# OpenMI Association Technical Committee meeting no 30

See also: OATC Wiki Home

Date: June 25, 2010
Time: 10:00 - 11:30 am
Venue: Skype Conference Call
Topic: Extendable version of OpenMI 2 (continued)

## Table of contents



## Participants


Rob Knapen, Alterra, Wageningen UR (Rob.Knapen@wur.nl)
Standa Vanecek, DHI (s.vanecek@dhi.cz)
Adrian Harper, Wallingford Software (adrian.harper@wallingfordsoftware.com)
Stef Hummel, Deltares (stef.hummel@deltares.nl)
Unknown User (don), Deltares (gennadii.donchyts@deltares.nl)
Jesper Grooss, DHI (jgr@dhigroup.com)
Peter Schade, Bundesanstalt fuer Wasserbau (peter.schade(at)baw.de)


## 1. Progress towards OpenMI 2 Beta release

### 1.1. Source code update status - C#

- Merging extendable branch into trunk 👍

The trunk has been tagged as `/svnroot/openmi/tags/OpenMI-2.0.0-20100618-BeforeExtendable`, before any of the extendable changes have been applied.

The OpenMI-2.0.0-extendable branch has been merged into the trunk. This means that the OpenMI-2.0.0-extendable branch should no longer be used (and eventually deleted).

- Add initialize method to IAdaptedOutput 👍
- IIdentifiable : IDescribable 👍 (still missing some cleanup)
- Moving exchange item event to IBaseExchangeItem 👍
- Remove GetHashCode() and Equals() overrides in Backbone package ❓
- Add missing value definition to IUnit - called `MissingDataValue` 👍
- Update the IQuality interface 👍⚠️
- Remove loop approach 👍
- Documentation of IElementSet and ITimeSet when space and/or time has not meaning
- IValueSet.ElementCount and IValueSet.TimesCount to be removed 👍
- Naming of new interfaces (include base in their names) 👍
- Add property to indicate lengths of the dimensions to IValueSet 👍
- Add ValueType to IValueSet 👍
- Refine adapted output factories (removing awkward GetAdaptedOutputDescription) 👍
- Introducing ISpatialAxis into standard. 👍 Fixed SDK by the use of C# extending methods.

Stef will check whether indeed all changes have been processed.

### Still missing update of source code documentation.

#### Add property to indicate lengths of the dimensions to IValueSet

Currently the text is for `GetIndexCount(int[] indices)` are:

```
/// <summary>
/// Returns the length (max index count) of the dimension specified by the
/// given indices. To get the size of the first dimension, use a zero-length
/// integer array as input argument. Length of indices must be a least one
/// smaller than the <see cref="NumberOfIndexes"/>
/// </summary>
/// <param name="indices">indexes of the dimension to get the length of</param>
/// <returns>length of the specified dimension</returns>
```

## 1.2. Documentation updates

(Peter) The current specifications define the ILinkableComponent, which is still time (Modified Julian Date) and space dependent.
Suggestion: Until 9. July mainly specify the IBaseLinkableComponent and a small general explanation of the Time+Space extension. The full Time+Space specification should follow after the sdk has been updated and tested.
(Stef) Maybe keep the documentation more or less as is, and only clearly describe the 'base' - 'timespace' leveling.
⚠️ To be decided next week.

# 2. Ongoing discussion topics

## 2.1. Note about OpenMI 2 compliancy

(Peter) To be decided: which compliancy is more appropriate?

(Rob) I prefer the second option. However besides stating when something is compliant, I think we should also make it more clear what this means in practice for the average modeler / user.

(Jesper) I would also vote for the second option. Though maybe rephrasing:

2. An OpenMI compliant component can also comply to one ore more extensions, by implementing the IBaseLinkableComponent interface and the extension interfaces which it wishes to comply to, according to the specifications provided as comments in the OpenMI.Standard2 source code.

### 2.1.1. Slightly changed from 1.4 compliancy

The IBaseLinkableComponent is the key interface in the OpenMI standard version 2. Any OpenMI compliant component must implement IBaseLinkableComponent.
OpenMI-compliance definition:

1. An OpenMI-compliant component must implement the IBaseLinkableComponent interface according to specifications provided as comments in the OpenMI.Standard2 interface source code. *
2. An OpenMI-compliant component must, when compiled, reference the OpenMI.Standard2.dll, which is released and compiled by the OpenMI Association.
3. An OpenMI-compliant component must be associated with an XML file, which complies to (can be validated with) the BaseLinkableComponent.xsd schema (same information a 4. - leave it out?)
4. An OpenMI-compliant component must be associated with an XML file, which complies to (can be validated with) the OpenMICompliancyInfo.xsd schema. This file must be submitted to the OpenMI Association.
5. The OpenMI Association provides two additional interfaces that OpenMI-compliant components may or may not implement: the IManageState interface and the IByteStateConverter interface. However, if these interfaces are implemented, each method and property must be implemented according to the comments given in the OpenMI.Standard2 interface source code.
6. The OpenMI Association's downloadable standard zip file provides the only recognized version of source files, XML schemas and assembly files.

* at 1. The OpenMI Association provides extensions to the standard with more specific interfaces, e.g. for time and space dependent components. Although these extensions are part of the standard they are formally not required for compliancy. But the OpenMI Association recommends to implement the appropriate extension for easier data exchange with other OpenMI compliant components.
This means a change in the implementation of the classical time and space dependent component, which had in 1.x just to implement the basic ILinkableComponent. For OpenMI 2 compliancy this component would have to implement the basic and the time and space specific extension interfaces.

### 2.1.2. Two level compliancy

The compliancy with the OpenMI 2 standard is split into two levels. The OpenMI compliancy on the first level refers to basic interfaces whereas the OpenMI extension-compliancy includes additional interfaces, e.g. for time and space dependent components.

The IBaseLinkableComponent is the key interface in the OpenMI standard version 2.

OpenMI-compliance definition:

1. An OpenMI-compliant component must implement the IBaseLinkableComponent interface according to the specifications provided as comments in the OpenMI.Standard2 interface source code.
2. An OpenMI extension-compliant component must implement the IBaseLinkableComponent interface and at least one of the extension interfaces according to the specifications provided as comments in the OpenMI.Standard2 interface source code. *
3. An OpenMI-compliant as well as an OpenMI extension-compliant component must, when compiled, reference the OpenMI.Standard2.dll, which is released and compiled by the OpenMI Association.
4. An OpenMI-compliant as well as an OpenMI extension-compliant component must be associated with an XML file, which complies to (can be validated with) the OpenMICompliancyInfo.xsd schema. This file must be submitted to the OpenMI Association.
5. The OpenMI Association provides two additional interfaces that OpenMI-compliant components may or may not implement: the IManageState interface and the IByteStateConverter interface. However, if these interfaces are implemented, each method and property must be implemented according to the comments given in the OpenMI.Standard2 interface source code.
6. The OpenMI Association's downloadable standard zip file provides the only recognized version of source files, XML schemas and assembly files.

* at 2.: This leads to a main difference between OpenMI 1.x and OpenMI 2. In order to become 1.x compliant the classical time and space dependent component just had to implement the basic ILinkableComponent. This component would have to implement the basic and the time and space specific extension interfaces for becoming OpenMI 2 compliant.

Decision 👍 : We choose for the second description, including the rephasing of Jesper

## 2.1.3. What does the compliancy definition mean for the user?

(Peter)
"Linking OpenMI 2 compliant components has become much more flexible than it used to be with OpenMI 1.x. Components with different concepts or in OpenMI language extensions can exchange their data. A component with the classical Modified Julian Day time definition can for example provide a consumer component, defining its simulation time by an index, with data. The provider would implement the time and space extension whereas the consumer could implement an extension for index-related dictionaries. The provider should contain an AdaptedOutput converter using both time definitions and thus being able to convert the time to the index value. More knowledge about the underlying extensions is required from the user in order to select the appropriate items and to check the plausibility of the results. This is the price being paid for higher flexibility, less due to the OpenMI but to the complexity of the task.
If both LinkableComponents implement the same OpenMI extension interface, their provider consumer relationship will remain as straight forward as with OpenMI 1.x, the conversions in AdaptedOutput will be easier to use.

When writing this documentation in Summer 2010, only the base interfaces and an extension for the classical time and space dependent components are available. Future extensions could support parallel computing, compliancy with the standards of the Open Geospatial Consortium OGC and much more.

One should remember that compliancy does not mean a pure plug and play connection. From the technical point of view most of the input and output of two LinkableComponents can be connected for data exchange. But neither the standard nor the related sdk offer too much methods for checking the correctness of the selected variables. The user has the responsibility for selecting the appropriate quantities and the locations where they are defined. The approved, but updated interface definitions as well as the extensions preserve the slimness of the OpenMI."

(Adrian)
"Potential users of an OpenMI compliant component should understand the following point

"Being compliant is no guarantee that it can be usefully linked to another OpenMI compliant component. It is a necessary but not sufficient condition."

To be usable the user must also consider the following factors

1. Do both components target the same system infrastructure. Currently Java and .Net components cannot be mixed within the same composition.

2. Do both components implement the same OpenMI version. Development is under way to allow 1.4 components to be loaded alongside version 2 components within version 2 of the configuration editor; however, currently this is not possible. Even when possible, there will still be restrictions as version 2 of the standard exposes new (additional) data exchange concepts which are incompatible with version 1.4

3.Do both components provide and consume all the required data interactions for the underlying physics for the two models interaction. Breaking the underlying physics will cause instabilities (at best) or wrong answers (at worst). The underlying issue here is that OpenMI provides the means to couple models together, but cannot make judgements on the advisability of doing so. The responsibility for this lies with the user; it is expected that they should have competence in the use and application of both models.

4. Are all required output adapters implementations compatible. Version 2 allows for adapters tailored for specific applications and hence some combinations could be invalid. Whilst it can be reasonably expected that adapter implementations from the same software source (SDK) should either be compatible or create sensible runtime warnings if not (if developed well). This is unlikely to be the case when mixing adapters from different software sources. Hence, when using output adapters care should be taken to understand their implementation details."

(Peter at Adrian):
All four points are very true.
I'd suggest to add 1. and 2. to the compliancy info
and to put 3. and 4. into the StandardSpecificationOpenMI_20.pdf,
Chapter "2.8. Assumptions underlying the OpenMI architecture".

Decision 👍 : we will move 4. to the assumptions underlying the OpenMI architecture.

## 2.1.4. What does the compliancy definition mean for an application developer?

todo, after 2.2. has been decided

## 2.2. How to define the extension interfaces

(Rob) Still considering the two approaches: using a hierarchy (i.e. ITimeSpaceLinkableComponent extends IBaseLinkableComponent, MyComponent implements ITimeSpaceLinkableComponent) or composition (i.e. MyComponent implements ILinkableComponent, ILinkableComponentTimeSpaceExtension). Should consider the effect when adding a linkable component interface for the loop method(s) or for the FRAMES dictionary support. Also see how OGC spatial element sets can be added as an extension and how this effects the interfaces.

(Jesper) I have added a third alternative, which is somewhat in between. The more I look at it, the more I prefer alternative 2.

(Peter) The second alternative has the advantage that a developer sees clearer what is implemented in e.g. MyParallelTimeSpaceComponent. Alternative 1 and 3 mean that he has to know or to look up that IBaseLinkableComponent is included. (Shouldn't it be IBaseLinkableComponent in the first line of alt. 2?)
On the other hand, if an extension has a problem with a method of ILinkableComponent which is not part of the extension, will it be allowed to overwrite it? In this case the second alternative would mean two different versions of the method, the original one and the overwritten one.

Alternative 1 (interface hierarchy):

```
IBaseLinkableComponent { base methods };
ITimeSpaceLinkableComponent extends IBaseLinkableComponent { time-space methods };
IParallelTimeSpaceLinkableComponent extends ITimeSpaceLinkableComponent { loop methods };

IDictionaryLinkableComponent extends IBaseLinkableComponent { dictionary methods };
IParallelDictionaryLinkableComponent extends IDictionaryLinkableComponent { loop methods };

MyParallelTimeSpaceComponent implements IParallelTimeSpaceLinkableComponent {};
MyParallelDictionaryComponent implements IParallelDictionaryLinkableComponent {};
```

Alternative 2 (uncoupled interfaces):

```
ILinkableComponent { base methods };
ILinkableComponentTimeSpaceExtension { time-space methods };
ILinkableComponentParallelExtension { loop methods };
ILinkableComponentDictionaryExtension { dictionary methods };

MyParallelTimeSpaceComponent implements ILinkableComponent, ILinkableComponentTimeSpaceExtension,
ILinkableComponentParallelExtension {};
MyParallelDictionaryComponent implements ILinkableComponent, ILinkableComponentDictionaryExtension,
ILinkableComponentParallelExtension {};
```

Alternative 3 (in between):

```
IBaseLinkableComponent { base methods };
ITimeSpaceLinkableComponent extends IBaseLinkableComponent { time-space methods };
IParallelLinkableComponent extends IBaseLinkableComponent { loop methods };
IDictionaryLinkableComponent extends IBaseLinkableComponent { dictionary methods };

MyParallelTimeSpaceComponent implements ITimeSpaceLinkableComponent, IParallelLinkableComponent {};
MyParallelDictionaryComponent implements IDictionaryLinkableComponent, IParallelLinkableComponent {};
```

Decision 👍 : We keep the exchange items as an hierarchy, and will use alternative 2 for the linkable components, together with combined empty interfaces mentioned at the previous meeting:

```
ITimeSpaceLinkableComponent : IComponent, ITimeSpaceLinkableComponent {};
```

# 3. New discussion topics

## 3.1. How to package and distribute extensions

(Rob) When an extension is approved by OATC will it be included into the Standard library (dll or jar), or will each extension be distributed in its own library?

(Peter) To keep it in the standard library will make the check on compliancy easier, s. 2.1.2. saying compliant models have to reference OpenMI2.Standard. dll/jar. One argument pro an own library is that new extensions can be more frequently updated than the standard. Anyway, I guess the OAEC has to decide formally about new extensions as it did about compliancy of models in the past.

(Jesper) I believe (at least for C#, I am not sure with java) when we first have released the OpenMI2.Standard.dll, then there are issues in case this dll is updated to include e.g. a dictionary extension: Two different OpenMI2.Standard.dll files having the same version number, but different content. It would be much more safe to release an OpenMI2.Standard.DictionaryExtension.dll, and leaving the original content in the OpenMI.Standard.dll. It may be possible to put it all in a new standard dll, however that requires "intelligent" installers: Assume I am to install a TimeSpace engine and a Dictionary engine, which come with each their version of the OpenMI2.Standard.dll that they want to install and register in the GAC. Depending on the order in which I install the engines, each installer needs to know whether they are to reregister their version of the OpenMI2.Standard.dll in the GAC, to assure that the newest version is the one registered.

## 3.2. Events in the TimeSpaceExchangeItem

(Jesper) It was discussed in the last meeting to add extra events to the TimeSpaceExchange item to keep track of when the timeset and elementset changed. I would suggest to put these directly on the ITimeSet and ISpatialAxis/IElementSet instead. Especially, for the ISpatialAxis/IElementSet it could also be nice with a property `bool IsDynamic`, to indicate wether the ISpatialAxis/IElementSet can change at all.

Decision 👎 : Could lead to a managemanagement problem (keeping track of all listeners).

## 3.3. Revisit discussions left out from meeting 28

There is a number of discussions there, that has not yet been decided on.

## 3.4. Should an ITimeSpaceLinkableComponent override some of the IBaseLinkableComponent methods:

(Jesper) Currently an ITimeSpaceLinkableComponent provides its inputs and outputs as IBaseInput and IBaseOutput. Are these always transferable to ITimeSpaceInput and ITimeSpaceOutput, or could there be cases where other types of inputs and outputs are suitable? Would we even allow that? It could be considered to add methods to the ITimeSpaceLinkableComponent like:

```
public interface ITimeSpaceComponent : IBaseLinkableComponent
{
  new IList<ITimeSpaceInput> InputItems { get; }
  new IList<ITimeSpaceOutput> OutputItems { get; }
}
```

That would require all inputs and outputs to be of type ITimeSpaceInput and ITimeSpaceOutput. I guess that would make it difficult to make an ITimeSpaceDictionaryComponent, combining the TimeSpace and the Dictionary extension, since they would each require a different type of their inputs and outputs lists.

Decision 👎 : Avoid too specialized components.