

# OpenMI: The Essential Concepts and their Implications for Legacy Software

**J. B. Gregersen<sup>a</sup> and M. Blind<sup>b</sup>**

<sup>a</sup> *DHI Water & Environment, Denmark*

<sup>b</sup> *RIZA Institute for Inland Water Management and Waste Water Treatment, The Netherlands*

**Abstract:** Information & Communication Technology (ICT) tools such as computational models are very helpful in designing river basin management plans (rbmp-s). However, in the scientific world there is consensus that a single integrated modelling system to support the Water Framework Directive cannot be developed and that integrated systems need to be very much tailored for the local situation. As a consequence there is an urgent need to increase the flexibility of modelling systems, such that dedicated model systems can be developed from available building blocks. The HarmonIT project aims at precisely that. Its objective is to develop and implement a standard interface for modelling components and other relevant tools: The Open Modelling Interface (OpenMI). The architecture for OpenMI is complete and has been documented. It relies entirely on the “pull” principle, where data are pulled by one model from the previous model in the chain. There are, of course, complications in satisfying the needs for iteration, buffering and feedback loops. This paper and presentation gives an overview of the architecture, explains the foremost concepts and the rationale behind them. Most importantly it will provide workable insight in the consequences for existing software. The paper contains information for water model developers about how they would put an OpenMI “wrapper” around their model for incorporation in integrated suites of models that use the OpenMI standard. This should give modellers opportunities to make their models available to a wider range of users world-wide.

**Keywords:** Model linking, decision support systems, interface standard, legacy model

## 1. INTRODUCTION

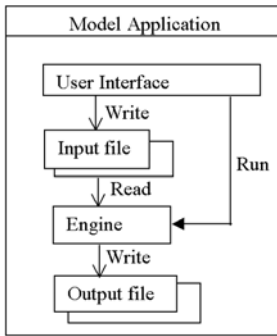
The concept of integrated catchment management has arisen because managing environmental processes independently does not always produce sensible decisions when the wider view is taken. Therefore, it becomes important to be able to model not only the individual catchment processes such as ground water, river flow, irrigation, etc, but also their interactions. However, most existing models tend to address only single issues. The objectives of the HarmonIT project address these problems through the development of an open modelling interface (OpenMI) that will facilitate easy linking of existing and new models.

The HarmonIT project will, apart from the development of the OpenMI standard, migrate about 30 existing legacy models to the OpenMI platform. We believe that the number of OpenMI compliant models will keep growing also beyond the timeframe of the HarmonIT project. Most existing hydrological decision support systems use

combined hydrological models as the one of the main building blocks. Creation of such systems has so far been prerogative to the model suppliers. Now with OpenMI compliant models available also third parties can create such systems for their specific needs.

## 2. EXISTING MODEL SYSTEMS

Before going into detail about OpenMI some definitions of the existing model systems are given.

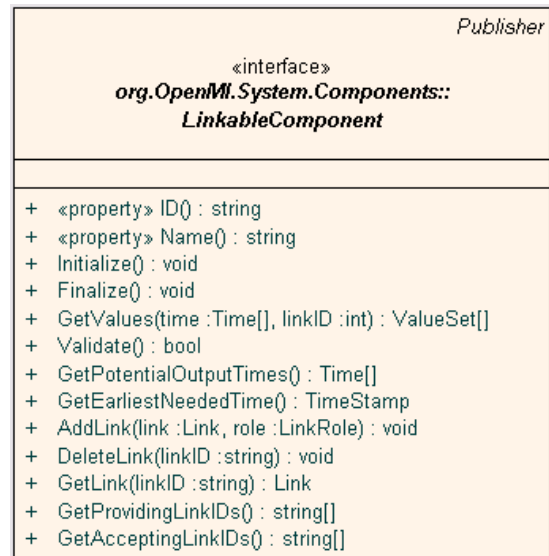


**Figure 1:** Model application pattern

A *model application* is the entire model software system that you install on your computer. Normally a model application consists of a user interface and an engine. The engine is where the calculations take place. The user supplies information through the user interface upon which the user interface generates input files for the engine. The user can run the model simulation e.g. by pressing a button in the user interface, which will deploy the engine (see figure 1). The engine will read the input files and perform calculations and finally the results are written to output files. When an engine has read its input files it becomes a *model*. In other words a model is an engine populated with data. A model can simulate the behaviour of a specific physical entity e.g. the River Rhine. If an engine can be instantiated separately and has a well-defined interface it becomes an *engine component*. An engine component populated with data is a *model component*. There are many variations of the model application pattern described above, but most important from the OpenMI perspective is the distinction between model application, engine, model, engine component, and model component.

### 3. OpenMI

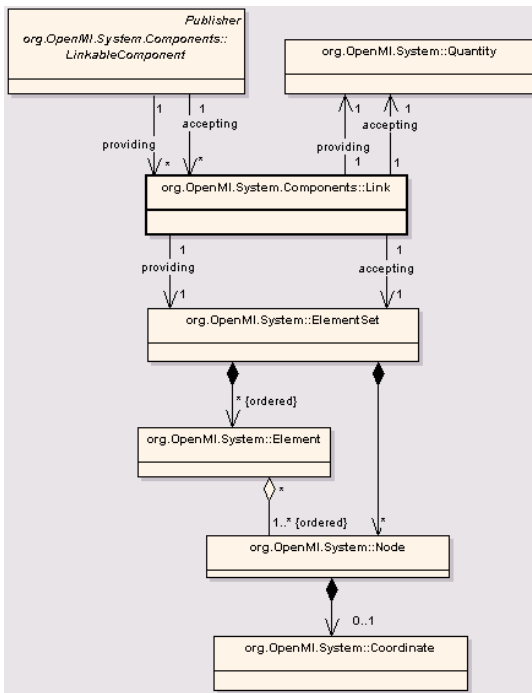
Basically, a model can be regarded as an entity that can provide data and/or accept data. Most models receive data by reading input files and provide data by writing output files. However, the approach for OpenMI is to access the model directly at run time and not to use files for data exchange. In order 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 for engine components (ILinkableComponent, see figure 2) that OpenMI compliant engine components must implement. When an engine component implements the ILinkableComponent interface it becomes a LinkableComponent.



**Figure 2:** Linkable component interface

One LinkableComponent can retrieve data from another LinkableComponent by invocation of the GetValues method. However, this is only possible if the two components have information about each others existence and have a clear definition of the kind of data that is requested. This information is contained in the OpenMI Link class. Before invocation of the GetValues method a Link object must be created, populated and added to the two components by use of the AddLink method.

The Link object holds a reference (handle) to the two linked components. The Link object also contains information about *what* is requested, *where* the requested values apply, and *how* the requested data should be calculated. This information is included in the OpenMI Quantity class, the OpenMI ElementSet class, and the OpenMI DataOperation class, respectively. Figure 3 shows how all this information is organized into the Link class.



**Figure 3:** The Link class and associated classes

The Quantity object defines what should be retrieved. This could be e.g. water level or flow. The Quantity class represents this information simply as a text string (quantity description). OpenMI does not provide any naming convention for quantities. The Link class has an accepting Quantity object and a providing Quantity object. The quantity description in the providing Quantity object must be recognizable by the providing LinkableComponent and the quantity description in the accepting Quantity object must be recognizable by the accepting LinkableComponent. It is the responsibility of the person that configures the linked system to ensure that the combination of the two particular quantities makes sense physically.

The ElementSet object defines where the retrieved values must apply. A ground water model may be requested for either the ground water level at a particular point or the ground water level as an average value over a polygon. A river model may be requested for the flow at a particular calculation node. These locations are defined in the ElementSet. The ElementSet is a collection of Elements, where each element can be an ID-based entity like a particular node or a geometrical entity. A geometrical entity is either a point, a polyline, a polygon, or a 3D shape. The GetValues method returns a ValueSet, which is an array of values. Each value in the returned ValueSet applies to one Element in the accepting ElementSet.

The DataOperation object defines how the requested values should be calculated. Examples of data operations could be time accumulated, spatially averaged, maximum values etc. There are no OpenMI conventions for data operations. As for the quantities, the data operations are simply defined by a text string, which is recognizable by the providing LinkableComponent.

The Link class defines a specific connection between two LinkableComponents. For two specific LinkableComponents many possible links may exist.

When model links are created and populated, information about which quantities, locations and operations each LinkableComponent will support is needed. Therefore, it is required that a LinkableComponent is associated with an XML document that holds this information. These XML documents must comply with an OpenMI exchange model schema definition and they must contain information about potential output and input quantities, potential output and input ElementSets, potential output DataOperations, and the time frame for the model. This information is organized hierarchically. This means that the XML document can be queried for potential output quantities, then queried for potential output ElementSets for a particular quantity, and then queried for potential data operations for a particular quantity and ElementSet. The same type of queries can be made for input quantities and input ElementSets.

Time in OpenMI is defined either by a TimeStamp class or a TimeSpan class, both classes inherited from the abstract OpenMI Time class. A time stamp is a single point in time whereas the time span is a period from a begin time to end time. Each of these times is represented by the Modified Julian Date. A modified Julian date is the Julian date minus 2400000.5. A modified Julian date represents the number of days since midnight November 17, 1858 Universal Time on the Julian Calendar.

Now let us move back to the essence of OpenMI, the GetValues method. When one LinkableComponent invokes the GetValues method of another LinkableComponent, the providing LinkableComponent must return the values for the specified quantity, the specified time stamp or time span, and at the specified location. If the LinkableComponent is of the time stepping kind of numerical model it will do no calculation until it receives a GetValues call. Once the

GetValues method is invoked, it will calculate as long as it is necessary to obtain the needed data. Usually it will be necessary for the providing component to interpolate or extrapolate its internal data in time and space before these can be returned.

The OpenMI architecture puts a lot of responsibilities on the LinkableComponents. One of the reasons for this is that we feel that any data conversion like interpolations can be done in the optimal way by the providing component. If the providing component is e.g. a ground water model any interpolations of the ground water levels are most safely done by the ground water model itself rather than some external tool.

The OpenMI framework is very simple or you may say that there is no framework. All there is, is LinkableComponents. Once the system of linked model components is created, the invocation GetValues methods from one model component to another is driving the calculations. Since the ILinkableComponent interface (figure 2) does not have any methods that can be used to start the chain of calculations, a trigger component is needed. The trigger component is a Linkable component that has an additional method for starting calculations (see example below).

#### 4. EXAMPLE

Let us look at a simple example: A conceptual lumped rainfall runoff (RR) model provides inflow to a river model. The populated link class is shown in figure 4.

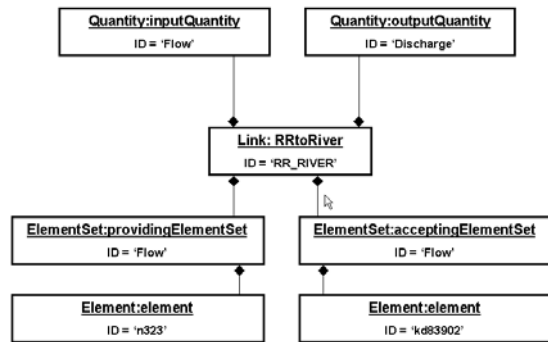
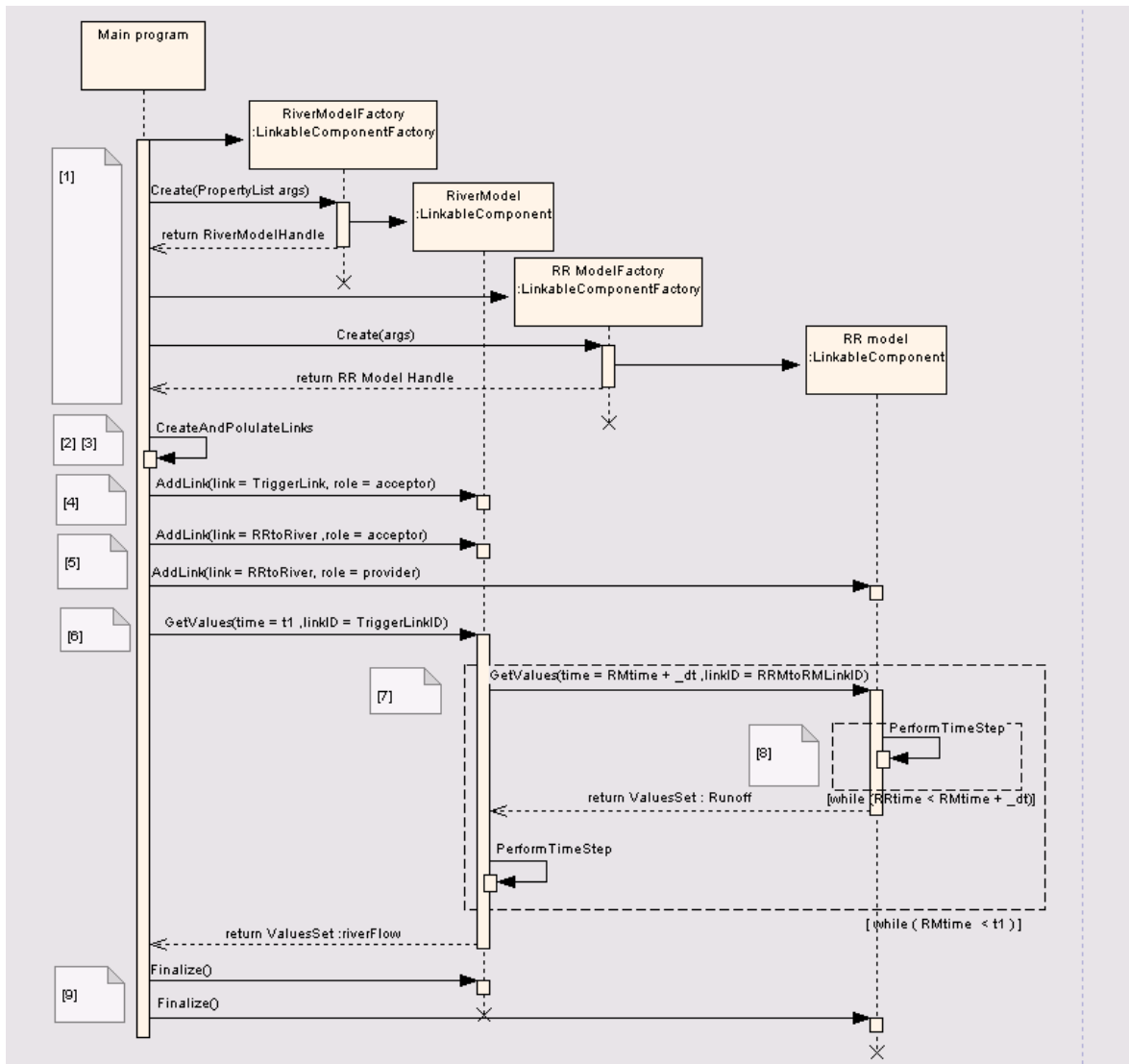


Figure 4: The populated Link class

The sequence diagram in figure 5 shows how the calling sequence for a configuration with a river model linked to a rainfall-runoff model is set-up. The diagram demonstrates how things would look if a “hardcoded” configuration were used. For normal usage of OpenMI a configuration editor would create a configuration for you.

The sequence diagram has the following steps:

1. Each LinkableComponent must have an associated LinkableComponentFactory. The Factory class will construct the LinkableComponent. As a part of the construction the LinkableComponent object will be initialized, which typically involves reading input files. The LinkableComponentFactory object can then return a handle to a LinkableComponent object that is initialized and ready to go.
2. The Link object between the RR model and the river model is created and populated (see figure 4).
3. A trigger link to the river model is created and populated. The purpose of this link is to enable the first GetValues call to the RiverModel. This call is what triggers the calculation chain.
4. Add the trigger link to the river model component.
5. Add links to the model components.
6. Invoke the GetValues method in the River model component. The time argument defines the time for which results are expected from the river model component.
7. The river model will evaluate whether its internal time (RMtime) is before or after the requested time (t1). While RMtime is less than t1 the river model will perform time steps. The river model will, before each time step, retrieve the runoff from the RR model by invoking the GetValues method in the RR model.
8. The RR model will perform as many time steps as necessary in order to calculate the requested value. Note that the two models do not need to have matching time steps. It is the responsibility of the delivering model to do any needed interpolation in order to return a value that represents the time argument in the GetValues call.
9. When the calculations are completed the Finalize method in each component is called. Typically the components will free the memory, close the file and the like.



**Figure 5:** Sequence diagram for the example

## 5. OpenMI COMPLIANCE

An OpenMI compliant model must provide:

1. A component that implements the ILinkableComponent interface
2. A LinkableComponentFactory class that can construct and initialize the LinkableComponent
3. A model description XML – file that follows the OpenMI model description schema
4. An exchange model XML file that follows the OpenMI exchange model schema.

## 6. OpenMI UTILITIES AND TOOLS

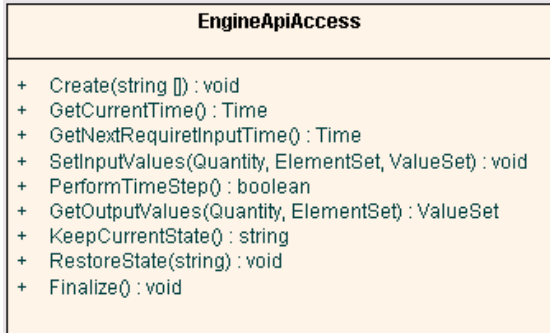
At first hand it may seem like a huge challenge to turn a model engine into an OpenMI compliant LinkableComponent. However it is not so difficult. OpenMI provides guidelines for migration of models and a great number of software tools and utilities that will make migration easier. These tools and utilities can be used by anyone who is migrating a model but are not required in order to comply with the OpenMI standard.

For existing model engines wrapping is the recommended technology for migration. If a model engine is e.g. a numerical model programmed in Fortran this engine can be compiled into a dynamic link library (dll). If this dll is organized with a specific Win32API (see figure 6), an OpenMI utility wrapper class (SmartWrapper) can be used.

This class will take care of all the bookkeeping associated with handling links, and all the interpolation in time and space. We estimate that migration of models will take between a few weeks to a few months of work, depending on how well organized the program code is.

## 8. REFERENCES

[1] The OpenMI web site: <http://www.OpenMI.org>



**Figure 6:** Required WIN32 dll API for model engines, when the SmartWrapper is used.

The SmartWrapper can be used for most numerical time stepping engines. Implementation of the KeepCurrentState and the RestoreState methods are optional since they are only used for systems where iterations are needed. So in order to use the SmartWrapper you need to make sure that your engine complies with the following:

1. Initialization must be a separate function or subroutine.
2. The time stepping loop must be broken up so that a function or subroutine performs only one single time step.
3. A function or a subroutine that can get values from the engine core must be created.
4. A function or a subroutine that can set values in the engine core must be created.

OpenMI also provides software class libraries for working with exchange models (the XML files and associated data). The configuration package will also support creation of configured systems and the deployment of these. For end-users that only need to link and deploy existing OpenMI compliant models, a configuration editor is provided.

## 7. CONCLUSIONS

We feel that the OpenMI architecture and associated tools and utilities will be an attractive way for model providers and model users to create systems for integrated catchment management. More information about OpenMI and HarmonIT is provided on our web site [1]