

The OpenMI Standard in a nutshell

Peter J.A. Gijssbers (WL | Delft Hydraulics)
Jan B. Gregersen (DHI Water & Environment)



This paper

This paper contains the summary of the org.OpenMI.Standard, the Open Modelling Interface specification to explicitly define, describe and transfer data, (i.e. numerical data variable in space and time), between components on a time step basis, including associated component access. The full text version is contained in the document:

Gijssbers, P.J.A. (ed.) (2004) The HarmonIT Document Series Part C - The org.OpenMI.Standard interface specification. (v.0.99)

This full text version is available on www.openmi.org and www.harmonit.org.

OpenMI is developed as part of the HarmonIT research project co-funded by the European Commission (Contract EVK1-CT-2001-00090). It is the ambition of the HarmonIT consortium to turn OpenMI into a widely accepted 'standard' for model linkage in the water and possibly environmental domain.

Introducing OpenMI

Integrated catchment management asks for integrated analysis that can be supported by integrated modelling systems. These modelling systems can only be developed and maintained if they are based on a collection of interlinked models. OpenMI has been developed to provide a widely accepted unified method to link models, both legacy code and new ones.

Model applications consist of many parts, the most common being the user interface, input files, the engine and output files. OpenMI only addresses the engine part of a model application, where a model can be regarded as an entity that can provide data and/or accept data. Using this data exchange paradigm as a leading principle for model linkage, OpenMI is based on direct access of the model at run time, thus not using files for data exchange. In order to make this possible, the engine needs to be turned into an engine component and the engine component needs to implement an interface through which the data inside the component is accessible. OpenMI defines a standard interface that engine components must implement to become OpenMI compliant engine components. When an engine component implements this interface it becomes a linkable component.

The linkable component interface enables on-line (memory based) data exchange between components on a time (e.g. a time stamp or a time span). By combining this linkable component interface with a state management interface, advanced model combinations can be developed with simple controllers for iteration or optimization purposes. Most important however is the fact that OpenMI is not based on framework, it is based communicating linkable components only.

In summary, OpenMI primarily focuses on providing a complete protocol to explicitly define, describe and transfer (numerical) data between components on a time basis, including associated component access.

OpenMI: a request & reply architecture

OpenMI is based on the ‘request & reply’ mechanism. According to Buschmann et al. (1996), OpenMI is a pull-based pipe and filter architecture which consists of communicating components (source and target components) which exchange data in a pre-defined way and in a pre-defined format. OpenMI defines both the component interfaces as well as how the data is being exchanged. The components in OpenMI are called linkable components to indicate that it involves components that can be linked together.

From the data exchange perspective, OpenMI is a purely single-threaded architecture where an instance of a linkable component handles only one data request at a time before acting upon another request. Data exchange in the OpenMI-architecture is triggered by a component at the end of the component chain. Once triggered, components exchange data autonomously without any type of supervising authority. If necessary, components start their own computing process to produce the requested data. Only when output needs to converge to a certain criteria, a LinkableComponent with controlling functionality might need to be incorporated.

How OpenMI addresses general issues of model linkage

The OpenMI standard addresses the following items required for model linkage:

- **Data definition:**
The base data model of OpenMI addresses the numerical values itself, and its semantics in terms of quantity (what), element set (where), time (when), and data operations (how). The relevant interfaces are represented in Figure 1.
- **Meta data defining potentially exchangeable data:**
Quantities, elements sets and data operations are combined in exchange item definitions to describe the data that can potentially be provided and accepted by a linkable component (see Figure 2)
- **Definition of actually exchanged data:**
A link describes the data to be exchanged, in terms of a quantity on an element set using certain data operations. (See Figure 2)
- **Data transfer:**
Linkable components can exchange data by a pull mechanism, meaning that a (target) component that requires input asks a source component for a (set of) value(s) for a given quantity on a set of elements (i.e. locations) for a given time. If required, the source component calculates these values and returns them. This pull mechanism has been encapsulated in one single method, the GetValues()-method. Dependent on the status of the source component, this call may require associated computation and even more requests for data. An important feature is the obligation that components always deal with requests in order of receipt.
- **Generic component access:**
All functionality comes available to other components through one base interface, the linkable component interface (see Figure 2). This interface needs to be implemented by any component to become OpenMI complaint. Two optional interfaces have been defined to extend its functionality with discrete time information and state management (see Figure 2). To locate and access the binary software unit implementing the interface, the OMI-file has been defined. The OMI file is an XML file of a predefined XSD-format which contains information about the class to instantiate, information about the assembly hosting the class and the arguments needed for initialization (see Figure 3, Figure 4 and Figure 5).

- **Event mechanism**

In addition to the above mentioned functionalities to link components, a lightweight event mechanism has been introduced (interfaces defined in Figure 2). Via this mechanism a wide range of messages can be passed to enable call stack tracing, progress monitoring and to flag status changes which might trigger other components (e.g. visualization tools) to request for data via a `GetValues()`-call.

By convention a linkable component has to throw an exception if an internally irrecoverable error occurs. This exception should be based on the `Exception`-class as provided by the development environment.

Consequences of the OpenMI architecture for a model

The OpenMI enables model engines to compute and exchange data at their own heartbeat, without any external control mechanism. Deadlocks are prevented by the obligation of a component always to return a value whatever the situation. When each model is asked for data it decides how to provide it – it may already have the data in a buffer because it has previously run the appropriate simulation, or by running its own simulation or calculation, or by making a best estimate via interpolation or extrapolation. Or it may not be able to provide the requested data, so will raise an exception. The exchange of data at run-time is automated and driven by the pre-defined links, with no human intervention.

To become an OpenMI linkable component, a model has to:

- be able to expose information (what, where) to the outside world on the modeled variables which it is can provide, or which it is able to accept;
- submit to run-time control by an outside entity;
- be structured in a way that initialization is separate from computation, where boundary conditions are collected in the computation phase and not during initialization;
- be able to provide the values of the modeled variables for the requested points in time and space;
- be able to respond to a request, even when the component itself is time independent; if such response requires data from another component, the component should be able to pass on the time as well in its own request.
- In case some values in the value set are missing, this should be flagged in the value set.
- In the exceptional case that an entire value set is unavailable, an exception needs to be thrown. Be aware that such exception will stop the entire computation process and thus should be prevented whenever possible.

The utilization phases of an OpenMI linkable component

An OpenMI linkable component provides a variety of services which can be utilized in various phases of deployment. Figure 6 provides an overview of the phases that can be identified, and the methods which might be (logically) invoked at each phase. While the sequence of phases is prescribed, the sequence of calls within each phase is not prescribed.

ID	Phase	Description
I	Initialization	This phase ends by the situation where a linkable component has sufficient knowledge to populate itself with model data and expose its exchange items. Whether the linkable component has been populated with model data depends on the solution chosen by the code developer.
II	Inspection & Configuration	Dependent on the setting this phase might be very static and straightforward or very dynamic. The end situation of this phase is the following: The links have been defined and added and the component has validated its status
III	Preparation	This phase is entered just before the computation/data retrieval process starts. Its main purpose is to define a clear take off position before the bulky work load starts.
IV	Computation / execution	During this phase, the heavy work load will be executed and associated data transfer will get bulky. The data transfer mechanism of OpenMI is defined as a request-reply service mechanism, having direct interaction between two linkable components without any involvement of external facilities. Two types of data transfer are distinguished: unidirectional data transfer and bi-directional data transfer.
V	Finish	This phase comes directly after the computation/data retrieval process is completed. Code developers can utilize this phase to close their files and network connections, clean up memory etc.
VI	Disposure	This is phase is entered at the moment an application is closed. All remaining objects are cleaned and all memory (of unmanaged code) is de-allocated. Code developers are not forced to accommodate re-initialization of a linkable component after Dispose() has been called.

Note that linkable components may support dynamic adding and removing links at computation time. However, as addressed in the grouping of deployment phases, this requirement is not enforced. Those who do not support this method-call at computation-time should throw an exception.

Final remarks

In principle, the GetValues()-call stack of all linkable components takes place in one thread. Linkable Components may internally use distributed computing techniques to improve computational efficiency.

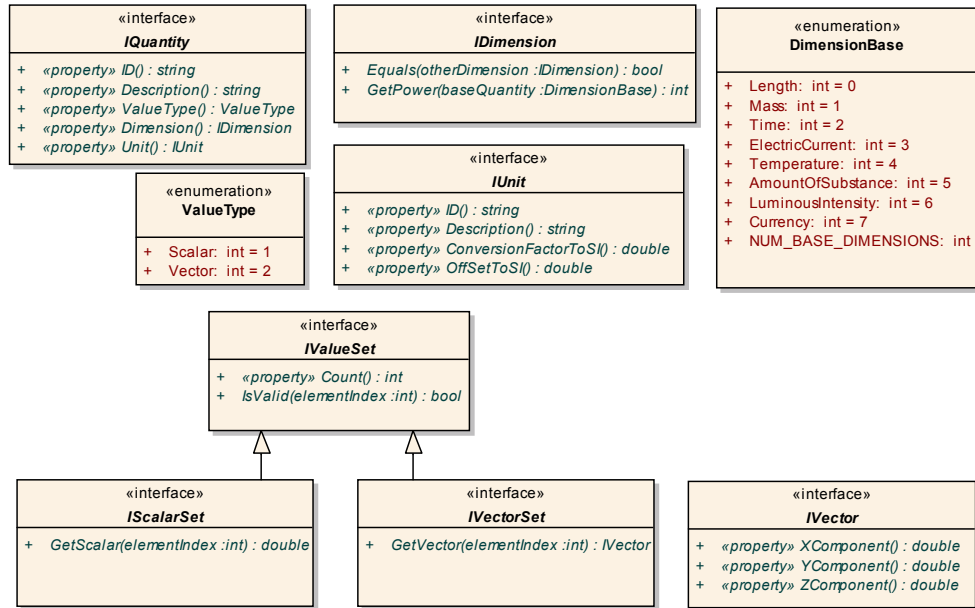
Code developers may create container components holding other components, as long as the container implements the linkable component interface.

By separating various phases of deployment, code developers can choose themselves when to instantiate and populate the engines. Exchange item information might be obtained from the engine, but may also be captured in files which are parsed during the inspection phase. Exchange item information might also be derived dynamically by querying linked components.

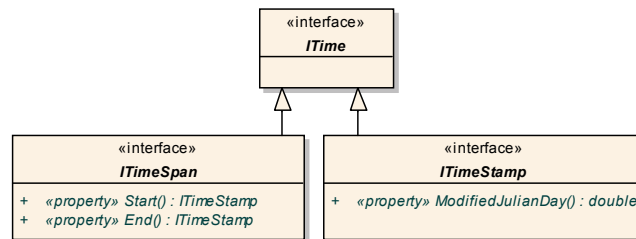
The element set contains a version number which can be utilized to accommodate dynamic changes of the content.

data definitions

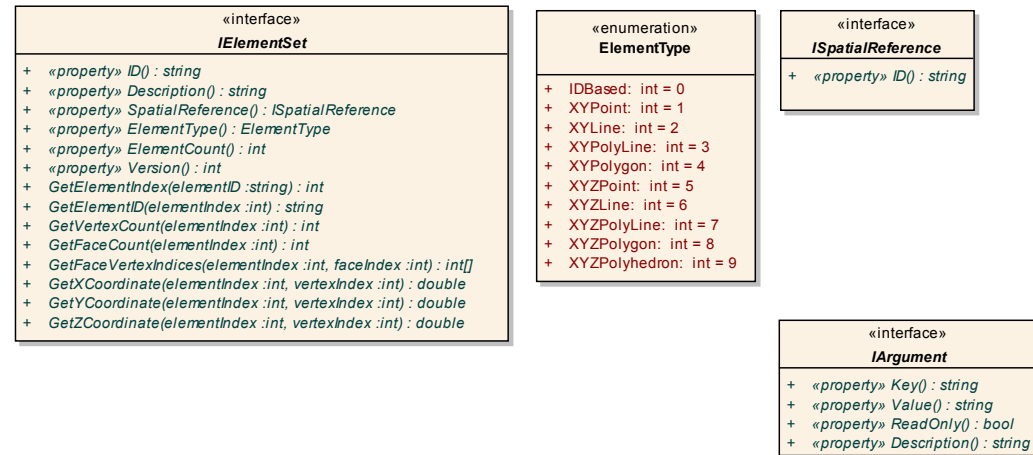
What



When



Where



How

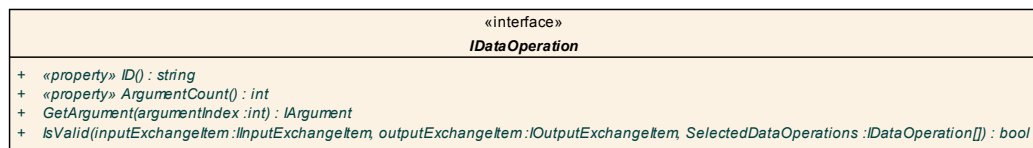
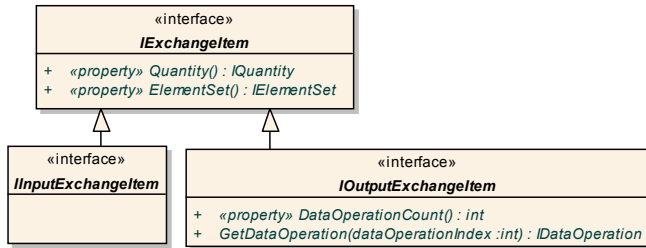
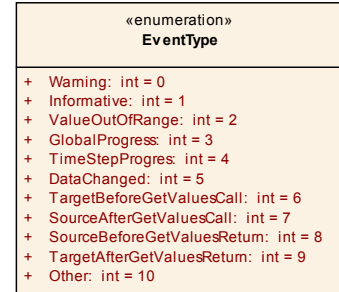
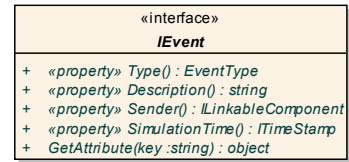


Figure 1 Data definitions in the OpenMI interface specification

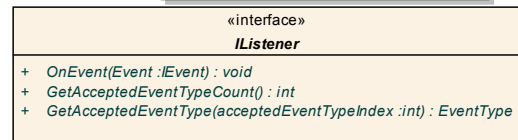
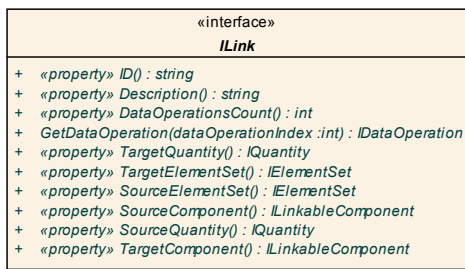
meta data to express what can be exchanged



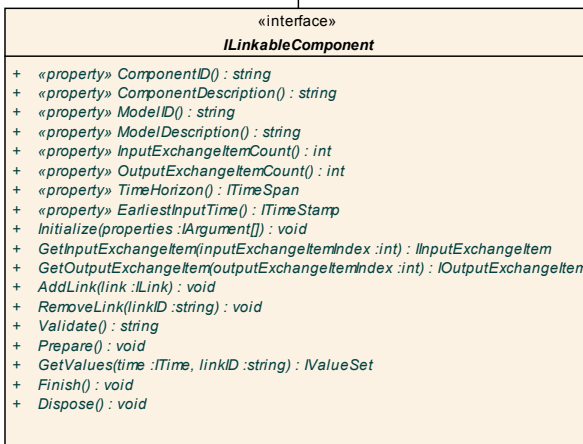
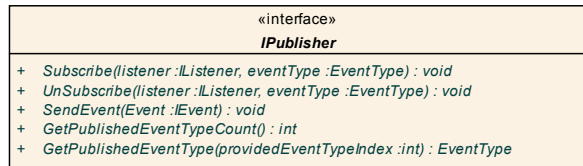
message definition



specification what will be exchanged and how



component interfaces for generic component access



advanced component interface extensions (optional)

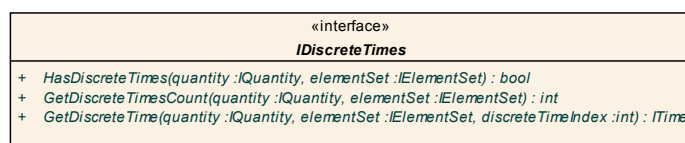
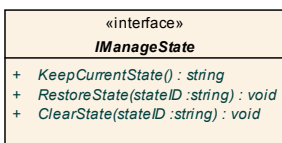


Figure 2 Other interfaces of the OpenMI

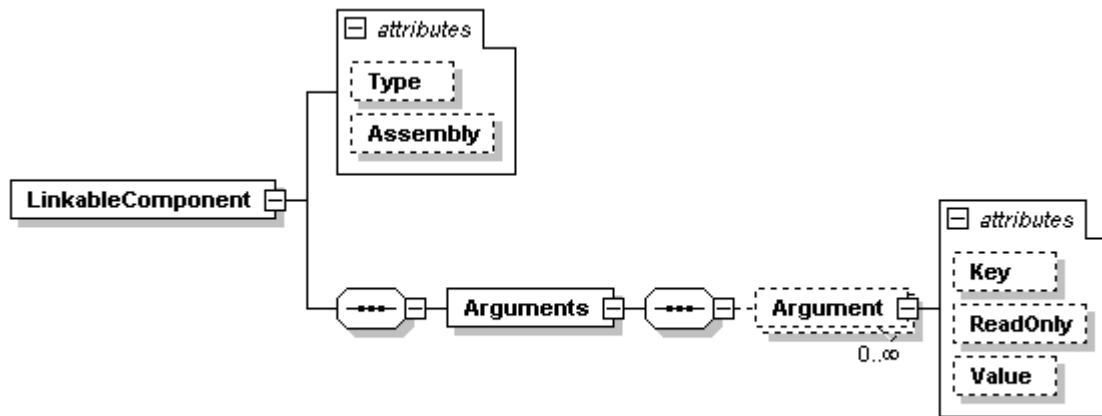


Figure 3 Graphical view of the OMI file structure

```

<?xml version="1.0" ?>
<xs:schema id="LinkableComponent" targetNamespace="http://tempuri.org/LinkableComponent.xsd"
  xmlns:mstns="http://tempuri.org/LinkableComponent.xsd" xmlns="http://tempuri.org/LinkableComponent.xsd"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
  attributeFormDefault="qualified" elementFormDefault="qualified">
  <xs:element name="LinkableComponent">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Arguments" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Argument" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="Key" form="unqualified" type="xs:string" />
                  <xs:attribute name="ReadOnly" form="unqualified" type="xs:boolean" use="optional" />
                  <xs:attribute name="Value" form="unqualified" type="xs:string" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="Type" form="unqualified" type="xs:string" />
      <xs:attribute name="Assembly" form="unqualified" type="xs:string" use="optional" />
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Figure 4 The Xml Schema Definition (XSD) of an OMI file

```

<?xml version="1.0"?>
<LinkableComponent Type="w1Delft.OpenMI.WLLinkableComponent" Assembly="w1Delft.OpenMI, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=8384b9b46466c568" xmlns="http://tempuri.org/LinkableComponent.xsd">
  <Arguments>
    <Argument Key="Model" ReadOnly="true" Value="RR" />
    <Argument Key="Schematization" ReadOnly="true" Value="D:\Rain-RR-CF\Model\Cmtwork\sobek_3b.fnm" />
  </Arguments>
</LinkableComponent>

```

Figure 5 Example of an OMI file

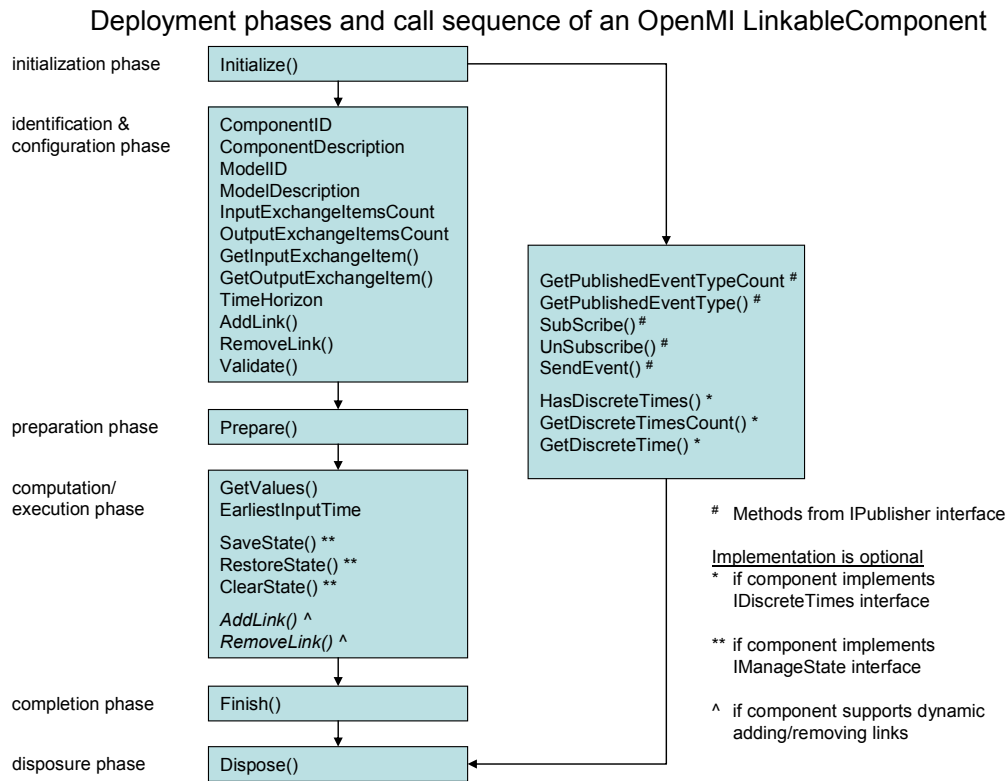


Figure 6 Deployment phases and call sequence of a Linkable Component