2006-05-11

A Study of Model Integration in Conjunction with the eXtensible Model Definition Format

Melanie A. Fife
Brigham Young University - Provo

A STUDY OF MODEL INTEGRATION IN CONJUNCTION WITH THE EXTENSIBLE MODEL DEFINITION FORMAT

by

Melanie Ann Fife

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Civil and Environmental Engineering

Brigham Young University

August 2006

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Melanie A. Fife

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Norman L. Jones, Chair
Date	E. James Nelson
Date	Alan K. Zundel
Date	Michael J. Swenson

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Melanie A. Fife in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Norman L. Jones Chair, Graduate Committee

Accepted for the Department

E. James Nelson
Graduate Coordinator

Accepted for the College

Alan R. Parkinson
Dean, Ira A. Fulton College of Engineering and Technology

ABSTRACT

A STUDY OF MODEL INTEGRATION IN CONJUNCTION WITH THE EXTENSIBLE MODEL DEFINITION FORMAT

Melanie Ann Fife

Department of Civil and Environmental Engineering

Master of Science

A considerable amount of research has been done to connect or integrate separate numerical models. The Environmental Modeling Research Laboratory (EMRL) has developed a generic file format, the eXtensible Model Data Format (XMDF). One of the objectives of the XMDF project is to facilitate the spatial interpolation and data-sharing necessary when linking models. The objective of this research is to investigate how model linking capabilities can be added to the XMDF by defining a Model Linkage Object (MLO) that is compatible with current model linking frameworks.

A potential design for an MLO is defined in this thesis after a detailed review of existing model linking frameworks. The major model-linking systems studied in this research are FRAMES, DIAS, and OpenMI. Other software was also researched and a brief description of each is provided. To limit the scope of this research the MLO is

analyzed for a two-dimensional mesh using only two of the model linking systems. FRAMES was chosen because the Army Corps of Engineers has already done extensive work with this software. OpenMI was selected because of its ability to link models during runtime.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Norm Jones, for his guidance and direction in my research, for all the time he took to answer my questions, and for the countless hours he spent reviewing my work.

I am grateful to Rob Wallace at the Army Corps of Engineers for his efforts in funding the research. Without their support this research would have never taken place. The feedback that he provided was also invaluable.

Finally, I appreciate the willingness of my family to look over initial drafts of my thesis even though they didn't understand it. A special thanks to my dad for actually reading each chapter and offering his comments.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

As the capabilities of computers have increased and the cost of hardware has decreased, the ability to simulate complex water modeling systems has improved. These simple models typically analyze only a portion of the water system, such as the river or the groundwater. In many cases, a model of the entire water cycle, or at least several components of the water cycle is needed. Since the development of the simple models represents a substantial investment of both time and money, it is not desirable to recreate these same models in order to build a more complex system. Therefore, a considerable amount of research is being done to develop frameworks that can link models together so that existing models can continue to be used in combination with other models to simulate more complex systems.

1.1 XMDF File Format

The U.S. Army Corps of Engineer's Research and Development Center (ERDC) has funded the Environmental Modeling Research Laboratory (EMRL) at Brigham Young University to develop a standard file format, the eXtensible Model Data Format (XMDF), as a preliminary step to facilitating model linkage. XMDF stores the most commonly used computational modeling geometry, namely:

- River cross-sections

- Two- and three-dimensional structured grids
- Two- and three-dimensional unstructured meshes
- Geometric paths through space with associated time data

Data sets associated with these objects are also stored. Data sets represent transient or steady-state vector and scalar arrays connected to the nodes of the geometric objects. Using a standard data format simplifies the model linkage process by allowing temporal and spatial interpolations to take place inside the Application Programming Interface (API).

The goal of this research is to review the current integrated modeling frameworks to develop a preliminary design for adding model linking capabilities to XMDF that will be compatible with existing technology. A Model Linkage Object (MLO) will be defined for XMDF based on the most applicable integration systems.

1.2 Model Linking Systems

In reviewing existing model linking systems there are several issues that must be addressed. Integrated models share data during the simulation process. Because modeling software uses file formats and data requirements specific to their individual needs, there is a vast array of models in incompatible forms. The model-linking systems must be able to reconcile these discrepancies so the models can exchange data.

Additionally, a model must often be modified to be included in the integrated system. Because of the time and effort already spent in building the models, it is desirable to require as few adjustments as possible to already existing models. A common way to modify existing models is by wrapping the model with a pre- and post-processor. The pre-processor modifies input to the model so that it matches the model's format, while the post-processor will change the model's output to conform to the linked system. The advantage to using a model wrapper is that the original model code remains unchanged.

2 FRAMES

The Framework for Risk Analysis in Multimedia Environmental Systems (FRAMES) is developed by the Pacific Northwest National Laboratory. It has been used in applications for the U.S. Environmental Protection Agency (EPA), U.S. Army Corps of Engineers, U.S. Nuclear Regulatory Commission, U.S. Department of Energy (DOE), American Chemistry Council, and other federal and state agencies along with private organizations.

2.1 Background

The development of FRAMES began as the EPA and the DOE recognized a common focus towards environmental waste and clean up efforts. Both agencies use models as tools to aid them in assessing environmental needs. The benefit to being able to access each other's models was recognized, giving rise to the development of a framework where models with differing attributes could communicate with each other.

The objective was to use a flexible framework that could meet the needs of both organizations to combine existing models that assess environmental, safety, and health risks. FRAMES has been created to integrate diverse modeling components that support DOE and EPA regulatory problems. The flexible nature of the framework is such that other agencies can use it to choose the most appropriate codes to solve their environmental simulation requirements. This will document the current state of FRAMES as well as direction of future development.

2.2 Definitions

FRAMES makes a distinction between models and modules. A model is simply defined as the code or mathematical representation of the desired process. In contrast, a module is made up of the model description, a module user interface, and the execution of the model. The model description contains the information on the purpose of the model and how it fits into the system. The user interface represents the model input and the execution is the model run.

The importance of this distinction is that the module concept keeps the user interface, input files, and executable separate from each other. This means that while there is no difference to the user in the interface, the input may come from the user, from a database, or from another model. Also, by separating the input from the executable, sensitivity analysis on the input data is possible.

This distinction will be maintained when referring to FRAMES models and modules.

2.3 Data Exchange

Models created from different agencies that reflect multiple scientific disciplines can vary widely in format and data requirements. It is vital to understand how FRAMES exchanges information so that it is compatible and correctly used.

2.3.1 File Transfer

FRAMES exchanges data through input and output files. Modules are linked in a process flow diagram and implemented one at a time. This ensures that the output file of one module will be used as the input for the next module. Input and output are kept in separate files so the sensitivity/uncertainty analysis can run efficiently.

Because data exchange is handled through files, the format is critical. All specifications for the simulation are located in the Primary Data Communication Files (PCDF). These files keep the data associated with the linking separate. Three of the standard PCDF files are required so that FRAMES recognizes the data. The files that must be included are:

- Error (ERR) file
- Global Input Data (GID) file
- Description (DES) file

The ERR file ensures incorrect results aren't mistaken for valid results. It is created when a simulation begins and is deleted when the program finishes successfully. Therefore the presence of the ERR file indicates that the completed process contains errors.

The GID file contains the connection and user input data for the module. Because the user can change this type of data at will, it is organized to allow changes to be made easily and efficiently.

The DES file describes the module so it can be used properly. It contains a description that can be seen by users as they select a module and includes information such as what the module is usually used for, limiting assumptions, time scale, references to formulation, verification and validation documents (if they exist), hardware requirements, and contact information.

Other PCDF files are specific to the type of environmental assessment that is being simulated. They only need to be included for specific assessments. These files include the following:

- Air Flux File (AFF)
- Atmospheric Transport Output File (ATO)
- Body Burden File (BBF)
- Exposure Pathways File (EPF)
- Ecological Effects File (EXF)
- Health Impacts File (HIF)
- Receptor Intake File (RIF)
- Soil Concentration File (SCF)
- Sensitivity/Uncertainty File (SUF)
- Water Concentration File (WCF)
- Water Flux File (WFF)

Other files that can be used include imported files that contain information that models need (such as a spreadsheet containing lab results), exported data files with results of previous tests, and databases that are maintained by other individuals.

2.3.2 System Requirements

FRAMES software will operate on IBM compatible PCs. The FRAMES user interface and all other user defined source modules operate on Windows95 or higher.

2.4 Setting up the Simulation

FRAMES implements drag-and-drop capabilities to prepare and visualize the simulation in a straightforward and simple manner. Simulations are created by dragging modules from a selection tree into the workspace. Modules in the workspace can be moved, deleted, connected, and disconnected with the mouse. Figure 2-1 shows how the user interface is organized.

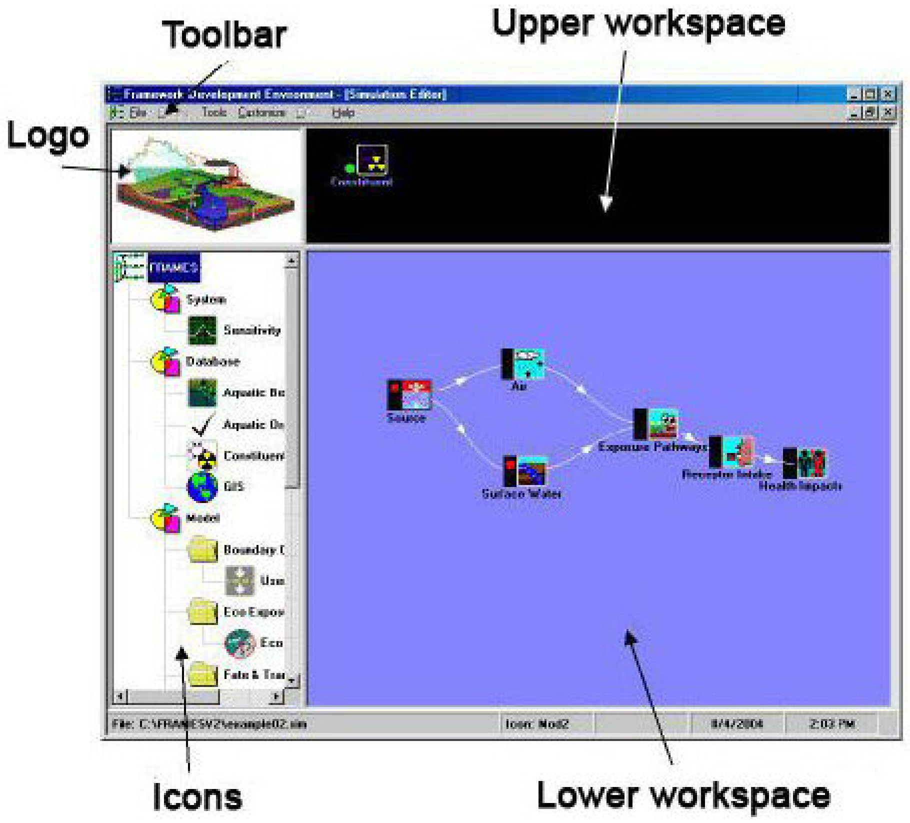


Figure 2-1 FRAMES user interface [source: adopted from the FRAMES website]

Two workspaces are available for the user's convenience, as shown in Figure 2-1. Modules are linked in the lower workspace to form simulations, while an upper workspace is provided for data that must be available for multiple modules. For example, a database of chemicals might need to be supplied to each module. Instead of creating individual connections to each module it links with, it can be placed in the upper workspace where its database will be made available to each module in the lower workspace.

Modules that are added to the workspace are given a status light to indicate the completion status of the module. No light means that no module has been chosen. A red light means the module has been chosen. A yellow light indicates the input data are complete, while a green light means the code calculations have been successfully run. Results can be viewed in text or graph format from modules with green status lights.

Each module also contains its own user interface. The user interface for a specific module can be brought up by right-clicking on the module in the workspace. Data can then be input to the module.

Once all the data have been input for a module it can be run. Each module can be run individually or the entire sequence can be run, as long as the input for each module has been supplied. This is convenient for re-running simulations after making changes to module inputs. It is not possible to do cyclic or concurrent processes in FRAMES since each new module depends on the results of the proceeding one.

2.4.1 The Framework User Interface

The Framework User Interface (FUI) helps the user decide what modules to use, controls the information flowing between modules, and keeps the input and output data separate. Specifically it ensures:

- Modules are connected properly

- The module has unlimited access to data
- Multiple forms of data entry are allowed
- Rules for running components are established (a module won't run until the input requirements are complete or data required from other modules are accessible)
- If one module behaves incorrectly it will not damage data in other modules and the user will be informed of the error
- User data and results are stored correctly
- Informing the user which modules are available in FRAMES

2.5 Including a Model

The method of incorporating a model in FRAMES is different for different versions of FRAMES. Both methods are described here.

2.5.1 FRAMES 1.x

A FRAMES module must be able to get data from the user through a user interface, run the model, and describe itself to FRAMES. These three tasks correspond to the three steps necessary to make a model compatible with the FRAMES software.

1. Create a new user interface or wrap an existing one.
2. Create a new model or wrap an existing model or visualization program.
3. Create a description file (DES) file so that FRAMES will recognize the module.

Wrapping a model means that a pre- and post-processor is included with the model execution. The pre-processor modifies input to the model so that it matches the model's format, while the post-processor will change the model's output to conform to the linked system. The advantage to defining a model wrapper is that the model code remains unchanged.

Before this process can begin the specifications that apply to the model must be identified. These specifications correspond to file formats and have been identified previously in the data exchange section. The module does not need to match FRAMES specifications directly, but a wrapping program must be able to convert the output into FRAMES specifications. The ERR, GID, and DES files are necessary because all modules in FRAMES are required to interact with these file types.

2.5.2 FRAMES 2.x

In the newer release of FRAMES that is under development, simplifications have been made so that model developers do not need to match a file specification. The old approach increased code maintenance and made correct file specification critical. In FRAMES 2.0 the file format is separated from the data being stored by a series of functions in an API. Transforming a model into a FRAMES module now consists of a five step process.

2.5.2.1 Units and Measures

The first step is to describe the model's units and measures in FRAMES. A measure describes the type of measurement that must be made, such as concentration or flow rate. Each measure is bound to the units that can be used to describe the measurement. For example, the measure "concentration" is bound to the unit "μg/g". All units that are bound to a measure are convertible between other units in that measure.

This step involves cataloging all measures and associated units that are used as inputs or outputs by the model. If any of these units and measures is not included in FRAMES it can be added through the Conversion Editor.

2.5.2.2 Model Inputs and Boundary Conditions

The second step is to define the inputs the model will consume and the outputs the model will produce. This is done by defining dictionaries in FRAMES. The term dictionary represents metadata and can be considered the declaration of a class. A dataset refers to an instance of the class. All data that flows in and out of FRAMES modules goes through the dictionary-dataset construct.

There are several types of dictionaries that can be created. A user provides input for the Module Input Dictionary, while datasets that are produced or consumed as part of the model's calculations are defined in Boundary Conditions Dictionaries. These dictionaries can either be System Developer Dictionaries which are available to all modules or Module Developer Dictionaries which are specific to a module or system of modules.

Dictionaries are created using the Dictionary Editor which creates a DIC file so that FRAMES will recognize it. Variables with defined properties must be added to the dictionary. Properties include the following:

- A brief description of the parameter
- The data type (float, string, integer, or boolean)
- The number of dimensions associated with the parameter
- The minimum and maximum values allowed

- The measure and the unit
- Whether the parameter can be modified by sensitivity and uncertainty tools
- A list of other variables in the system that help define the parameter
- A flag that indicates whether the parameter is scalar or vector

2.5.2.3 The Model Call

The third step is to prepare the model so that it can be called by the FRAMES API. In the old method for including a model in FRAMES each model needed to provide a user interface. This is no longer necessary since FRAMES 2.0 provides a General Module User Interface through the Data Client Editor. Because the rules for this interface are very general, a model may want to provide its own user interface when interdependencies between parameters exist.

There are two options for calling the model. One option is to pass three arguments directly to the OpenIO function which connects the model to the FRAMES 2.0 system. These arguments tell the module which invocation the module should process, what simulations files are related to the invocation, and where the files are located. This function will return a program id (pid) which indicates whether or not the system was able to register the model.

The other option is to use a model wrapper so that the testing and use of the model is undisturbed. The FRAMES API consists of the System Developer, DataSet, Error, and Conversion subcomponents. The model's pre-processor must be able to access the System Developer and DataSet functions. It can then pass the three arguments to the OpenIO function. Just as before, a pid is returned to the model. When the model is finished reading the input values and writing them to its native file format it calls CloseIO, signaling to the FRAMES API that the process is complete. OpenIO is called again because other API functions require the model's pid. The post-processor allows the model to produce a dataset for more than one output module.

2.5.2.4 Model Description

The fourth step is to provide a description of the model so that modules will be connected accurately. Typical model information includes the name of the model developer, the amount of memory needed, and basic information about the model. However, because this is not complete enough for FRAMES to determine if the module can provide the input needed by another module, relationships must be explicitly described in FRAMES' Connection Schemes. This is where the module records the number and type of datasets it can consume and produce. It also defines what dictionary is used for user input.

2.5.2.5 Categorizing the model

The final step is to categorize the module so the modules are displayed in an organized manner. This is done through the Domain Editor. Classifying modules into domains allows the user to conceptualize their modeling. Domains are subdivided into groups and sub-groups. For example, a group titled "Water Models" might contain the subgroups "River", "Groundwater", and "Vadose Zone".

2.6 Special Considerations

FRAMES allows models with different scales to interact with each other. Databases can also be linked in the framework.

2.6.1 Scale and Resolution

Although global analyses will need to run from different frameworks than site-specific analyses, they should still be able to interact with each other. In FRAMES the results of a site-specific analysis can be used as a boundary condition for a regional analysis. Four basic scales have been defined to help keep these differences separate: medium-specific (site specific), watershed, regional, and global.

Similarly, analyses can also be divided up by resolution, depending on the level of detail required for an assessment to be made. Resolutions can be low, medium, or high and are dependent on the boundary conditions, not the model. Different approaches can be categorized by scale and resolutions to help develop an appropriate framework.

2.6.2 Databases

Differences in database formats can cause difficulty in accessing, but common databases to use in FRAMES include population census data, NOAA climatological database, USGS gaging station database, soils databases, hydrogeologic databases, toxicological databases, chemical databases, exposure parameter databases, and waste management unit databases.

2.7 Application – ARAMS

The Army Risk Assessment Modeling System (ARAMS) is developed by the US Army Engineer Research and Development Center in Vicksburg, Mississippi to determine safe levels for military-relevant compounds and to evaluate remediation alternatives.

ARAMS provides links to web-based databases, stand-alone models, and relevant websites. These are easily accessible through the main toolbar illustrated in Figure 2-2. A tool is also provided to launch FRAMES. ARAMS uses FRAMES to visually specify the links between models and databases.

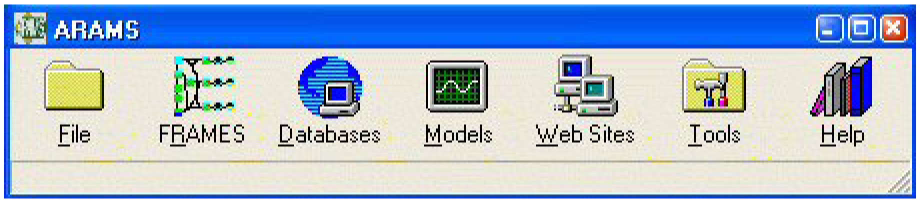


Figure 2-2 ARAMS toolbar [source: adopted from the ARAMS website]

2.7.1 Setting up the Simulation

To set up a linked model, the DES file must be created and exported in ARAMS. The FRAMES tool is then launched and a global input data (GID) file is created in the conceptual site model. A simulation can be set up in the same manner a simulation would be set up in FRAMES, by dragging modules onto the screen. These modules can be edited with the mouse and given status lights. Results can be viewed by right clicking on a module and selecting a visualization option from a list.

2.7.2 System Requirements

ARAMS is compatible with Windows 2000 and Windows XP.

3 DIAS

The Dynamic Information Architecture System (DIAS) is developed by the Argonne National Laboratory's Decision and Information Sciences Division. Its development has benefited from other agencies and organizations as well, including the USA Topographic Engineering Center, USA Waterways Experiment Station, and the USAF Phillips Laboratory Geophysics Directorate.

3.1 Background

DIAS is a framework where models, information processing applications, and databases can be placed to interact with each other. This project was designed to address the complexity and dynamics of "real world" systems. It has a flexible domain which is determined by the models and data-processing applications gathered by the users to address specific concerns.

The first well-developed application to use the DIAS concept was the Dynamic Environmental Effects Model (DEEM) in 1993. DEEM allows integration of environmental effects models and has been used as a framework for hydrologic modeling by the U.S. Army Corps of Engineers. Other large applications of DIAS include a healthcare management simulator (KARE*PLAN) and a logistics application (DIAL).

3.2 Definitions

There are several object classes that are used in DIAS to facilitate its flexible design. These objects are summarized in Table 3-1. A more detailed explanation of how the objects interact with each other will be explained in subsequent sections.

Table 3-1: DIAS object classes

Analysis Frame	Defines the conceptual bounds of the simulation
Domain Object	Environmental component that interacts in the simulation
Spatial Data Set Object	Holds the geometric specification for a domain object
Parameter Object	Contains the state properties of a domain object
Aspect Object	Represents a single behavior of a domain object
Process Object	Addresses a particular domain object's behavior
Context Object	Links aspect objects with process objects and identifies the dependencies between them
Model Object	Represents a DIAS model with references to all necessary process objects
Model Controller	Provides linkage to the necessary source codes and data structures that are not part of the DIAS framework
Event Object	Signals a domain object to perform a stated aspect of its behavior

3.3 Data Exchange

In order to make DIAS a flexible system, the models never communicate directly with each other, as Figure 3-1 shows. They are connected through a series of domain objects which are abstract descriptions of an object's behavior. This keeps adding, deleting, and swapping models in a simulation uncomplicated because no recoding must be performed.

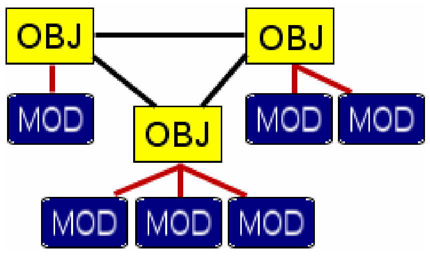


Figure 3-1 DIAS model separation
[source: adopted from the DIAS website]

3.3.1 Object-Model Interactions

In a DIAS simulation the data that is exchanged is kept separate from the models. An integrated model system consists of two basic components, the domain objects and the models. Domain objects represent real world entities, such as a river or an aquifer, and contain abstract descriptions of the behaviors of that entity. A behavior is a process that the domain object is capable of performing. Behaviors of a river domain object might include sediment transport or flood plain delineation. The specific applications of those behaviors are contained in the models.

3.3.1.1 Domain Objects

Domain objects are made up of parameter and aspect objects and may contain a spatial data set object. The spatial data set object specifies the geometric specification (grid, mesh, etc.) of the domain object and allows spatial dependencies to be defined between domain objects. The parameter objects contain the properties of the domain object. For a river domain object this could include parameters such as stage and flow rate. The behaviors of the domain object are described in the aspect objects. Each aspect object depicts a single behavior of the domain object. This means that for every process that needs to be simulated by a model, an aspect object must be defined in the domain object.

Because domain objects are general representations of real world objects, they can be saved and modified for use in different simulations. An object library is provided to store domain objects. When new models are used in DIAS, new domain objects may need to be added to the library or existing objects may need to be edited. This allows the object library to expand.

3.3.1.2 Models and Applications

The DIAS framework is able to access external models by defining a model object and a model controller. The model controller provides the linkage to the model codes and data structures. The model object represents the model inside of the DIAS analysis frame. It contains the physical location of the model and references to all process objects used by the model.

3.3.1.3 Process Objects

The process objects link the models to the domain objects by connecting the model process that needs to be performed with the corresponding aspect object in the domain object. Each process object represents a single aspect object. While model objects directly specify which process objects it will use, the aspect objects are dynamically linked to process objects at run time. The process objects are the only objects that contain information about both domain objects and implementation. They (un)package data that are passed between the models, perform (dis)aggregation of data, transform coordinate systems, and make unit conversions.

3.3.1.4 Context Manager

A Context Manager is provided to make sure that process objects are linked to the most appropriate domain object according to constraints supplied by the user. The user can specify a context with any level of detail, including specifically assigning process objects to aspect objects. The Context Manager will create dynamic links between aspect objects and model processes or it will inform the user if the goals are unattainable.

3.3.2 System Requirements

DIAS can be run on UNIX and PCs using Windows NT. Models in DIAS can be distributed across a network of processors or can be run on a parallel distributed memory computer.

While the core components are written with Smalltalk along with C and C++, the model and applications integrated into DIAS are included in the programming language they were developed in.

3.4 Setting up the Simulation

An integrated model is created by implementing a conceptual frame containing all the domain objects that will be needed in the simulation. Models and applications can then be chosen by the user or the Context Manager can automatically select models, applications, and databases that are already integrated into DIAS.

The Discrete Event Simulation Manager processes events in a time-ordered sequence. Events are created by user interaction or by a domain object's behavior. Each event has a time stamp associated with it so it will be implemented in the proper order.

When a domain object receives an event it will route it to the appropriate aspect object. The aspect object responds by calling for the execution of its related process object.

3.5 GeoViewer

The GeoViewer is a GIS toolkit provided to query, view, and manipulate objects in the analysis frame. It can display object positions within the analysis frame or as a standalone application. The graphical objects used in the GeoViewer are dynamically linked to their corresponding domain objects in the analysis frame so that changes in the domain object are reflected in the GeoViewer's display.

3.6 Including External Models

External models are added to the DIAS database by adding or editing domain objects that correspond to the purposes of the model. For each domain object, parameters must be specified and aspect objects must be defined. Process objects must also be built for each process that is represented in the model's source code.

In addition, a model controller must be created that can execute portions of the model code depending on the process that is called. It is also responsible for relating the model's data references to the corresponding references in DIAS.

3.7 Application – MIMS

The Multimedia Integrated Modeling System (MIMS), developed by the US Environmental Protection Agency, was created to help link models created by different agencies within the EPA, including models from different disciplines.

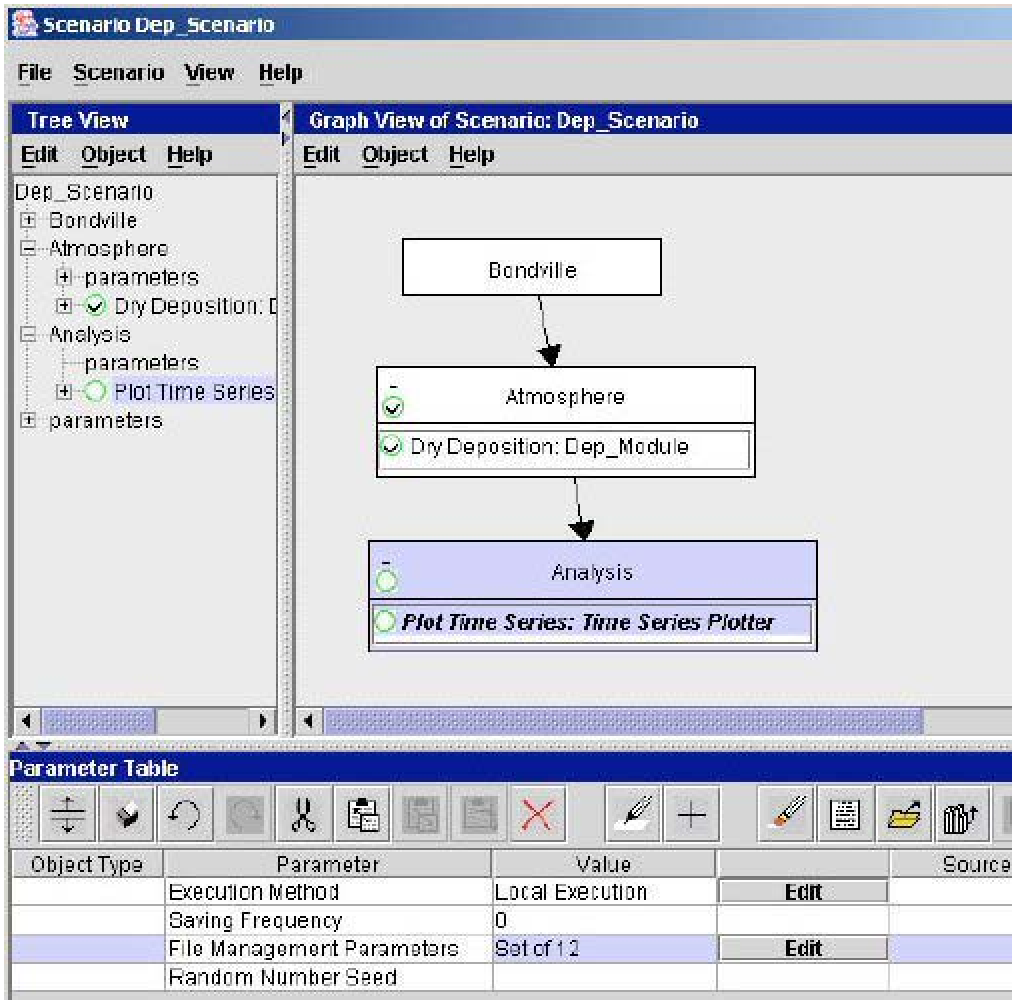
The MIMS framework layers on top of the DIAS user interfaces to facilitate bringing existing models into the simulation. A module is prepared for linking by creating a control file and defining the parameters that will be involved in the simulation. After the module has been defined, its corresponding domain object can also be created. Processes that will occur in the simulation must be defined with the domain object.

3.7.1 Data Exchange

Since MIMS depends on DIAS, the data exchange process is the same. Relationships between models are defined by a connection. The sequence of these connections determines the order of operations in the simulation. Parameters are passed between models through these connections as well. A module assumes the same value for a specified parameter as the domain object it is connected to. Tools are also available to make parameter connections explicitly. Each module has a control file to keep track of the parameters and their connections.

3.7.2 Setting up a Simulation

To create a linked model, domain objects are selected and flow arrows are created to indicate how information is exchanged between modules. Parameters can be explicitly connected at this time. When all inputs are supplied for each domain object, the model can be run. Figure 3-2 demonstrates how an integrated model is created in MIMS. The tree lists available domain objects and their corresponding processes. At the bottom is the parameter table which links parameters to domain objects. This table contains other pertinent information about the parameters as well.



Object Type	Parameter	Value		Source
	Execution Method	Local Execution	Edit	
	Saving Frequency	0		
	File Management Parameters	Set of 12	Edit	
	Random Number Seed			

Figure 3-2 MIMS user interface [source: adopted from MIMS User Guide]

To view results, an analysis domain object must be included in the linked model. Just like the individual models, the type of analysis that needs to be done must be specified.

3.7.3 System Requirements

The MIMS Framework is designed to operate in the Java Runtime Environment (JRE) version 1.4. It can be used on both Windows and UNIX systems.

4 OpenMI

The Open Modelling Interface and Environment (OpenMI) is developed by the European HarmonIT, an international project sponsored by the European Commission. A team was formed from 14 organizations and seven different countries, with the development being led by DHI Water and Environment, Delft Hydraulics, and HR Wallingford.

4.1 Background

The European Parliament and Council passed the Water Framework Directive in 2000 with the objective to achieve 'good ecological status of Europe's water resources by 2015'. Part of this objective is to have a plan established for each river basin in Europe, which will involve models crossing country boundaries.

Since integration is vital to meeting their goal, the main focus of the HarmonIT project is to provide a framework flexible and generic enough so that models can be linked to exchange information during execution of the simulation. Water and IT experts have been employed to develop a model linking process that can be used by both new and existing models from various organizations.

In 2004 the OpenMI was implemented and models were migrated to the new system. Beginning in December 2004 guidance was provided on how to use OpenMI. OpenMI was officially launched early in 2006.

4.2 Definitions

A *model application* is the entire model software system. It includes a *user interface* and an *engine*. The user interface allows the user to supply information and creates input files to be used by the engine. The engine takes those input files, runs calculations, and writes the results to output files.

When an engine has read the input information it becomes a *model*. When an engine can be represented separately and has a well-defined user interface, it is an *engine component* and a model created from this engine is a *model component*.

4.3 Data Exchange

OpenMI is not a framework where models are added and connected to each other. It defines an interface that a model can adopt to communicate with other models. By specifying a data transfer protocol, models using different file formats and languages can communicate with each other.

The basic concept behind the OpenMI interface is that one model requests information from another. It is simply two components connected by a link. More complex systems can be built by chaining more components with links. Each component would request information from the component at the other end of its link. The solution computed by one model becomes the input for the next. This exchange of data is automated and run without any user intervention.

Three components are defined in the data exchange process: the component requesting data (target), the component providing data (source), and the link. When a model is able to request or provide data through the OpenMI interface, it is considered to be a linkable component.

An OMI file is necessary for deploying a model. This is an XML file that describes the entry point for the linkable object and defines all the arguments that are needed to initialize the component.

4.3.1 Information Exchanged

When a request is made for information, the target component must specify an exchange item and a time. An exchange item consists of what the values represent (quantity), and where the values apply (element set).

4.3.1.1 Quantity

The quantity is a class that consists of an id string, a description string, the value type, the unit, and the dimension. The value type is either scalar, which is represented as a double, or vector, composed of an x, y, and z component.

The dimension is used to perform unit conversions. It uses base quantities such as length, time, and mass to define the value and to check if the dimensions are compatible. Base quantities defined in OpenMI are length, mass, time, electric current, temperature, amount of substance (mol), luminous intensity, and currency. This means that a unit of

m^3/s would have a dimension of $length^3time^{-1}$. Dimensionless units would have a power of zero for each base quantity.

The unit also contains an id and a description as well as a conversion factor and an offset factor to assist in performing unit conversions.

4.3.1.2 Element Set

The element set is an ordered set of elements. Each element contains a set of nodes which can be geo-referenced. Each element set contains an id string, a description string, a spatial reference id string, an integer indicating the element type, the element count, an integer indicating the version, as well as several functions to query the element set.

The version number is for element sets that might change over time, such as a wave model. The element type identifies the type of elements contained in the set. Elements of the same type can be combined in one element set, but models with different types of elements must have multiple sets. The functions provided in the element set will get the index, the id, the vertex count, the face count, the face vertex indices, the x coordinate, the y coordinate and the z coordinate.

Table 4-1 includes the list of element types that are defined in OpenMI. For all lines and polylines the first and last vertices indicate the direction of any fluxes. Vertices are defined counterclockwise for polygons and polyhedrons and the first and last vertices must be the same.

The functions provided in the element set will get the index, the id, the vertex count, the face count, the face vertex indices, the x coordinate, the y coordinate and the z coordinate.

Table 4-1: Element types available in OpenMI

IDBased	Elements that may or may not be geo-referenced
XYPoint	Geo-referenced point in the XY plane
XYLine	Geo-referenced line segments that connect two vertices/nodes in the XY plane
XYPolyline	Geo-referenced polyline that connects at least two vertices/nodes in the XY-plane
XYPolygon	Geo-referenced polygon in the XY plane
XYZPoint	Geo-referenced point in 3-dimensional space
XYZLine	Geo-referenced line segments that connect two vertices/nodes in 3-dimensional space
XYZPolyline	Geo-referenced polyline that connects at least two vertices/nodes in 3-dimensional space
XYZPolygon	Geo-referenced polygon in 3-dimensional space
XYZPolyhedron	Geo-referenced polyhedron in 3-dimensional space

4.3.1.3 Time

Because the models are exchanging data during run-time, it is important to know when a model is requesting data from another model. Time values can be represented as either a moment in time (time stamp) or as a time span, specified by a beginning and ending time. All time is referenced using the Modified Julian Date format.

4.3.2 Defining the Linkable Component

A linkable component must also have certain definitions in order to be recognized in OpenMI. A linkable component consists of an id string, a description string, a model id string, a model description string, an input exchange item count, an output exchange item count, the time horizon, the earliest input time, as well as a number of functions to assist in linking. These definitions, along with a brief description are provided in Table 4-2.

Table 4-2: Members of a Linkable Component

Component ID	Identification of the software unit
Component description	Description of the software unit
Model ID	Identification of the model provided by the software unit
Model description	Description of the model provided by the software unit
Input exchange item count	Number of data sets that can be accepted as input
Output exchange item count	Number of data sets that can be provided as output
Time horizon	Time span over which a component can provide data
Earliest input time	The earliest time that a component can provide data

4.3.3 Data Operations

Ideally the source component would provide the requested data exactly as the target component requires. Because there could be discrepancies in the two models, OpenMI provides a way for the source component to modify its data to match the requirements for the target component. The source component includes a number of data operations to define changes that can be made to the data. These can include spatial and temporal interpolations as well as unit conversions.

4.4 Configuring a Link

The link is defined by an id string, a description string, the Target Component, the Target Quantity, the Target Element Set, the Source Component, the Source Quantity, the

Source Element Set, and the Data Operations necessary to transform data to the correct format. The four steps in configuring a link are:

1. Select the source and target components
2. Select the source and target quantity
3. Select the source and target element set
4. Select the data operations

Because the link configuration is complex, OpenMI offers tools to assist in defining what element sets are available for linking. It also helps to visualize the link.

In order to prepare a system of linkable components with the associated links, OpenMI has prepared a Configuration Editor, shown in Figure 4-1, which allows the user to select all the linkable components, create the links, configure the links, and set any other parameters for the simulation. Linkable components are stored and can be easily accessed to select from. A trigger is also included in the simulation to prompt the simulation to begin.

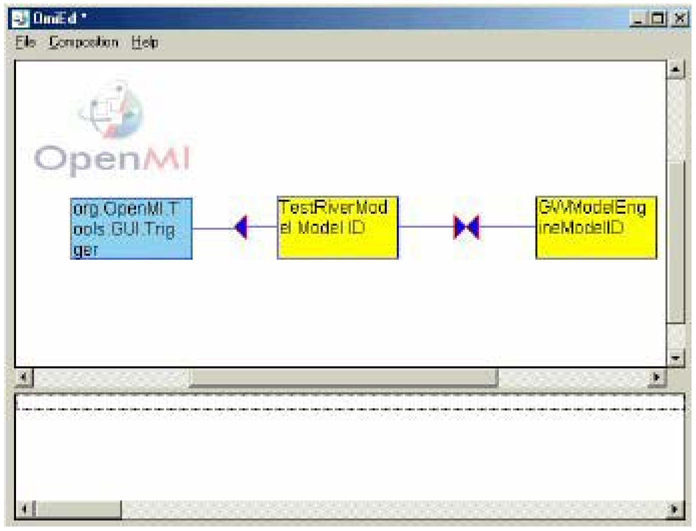
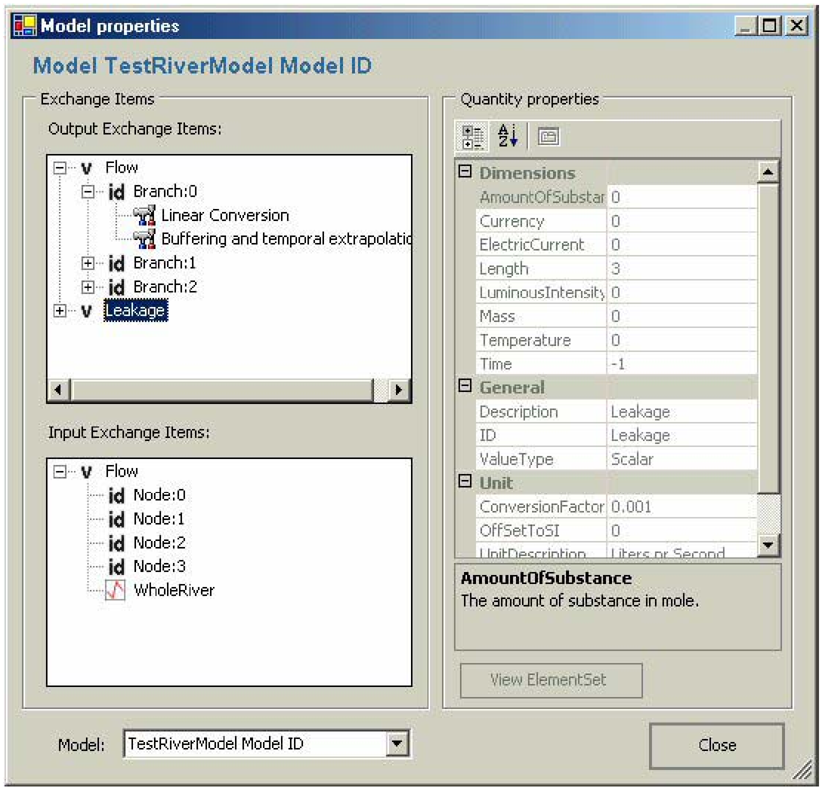


Figure 4-1 OpenMI Configuration Editor
[source: adopted from (Tindall, 2005)]

The link is formed simply by drawing a connecting line from one model to the next model with the mouse. Separate links must be formed if more than one quantity is exchanged. Double clicking on the link brings up a dialog which allows the user to configure the link by providing available quantities, element sets and data operations. Each model is identified by an OMI file. This is an XML formatted file that provides metadata for each linkable component to allow modelers to know how to link models together.

The properties of an individual model can be viewed by double clicking on its icon. The Configuration Editor will bring up a Model Properties dialog as illustrated in Figure 4-2. It displays the available input and output exchange items along with the associated quantity properties.



Output Exchange Items:

```
Flow
  Branch:0
    Linear Conversion
    Buffering and temporal extrapolatic
  Branch:1
  Branch:2
Leakage
```

Input Exchange Items:

```
Flow
  Node:0
  Node:1
  Node:2
  Node:3
  WholeRiver
```

Dimensions	
AmountOfSubstar	0
Currency	0
ElectricCurrent	0
Length	3
LuminousIntensity	0
Mass	0
Temperature	0
Time	-1
General	
Description	Leakage
ID	Leakage
ValueType	Scalar
Unit	
ConversionFactor	0.001
OffSetToSI	0
UnitDescription	Liters pr Second

AmountOfSubstance
The amount of substance in mole.

Model: TestRiverModel Model ID

Figure 4-2 OpenMI Model Properties dialog [source: adopted from (Tindall, 2005)]

4.5 Establishing a Compliant System

There are several phases involved in preparing an integrated model system. A linked object must be able to support each of these steps.

4.5.1 Instantiate and Initialize

The linkable component must know how to populate itself with model data and reveal the items to be exchanged. To do this, the OMI file is read which refers to the software that implements the linkable component. This allows the linkable component to be created. The OMI file also provides the arguments necessary to initialize the linkable component. Usually the arguments refer to data files.

4.5.2 Inspection and Configuration

The steps involved in this phase include creating the relevant links, quantities, element sets and data operations, checking the validity and populating them. The links must then be added to the components and validated. If the exchange items are static and known they can be retrieved by asking for the number of exchange items and looping over the list. However, if the exchange items depend on the connected components, a dynamic query will take place.

4.5.3 Preparation

This phase is to help define a clear starting point for the computation. At this time connections to databases or networks could be established and model engines can be prepared by populating themselves with the necessary input data, opening output files, and organizing buffers. This phase must also include a final validation on the status of the linkable component.

4.5.4 Computation/Execution

This is when the data transfer occurs. It is recommended that each time step requests data from the source component instead of simply using the end value since control doesn’t return to the main application until the computation is finished. This way the application has an opportunity to react to any external events.

4.5.5 Completion

The completion phase is used to close files and network connections and to clean up the memory.

4.5.6 Disposal

The final phase is provided to de-allocate memory.

4.6 Making Models OpenMI Compliant

The following requirements must be met to be compliant with OpenMI:

- The component must implement the org.OpenMI.Standard.ILinkableComponent interface.
- The component must be associated with a properly configured XML file, which contains information necessary for deployment.
- The component must be able to handle functions called in the proper order.

Models are usually compiled in executable files which are not accessible by other components and can't be used with OpenMI. Instead it must be compiled into a dynamic link library (dll) file. A new executable should be created that calls a function in the dll that will run a full simulation.

The second step in transferring the model is to create the appropriate wrapper classes. Wrapping entails creating a C# class that implements the Linkable Component interface. All communication between the model engine and the interfaces takes place through the wrapper. This keeps OpenMI implementations separate from the engine core.

The Utilities Wrapper package is provided by OpenMI to help model developers make their models OpenMI compliant. It is not necessary to use the Utilities Wrapper package, but it contains a generic wrapper called the Linkable Engine that can be used for models doing time step-based computations. The Linkable Engine provides a default implementation of the Linkable Component interface as well as required functions that the model may not be equipped to perform.

Some of the functions that are included with the wrapper are temporal interpolation and extrapolation, spatial operations, and event handling. The Linkable Engine also handles buffering. A model may be queried for values that correspond to a previous time step. Since most models only keep values for the current times step, the buffer stores data associated with the link. Classes associated with the Linkable Engine are described in Table 4-3. The relationship between these classes is illustrated more clearly in Figure 4-3.

Table 4-3 Linkable Engine classes

MyEngineDLL	Represents the compiled core engine code
MyEngineDLLAccess	Translates the Win32Api from MyEngineDLL to .NET
MyEngineDotNetAccess	Makes sure the calling conventions and exception handling follow .NET conventions
MyEngineWrapper	Allows the Engine Access interface to be accessed by the Linkable Engine class
MyLinkableEngine	Creates the MyEngineWrapper class

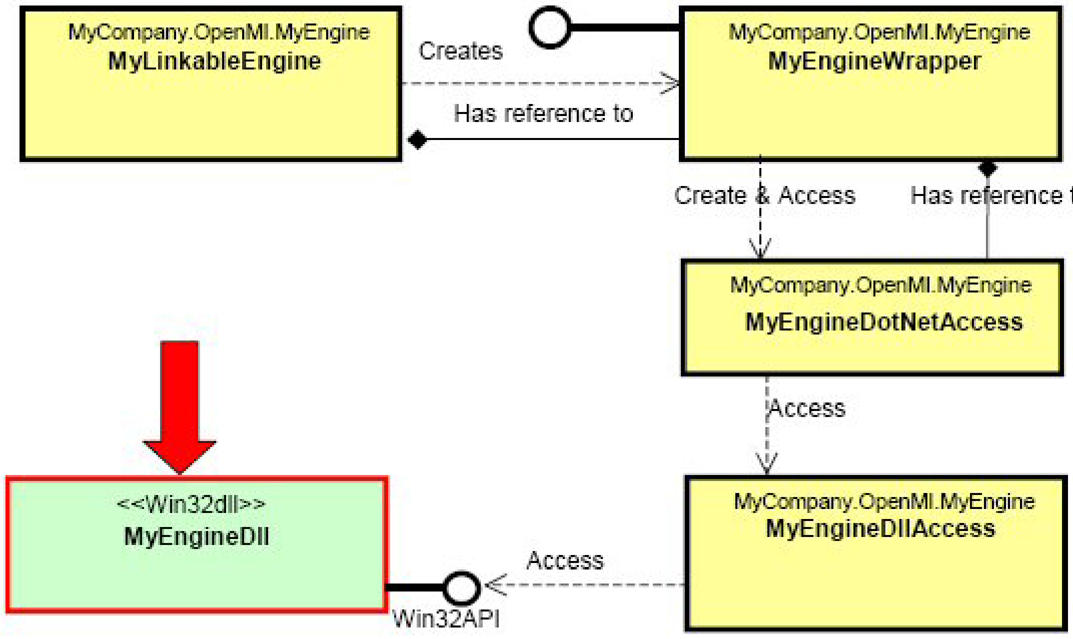


Figure 4-3 Linkable Engine wrapper classes [source: adopted from (Tindall, 2005)]

The process involved in making a model OpenMI compliant begins by converting the model engine from an .exe file to a .dll, which can be accessed by other components. After the wrapper classes listed in Table 4-3 have been created in the .NET development environment, they can be implemented in the model code. Although the order of implementation is not required, it is specified here so that each class can be tested as it is added.

First implement the MyEngineDLLAccess class, which converts all exported functions in the engine core code to public .NET methods. The MyEngineDotNetAccess class should be implemented next. It will change the calling conventions to C# conventions and convert error messages into .NET exceptions. Implement the MyEngineWrapper class next, which will execute the Linkable Engine interface. Once that is complete the MyModelLinkableEngine class can be implemented. This is the linkable component that will be accessed by other models.

After the basic structure of the engine and wrapper is complete, further implementation must be done. This implementation can either be done in the engine wrapper or the engine core. This will vary depending on the model. The first thing that must be considered is the different simulations that could be run from the model. With this in mind, exchange items can be defined. This includes both what information needs to be exchanged (quantity), and where (element set), as well as if it is available for input or output.

4.7 Linking Other Applications

A linkable component does not need to be a model system. OpenMI allows a model to be linked to a variety of data providers.

4.7.1 ASCII files

Often the input for models is located in an ASCII formatted file. These files can be made available to use in OpenMI by wrapping them. This is especially useful when the output of one component is the input for another.

4.7.2 Spreadsheets

It is also possible to access spreadsheets through OpenMI. There are several ways to do this. One is to redirect the output to a file that can be read by desktop applications, such as the “comma separated value” (.csv) file used to export data from Excel or any file that can be imported by a GIS system. Another way is to use “Visual Tools for Office” in Visual Basic .Net which allows communication between an OpenMI component and a Microsoft Word or Excel document.

4.7.3 Crystal Report Engine

It is also possible to develop an OpenMI compatible tool that will use the Crystal Report Engine (from the .Net environment) to define a standard report that can be exported to different industry standards. These include Adobe Acrobat (.pdf), Crystal for Visual Studio .Net (.rpt), HTML 3.2 and 4.0 (.html), Microsoft Excel (.xls), Microsoft Rich Text (.rtf), and Microsoft Word (.doc).

4.7.4 Databases

Databases can be accessed by OpenMI as well. Since databases do not progress in time, it might be necessary to interpolate or extrapolate for the necessary time step. The .Net environment has a set of classes that provide access to databases.

4.8 Advanced Controllers

In some instances additional control capacity is needed for computation. Advanced controllers are available to allow the integrated model to perform iterations, optimization, and calibration.

4.8.1 Iteration Controller

The iteration controller is placed between any components that need to be iterated. It is recommended that a bi-directional link be used instead of an iteration controller because it runs faster and is easier to set up. Also, models need to be able to return to their original state for the iteration controller to work, which not all models are capable of doing.

4.8.2 Optimization Controller

An objective function is needed to use the optimization controller. The objective function takes a number of parameters and calculates values which it returns to the optimizer. The optimizer tries to minimize the objective function. The OMI file specifies the number of parameters, the minimum, the maximum, and the starting value for the optimization.

Calibration can be done using the optimizer by selecting a special objective function that takes the difference between the measured and calculated data and returns it to the optimizer, which sets the parameters for the calculation model.

4.9 Summary

OpenMI allows models to exchange data by implementing the OpenMI interface. The three components involved in the data exchange process are the target component, the source component, and the link. Each linkable component must have an OMI file associated with it to be recognized by OpenMI.

An overview of the steps necessary to make a model OpenMI compliant are:

1. Create a dll that will run the simulation.
2. Create wrapper classes.
3. Provide definitions for the information that will be exchanged.

The information that must be defined when exchanging data include:

- What the values represent (quantity)
- Where the values apply (element set)
- When the values apply (time)

Data operations can be provided as well so that the source component can convert the data into the format required by the target component.

Advanced controllers are also available to allow the integrated model to perform iterations, optimization, and calibration.

5 Other Integration Software

There are several organizations that are linking models together to form complex water systems. This chapter contains brief descriptions of additional model integration software that has been developed.

5.1 MMS

The US Geological Survey's National Research Program is developing the Modular Modeling System (MMS) to address the problems associated with model selection, application, and analysis. The objective is to create the ideal model for a specific application by selecting the most appropriate algorithms from applicable models.

5.1.1 System Requirements

MMS was originally developed for UNIX-based workstations and uses X-windows and Motif for the GUI. Operating systems that have been successfully used are Sun (both SunOS 4 and Solaris), Data General (DGUX), Hewlett-Packard (HP-UX), IBM (AIX), Silicon Graphics, and PC (Linux). More recently, MMS has been precompiled to run under the Windows operating system using the CYGWIN software package.

5.1.2 File Control

Each module stores parameters, variables, and dimensions in three separate internal databases and can be passed between modules by library functions. The integrated model also has a control database that saves the status of all features so that it can be reopened with all of its parameters defined. Additionally, all models have an environment file that specifies directory paths, system files, and other MMS related utilities. Most importantly, this file specifies the location of the control file or database.

A user directory must first be created so that modules can be added to it. FORTRAN or C modules can be chosen from the module library and linked together using an interactive model builder interface. New modules can also be created in MMS and existing code can be modified to be used as a module in MMS. When a model is first initiated, all the modules are read through to create the parameters database. There is functionality provided to edit parameter files and data files.

5.1.3 Organization

As illustrated in Figure 5-1, MMS is organized into pre-processing, model application, and post-processing functions. Pre-processing contains tools to prepare spatial and time-series data for use in the model application. This includes GIS and data collection tools. Model integration occurs in the model application section. A module library contains the modules necessary to simulate various processes. Post-processing uses a variety of visualization tools to analyze and apply model results, including a GIS tool for spatial analysis.

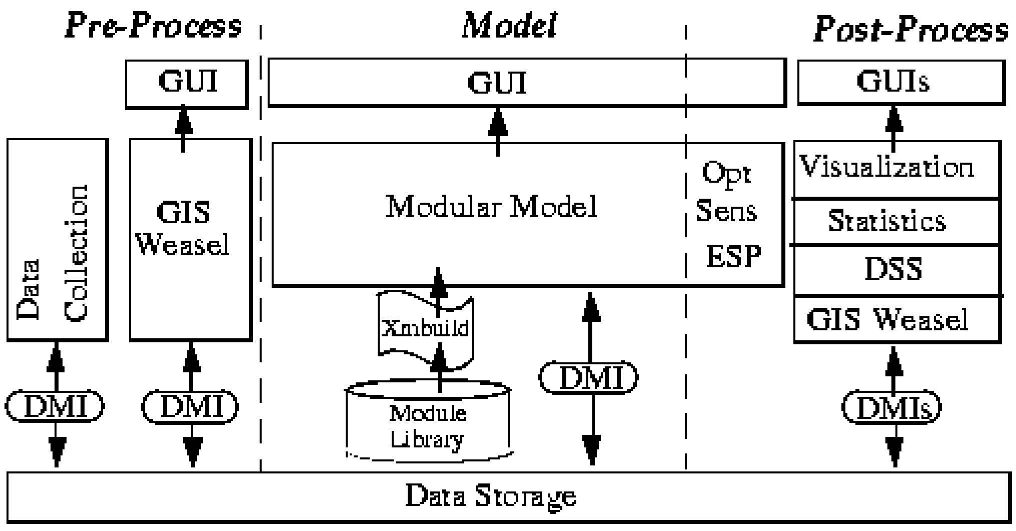


Figure 5-1 MMS organization [source: adopted from MMS User's Manual]

5.1.4 Running a Simulation

Multiple files can be read simultaneously for a single model run. Data lines in all input files must be ordered from earliest to latest. When multiple files are used, the first line from each file is read and the earliest time stamp is selected. When data are needed for the next time step, the next line in the file is compared with the other files to select the earliest time stamp. This process continues until there are no more data to be processed.

MMS provides several graphs to view the results of the model run. Sensitivity analysis graphs allow the user to view contours and plots as well as a simple 3D view. Probability distribution curves can also be viewed for various parameters including the maximum flow and volume.

5.2 OMS

The Object Modeling Software is developed by the Great Plains System Research Unit. It is a collection of software tools that can be connected into an open modeling platform. It uses NetBeans Desktop Application Framework which allows customization of the application based on user requirements.

5.2.1 Description

OMS is composed of two parts, the OMS Platform and OMS Modules. The platform is a skeleton to which users can add tools. The NetBeans application core provides management support, file system interfaces, event handling, data persistence service, and an automated update facility.

OMS Modules provide support for modeling. The modules provide templates for component development, create model applications, visualize data, and interface with GIS. OMS modules can be plugged into other platforms written for the NetBeans Platform.

5.2.2 Component Based Design

Components are executable software pieces that are linked to form specific simulation models. Each component is a self contained software unit that is separated from its environment.

The component-based system makes the component an object-oriented unit that can be developed and tested independently. It interacts through well-defined interfaces. This keeps the call for information separate from the implementation. It also allows a single component to be reused in several different applications.

One of the attributes of a component is that it has one specific function in the model. It can represent a physical process, a management action, a data gathering element, or the presentation of results. A component must also have its function, data requirements, and available data completely defined.

Components are divided into four groups: scientific, scientific utility, control, and data input/output. The scientific components use methods and equations to represent a physical process. Scientific utility components provide statistical analysis methods that can be made available to the scientific components. The control components manage the execution of the model. Examples include time management or a convergence criterion. The Data input/output component provides data to other components in the simulation. They control data transfer from files or databases. Visualization components also fall under this category.

5.3 WEAP

The Water Evaluation and Planning System (WEAP) is developed by the Stockholm Environment Institute's Boston Center at the Tellus Institute, with the US Army Corps of Engineers' Hydrologic Engineering Center funding significant enhancements. Project support has been provided by agencies such as UN, World Bank, USAID, US EPA, IWMI, AwwaRF, and the Global Infrastructure Fund of Japan. WEAP is available in English, French, and Korean, with Chinese, Spanish, and Portuguese versions almost finished.

WEAP is designed to provide a holistic analysis of water management by applying supply, demand, water quality and ecological considerations to the analysis. It does not link separate models together, but rather incorporates all these considerations into one single model.

5.3.1 Description

An end-use, demand-driven approach is used in analysis. Agricultural, urban, and industrial water demands can be broken down to their specific needs to place the development objectives at the foundation of the analysis. For example, industrial demands can be broken down by sector as well as into process water and cooling water needs. These simulations also take pollution factors into account and can track polluted water from generation to treatment to outflow into surface water and ground water.

Different scenarios can be analyzed by first creating a current account of the water system and establishing a base case, or typical water policy to be used as a reference. Alternate scenarios can then be established and compared to the base case.

5.3.2 Setting up the Model

Creating a study area is the first step in the analysis. The study area contains a set of information and assumptions about a system of linked supply and demands. The same geographic area can be defined with different configurations or operating assumptions, forming different study areas.

The system is represented in terms of its supply sources; withdrawal, transmission and waste water treatment facilities; water demands; pollution generation; and ecosystem requirements. Additionally, the time frame, spatial boundaries, and configuration of the problem must be established.

A system schematic can be created by the "drag and drop" method, using a GIS-based graphical interface. Elements can be overlain on an ArcView map and other standard GIS and graphic files.

Nodes, representing physical components such as demand sites or water treatment facilities, are connected by lines which correspond to the water conduits such as rivers or pipelines. Nodes can include a start up year so that facilities built after the base year can be included. Figure 5-2 shows how these connections are represented in WEAP.

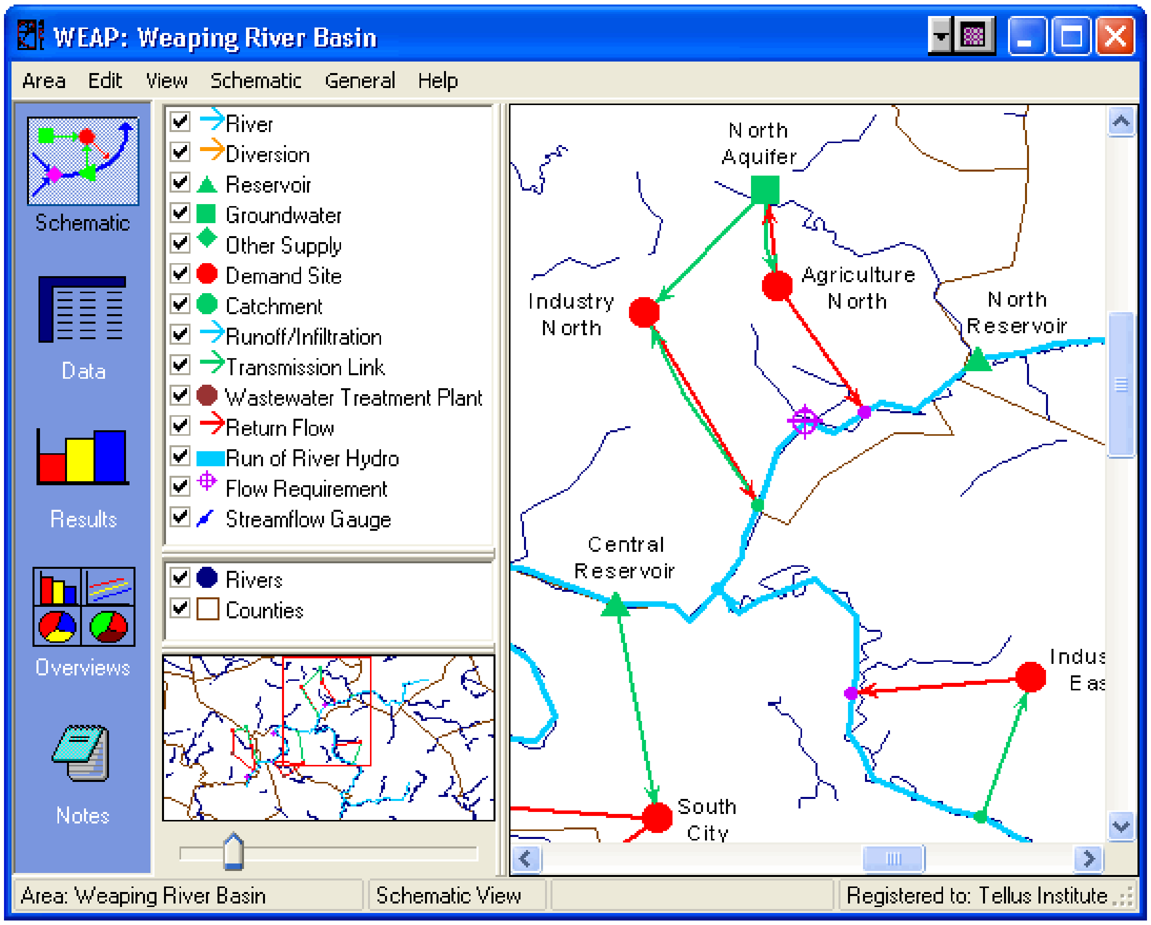


Figure 5-2 WEAP user interface [source: adopted from the WEAP website]

The model is then calibrated with actual water demand, pollution load, resource, and supply data for the system.

Various assumptions about impacts of policies, cost, and climate on water demand, supply, hydrology and pollution can be investigated. These scenarios are evaluated according to water sufficiency, costs and benefits, compatibility with environmental targets, and sensitivity to uncertainty in key variables.

Once all the data has been entered and the simulation run, reports can be generated depicting all aspects of the system. Either monthly or yearly results can be displayed for any time period within the study. Reports are available as graphs or tables and can be saved as text, graphics, or spreadsheet files. Results for up to nine different scenarios can be viewed side by side. Reports can be customized and saved.

5.4 HEC-WAT

The U.S. Army Corps of Engineers Hydrologic Engineering Center's Watershed Analysis Tool (HEC-WAT) is designed to help coordinate project studies to provide greater consistency between multi-disciplinary teams. A BETA version is expected to be released in September 2006 and an official release in September 2007.

5.4.1 Objectives

The main objective of HEC-WAT is to assist in comprehensive water management studies by simplifying the analytical process, helping to coordinate between team members, and providing more consistent results.

This is done by supplying methods for data sharing and model interfacing along with allowing results to be viewed from a single interface. Modeling teams will be able to track the progress of other models and retrieve results from previous model runs, while management will be able to track the progress status and display results for meetings.

A control and visualization interface (CAVI) will allow data and results to be shared, a common schematic assembly, alternative definition and representation, model setup, editing and implementation, and data and results visualization.

5.4.2 Description

The CAVI is used to develop the spatial representation of the study area. This can be used to simulate multiple alternatives. An example of a HEC-WAT simulation is shown in Figure 5-3.

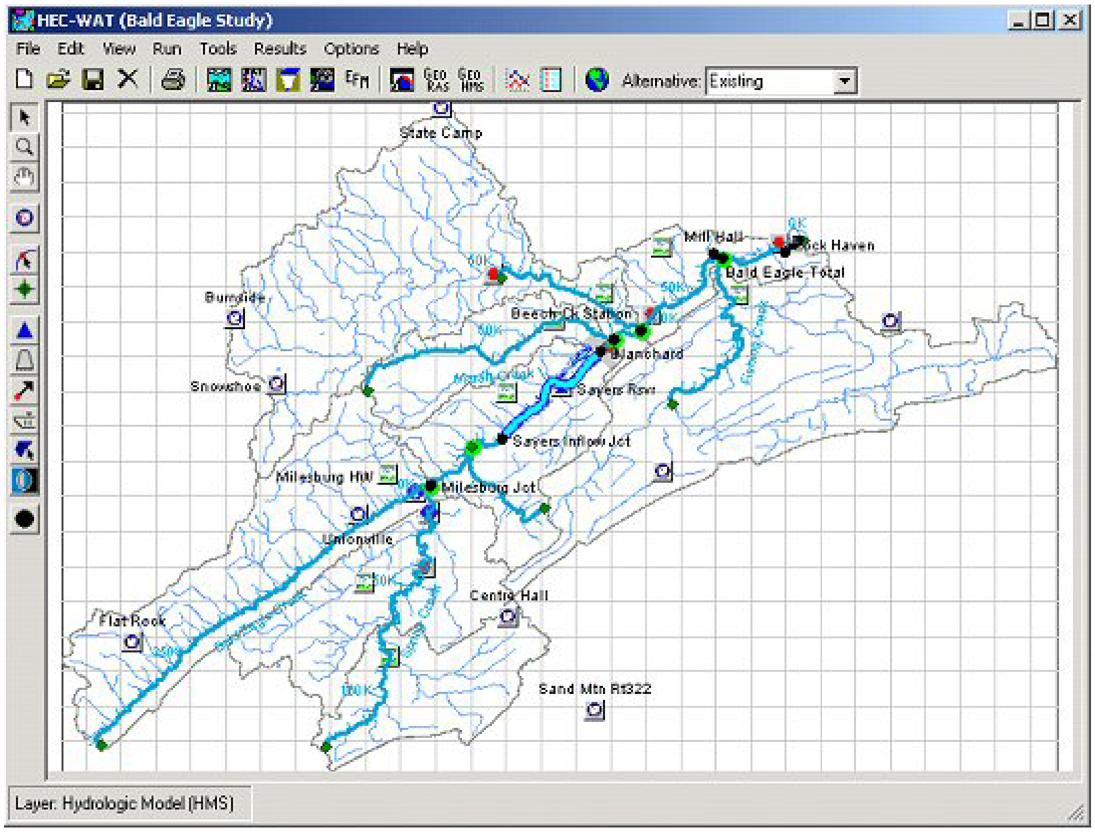


Figure 5-3 HEC-WAT simulation [source: adopted from (Dunn, 2005)]

The first step in the study is to define the watershed as well as background layers such as gage layers, stream alignment layers, river station coordinate system, and surface TIN's or DEM's. Next, a stream alignment must either be drawn with the CAVI or imported. The third step is to define common computation points (CCP), which are locations where one model transfers data to another model or where results are needed for further analysis.

An alternative is defined by the watershed extents, the stream alignment, common computational points, and the flood control and ecosystem restoration measures. A base alternative is established first by defining the existing conditions. Additional alternatives can be defined by changing the measures. This leads to more consistent modeling and results because each team can use the same schematic to do their modeling. When a model is built it is entered into a table for its associated alternative. When all models for a particular alternative have been built and entered into the table, parameters for a single model can be edited and the whole series of models rerun.

5.4.3 File Organization

The data for each HEC-WAT study is located together in a directory so that individual model data can be easily located. All information is stored under the directory "WAT" which contains various project directories. All project directories contain directories for each model in the study as well as a "Common" directory, "Maps" directory, and "WAT" directory.

The common directory contains files shared by models or by a model and the HEC-WAT program. Included in this directory is a common Data Storage System (HEC-DSS) time series data file that contains all DSS time series data for all models, alternatives, and time windows. The maps directory contains background maps and images.

A model created outside of HEC-WAT can be incorporated into the system by copying its contents into the HEC-WAT directory and rewiring the DSS file and common data to the common directory.

When a model is executed, the input and output will be passed to the model along with the time window and other information. The way it is passed in will depend on how the model is executed in HEC-WAT. Usually it will be supplied as a script or an input file.

5.4.4 Visualization of Results

Any hydrologic object or element may be selected from a particular alternative and a list of graphical and tabular results will be presented. Selecting the table or graph of choice will display the appropriate output onto the screen. Multiple alternatives may be selected so that results can be compared.

5.5 SMS Steering Module

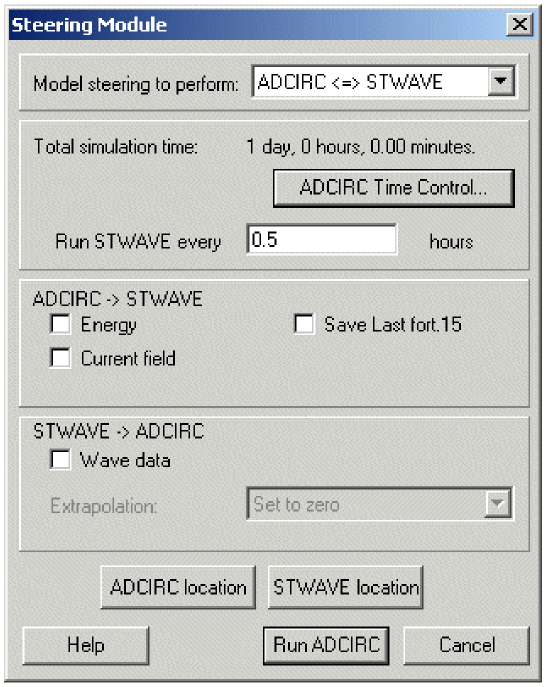
The Steering Module is developed for use in the Surface water Modeling System (SMS) software. It is designed to facilitate sharing of data between models. It addresses both single model control and multiple model coupling. To fit the purposes of this research, only the model coupling aspect will be mentioned.

5.5.1 Background

The STWAVE and ADCIRC models included in SMS can use the output files generated in one model as the input for the other model. Differences in the models create challenges for exchanging data. STWAVE performs simulations with a finite difference grid over a small domain, while ADCIRC uses a finite element mesh for computation over a much larger domain. The Steering Module provides a method for automating the information transfer so that data are interpolated or extrapolated accurately.

5.5.2 Description

Because the models are already integrated into the SMS interface, each model must be set up with its necessary parameters in SMS. The Steering Module dialog, shown in Figure 5-4, can then be implemented to control the link between the models.


Steering Module

Model steering to perform: ADCIRC <=> STWAVE

Total simulation time: 1 day, 0 hours, 0.00 minutes.

Run STWAVE every 0.5 hours

ADCIRC -> STWAVE
☐ Energy
☐ Current field
☐ Save Last fort.15

STWAVE -> ADCIRC
☐ Wave data

Extrapolation: Set to zero

Figure 5-4 SMS Steering Module dialog [source: adopted from (Zundel, 2002)]

In the steering module dialog the user can specify how long one model will run before reading in information supplied by the other model. There are also options to select which files should be saved. After setting these parameters, the process is completely automated, with the Steering module controlling the model runs.

5.5.3 Running the Simulation

There are three options available for linking the models together. A one-way connection can be established where ADCIRC provides input for STWAVE or for STWAVE to provide input for ADCIRC. A two-way link can also be created.

The process of data exchange is similar for each link option. One model will run for the specified amount of time before needing information from the other model. The first model will break and the second model will run, generating the output files necessary for the first model to continue running. This may happen several times during a simulation.

5.5.4 System Requirements

SMS runs on Windows XP and 2000.

5.6 Summary

There is a great diversity in the model integration software that is available. Several organizations have developed their own model linking systems to address their specific needs, such as HEC-WAT and the SMS Steering Module. HEC-WAT is used to coordinate model studies in large projects, while the Steering Module facilitates the interaction between two specific models. Other software is created so that any type of model can make use of the technology, such as MMS and OMS. WEAP has been developed to create complex water systems without the need to link models together. This great diversity in model integration software reflects the complexity of problems involved in linking models together.

6 MLO

As explained in the introduction, XMDF is a file format that stores the most commonly-used modeling geometry (meshes, grids) and associated arrays. The main objective of this research is to design a Model Linkage Object (MLO) in XMDF that facilitates model linkage for models that have adopted the XMDF format. To keep the scope of the research manageable, the MLO was designed initially for two-dimensional meshes and the main focus was on FRAMES and OpenMI. FRAMES was selected because the Army Corps of Engineers already use this framework in many of their simulations, while OpenMI was chosen because it allows models to connect and exchange information during run time. The MLO makes use of the mesh data and functions already stored in XMDF.

An MLO is a sub-region of a mesh that has been identified by the user as a potential model-linkage location. In general terms, an MLO could consist of a set of grid cells or cell faces, or a set of mesh nodes, edges, elements, or 3D element faces. An MLO should include methods for automatically extracting data from solutions written in XMDF format and packaging the data in the format required by the model-linkage system. While an MLO would not fully automate the linkage process, it should greatly reduce the amount of customization and data translation required by the model developer/user.

The XMDF format already provides some functions to query the mesh data, including functions that will get the number of nodes, the number of elements, the element types, and the node locations. The MLO should contain additional functions to allow the data to be modified or queried.

The main benefit to including the MLO in XMDF is that all of the interpolation and extrapolation can occur directly inside the API. If the nodes in the MLO correspond directly with the computational points on the model to which a linkage is being made, no interpolation needs to occur. However, if the MLO nodes are not the same as the target computation points, the API will need to interpolate values from the mesh. A method of interpolation can be stored with the MLO or a default method of interpolation can be used. Examples of interpolation methods that could be used include linear, inverse-distance weighted, and kriging. In addition to these methods, it is also possible that a user may desire to obtain the mean, maximum, or minimum value of a set of nodes. This may be applicable when one value is desired for the entire computational area corresponding to the MLO.

6.1 Creating an MLO

There are several properties and methods that would be required in order for an MLO to be useful for model linkage. There must be a method for selecting the locations where data will be exchanged. The units and the coordinate system must also be specified. Since the mesh already has units and a coordinate system, the MLO will be populated with these data when first created. The data sets associated with the mesh and therefore available via the MLO must be indicated as well. Table 6-1 summarizes the properties that should be associated with the MLO.

Table 6-1 MLO properties

ID for the MLO	Identification for the MLO
Description	Description of the MLO
Type	Type of location (e.g. node or node string)
Units	The unit of the object requesting information
Coordinate System	Coordinate system of the object requesting information
Locations	Coordinates of the vertices in the MLO
Associated data set name(s)	All data sets connected to the mesh (e.g. recharge)
Transient flag	Indicates if the data sets are steady state or transient

There should also be functions associated with the MLO to allow it to query the data and perform the necessary interpolations. Functions included with the MLO are described in Table 6-2. Of special importance is the ability to interpolate the data. There are several options that can be provided to interpolate the data, including linear, inverse distance weighted, and kriging, among others. An initialization function is provided so that the interpolation option can be selected and the units and coordinate system of the target system can be specified. After the interpolation has been initialized, the interpolation function takes an XYZ coordinate and a time value and returns the interpolated value. All unit and coordinate conversions are performed in this process since the units and coordinate system of the target system were provided in the initialization function. There are is also a function provided take a time value and return the min, max, or mean of the locations provided in the MLO.

Table 6-2 MLO functions

Get Num Values	Get the number of values being requested
Get Node Index	Get the index of a particular node in the MLO
Get Node Id	Get the id of a particular node in the MLO
Get Num Nodes	Get the number of Nodes in the MLO
Get X	Get the x coordinate of a particular node
Get Y	Get the y coordinate of a particular node
Get Z	Get the z coordinate of a particular node
Get Units	Get the units specified for the mesh
Get Coordinate System	Get the coordinate system defined for the mesh
Initialize Interpolation	Specify the interpolation scheme along with the units and coordinate system of the target object
Interpolate Data	Takes an XYZ coordinate and a time value and returns an interpolated value
Single Value	Takes a time value and the option desired (min, max, or mean) and returns a single value

6.2 Linking to FRAMES

This section describes how the MLO could be used to facilitate a link with the FRAMES system. This may require a few additional properties or functions beyond the general set described above. The MLO not only organizes the model linkage location, but also helps to package the data in the format required by the model linkage system. Because the data exchange in FRAMES occurs between input and output files, the MLO must be able to organize the locations and their associated values into the appropriate files for FRAMES.

6.2.1 FRAMES 2.x Files

Data passed between models are grouped into datasets. Datasets (in the context of FRAMES) represent information such as locations, concentrations, and time values. FRAMES 2.x uses comma-delineated text files stored with a DIC extension to define the datasets being exchanged. These files are referred to as data dictionaries. The main purposes for these dictionaries are:

- To describe the system or its components (system DICs)
- To provide data specific to a particular module (module-specific DICs)
- To provide information to multiple modules or transfer data between components (boundary condition DICs)

Module specific dictionaries describe the data required by a specific module to operate, while the boundary condition dictionaries represent the output results from a module and describe the information that is passed from one module to another.

Because the MLO is concerned with data exchange, it will only need to interact with the boundary condition dictionary files. The model itself may need to create other dictionary files to be compatible with FRAMES. For example, in order for a module to be incorporated into a FRAMES simulation, a user interface dictionary must be provided. The user interface dictionary is an input dictionary for a specific module and describes the data a model must have in order to run. No other modules can access this dictionary.

Additionally, each module is recognized in FRAMES 2.x by an ASCII file with a MOD extension. Information in the MOD file includes executable information, reference information, contact information, online information, requirement information, and connection-scheme information.

6.2.2 FRAMES 2.x Dictionaries

In addition to the MLO properties mentioned earlier, the MLO will need to be able to populate the correct dictionaries defined in FRAMES with the information that will be exchanged with other modules.

Several data dictionaries that describe locations have already been defined in FRAMES 2.x. These include Aquifer Polygons, Aquifer Points, Surface Water Polygons, Surface Water Points, and GeoReference dictionaries. These dictionaries are outlined in Appendix A. Dictionaries can be edited or added to FRAMES as needed. For a two-dimensional mesh, the point dictionaries and the GeoReference dictionary would be the most applicable dictionaries for the MLO to populate.

The GeoReference Dictionary is provided to allow modules to be coordinated with each other based on geo-spatial information. Currently GIS middleware is being developed so that this can be implemented. Since the necessary conversions to GIS data are still being made the GeoReference Dictionary may be modified in the near future. The point dictionaries contain the x, y, and z values of the vertices.

Dictionaries are defined by variables such as recharge and water flux. These variables can be tied to other variables within the same dictionary or in other data dictionaries. In this way, recharge or flux values can be tied to locations. For example, the “recharge” variable in the Aquifer Flux dictionary is tied to the “feature” variable in the Aquifer Polygon dictionary, which identifies the location of the recharge.

For each variable in the dictionary, properties must be stored to describe how the data can be used. Some of these properties include the data type (integer, double, string, Boolean), whether the data is scalar or vector, minimum and maximum value constraints, and the type of measure and unit. Default properties are stored in the dictionary until it is populated by the dataset. These are the properties that the MLO must populate with data.

Table 6-3 shows how properties in the FRAMES location dictionaries can be populated with data from the MLO. It should be noted that some of the properties provided by the MLO are unnecessary for FRAMES.

Table 6-3 Populating FRAMES dictionary properties with the MLO

DIC property	DIC property attributes	MLO property or function
Feature Point	• Unit • Measure • Location	• Get Units function • Get Units function • Location property
Attributes	• Identify the data set	• Associated data set name(s)
Axis	• Axis labels	• Get Coordinate System function
Coordinates	• Coordinates of the origin • Unit • Measure	• Get Coordinate System function • Get Units function • Get Units function
Point Id	• Name of the origin	• Get Coordinate System function

The location dictionaries also define DataBinding, DataQuery, DataTable, and Feature attributes. These are optional parameters that contain a reference to a geographic object contained in a database. By providing the reference to the object FRAMES does not need to transfer all the associated information. The MLO does not provide any functionality for these attributes.

Because the MLO is related to location information, only the dictionaries related to location have been studied in detail. FRAMES also defines dictionaries for other data that may need to be exchanged, such as aquifer flux and aquifer chemical concentrations. These are not reviewed in detail here because the dictionaries required will vary depending on the functions of the specific models exchanging data.

6.2.3 Using the MLO with FRAMES

In order to allow FRAMES to access the information, the MLO will need to be able to provide the necessary datasets in the file format required by the data dictionary. This means that the MLO must have a function to write the necessary dataset files. Since a file is required for each dictionary, the MLO will need to be able to write the coordinate information for the GeoReference dictionary and the other location information for the point dictionary. The required format is a comma-delineated text file. Once the information is in the required format, FRAMES will be able to access it.

FRAMES has several extraction tools which enable it to obtain the necessary data. FRAMES Data Owner Tool (DOT) maps the database structure to a file. This file describes how the data will be mapped and holds the database schema and the schema of the database dictionary. When FRAMES needs the information provided in the database, the Data Client Editor (DCE) calls the Data Extraction Tool (DET) which retrieves the extraction plan from the DOT database, uses a Structured Query Language (SQL) query to extract the appropriate data, and returns the data to the DCE through the Hypertext Transfer Protocol (http).

6.2.4 Additional Linkage Requirements

In addition to creating an MLO the following steps are necessary to make a model compatible with FRAMES 2.x:

- The units and measures must be defined in the Conversion Editor.
- Data dictionaries, as described above, must be defined. FRAMES provides a Dictionary Editor to create these from within FRAMES.
- Three arguments must be passed on the command line.
- Connection schemes must be defined in the Domain Editor.
- To facilitate the user in creating a linked simulation, the model must be categorized.

Further details regarding these steps can be found in the FRAMES section. Most of these parameters will be defined inside of FRAMES. The parameters that are defined in the model itself are minimal. There are three parameters that must be passed when a model execution is called:

1. Which invocation the module should process to produce the desired values
2. What simulation files are part of the invocation
3. Where the files are located

6.3 Linking to OpenMI

Unlike FRAMES, OpenMI does not exchange information through input and output files, but data transfer occurs during run time. Because of this there is more complexity involved than simply formatting the data that will be transferred. It is important to differentiate which aspects of the linking processes can be facilitated by the

MLO and what changes need to be made in the model itself in order for the model to exchange information through OpenMI.

The basic requirements for a model to be OpenMI compliant are:

1. It must implement the org.OpenMI.Standard.ILinkableComponent interface.
2. It must be associated with an XML file with an OMI file extension containing deployment information.
3. It must be able to handle the necessary function calls shown in Figure 6-1.

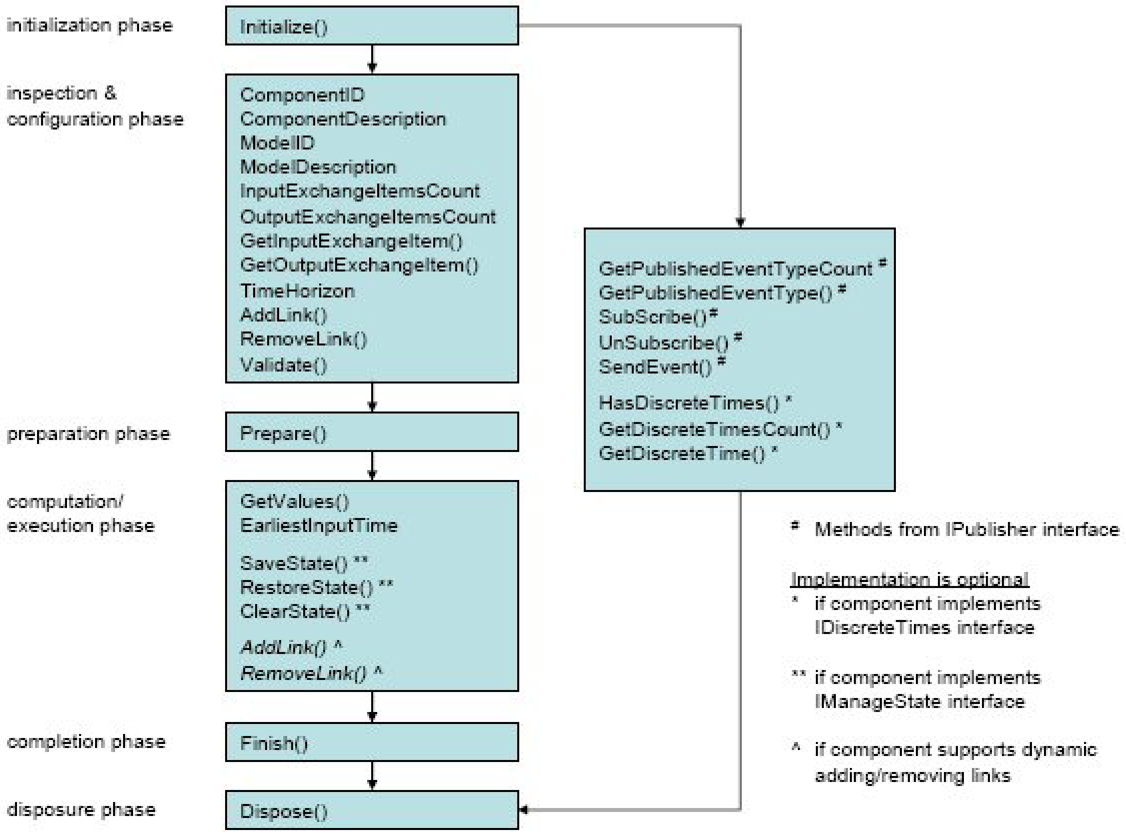


Figure 6-1 Required sequence for invoking methods in the Linkable Component [source: adopted from (Gijsbers, 2005)]

While these function calls will need to be implemented in the model, some of them also need to access the MLO. For example, in the inspection and configuration phase there are functions to get the exchange items or the number of exchange items. The MLO has a function that will get the number of values. A function must be added to get the exchange items. This function will be a combination of query functions that already exist in the MLO. The exchange item needs to know the locations where the data will be exchanged, which means the MLO will need to provide the necessary locations. The exchange item also describes the type of data being transferred, which the MLO can populate by querying the units of the mesh. The data operations interface needs to know the number of functions the model provides to reformat the data. The MLO must be able to send the number of unit conversions, spatial translations, and interpolations the model can perform.

The MLO will also need to be able to interact with the OpenMI interfaces. The OpenMI interface contains several interfaces that are composed of low level data types such as strings, integers, doubles, and Boolean values. These interfaces describe the data that are being exchanged. Interfaces should be defined for the element set, time, quantity, the value set, data operations, the link, and a linkable component. Other interfaces are optional.

OpenMI provides some packages that help to implement these necessary interfaces. The Backbone package provides the minimum interface classes. The Utilities namespace provides some wrapping functions to allow the model to interact with the interfaces. The Development Support namespace contains generic software to allow parsing of the XML files used in OpenMI. None of these packages are necessary to make a model OpenMI compliant, but they are provided to facilitate making the necessary model modifications.

6.3.1 Using the MLO with OpenMI

The MLO will be used to populate the OpenMI interfaces with the necessary data. The interfaces that are related to location are the element set and the data operations. The element set not only contains the locations of the nodes, but also describes the type of object that is storing the data, such as a point, line, or polygon. An id-based object can also be stored when geo-referencing, but is not needed. For two-dimensional meshes, the MLO will only need to differentiate between nodes and node strings. The data operations interface includes a list of functions that can be used to manipulate the data. This is where various methods of interpolation can be accessed.

Table 6-4 shows how the MLO properties are related to the properties in the OpenMI interfaces. It should be noted that some of the properties provided by the MLO are unnecessary for OpenMI.

As seen in the table, OpenMI requires several properties not found in the MLO. Information related to face values is not provided because the MLO is designed for two-dimensional meshes. The version number in the element set is an optional value intended for meshes that can change over time (such as wave models). A function should be added to the MLO to obtain whether the values are scalar or vector.

Table 6-4 OpenMI interface properties related to MLO properties

OpenMI interface	Interface properties	MLO property or function
Element Set	• Element type • Element count • Version number • Element index • Element id • Vertex Count • Face Count • Face Vertex Indices • X Coordinate • Y Coordinate • Z Coordinate	• Type property • Get Num Nodes function • *No method available* • Get Node Index function • Get Node Id function • Get Num Nodes function • *No method available* • *No method available* • Get X function • Get Y function • Get Z function
Quantity	• Unit • Dimension • Scalar or vector	• Get Unit function • Get Unit function • *No method available*
Data Operations	• Id for each function • Number of functions	• Interpolation option • *No method available*

While these properties only contain the data necessary to describe the location for data transfer, there are other interfaces that must also be populated for OpenMI to allow information to be exchanged. The time parameter needs to be sent with the data that is transferred. XMDF stores time in Julian Date format or referenced from an arbitrary zero value. A data operation will need to be provided to convert this to the Modified Julian Date format, the default time format for OpenMI. The description of the values will also need to be provided for the Quantity interface. This includes describing the units as well as giving conversion and offset factors and reporting if the value is scalar or vector.

From these basic parameters, exchange items can be created which are necessary for the linkable components and the links to be populated. These interfaces all must contain data for an exchange to take place using the OpenMI interface. Descriptions of the required interfaces can be found in Appendix B.

6.3.2 Additional Linkage Requirements

In addition to creating an MLO a model must be compiled into a dynamic link library (dll) file instead of an executable file which would not be accessible by OpenMI. Additionally, the model must be wrapped to be able to access the OpenMI interfaces and implement the necessary functions. Further details regarding these steps can be found in the OpenMI section.

7 Summary and Conclusion

This research has reviewed the current state of the art in model integration systems, specifically relating to water modeling software. Each system has unique characteristics to address the complexities involved in model linkage. An overview of each system is given here, along with a summary of features that would need to be included in a Model Linkage Object in XMDF.

7.1 Summary

FRAMES exchanges data through input and output files, which means the file format is extremely important. In version 1.x all specifications for the simulation are located in the Primary Data Communication Files (PCDF). Version 2.x uses data dictionary files to define the information that is passed between models. Modules are linked in a process flow diagram using drag-and-drop capabilities. This allows for straightforward and simple implementation.

DIAS links models by connecting abstract descriptions of real world applications, such as a river, with an applicable model. The domain objects can be connected to other domain objects that they might need to interact with (e.g. a river domain object that needs to interact with a coast domain object). Because the domain objects don't contain any details of the models, they can be saved and reused in several different simulations with many different models.

OpenMI models, databases, and tools exchange information during run time. This is achieved by adopting the OpenMI interface, which facilitates data transfer. A model needing information requests a value set from another model or database for a specific time at a specific location. Information is passed directly between models without the use of intermediary files.

MMS is organized into pre-processing, model application, and post-processing functions. This separates the functions of preparing spatial and time-series data, model integration, and visualization of the results. Data is exchanged through internal databases that are set up when MMS is downloaded.

OMS is composed of the OMS Platform and OMS Modules. The platform is a skeleton to which users can add tools, while the modules provide templates for component development. The NetBeans application core provides management support, file system interfaces, event handling, data persistence service, and an automated update facility. Because OMS uses NetBeans Desktop Application Framework, OMS modules can be plugged into other NetBeans platforms.

WEAP does not link separate models together, but rather incorporates all the considerations of a complex systems into a single model. It. provides a holistic analysis of water management by applying supply, demand, water quality, and ecological considerations to the analysis.

HEC-WAT is designed to help coordinate project studies to provide greater consistency between multi-disciplinary teams. It does not link models together so that they run concurrently, but creates a spatial representation that all HEC models can use for analysis. It assists in managing the progress of each model and allows a simple way to consider different alternatives.

The SMS Steering Module is designed to facilitate sharing of data between two specific models, STWAVE and ADCIRC. The Steering Module accounts for the differences in the two models and automates the information transfer so that data are interpolated or extrapolated accurately.

The MLO consists of a set of grid cells or cell faces, or a set of mesh nodes, edges, elements, or 3D element faces. An MLO will include methods for automatically extracting data from solution data written in XMDF format and packaging the data in the format required by the model linkage system. Basic properties of the MLO include:

- An id for the MLO
- A description
- The type of location selected (e.g. point or element edge)
- Units
- Coordinate System
- An array of locations
- Associated data set name(s)
- Identifiers for each data operation

7.2 Conclusion

From an extensive study done of various model integration software, FRAMES and OpenMI have been selected as two systems that XMDF will design a model linkage object for. FRAMES is a framework that connects modules and allows them to exchange data through input and output files. It was selected as one of the model linking systems that the MLO would be designed for because the Army Corps of Engineers already use this framework in many of their simulations. OpenMI is an interface that allows models to connect and exchange information during run time. It is because of this capability and other advanced capabilities that the MLO was designed for OpenMI.

An MLO has been designed for two-dimensional meshes that will be compatible with FRAMES and OpenMI. A basic design was proposed that is applicable to a general case of a two-dimensional mesh. Functionality was then added for each specific software system. This pattern can be followed to allow the MLO to work with other model integration systems.

7.3 Future Work

From the research that has been done, a prototype can be created to test the design of the MLO. This will make certain it is a feasible design and that all reasonable options have been explored. More fine tuning of the design may be necessary as it is developed.

The two-dimensional mesh design is only the beginning to the capabilities that the MLO can be used for. The ideas and techniques from this research can be used to expand the MLO design to store three-dimensional geometry. Further work will also need to be done to allow the MLO to be used with two- and three-dimensional grids.

There were several model integration systems that were analyzed in the course of this research. Further study can be done to make a component of the MLO that will be compatible with these and other integrated systems as well.

References

ARAMS website. Provided by US Army Corps of Engineers, 2004. http://el.erdc.usace.army.mil/arams/. Accessed 11 July 2005.

Blind, M. and J.B Gregersen. "Towards an Open Modelling Interface (OpenMI): The HarmonIT Project," *IEMSS 2004 Conference*, 2004. Available from HarmonIT website. http://www.iemss.org/iemss2004/pdf/integratedmodelling/blintowa.pdf. Accessed 2 May 2005.

Decision and Information Sciences Division, Advanced Computer Applications Center. *The Dynamic Information Architecture System: A High Level Architecture for Modeling and Simulation*. Argonne, IL: Argonne National Laboratory, 1995. http://www.dis.anl.gov/DEEM/DIAS/diaswp.html. Accessed 13 January 2006.

Dunn, C.N., G.W. Brunner, and J. Harris. "Software Integration for Watershed Studies: Hydrologic Engineering Center's Watershed Analysis Tool (HEC-WAT)." In *EWRI 2005: Impacts of Global Climate Change: Proceedings of the 2005 World Water and Environmental Resources Congress Held in Anchorage, Alaska May 2005*.

Campbell, A.P. and J.R. Hummel. *The Dynamic Information Architecture System: An Advanced Simulation Framework for Military and Civilian Applications*. Argonne, IL: Argonne National Laboratory. http://www.dis.anl.gov/DIAS/papers/SCS/SCS.html. Accessed 23 August 2005.

Christiansen, J.H. "A Flexible Object-Based Framework for Modeling Complex Systems with Interacting Natural and Societal Processes." In *4th International Conference on Integrating GIS and Environmental Modeling (GIS/EM4): Problems, Prospects and Research Needs Held in Banff, Alberta, Canada September 2000*.

DIAS website. Provided by Argonne National Laboratory, Decision and Information Sciences Division. http://www.dis.anl.gov/DIAS/index.html. Accessed 12 July 2005.

Environmental Modeling Research Laboratory. *Generic Model Data Format (XMDF) Reference Manual*. Brigham Young University, Provo, UT, 2005. http://emrl.byu.edu/xmdf1/index.html. Accessed 1 May 2005.

Gelston, G., N. Ullah, and K. Garza. Introduction to FRAMES 2.0 website, 2005. http://mepas.pnl.gov/geniitutorial/chapter2/frames_intro.html. Accessed 20 June 2005.

Gelston, G. FRAMES 1.5 website, 2002. http://mepas.pnl.gov/FRAMESV1/index.html. Accessed16 June 2005.

Gijsbers, P., et al. "Part C – the org.OpenMI.Standard Interface Specification." *The OpenMI Document Series*, 2005. Available from OpenMI website. http://www.openmi.org/content/blogsection/6/35/. Accessed13 March 2006.

Hoopes, B.L., M.A. Pelton, and K.J. Castleton. *Documentation for the Dictionary Editor of the FRAMEwork System (FRAMES)*. Richland, WA: Battelle Pacific Northwest Division, 2005.

Hoopes, B.L., M.A. Pelton, and K.J. Castleton. *Documentation for the Module Editor of the FRAMEwork System (FRAMES)*. Richland, WA: Battelle Pacific Northwest Division, 2005.

Hutchings, C., J. Struve, S. Westen, K. Millard, and D. Fortune. "State of the Art Review." Available from OpenMI website, 2002. http://www.harmonit.org/docs/repu_01_09_soa_review_approved.pdf. Accessed 1 July 2005.

Jones, N.L., R.D. Jones, C.D. Butler, and R.M. Wallace. "A Generic Format for Multi-Dimensional Models." *Proceedings of the Groundwater Symposium: EWRI 2004 World Water and Environmental Resources Congress Held in Salt Lake City, UT 27 June – 1 July 2004.*

Leavesley, G.H., P.J. Restrepo, S.L. Markstrom, M. Dixon, and L.G. Stannard. *The Modular Modeling System (MMS): User's Manual*. Available from MMS website,1998. http://wwwbrr.cr.usgs.gov/mms/html/mms_paper.html. Accessed 13 July 2005.

MIMS website. Provided by US Environmental Protection Agency, 2005. http://www.epa.gov/asmdnerl/Multimedia/MIMS/. Accessed 12 July 2005.

Olaf, D. and F. Geter. *Object Modeling System User Manual*. Available from Fort Collins, CO: Great Plains System Research, 2006. http://oms.ars.usda.gov/manual/ch01.html. Accessed 27 July 2005.

Sieber, J., et al. *WEAP User Guide for WEAP21*. Boston, MA : Stockholm Environment Institute, Tellus Institute, 2005.

Tindall, I., et al. "Part B – Guidelines for the OpenMI (version1.0)," *OpenMI Document Series*. Available from HarmonIT website, 2005. http://www.openmi.org/content/blogsection/6/35/. Accessed 13 March 2006.

US Environmental Protection Agency. *Multimedia Integrated Modeling Systems User Guide*. Available from Raleigh, NC: Science Application International Corps, 2004. http://www.epa.gov/asmdnerl/Multimedia/MIMS/MIMSdownload.html. Accessed 12 July 2005.

WEAP website. Provided by Stockholm Environmental Institute's Boston Center, 2005. http://www.weap21.org/index.asp?doc=01. Accessed: 19 July 2005.

Whelan, G., et al. *Concepts of a Framework for Risk Analysis in Multimedia Environmental Systems*. Available from Richland, WA: Pacific Northwest National Laboratory, 1997. http://mepas.pnl.gov/FRAMESV1/frames_doc.pdf. Accessed 4 May 2005.

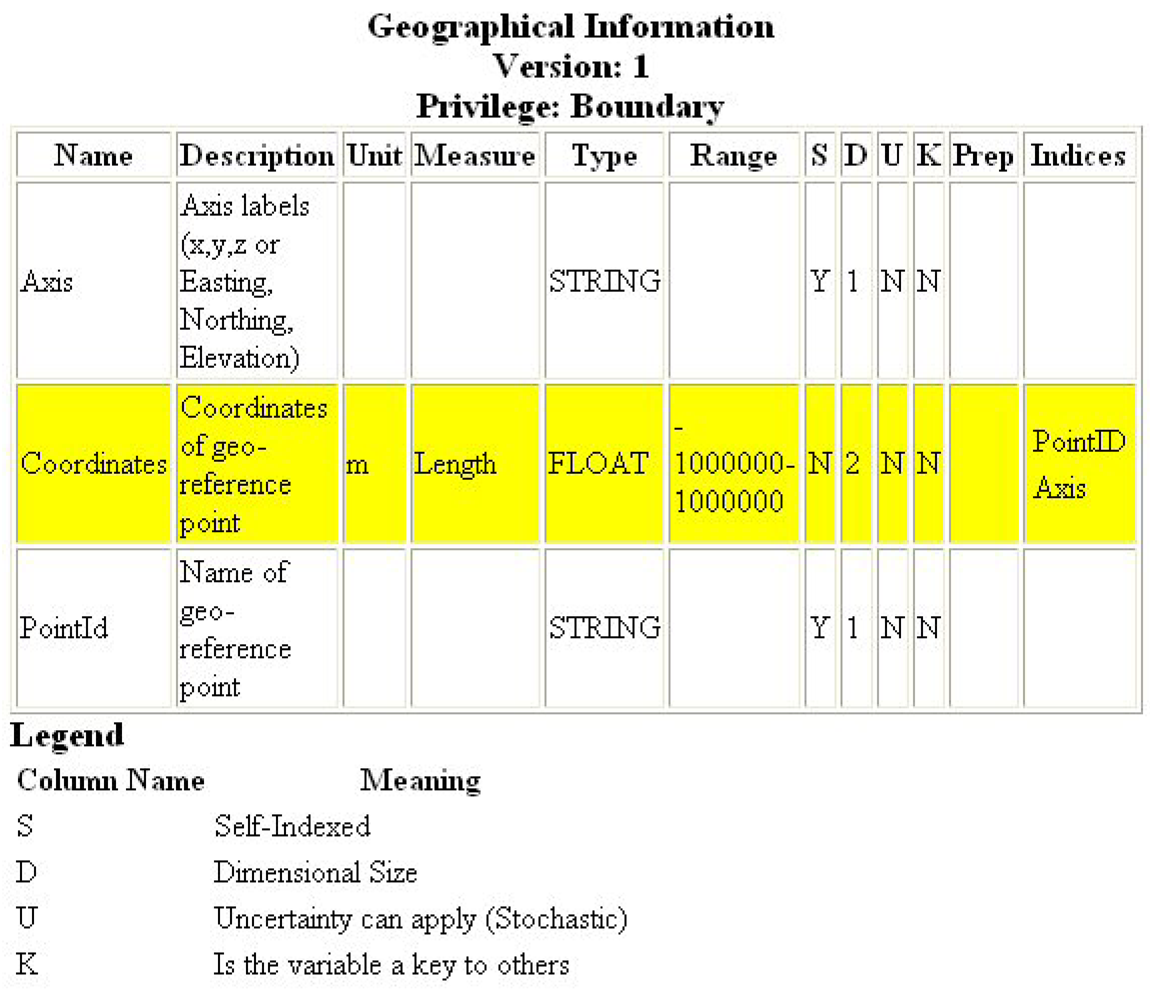
Whelan, G., and K.J. Castleton. *Examining the Linkage Between FRAMES and GMS*. Richland, WA: Pacific Northwest National Laboratory, 2006.

Whelan, G., and K.J. Castleton. *Groundwater Modeling System Linkage with the Framework for Risk Analysis in Multimedia Environmental Systems*. Richland, WA: Pacific Northwest National Laboratory, 2006.

Zundel, A. K., M. A. Cialone, and T. J. Moreland. "SMS steering module for coupling waves and currents, 1. ADCIRC and STWAVE," *Coastal and Hydraulics Engineering Technical Note ERDC/CHL CHETN-IV-41*. Vicksburg, MS: U.S. Army Engineer Research and Development Center, 2002.

Appendix A. Frames Data Dictionaries

These dictionary descriptions contain information on the location data that is required in exchanging data between models (Hoopes, 2005).



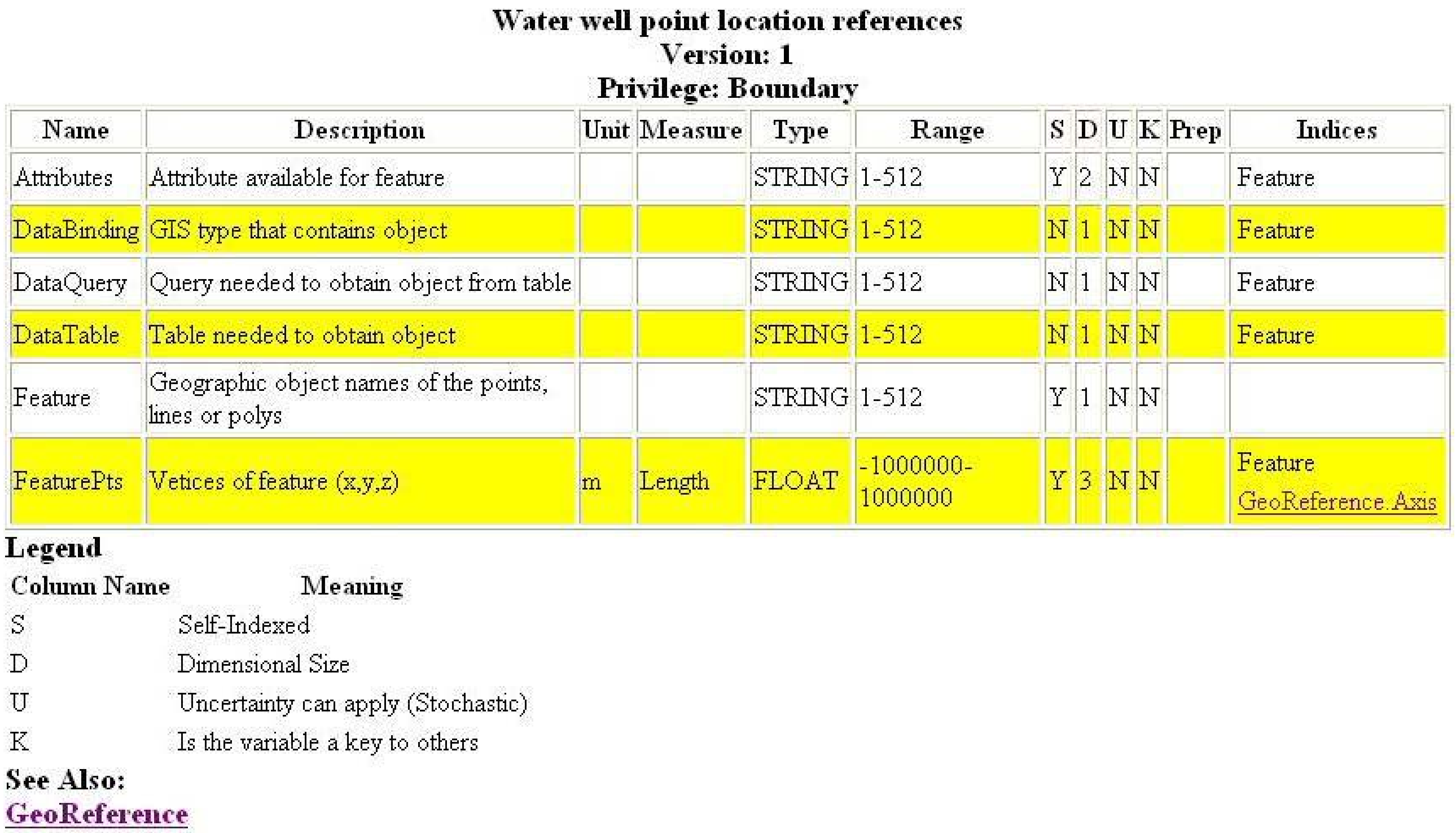
Geographical Information
Version: 1
Privilege: Boundary

Name	Description	Unit	Measure	Type	Range	S	D	U	K	Prep	Indices
Axis	Axis labels (x,y,z or Easting, Northing, Elevation)			STRING		Y	1	N	N		
Coordinates	Coordinates of geo-reference point	m	Length	FLOAT	-1000000-1000000	N	2	N	N		PointID Axis
PointId	Name of geo-reference point			STRING		Y	1	N	N		

Legend

Column Name	Meaning
S	Self-Indexed
D	Dimensional Size
U	Uncertainty can apply (Stochastic)
K	Is the variable a key to others

A- 1 GeoReference Dictionary



Water well point location references
Version: 1
Privilege: Boundary

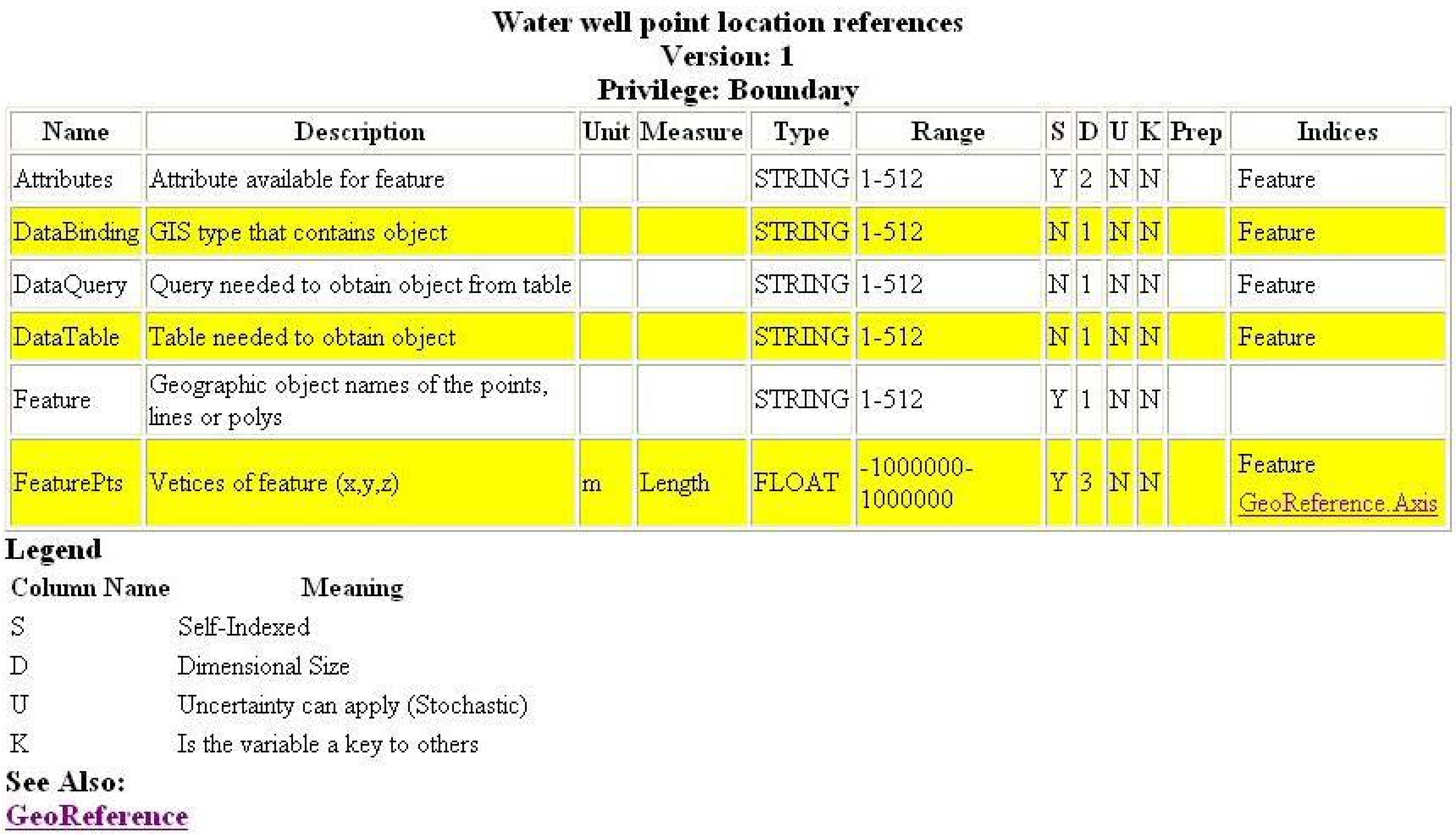
Name	Description	Unit	Measure	Type	Range	S	D	U	K	Prep	Indices
Attributes	Attribute available for feature			STRING	1-512	Y	2	N	N		Feature
DataBinding	GIS type that contains object			STRING	1-512	N	1	N	N		Feature
DataQuery	Query needed to obtain object from table			STRING	1-512	N	1	N	N		Feature
DataTable	Table needed to obtain object			STRING	1-512	N	1	N	N		Feature
Feature	Geographic object names of the points, lines or polys			STRING	1-512	Y	1	N	N		
FeaturePts	Vetices of feature (x,y,z)	m	Length	FLOAT	-1000000-1000000	Y	3	N	N		Feature GeoReference.Axis

Legend

Column Name	Meaning
S	Self-Indexed
D	Dimensional Size
U	Uncertainty can apply (Stochastic)
K	Is the variable a key to others

See Also:
GeoReference

A- 2 Aquifer Points Dictionary



Water well point location references
Version: 1
Privilege: Boundary

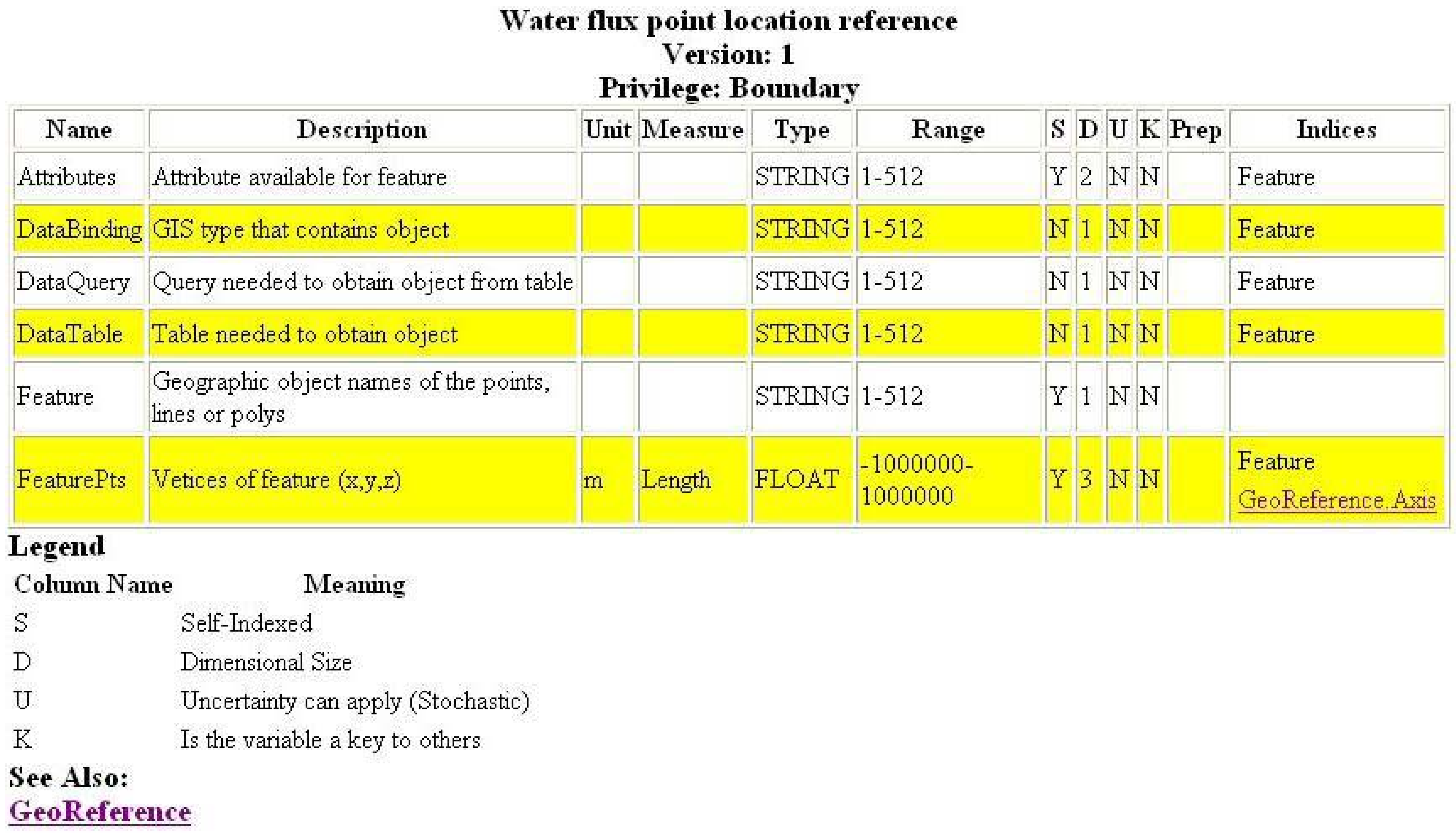
Name	Description	Unit	Measure	Type	Range	S	D	U	K	Prep	Indices
Attributes	Attribute available for feature			STRING	1-512	Y	2	N	N		Feature
DataBinding	GIS type that contains object			STRING	1-512	N	1	N	N		Feature
DataQuery	Query needed to obtain object from table			STRING	1-512	N	1	N	N		Feature
DataTable	Table needed to obtain object			STRING	1-512	N	1	N	N		Feature
Feature	Geographic object names of the points, lines or polys			STRING	1-512	Y	1	N	N		
FeaturePts	Vetices of feature (x,y,z)	m	Length	FLOAT	-1000000-1000000	Y	3	N	N		Feature GeoReference.Axis

Legend

Column Name	Meaning
S	Self-Indexed
D	Dimensional Size
U	Uncertainty can apply (Stochastic)
K	Is the variable a key to others

See Also:
GeoReference

A- 3 Surface Water Point Dictionarv



Water flux point location reference
Version: 1
Privilege: Boundary

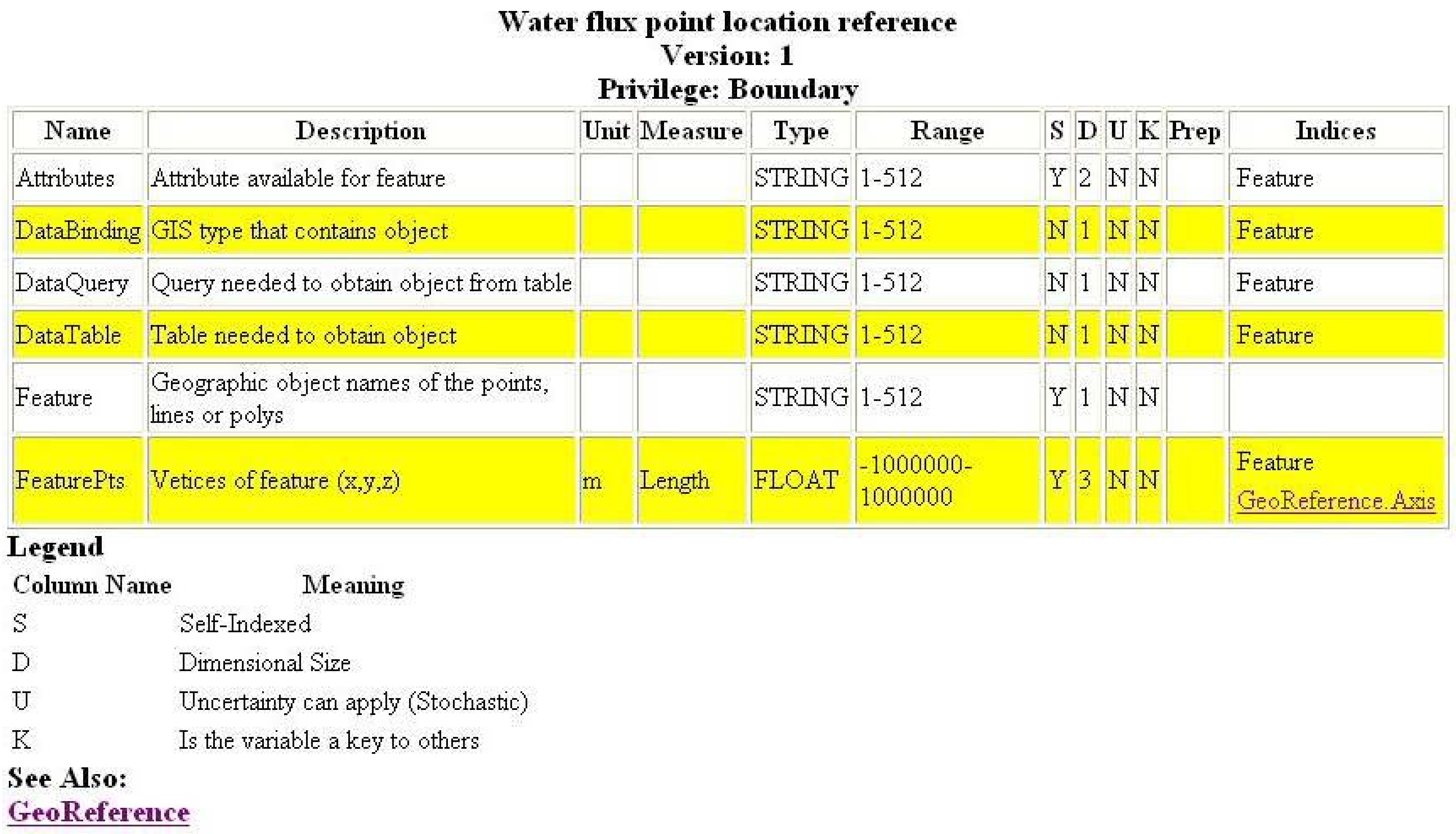
Name	Description	Unit	Measure	Type	Range	S	D	U	K	Prep	Indices
Attributes	Attribute available for feature			STRING	1-512	Y	2	N	N		Feature
DataBinding	GIS type that contains object			STRING	1-512	N	1	N	N		Feature
DataQuery	Query needed to obtain object from table			STRING	1-512	N	1	N	N		Feature
DataTable	Table needed to obtain object			STRING	1-512	N	1	N	N		Feature
Feature	Geographic object names of the points, lines or polys			STRING	1-512	Y	1	N	N		
FeaturePts	Vetices of feature (x,y,z)	m	Length	FLOAT	-1000000-1000000	Y	3	N	N		Feature GeoReference.Axis

Legend

Column Name	Meaning
S	Self-Indexed
D	Dimensional Size
U	Uncertainty can apply (Stochastic)
K	Is the variable a key to others

See Also:
GeoReference

A- 4 Aquifer Polygons Dictionary



Water flux point location reference
Version: 1
Privilege: Boundary

Name	Description	Unit	Measure	Type	Range	S	D	U	K	Prep	Indices
Attributes	Attribute available for feature			STRING	1-512	Y	2	N	N		Feature
DataBinding	GIS type that contains object			STRING	1-512	N	1	N	N		Feature
DataQuery	Query needed to obtain object from table			STRING	1-512	N	1	N	N		Feature
DataTable	Table needed to obtain object			STRING	1-512	N	1	N	N		Feature
Feature	Geographic object names of the points, lines or polys			STRING	1-512	Y	1	N	N		
FeaturePts	Vetices of feature (x,y,z)	m	Length	FLOAT	-1000000-1000000	Y	3	N	N		Feature GeoReference.Axis

Legend

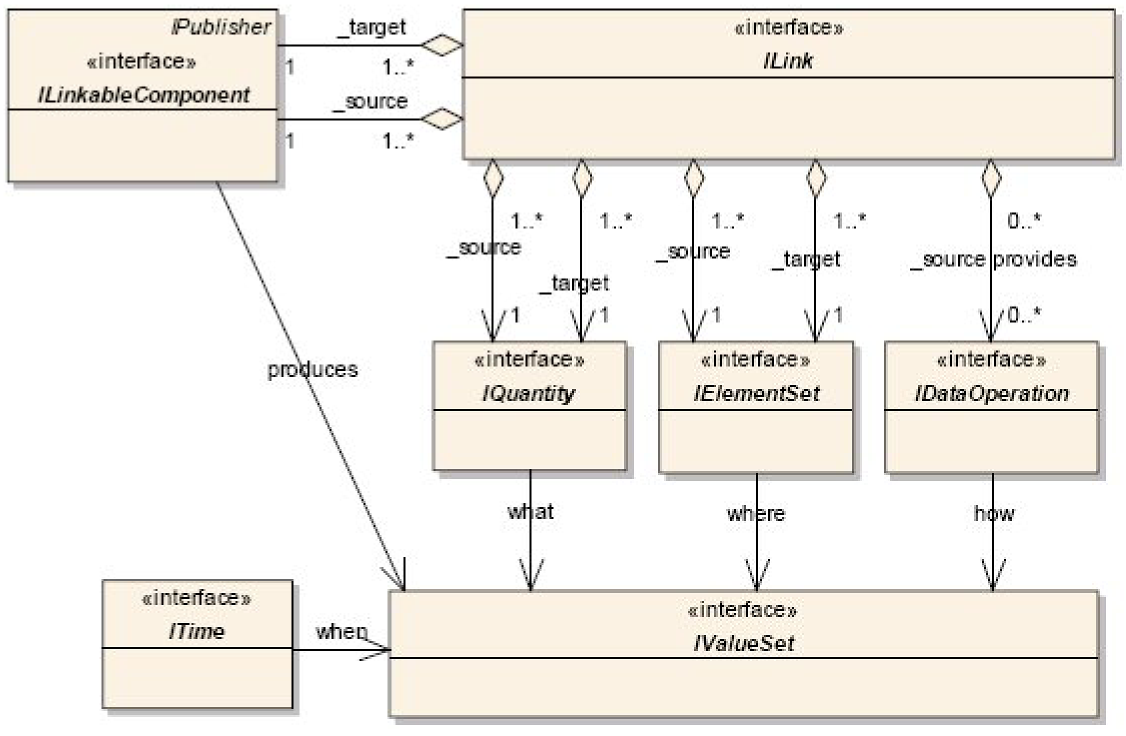
Column Name	Meaning
S	Self-Indexed
D	Dimensional Size
U	Uncertainty can apply (Stochastic)
K	Is the variable a key to others

See Also:
GeoReference

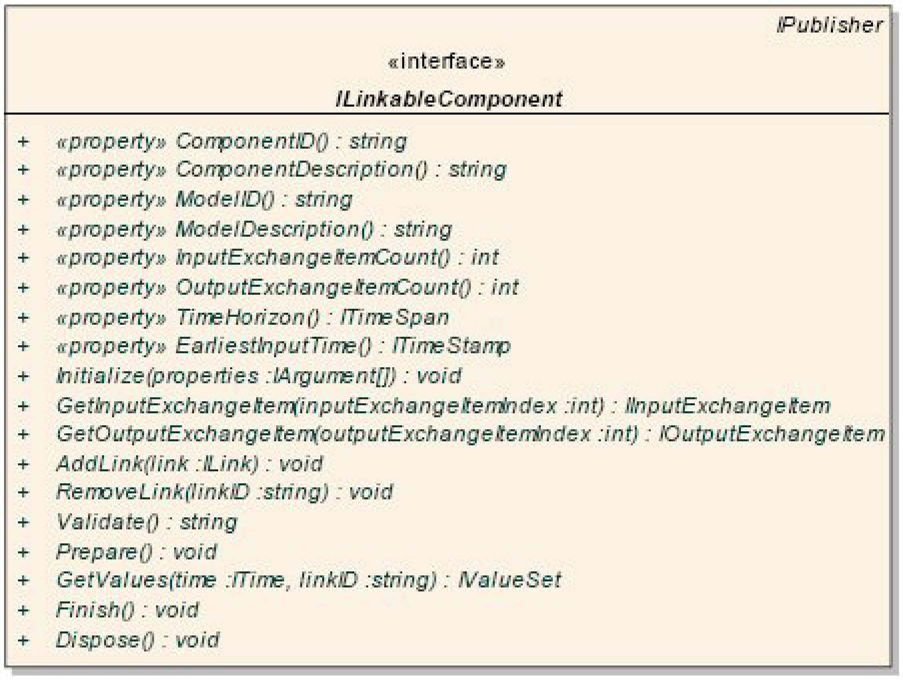
A- 5 Surface Water Polygons Dictionary

Appendix B. OpenMI Interface Definitions

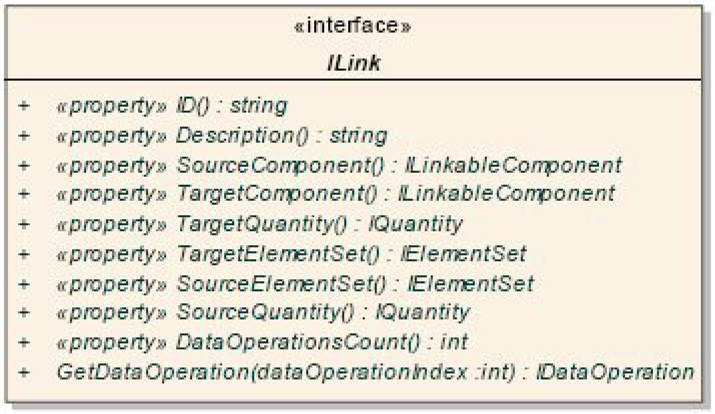
The figures in this appendix were taken from Part C of the OpenMI Document Series (Gijsbers, 2005).



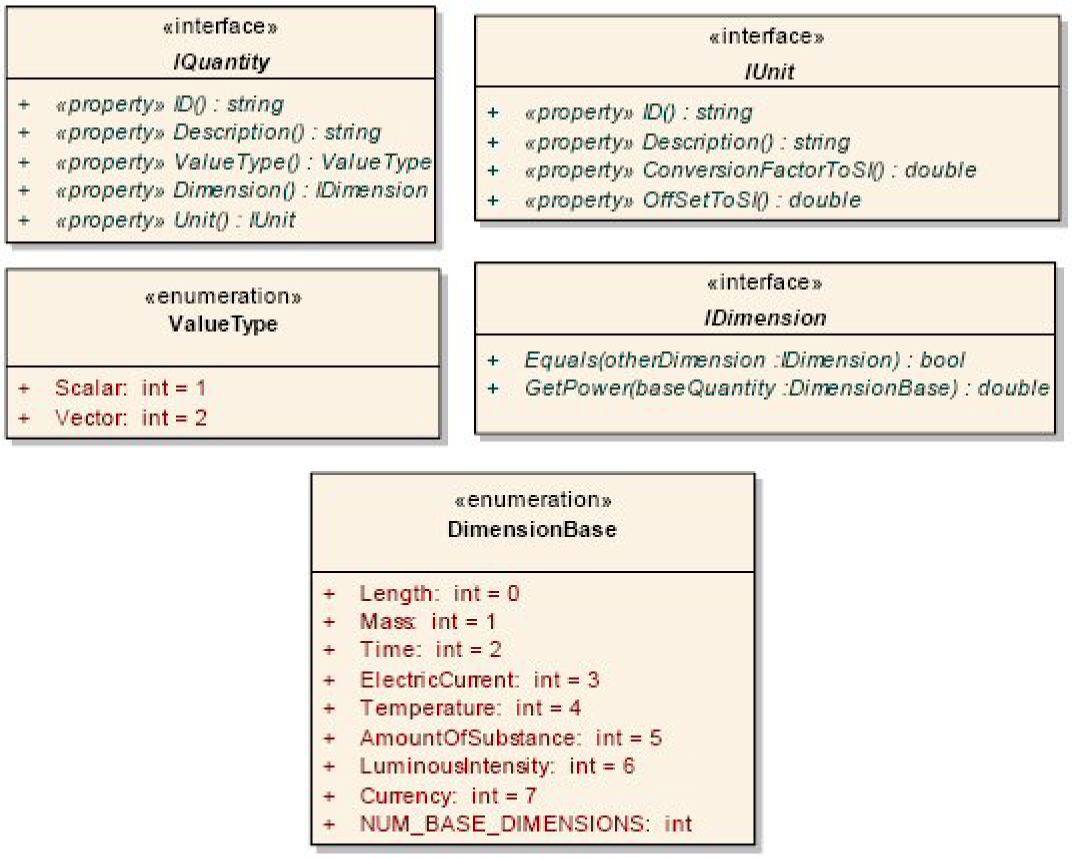
B- 1 Relationship between interfaces


IPublisher
«interface»
ILinkableComponent
+ «property» ComponentID() : string
+ «property» ComponentDescription() : string
+ «property» ModelID() : string
+ «property» ModelDescription() : string
+ «property» InputExchangeItemCount() : int
+ «property» OutputExchangeItemCount() : int
+ «property» TimeHorizon() : ITimeSpan
+ «property» EarliestInputTime() : ITimeStamp
+ Initialize(properties :IArgument[]) : void
+ GetInputExchangeItem(inputExchangeItemIndex :int) : IInputExchangeItem
+ GetOutputExchangeItem(outputExchangeItemIndex :int) : IOutputExchangeItem
+ AddLink(link :ILink) : void
+ RemoveLink(linkID :string) : void
+ Validate() : string
+ Prepare() : void
+ GetValues(time :ITime, linkID :string) : IValueSet
+ Finish() : void
+ Dispose() : void

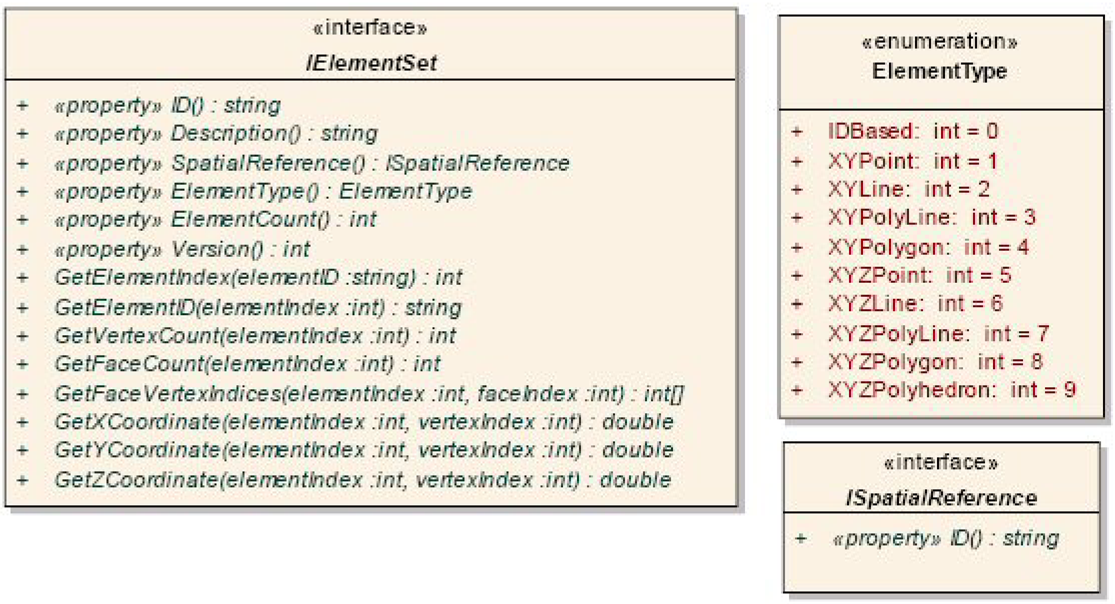
B- 2 Linkable Component interface


«interface»
ILink
+ «property» ID() : string
+ «property» Description() : string
+ «property» SourceComponent() : ILinkableComponent
+ «property» TargetComponent() : ILinkableComponent
+ «property» TargetQuantity() : IQuantity
+ «property» TargetElementSet() : IElementSet
+ «property» SourceElementSet() : IElementSet
+ «property» SourceQuantity() : IQuantity
+ «property» DataOperationsCount() : int
+ GetDataOperation(dataOperationIndex :int) : IDataOperation

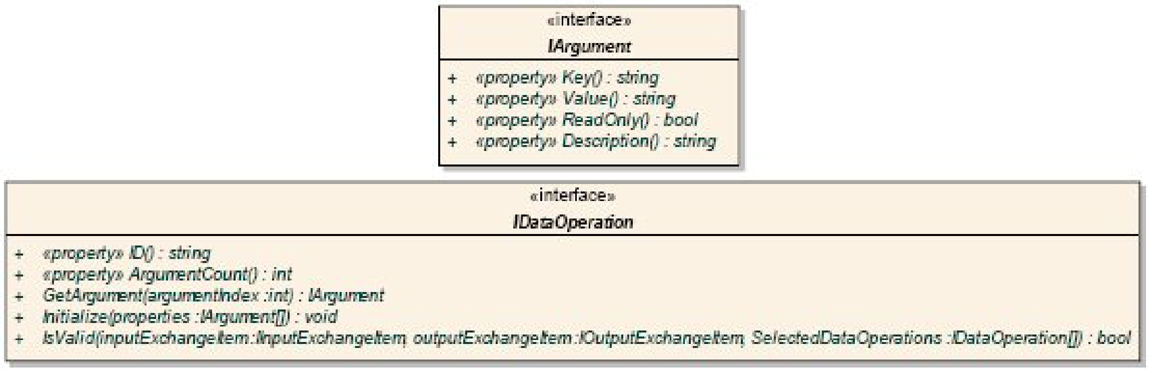
B- 3 Link interface


«interface»
IQuantity
+ «property» ID() : string
+ «property» Description() : string
+ «property» ValueType() : ValueType
+ «property» Dimension() : IDimension
+ «property» Unit() : IUnit
«interface»
IUnit
+ «property» ID() : string
+ «property» Description() : string
+ «property» ConversionFactorToSI() : double
+ «property» OffSetToSI() : double
«enumeration»
ValueType
+ Scalar: int = 1
+ Vector: int = 2
«interface»
IDimension
+ Equals(otherDimension :IDimension) : bool
+ GetPower(baseQuantity :DimensionBase) : double
«enumeration»
DimensionBase
+ Length: int = 0
+ Mass: int = 1
+ Time: int = 2
+ ElectricCurrent: int = 3
+ Temperature: int = 4
+ AmountOfSubstance: int = 5
+ LuminousIntensity: int = 6
+ Currency: int = 7
+ NUM_BASE_DIMENSIONS: int

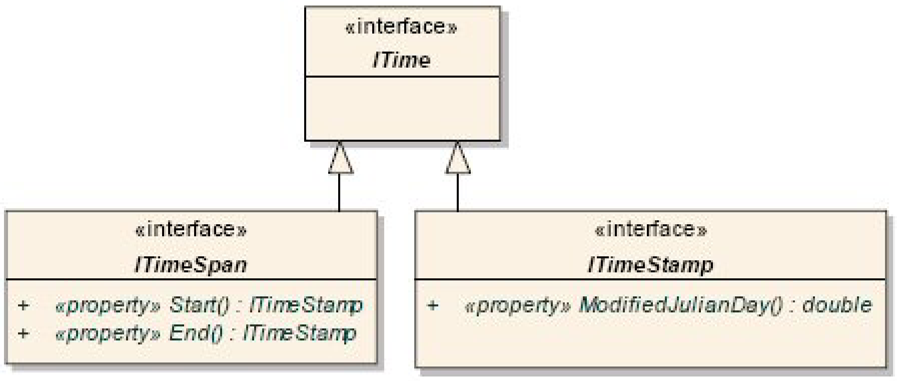
B- 4 Quantity interface


«interface»
IElementSet
+ «property» ID() : string
+ «property» Description() : string
+ «property» SpatialReference() : ISpatialReference
+ «property» ElementType() : ElementType
+ «property» ElementCount() : int
+ «property» Version() : int
+ GetElementIndex(elementID :string) : int
+ GetElementID(elementIndex :int) : string
+ GetVertexCount(elementIndex :int) : int
+ GetFaceCount(elementIndex :int) : int
+ GetFaceVertexIndices(elementIndex :int, faceIndex :int) : int[]
+ GetXCoordinate(elementIndex :int, vertexIndex :int) : double
+ GetYCoordinate(elementIndex :int, vertexIndex :int) : double
+ GetZCoordinate(elementIndex :int, vertexIndex :int) : double
«enumeration»
ElementType
+ IDBased: int = 0
+ XYPoint: int = 1
+ XYLine: int = 2
+ XYPolyLine: int = 3
+ XYPolygon: int = 4
+ XYZPoint: int = 5
+ XYZLine: int = 6
+ XYZPolyLine: int = 7
+ XYZPolygon: int = 8
+ XYZPolyhedron: int = 9
«interface»
ISpatialReference
+ «property» ID() : string

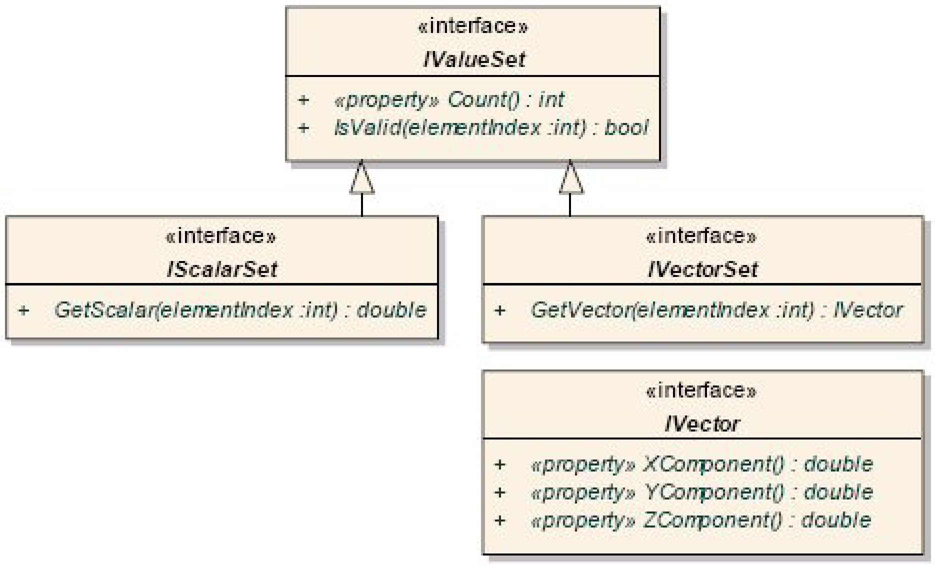
B- 5 Element Set interface


«interface»
IArgument
+ «property» Key() : string
+ «property» Value() : string
+ «property» ReadOnly() : bool
+ «property» Description() : string
«interface»
IDataOperation
+ «property» ID() : string
+ «property» ArgumentCount() : int
+ GetArgument(argumentIndex :int) : IArgument
+ Initialize(properties :IArgument[]) : void
+ IsValid(inputExchangeItem :IInputExchangeItem, outputExchangeItem :IOutputExchangeItem, SelectedDataOperations :IDataOperation[]) : bool

B- 6 Data Operations interface


«interface»
ITime
«interface»
ITimeSpan
+ «property» Start() : ITimeStamp
+ «property» End() : ITimeStamp
«interface»
ITimeStamp
+ «property» ModifiedJulianDay() : double

B- 7 Time interface


«interface»
IValueSet
+ «property» Count() : int
+ IsValid(elementIndex :int) : bool
«interface»
IScalarSet
+ GetScalar(elementIndex :int) : double
«interface»
IVectorSet
+ GetVector(elementIndex :int) : IVector
«interface»
IVector
+ «property» XComponent() : double
+ «property» YComponent() : double
+ «property» ZComponent() : double

B- 8 Value Set interface