Environmental Modelling & Software 39 (2013) 283–294

Contents lists available at SciVerse ScienceDirect

Environmental Modelling & Software

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

The Simple Script Wrapper for OpenMI: Enabling interdisciplinary modeling studies☆

T. Bulatewicz [a,*], A. Allen [b], J.M. Peterson [d], S. Staggenborg [c], S.M. Welch [c], D.R. Steward [b]

[a] Dept. of Computing and Information Sciences, 234 Nichols Hall, Kansas State University, Manhattan, KS 66502, USA
[b] Dept. of Civil Engineering, Kansas State University, Manhattan, KS 66502, USA
[c] Dept. of Agronomy, Kansas State University, Manhattan, KS 66502, USA
[d] Dept. of Agricultural Economics, Kansas State University, Manhattan, KS 66502, USA

ARTICLE INFO

Article history:
Received 18 April 2012
Received in revised form
6 July 2012
Accepted 11 July 2012
Available online 4 August 2012

Keywords:
OpenMI
Interdisciplinary modeling
Scripting languages
Model integration
Integrated modeling

ABSTRACT

Integrated environmental modeling enables the development of comprehensive simulations by compositing individual models within and across disciplines. The Simple Script Wrapper (SSW), developed here, provides a foundation for model linkages and integrated studies. The Open Modeling Interface (OpenMI) enables model integration but it is challenging to incorporate scripting languages commonly used for modeling and analysis such as MATLAB, Scilab, and Python. We have developed a general-purpose software component for the OpenMI that simplifies the linking of scripted models to other components. Our solution enables scientists to easily make their scripting language code linkable to OpenMI-compliant models fostering collaborative, interdisciplinary integrated modeling. The simplicity afforded by our solution is presented in a case study set in the context of irrigated agriculture. The software is available online as supplementary material and includes an example that may be followed to employ our methods.

Published by Elsevier Ltd.

Software availability

Software name: SSW
Developer: GRoWE/Kansas State University
Contact address: 234 Nichols Hall, Kansas State University, Manhattan, KS, 66502, 785-532-6350
E-mail: tombz@ksu.edu
Year first available: 2012
Hardware required: Architecture independent
Required software: Windows/Linux
Program language: C#
Program size: 2 MB
Availability: Download available under MIT License at: http://code.google.com/p/simple-script-wrapper
Cost: Free

1. Introduction

The importance of integrated modeling in addressing current global environmental challenges is well established (Parson, 1995; Rotmans and Van Asselt, 1996; Harris, 2002; Pahl-Wostl, 2002; Parker et al., 2002; Jakeman and Letcher, 2003). The application of disjoint disciplinary models, methodologies, and ontologies can only offer insight into the behavior of processes within a specific domain precluding investigation into the interactions between processes in different domains. Comprehensive simulations in which multidisciplinary models are collectively applied can capture these process interactions and feedbacks as they propagate across domains. Integrated modeling that bridges traditionally isolated domains is an emerging field of study and is necessary for such initiatives as the European Union Water Framework Directive and studies envisioned by the Community Surface Dynamics Modeling System (CSDMS) team at the University of Colorado, Boulder (Peckham and Hutton, 2009) and the Consortium of Universities for the Advancement of Hydrologic Science (CUASHI, 2012) such as the Community Hydrologic Modeling Platform (CHyMP) (Famiglietti et al., 2011).

An effective approach to interdisciplinary integrated modeling is *model linking* (or *coupling*) in which models remain independent

☆ Thematic Issue on the Future of Integrated Modeling Science and Technology.

* Corresponding author. Tel.: +1 505 490 9681.
E-mail addresses: tombz@ksu.edu (T. Bulatewicz), andya@ksu.edu (A. Allen), jpeters@ksu.edu (J.M. Peterson), sstaggen@ksu.edu (S. Staggenborg), welchsm@ksu.edu (S.M. Welch), steward@ksu.edu (D.R. Steward).

1364-8152/$ – see front matter Published by Elsevier Ltd.
http://dx.doi.org/10.1016/j.envsoft.2012.07.006

from one another and communicate to collectively carry out a simulation. The Open Modeling Interface (OpenMI) (Moore and Tindall, 2005; Gregersen et al., 2007) is an emerging standard that facilitates model integration through the dynamic linking and execution of models. The expertise necessary and the level of effort required to enable an existing model to work with the OpenMI (or create a new model that works with the OpenMI) are influenced by the design of the model, the programming language that it is written in, and the skill of the programmer. In cases where the programming language of a model differs from that of the software tools being used to link and execute it, language interoperability techniques must be incorporated into the model to bridge the languages. This can make the process of conforming a model to the OpenMI prohibitively difficult in cases where a scientist's programming skills lie within the predominant language of his or her discipline (e.g. MATLAB; Scilab or Python). In addition, many academic programmers are not very familiar with object oriented programming concepts, making it difficult to implement the OpenMI interfaces. Scientists must be free to use the tools and methods customary in their discipline without sacrificing simple integration and collaboration with scientists in other domains.

Scripting languages such as MATLAB are pervasive throughout research and education intersecting many different disciplines. Languages for mathematics (MATLAB; Scilab; Mathematica) and statistics (SAS; SPSS) are common in engineering, in applied sciences such as agronomy, and in social sciences such as economics (Kendrick and Amman, 1999) and sociology. General-purpose scripting languages (Perl; Python; Ruby) are highly flexible and have varied uses such as building Geographic Information Systems (GIS) workflows and creating frameworks for optimization and parameter estimation. Despite the widespread use of scripting languages, it remains an open research question how to make scripts OpenMI-compliant.

We have developed a general-purpose software component called the Simple Script Wrapper (SSW) that enables interoperability between OpenMI components and the scripting languages MATLAB, Scilab, and Python. Our solution enables scientists to simply integrate their model codes with the codes of others fostering collaborative, interdisciplinary integrated modeling. Note that models or modules that have been integrated into our SSW also have the potential to be integrated with other SSW enabled tools written in the same language (e.g. MATLAB), enabling further interoperability and development of comprehensive modeling tools. The SSW extends the Simple Model Wrapper (SMW) (Castronova and Goodall, 2010) by adding support for scripting languages. We demonstrate our solution in an example that brings together groundwater, economics, and agricultural models for multidisciplinary studies. The software is available online as supplementary material and includes an example that may be followed to employ our methods.

In the following section we introduce the OpenMI and discuss approaches to supporting different programming languages. This is followed by an explanation of how the SSW is used and implemented. We then describe the process of creating linkable scripts for the SSW and demonstrate its operation through an ongoing case study.

2. Background

2.1. Model linking

Model integration remains a challenging task due to the inherent differences between models. The incompatibilities between models must be resolved, such as variation in spatial and temporal scale and differences in the programming languages in which they are implemented. Creating integrated models by merging them together into a single source code can be complicated (particularly in the case of legacy models) and the software maintenance (e.g. fixing errors and updating dependencies) of many copies of a model becomes increasingly challenging. Integrating models by linking them together keeps them independent from one another and improves reusability. There has been considerable work toward the development of methods and tools for model linking (Larson et al., 2001; David et al., 2002; Buis et al., 2006; Bulatewicz and Cuny, 2006; Ford et al., 2006; Joppich and Kurschner, 2006; Gregersen et al., 2007).

In most approaches to model linking the models are modified and given the ability to exchange data either in an ad-hoc or standardized way. Standards enable models to be shared between researchers and foster collaboration and the creation of a common repository of linkable models. One of the primary ways in which standards differ from one another is in how they balance the expressiveness of the interactions between models and the level of specification necessary to link them together. At one extreme scientists have very fine control over how models execute and exchange data but require a highly detailed specification to link them. At the other extreme models may only exchange limited kinds of data and execute in prescribed ways but require a simple specification to link them. The latter is well-suited to use cases where model linkages are frequently changed for experimentation or prototyping and when it is desirable to minimize the skills and knowledge necessary to link models. The Open Modeling Interface is an example of such a standard and we have found it to be an effective means to integrating multidisciplinary models in previous work (Bulatewicz et al., 2010).

2.2. The Open Modeling Interface

The Open Modeling Interface (OpenMI) defines a standard way for software components to exchange data with each other and coordinate their execution. It defines a set of capabilities that a model must possess in order for it to be linked to other models. These capabilities are both descriptive, to support the task of specifying model interactions at the domain level, and functional, to support the execution of a set of linked models. To fulfill the descriptive requirements, a component must be capable of providing a list of the domain quantities that it can provide and those that it can consume, along with the units and spatial distribution of each. These are called *output exchange items* and *input exchange items*, and in the case of model components, there is usually one output item for each quantity that it simulates and one input item for each of its inputs. To fulfill the functional requirements, a component must possess a GetValues method through which it provides data (that correspond to the output exchange items) at runtime. A model that meets the requirements of the interface is called a *linkable component* and can be dynamically linked to other models at runtime. Using visual software tools (provided by the OpenMI Association Technical Committee), a scientist chooses a set of linkable components of interest, interactively specifies the quantities to be exchanged between them, and then executes the linked model. Components can be added and removed and the links between them reconfigured, facilitating experimentation and rapid prototyping of linked models.

The GetValues method (member of the OpenMI.Standard.ILinkableComponent namespace) has two parameters and returns an array of numbers. The parameters specify the input link on which the call should be made and the simulation time at which input is required. As each link represents the transfer of a spatially-distributed quantity, the parameters collectively request a set of numbers that represent the state of a quantity at a single point in time over some spatial domain, as illustrated in Fig. 1. A

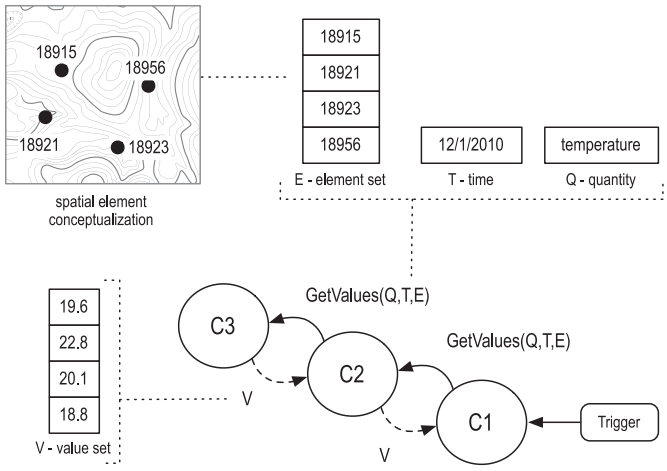

Fig. 1. Operation of GetValues. Solid lines indicate function calls and dashed lines indicate the flow of data.

quantity is represented by an object with several properties such as a textual identifier and units information, the time is represented by a modified Julian date, and the spatial domain is represented by a list of spatial elements called an *element set*. An element set is a list of geo-referenced spatial elements (i.e. points, lines, or polygons) or non-spatial identifiers. The GetValues function returns a list of floating-point numbers called a *value set* that describes the state of a single quantity at a specific point in time across an element set. Each number in a value set represents the state of the quantity and time at a different geographic location. The number at a particular index in the value set represents the spatial element in the corresponding index of the element set.

The GetValues method not only provides a means for the exchange of data between components but it also provides a means for the coordinated execution of a set of components at runtime. When the GetValues method is called on a component it must return the value set that corresponds to the requested quantity, time, and locations. If the component has not yet simulated the requested point in time then it must perform a series of time steps to advance its simulation time to that point. Prior to the simulation of each time step, the component may call GetValues on other components to collect the inputs necessary for the computation of the step. Those components may in turn perform time steps themselves and call GetValues on other components before they return the requested value sets. A linked simulation begins when the GetValues method of one of the components is called. The OpenMI Software Development Kit (SDK) includes a special component called a *trigger* that can be used to initiate the first call. The trigger repeatedly calls GetValues on the component until the component's simulation time reaches the simulation end time defined by the scientist. A component always waits until each call to GetValues completes before it computes a time step. In this way the components take turns executing and pull data from each other until the component attached to the trigger reaches the simulation time requested by the trigger.

The structure of a time-stepped (simulation) model that has been made OpenMI-compliant is typically organized into three primary functions called Initialize, Perform Time Step, and Finish (Gijsbers et al., 2005). The Initialize function reads the model inputs and prepares the simulation and is called once at the start of a simulation. The Perform Time Step function simulates a single time step and is called as needed by the GetValues function. The Finish function writes the final output and is called once at the conclusion of a simulation.

There are 18 functions that every linkable component must possess (Gijsbers et al., 2005) to implement the OpenMI.Standard.ILinkableComponent interface. Ten of these make up the functional requirements and facilitate the execution of the linked model during runtime (such as GetValues) and eight of them constitute the descriptive requirements (such as GetModelDescription). The OpenMI SDK includes a collection of classes (within the Oatc.OpenMI.Sdk.Wrapper namespace) that perform some of the underlying responsibilities of a component such as adding links, caching exchanged data, and performing time steps in response to calls to the GetValues function, but several time-related functions and descriptive functions must still be added to a model. The Simple Model Wrapper (SMW) (Castronova and Goodall, 2010) was developed independently from the OpenMI SDK as a 3rd party extension to simplify the model creation process. It abstracts the Wrapper class and implements the 15 additional time-related and descriptive functions so that a model need only include the three primary functions thus minimizing the amount of programming necessary to create a linkable component. The SMW can implement the time-related functions because it assumes that the model uses a fixed-length time step and it keeps track of the simulation time and advances it as necessary. The SMW can implement the descriptive functions because it has the ability to read the model metadata from a *configuration* file. In this way the SMW facilitates the creation of linkable components requiring that the scientist (1) adds 3 functions to a model's source code and (2) creates a file that contains information about the model.

2.3. Language interoperability

Although the OpenMI defines a standard way to link models together it does not address language differences between models. The OpenMI concepts are not language dependent but its design mandates that a set of models and the tools used to link and execute them be implemented in a common language. In cases where model languages differ, language interoperability techniques can be utilized. This is common in an interdisciplinary context as different disciplines use different modeling methodologies, languages, and tools (see the description of Babel in Peckham and Hutton (2009) for an example). Scripting languages (also called dynamic languages) are an important class of programming languages that are pervasive across many different scientific disciplines. Although lacking a formal definition, scripting languages are often identified by their purpose of coordinating the execution of programs (Ousterhout, 1998).

There are two broad approaches to enabling support for scripting languages, and alternative programming languages in general, for the OpenMI as illustrated in Fig. 2. Given a language *A*, either the OpenMI standard may be implemented for the language (top) or an existing implementation for a different language *B* may be used in conjunction with language interoperability (bottom). The former approach is well-suited for cases where models are written in a common language whereas the latter approach accommodates models written in different languages.

The standard itself (org.OpenMI.Standard namespace) is defined in terms of the object-oriented programming concept of an interface, which defines a set of operations that a class must implement. In addition, the OpenMI SDK (Oatc namespace) provides an implementation of the OpenMI Standard that relies on the object-oriented capabilities of a language, such as inheritance. Object-oriented languages are thus predisposed for implementation of the standard and are required to implement the existing SDKs. Implementations of the standard and SDK exist for both C# and Java

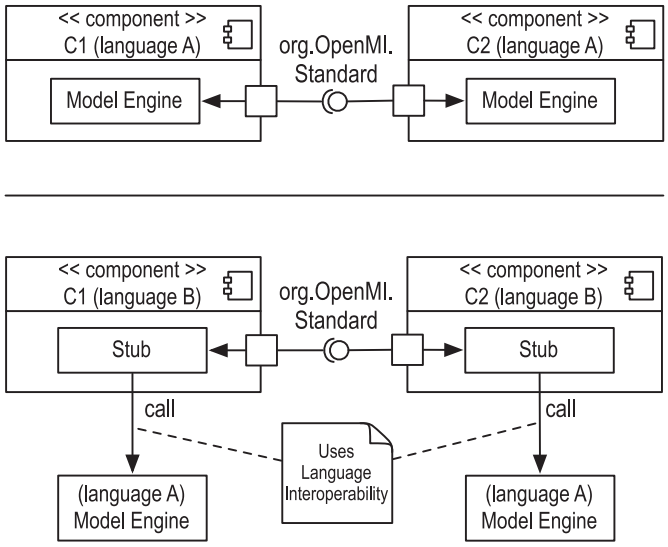

Fig. 2. UML class diagram indicating two alternative approaches to supporting a programming language *A* for the OpenMI.

and additional implementations could be created for other object-oriented languages as well.

The fundamental concepts behind the OpenMI are not language-dependent, such as the pull-based execution model and input/output definitions composed of quantities and element sets. By conceptually broadening the object-oriented interface to a generalized *contract* it would be possible to create a variant of the OpenMI for non-object-oriented languages. An Open Modeling Contract could be envisioned that defines similar operations to the OpenMI standard but does so without requiring a language to possess an explicit interface construct. This could be accomplished in a variety of ways. In the context of a dynamic language, for example, a model could provide a file that specifies the name of each of its functions that implements each operation in the contract. At runtime, other models would use this file to lookup the function name to call on that model for a particular operation.

In the case of a language for which implementing the standard is not feasible or desirable, or in the case of models written in differing languages, components may implement the standard in a common language and internally interoperate with other languages as illustrated in Fig. 2 (bottom). In this case both components implement the interface in a common language enabling them to directly call methods on one another (see Damgaard (2004) for an example). Within each component, the responsibility of performing the implementation of an interface method is delegated to a set of model functions written in another language through the use of language interoperability techniques. This interoperability can be accomplished in many different ways depending on the characteristics of the two languages but can be broadly classified into intra-process and inter-process approaches. In cases where different languages are compiled into compatible object code or bytecode, functions can be called directly across languages within a process (with care to properly marshall arguments as necessary). One example of this is the native object code generated by many C and Fortran compilers. Another example is the bytecode generated by the Microsoft .NET framework for languages such as C# and Visual Basic, which is executed on the Common Language Runtime (CLR) virtual machine. The Microsoft Dynamic Language Runtime (DLR) extends the CLR to dynamic languages such as Python, JavaScript, and Ruby while preserving interoperability with the static languages of the CLR. Interoperability between virtual machine-based execution environments such as the CLR and Java Virtual Machine (JVM) with other execution environments may be possible through capabilities provided by the virtual machine. Examples include the Platform Invocation Services (PInvoke) capability of the CLR and the Java Native Interface (JNI) capability of the JVM that allow code running within a virtual machine to call and be called by native applications and libraries written in other languages such as C, C++, and assembly.

Interoperability can also be achieved through inter-process communication between processes whose underlying program implementations are in different languages. Operating systems typically provide several means for inter-process communication such as named pipes, shared memory, and sockets. In addition, files may be used as a means for communication across processes although this may incur limitations in the frequency of data exchanges (see Bulatewicz et al. (2010) for an example). The use of sockets for inter-process communication enables interoperability between processes executing on independent machines through remote method invocation. Inter-process communication approaches generally offer lower performance compared to the intra-process approaches because execution must be dispatched across processes (and possibly across machines) and data must be copied across address spaces.

Utilizing the OpenMI in the context of multidisciplinary modeling requires the ability to link models written in different languages, as programming languages differ across domains. For this reason it is necessary to utilize language interoperability. In addition, utilizing a popular implementation of the interface fosters collaboration between modelers. To maximize efficiency and performance, the SSW uses the most direct form of interoperability available between C# and MATLAB, Scilab, and Python.

The SSW utilizes the Engine Library of the C/C++ and Fortran API provided by MATLAB to instantiate an interpreter and execute scripts within it. The API is provided by MATLAB in the form of native library files (included with the MATLAB application) requiring the SSW to use PInvoke to call the unmanaged dynamic link library functions from a managed context and marshall the data exchanged in those calls. The interpreter runs as an independent background process and the MATLAB API handles the inter-process communication. An independent interpreter process is started for each instance of the SSW component to prevent global data from being accessed across models.

The SSW utilizes the Scilab Gateway API provided by Scilab which has similar functionality to the MATLAB Engine Library API and is also provided in the form of dynamic link libraries (included with the Scilab application) that are accessed using PInvoke. Rather than run as a separate process, the Scilab interpreter is instantiated within the memory of the process in which the SSW is loaded. A single Scilab interpreter may exist within the memory space of a process, so since all components are loaded into a single memory space (per OpenMI) the interpreter is shared among all instances of the SSW. This means that global variables and operations on them affect all instances of the SSW component using Scilab and care must be taken to not use global variables with the same name in different models and to not clear all global variables.

The SSW utilizes IronPython (IronPython) to create a Python interpreter and execute scripts within it. IronPython is built on the Dynamic Language Runtime (DLR) of the Microsoft .NET framework enabling the SSW to instantiate a Python interpreter engine directly from C# and execute Python functions via method calls. Each instance of the SSW component creates its own instance of a Python interpreter. As IronPython is implemented in .NET, model scripts cannot access the CPython modules commonly used in Python programming (although the .NET API is accessible). The SSW can be extended to support both additional scripting languages and alternative means for executing scripts of a single

language making it possible to add support for the execution of Python scripts via CPython.

Scripting languages present unique challenges and unique opportunities for creating OpenMI-compliant linkable components from scripted model codes. Scripting languages complicate language interoperability due to their reliance on an interpreter program for execution. The use of an interpreter, though, affords the opportunity for a single component to execute any scripted model, alleviating the need to create a separate linkable component for each model. A single component that can interoperate with different script interpreters can serve as a general-purpose means to making any script linkable. Thus, our SSW is opening the OpenMI world to those who develop scripting language computer code. In this work we present an approach to creating linkable model components from scripting languages that capitalizes on the dynamic nature of these languages obviating the need for model-specific components and allowing model scripts to be used as OpenMI-compliant models without the need for programming outside the model code itself. In the following section we describe the process of creating a linkable script and then present the design of the SSW.

3. Methods

3.1. Using the SSW

The process of creating a linkable script consists of two steps: (1) write the configuration file and (2) write the three primary script functions. After the linkable script has been created, the component can be added to a composition, linked to other OpenMI components, and executed.

An example of a configuration file (Castronova and Goodall, 2010) taken from the case study in Section 4 is shown in Fig. 3 (simplified). A configuration file includes three sections that describe a model's basic properties, temporal characteristics, and exchange items. The basic properties define a unique identifier for the model, its name, and its scripting language. The temporal characteristics specified in the configuration file includes the earliest and latest points in time that the model is capable of simulating (called the *time horizon*), the time step length and units, and the relative input time at which inputs from other models are collected. Each exchange item identifies a quantity, an element set, and a unit along with its dimensions (not shown). An element set is typically defined in a separate file (so that it can be referenced from multiple configuration files) and consists of a list of elements described in XML. Each element is defined by an identifier and a type (point, line, polygon, or non-spatial) along with any geo-referenced points necessary to describe the geometry of the element. The SMW includes support for automatically generating the element set file from a GIS.

An example of a MATLAB Perform Time Step script is shown in Fig. 4. All of the inputs to the model (as defined in the configuration file) are collected from the other components by the SSW prior to the Perform Time Step script being called and are accessible to the script for use in the calculation of the time step. Thus the script can assume that the inputs from the other components that correspond to the (specified offset of the) current simulation time have already been received. At the end of the script, the outputs from the model (as defined in the configuration file) are saved. The SSW increments the simulation time by one time step after each invocation of the Perform Time Step script.

The SSW provides a set of *accessor* functions (Table 1) that can be called by a script to obtain input values from other components, save output values to be later delivered to other components, and obtain other information such as the current simulation time. The sswGetInput function returns the value of an input quantity for a specific element. The sswSetOutput function stores the value for an output quantity and element so that it may later be provided to another component. The sswGetCurrentTime function returns the current simulation time. The

```
<Configuration>
  <ModelInfo>
    <ID>Economic Model</ID>
    <Description>Economic model</Description>
    <ScriptingLanguage>MATLAB</ScriptingLanguage>
  </ModelInfo>
  <TimeHorizon>
    <StartDateTime>01/01/1990 00:00:00</StartDateTime>
    <EndDateTime>01/01/2004 00:00:00</EndDateTime>
    <TimeStep>31536000</TimeStep>
    <InputTimeOffset>0</InputTimeOffset>
  </TimeHorizon>
  <ExchangeItems>
    <InputExchangeItem>
      <Quantity>
        <ID>SatThick</ID>
        <Description>Saturated thickness</Description>
        <ValueType>Scalar</ValueType>
      </Quantity>
      <ElementSet>
        <ID>SheridanCoWells</ID>
        <Description>Points of diversion in Sheridan County, KS</Description>
        <XmlFilePath>ElementSets.xml</XmlFilePath>
        <Version>1</Version>
      </ElementSet>
    </InputExchangeItem>
    <OutputExchangeItem>
      <Quantity>
        <ID>CropChoice</ID>
        <Description>Crop choice</Description>
        <ValueType>Scalar</ValueType>
      </Quantity>
      <ElementSet>
        <ID>SheridanCoWells</ID>
        <Description>Points of diversion in Sheridan County, KS</Description>
        <XmlFilePath>ElementSets.xml</XmlFilePath>
        <Version>1</Version>
      </ElementSet>
    </OutputExchangeItem>
  </ExchangeItems>
</Configuration>
```

Fig. 3. An example of a configuration file. Unit and dimension elements not shown.

```
function sswPerformTimeStep()

global input_folder;
global current_year;
global well_filename;
global coeff_filename;

% get the current time from the SSW
current_time = sswGetCurrentTime();
current_year = current_time(1);

% we want to simulate the next year (since after this function is called,
% the SSW will advance the time of this component to the next year and
% assumes these values correspond to that year)
current_year = current_year + 1;

% set the paths to the input files for this year
well_filename = sprintf('%s/data_wells.txt', input_folder);
coeff_filename = sprintf('%s/data_crop_coefficients.txt', input_folder);

% get the size of the element set
element_count = sswElementCount('SatThick');

% get the saturated thickness inputs from the SSW
global sat_thick;
sat_thick = [];
for i=1:element_count
    element_id = sswElementIDAtIndex('SatThick', i);
    sat_thick(i,1) = sswGetInput('SatThick', element_id);
end

% solve the regression equation
[crop_choices crop_names probabilities] = SolveCropChoices(current_year);

% write the output files for this time step
WriteFiles(crop_choices, crop_names, probabilities, current_year);

% save the calculated results for the SSW
for i=1:element_count
    element_id = sswElementIDAtIndex('CropChoice', i);
    ExchangeOutSet('CropChoice', element_id, crop_choices(i));
end

return
```

Fig. 4. An example of a MATLAB model script for the ssw Perform Time Step function.

sswGetScriptFolder function returns the absolute path to the folder where the script files are located from which input files may be read and output files written. The sswElementCount function returns the number of elements in a quantity's element set and the sswElementIDAtIndex function returns the identifier of an element at a particular index in the element set.

3.2. Implementation of the SSW

The Simple Model Wrapper (SMW) simplifies the process of creating a linkable component that conforms to the OpenMI by abstracting the interface to consist of 3 methods: Initialize, Perform Time Step, and Finish. The remaining interface methods are replaced with a configuration file that provides the model metadata (e.g. the list of inputs to a model). The metadata is read by the component at runtime and thus may be changed without recompiling the component itself. This is necessary if a component is to be capable of representing any model, as metadata differs between models. For this reason we utilized the SMW in the implementation of the SSW. Whereas the SMW is a code library that is utilized to develop components, the SSW is a complete OpenMI-compliant component that is downloaded and linked to other components.

The SMW was initially created for the 1.4 version of the OpenMI standard and thus so too the SSW. The methods employed by the SSW may be adapted to version 2.0 of the standard making it possible to link scripted models in the same way (which is not directly supported in 2.0) and is currently in progress.

Our utilization of the SMW required two minor extensions. First, we added a ScriptingLanguage element to the configuration file that is used to specify the language of a particular set of scripts. Second, we added an InputTimeOffset element to the configuration file that allows the scientist to specify the point in simulation time at which inputs should be collected from other components. The input time defaults to the current simulation time but some models require inputs from an earlier or later point in time, so the InputTimeOffset value allows an offset from the current simulation time to be specified as a number of time steps. For example, specifying a value of 1 indicates that inputs collected on a given time step should reflect the following time step (i.e. the point in time that is being computed).

The SSW extends and inherits the functionality of both the OpenMI SDK's Oatc.OpenMI.Sdk.Wrapper.LinkableEngine class and the Simple Model Wrapper's SMW.Engine class as shown in Fig. 5. The SMW.Engine (1) reads and provides the contents of the configuration file via the appropriate interface methods and (2) temporarily stores both the input data received from other components and the output data produced by its child class, effectively serving as a data relay between the LinkableRunEngine and SSW.Engine. The SSW.LinkableEngine class completes its parent class by providing an implementation of the SetEngineApiAccess method which simply creates an instance of the SSW.Engine class. The SSW.Engine class completes its parent class by providing implementations of the Initialize, PerformTimeStep, and Finish methods and contains an instance that conforms to the ILanguageAdapter interface which provides interoperability with a specific scripting language. The Initialize method instantiates the appropriate language adapter and the Finish method destroys it. The Perform Time Step method instructs the interpreter to perform a time step.

The ILanguageAdapter interface consists of 5 methods: Initialize, Finish, SetValues, GetValues, and Perform Time Step. Each realization of the interface must be capable of starting an interpreter, creating and reading variables inside the interpreter, and issuing commands such as invoking script functions. The Initialize method creates the MATLAB, Scilab, or Python interpreter and loads the model scripts into it. It then creates an array variable in the interpreter for each input and output exchange item so that the model scripts can access the input data and have a place to store their outputs. The SetValues and GetValues methods write and read

Table 1
Accessor functions callable from scripts.

Function	Description
sswGetInput	Returns values from other components
sswSetOutput	Saves values for other components
sswGetCurrentTime	Returns the current simulation time
sswGetScriptFolder	Returns the path to the script folder
sswElementCount	Returns the size of the element set
sswElementIDAtIndex	Returns the identifier of an element at an index

to these interpreter variables. It dynamically generates the accessor function code tailored to the input and output exchange items and loads them into the interpreter. It also calls the Initialize script that prepares the model for execution. The SSW may be extended to support additional scripting languages (and alternative implementations for a single language) through the creation of new language adapters.

3.3. Operation of the SSW

The operation of the SSW is illustrated in Fig. 6. During the execution of a linked model, components call the GetValues function on the SSW component (Data Is Requested of Fig. 6) requesting a quantity across an element set at a particular time.

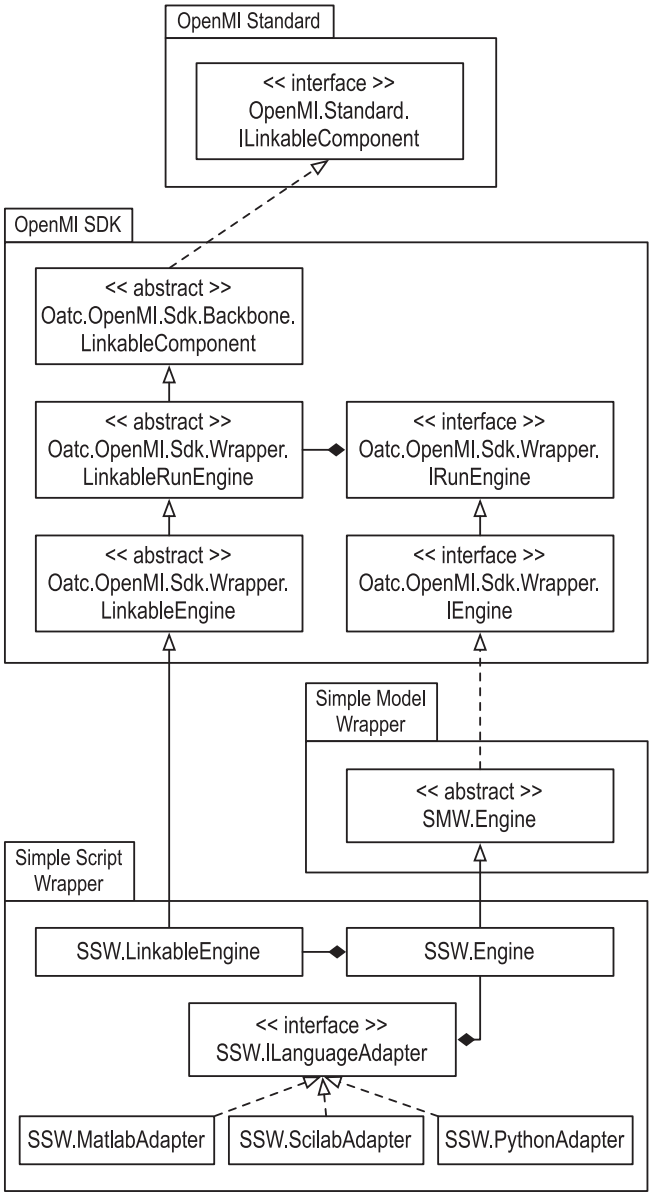

Fig. 5. UML class diagram specifying class relationships within and across packages.

The requested time is compared to the current simulation time to determine whether the requested data has already been simulated or if it is necessary to advance the model forward. If the requested data has not yet been simulated a call to GetValues is made to the components that are configured to provide input and the obtained data is saved. Once all the input data has been obtained and saved, the interpreter is instructed to perform a time step and the results of the time step are saved. The updated simulation time is compared to the original requested time and if the data has been simulated it is returned to the original requesting component.

Fig. 7 provides additional detail in how a time step is carried out (corresponding to Perform Time Step in Fig. 6). The SSW.Engine relays the input data obtained from other components to the adapter object via a call to SetValues (2 in the Fig. 7). The adapter in turn inserts the values into the interpreter engine using the appropriate language-specific API function (2.1 in the Fig. 7). The SSW.Engine then calls Perform Time Step on the adapter (3 in the Fig. 7) which in turn calls a language-specific API function that executes the ssw Perform Time Step script within the interpreter engine (3.1 in the Fig. 7). During the execution of the script it calls the accessor functions (Table 1) to read the input data and save its output data. Although the variables could be accessed directly, the use of accessor functions allows the model script to request quantities by name rather than the mangled internal variable name used by the SSW. The SSW.Engine calls GetValues on the adapter (4 in the Fig. 7) which in turn calls a language-specific API function that reads the results data from the interpreter variables (4.1 in the Fig. 7).

This is a general design that can be applied to any dynamic scripting language that provides an API that external programs can use to create and interact with an interpreter. The implementation of the SSW component, however, must include support for a specific set of languages due to differences in the interpreter API of each language. In our current implementation the SSW component supports MATLAB, Scilab, and Python under Windows and Linux (partial).

4. Case study

In this study we demonstrate how the SSW can be utilized to create an integrated model that is composed of domain-specific models written in different programming languages. The application of the SSW to interdisciplinary investigations begins with the conceptualization of a problem as a set of domain models that interact. This is followed by the identification of the quantity exchanges between the models that represent process interactions. The processes are then modeled using a scripting language with the given input and output quantities and the code is designed around the basic SSW exchange functions. The scripted models are then linked together using the SSW and the simulation is performed. We follow this general 4-step schema in the following sections in our presentation of an example of how to utilize the SSW for an important interdisciplinary challenge with global significance.

4.1. Conceptualization

Groundwater is of primary importance in arid and semi-arid regions around the world for both human consumption and crop production. In many areas of intensive irrigation natural recharge is not sufficient to compensate for groundwater withdrawals and results in diminishing groundwater stores and eventually depletion of the aquifer (Konikow and Kendy, 2005; Foster and Loucks, 2006; Scanlon et al., 2007; Shah et al., 2007; Rodell et al., 2009; Wang et al., 2009). Irrigated agricultural systems involve close interactions between biological, hydrologic, and economic processes and changes from within these processes or from exogenous factors can influence the availability and sustainability of water, crop production, and economic profitability. By incorporating these multidisciplinary processes into a comprehensive simulation the cross-domain impacts may be investigated and used to inform the way in which water consumption may be transitioned toward sustainable use.

For the purposes of this study we abstracted the irrigated agricultural system into 3 disciplinary areas such that processes within each are simulated by different domain-specific models and interact on an annual basis, as illustrated in Fig. 8. A groundwater model simulates the elevation and movement of groundwater in response to changes in natural or anthropogenic forcings such as

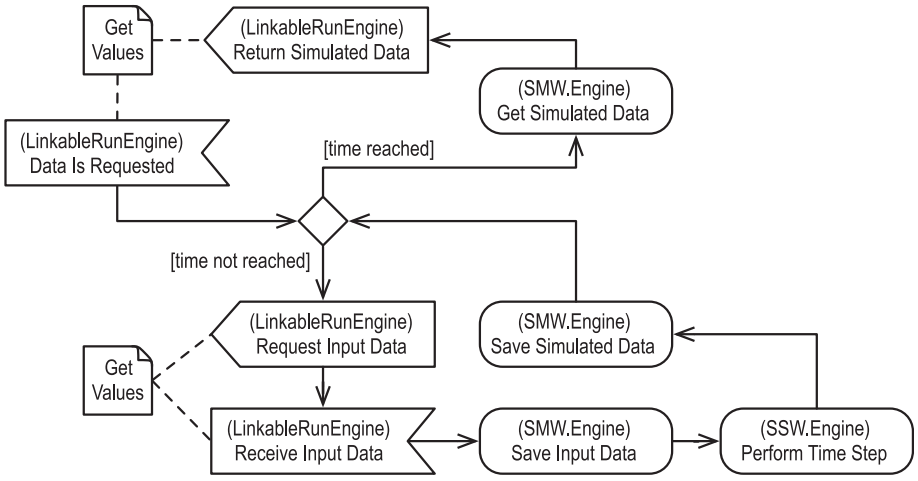

Fig. 6. UML activity diagram specifying the behavior of the SSW. Responsible parties are indicated in the annotations of each action.

climate change or changes in withdrawal rates (Steward, 2007; Steward et al., 2009). The simulation of crop growth (biomass) and irrigated water use is provided by the EPIC (Williams et al., 1990; Williams, 1995) model. An economic decision-making model predicts the choice of which crop is planted (alfalfa, corn, sorghum, or soybean) based on several external and internal factors such as crop prices, energy prices (which influence pumping costs), soil attributes, and aquifer saturated thickness (Hendricks, 2007).

4.2. Quantity exchanges

There are two direct interactions between the modeled processes: (1) the extraction of water from the aquifer for irrigation, and (2) the decision of which crop to plant which is based in part on the availability of groundwater. These interactions correspond to three exchanges of data between the models. At the start of each annual growing season the groundwater model provides the saturated thickness of the aquifer to the economic model. The economic model incorporates this information into its decision of the crop to plant for each parcel and then provides these choices to the crop model. The crop model simulates the growth of the crops on each parcel over the course of the year and provides the groundwater model with the amount of water pumped from the aquifer. The groundwater model uses this information to predict the water depth at the end of the year. This circular exchange of data is repeated for each year of the simulation period.

4.3. Linkable model scripts

There are two steps to creating a linkable scripted model: (1) create a configuration file, and (2) write the model scripts. The following sections describe these two steps using the economic model as an example.

4.3.1. Creating a configuration file

The configuration file specifies a model's metadata organized into 3 sections that describe basic model information, temporal characteristics, and the data that can be exchanged with other components, as shown in Fig. 3. Basic model properties are specified in the ModelInfo element. We chose to identify the model by

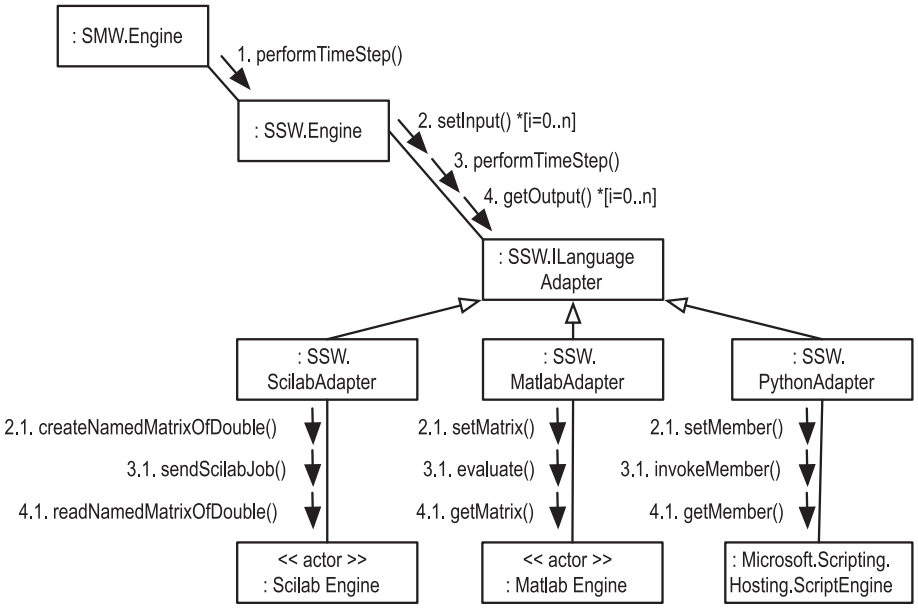

Fig. 7. UML communication diagram specifying the sequence of steps performed by the SSW.Engine when performing a time step.

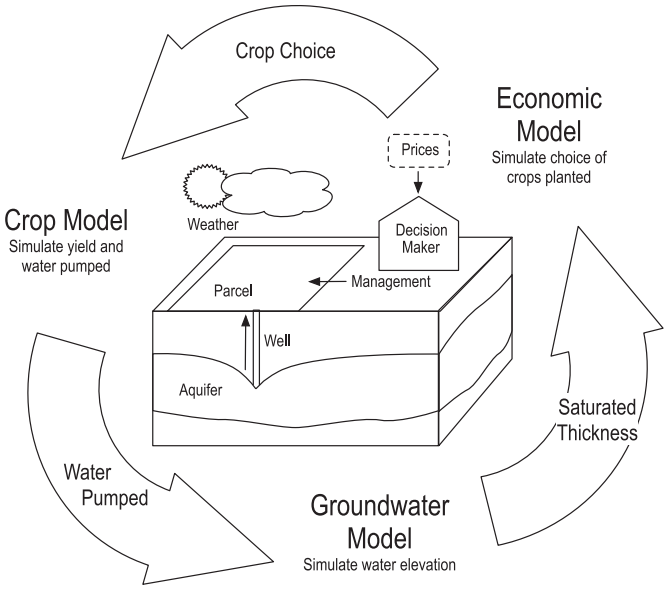

Fig. 8. System conceptualization and exchanges between models.

the name "Economic Model" and set the name of the scripting language to be "MATLAB" in this case.

The temporal characteristics of the model are specified in the TimeHorizon element. The time span that the model is capable of simulating is specified by the StartDateTime and EndDateTime elements and in this case the model is calibrated to the time period 1991 to 2004. The time step length is set to 1 year (expressed in seconds) via the TimeStep element and the InputTimeOffset is set to 0 to indicate that inputs from other models should correspond to the current time step, since the model uses the groundwater information for a given year to predict the crop choice of the following year.

The inputs and outputs of the model are specified in the ExchangeItems element. Each exchange item is described by a quantity, element set, dimension, and units (the latter two are not shown in the figure). The quantity ID both appears in the user interface of the software used to link models together and is used within model scripts to identify a specific input or output. In this case the configuration file includes one input exchange item that represents the saturated thickness data from the groundwater models and one output exchange item that represents the choice of which crop is planted (encoded as sentinel values 0, 1, 2, and 3). The elements of the element set are specified in a separate XML file (not shown in the figure). In the case of the economic model the element set consists of a collection of 2D points that correspond to the locations of the parcels within the study area. This file accompanies the model script files and is read by the SSW component during initialization.

4.3.2. Writing model scripts

Following the design of the OpenMI, the SSW expects a model's source code to be logically organized into 3 parts. The model source code must be provided to the SSW as 3 script files each of which contains one of the functions sswInitialize, ssw Perform Time Step, and sswFinish. The SSW reads these script files during initialization and executes the functions at runtime.

The sswInitialize function typically reads input files and initializes a model's internal state variables. The ssw Perform Time Step function calculates a time step. The sswFinish function closes any resources in use and writes final output files as necessary. These functions may call the accessor functions, given in Table 1, to obtain information about the state of the SSW component, such as the current simulation time, or the path to the folder in which the scripts are located (to, for example, read input files and write output files). The ability to obtain the current simulation time from the SSW component is necessary because the component (specifically the functionality provided by the SMW) keeps track of the model's simulation time and advances it as necessary.

The accessor functions also provide a means for a script, typically the ssw Perform Time Step function, to obtain the input data provided by other components and save output data calculated during a time step so that it can be made available to other components. Thus the source code of this function typically has 3 parts that (1) call the sswGetInput function to obtain input data, (2) perform the computation of a time step, and (3) call the sswSetOutput function to save the results of the computation. These regions are apparent in the ssw Perform Time Step function of the economic model, shown in Fig. 4. The function begins by obtaining the current simulation time and locating the necessary input files. It then calls the sswGetInput function (passing one of the quantity ID's specified in the configuration file) to obtain the saturated thickness value for each spatial element in the element set. The calculation of the time step is performed by the SolveCropChoice function which solves the regression equation for each spatial element and returns the crop choices as an array. The function concludes by calling the sswSetOutput function for each spatial element using the quantity ID "CropChoice" as defined in the configuration file. The sswSetOutput function saves a copy of the values in a special variable that is read by the SSW following the completion of the ssw Perform Time Step function.

Model scripts may be written and tested separately from the SSW outside of a linked environment by supplying placeholder functions in place of the accessor functions. For example, when writing the scripts for the economic model, we wrote a surrogate function called sswGetInput that read saturated thickness data from a file. This way we could test the ssw Perform Time Step function, which calls the sswGetInput function, without the SSW. After we verified that the operation of the model was correct, we removed the surrogate functions and the scripts were ready for execution by the SSW. The SSW dynamically generates the accessor functions at runtime for each set of model scripts based on the input and output exchange items defined in the configuration file.

4.4. Performing the simulation

We created a configuration file and model scripts for the Scilab groundwater model in the same way as the economic model described above and configured all 3 models for the study region and time period (Steward et al., 2009; Bulatewicz et al., 2009; Hendricks, 2007). We used an existing OpenMI component for the EPIC model (Bulatewicz et al., 2010).

We chose Sheridan County, Kansas as the study region, as the Ogallala Aquifer has supported irrigated agriculture in this county for over 50 years, as well as the surrounding semi-arid grasslands of the Central Plains of the United States. The scale of a single county (approximately 50 km $\times$ 50 km) is the typical aggregation level for economic data such as prices and yields and the chosen county reflects the groundwater properties, economic context, and agricultural practices for the region.

In this retrospective study, we show how cropping patterns and water use would have changed over the years 1991–2004 had energy prices been higher during that period. Since that time, energy prices have increased substantially and cropping patterns have changed, but a number of other contributing factors such as crop prices have also changed. This counterfactual simulation disentangles the separate effect of energy prices from all other

factors, which are fixed at their observed values. We conducted simulations of two scenarios. In the *historical* scenario the linked model was configured to reproduce observations over the simulation period (calibration described in Bulatewicz et al. (2010)). In the *high energy price* scenario the exogenous energy price input to the economic model was increased by 15% per well for each year. The energy price (average of 2.32 $/mcf over study period) is the natural gas futures price (normalized by index of prices paid) as the February monthly average of June and July contracts in the United States (source: Futures – CRB PowerGen).

Using the Configuration Editor application (OpenMI Association Technical Committee) we interactively created a new composition and added the agricultural component and two instances of the SSW component along with the necessary input and output links. We attached the trigger to the groundwater model because its outputs for a given year are used by the other models for the simulation of the following year and attaching the trigger to the agricultural or economic model would result in the groundwater model not simulating the final year of the simulation period. When we executed the linked model the dependencies were automatically identified and the order in which the components advanced their simulation times and exchanged data was fully automated.

The execution of the linked model begins with the trigger calling GetValues on the groundwater model requesting the saturated thickness for 1991. The groundwater model requires the water-use in 1991 in order to compute the next time step so it calls GetValues

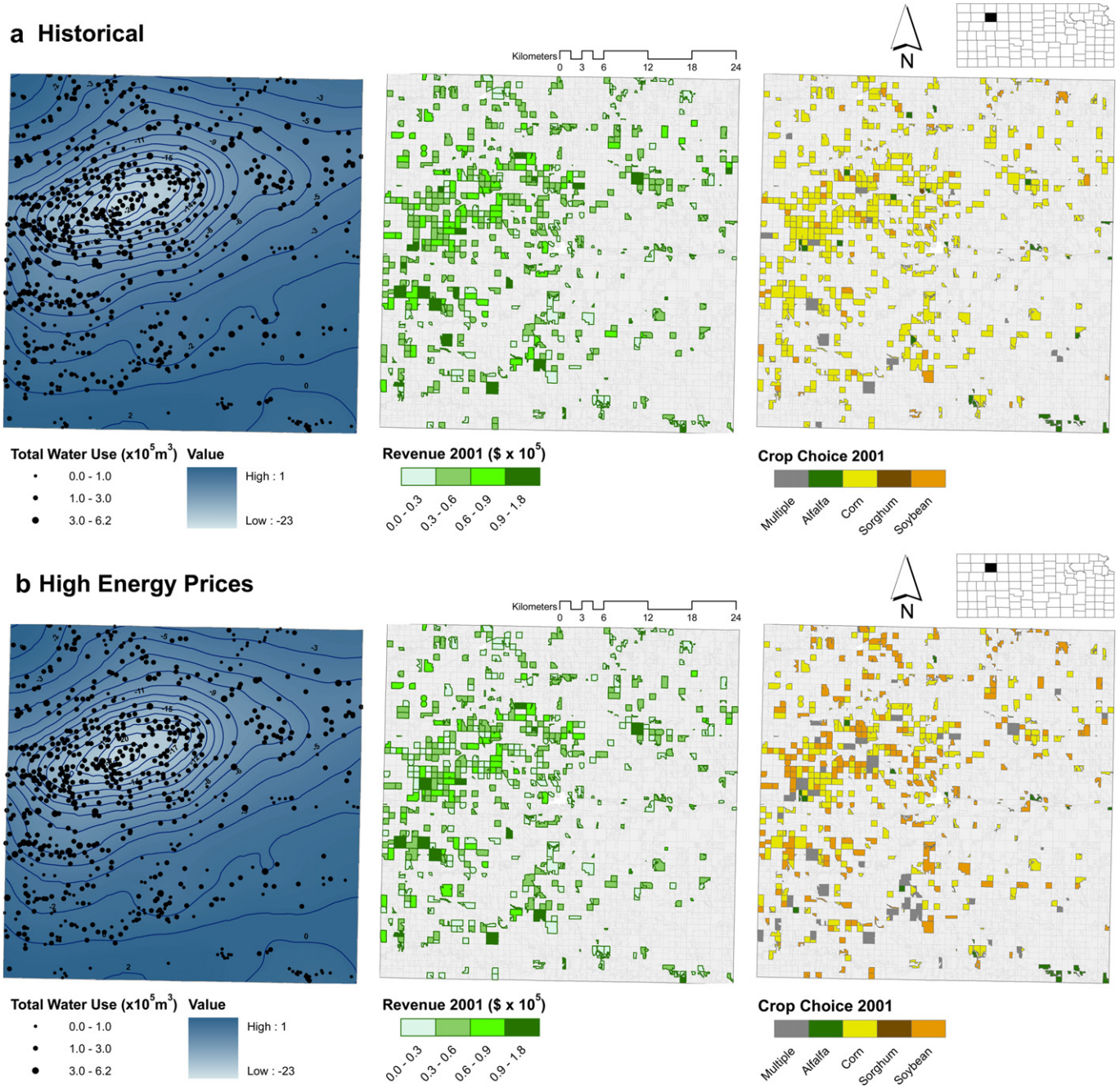

Fig. 9. Results of the historical and high energy price scenarios. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

on the agricultural model requesting it. In response to this request the agricultural model calls GetValues on the economic model to obtain the crop choice in 1991 which it needs before it can simulate the crop growth and water-use that year. The economic model in turn calls GetValues on the groundwater model requesting the saturated thickness at the end of 1990 which is necessary to predict the crop choice in 1991. The groundwater model returns the saturated thickness for 1990 (observed values are used for the first year) to the economic model which uses the values to predict the crop choice in 1991 and then provides those results to the agricultural model. The agricultural model simulates the crop growth and water-use of the chosen crops and provides the water-use values to the groundwater model which uses the values to simulate the saturated thickness in 1991. The trigger then calls GetValues on the groundwater model for the next year (1992) and the chain of requests repeats.

The results of each model for the historical and high energy price scenarios are illustrated in Fig. 9 and briefly interpreted to explain the SSW results for this interdisciplinary problem. The high energy prices resulted in the economic model predicting less planting of the water intensive crops, corn and alfalfa, and more planting of soybean and sorghum which have lower water requirements. Over the 14-year period the acreage planted decreased for corn (−4.92%) and alfalfa (−0.45%) and increased for soybean (+5.34%) and sorghum (+0.04%). There was a corresponding decrease in total production (across all crops) by 4.31% (3.90×10^6 t−3.73×10^6 t), a decrease in revenue by 1.39% ($\$3.23 \times 10^8$–$\$3.19 \times 10^8$), and a decrease in water-use by 1.26% (1.54×10^9 m^3–1.52×10^9 m^3).

The impact of the change in energy prices on the crop choice varied by year according to the magnitude of the increase. In years where the energy prices were low, the 15% increase had a smaller magnitude and smaller effect on the crop choice. Conversely, years with higher energy prices resulted in a larger magnitude and larger effect on crop choice.

The smallest change occurred in 1996 in which the difference in acreage planted for each crop was less than 0.60%. The greatest impact of the change in energy prices on the crop choice was in year 2001 in which the share of acreage planted to alfalfa, corn, and sorghum decreased by 0.96%, 34.27%, and 0.24% and soybean production increased by 35.48%. The spatial distribution of this change in crop choice for year 2001 is shown in Fig. 9 along with the corresponding change in revenue. Production in this year decreased by 24.66% with corresponding decreases in revenue by 11.35% and water-use by 14.74%. There was a similar impact for the years 2003 and 2004 and those three years accounted for much of the change over the simulation period.

Our results accord with a typical economic property of input demand functions—quantities are relatively more sensitive to price changes at high prices than at low prices. At high energy prices, we find an implied elasticity of water use with respect to the energy price of $6.9/15 = 0.46$, which is within the range of estimates from other irrigated regions (Scheierling et al., 2006). In periods when energy prices are high, the additional energy costs will dampen farmers' incentives to plant water intensive crops somewhat, although the response is still relatively small in elasticity terms and may be outweighed by other factors such as increased commodity prices.

5. Conclusions

In this work we have presented the design of a software component called the Simple Script Wrapper (SSW) that enables scientists to link models and tools written in scripting languages to OpenMI components. The SSW provides the interoperability between scripting languages and the C# language used by the OATC SDK allowing the scientist to work exclusively within a scripting language. It capitalizes on the dynamic nature of scripting languages making it general-purpose and usable to link any scripts written in the supported languages. The SSW thus enables modeling tools developed with scripting languages to become OpenMI-compliant. This makes model linking to scripting languages accessible to both scientists and students alike and has been employed in the classroom as an activity in multidisciplinary model linking.

The process of making a scripted model into an OpenMI linkable component involves two steps in which the scientist (1) writes a file that describes the configuration of a model such as its name, inputs, and outputs and (2) adds three functions to the model script for initialization, performing a time step, and finishing its simulation. The SSW collects inputs from the other components as necessary and makes them available to the model script through a function call. It instructs the model when to perform time steps in response to requests from other components. The model script calls functions to save the results of its time steps and the SSW relays these results to other components as necessary. The current implementation can be used with any scripts written in MATLAB, Scilab, or Python and can be extended to support additional scripting languages.

We conducted a case study in the context of irrigated agricultural systems that demonstrates how the SSW can be utilized for interdisciplinary model integration. The integrated model was composed of agricultural, groundwater, and economic models and the latter two were scripted models that utilized the SSW. Modeled quantities were exchanged between the models in a circular fashion (Fig. 8). We utilized the integrated model to carry out the simulation of two scenarios to investigate the impact of changes in exogenous energy prices on an irrigated agricultural system. The results of the integrated model for the historical scenario between 1991 and 2004 are shown in Fig. 9. The results of the retrospective scenario in which energy prices were 15% higher throughout the study period are shown in Fig. 9. In both cases the results are presented spatially in terms of groundwater elevation, crop choice, and economic productivity. The results illustrate the spatial patterns of changes in water-use, production, and revenue that result from an increase in energy prices.

Multidisciplinary integrated models are uniquely suited to contribute to the understanding of complex coupled human-natural systems. The technical challenges of integrating disparate models can be prohibitive and results in isolated modeling studies by scientists and minimal exposure to quantitative integrated assessment techniques for students. The development of accessible methods for model integration, such as the SSW, are necessary for the practice of integrated modeling to be broadly adopted in the modeling and simulation community. Our SSW implementation provides a foundation for those who write scripting language tools to make their models OpenMI-compliant, and thus integrable with other OpenMI codes. It is through such practices that large-scale natural resources can be managed in an environmentally sound, economically viable, and socially acceptable way.

Acknowledgments

This work was supported in part by the National Science Foundation (grant GEO0909515) and the United States Department of Agriculture/Agricultural Research Service (Ogallala Aquifer Initiative). Any findings, opinions, conclusions, or recommendations expressed herein are those of the authors and do not necessarily reflect the views of any funding units.

References

Buis, S., Piacentini, A., Declat, D., 2006. PALM: a computational framework for assembling high-performance computing applications. Concurr. Comp. -Pract. E 18 (2), 231–245.

Bulatewicz, T., Cuny, J., 2006. A domain-specific language for model coupling. In: Proceedings of the 2006 Winter Simulation Conference, Monterey, California, 3–6 December, pp. 1091–1100.

Bulatewicz, T., Jin, W., Staggenborg, S., Lauwo, S.Y., Miller, M., Das, S., Andresen, D., Peterson, J., Steward, D.R., Welch, S.M., 2009. Calibration of a crop model to irrigated water use using a genetic algorithm. Hydrol. Earth Syst. Sci. 13, 1467–1483.

Bulatewicz, T., Yang, X., Peterson, J.M., Staggenborg, S., 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 (3), 521–534.

Castronova, A.M., Goodall, J.L., 2010. A generic approach for developing process-level hydrologic modeling components. Environ. Model. Softw. 25, 819–825.

CUASHI, 2012. CUASHI website. http://www.cuashi.org.

Damgaard, C.F., 2004. Coupling between the river basin management model (mikebasin) and the 3d hydrological model (mike she) with use of the OpenMI system. In: 6th International Conference on Hydroinformatics, Singapore, June 2004, pp. 21–26.

David, O., Markstrom, S.L., Rojas, K.W., Ahuja, L.R., Schneider, I.W., 2002. The Object Modeling System. In: Ahuja, L., Ma, L., Howell, T.A. (Eds.), Agricultural System Models in Field Research and Technology Transfer. Lewis Publishers, CRC Press LLC, pp. 317–337.

Famiglietti, J.S., Murdoch, L.C., Lakshmi, V., Arrigo, J., Hooper, R., March 15–17, 2011. Establishing a framework for community modeling in hydrologic science, In: Report from the 3rd Workshop on a Community Hydrologic Modeling Platform (CHyMP): a Strategic and Implementation Plan, Beckman Center of the National Academies, University of California at Irvine.

Ford, R.W., Riley, G.D., Bane, M.K., Armstrong, C.W., Freeman, T.L., 2006. GCF: a general coupling framework. Concurr. Comp. -Pract. E 18 (2), 163–181.

Foster, S., Loucks, D.P., 2006. Non-renewable Groundwater Resources. In: Series on Groundwater No. 10. UNESCO, Paris.

Gijsbers, P., Brinkman, R., Gregersen, J., Hummel, S., Westen, S., 2005. OpenMI: Open Modeling Interface. The OpenMI Document Series: Part C The org.OpenMI. Standard Interface Specification (version 1.0). Available at: http://www.OpenMI.org.

Gregersen, J.B., Gijsbers, P.J.A., Westen, S.J.P., 2007. OpenMI: Open Modeling Interface. J. Hydroinform 9 (3), 175–191.

Harris, G., 2002. Integrated assessment and modelling: an essential way of doing science. Environ. Model. Softw. 17 (3), 201–207.

Hendricks, N.P., 2007. Estimating Irrigation Water Demand with a Multinomial Logit Selectivity Model. Department of Agricultural Economics, Kansas State University, Manhattan, Kansas.

IronPython. IronPython Community. http://www.ironpython.net.

Jakeman, A.J., Letcher, R.A., 2003. Integrated assessment and modelling: features, principles and examples for catchment management. Environ. Model. Softw 18 (6), 491–501.

Joppich, W., Kurschner, M., 2006. MpCCI – a tool for the simulation of coupled applications. Concurr. Comp. -Pract. E 18 (2), 183–192.

Kendrick, D.A., Amman, H.M., 1999. Programming languages in economics. Comput. Econ. 14, 151–181.

Konikow, L.F., Kendy, E., 2005. Groundwater depletion: a global problem. Hydrogeol. J. 13, 317–320.

Larson, J.W., Jacob, R.L., Foster, I.T., Guo, J., 2001. The model coupling toolkit. In: Lecture Notes in Computer Science. Proceedings of the International Conference on Computational Sciences-Part I, vol. 2073. Springer-Verlag, pp. 185–194.

Mathematica. Wolfram. http://www.wolfram.com.

MATLAB. Mathworks. http://www.mathworks.com.

Moore, R.V., Tindall, C.I., 2005. An overview of the Open Modelling Interface and environment (the OpenMI). Environ. Sci. Policy 8 (3), 279–286.

OpenMI Association Technical Committee. Configuration editor software. http://sourceforge.net/projects/openmi.

Ousterhout, J., March, 23-30 1998. Scripting: higher level programming for the 21st century. IEEE Comput.

Pahl-Wostl, C., 2002. Participative and stakeholder-based policy design, evaluation and modeling processes. Integr. Assess. 3 (1), 3–14.

Parker, P., Letcher, R., Jakeman, A., Beck, M.B., Harris, G., Argent, R.M., Hare, M., Pahl-Wostl, C., Voinov, A., Janssen, M., Sullivan, P., Scoccimarro, M., Friend, A., Sonnenshein, M., Barker, D., Matejicek, L., Odulaja, D., Deadman, P., Lim, K., Larocque, G., Tarikhi, P., Fletcher, C., Put, A., Maxwell, T., Charles, A., Breeze, H., Nakatani, N., Mudgal, S., Naito, W., Osidele, O., Eriksson, I., Kautsky, U., Kautsky, E., Naeslund, B., Kumblad, L., Park, R., Maltagliati, S., Girardin, P., Rizzoli, A., Mauriello, D., Hoch, R., Pelletier, D., Reilly, J., Olafsdottir, R., Bin, S., 2002. Progress in integrated assessment and modelling. Environ. Model. Softw. 17 (3), 209–217.

Parson, E.A., 1995. Integrated assessment and environmental policy making: in pursuit of usefulness. Energ. Policy 23 (4–5), 463–475.

Peckham, S.D., Hutton, E., 2009. Componentizing, standardizing and visualizing: how CSDMS is building a new system for integrated modeling from open-source tools and standards. Eos Trans. AGU 90 (52). Fall Meeting Abstracts Suppl., abstract IN11A–1045.

Perl. http://www.perl.org

Python. Python Software foundation. http://www.python.org.

Rodell, M., Velicogna, I., Famiglietti, J.S., 2009. Satellite-based estimates of groundwater depletion in India. Nature 460, 999–1002.

Rotmans, J., Van Asselt, M., 1996. Integrated assessment: a growing child on its way to maturity. Clim. Chang. 34 (3–4), 327–336.

Ruby. http://www.ruby-lang.org.

SAS. SAS Institute Inc. http://www.sas.com/.

Scanlon, B.R., Jolly, I., Sophocleous, M., Zhang, L., 2007. Global impacts of conversions from natural to agricultural ecosystems on water resources: quantity versus quality. Water Resour. Res. 43, W03437.

Scheierling, S.M., Loomis, J.B., Young, R.A., 2006. Irrigation water demand: a meta-analysis of price elasticities. Water Resour. Res. 42, W01411.

Scilab. Inria. http://www.scilab.org

Shah, T., Burke, J., Villholth, K., 2007. Groundwater: a Global Assessment of Scale and Significance. Earthscan, London, UK and IWMI, Colombo, Sri Lanka. Molden, D.

SPSS. IBM Corporation. http://www.sas.com/.

Steward, D.R., 2007. Groundwater response to changing water-use practices in sloping aquifers. Water Resour. Res. 43 (W05408), 1–12.

Steward, D.R., Peterson, J.M., Yang, X., Bulatewicz, T., Herrera-Rodriguez, M., Mao, D., Hendricks, N., 2009. Groundwater economics: an object oriented foundation for integrated studies of irrigated agricultural systems. Water Resour. Res. 45, W05430.

Wang, J., Huang, J., Rozelle, S., Huang, Q., Zhang, L., 2009. Understanding the water crisis in northern China: what the government and farmers are doing. Int. J. Water Resour. D. 25 (1), 141–158.

Williams, J.R., 1995. The EPIC model. Comput. Models Watershed Hydrol., 909–1000. Chapter 25.

Williams, J.R., Dyke, P.T., Fuchs, W.W., Benson, V.M., Rice, O.W., Taylor, E.D., 1990. EPIC – Erosion/productivity Impact Calculator: 2 User Manual. USDA. Technical Bulletin No. 1768.