



The OpenMI Document Series

The OpenMI ‘in a Nutshell’

For the OpenMI (Version 2.0)



Grant agreement number LIFE06 ENV/UK/000409



Title	OpenMI Document Series: The OpenMI 'in a Nutshell' for the OpenMI (Version 2.0)
Editor	Roger Moore, Centre for Ecology and Hydrology, Wallingford, UK
Authors	The OpenMI Association Technical Committee (OATC)
Current version	V 1.0
Date	30/11/2010
Status	Final © The OpenMI Association
Copyright	All methodologies, ideas and proposals in this document are the copyright of the OpenMI Association. These methodologies, ideas and proposals may not be used to change or improve the specification of any project to which this document relates, to modify an existing project or to initiate a new project, without first obtaining written approval from those of the OpenMI-LIFE participants who own the particular methodologies, ideas and proposals involved.

Preface

This document, based on an earlier paper¹, contains a brief overview of the OpenMI, a software standard that facilitates the linking of software-based applications to realise integrated modelling.

The OpenMI was originally developed for integrated water resource management, but its application has been extended to other domains of environmental management, such as agriculture and economics.

This document is part of the OpenMI report series, which specifies the OpenMI interface standard, provides guidelines on its use and describes software facilities for migrating, setting up and running linked models.

Titles in the series include:

- Scope
- **The OpenMI 'in a Nutshell'** (this document)
- OpenMI Standard 2 Reference
- OpenMI Standard 2 Specification

The OpenMI is maintained by the OpenMI Association and this document, along with other more detailed documentation, can be obtained from www.openmi.org.

The official reference to this document is:

The OpenMI Association (2010) *The OpenMI 'in a Nutshell' for the OpenMI (Version 2.0)*. Part of the OpenMI Document Series

Disclaimer

The information in this document is made available on the condition that the user accepts responsibility for checking that it is correct and that it is fit for the purpose to which it is applied.

The OpenMI Association will not accept any responsibility for damage arising from actions based upon the information in this document.

Acknowledgement

The OpenMI Association's members would like to acknowledge the contribution of the European Commission in co-funding the HarmonIT and OpenMI-LIFE projects. In particular, we would like to thank the Commission's staff for their sustained encouragement and support over many years.

Further information

Further information on OpenMI-LIFE can be found on the project website, www.OpenMI-Life.org.

Information on the OpenMI Association and the Open Modelling Interface (OpenMI) can be found on www.openmi.org.

Contents

1	Introduction	7
2	Standard.....	9
2.1	Definitions.....	10
2.1.1	Model.....	10
2.1.2	Value	11
2.1.3	Spatial	11
2.2	Composition building.....	12
2.3	Run-time.....	13
2.3.1	Pull driven.....	13
2.3.2	Loop driven.....	14
2.4	Compliance	14
3	Tools and utilities.....	16
4	Chronology	18
5	References.....	19

1 Introduction

The OpenMI is primarily concerned with integrated software systems. These systems can only be developed and maintained if they are based on a collection of dynamically interlinked applications. So the OpenMI Association has developed a unified approach to linking applications, which it promotes as the OpenMI Standard. The standard is intended for use with both legacy and new applications and has been widely adopted.

Large software packages typically consist of a number of discrete phases (Figure 1a), the most typical being:

- Generation of input file(s) (often via a User Interface)
- Computation
- Generation of output file(s) (results)

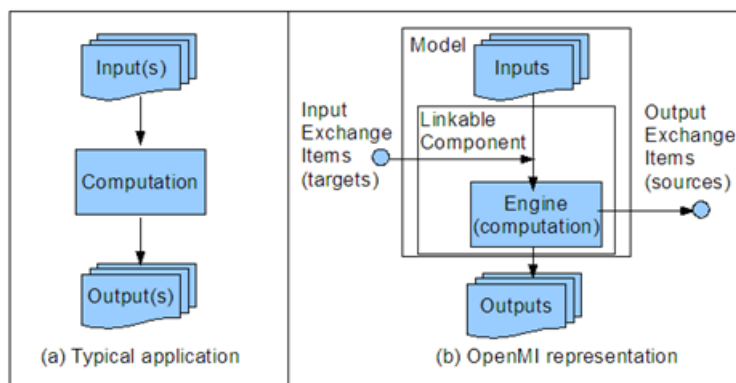


Figure 1 Terminology

The OpenMI focuses on the computation phase of the application, and the code base that performs that computation is referred to as the *engine*. The OpenMI also defines a *linkable component* as an engine that has been adapted to provide dynamic data exchange. Dynamic data is transferred between components by the linking of *output exchange items* and *input exchange items* (sometimes also referred to as *targets* and *sources* for brevity). A *model* is a specific instance of a component that also includes input data (Figure 1b). For instance, a ground water model would be a ground water component with input data for a specific geographic region of interest. This input data can often be considered as default data, since the act of linking the model could result in the overriding of this data with dynamic values from another source. When two or more models are linked together they are considered to be in a *composition*.

This approach breaks the pattern of importing all input data before computation and not writing any outputs until computation has been completed. To reiterate, an OpenMI model will:

- Only request inputs immediately prior to their use in the engine
- Provide outputs immediately after they have been computed in the engine

In order to achieve this, the application needs to be turned into an engine component that implements this dynamic data transfer in a way that other engines can also recognize and utilize.

The OpenMI addresses these issues by defining a set of software interfaces (the *OpenMI Standard*), which, if implemented around the engine, will allow other similarly modified engines to transfer data dynamically between each other. If an engine implements this standard in an approved way (i.e. as recognized by the Association) the application is deemed to be *OpenMI compliant*. Users of a compliant model will know that they can perform dynamic data exchange with that engine. However, they will not necessarily know whether the data exchange is of a type or form that would be useful to their particular interests. Hence, an additional task of the OpenMI is to define a uniform way of describing the available targets and sources.

The standard is intended to enable on-line (memory based) data exchange between models on a time basis (e.g. a time stamp or a time span). Given that different models will be running on different temporal and spatial domains, the mechanism for data conversions between domains also needs to be prescribed. Consequently, the standard must not only specify definitions, but also provide guidelines as to how those definitions are implemented.

The standard is not limited to connecting time-based engines. Version 2 of the standard has specifically concentrated on simplifying the mixing of temporal and non-temporal models (e.g. databases) within a single composition.

2 Standard

The standard is essentially defined by a set of interfaces that a component must implement. These are described using Unified Modeling Language (UML) diagrams (Figure 2).

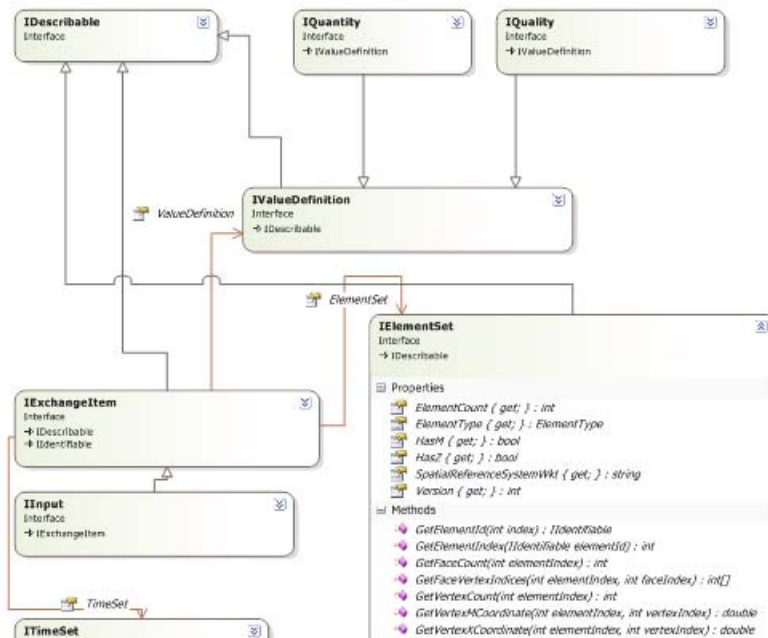


Figure 2 Example of OpenMI UML diagrams

Additional implementation requirements are provided as embedded comments within the interface definitions (Figure 3). Currently the OpenMI Association provides implementations for two software frameworks, Microsoft .NET² and SUN Java³. To be OpenMI-compliant, the component must implement at least one of these frameworks.

It should also be stressed that this does not preclude the use of engines written in procedural languages such as C or FORTRAN (or indeed other languages such as C++, F#, Matlab etc.) However, these engines must use intermediate code (*wrappers*) that provide the bridge between themselves and their chosen framework. Many examples exist that do this, and the wrappers are often made openly available for general use, frequently in the form of code libraries (often referred to as *Software Development Kits* or *SDKs*).

To reiterate, the OpenMI Standard is defined to be framework neutral; it is only specific implementations that might focus on particular frameworks and platforms. The Association actively promotes the uptake and use of the OpenMI with different programming languages, frameworks and platforms.

The screenshot displays a Java IDE window for the `org.openmi.standard` package. The main focus is the `ILinkableComponent` interface. The interface is defined as follows:

```

public interface ILinkableComponent
    extends IIdentifiable, IDescribable

The ILinkableComponent is the key interface in
the OpenMI standard. Any OpenMI compliant
component must implement ILinkableComponent.
OpenMI compliance definition:
§ 1) An OpenMI compliant component i
mplements the ILinkableComponent interface accor
ding to the specifications provided as comments in I
LinkableComponent.
§ 2) An OpenMI compliant component i
mplements the ILinkableComponent interface accor
ding to the specifications provided as comments in I
LinkableComponent.
    
```

The interface includes several methods: `Prepare()`, `Update(params OpenMI.Standard2.IK...)`, `Validate()`, `AdaptedOutputFactories`, and `Arguments`. The IDE also shows a class tree on the left with `ILinkableComponent` selected, and a summary of the interface on the right.

Figure 3 Implementation guidelines

2.1 Definitions

The standard interfaces prescribe specific requirements as to how exchange items should be defined. This allows users, whether using the interfaces directly or browsing indirectly via a Graphical User Interface (GUI), to get a consistent view on what is available from different components.

This is important as users need to be able to apply engineering judgement as to what is compatible: for example, whether a source exchange item can be linked with a target exchange item, from a different component, in a way that is meaningful from both physical and modelling viewpoints.

2.1.1 Model

To allow an OpenMI component to be usable by other components (i.e. be linkable), it is necessary that it can be found and its OpenMI-specific information extracted. The standard requires that a compliant component must provide metadata about itself. This consists of:

- How to locate a code library that implements the OpenMI Standard `ILinkableComponent` interface.
- The type name of a class within that code library that can be used to create an instance of the `ILinkableComponent` interface.
- Any additional values required for the execution of the model via that `ILinkableComponent` instance.

The metadata must be provided as XML. This XML must validate against a specific OpenMI-prescribed XSD schema. A model is *not* compliant if it cannot be successfully validated against this schema. Typically this XML is provided in a file (known as an *OMI file*); this however is not a requirement for compliance.

2.1.2 Value

Value definitions are subdivided into Quantities and Qualities:

- Quantities

For values that can be expressed using SI units, e.g. flow in m³/s

- Qualities

For values that are qualitative, e.g. bird population referred to by categories 'Safe', 'Endangered' or 'Critical'

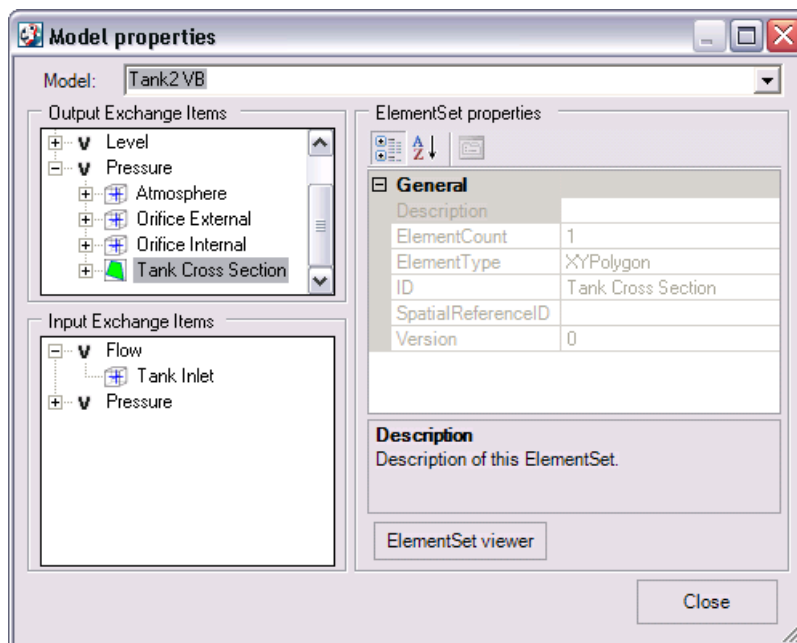


Figure 4 GUI representation of value definitions

2.1.3 Spatial

Typically values will be associated with a spatial domain. When this is not the case the transferred values are considered to be *ID based*. Spatial domains might be based on points (rain gauge locations), polylines (river centre lines), polygons (catchment) or polyhedrons (porous rock strata). The OpenMI defines an *element set* to encapsulate the spatial representation. An element set can contain one or many elements but must not mix element types, e.g. it cannot contain both points and polylines together.

An element set can be interrogated to obtain the defining co-ordinates, which may be plotted via a GUI to assist the user in making linkage decisions when building a composition. Co-ordinates can be geo-referenced against a specific GIS 'well known type' definition.

An element set could be a vast structure, say all the cells in a computation grid. For reasons of efficiency, the standard therefore does not explicitly define an element. Specific element information is extracted indirectly via the element set interface.

2.2 Composition building

When a user links two or more models together they are creating a *composition*. Linkages are created by connecting output exchange items (sources) from one model to input exchange items (targets) on another (Figure 5a).

An exchange item is defined by the pairing of a value definition with a spatial definition.

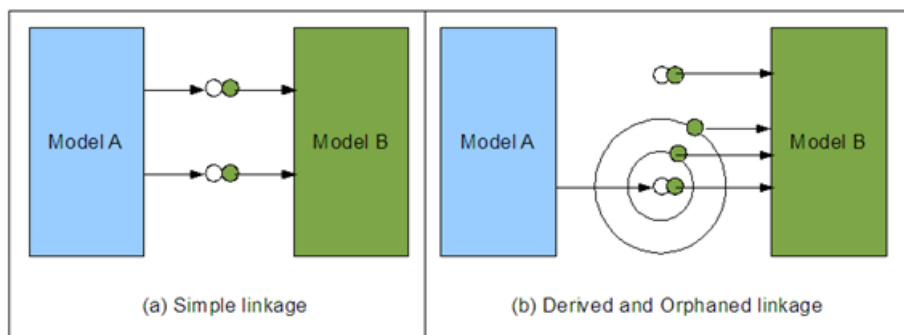


Figure 5 A composition with linkage

The onus is on the user who links models together to ensure that the linkages are physically and numerically meaningful. Clearly in many instances the sources made visible on one component will not be compatible with the targets on another. For example:

- A river model with a leakage source on a polyline element set representing the river centre line
- A ground model with an inflow target on a polygon element set representing the 2D surface of the computational grid

For this reason, the OpenMI allows for the generation of *derived* sources. A derived source is also an output exchange item; it takes the existing exchange item and modifies it so that it can be connected to a different target (Figure 5b). Derived exchange items can be provided either by the source component or by a third-party source library. For example the *OpenMI Association Technical Committee (OATC)* provides an example implementation of a third-party library of simple conversions between different element set types.

The main advantages of derived sources are:

- Derived sources that involve considerable computational expense can be repeatably used on multiple targets without duplication of that expense.
- Exchange only occurs between something that is a source to something that is a target. Therefore no additional connection mechanism has to be defined; i.e. there is no need for the OpenMI to define a link interface.

The standard also allows *orphaned sources*. An orphaned source has no providing model attached to it. This is specifically useful for linking with simple data sources that involve no computation: e.g. data files, databases, web services. In these cases, the provider of the orphaned source has a much simpler implementation task as they do not have to implement a component.

2.3 Run-time

During run-time a composition must be driven to ensure the correctly timed exchange of data between models. The standard advocates the ‘pull driven’ approach, described below.

2.3.1 Pull driven

The pull driven approach is based on the ‘request & reply’ mechanism according to Buschmann et al. (1996)⁴. A component is defined as the primary driver of the composition; update is repeatedly called on this primary component until it reaches completion. On each update call, before the component performs its computation, it updates all its active targets until they can provide the required input data. If necessary, these targets will in turn call update on their connected source components and so on. This update and compute mechanism then propagates data around all the components in the composition in a sequential manner.

To avoid grid-lock – a scenario where two components require inputs from each other simultaneously – there is an additional requirement that if a component is requested for output which it cannot provide by computation; then it must still supply a value, even if it has to guess. Typically this results in an extrapolation from previously computed values. Apart from the primary component, other components will only progress their computation if values have been requested. So it is quite possible that at the end of a composition run, not all models have proceeded to the same final temporal moment.

This approach has the advantage that, whatever the composition, it will always progress to completion without any additional knowledge about the composition being solved. This makes it ideal for implementation by generic GUIs. Version 1 of the standard only focused on this mode of operation.

The primary disadvantage of this approach is that it is a purely synchronous process, i.e. single threaded. One component will not progress until another has finished. Note that this does not preclude a component utilising multi-threading techniques itself during its computation phases, merely that the inter-component communication is sequential. Figure 6 shows a simple three-component composition; model A cannot ask models B and C to progress simultaneously from time t1 to t2 even though they have no data interdependences. Model A must ask model B to progress, wait until it returns and then request model C to do likewise (or visa versa). Actually, for this simple example, a clever implementation could resolve this issue generically, but in real life, compositions rapidly become too complicated to be so optimized without additional user run-time input.

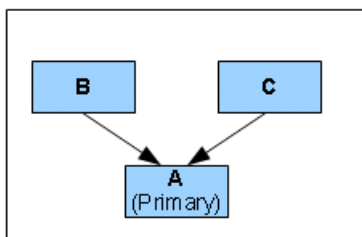


Figure 6 Sequential composition execution

2.3.2 Loop driven

The 'Pull driven' approach is not the only way of driving compositions. One other example would be 'Loop driven'.

This approach assumes the presence of an external controller with additional knowledge about the composition being solved. Therefore, currently, it is largely of benefit for programmers interacting directly with the interfaces rather than users running indirectly using a generic UI or one of the standard SDKs. It allows the controller to update components in any order and the controller must protect against and resolve any grid-lock situations.

Whilst Version 2 of the standard intends to make the loop approach usable, there is insufficient experience in its application for the Association to formally advocate it.

2.4 Compliance

As mentioned earlier, it is a condition of compliance that the implementer must provide metadata about the model. Apart from this model-specific information, additional general information is also required. The purpose of this additional metadata is to provide an overview of the component's purpose, usage and implementation. For compliance, a supplier should communicate this metadata as XML to the OpenMI Association. The Association will then review, and if accepted, publish the data on the Association's website to promote its availability. The metadata provides information such as:

- What the component models and its utility
- Who owns the component and its availability for usage by others
- Which exchange items the component exposes
- Which computational frameworks and platforms it implements

Home / Users Print

Compliant Software

About the listed components

In order to facilitate OpenMI users the OpenMI Association maintains a list of currently OpenMI compliant components. The OpenMI Association does not perform any testing of these components. Consequently, the model providers have the full responsibility for that these components work as they are supposed to, both in OpenMI configurations and as stand-alone components.

To submit a new OpenMI compliant model

Would you like your model/model component to be added to the list below ?
Please submit the OpenMI Standard compliancy XML file for your component to the [OpenMI Association Technical Committee](#).

Provider	Component	Description
British geological survey & University of Birmingham	ZOOMQ3D Compliant to: OpenMI Standard 1.4 .Net XML Info , HTML info	Finite-difference groundwater flow model
Wallingford Software Ltd	InfoWorks CS 10.0 Compliant to: OpenMI Standard 1.4 .Net XML Info , HTML info	Hydrological modeling for the urban water cycle
	InfoWorks RS 10.0 Compliant to: OpenMI Standard 1.4 .Net XML Info , HTML info	Flow simulation for rivers, channels and floodplains

Figure 7 Compliance metadata presentation on Association website

3 Tools and utilities

As part of the development of the standard, the *OpenMI Association Technical Committee (OATC)* developed a number of tools, libraries, utilities and unit tests that it used to implement and verify the standard. There is no requirement to use these; they are not part of the standard, and indeed the Association does not formally support them. However, they are made available as open source utilities to assist users in developing their implementations. The two primary utilities that are available are the SDK and Configuration Editor.

- OATC Software Development Kit (SDK)
A library of C# classes that provide implementations, examples and unit tests for the standard.
- OATC Configuration Editor
A simple editor that allows users to add, link and run OpenMI compositions.

These utilities can be obtained from the OpenMI SourceForge project site, using one of these two methods:

- <http://sourceforge.net/projects/openmi> (Figure 8)
- <https://openmi.svn.sourceforge.net/svnroot/openmi>

SourceForge.net > Find Software > OpenMI > Browse Files



OpenMI by adharper, hartnack, hrajagers, jan_gregersen, jgr-dhigroup, ...

Summary | **Files** | Support | Develop

The Open Modeling Interface and Environment (OpenMI) defines a standardized way to exchange data between computational models that run simultaneously. OpenMI aims to enhance process interaction representation in integrated environmental modelling.

Download Now!  OR [View all files](#) >

Oato.ConfigurationEditor_... (1.2 MB)

Browse Files for OpenMI














File/Folder Name	Platform	Size	Date ↓	Downloads	Notes/Subscribe
Newest Files					
 Oato.SDK_1.4.0.0.zip		4.8 MB	2007-12-20	1,124	
 Oato.ConfigurationEditor_1.4.0.0.msi		1.2 MB	2007-12-20	1,922	
All Files					
▶  SDK		4.8 MB	2007-12-20	1,124	 
▶  Configuration Editor		1.2 MB	2007-12-20	1,922	 
▶  Superseded (Deprecated)		16.4 MB	2006-08-30	1,967	 

Figure 8 <http://sourceforge.net/projects/openmi/files/>

4 Chronology

- 2001–2005

The OpenMI initially developed as part of the HarmonIT⁵ project, an EC co-funded research project.

Key milestones:

- Release of version 1 of the OpenMI standard

- 2005–2010

The OpenMI further developed as part of the OpenMI-Life⁶ project, an EC co-funded research project.

Key milestones:

- Formation of the OpenMI Association to protect the long-term future of the Standard.
- Version 1 of the standard tested on several large-scale real-world projects.
- Version 2 of the standard developed and submitted to peer review prior to formal ratification by the Association.

5 References

- ¹ Based on an original paper "The OpenMI Standard in a nutshell", Sept 2007 by Peter J.A. Gijssbers (WL | Delft Hydraulics) and Jan B. Gregersen (DHI Water & Environment)
- ² .NET Framework, msdn.microsoft.com/netframework
- ³ Java Software Platform, www.java.com
- ⁴ "A System of patterns", F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. (ISBN 0 471 95869 7) 1996.
- ⁵ HarmonIT research project co-funded by the European Commission (Contract no: EVK1-CT-2001-00090)
- ⁶ OpenMI-Life project as part of the EU-LIFE Environment programme (Contract no: LIFE06 ENV/UK/000409).