*International Environmental Modelling and Software Society (iEMSs)*
*2010 International Congress on Environmental Modelling and Software*
*Modelling for Environment's Sake, Fifth Biennial Meeting, Ottawa, Canada*
*David A. Swayne, Wanhong Yang, A. A. Voinov, A. Rizzoli, T. Filatova (Eds.)*
*http://www.iemss.org/iemss2010/index.php?n=Main.Proceedings*

# OpenMI 2.0 - What's new?

**Gennadii Donchyts[1,2], Stef Hummel[1,3], Stanislav Vaneçek[1,4], Jesper Groos [1,5], Adrian Harper [1,6], Rob Knapen[1,7], Jan Gregersen[1,8], Peter Schade[1,9], Andrea Antonello[1,9], Peter Gijsbers[1,10]**

[1] *The OpenMI Association, Lelystad, The Netherlands*
[2] *Stichting Deltares, Delft, The Netherlands Gennadii.Donchyts@deltares.nl;*
[3] *Stichting Deltares, Delft, The Netherlands Stef.Hummel@deltares.nl;*
[4] *DHI, Prague, Czech Republic, S.Vanecek@dhi.cz*
[5] *DHI, Hørsholm, Denmark, jgr@dhigroup.com*
[6] *MHW Soft, Wallingford, UK Adrian.Harper@wallingfordsoftware.com*
[7] *Alterra, Wageningen, The Netherlands Rob.Knapen@wur.nl*
[8] *Hydroinform, , Denmark Gregersen@hydroinfrom.com*
[9] *Bundesanstalt für Wasserbau, Germany Peter.Schade@baw.de*
[10] *University of Trento, andrea.antonello@gmail.com*
[10] *Deltares USA Inc., Silver spring, Maryland, USA Peter.Gijsbers@deltares-usa.us*

**Abstract:** The first version of the OpenMI standard was developed as a joint effort of several European research organizations. OpenMI stands for Open Modelling Interface and aims to deliver a standardized way of linking environmental models at run-time. In the new version of the standard several new goals were defined based on experience obtained during migration and use of the OpenMI-compliant models. This includes on one side different IT aspects such as better object-oriented design of the standard and re-use of well-known engineering practices and patterns. On the other side, after successful implementation of OpenMI in many environmental models it was also decided to extend scope of the OpenMI standard to a broader set of applications such as GIS data types, monitoring databases, running models in parallel (versus sequential pull-driven approach), improved workflow management and many others. This paper gives the details of OpenMI 2.0.

***Keywords:*** Open Modeling Interface; linking; model interoperability; integrated modeling

## 1. INTRODUCTION

The first version of the OpenMI standard was developed with an extensive focus on coupling of the numerical models, primarily in the field of surface and ground water hydraulics. Although notice was taken of the needs of other domains, e.g. economics, this hardly influenced the outcome of the OpenMI version 1.4. However, usage of OpenMI in these other domains, e.g. in the SEAMLESS project (Knapen et al. [2009]), or during application of OpenMI for web-based systems (Goodall [2007]) have shown that OpenMI requires considerable improvements to fit more properly to these type of applications (see Gijsbers et al. [2010], for details and argumentation).
Also for other reasons OpenMI turned out not to be perfect yet. While the list of OpenMI-compliant models grew[1] it became clear that there were more issues to be considered then the domain area only. Better support was needed for time independent models, for algebraic models, and for GIS based applications.
Another important reason to adjust OpenMI was more technical: improve the quality of the standard by applying more object-oriented design patterns (Gamma et al. [1994]), and by using language and platforms specific features such as events, properties and standard collections, as available in both the .NET Framework and java. Making the standard more

[1] See the list of OpenMI-compliant models on http://openmi.org

intuitive and self-explaining while at the same time expanding its scope was the main goal of the OpenMI Association's Technical Committee when developing OpenMI version 2.0. An important additional aspect taken in account was the fact that the implementation of an OpenMI 1.4 component usually relies quite heavily on the OpenMI 1.4 Software Development Kit. For computational cores this SDK provides more intuitive interfaces then the ones defined in the standard itself, so often component developers implemented the SDK's IEngine/IRunEngine interfaces instead of the more abstract ILinkableComponent in the standard. In the OpenMI 2.0 this will not be the case anymore, since ILinkableComponent (Figure 1) now better covers handling of the computational cores.

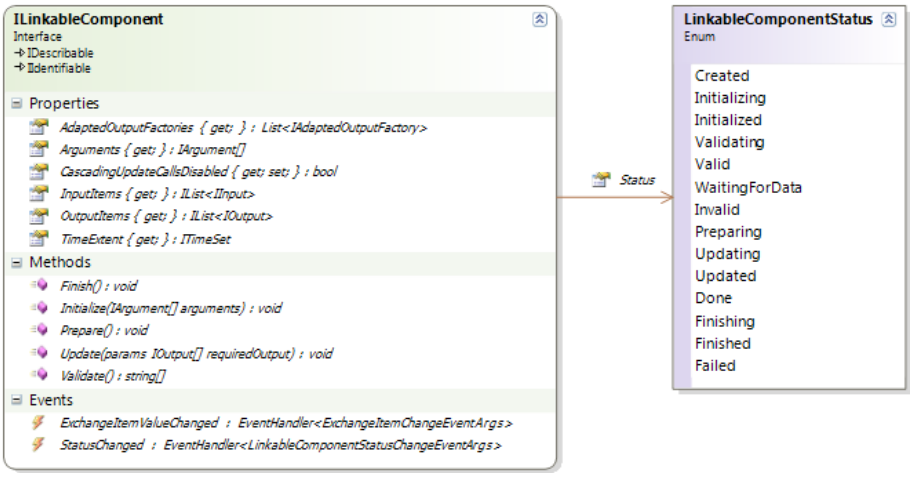



**Figure 1** ILinkableComponent class diagram

The goal of this paper was to provide a technical overview of all major changes in OpenMI 2.0. The list below summarizes some main code changes of the OpenMI 2.0 which will be discussed in the present article:

- Combine all three concepts of *Where*, *When* and *What* in the IExchangeItem, whereas in the previous version *Where*, *When* in the previous version. *What* was part of the ILinkableComponent in the OpenMI 1.4.
- Remove the ILink interface and apply the Observer design pattern by connecting IOutput and IInput items by means of a provider/consumer relationship.
- Separate the concepts of *Perform Computation* and *Retrieve Values*. In OpenMI 1.4 they were combined in the GetValues() call.
- Make ILinkableComponent behave like a workflow activity, changing its status depending on operation performed.
- Use of the Adapter design pattern for data operations.
- Introduce language and platform-specific features into standard by allowing use of events, properties and standard collections Introduce loop-driven approach to run components
- Introduce, in addition to the pull driven control flow, a loop-driven control flow approach.
- Improve the management of a component's persistent state(s)
- Extend the exchangeable types of values with categorized variables that can represent either nominal or ordinal value categories
- Simplify time-related interfaces.
- Make OpenMI more OGC-friendly

## 2. NEW FEATURES

This paper discusses the technical details related to the mentioned changes, and provides argumentation on why they were applied.

### 2.1 Combining concepts of What, Where and When within IExchangeItem

From its beginning OpenMI used a concept of *What*, *Where* and *When* in order to describe values exchanged between different components. Probably the main change in implementations of these concepts in OpenMI 2.0 is that IExchangeItem now holds all meta information describing all of them (see Figure 2).
Typical steps required to exchange values in the OpenMI 1.4 are:

- Query information about element sets (*Where*) and quantities (*What*) from the exchange items defined in components.
- Create links between source and target components, element sets and quantities and add these links to the corresponding components.
- Prepare and initialize components.
- Perform the call *GetValues(ITime time, string linkID)* on a component, which on its turn can also pull values from other components by performing the *GetValues* call. The time for which values are required is provided as an argument to the *GetValues* method.

Note that *Where* and *What* are mainly used at configuration time and are defined in IExchangeItem, while *When* is used at a runtime as an argument to ILinkableComponent.GetValues call. IExchangeItem in OpenMI 1.4 did not provide any information about time, nor did it allow querying values available for an exchange item.
In OpenMI 2.0 all three aspects: *What*, *Where* and *When* are defined on IExchangeItem level, which results in a better separation of concepts. Figure 2 shows the interfaces used, and gives an example of an exchange item, e.g. water level, on an element set containing 5 elements (e1-5) and time set containing 2 time steps (10:00, 12:00). So in OpenMI 2.0 IExchangeItem is fully responsible to hold all meta-information plus values being exchanged while ILinkableComponent is responsible for performing an operation, e.g. query database, convert raster data to something else, and perform a computation for a few time steps, etcetera.

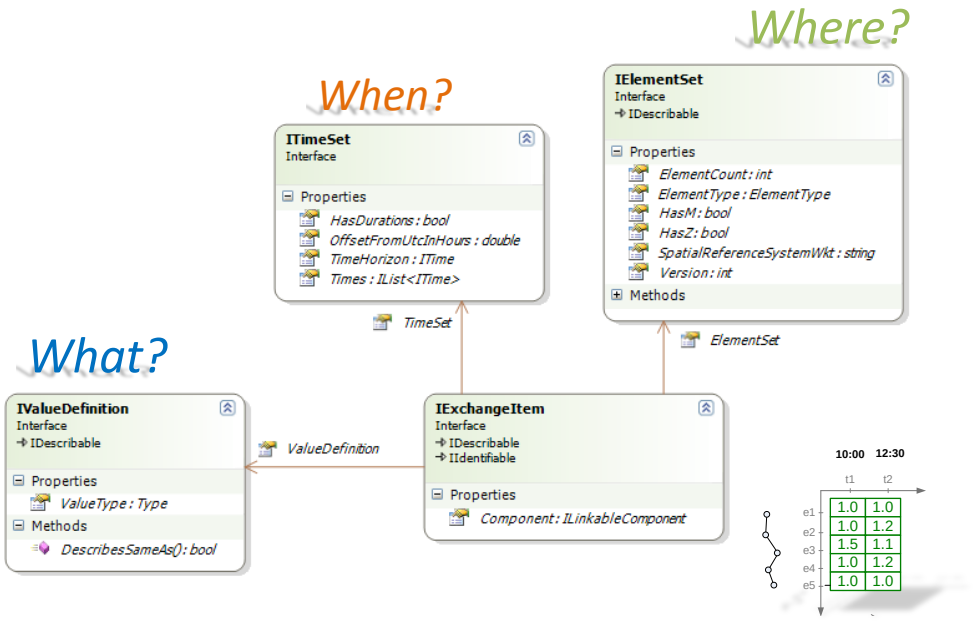



**Figure 2** IExchangeItem class diagram, concepts of Where, When and What

Note that in the new version IExchangeItem uses a property called ValueDefinition instead of Quantity in the previous version. The value definition allows the usage of either a Quantity or a Quality, in order to define values for nominal or ordinal variables, e.g. land use types, concentration level (low, medium high). Quantity or Quality are defined as interfaces extending IValueDefinition.

### 2.2 Linking components, remove ILink and use of an Observer design pattern

In the new version we do not use ILink anymore. For several reasons the link turned out to be more a burden then a benefit: confusion arises when more than one link has been connected to one input item; even for simple value retrieval a full link has to be created; the link needs a target component, so the values retriever always has to be a linkable component itself. In the new version we use the Observer design pattern defined on IExchangeItem level, see Figure 3. The figure shows the IInput and IOutput interfaces which represent different types of exchange items. The main difference from the previous version is that exchange items are now fully self-contained, responsible to keep all information which can be exchanged for a single variable, including values currently available. The linkable component is responsible to fill in values in all these exchange items. Implementations of the exchange items can be used and tested separately from the components.

A single output exchange item may provide values for multiple input exchange items. At configuration time output exchange items must be connected to input exchange items by means of the Consumers and Provider property. These properties define all established links, and therefore can be used to check how exchange items are connected. Once all exchange items are connected, components owning these exchange items may generate values by means of the Values property. Usually this happens after the component has performed some work (at the end of the Update() call, see chapter 6 for more details).

Another way to retrieve data, more known to users of OpenMI 1.4, is to perform a *GetValues(IExchangeItem query)* call on output exchange item. In OpenMI 2.0 this method is defined on the output exchange item instead of on ILinkableComponent, as it was in the previous version.

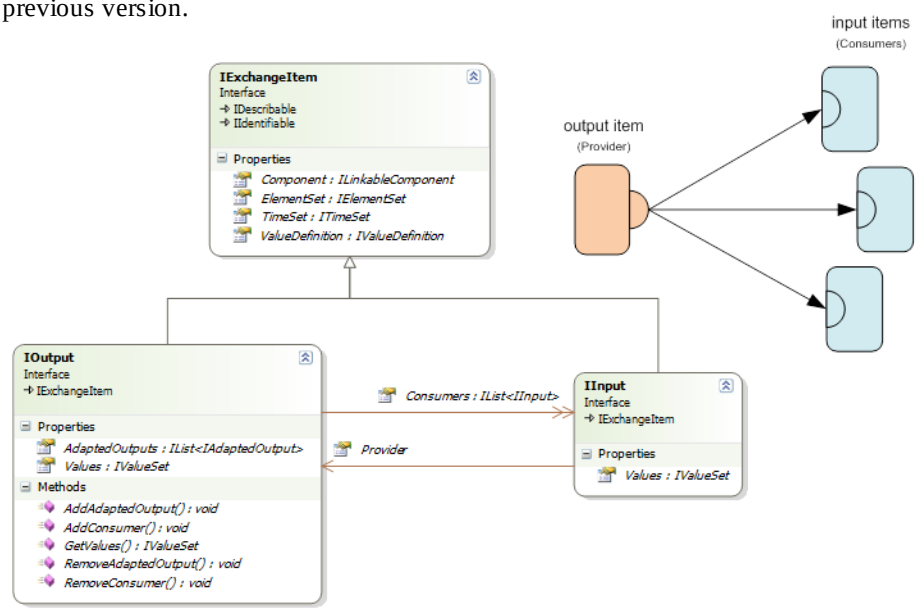



**Figure 3** Output and Input exchange items

### 2.3 Use of Adapter design pattern for data operations

OpenMI 1.4 allowed user to define various data operations when linking exchange items. However, they were not allowing performing different conversions in a chain, and the use of the data operations was not intuitive enough. They were defined on IOutputExchangeItem, and then had to be passed to the ILink implementation. OpenMI 2.0 simplifies this logic by introducing another type of output exchange item, called IAdaptedOtuput, which can wrap any output exchange item (adaptee) in order to perform a certain conversion.

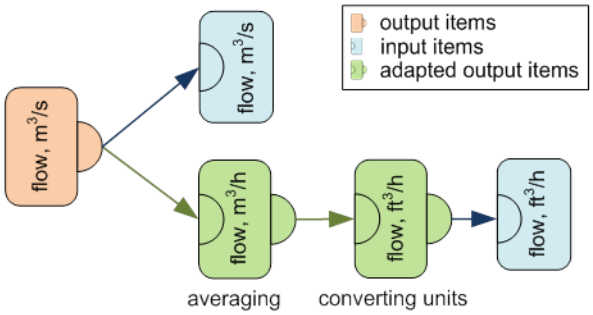



**Figure 4** Adapted output items

### 2.4 Separating compute and accessing values logic in the ILinkableComponent

The new version of OpenMI provides much better control over the workflow by means of separation of "Perform Calculation" and "Query Values" steps. In the previous version there was no way to query already computed values from the component unless component itself provided some kind of buffering (e.g. using buffer classes available in SDK). In the new version this mechanism was completely reworked so that current values can be queried at any time. As well as buffered values, if buffering is implemented using IAdaptedOutput. Additionally the new standard allows checking the status of the component, which might be required in order to know if values available on exchange items are up-to-date. Otherwise the **Update()** method must be called on the component. The table below summarizes differences between steps required to perform computation and query values in both versions.

**Table 1** Querying values in OpenMI 1.4 and 2.0

| Version | Perform operation | Query values |
|---|---|---|
| **1.4** | *IValueSet ILinkableComponent.GetValues(ITime time, string linkId)* – returns set of values provided by the source component of link defined between 2 components, quantities and element sets | |
| **2.0** | ILinkableComponent.Update() – updates component to the next valid state. Usually calling this methods performs a time step, queries data from a database or performs any other activity required to generate values in component output exchange items | IOutput.Values – property which can be used after component was updated and values were set on all required output exchange items. |
| | | *IValueSet IOutput.GetValues(IExchangeItem query)* – returns set of values for a given value definition, times, element set provided by query. If necessary – it may call update of the component where this exchange item is used, and perform GetValues calls on other components |

As can be seen from the table, OpenMI 2.0 provides 2 ways to query values from exchange items. The Values property can be used to get values which were generated previously, mainly after the Update() call. Additionally, the GetValues() method can be used to get a specific set of values from the exchange item. This is very useful for instance when target component / exchange item is not interested in all values available in the output exchange item but only in a subset. When specifying a query time that lies in the future, the providing component will have to propagate itself to the requested time; this leads to exactly the same pull driven control flow as in OpenMI 1.4.

### 2.5 Making ILinkableComponent behave like a workflow activity

Before version 2.0 OpenMI was mainly used in a single threaded environment. Components were not supposed to be used from another thread during e.g. GetValues() call. The new version does not have this limitation anymore. ILinkableComponent provides a new property *LinkableComponentStatus Status { get; }* which allows to determine what the component is doing right now. This opens new possibilities to use of OpenMI. However developers of the components should implement their components as thread-safe, if these components are expected to be used in a multi-threaded environment. In fact it means that

IOutput.GetValues() andValues property of IInput and IOutput interfaces must be thread-safe. The OpenMI 2.0 does not imply that all its methods need to be thread-safe, for example linking components to each other most probably will happen in a single-threaded environment.

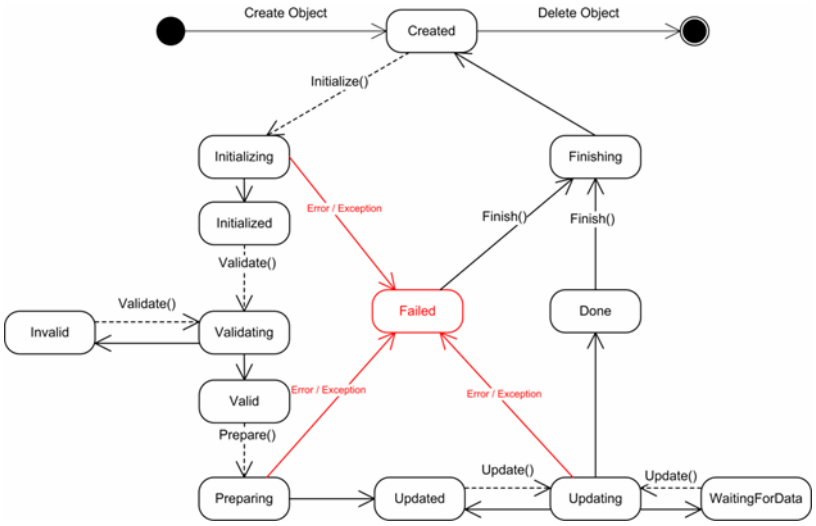



**Figure 5** Linkable component state chart diagram

### 2.6 Pull and Loop driven approach –the actual data exchange between components

In OpenMI 2.0 components can support two ways of control flows, called *pull-driven* and *loop-driven*. In the *pull-driven* control flow (default) components work like in OpenMI 1.4, Which means that a component may call its own Update(), as well as the GetValues()call on output items of the other components.
Another way to run components (*loop-driven*) assumes that components or exchange items should never trigger their own Update() call, nor the Update() and GetValues() of other components and their output items. Propagating the system by means of the Update() call should happen somewhere outside, in the control program containing these components. The *loop driven* approach requires more careful implementation, but it also opens a new ways to run components. For example, it allows control program to run components in the different threads, processes of even machines. In order to specify if a component supports *loop-driven* way of work a new property was introduced on ILinkableComponent: *bool ILinkableComponent.CascadingUpdateCallsDisabled { get; set; }. The* default value is false, which means that component is allowed to trigger other components. If this property is set to true, the component must never trigger other components. Usually this means that when a componentrequires certain input for its input exchange items that is not available on the related output exchange item of a connected component, the component should change its status to *WaitingForData* and check if the data is available during the next *Update()* call.

### 2.7 Improvements of the component's persistent state management.

OpenMI 2.0 facilitates a way to handle persistent states of linkable components. In the previous version it was only possible to trigger a component to remember its state, resulting in a string identifying the remembered state. The new version allows external programs to remember component states. In this case component must implement IByteStateConverter interface in addition to IManageState, see Figure 6.

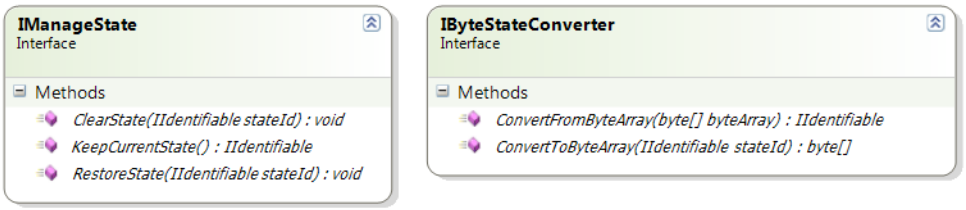



**Figure 6** Persistent state management interfaces

### 2.8 Simplification of the time-related interfaces

After review of the time-related interfaces in OpenMI 1.4 a few interfaces were removed from the standard. It showed that a single interface ITime can provide sufficient functionality for specifying a time stamp or a time span. ITime represents a time interval by defining the start of that interval as a time stamp and by specifying a duration. Duration equal to 0 simply means a time stamp. This approach simplifies the use of this interface in the Times property of the ITimeSet, a new interface in the OpenMI 2.0. ITimeSet in this case works similar as IElementSet: the elementset defines the spatial properties of an exchange item, the time set defines the time frame properties.

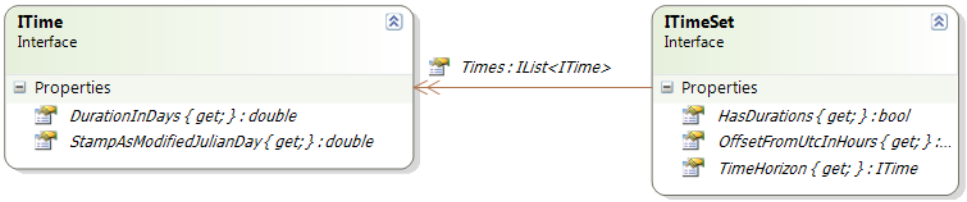



**Figure 7** Time interfaces class diagram

### 2.9 Making OpenMI more OGC-friendly

OpenMI uses a single interface IElementSet in order to define geometry of the exchange items. This interface was improved in order to simplify interoperability with OGC Simple Geometry Specifications standards. Since there are no standard C# and java versions of the OGC standards available yet, except GeoAPI.NET open-source project effort, it was decided to keep IElementSet as it was, and only extend it with a few properties: SpatialReferenceSystemWkt, HasM, HasZ.

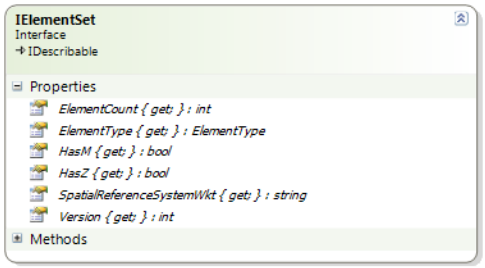



**Figure 8** IElementSet interface

This allows OpenMI to be generic enough but also to implement OGC-based element sets in the SDK, e.g. FeatureElementSet, RasterElementSet providing more GIS-specific functionality.

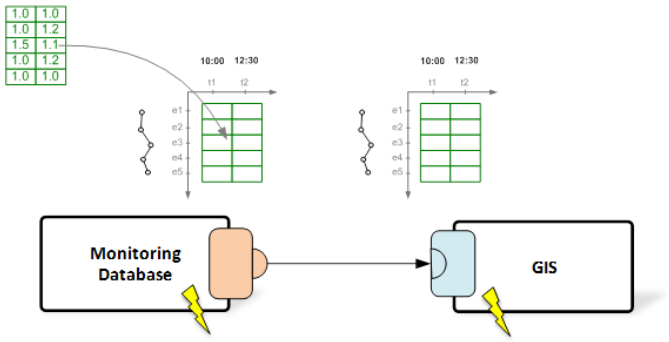
### 2.10 Language and platform specific features

After careful consideration OpenMI was extended with platform-specific features available in C#/.NET and in java. These features include:

- Events

- Properties (in java still represented by get/set methods)
- Collections

It was decided that the benefits that developers will gain during use of these features will make the standard more acceptable in the .NET and the java communities. And finding a proper alternative to those features is always possible when OpenMI has to be used in any other nowadays language. This allows the use of linkable components in a way as shown in code listing on Figure 9.




```
[Test]
public void ListenToStatusChangeEvents()
{
    var monitoringDatabase = new MonitoringDatabase();
    var gis = new Gis();

    // display debug log message when component changes it's status
    monitoringDatabase.StatusChanged += delegate
    {
        log.DebugFormat("Monitoring database status changed to {0}", monitoringDatabase.Status);
    };

    gis.StatusChanged += delegate
    {
        log.DebugFormat("GIS component status changed to {0}", gis.Status);
    };
}
```

**Figure 9** Use of events in OpenMI 2.0

## CONCLUSIONS

The list of the major changes to the OpenMI standard has been presented. OpenMI 2.0 is certainly a major step forward in the field of environmental modeling and we hope that the new features discussed here will simplify its application and result in better interoperability between different components. It will take a while before the new standard will fully replace the previous version. However, taking into account the new possibilities which OpenMI 2.0 opens to the developers, we hope that this paper stimulates the migration of existing components to the new standard and the development of new OpenMI components.

## ACKNOWLEDGEMENT

The development of OpenMI version 2.0 has been funded by the members of the OpenMI Association as well as the European Commission through various 6th Framework Programs and the LIFE Environment program.

## REFERENCES

Gamma Erich et al., Design Patterns: Elements of Reusable Object-Oriented Software *Addison-Wesley Professional* (ISBN 0-201-63361-2), 1995

Gijsbers P., From OpenMI 1.4 to 2.0. *iEMSS 2010, in review*. 2010

Goodall, J. L.; Robinson, B. F.; Shatnawi, F. M. and Castronova, A. M. Linking hydrologic models and data: The OpenMI approach, *American Geophysical Union, Fall Meeting 2007, abstract #H13H-1682*, 2007

Knapen, M.J.R.; Athanasiadis, I.N.; Jonsson, B.; Huber, D.; Wien, J.J.F.; Rizzoli, A.E.; Janssen, S. (2009) Use of OpenMI in SEAMLESS *AgSAP proceedings*, 2009