OpenMI: A glue for model integration

[1]Gijsbers, P.J.A. and [2]J.B..Gregersen

[1]WL | Delft Hydraulics, [2]DHI Water & Environment, E-Mail: peter.gijsbers@wldelft.nl

Keywords: *OpenMI, model integration, data exchange, interface specification.*

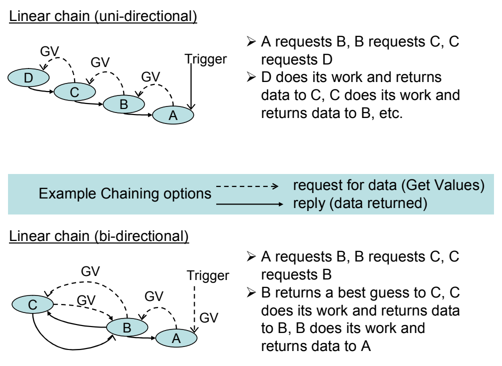
EXTENDED ABSTRACT

Management issues in many sectors of society demand for integrated analysis, which can be supported by integrated modelling. Since the all-inclusive modelling software is difficult to achieve, and possibly even undesirable, integrated modelling will require the linkage of individual model or model components that address specific domains. Emerging from the water sector, OpenMI has been developed with the purpose of being the glue which can link legacy and non-legacy model components from various origins together. OpenMI provides a standardized *interface* to define, describe and transfer data on a time basis between software components that run simultaneously. This paper presents the technical concepts of OpenMI as well as a nearly complete interface specification (see HarmonIT, 2005).

Essential concepts within OpenMI are the distinction of quantities, element sets, time and values. Elements can be non-geo-referenced or geo-referenced in 0/1/2/3D. The entities are supported by meta-data interfaces describing what the data represent, to which location it refers, for which time (time stamp or time span) they are valid and how they are produced.

The developers of OpenMI created a full software implementation, called the OpenMI environment, in C# (.NET) and a less extensive one in Java (under construction). The focus of development was primarily oriented to data exchange issues. Hence a wide range of utilities is provided to enhance the implementation of the OpenMI interfaces from typically legacy code. E.g. a wrapper package provides facilities for book keeping of links and data handling such as buffering and spatial and temporal mapping. Simple tools are available to define links, to run the system and to display results.

OpenMI is based on the request-reply architecture concept. The basic workflow is the following. At configuration time, meta data of a component is inspected to identify and define the links between the components. At run-time a component is requested by another component for values at a specific time and location (element set) using the GetValues-function call (see Figure 1). This providing component is obliged to reply to this request and provide the data at the time and location as requested. In order to reply, he may need to do internal computations to progress in time. This computation may require external data that can be requested from other components by a GetValues-call. Once the internal time progressed at or past the requested time, data transformations can be applied to return the values at the exact time, element set and units as requested.


Linear chain (uni-directional)
GV
GV
GV
Trigger
D
C
B
A
Example Chaining options
request for data (Get Values)
reply (data returned)
Linear chain (bi-directional)
GV
GV
GV
GV
Trigger
C
B
A

- A requests B, B requests C, C requests D
- D does its work and returns data to C, C does its work and returns data to B, etc.

- A requests B, B requests C, C requests B
- B returns a best guess to C, C does its work and returns data to B, B does its work and returns data to A

Figure 1 The basis of OpenMI: the request-reply concept

The interface orientation of OpenMI does not prescribe the use of the OpenMI environment. Hence OpenMI can also be used to glue model engines with existing modelling frameworks through the OpenMI interface. OpenMI is expected to satisfy the modeling requirements of a wide group of users in various engineering domains, such as model coders, model developers, data managers and end users. The expected impacts are the simplification of the model linking process, the ability to represent feedback loops and process interactions, the establishment of a communication standard for modelling and a reduction in development time for decision support systems.

The architecture for OpenMI is fully documented and available on www.openmi.org. The open source implementation is available on sourceforge.net\projects\openmi. Other papers in this conference provide case studies. This paper discusses the technical aspects of the OpenMI interface specification. .

1. INTRODUCTION

Integrated Water Management requires an understanding of catchment processes and the ability to predict how they will respond under different management policies. Most traditional modelling systems have not been able meet these requirements as models have tended to represent individual processes and have been run independently. Hence, their output may not reflect the interactions between different aspects of the environment. Clearly, it is not practical to construct a single model that could simulate all catchment processes and to do so would be wasteful of the large number of existing models. A better solution is to couple models and hence enable them to exchange data as they run, thus allowing interactions to be represented in the simulation.

In response to the need created by the Water Framework Directive from its introduction of integrated water management, the HarmonIT project has developed the Open Modelling Interface and Environment (the OpenMI) to allow models to exchange data. This has been developed with the following objectives in mind: (i) the standard should be applicable to new and existing models, requiring the minimum of change to the program code; (ii) the standard should impose as few restrictions as possible on the freedom of the model developer; (iii) the standard should be applicable to most, if not all, time-based simulation techniques; and (iv) implementation of the standard should not unreasonably degrade performance.

The majority of model applications that are the result of a design process (as opposed to those that just evolved over time) have a common structure comprising a user interface, input files, a calculation engine and output files. Typically, the *user interface* enables the user to create or point to *input* file*s* and allows the visualization of the *output* or *result* files. The calculation *engine* contains the model algorithms and becomes a *model* of a specific process, e.g. flow in the Rhine, once it has populated itself by reading the input files. A model can compute output. If an engine can be instantiated separately, it is an *engine component*. If, further, it supports a well defined interface, it becomes a *component.* Finally, if the engine supports the OpenMI Linkable component interface, then the engine is said to be *OpenMI-compliant*. An engine component populated with data is a *model component.*

The OpenMI focuses on the run-time exchange of data between populated components. These components can be data providers (e.g. models, databases, text files, spreadsheets, pre/post-processors, data editors, in-situ monitoring stations etc.) and/or data acceptors (e.g. models or on-line visualization tools). Thus, the OpenMI potentially allows the development of a complete integrated modelling system consisting of GUI, engines and databases. When this level of system integration will be achieved, depends on the adoption of OpenMI as *a component linkage standard* by the environmental model and software development community.

At the start of HarmonIT, the components were foreseen as running within a framework, but gradually this concept was replaced by standardizing the run-time interface of linkable components, thus allowing direct communication between components. The linkable component interface, described in the namespace org.OpenMI.standard, can be implemented in a variety of ways, of which the OpenMI environment, developed by the HarmonIT project, is just one.

This paper presents in an overview of the technical details of the OpenMI standard. It also briefly describes how existing modelling software can be made OpenMI-compliant. The full specification can be downloaded from the OpenMI-website www.openmi.org or can be downloaded from the documentation section of the OpenMI environment software installation package (incl. source code) available from the CVS-repository on sourceforge.net/projects/openmi.

2. THE OPENMI STANDARD

2.1. The OpenMI: a request & reply architecture

OpenMI is based on the ‘request & reply’ mechanism. It consists of communicating components (source and target components) which exchange data in a pre-defined way and in a pre-defined format. The OpenMI defines both the component interfaces as well as how the data is to be exchanged. The components are called linkable components to indicate that the OpenMI involves components that can be linked together.

From the data exchange perspective, the 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. A component at the end of the component chain triggers the data process. 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. No overall controlling functionality is needed to guide time synchronisation. Sometimes a local controller is needed to control convergence of data being exchange

2.2. Addressing general linkage issues

The OpenMI standard provides the following facilities for model linkage:

- *Data definition*: The base data model of the OpenMI describes the numerical values to be exchanged in terms of quantities (what), element sets (where), times (when), and data operations (how). (For details, see Figure 5)

- *Meta data defining potentially exchangeable data*: Quantities, elements sets and data operations are combined in exchange item definitions to indicate what data can potentially be provided and accepted by a linkable component (For details, see Figure 6)

- *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. (For details, see Figure 6)

- *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 a single method, the GetValues()-method. Dependent on the status of the source component, a call of the GetValues() method may trigger computation and , possibly, lead to further requests for data from other components. An important feature is the obligation that components always deal with requests in order of receipt.

- *Generic component access*: All functionality becomes available to other components through one base interface, the linkable component interface (for details, see Figure 6). This interface needs to be implemented by a component for it to become OpenMI-complaint. Two optional interfaces have been defined to extend its functionality with respect to discrete time information and state management (for details, see Figure 6). 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, the assembly hosting the class and the arguments needed for initialization.

- *Event mechanism*: A lightweight event mechanism has been introduced (see Figure 6) to pass messages for 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 is based on the Exception-class as provided by the development environment.

2.3. Utilization and deployment phases

An OpenMI linkable component provides a variety of services which can be utilized in various phases of deployment. Figure 2 provides an overview of the phases that can be identified, and the methods which might be invoked at each phase. While the sequence of phases is prescribed, the sequence of calls within each phase is not prescribed. The phases are:

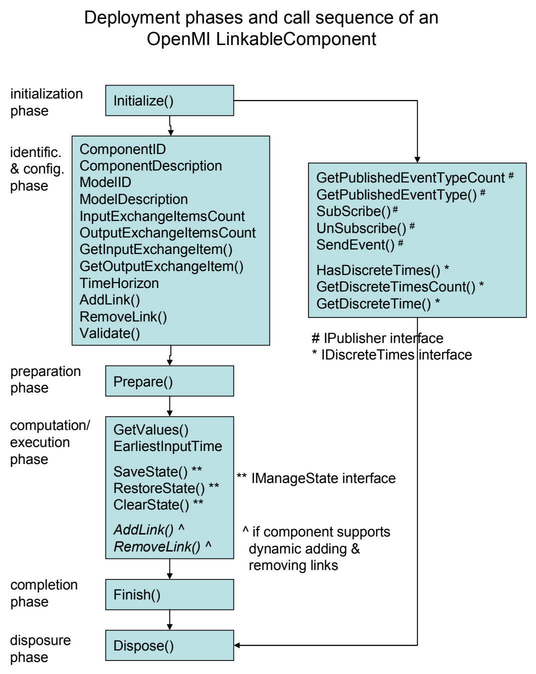


Figure 2 Call phasing of Linkable Component

1. *Initialization*: This phase ends when a linkable component has sufficient knowledge to populate itself with data and expose its exchange items.

2. *Inspection & Configuration*: At the end of the phase, the links have been defined and created and the component has validated its status.

3. *Preparation*: This phase enables you to prepare all conversion matrices before the computation/data retrieval process starts.

4. *Computation / execution*: During this phase, the main code of the component is executed. Typically, it represents the simulation of a process such as river flow. This simulation may itself generate calls to other components.

5. *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.

6. *Disposal*: 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.

Linkable components may support the dynamic addition and removal of links at computation time. However, as described in the grouping of deployment phases, this requirement is not enforced. Those who do not support this call at computation-time should throw an exception.

2.4. Other features

In principle, the GetValues()-call stack of all linkable components is located 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 when to instantiate and populate the engines. Exchange item information can be obtained from the engine, from files being parsed during the inspection phase or by dynamic querying of linked components.

3. MEETING THIS SPECIFICATION

3.1. Consequences of the OpenMI for a model

The OpenMI enables model engines to compute and exchange data at their own time step, 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 steps, or it might have to progress its own calculation, or it might have to estimate via interpolation or extrapolation. If the component is not able to provide all the requested data, an exception will be raised. 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 (i) be able to expose information (what, where) to the outside world on the modelled variables that it can provide, or which it is able to accept; (ii) submit to run-time control by an outside entity; (iii) be structured in that initialization is separated from computation - boundary conditions must be collected in the computation phase and not during initialization; (iv) be able to provide the values of the modelled variables for the requested times and locations; (v) be able to respond to a request, even when the component itself is time independent; and if the response requires data from another component, the component should be able to pass on the time as well in its own request; (vi) flag missing values and (vii) in the exceptional case that an entire value set is unavailable, throw an exception. Be aware that such exception will stop the entire computation process and thus should be prevented.

3.2. Model wrapping

The above mentioned requirements do not match the nature of most legacy codes, but most of it can be captured by wrapping. Figure 3 illustrates the OpenMI wrapping pattern that is adopted in the implementation of the OpenMI environment.

The wrapper uses an internal interface to access the engine. This interface, IEngine, may be split in a section addressing the meta-data issues and in run-time section for the numerical computation and associated data exchange. For the computational part, this wrapping concept still requires that engine cores are re-engineered according to the pattern as illustrated in Figure 4.

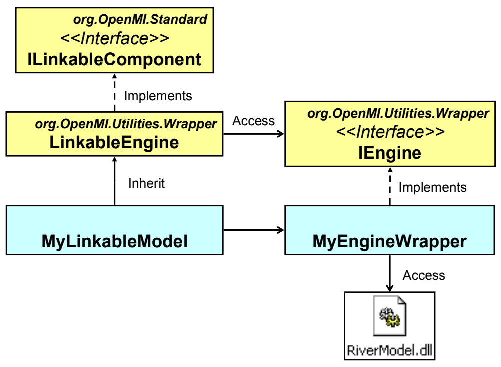


Figure 3 The engine wrapping pattern

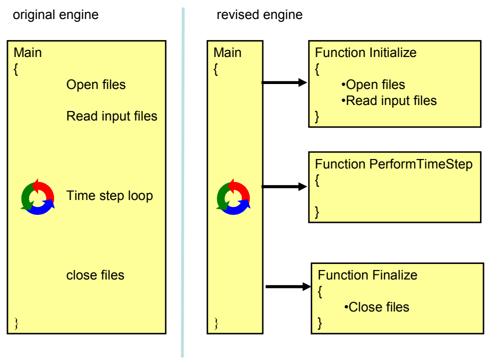


Figure 4 Engine core reengineering pattern

3.3. A glue for other frameworks ?

The OpenMI is designed in a way that components have direct communication at the component level via standardized interfaces. Components thus are not required to adopt the OpenMI environment. As already identified by Argent and Rizolli (2004), the OpenMI might also play a role as glue for interoperability between frameworks. The authors indicate that each framework would separately have to contain the methods and classes necessary to use the OpenMI, as well as components, created as discrete objects with published interfaces, which have been made compatible with the OpenMI. The most difficult task is assuring similar semantics of the data exchange between components.

3.4. Ensuring similar semantics

Ensuring the sound semantics of model linkages is a crucial aspect in model integration, no matter the origin of the codes. Dudley et al. (2005) illustrates issues that may arise when linking water quality models from various vendors. The OpenMI development team has, on purpose, allocated an important role to the human expert to ensure that semantically similar entities are connected in a scientific sound way. The interface specification of the OpenMI can help via its meta data tags on quantity units, dimensions etc., but explicit descriptions remain most valuable for this activity.

4. OPENMI NOW AND IN THE FUTURE

The OpenMI is currently being applied by a range of software developers in the catchment domain (sewers, open channels, hydrology, groundwater, waste water treatment, water quality, socio-economy) and is starting to be adopted by the estuarine and marine domain. The concepts of the OpenMI have shown to be very powerful, but the experiences also have indicated the desire for further refinement of the interfaces to support a persistent state (to enable hot starts); to improve performance; to improve the element set definition for topology and the exact positioning of the data value; to enable clustering of quantities; to address component to component connections in relation to the link definition and to simplify the implementation of the interfaces.

It is foreseen that an update of the OpenMI is desirable to obtain a mature standard. A thread is started on the sourceforge.net/projects/openmi forum to stimulate the development discussion on the OpenMI.

In the mean time, the core partners of the HarmonIT consortium have started initiatives to stimulate business development with the OpenMI and to setup a large scale demonstration project for the Water Framework Directive Implementation Strategy. Furthermore they have committed themselves to set up an open organisation (the OpenMI Association) that will maintain the OpenMI and stimulate its development and uptake by the modelling software community.

5. ACKNOWLEDGMENTS

The authors would like to thank the EU for co-funding the HarmonIT-IT Frameworks project (Contract no: EVK1-CT-2001-00090) that led to the development of the OpenMI.

6. REFERENCES

Argent R.M. and A.E.Rizolli (2004), Development of Multi-Framework Components. iEMSs conference, Osnabrŭck, Germany, p.365-370

Dudley, J, et al. (2005) Applying the Open Modelling Interface (OpenMI), this conference

HarmonIT (2005), The org.OpenMI.Standard interface specification. Part C of the OpenMI Document Series. IT Frameworks (HarmonIT) ECFP5 Contract EVK1CT200100090

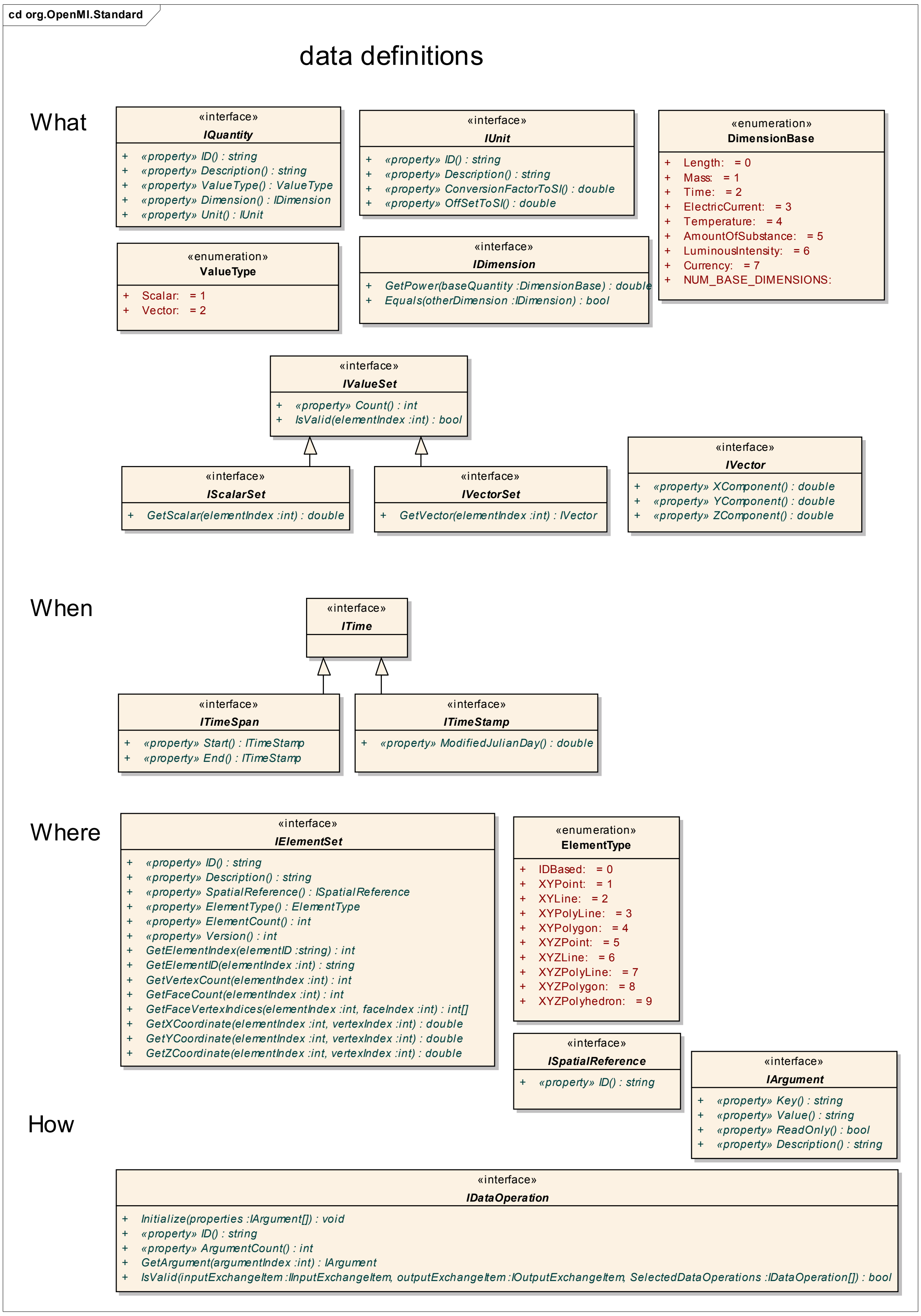

cd org.OpenMI.Standard
data definitions
What
«interface» IQuantity
+ «property» ID() : string
+ «property» Description() : string
+ «property» ValueType() : ValueType
+ «property» Dimension() : IDimension
+ «property» Unit() : IUnit
«interface» IUnit
+ «property» ID() : string
+ «property» Description() : string
+ «property» ConversionFactorToSI() : double
+ «property» OffSetToSI() : double
«enumeration» DimensionBase
+ Length: = 0
+ Mass: = 1
+ Time: = 2
+ ElectricCurrent: = 3
+ Temperature: = 4
+ AmountOfSubstance: = 5
+ LuminousIntensity: = 6
+ Currency: = 7
+ NUM_BASE_DIMENSIONS:
«enumeration» ValueType
+ Scalar: = 1
+ Vector: = 2
«interface» IDimension
+ GetPower(baseQuantity :DimensionBase) : double
+ Equals(otherDimension :IDimension) : bool
«interface» IValueSet
+ «property» Count() : int
+ IsValid(elementIndex :int) : bool
«interface» IScalarSet
+ GetScalar(elementIndex :int) : double
«interface» IVectorSet
+ GetVector(elementIndex :int) : IVector
«interface» IVector
+ «property» XComponent() : double
+ «property» YComponent() : double
+ «property» ZComponent() : double
When
«interface» ITime
«interface» ITimeSpan
+ «property» Start() : ITimeStamp
+ «property» End() : ITimeStamp
«interface» ITimeStamp
+ «property» ModifiedJulianDay() : double
Where
«interface» IElementSet
+ «property» ID() : string
+ «property» Description() : string
+ «property» SpatialReference() : ISpatialReference
+ «property» ElementType() : ElementType
+ «property» ElementCount() : int
+ «property» Version() : int
+ GetElementIndex(elementID :string) : int
+ GetElementID(elementIndex :int) : string
+ GetVertexCount(elementIndex :int) : int
+ GetFaceCount(elementIndex :int) : int
+ GetFaceVertexIndices(elementIndex :int, faceIndex :int) : int[]
+ GetXCoordinate(elementIndex :int, vertexIndex :int) : double
+ GetYCoordinate(elementIndex :int, vertexIndex :int) : double
+ GetZCoordinate(elementIndex :int, vertexIndex :int) : double
«enumeration» ElementType
+ IDBased: = 0
+ XYPoint: = 1
+ XYLine: = 2
+ XYPolyLine: = 3
+ XYPolygon: = 4
+ XYZPoint: = 5
+ XYZLine: = 6
+ XYZPolyLine: = 7
+ XYZPolygon: = 8
+ XYZPolyhedron: = 9
«interface» ISpatialReference
+ «property» ID() : string
«interface» IArgument
+ «property» Key() : string
+ «property» Value() : string
+ «property» ReadOnly() : bool
+ «property» Description() : string
How
«interface» IDataOperation
+ Initialize(properties :IArgument[]) : void
+ «property» ID() : string
+ «property» ArgumentCount() : int
+ GetArgument(argumentIndex :int) : IArgument
+ IsValid(inputExchangeItem :IInputExchangeItem, outputExchangeItem :IOutputExchangeItem, SelectedDataOperations :IDataOperation[]) : bool

Figure 5 OpenMI interface specification details defining the data model underlying OpenMI

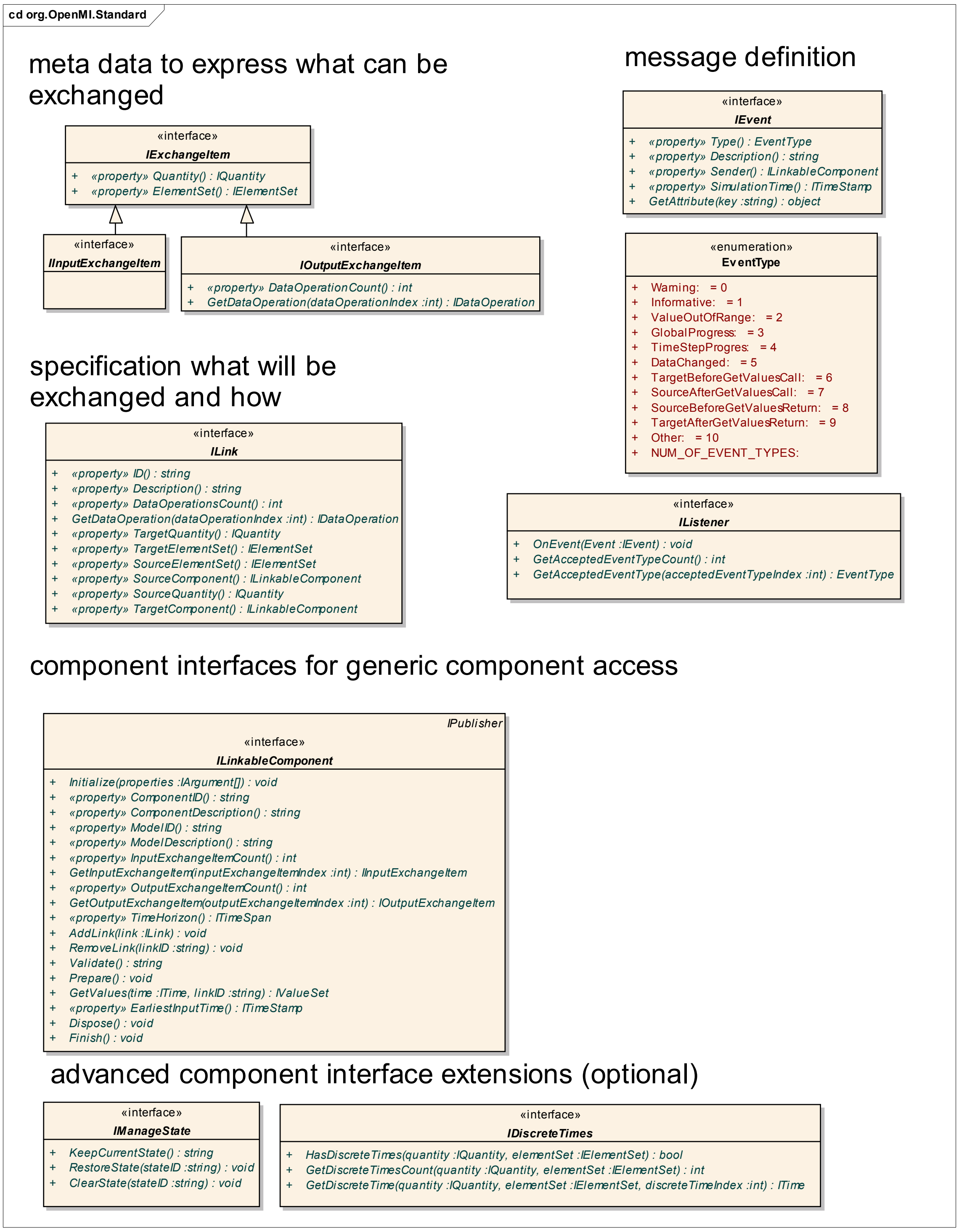

cd org.OpenMI.Standard
meta data to express what can be exchanged
«interface»
IExchangeItem
+ «property» Quantity() : IQuantity
+ «property» ElementSet() : IElementSet
«interface»
IInputExchangeItem
«interface»
IOutputExchangeItem
+ «property» DataOperationCount() : int
+ GetDataOperation(dataOperationIndex :int) : IDataOperation
message definition
«interface»
IEvent
+ «property» Type() : EventType
+ «property» Description() : string
+ «property» Sender() : ILinkableComponent
+ «property» SimulationTime() : ITimeStamp
+ GetAttribute(key :string) : object
«enumeration»
EventType
+ Warning: = 0
+ Informative: = 1
+ ValueOutOfRange: = 2
+ GlobalProgress: = 3
+ TimeStepProgres: = 4
+ DataChanged: = 5
+ TargetBeforeGetValuesCall: = 6
+ SourceAfterGetValuesCall: = 7
+ SourceBeforeGetValuesReturn: = 8
+ TargetAfterGetValuesReturn: = 9
+ Other: = 10
+ NUM_OF_EVENT_TYPES:
specification what will be exchanged and how
«interface»
ILink
+ «property» ID() : string
+ «property» Description() : string
+ «property» DataOperationsCount() : int
+ GetDataOperation(dataOperationIndex :int) : IDataOperation
+ «property» TargetQuantity() : IQuantity
+ «property» TargetElementSet() : IElementSet
+ «property» SourceElementSet() : IElementSet
+ «property» SourceComponent() : ILinkableComponent
+ «property» SourceQuantity() : IQuantity
+ «property» TargetComponent() : ILinkableComponent
«interface»
IListener
+ OnEvent(Event :IEvent) : void
+ GetAcceptedEventTypeCount() : int
+ GetAcceptedEventType(acceptedEventTypeIndex :int) : EventType
component interfaces for generic component access
IPublisher
«interface»
ILinkableComponent
+ Initialize(properties :IArgument[]) : void
+ «property» ComponentID() : string
+ «property» ComponentDescription() : string
+ «property» ModelID() : string
+ «property» ModelDescription() : string
+ «property» InputExchangeItemCount() : int
+ GetInputExchangeItem(inputExchangeItemIndex :int) : IInputExchangeItem
+ «property» OutputExchangeItemCount() : int
+ GetOutputExchangeItem(outputExchangeItemIndex :int) : IOutputExchangeItem
+ «property» TimeHorizon() : ITimeSpan
+ AddLink(link :ILink) : void
+ RemoveLink(linkID :string) : void
+ Validate() : string
+ Prepare() : void
+ GetValues(time :ITime, linkID :string) : IValueSet
+ «property» EarliestInputTime() : ITimeStamp
+ Dispose() : void
+ Finish() : void
advanced component interface extensions (optional)
«interface»
IManageState
+ KeepCurrentState() : string
+ RestoreState(stateID :string) : void
+ ClearState(stateID :string) : void
«interface»
IDiscreteTimes
+ HasDiscreteTimes(quantity :IQuantity, elementSet :IElementSet) : bool
+ GetDiscreteTimesCount(quantity :IQuantity, elementSet :IElementSet) : int
+ GetDiscreteTime(quantity :IQuantity, elementSet :IElementSet, discreteTimeIndex :int) : ITime

Figure 6 OpenMI interface specification details for component access and linking