based models (i.e., BMI and OpenMI). However, linking only links the functions between the MSS and MTS, and any changes in the MSS would change or even destroy the target model. Thus, a linked connection might be more flexible and lighter to make models interoperable and

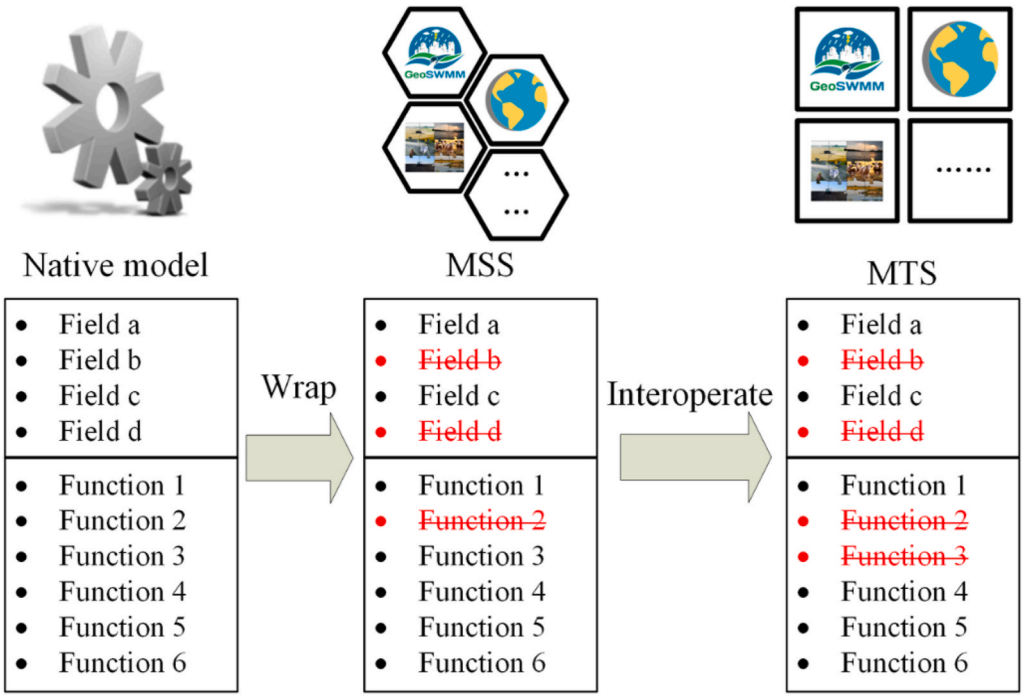

Fig. 17. Liebig's law in model interoperation.

might be more suitable for service-oriented models (i.e., OpenGMS).

6.3. Liebig's law

In the model interoperation, comparison with the native model and the MSS, the MTS always has fewer fields and functions. For example, the OpenGMS model service doesn't include the related grid information of input data of the BMI model component. After the model interoperation, we found that the MTS followed the Liebig's law in fields and functions transferred from native models. Liebig's law, also called the Liebig law of the minimum, states that growth is not determined by the total available resources but by the scarcest resources, i.e., the limiting factor (Danger et al., 2008). This law is applicable to interoperability among standards. Owing to the limitations of standards and the native models, the numbers of description fields and available functions decreases from the native model to the MTS. As shown in Fig. 17, once the native model is wrapped in MSS, owing to the limitations of the wrapping standard, *Field b*, *Field d*, and *Function 2* are missing. When the model is made interoperable with MTS, in addition to *Field b*, *Field d*, and *Function 2*, *Function 3* becomes also unavailable. Thus, all the functions in the final standard are determined by the intersection among native models and standards.

6.4. Models' independency of standards

After model interoperation, some models may still not be interoperable given the new standards. That is, models that are using the same standard could still be coded using a different architecture. Owing to the different habits or development styles of researchers and scholars, despite using the same standard, these models could still need additional files or components for invoking. For example, models can be part of a special system or framework and, as such, require dependency files to run. In the dependency of models, some are based on standards or system libraries, such as system dynamic link libraries, or BMI interface files. However, some dependencies are customized for special application, such as personal library files (*.dll, *.py). These models are tightly coupled in a framework and are no longer independent components that can be reused in other systems or frameworks. Therefore, although they follow the same standard, dependent models cannot be shared and reused in other systems that follow specific standards.

7. Conclusions and future work

This research analyzed the interoperability of models that are developed using different standards. By comparing three potential solutions to make models more interoperable, this research offers a suitable solution for model interoperation among different standards (OpenMI, BMI, and OpenGMS-IS). The solution includes a design that consists of three modules for field mapping, function conversion and component reorganization to interoperate models across these standards. By means of this design, this research developed an interoperability engines among OpenMI, BMI, and OpenGMS-IS that can reuse models across these standards. This research used models (including SWMM, Permamodel, FDS, and FVCOM) to demonstrate that such a design can be helpful in model interoperation among different standards. Finally, some key points for model interoperation are discussed, including model selection and interoperation approaches. The presented work also identified certain limitations.

The engines presented in this paper are tight couplings, which makes them more difficult for reuse. With the help of a basic universal standard, the engine development is simpler, but there is still much work needed to develop each engine. With more model standards being developed, the need for engines is increasing as well. Therefore, reusable and plug-and-play components for engine development are necessary to grant interoperability between different model standards.

Different standards have different rules for their data exchange within models. OpenMI has interface IExchangeItem for data exchanging when users link different models. BMI has standard names for their input and output data exchanging in different models. OpenGMS also has UDX for data exchanging in model integration. This research presented a data exchanging method for format transmitting and content mapping, which can be helpful for data exchanging among different standards. However, owing to heterogeneity of data exchange rules, the data in models can be problematic when reused in other models. So, a set of data preprocessing or post-processing methods should be provided in data exchanging.

The engine can be customized in different modules for the engine design. Based on the designed universal standards, the modules in the engine design can be reused in other engines. The modules (including field mapping, function conversion and component reorganization) in the engine between two standards have something in common, so it can be a base class supporting the design of all modules.

More standards could be incorporated into the method to make models interoperable as described in this paper, thereby extending the community of interoperability. This research only presented three standards for model interoperability, but there are more standards for model sharing and reuse that need to be considered. The basic universal standard, as presented in this research, would be extended accordingly. The design of these standards benefits from flexibility to enable them to be incorporated into this method.

Software availability

Software name: *Model Interoperable Engine*.
Developer: Fengyuan Zhang, Min Chen.
Year first official release: 2020.
Hardware requirements: PC.
System requirements: Windows, Linux, Mac.
Program language: Python3.7.
Program size: 5 MB.
Availability: https://github.com/franklinzhanggis/model-interoperable-engine.
License: MIT.
Documentation: https://github.com/franklinzhanggis/model-interoperable-engine/blob/master/README.md.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We appreciate the detailed suggestions and comments from the anonymous reviewers. We express heartfelt thanks to the other members of the OpenGMS team. This work was supported by NSF of China [grant numbers 41930648, 42071363, 42071361, 41871285 and U1811464].

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.envsoft.2021.105164.

References

Baig, M.R.I., Ahmad, I.A., Shahfahad, Tayyab, M., Rahman, A., 2020. Analysis of shoreline changes in Vishakhapatnam coastal tract of Andhra Pradesh, India: an application of digital shoreline analysis system (DSAS). Spatial Sci. 26 (4), 361–376.
Belete, G.F., Voinov, A., Laniak, G.F., 2017a. An overview of the model integration process: from pre-integration assessment to testing. Environ. Model. Software 87, 49–63.
Belete, G.F., Voinov, A., Morales, J., 2017b. Designing the distributed model integration framework–DMIF. Environ. Model. Software 94, 112–126.
Bulatewicz, T., Yang, X., Peterson, J.M., Staggenborg, S.A., Welch, S.M., Steward, D.R., 2010. Accessible integration of agriculture, groundwater, and economic models using the Open Modeling Interface (OpenMI): methodology and initial results. Hydrol. Earth Syst. Sci. 14, 521–534.
Castronova, A.M., Goodall, J.L., Ercan, M.B., 2013. Integrated modeling within a hydrologic information system: an OpenMI based approach. Environ. Model. Software 39, 263–273.
Chen, M., Lin, H., Hu, M., He, L., Zhang, C., 2013. Real-geographic-scenario-based virtual social environments: integrating geography with social research. Environ. Plann. Plann. Des. 40 (6), 1103–1121.
Chen, Y., Zhou, H., Zhang, H., Du, G., Zhou, J., 2015. Urban flood risk warning under rapid urbanization. Environ. Res. 139, 3–10.
Chen, M., Voinov, A., Ames, D.P., Kettner, A.J., Goodall, J.L., Jakeman, A.J., Barton, M. C., Harpham, Q., Cuddy, S.M., DeLuca, C., Yue, S., Wang, J., Zhang, F., Wen, Y., Lü, G., 2020. Position paper: open web-distributed integrated geographic modelling and simulation to enable broader participation and applications. Earth Sci. Rev., 103223
Chen, M., Yue, S.S., Lü, G., Lin, H., Yang, C.W., Wen, Y.N., Hou, T., Xiao, D.W., Jiang, H., 2019. Teamwork-oriented integrated modeling method for geo-problem solving. Model. Softw. 119, 111–123.
Chen, M., Lv, G., Zhou, C., Lin, H., Ma, Z., Yue, S., Wen, Y., Zhang, F., Wang, J., Zhu, Z., Kai, X., He, Y., 2021. Geographic modeling and simulation systems for geographic research in the new era: some thoughts on their development and construction. Sci. China Earth Sci. 64, 1207–1223.
Conde-Cid, M., Fernández-Calviño, D., Nóvoa-Muñoz, J.C., Núñez-Delgado, A., Fernández-Sanjurjo, M.J., Arias-Estévez, M., Álvarez-Rodríguez, E., 2019. Experimental data and model prediction of tetracycline adsorption and desorption in agricultural soils. Environ. Res. 177, 108607.
Danger, M., Daufresne, T., Lucas, F., Pissard, S., Lacroix, G., 2008. Does Liebig's law of the minimum scale up from species to communities? Oikos 117 (11), 1741–1751.
Drost, N., Hut, R., Van De Giesen, N., van Werkhoven, B., Aerts, J.P., Camphuijsen, J., Pelupessy, I., Weel, B., Verhoeven, S., van Haren, R., Hutton, E., Alidoost, F., van den Oord, G., Dzigan, Y., Andela, B., Kalverla, Peter, 2020, May. Coupling Hydrological models using BMI in eWaterCycle. In: EGU General Assembly Conference Abstracts, 11730.
Granell, C., Schade, S., Ostländer, N., 2013. Seeing the forest through the trees: a review of integrated environmental modelling tools. Comput. Environ. Urban Syst. 41, 136–150.
Gregersen, J.B., Gijsbers, P.J.A., Westen, S.J.P., 2007. OpenMI: open modelling interface. J. Hydroinf. 9 (3), 175–191.
Goodall, J.L., Robinson, B.F., Castronova, A.M., 2011. Modeling water resource systems using a service-oriented computing paradigm. Environ. Model. Software 26 (5), 573–582.
Harpham, Q.K., Hughes, A., Moore, R.V., 2019. Introductory overview: the OpenMI 2.0 standard for integrating numerical models. Environ. Model. Software 122, 104549.
Hutton, E.W.H., Piper, M.D., Tucker, G.E., 2020. The Basic Model Interface 2.0: A Standard Interface for Coupling Numerical Models in the Geosciences. JOSS. https://doi.org/10.21105/joss.02317.
Jiang, P., Elag, M., Kumar, P., Peckham, S.D., Marini, L., Rui, L., 2017. A service-oriented architecture for coupling web service models using the Basic Model Interface (BMI). Environ. Model. Software 92, 107–118.
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. Software 39, 274–282.
Laniak, G.F., Olchin, G., Goodall, J., Voinov, A., Hill, M., Glynn, P.W., Whelan, G., Geller, G., Quinn, N., Blind, M., Peckham, S., Reaney, S., Gaber, N., Kennedy, R., Hughes, A., 2013. Integrated environmental modeling: a vision and roadmap for the future. Environ. Model. Software 39, 3–23.
Lin, H., Chen, M., Lu, G., Zhu, Q., Gong, J., You, X., Wen, Y., Xu, B., Hu, M., 2013a. Virtual geographic environments (VGEs): a new generation of geographic analysis tool. Earth Sci. Rev. 126, 74–84.
Lin, H., Chen, M., Lu, G., 2013b. Virtual geographic environment: a workspace for computer-aided geographic experiments. Ann. Assoc. Am. Geogr. 103 (3), 465–482.
Lin, H., Chen, M., 2015. Managing and sharing geographic knowledge in virtual geographic environments (VGEs). Spatial Sci. 21 (4), 261–263.
Lü, G., Batty, M., Strobl, J., Lin, H., Zhu, A.X., Chen, M., 2019. Reflections and speculations on the progress in Geographic Information Systems (GIS): a geographic perspective. Int. J. Geogr. Inf. Sci. 33 (2), 346–367.
Ma, Z., Chen, M., Yue, S., Zhang, B., Zhu, Z., Wen, Y., Lü, G., Lu, M., 2021. Activity-based process construction for participatory geo-analysis. GIScience Remote Sens. 58 (2), 180–198.
McGrattan, K., Hostikka, S., McDermott, R., Floyd, J., Weinschenk, C., Overholt, K., 2013. Fire Dynamics Simulator User's Guide, vol. 1019. NIST special publication, 6.
Nelson, F.E., Outcalt, S.I., 1987. A computational method for prediction and regionalization of permafrost. Arct. Alp. Res. 19 (3), 279–288.
Nourani, V., Gökçekuş, H., Umar, I.K., 2019. Artificial intelligence based ensemble model for prediction of vehicular traffic noise. Environ. Res., 108852
Overeem, I., Jafarov, E., Wang, K., Schaefer, K., Stewart, S., Clow, G., et al., 2018. A modeling toolbox for permafrost landscapes. Eos, Trans. Am. Geophys. Union (Online) 99. LA-UR-18-20806).
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.
Rossman, L.A., 2010. Storm Water Management Model User's Manual. version 5.0. National Risk Management Research Laboratory, Office of Research and Development, US Environmental Protection Agency, Cincinnati, p. 276.
Serreze, M.C., 2011. Climate change: rethinking the sea-ice tipping point. Nature 471 (7336), 47.
Shi, X., Lin, H., 2020. Introduction: advances in geospatial analysis for health research. Spatial Sci. 26 (3), 217–218.
Shrestha, N.K., Leta, O.T., De Fraine, B., Van Griensven, A., Bauwens, W., 2013. OpenMI-based integrated sediment transport modelling of the river Zenne, Belgium. Environ. Model. Software 47, 193–206.
Sun, A.Y., Zhong, Z., Jeong, H., Yang, Q., 2019. Building complex event processing capability for intelligent environmental monitoring. Environ. Model. Software 116, 1–6.

Tóth, G., Hermann, T., Da Silva, M.R., Montanarella, L., 2016. Heavy metals in agricultural soils of the European Union with implications for food safety. Environ. Int. 88, 299–309.
Wang, J., Chen, M., Lü, G., Yue, S., Chen, K., Wen, Y., 2018. A study on data processing services for the operation of geo-analysis models in the open web environment. Earth Space Sci. 5 (12), 844–862.
Wang, X., Huang, R., Li, L., He, S., Yan, L., Wang, H., Wu, X., Yin, Y., Xing, B., 2019. Arsenic removal from flooded paddy soil with spontaneous hygrophyte markedly attenuates rice grain arsenic. Environ. Int. 133, 105159.
Whelan, G., Kim, K., Pelton, M.A., Castleton, K.J., Laniak, G.F., Wolfe, K., Parmar, P., Babendreier, J., Galvin, M., 2014. Design of a component-based integrated environmental modeling framework. Environ. Model. Software 55, 1–24.
Xiao, D., Chen, M., Lu, Y., Yue, S., Hou, T., 2019. Research on the construction method of the service-oriented web-SWMM system. ISPRS Int. Geo-Inf. 8 (6), 268.
Yue, S., Wen, Y., Chen, M., Lu, G., Hu, D., Zhang, F., 2015. A data description model for reusing, sharing and integrating geo-analysis models. Environ. Earth Sci. 74 (10), 7081–7099.
Yue, S.S., Chen, M., Song, J., Yuan, W.P., Chen, T.X., Lu, G.N., Shen, C.R., Ma, Z.Y., Xu, K., Wen, Y.N., Song, H.Q., 2020. Participatory intercomparison strategy for terrestrial carbon cycle models based on a service-oriented architecture. Future Generat. Comput. Syst. 112, 449–466.
Zhang, F., Chen, M., Ames, D.P., Shen, C., Yue, S., Wen, Y., Lü, G., 2019. Design and development of a service-oriented wrapper system for sharing and reusing distributed geoanalysis models on the web. Environ. Model. Software 111, 498–509.
Zhang, M., Jiang, L., Yue, P., Gong, J., 2020a. Interoperable web sharing of environmental models using OGC web processing service and Open Modeling Interface (OpenMI). Environ. Model. Software 133, 104838.
Zhang, F., Chen, M., Yue, S., Wen, Y., Lü, G., Li, F., 2020b. Service-oriented interface design for open distributed environmental simulations. Environ. Res. 191, 110225.