The OpenMI Document Series

Part F - org.OpenMI.Utilities technical documentation

For OpenMI (Version 1.4)

Title	OpenMI Document Series: Part F - org.OpenMI.Utilities technical documentation for the OpenMI (version 1.4)
Editor	Jan Gregersen, LicTek ApS on behalf of DHI Water and Environment, Hørsholm, Denmark Peter Sinding, DHI Water and Environment, Hørsholm, Denmark
Authors	Peter Gijsbers, WL | Delft Hydraulics, Delft, The Netherlands Stefan Westen, Wallingford Software Ltd, Wallingford, UK Rob Brinkman, WL | Delft Hydraulics, Delft, The Netherlands Jan Curn, DHI Hydroinform, Prague, Czech Republic
Document production	Peter Gijsbers, WL | Delft Hydraulics, Delft, The Netherlands
Current version	V1.4
Date	21/05/2007
Status	Final © The OpenMI Association
Copyright	All methodologies, ideas and proposals in this document are the copyright of the OpenMI Association. These methodologies, ideas and proposals may not be used to change or improve the specification of any project to which this document relates, to modify an existing project or to initiate a new project, without first obtaining written approval from the OpenMI Association who own the particular methodologies, ideas and proposals involved.
Acknowledgement	This document has been produced as part of the OpenMI-Life project. The OpenMI-Life project is supported by the european Commission under the Life Programme and contributing to the implementation of the thematic component LIFE-Environment under the policy area "Sustainable management of ground water and surface water managment" Contract no : LIFE06 ENV/UK/000409. The first version of this document has been produced as part of the HarmonIT project; a research project supported by the European Commission under the Fifth Framework Programme and contributing to the implementation of the Key Action “Sustainable Management and Quality of Water” within the Energy, Environment and Sustainable Development. Contract no: EVK1-CT-2001-00090.

Preface

OpenMI stands for Open Modeling Interface and aims to deliver a standardized way of linking of environmental related models. This document describes the wrapping and other utilities as being provided in the OpenMI Software Development Kit. It is the sixth document in the OpenMI report series, which specifies the OpenMI interface standard, provides guidelines on its use and describes software facilities for migrating, setting up and running linked models.

Other titles in the series include:

A. Scope

B. Guidelines

C. org.OpenMI.Standard interface specification

D. org.OpenMI.Backbone technical documentation

E. org.OpenMI.Development Support technical documentation

F. org.OpenMI.Utilities technical documentation (this document)

The interface specification is intended primarily for developers. For a more general overview of the OpenMI, see Part A (Scope).

The official reference to this document is:

OpenMI Association (2007) *The org.OpenMI.Utilities technical documentation*. Part F of the OpenMI Document Series

Disclaimer

Further information

Further information on the OpenMI Association and the Open Modelling Interface can be found on http://www.OpenMI.org.

Table of contents

1 Introduction

1.1 Background

OpenMI stands for Open Modeling Interfaces. It aims to deliver a standardized way to link environmental related computational models that run simultaneously. 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. It thus enables process interaction being represented more accurately, compared to sequential linkages.

The establishment of OpenMI will support and assist the scientific, consultancy and water management community in the integrated assessment of water management systems and thus strategic planning and integrated catchment management required by the European Water Framework Directive.

This standardized way of linking models is achieved by an intelligent protocol to describe, define and transfer data. This protocol is translated into a strict set of rules, i.e. formal interfaces, to be implemented by software code in the org.OpenMI.Standard namespace. Any component that utilizes this namespace is called an OpenMI compliant component.

Europe has a huge number of existing models which it is neither feasible nor desirable to rewrite. Therefore, a primary design objective of the OpenMI is that the cost, skill and time required to migrate an existing model to the standard should not be a deterrent to its use. To achieve this objective, a software layer has been designed with a number of utilities that supports the encapsulation of model components. This software layer is part of the OpenMI Software Development Kit. Application of this SDK is not mandatory, but it may save costs and effort to join. This report provides the technical documentation of these utilities, combined in the org.OpenMI.Utilities namespace. The relation with other namespaces in the OpenMI architecture is displayed in Figure 1-1.

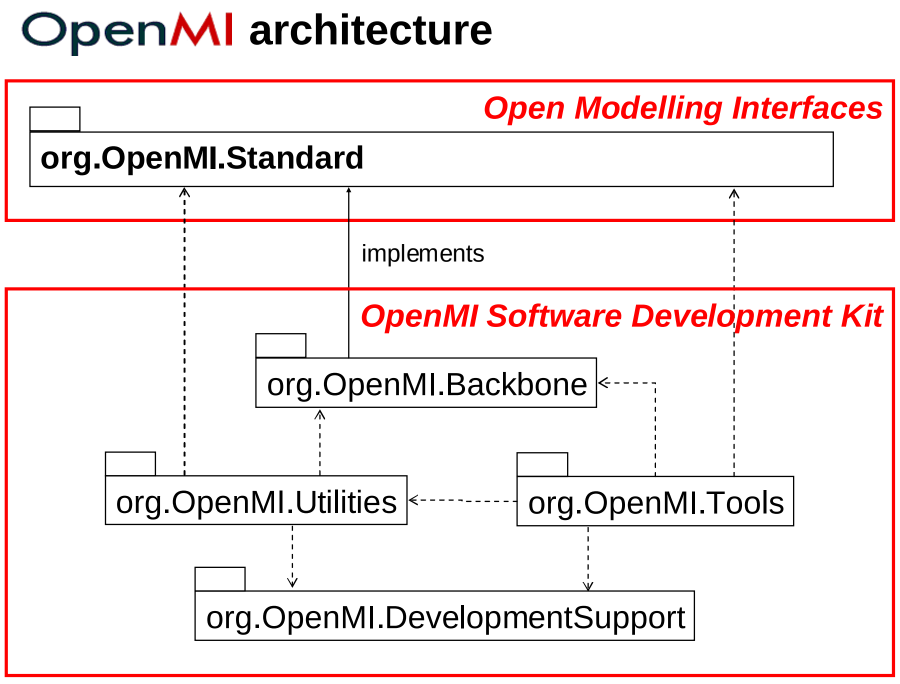


Figure 1-1 Namespaces in the OpenMI architecture

1.2 Requirements

The main purpose of the org.OpenMI.Utilities package is to reduce the reengineering effort of legacy code (mainly computational cores) in order to become OpenMI compliant components. As indicated a wrapping technology will be applied for this purpose.

The org.OpenMI.Utilities namespace is designed taking the following conditions as a starting point:

- [Cond-UT1] - The model developer has full access to the source code of the model.
- [Cond-UT2] -The model a time step based model
- [Cond-UT3] -The model engine is silent
- [Cond-UT4] -The model engine can be compiled to a native/win32 dll
- [Cond-UT5] -The model engine is separate from any GUI.

The following requirements are identified for the org.OpenMI.Utilities namespace.

- [Req-UT1] - the namespace should provide functionality that handle as much as possible OpenMI specific requests without interfering with the engine core
- [Req-UT2] - the namespace should handle all messaging (event s ands exceptions) with the OpenMI
- [Req-UT3] - the namespace should provide an internal interface to the engine which is more natural from the engine perspective
- [Req-UT4] - the namespace should provide support to buffer data, and to clear its buffer when desired
- [Req-UT5] - the packages should only depend on primitives (int, string, double etc.) or on data structures as defined in the org.OpenMI.Standard namespace
- [Req-UT6] - the packages should provide minimal support for spatial and temporal mapping
- [Req-UT7] - the support should be as efficient as possible to ensure high performing modelling systems

1.3 Scope of this document

This report contains the technical documentation of the org.OpenMI.Utilities namespace. The technical documentation addresses the design as well implementation issues. After discussing the major concepts in chapter 2, the various packages are discussed in chapter 3 (org.OpenMI.Utilities.Buffer), chapter 4 (org.OpenMI.Utilities.Spatial), chapter 5 (org.OpenMI.Utilities.Wrapper), chapter 6 (org.OpenMI.Utilities.AdvancedControl) and chapter 7 (org.OpenMI.Utilities.Configuration).

1.4 Readership

This document is meant for code developers who have to implement, extend or maintain the source code of the OpenMI Software Development Kit. In order to understand this document, one needs to have basic understanding of model linking, object-orientation and UML-notation (particularly class diagrams and sequence diagrams). Within the text, the following style-convention is applied:

- OpenMI interface
- OpenMI method
- OpenMI property
- OpenMI argument

2 OpenMI utilities: Concepts

2.1 The OpenMI linking mechanism

OpenMI is a pull-based pipe and filter architecture, consisting of communicating components (providers and acceptors), which exchange data in a pre-defined way and in a pre-defined format. Sometimes, this type of architecture is also referred to as a context based request-reply architecture, in which the context (i.e. the instantiated component) processes and replies to the requests in synchronized order.

OpenMI defines both the component interfaces as well as how the data is defined that is being exchanged. The components in OpenMI are called LinkableComponents to indicate that it involves components that can be linked together.

The data transfer between instances of a LinkableComponent is handled within a single thread, i.e. an instance handles only one request at a time before acting upon another request. Invoking the last component in the chain of linked components triggers data exchange in an OpenMI-system. Once this component is triggered, computation, synchronization and data exchange between all components is handled autonomously without any type of supervising authority.

OpenMI accommodates legacy code by encapsulating this code into a wrapper (see Figure 2-1). As many tasks are similar for all wrappers, a generic design has been made and implemented to reduce the effort involved.

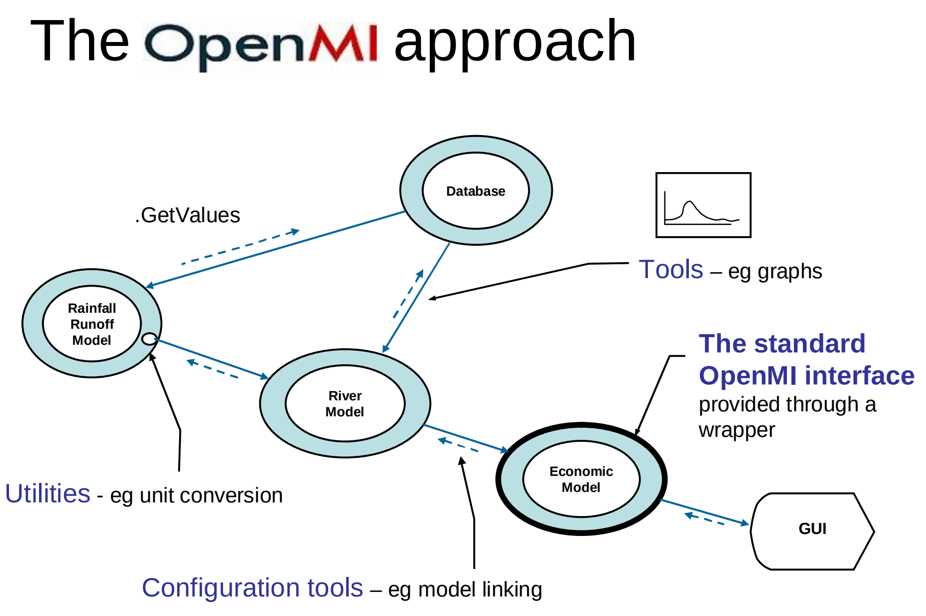

The OpenMI approach
Database
.GetValues
Tools – eg graphs
Rainfall Runoff Model
River Model
The standard OpenMI interface provided through a wrapper
Economic Model
Utilities - eg unit conversion
GUI
Configuration tools – eg model linking

Figure 2-1 The OpenMI approach for linking models

2.2 Task description of the OpenMI utilities

The utilities packages described in this document are not part of the OpenMI standard and as such not required to be used in order to make OpenMI compliant components.

The purpose of creating these utilities is to offer the model builder or the model integrator utilities that will make migration of models to OpenMI easier and faster. The intention is not to create utilities that do everything that a model builder could wish for. Thus very sophisticated methods will not be covered. Such methods should when needed be implemented by the model provider. However, the design of the tools should enable the model builder to extend the existing functionality and in this way obtain utilities with the required functionality.

The spatial utilities are intended accessed from the inside of the OpenMI linkable components. For most cases the actual access will take place from the inside of the model wrapper. This means that from the outside a linkable component will look the same whether the component uses the utilities or the component is using it's own implementation of such utilities.

2.3 Outline of the org.OpenMI.Utilities namespace

2.3.1 Packages

The org.OpenMI.Utilities namespace includes the following four packages:

- org.OpenMI.Utilities.Buffer
- org.OpenMI.Utilities.Spatial
- org.OpenMI.Utilities.Wrapper
- org.OpenMI.Utilities.AdvancedControl
- org.OpenMI.Utilities.Configuration

2.3.2 Relations to other namespaces

As illustrated in Figure 2-2, the packages depend on org.OpenMI.Standard, org.OpenMI.Backbone and org.OpenMI.DevelopmentSuport. Further, the Wrapper package depends on the Buffer package and the Spatial package.

More importantly all parameters used on public methods in the packages are assumed to implement the interfaces from the org.OpenMI.Standard namespace, only. Hence the packages are applicable for all components that implement the org.OpenMI.Standard.

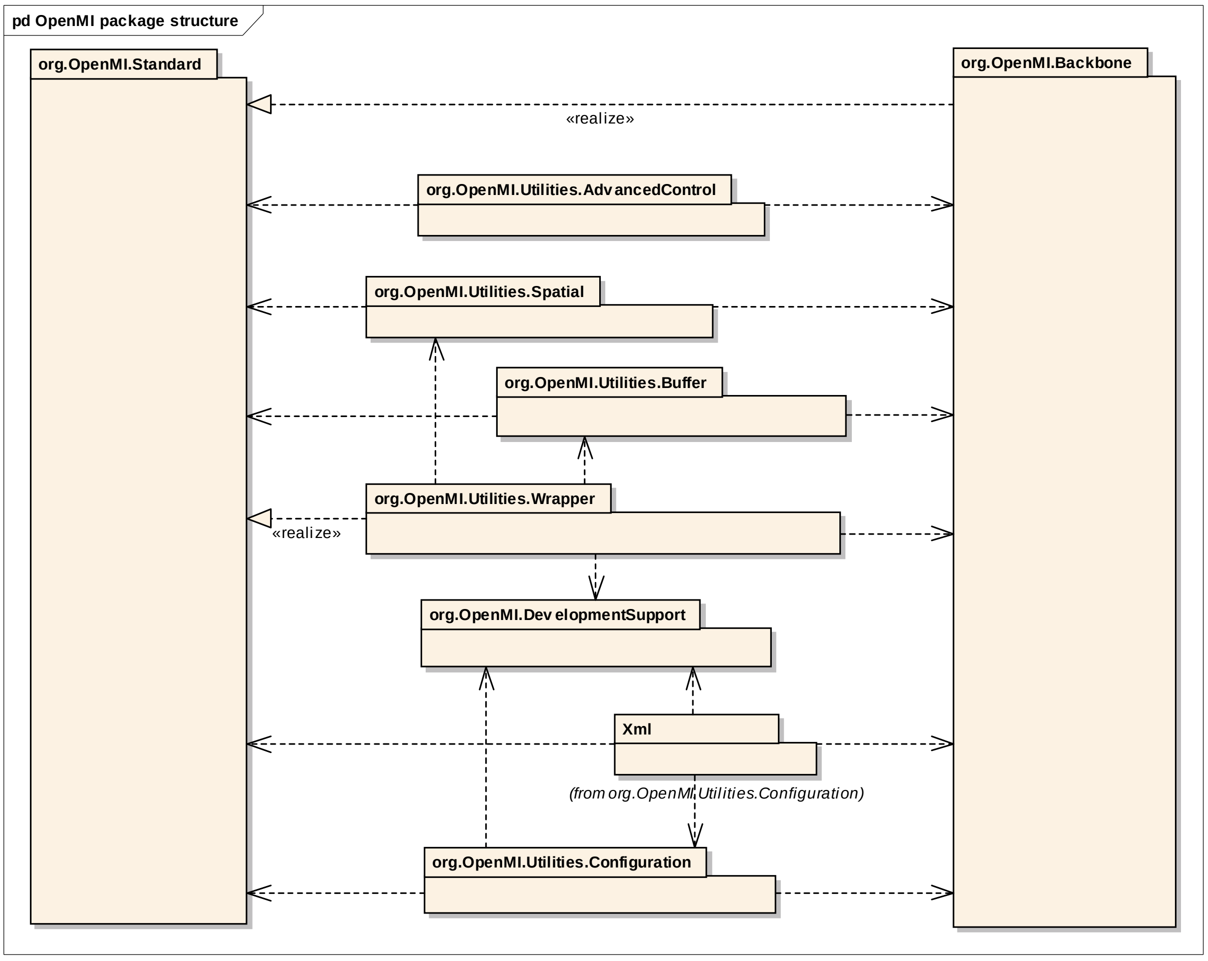

pd OpenMI package structure
org.OpenMI.Standard
org.OpenMI.Backbone
«realize»
org.OpenMI.Utilities.AdvancedControl
org.OpenMI.Utilities.Spatial
org.OpenMI.Utilities.Buffer
org.OpenMI.Utilities.Wrapper
«realize»
org.OpenMI.DevelopmentSupport
Xml
(from org.OpenMI.Utilities.Configuration)
org.OpenMI.Utilities.Configuration

Figure 2-2 Overview of package relations for org.OpenMI.Utilities

3 The org.OpenMI.Utilities.Buffer package

3.1 General description

Water related models are typically doing time step based calculations. Based on values calculated for one time step and external boundary values a set of values for the next time step is calculated. Normally only values for the current time step and the previous time step are kept in memory. When such models are running with links to other models or linkable components in an OpenMI context they need to be able also to deliver values associated to earlier time steps, values that are in-between time steps and even values that lies ahead of the current time step (extrapolated values).

The Buffer package provides buffering functionality that will store values needed for a particular link in memory and functionality that will interpolate values or extrapolate values.

3.2 org.OpenMI.Utilities.Buffer: Static View

The Buffer package consists of the *SmartBuffer* class only.

3.2.1 SmartBuffer

The buffering functionality is obtained through the public methods for filling the buffer, emptying the buffer and finally a method for retrieving interpolated, extrapolated or aggregated data.

The buffer filling is performed by the AddValues method that adds objects offering the IValueSet interface and associated objects offering the ITime interface.

The emptying of the buffer is undertaken by the Clear method that called with an object that implements the ITimeSpan interface deletes all ValueSets that falls within the given TimeSpan.

Retrieving data is performed by the GetValues method that called with an object that implements the ITime interface returns an object that implements the IValueSet (the org.OpenMI.Backbone.ValueSet). The GetValues determines itself whether interpolation, extrapolation or aggregation is to be performed

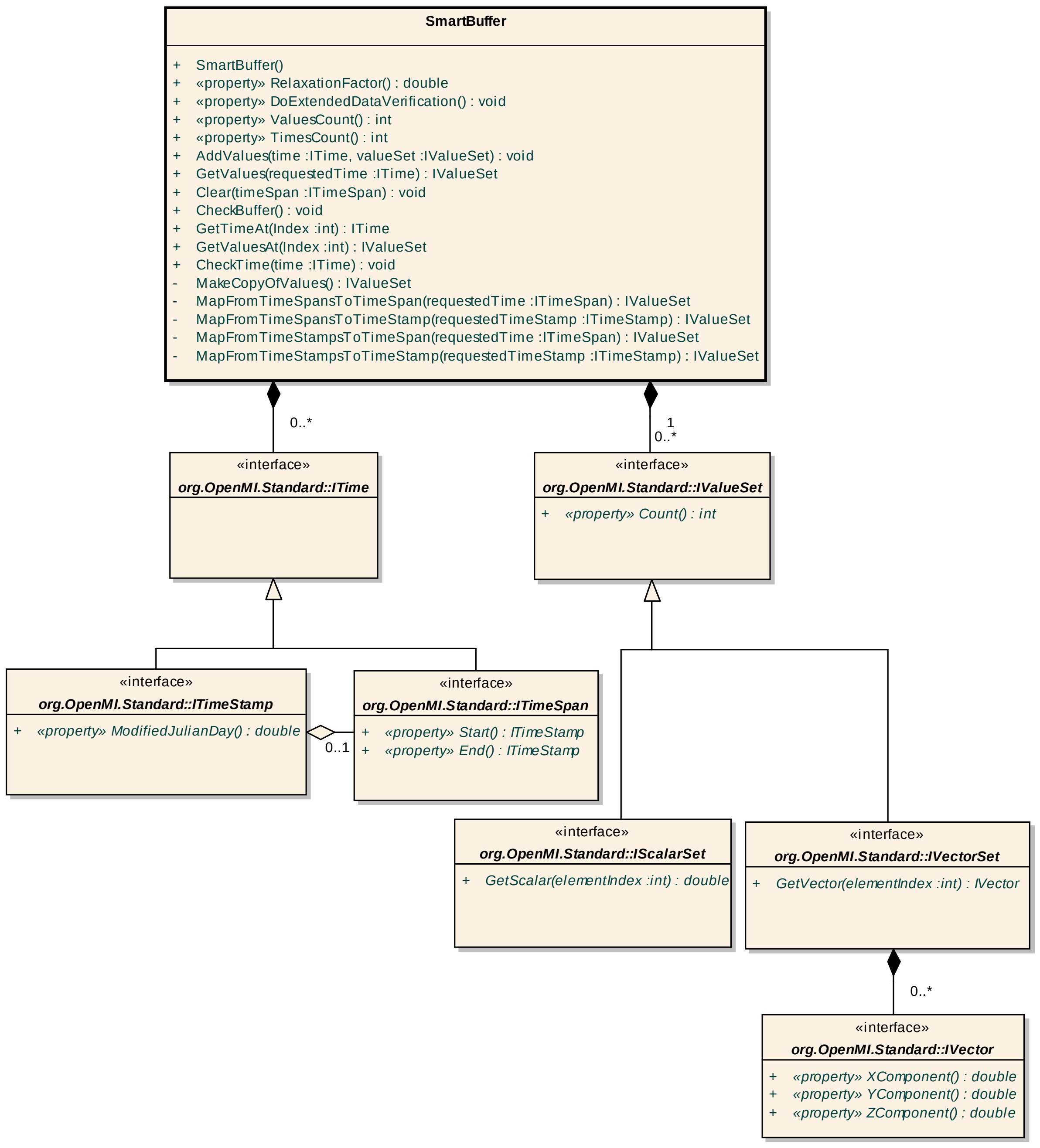

SmartBuffer
+ SmartBuffer()
+ «property» RelaxationFactor() : double
+ «property» DoExtendedDataVerification() : void
+ «property» ValuesCount() : int
+ «property» TimesCount() : int
+ AddValues(time :ITime, valueSet :IValueSet) : void
+ GetValues(requestedTime :ITime) : IValueSet
+ Clear(timeSpan :ITimeSpan) : void
+ CheckBuffer() : void
+ GetTimeAt(Index :int) : ITime
+ GetValuesAt(Index :int) : IValueSet
+ CheckTime(time :ITime) : void
- MakeCopyOfValues() : IValueSet
- MapFromTimeSpansToTimeSpan(requestedTime :ITimeSpan) : IValueSet
- MapFromTimeSpansToTimeStamp(requestedTimeStamp :ITimeStamp) : IValueSet
- MapFromTimeStampsToTimeSpan(requestedTimeStamp :ITimeSpan) : IValueSet
- MapFromTimeStampsToTimeStamp(requestedTimeStamp :ITimeStamp) : IValueSet
0..*
1
0..*
«interface»
org.OpenMI.Standard::ITime
«interface»
org.OpenMI.Standard::IValueSet
+ «property» Count() : int
«interface»
org.OpenMI.Standard::ITimeStamp
+ «property» ModifiedJulianDay() : double
0..1
«interface»
org.OpenMI.Standard::ITimeSpan
+ «property» Start() : ITimeStamp
+ «property» End() : ITimeStamp
«interface»
org.OpenMI.Standard::IScalarSet
+ GetScalar(elementIndex :int) : double
«interface»
org.OpenMI.Standard::IVectorSet
+ GetVector(elementIndex :int) : IVector
0..*
«interface»
org.OpenMI.Standard::IVector
+ «property» XComponent() : double
+ «property» YComponent() : double
+ «property» ZComponent() : double

Figure 3-1 SmartBuffer

3.3 Dynamic view

The buffer is utilized by the SmartWrapper. See Section 5.2 for details.

3.4 Implementation remarks

3.4.1 C#-implementation

The C#-implementation does not utilize any specific .NET features.

Table 3.4.1 provides an overview of the .NET framework assemblies that have been utilized in the various classes of org.OpenMI.Utilities.Buffer.

Table 3.4.1 Overview of .NET framework assemblies utilized within org.OpenMI.Utilities.Buffer

.NET framework assembly	**utilized by org.OpenMI.Utilities.Buffer classes**
System	all classes
System.Collections	SmartBuffer
System.IO	SmartBuffer

The correct implementation of all methods has been tested using dedicated unit tests in combination with the NUnit framework for testing.

The org.OpenMI.Utilities.Buffer package is available as open source under LGPL licence on Source Forge (http://sourceforge.net/projects/openmi)

3.4.2 Java-implementation

No information available yet

4 The org.OpenMI.Utilities.Spatial package

4.1 General description

Basically the spatial utilities will convert one ValueSet associated to one ElementSet to a new ValueSet that corresponds to another ElementSet. The conversion will be a two step procedure, where the first step (Initialize) will be executed only ones. Subsequently conversion will be executed by the MapValues method.

The Initialize method will create a conversion matrix with the same number of rows as the number of elements in the ElementSet associated to the accepting component and the same number of columns as the number of elements in the ElementSet associated to the providing component.

Mapping is possible for any zero, one, and two-dimensional elements. Mapping for three.dimensional ElementSets is not supported in the current implementation.

Zero-dimensional elements will always be points, one-dimensional elements will always be polylines, and two-dimensional elements will always be polygons.

The spatial package does not contain very advanced methods. This job will be left to the model provider that needs such methods. However, the architecture of the spatial utilities will be created in such a way that it will facilitate easy extension. This means that the model provider does not need to start from scratch but can build on what has already been provided in the Software Development Kit.

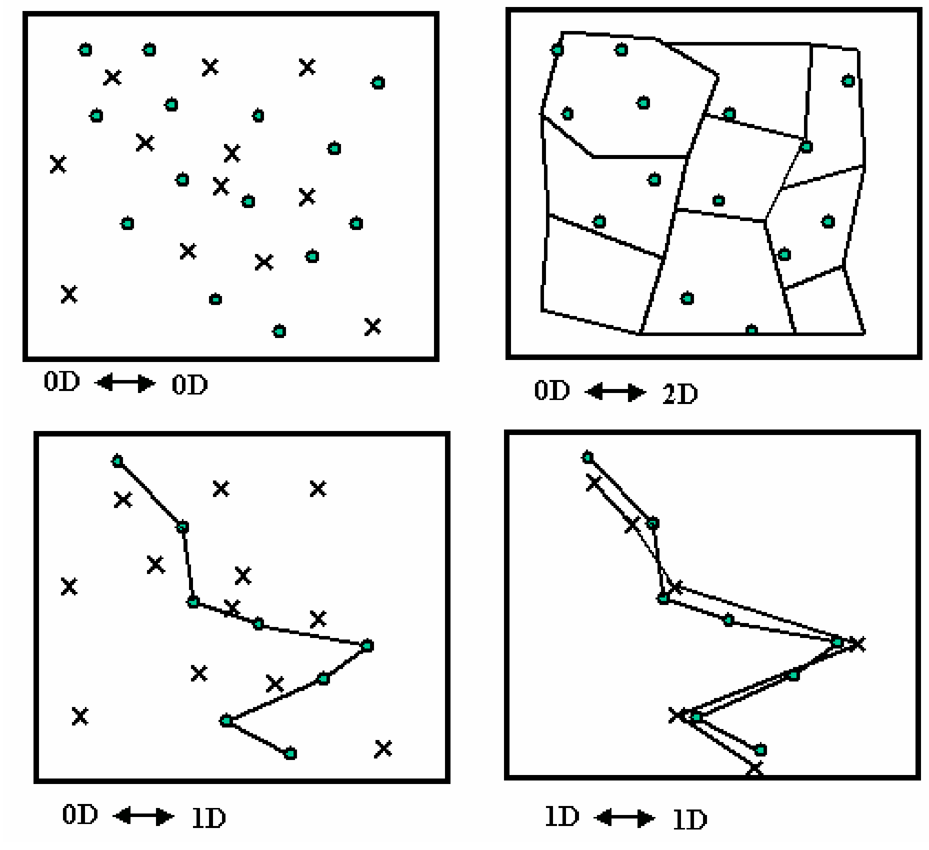


Figure 4.1.1: Spatial mapping examples

4.2 Static View

The org.OpenMI.Utilities.Spatial package consists of the ElementMapper class and a number of support classes. The ElementMapper class uses the functionality of the support classes. The user of the spatial utility package (the person building model engine wrappers) will access the functionality of ElementMapper class only. However, people that want to build their own ElementMapper or extend the functionality of the existing ElementMapper can use the support classes.

The support classes XYPoint, XYPolyline and XYPolygon are implementations for 0, 1 and 2 dimensional data respectively. XYLine is a support class used by XYPolyline and XYPolygon. The XYGeometry class is a collection of general geometry functions. All functions in XYGeometry are static methods that performs calculations on input given as parameters (typically input of type: XYPoint, XYLine, XYPolyline and/or XYPolygon) and returns a result.

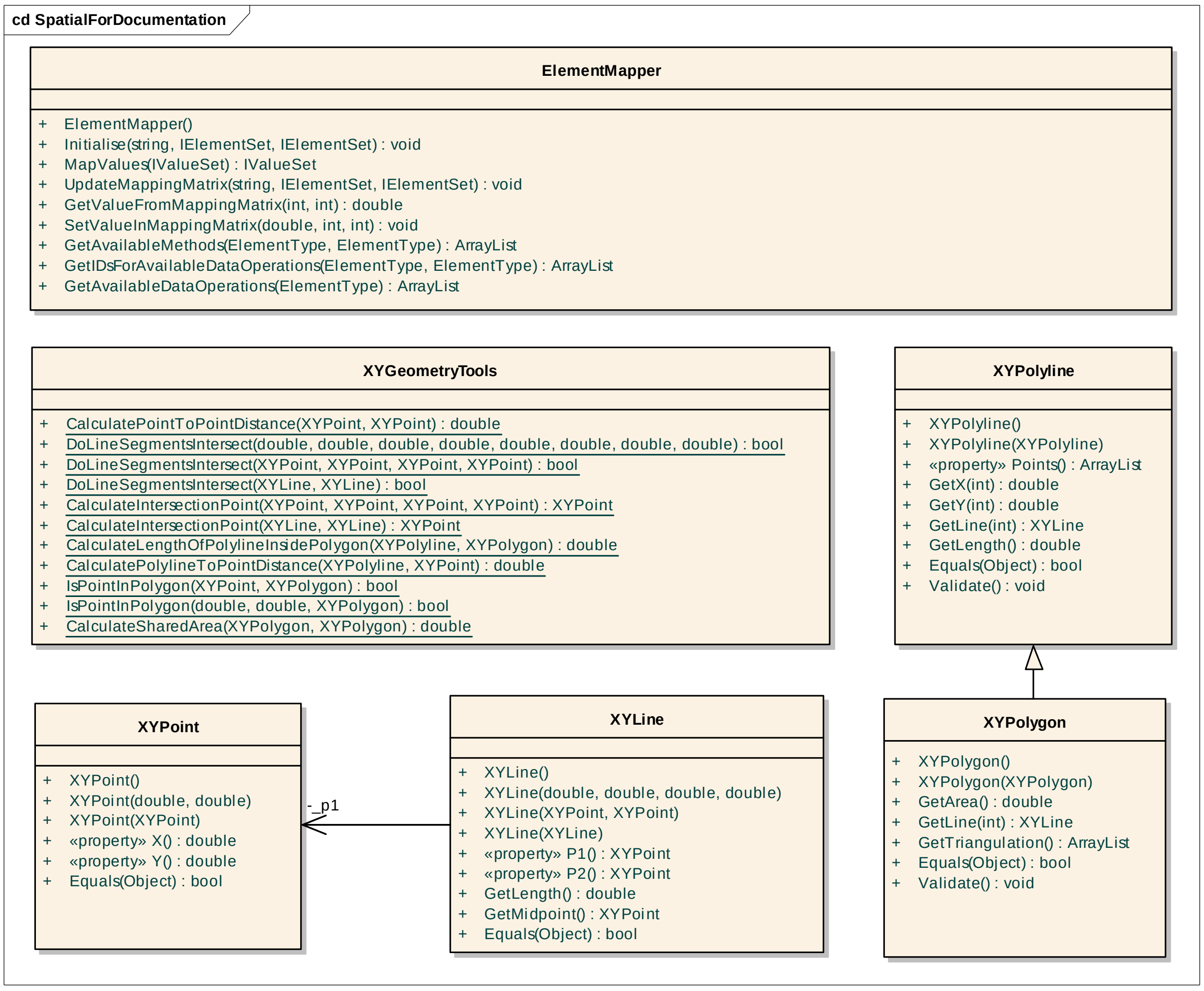

cd SpatialForDocumentation
ElementMapper
+ ElementMapper()
+ Initialise(string, IElementSet, IElementSet) : void
+ MapValues(IValueSet) : IValueSet
+ UpdateMappingMatrix(string, IElementSet, IElementSet) : void
+ GetValueFromMappingMatrix(int, int) : double
+ SetValueInMappingMatrix(double, int, int) : void
+ GetAvailableMethods(ElementType, ElementType) : ArrayList
+ GetIDsForAvailableDataOperations(ElementType, ElementType) : ArrayList
+ GetAvailableDataOperations(ElementType) : ArrayList
XYGeometryTools
+ CalculatePointToPointDistance(XYPoint, XYPoint) : double
+ DoLineSegmentsIntersect(double, double, double, double, double, double, double, double) : bool
+ DoLineSegmentsIntersect(XYPoint, XYPoint, XYPoint, XYPoint) : bool
+ DoLineSegmentsIntersect(XYLine, XYLine) : bool
+ CalculateIntersectionPoint(XYPoint, XYPoint, XYPoint, XYPoint) : XYPoint
+ CalculateIntersectionPoint(XYLine, XYLine) : XYPoint
+ CalculateLengthOfPolylineInsidePolygon(XYPolyline, XYPolygon) : double
+ CalculatePolylineToPointDistance(XYPolyline, XYPoint) : double
+ IsPointInPolygon(XYPoint, XYPolygon) : bool
+ IsPointInPolygon(double, double, XYPolygon) : bool
+ CalculateSharedArea(XYPolygon, XYPolygon) : double
XYPolyline
+ XYPolyline()
+ XYPolyline(XYPolyline)
+ «property» Points() : ArrayList
+ GetX(int) : double
+ GetY(int) : double
+ GetLine(int) : XYLine
+ GetLength() : double
+ Equals(Object) : bool
+ Validate() : void
XYPoint
+ XYPoint()
+ XYPoint(double, double)
+ XYPoint(XYPoint)
+ «property» X() : double
+ «property» Y() : double
+ Equals(Object) : bool
-_p1
XYLine
+ XYLine()
+ XYLine(double, double, double, double)
+ XYLine(XYPoint, XYPoint)
+ XYLine(XYLine)
+ «property» P1() : XYPoint
+ «property» P2() : XYPoint
+ GetLength() : double
+ GetMidpoint() : XYPoint
+ Equals(Object) : bool
XYPolygon
+ XYPolygon()
+ XYPolygon(XYPolygon)
+ GetArea() : double
+ GetLine(int) : XYLine
+ GetTriangulation() : ArrayList
+ Equals(Object) : bool
+ Validate() : void

Figure 4.2.1: Class diagram for the spatial package

The above mentioned classes work on consistent ElementSets, i.e. ElementSets where all elements have the same ElementType. By definition, an element of type XYPoint must contain one vertex; an element of type XYLine must contain two vertices, while an element of type XYPolyline requires at least 2 vertices. An element of type XYPolygon requires at least 3 vertices, while the last vertex of a polygon should correspond to the first one. To validate an ElementSet, a separate class has been developed called ElementSetChecker (see Figure 4.2.2).

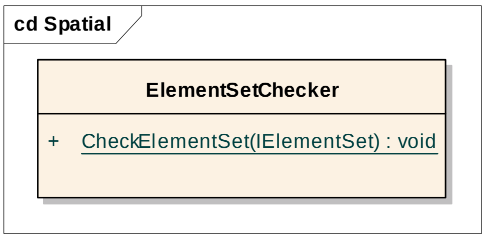


Figure 4.2.2: Class view of the ElementSetChecker

4.3 Dynamic view

The main concept of the ElementMapper is that a mapping matrix is calculated at initialization time and the MapValue method is called at simulation time. The MapValue method multiplies the from-values onto the mapping matrix to get the to-values during simulation. The core part of the ElementMapper is, hence, placed in the initialization part where the mapping matrix is calculated.

4.3.1 Retrieving the possible methods

When initialising the ElementMapper a methodID has to be passed as one of the parameters. The methodID is a string that needs to be recognisable to the ElementMapper, hence the string should preferably come from the ElementMapper itself.

For this purpose the ElementMapper offers a GetAvailableDataOperations method that given the from ElementTypes returns a list of method IDs. This list should be included edited or not in the DataOperation list of the IOutputExchangeItem. In this way the method IDs of the ElementMapper will be offered as data operations and subsequently passed back into the ElementMapper as an argument for the Initialize method.

The description strings for the available methods are listed in Table 4.3.1.

From ElementType	To ElementType	Method descriptor
XYPoint	XYPoint	Nearest
		Inverse
	XYPolyline	Nearest
		Inverse
	XYPolygon	Mean
		Sum
XYPolyline	XYPoint	Nearest
		Inverse
	XYPolygon	Weighted Mean
		Weighted Sum
XYPolygon	XYPoint	Value
	XYPolyline	Weighted Mean
		Weighted Sum
	XYPolygon	Weighted Mean
		Weighted Sum

Table 4.3.1: List of available data operations

4.3.2 Initialising the ElementMapper

A wrapper that adds the ILinkableComponent interface to an engine typically uses the ElementMapper. The SmartWrapper from the utility package is an example of such a wrapper.

The sequence diagram included in **Error! Reference source not found.** shows the calling sequence invoked by a wrapper that initialises an ElementMapper to be able to map values from an ElementSet named fromElements onto an ElementSet named toElements. Both elementSets consist of Elements of ElementType XYPolygon.

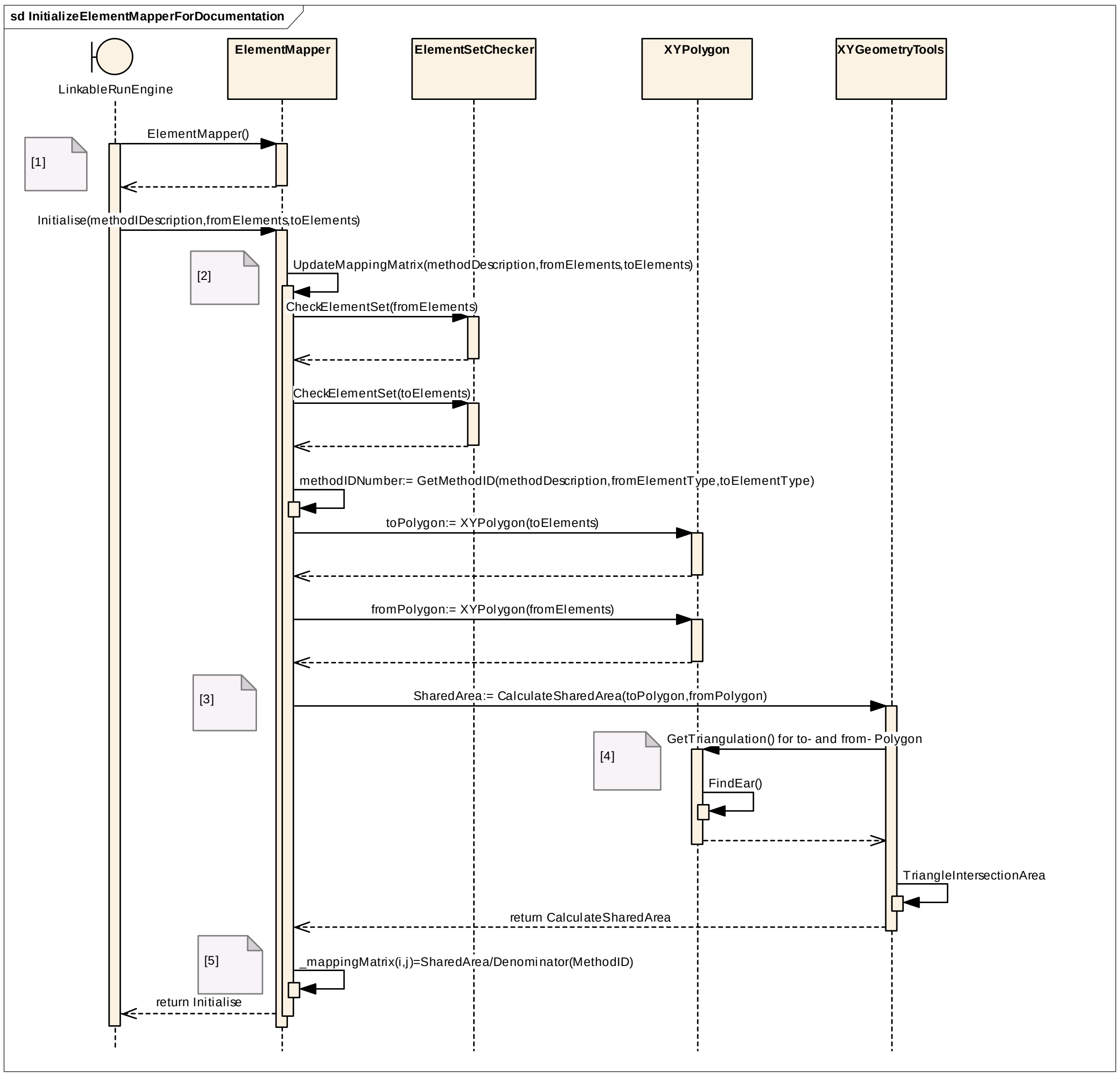

sd InitializeElementMapperForDocumentation
LinkableRunEngine
ElementMapper
ElementSetChecker
XYPolygon
XYGeometryTools
ElementMapper()
[1]
Initialise(methodIDescription,fromElements,toElements)
[2]
UpdateMappingMatrix(methodDescription,fromElements,toElements)
CheckElementSet(fromElements)
CheckElementSet(toElements)
methodIDNumber:= GetMethodID(methodDescription,fromElementType,toElementType)
toPolygon:= XYPolygon(toElements)
fromPolygon:= XYPolygon(fromElements)
[3]
SharedArea:= CalculateSharedArea(toPolygon,fromPolygon)
[4]
GetTriangulation() for to- and from- Polygon
FindEar()
TriangleIntersectionArea
return CalculateSharedArea
[5]
_mappingMatrix(i,j)=SharedArea/Denominator(MethodID)
return Initialise

Figure 4-1: Sequence diagram for ElementMapper Initialisation

1. At initialisation time the LinkableRunEngine creates an instance of the ElementMapper for each link. The ElementMapper is initialised using the ElementSets of the link and a mapping method descriptor from the link DataOperations.
2. Initialize will set a flag indicating that the ElementMapper is initialised and calculate the mapping matrix. Since recalculation of the mapping matrix is to be offered at simulation time, also, the calculation is performed in the public method called UpdateMappingMatrix. UpdateMappingMatrix initially verifies that the input ElementSets are valid ElementSets. Besides verification, the method includes a selection block that based on the ElementTypes of the source- and target- ElementSets

and the MethodDescriptor chooses the method to be used for the calculation of the mapping matrix elements.

3. The shared area between every XYPolygon in the source ElementSet and every XYPolygon in the target ElementSet is calculated. The shared area is used for the calculation of the mapping matrix elements.
4. In order to calculate the shared area of two arbitrarily shaped polygons each polygon is divided into a set of triangles using a simple ear-cutting algorithm. Subsequently the shared area of each combination of triangles is calculated and summed to give the shared area of the polygons.
5. The shared areas are nominated with either the total area of the polygon itself (weighted sum) or by the total area covered in that polygon (weighted mean) dependent on the mapping method specified with the methodDescriptor argument.

4.3.3 Possible Exceptions

The Spatial package uses nested exceptions through the use of the InnerException property. Exceptions thrown by private methods are caught in the public methods and added the Exception from the public method as InnerException. Table 4.3.2 provides an overview of possible exceptions.

Class	Method	Description	Has Inner-Exception
ElementMapper	MapValues	ElementMapper objects needs to be initialised before the MapValue method can be used	-
		Dimension mismatch between inputValues and mapping matrix	-
		Invalid datatype used for inputValues parameter. MapValues failed	-
	UpdateMappingMatrix	methodDescription unknown for point to point mapping	-
		Point to point mapping failed	+
		methodDescription unknown for point to polyline mapping	-
		Point to polyline mapping failed	+
		methodDescription unknown for point to polygon mapping	-
		Point to polygon mapping failed	+
		methodDescription unknown for polygon to point mapping	-
		Polyline to point mapping failed	+
		methodDescription unknown for polygon to point mapping	-
		Polyline to polygon mapping failed	+
		methodDescription unknown for polygon to point mapping	-
		Polygon to point mapping failed	+
		methodDescription unknown for polygon to polyline mapping	-
		Polygon to polyline mapping failed	+
		methodDescription unknown for polygon to polygon mapping	-
		Polygon to polygon mapping failed	+
		Mapping of specified ElementTypes not included in ElementMapper	-
		UpdateMappingMatrix failed to update mapping matrix	+
	GetValueFromMappingMatrix	GetValueFromMappingMatrix failed.	+
	SetValueInMappingMatrix	SetValueInMappingMatrix failed.	+
	ValidateIndicies	Negative row index not allowed.	-

		Row index exceeds mapping matrix dimension.	-
		Negative column index not allowed.	
		Column index exceeds mapping matrix dimension.	-
	GetMethodID	methodDescription: %1 not known for fromElementType: %2 and to ElementType: %3. GetMethodID failed. (where %1-%3 are the actual values passed as parameters to the method)	-
	CreateXYPoint	Cannot create XYPoint	-
	CreateXYPolyline	Cannot create XYPolyline	-
	CreateXYPolygon	Cannot create XYPolygon	-
XYGeometryTools	CalculateIntersectionPoint	Attempt to calculate intersectionpoint between non intersecting lines. CalculateIntersectionPoint failed	-
	TriangleIntersectionArea	Argument must be a polygon with 3 points	-
		Failed to find intersection polygon	-
		TriangleIntersectionArea failed	+

Table 4.3.2: Possible exceptions of the ElementMapper

4.4 Implementation remarks

4.4.1 C#-implementation

The C#-implementation does not utilize any specific .NET features.

Error! Reference source not found. provides an overview of the .NET framework assemblies that have been utilized in the various classes of org.OpenMI.Utilities.Spatial.

.NET framework assembly	**utilized by org.OpenMI.Utilities.Spatial classes**
System	all classes
System.Collections	ElementMapper, XYGeometryTools, XYPolygon, XYPolyline,

Table 4.4.1: Overview of .NET framework assemblies utilized within org.OpenMI.Utilities.Spatial

Function descriptions and interface descriptions are given as XML comments in the source code.

The correct implementation of all methods has been tested using dedicated unit tests in combination with the NUnit framework for testing.

The org.OpenMI.Utilities.Spatial package is available as open source under LGPL licence on Source Forge (http://sourceforge.net/projects/openmi)

4.4.2 Java-implementation

No information available yet

5 The org.OpenMI.Utilities.Wrapper package

5.1 General description

During the development of the OpenMI standard the project team was also migrating existing models. We discovered that for numerical model engines that are doing time step based calculation there were many commonalties in the wrappers even though those model were different with respect to domain and dimension. An example of such models could be a one-dimensional finite difference based river model engine and a two-dimensional finite element based ground water model engine. It was therefore decided to develop a Wrapper package that could assist developers in migration of such kind of model engines.

The OpenMI standard puts a lot of responsibilities on the LinkableComponents. Such components must implement all be bookkeeping for handling links added through the AddLink method and they may have to return values that are interpolated, extrapolated or aggregated in time and space in order to conform with the GetValues request. When you look at the ILinkableComponent interface you will not find any methods that directly will trigger the model engine to run or do a time step, such things are assumed to be handled behind the scene. So, to figure out, based on the interface definitions alone, what exactly do to your model system may seem difficult. The design of the Wrapper package, Buffer package, and the Spatial package will provide you with one possible way to go.

This chapter provides the technical documentation of the wrapper package. Descriptions of how to use this package are given in document B, Guidelines, book 4. It is highly recommended you read this book before reading this chapter.

5.2 Static View

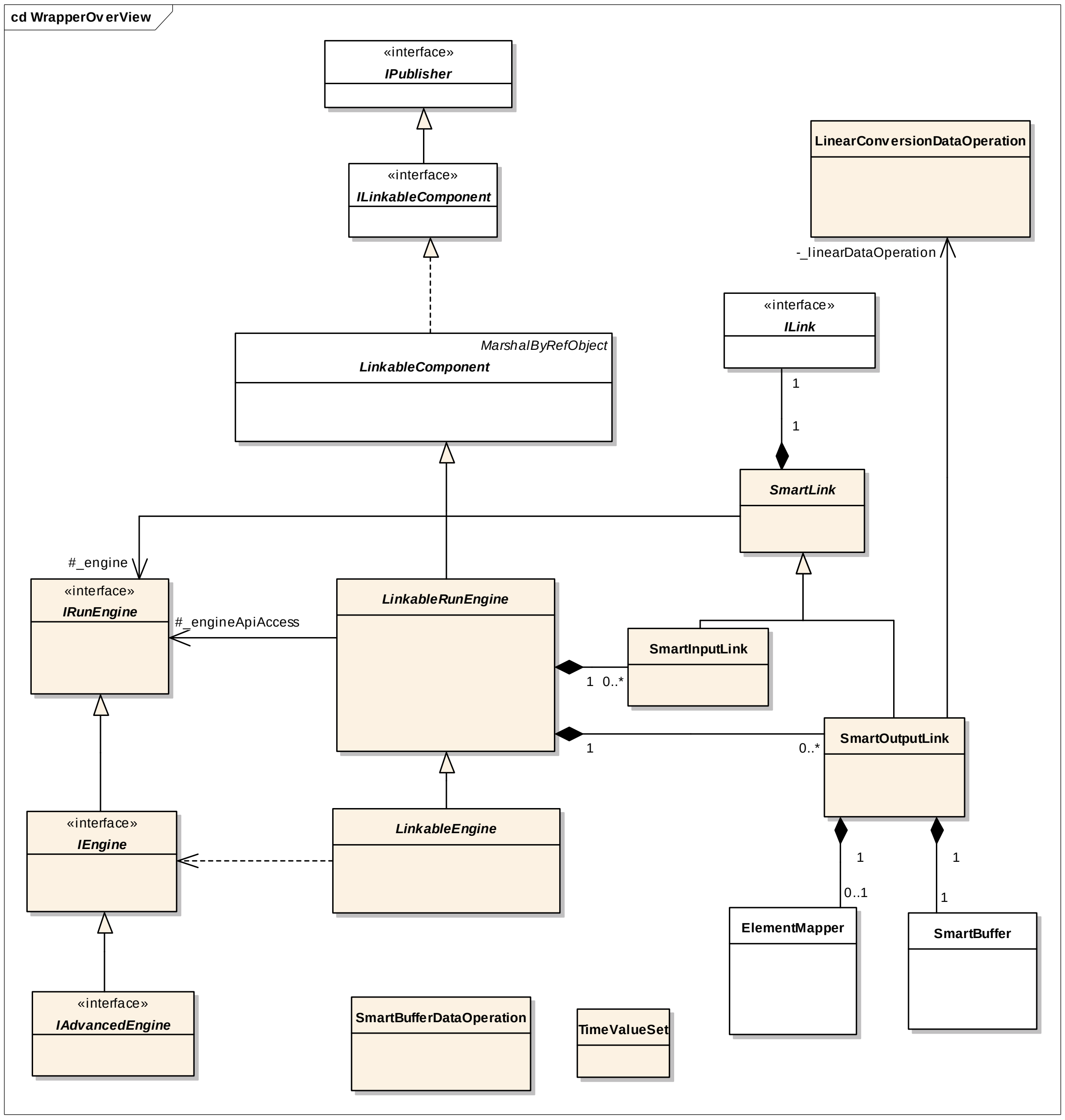


Figure 5-1 Class diagram for the Wrapper classes and classes used in other packages.

The Wrapper package classes are shown on figure 5-1 above. The collared classes on the figure belong to the wrapper package. The white classes are included in the figure in order to illustrate the relation between the wrapper classes and these classes.

The LinkableEngine class is the main class in the wrapper package. If you follow the inheritance hierarchy on figure 5-1, you can see that the LinkableEngine implements the ILinkableComponent interface. When using the wrapper package for model migration the developer will derive his linkable component from this class. The LinkableEngine in derived from the LinkableRunEngine class. There are historical reasons for this. Basically, the LinkableRunEngine class and the LinkableEngine class could be merged, but in order to maintain backwards compliancy two separate classes still exists in the wrapper package. The LinkableEngine class will access the specific model engine through the IEngine interface or the IAdvancedEngine interface. The LinkableEngine keeps internal lists of SmartInputLinks and SmartOutputLinks. Each of these SmartLinks has contains the backbone implementation of the ILink interface. Each SmartOutputLink owns an instance of the SmartBuffer class (from the Buffer package). If the link is geo-referenced also an instance of the ElementMapper class is owned by the SmartOutputLink

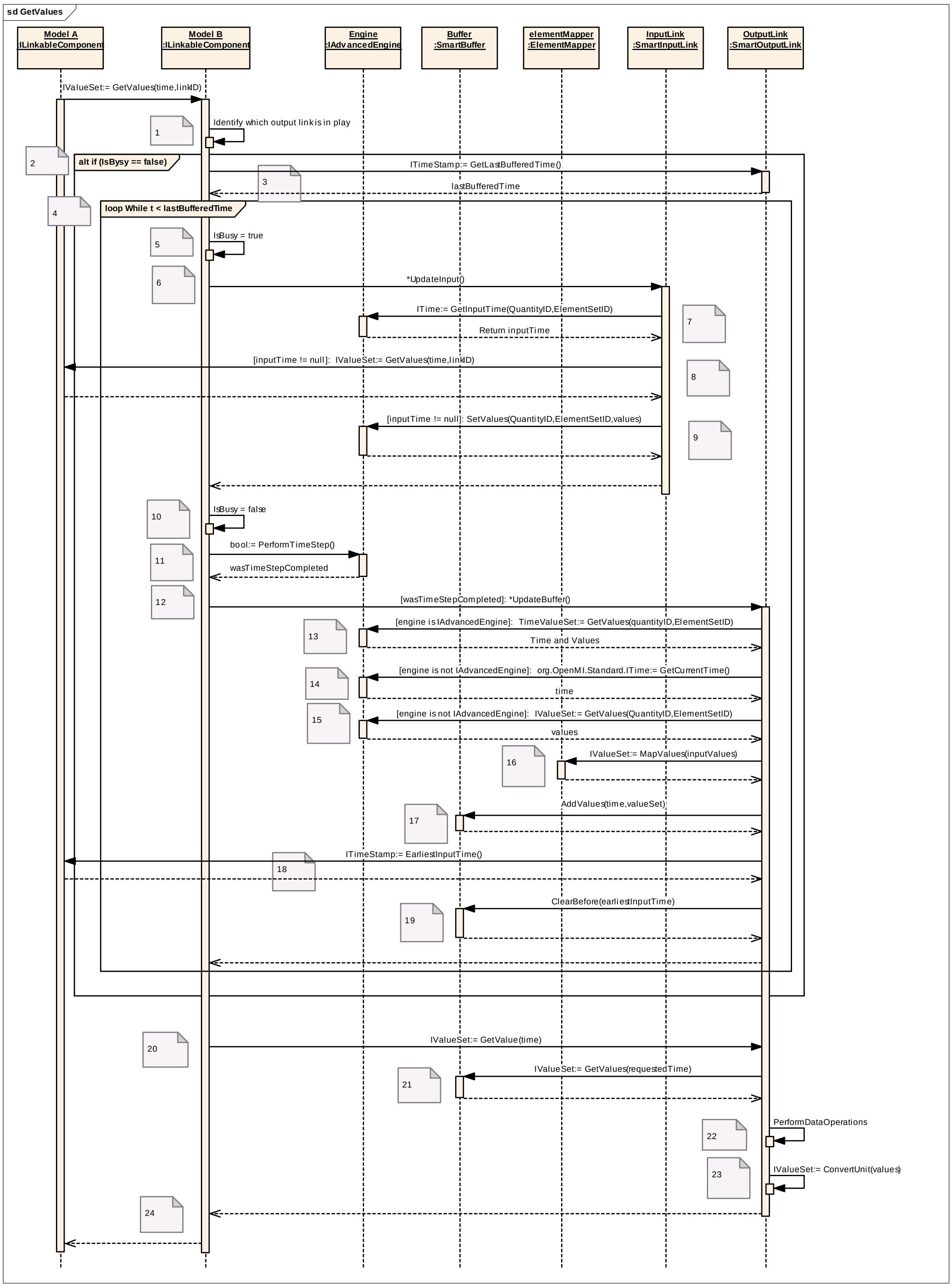
5.3 Dynamic view - GetValues

The key to understanding the wrapper package is the implementation of the GetValues method. The implementation of the GetValues method is shown on figure 5-2. the sequence diagram illustrates an example of two models linked bi-directionally with georeferenced links. The figure has notes with numbers. And explanation for each note is given below:

1. The LinkableEngine class will search through it's internal list of output links to find the link which has a LinkID identical to the LinkID passed in the argurment list of the GetValues call.
2. The LinkableEngine has an internal flag (Boolean) which when true indicates that the engine in the process of gathering data from other components in order to prepare for performing a time step. The purpose of this flag is to avoid deadlocks between bi-directional linked models. If this flag is true when the GetValues is invoked, the LinkableEngine will proceed to step 20, an extrapolate values from already available values in the buffer, rather that performing time steps.
3. The output link found in step 1 is asked for the last buffered time.
4. Based on information about the last buffered time, the LinkableEngine can determine if it needs to perform time steps or if the requested values already are available in the buffer. If values are available the LinkableEngine can proceed to step 18.
5. The engine is now in stage of gathering data from other component in order to prepare for the time step. In order to avoid deadlocks the IsBusy flag is set to true.
6. On each input link in the list of input links the UpdateInput method is invoked.
7. The GetInputTime method is invoked in the engine. If the engine does not need input data for this particular quantity for the current time step null is returned, otherwise the time for which input is needed is returned. The returned time can be either a TimeStamp or a timeSpan.
8. GetValues is invoked.
9. The retrieved values are set in the engine
10. The gathering of data from outside is completed and the IsBusy flag is changed back to false.
11. The Engine is now ready to do a time step and the PerformTimeStep is invoked. PerformTimeStep may return false, which indicates that the time step was not completed. The reason for this may be that the model is using adaptive time steps, and is in the process of reducing the time step length. So if performTimeStep returns false, things start from step 4 again.
12. The time step has now been completed and the buffers can be updated with the calculated values. For each OutputLink in the list of OutputLinks the UpdateBuffer method is invoked.
13. The engine can be accessed either through the ILinkebleEngine interface or through the IAdvancedEngine interface. The IAdvancedEngine interface is an extension to the IEngine interface. This extension was made in order to facilitate models where different quantities are calculated based on different time steps. If the engine implements the IAdvancedEngine

interface the engine is queried for a TimeValues object. The TimeValues object combines values and the time for which it applies.

14. If the engine does not implement the IAdvancedEngine interface it is assumed that the calculated values apply for the current time of the engine. Consequently, the GetCurrentTime is invoked in order to obtain this time.
15. If the engine does not implement the IAdvancedEngine interface the calculated values are obtained from the engine through invocation of GetValues.
16. If the current link is geo-referenced the ElementMapper object associated to this link is used to make the spatial transformations through invocation of the MapValues method.
17. The calculated and mapped values are added to the buffer associated to the current link through invocation of the AddValues method.
18. The EarliesInputTime property of the requesting linkable component is queried.
19. The buffer is cleared for values that lies before the EarliestInputTime obtained in step 18.
20. The current OutputLink object is queried for the calculated values.
21. The current OutputLink queries the associated buffer for values. Temporal operations are done as part of this operation, e.g. interpolation, extrapolation, or aggregation.
22. Any other data operations associated to the current link are performed. This could be linear conversion.
23. Unit conversion.
24. Final return to the requesting LinkableComponent.


sd GetValues
Model A
:ILinkableComponent
Model B
:ILinkableComponent
Engine
:IAdvancedEngine
Buffer
:SmartBuffer
elementMapper
:ElementMapper
InputLink
:SmartInputLink
OutputLink
:SmartOutputLink
IValueSet:= GetValues(time,linkID)
Identify which output link is in play
alt if (IsBysy == false)
ITimeStamp:= GetLastBufferedTime()
lastBufferedTime
loop While t < lastBufferedTime
IsBusy = true
*UpdateInput()
ITime:= GetInputTime(QuantityID,ElementSetID)
Return inputTime
[inputTime != null]: IValueSet:= GetValues(time,linkID)
[inputTime != null]: SetValues(QuantityID,ElementSetID,values)
IsBusy = false
bool:= PerformTimeStep()
wasTimeStepCompleted
[wasTimeStepCompleted]: *UpdateBuffer()
[engine is IAdvancedEngine]: TimeValueSet:= GetValues(quantityID,ElementSetID)
Time and Values
[engine is not IAdvancedEngine]: org.OpenMI.Standard.ITime:= GetCurrentTime()
time
[engine is not IAdvancedEngine]: IValueSet:= GetValues(QuantityID,ElementSetID)
values
IValueSet:= MapValues(inputValues)
AddValues(time,valueSet)
ITimeStamp:= EarliestInputTime()
ClearBefore(earliestInputTime)
IValueSet:= GetValue(time)
IValueSet:= GetValues(requestedTime)
PerformDataOperations
IValueSet:= ConvertUnit(values)
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24

5.3.1 Sequence diagram for the GetValues metod.

5.4 Implementation remarks

5.4.1 C#-implementation

Function descriptions and interface descriptions are given as XML comments in the source code.

The correct implementation of all methods has been tested using dedicated unit tests in combination with the NUnit framework for testing.

The org.OpenMI.Utilities.Wrapper package is available as open source under LGPL licence on Source Forge (http://sourceforge.net/projects/openmi)

5.4.2 Java-implementation

No information available yet

6 The org.OpenMI.Utilities.AdvancedControl package

6.1 General description

6.1.1 Advanced control functionality why and when

The data exchange and synchronization mechanism of OpenMI is designed in such way that LinkbaleComponents can autonomously exchange data without any centralized functionality to manage the data exchange. However, in some specific situations, extra control capacity is needed to direct convergence of computational results. This functionality typically is desired for iteration purposes, as well as for optimization and calibration.

A separate utility package has been designed to support these advanced control features. The package, named org.OpenMI.Utilities.AdvancedControl, utilizes the GetValues mechanism of the LinkableComponent, as well as the state management functionality (implemented through the IManageState interface) to direct the convergence of computational results. The controllers themselves are LinkableComponents as well, so their data (i.e. the new parameter values or boundary conditions) can be accessed by a model based LinkableComponent as well.

The following control functionality has been identified:

- The iteration controller is needed when implicit models are linked bi-directionally and iterations within time-steps are needed
- The calibration controller component is responsible for calibrating one model or a set of linked simulation engines. The optimisation controller can be used to find values of variables that minimize or maximize an objective function while satisfying a set of constraints. In the OpenMI framework this will normally be used in conjunction with a set of linked models.
- The logical switch allows switching inputs depending on an input condition

6.1.2 Iteration controller

When multiple implicit model engines are linked bi-directionally, iterations within time-steps may be required to obtain correct numerical values. An implicit model engine requires the input value on the next time-step in order to calculate the output value on the next time-step. An explicit model engine requires only the input on the current time-step to calculate the output on the next time-step. When simulation engines are linked bi-directionally and no iterations within time-steps are used, the link is effectively explicit. This may mean that the simulation engines have to use a much smaller time-step. An alternative may be to use iterations within time-steps. The simulation engines have to support saving and restoring of the engine state when the iteration controller is used.

When the IterationController is used, the models do not exchange values directly but through the iteration controller (see Figure 6-1). The iteration controller is in control of the iterations and requests the models to step back one time-step. The iteration controller decides when the iterations have converged. The iteration controller throws an exception when there is no convergence in the iteration.

The IterationController has a number of data slots numbered 1,2, 3, etc. In order to connect quantities through the IterationController, the output exchange item of the providing component should be connected to a certain slot (for instance 1), and the input exchange item of the receiving component should be connected to the same slot (for instance 1). One slot should only be used for 1 link.

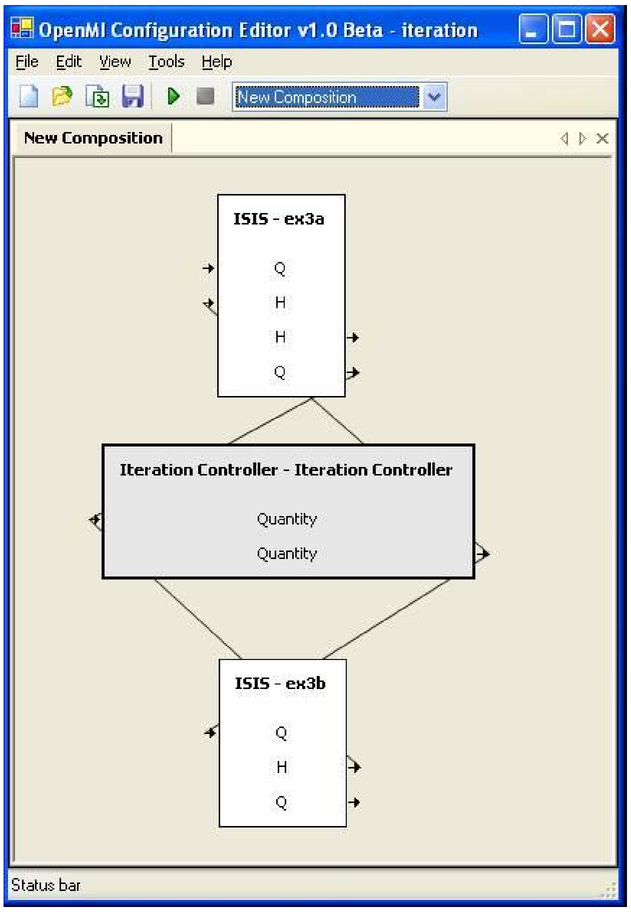

OpenMI Configuration Editor v1.0 Beta - iteration
File Edit View Tools Help
New Composition
New Composition
ISIS - ex3a
Q
H
H
Q
Iteration Controller - Iteration Controller
Quantity
Quantity
ISIS - ex3b
Q
H
Q
Status bar

Figure 6-1 Position of iteration controller within a model combination

The requirements of an IterationController are

- All bi-directional links are linked through the iteration controller and not directly. This is necessary because the iteration controller is in control of iterations. It allows the iteration controller to check that the values are converging.
- The iteration controller decides when to stop iterating, based on a stopping criterion.
- The iteration controller has a configurable number of minimum and maximum iterations. A flag is used to indicate if the iteration controller throws an exception when the maximum number of iterations is exceeded.
- The iteration controller is responsible for managing the simulation engine states during the iterations.
- The iteration controller is itself linkable component and it is triggered externally. From the outside the iteration controller and the linked simulation engines look like a LinkableComponent.
- The iteration controller does not have a Graphical User Interface (GUI), although a simple example GUI would be useful for testing. The GUI and the iteration controller should be completely separate components.

Before using an iterated link, it is highly recommended to try using a bi-directional link first. Only when there are strong backwater effects are iterated links really needed. Usually a bi-directional link with a small time-step will work, for instance for the example given here a bi-directional link with no iterations will gives the same result. A bi-directional link is much easier to set-up and will run much faster. Also, the models used will have to be able to restore their state to an earlier time, something that not all OpenMI compliant models will be able to do. Please check with the model supplier to ensure that the model supports restore state before trying to use any of the controllers.

6.1.3 Optimization controller

The main difference between optimization and calibration is that in calibration usually a large number of values are calibrated and in optimization usually a small number of values are calibrated. Optimization can also be to make a choice between different scenarios instead of finding the best value of a parameter. Optimization may also be the choice between different control strategies or to optimize a control strategy.

Optimization problems are made up of three basic ingredients:

- An objective function that we want to minimize or maximize. The objective function can be linear or non-linear.
- A set of unknowns or variables which affect the value of the objective function. The variables can be continuous or discrete or mixed.
- A set of constraints that allow the unknowns to take on certain values but exclude others. The constraints can be linear or non-linear.

The optimization controller is a generic component for finding the values of the variables that minimize or maximize an objective function while satisfying a set of constraints. The optimization controller will have a generic interface so that any optimization algorithm can be linked to it.

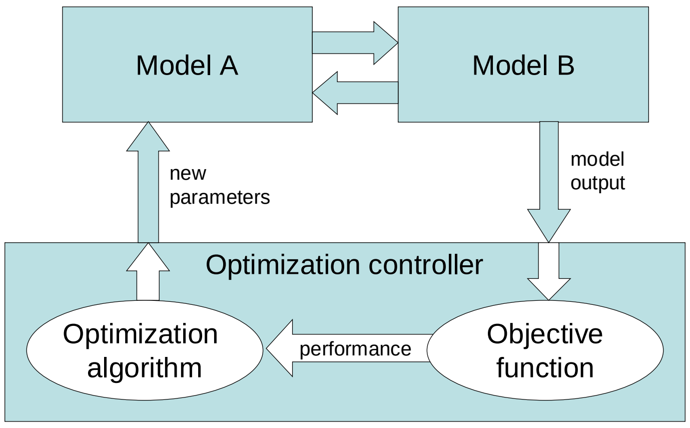


Figure 6-2 Position of the optimization controller (blue is OpenMI compliant)

The requirements that have been kept in mind for the OptimizationController are:

- Specify an objective function. The objective function could be a model in itself, with one output value. This would allow for good flexibility in the system because it is easy to change the objective function by just plugging in another one.

- Specify if the objective function should be minimized or maximized. This could be set during the initialization of the optimization controller.
- Specify the variables to be optimised. This could be done by linking variables in models to the optimisation controller.
- Specify constraints
    - Specify a minimum and maximum for variables
    - Specify linear constraints
    - Specify non-linear constraints
- Specify a starting value for variables.
- The optimization controller will raise events about optimization progress.
- The optimization controller will raise an error event when no solution can be found that satisfies all constraints.
- Manual and automatic optimization should be possible. For manual optimization, it should be possible to adjust values.
- The optimization algorithm (linear programming, dynamic programming, conjugate gradient, etc.) should be user definable and should plug in to the optimization controller so that it is easy to plug in different algorithms.

6.1.4 Calibration controller

Although models can be calibrated individually, when models are linked together into a linked model this linked model has to be calibrated as well. The calibration controller is a generic component to calibrate linked models. It will provide a generic interface so that specific calibration algorithms can be used. In most cases the objective of calibration is to minimise an error function. The error function is usually defined as the difference between measured data and calculated data. The purpose of the calibration is to find the set of parameter values that minimise the error function. Calibration can be seen as an optimisation problem because the error function is optimised and the parameter values are the unknowns in the optimisation. Figure 6-3 illustrates the data flow around a calibration controller.

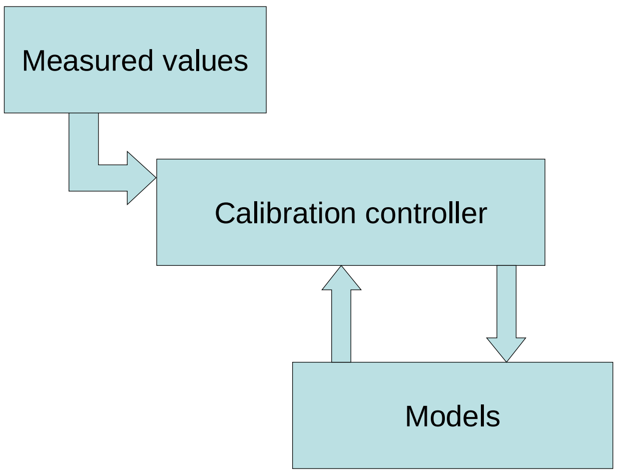


Figure 6-3 Position of calibration controller

The requirements of a CalibrationController are:

- Manual and automatic calibration should be possible. With the manual calibration, the user should be able to change parameter values and run the simulation. With automatic calibration, the user should be able to choose the parameters that are going to be calibrated.
- The calibration component should be able to compare computed with measured values and calculate statistics for the calibration.
- A mechanism is needed to specify the following:
  - Model parameters that can be changed. The simulation engines will perform a GetValues() call to the calibration controller to retrieve the model parameters.
  - Minimum and maximum value for each parameter.
  - Initial value for each parameter.
- A mechanism is needed to change the parameter values after each iteration.
- A mechanism is needed to decide when to stop calibrating.
- A mechanism to make the calibrated values “fixed”, e.g. to write them out to the input file. Once values are calibrated, the parameters are not inputs any longer. It should be possible to make calibrated values persistent.
- The calibration controller will send events about calibration progress and statistics

6.1.5 Logical switch

The logical switch is used to switch between different inputs depending on a logical condition. An example of a logical condition is a threshold for a water level or a threshold for a flow.

6.2 Static View

Figure 6-4 shows the UML diagram for the advanced controllers. The generic Controller class is derived from LinkableComponent and both the IterationController and OptimizationController classes derive from the Controller class. As can be seen, the controllers contain an internal buffer which holds buffer elements.

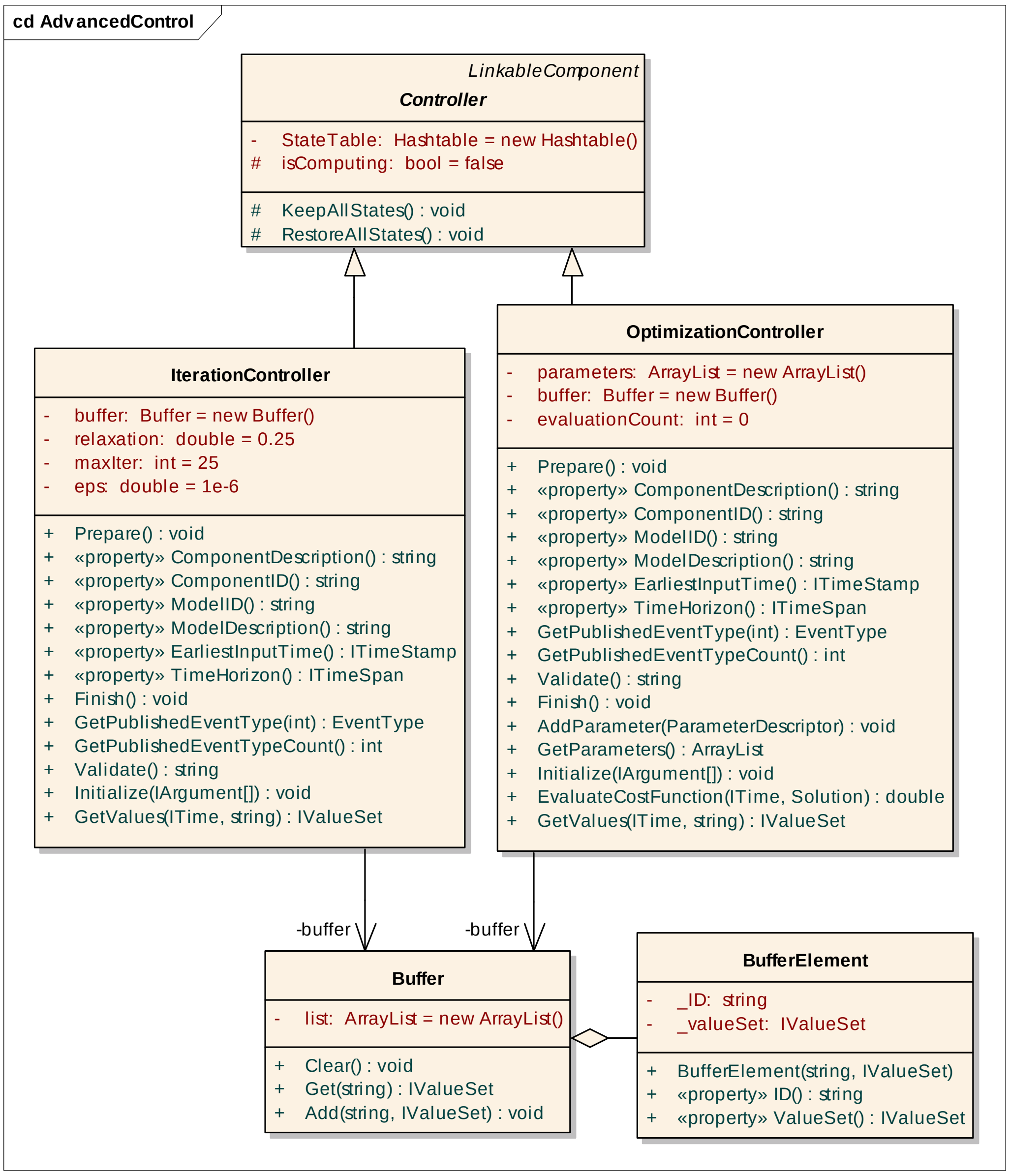

cd AdvancedControl
LinkableComponent
Controller
- StateTable: Hashtable = new Hashtable()
# isComputing: bool = false
# KeepAllStates() : void
# RestoreAllStates() : void
IterationController
- buffer: Buffer = new Buffer()
- relaxation: double = 0.25
- maxIter: int = 25
- eps: double = 1e-6
+ Prepare() : void
+ «property» ComponentDescription() : string
+ «property» ComponentID() : string
+ «property» ModelID() : string
+ «property» ModelDescription() : string
+ «property» EarliestInputTime() : ITimeStamp
+ «property» TimeHorizon() : ITimeSpan
+ Finish() : void
+ GetPublishedEventType(int) : EventType
+ GetPublishedEventTypeCount() : int
+ Validate() : string
+ Initialize(IArgument[]) : void
+ GetValues(ITime, string) : IValueSet
OptimizationController
- parameters: ArrayList = new ArrayList()
- buffer: Buffer = new Buffer()
- evaluationCount: int = 0
+ Prepare() : void
+ «property» ComponentDescription() : string
+ «property» ComponentID() : string
+ «property» ModelID() : string
+ «property» ModelDescription() : string
+ «property» EarliestInputTime() : ITimeStamp
+ «property» TimeHorizon() : ITimeSpan
+ GetPublishedEventType(int) : EventType
+ GetPublishedEventTypeCount() : int
+ Validate() : string
+ Finish() : void
+ AddParameter(ParameterDescriptor) : void
+ GetParameters() : ArrayList
+ Initialize(IArgument[]) : void
+ EvaluateCostFunction(ITime, Solution) : double
+ GetValues(ITime, string) : IValueSet
-buffer
-buffer
Buffer
- list: ArrayList = new ArrayList()
+ Clear() : void
+ Get(string) : IValueSet
+ Add(string, IValueSet) : void
BufferElement
- _ID: string
- _valueSet: IValueSet
+ BufferElement(string, IValueSet)
+ «property» ID() : string
+ «property» ValueSet() : IValueSet

Figure 6-4 The advanced controller classes

To enable convergence of a data set, additional range information is required about the parameter or quantity to be controlled. A ParameterDescriptor class (Figure 6-5) is used by the optimization and calibration controllers to describe the minimum, maximum, and default values for parameters.

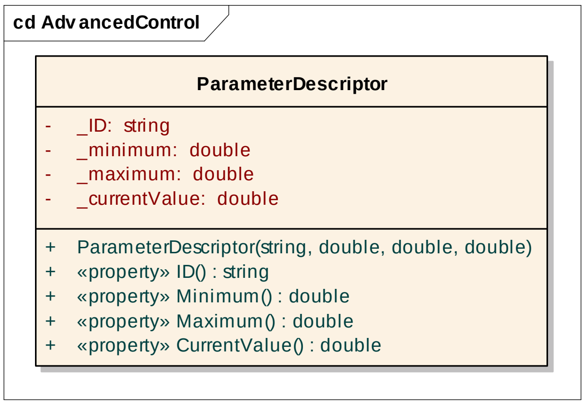

cd AdvancedControl
ParameterDescriptor
- _ID: string
- _minimum: double
- _maximum: double
- _currentValue: double
+ ParameterDescriptor(string, double, double, double)
+ «property» ID() : string
+ «property» Minimum() : double
+ «property» Maximum() : double
+ «property» CurrentValue() : double

Figure 6-5 ParameterDescriptor class

The optimization and iteration controllers have to store candidate solutions for their optimization problem. A Solution class is used for this purpose to store candidate solutions (Figure 6-6).

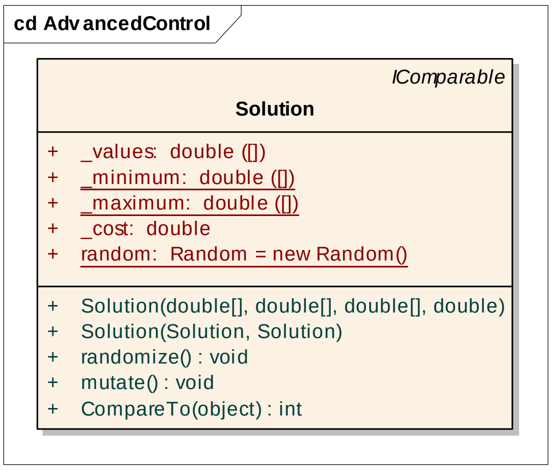


Figure 6-6 Solution class to store candidate optimal solutions

The optimization and iteration controller require a mathematical algorithm to assess the objective function in order to identify optimal solutions. For demonstration purposes a sum of squared differences is used. This algorithm has been implemented as a LinkableComponent (see Figure 6-7) to illustrate that any mathematical assessment could be implemented in a similar way. The controllers them selves use an internal search algorithm (a genetic algorithm) to identify new candidate solutions, but this action could also passed on to an external algorithm.

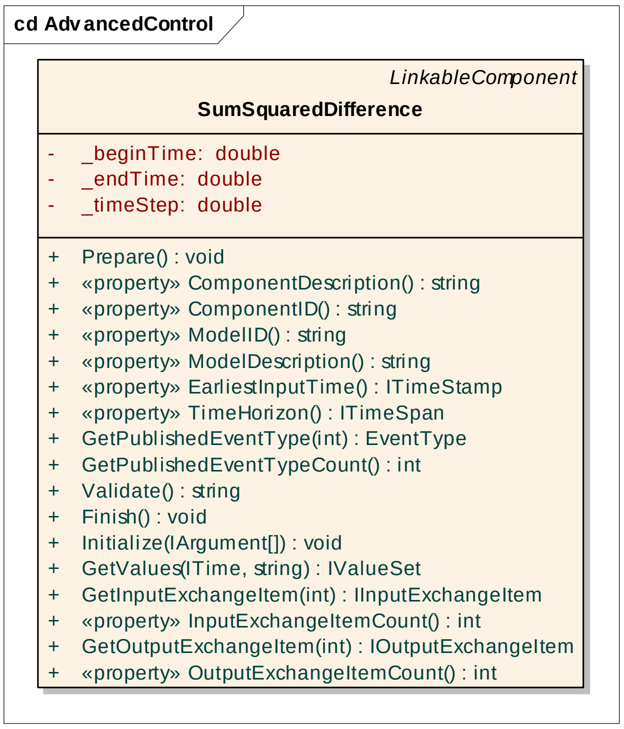

cd AdvancedControl
LinkableComponent
SumSquaredDifference
- _beginTime: double
- _endTime: double
- _timeStep: double
+ Prepare() : void
+ «property» ComponentDescription() : string
+ «property» ComponentID() : string
+ «property» ModelID() : string
+ «property» ModelDescription() : string
+ «property» EarliestInputTime() : ITimeStamp
+ «property» TimeHorizon() : ITimeSpan
+ GetPublishedEventType(int) : EventType
+ GetPublishedEventTypeCount() : int
+ Validate() : string
+ Finish() : void
+ Initialize(IArgument[]) : void
+ GetValues(ITime, string) : IValueSet
+ GetInputExchangeItem(int) : IInputExchangeItem
+ «property» InputExchangeItemCount() : int
+ GetOutputExchangeItem(int) : IOutputExchangeItem
+ «property» OutputExchangeItemCount() : int

Figure 6-7 Example performance assessor: SumSquareDifference

6.3 Dynamic View

For the iteration controller, all interaction is based on state management and GetValues calls (see Figure 6-8, the sequence diagram associated to two rivers connected in Figure 6-1). Before iteration starts, the model state is saved so it can be restored at each iteration loop. Each iteration loop starts with restoring the initial state, after which the controller asks for a Q-lower bound (receiving a call back for an H-lowerbound) and an H-upperbound (receiving a call back for Q-lowerbound). After a relaxation by the controller, the updated **Q'** and **H'** are returned by the controller thus enabling the models to H and Q.

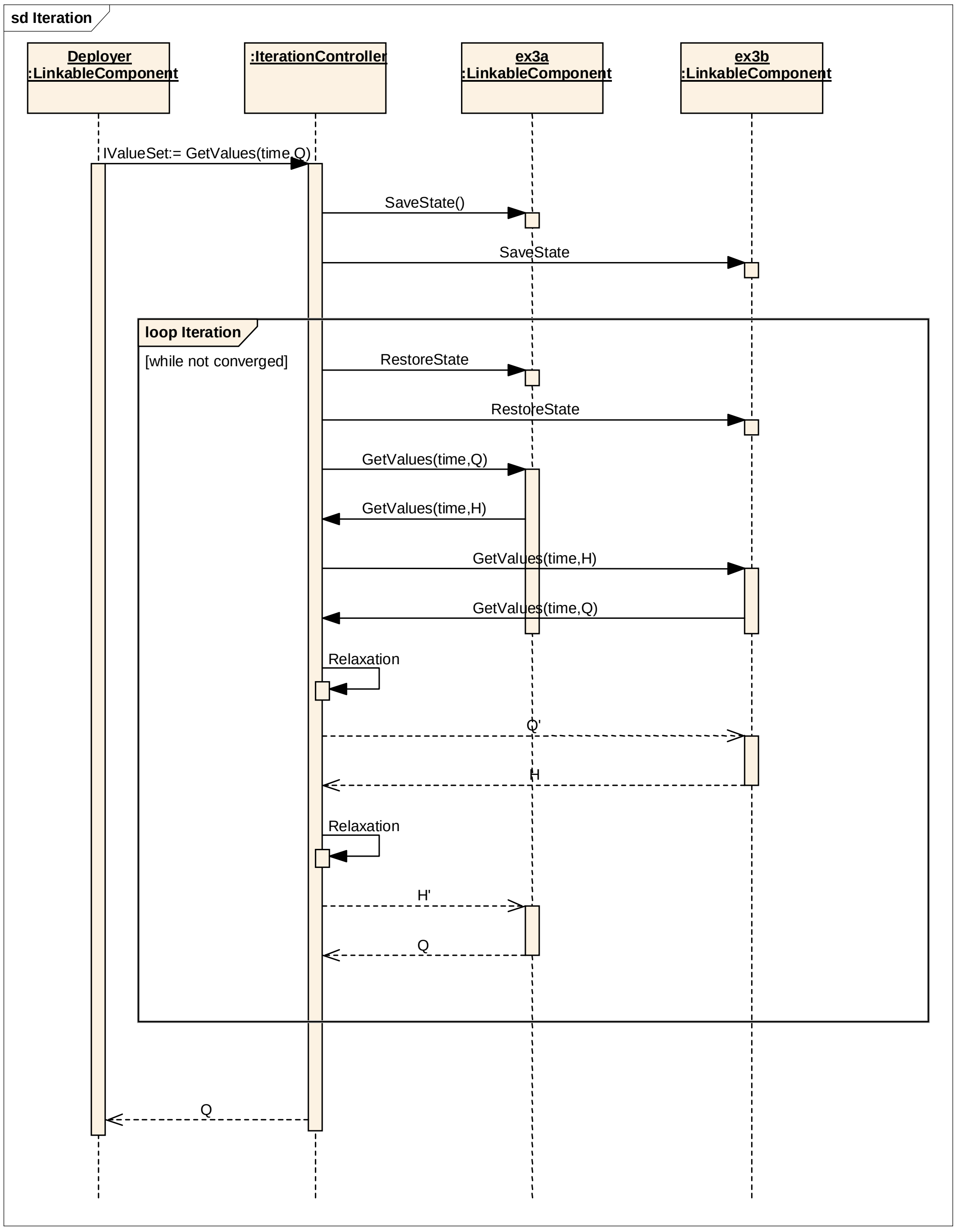


Figure 6-8 Sequence diagram for an iteration controller

The sequence diagram for the optimization controller (Figure 6-9) is much simpler. The optimization controller starts its optimization loop with a default parameter set and asks the Performance assessor (Cost Function) for the associated costs. To provide the costs, the CostFunction asks for the parameters and returns (after calculation) the costs. Using those costs, a new parameter set can be determined by the OptimizationController.

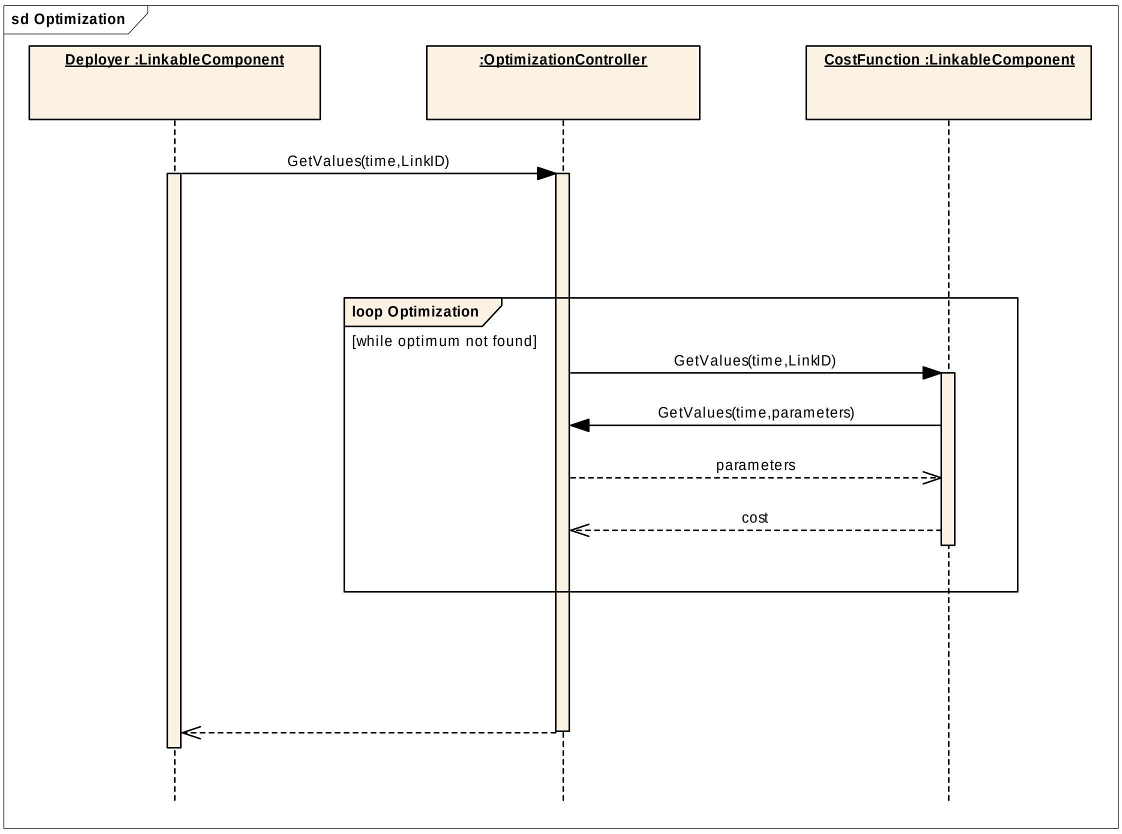


Figure 6-9 Sequence diagram for OptimizationController

The CalibrationController basically functions the same as the OptimizationController. However, it contains an extra control loop as it has to calibrate the model for an entire parameter set over all time steps. Figure 6-10 provides the associated sequence diagram.

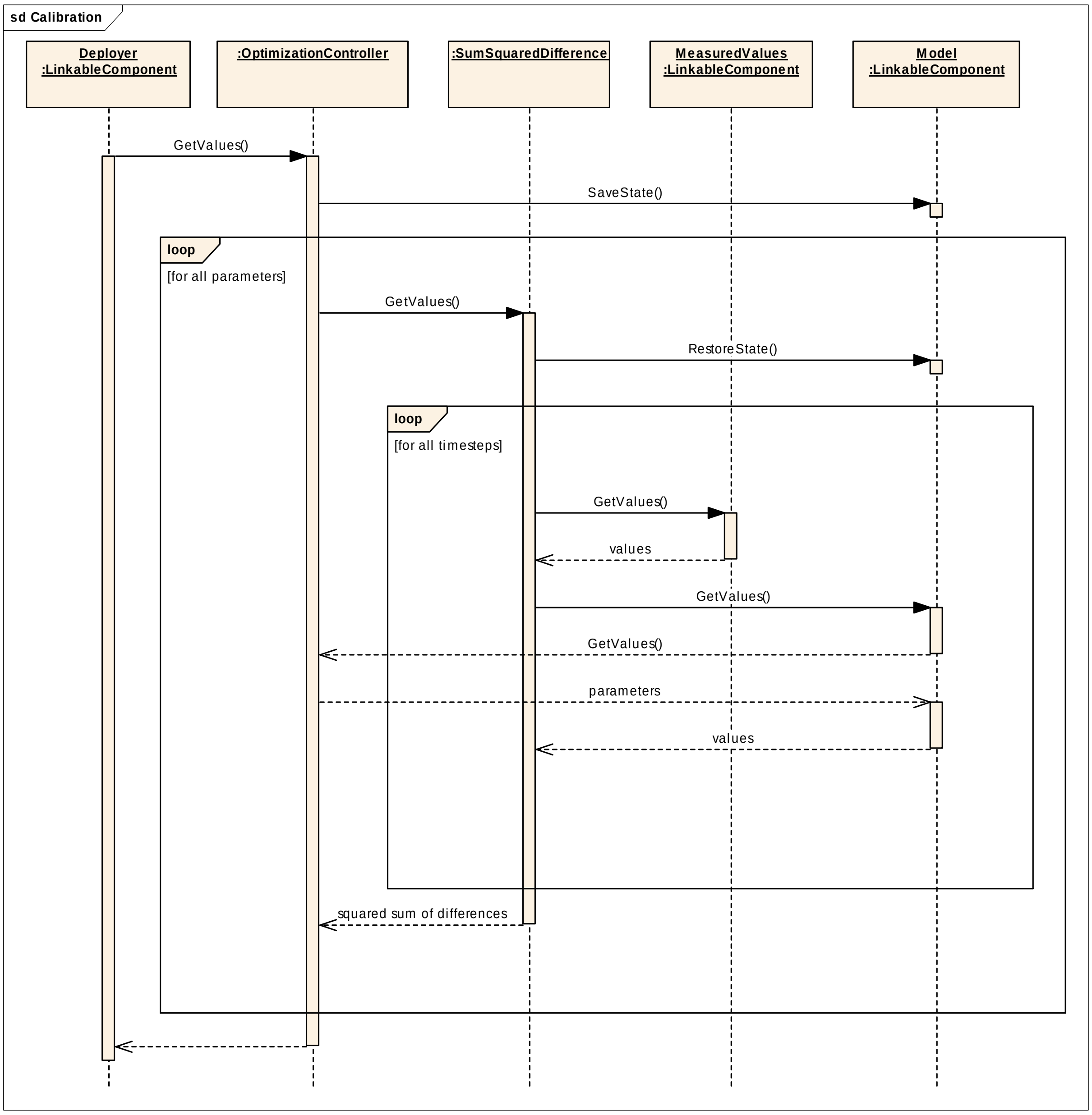


Figure 6-10 Sequence diagram of CalibrationController

6.4 Implementation remarks

6.4.1 C#-implementation

The C# implementation utilizes no specific .NET features. The classes require the System, the System.Diagnostics and the System.Collections assembly. The correct implementation of all methods has been tested using dedicated unit tests in combination with the NUnit framework for testing.

The org.OpenMI.Utilities.AdvancedControl package is available as open source under LGPL licence on Source Forge (http://sourceforge.net/projects/openmi)

6.4.2 Java implementation

Not yet available.

7 The org.OpenMI.Utilities.Configuration package

7.1 General description

The org.OpenMI.Utilities.Configuration namespace provide facilities to setup (i.e. configure), save/retrieve and deploy a combination of linked models. Three functional levels have been identified:

- administer the composition, i.e. the network of links and linked components
- persistent storage and retrieval in XML (parsing and serializing by the generic XML parser of the org.OpenMI.DevelopmentSupport package, customized by an XmlConfiguration class)
- run a composition, performed by the SystemDeployer

Note that this package has been developed to be support the incorporation of OpenMI in existing GUI's. The straightforward graphical user interface shipped with the open source release of OpenMI (i.e. OmiEd) does not utilize this package.

7.2 Static View

The following diagram displays the composition and classes it needs.

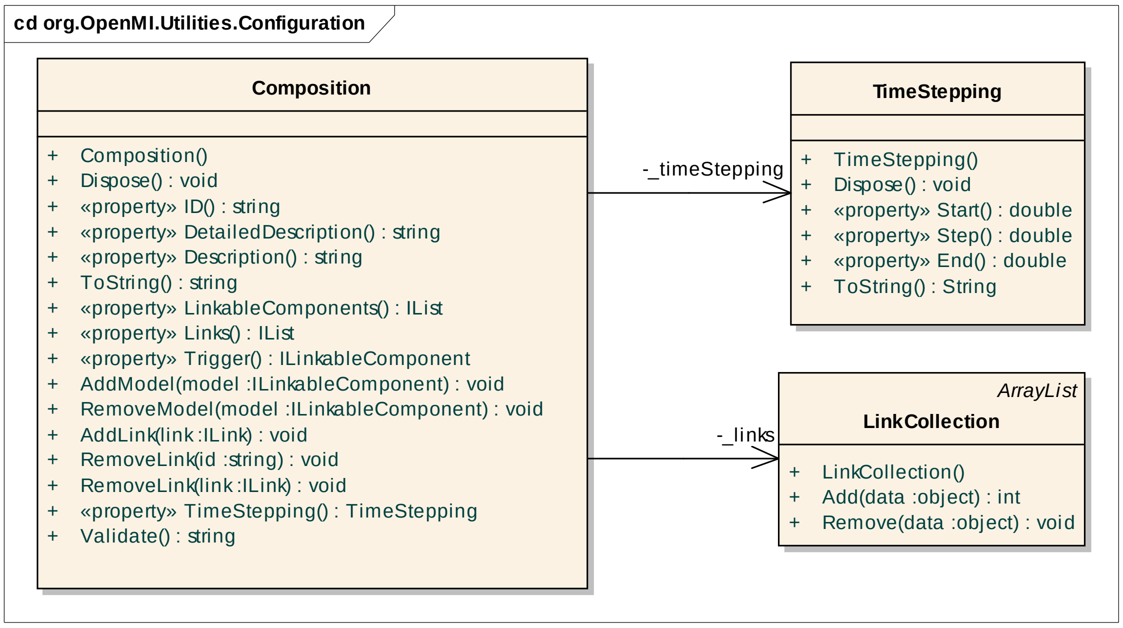

cd org.OpenMI.Utilities.Configuration
Composition
+ Composition()
+ Dispose() : void
+ «property» ID() : string
+ «property» DetailedDescription() : string
+ «property» Description() : string
+ ToString() : string
+ «property» LinkableComponents() : IList
+ «property» Links() : IList
+ «property» Trigger() : ILinkableComponent
+ AddModel(model :ILinkableComponent) : void
+ RemoveModel(model :ILinkableComponent) : void
+ AddLink(link :ILink) : void
+ RemoveLink(id :string) : void
+ RemoveLink(link :ILink) : void
+ «property» TimeStepping() : TimeStepping
+ Validate() : string
-_timeStepping
TimeStepping
+ TimeStepping()
+ Dispose() : void
+ «property» Start() : double
+ «property» Step() : double
+ «property» End() : double
+ ToString() : String
-_links
ArrayList
LinkCollection
+ LinkCollection()
+ Add(data :object) : int
+ Remove(data :object) : void

Figure 7-1 Composition and needed classes

The Composition class keeps a set of linkable components and links together. It consists basically of a collection of links and a collection of linkable components. The Composition takes care that the links in this collection are synchronized with the links added to the linkable components through the methods in the Composition class. Except these collections a trigger component and time stepping information are essential properties. The Validate method is essential for validation of the composition.

Method / Property	**Notes**
Composition ()	Default constructor
Dispose ()	Default dispose
ID *: string*	Unique identification of the composition
DetailedDescription *: string*	Meaningful description for the end user
Description *: string*	Meaningful name for the end user
ToString() *: string*	Name of the object in various win-controls
LinkableComponents *: IList*	List of all linkable components used in the composition
Links *: IList*	List of all links in the composition. Adding or removing a link from this list adds / removes that link also to / from the linkable components specified in the link
Trigger *: ILinkableComponent*	Linkable component which is to be invoked as the first in the chain of linkable components
AddModel (*ILinkableComponent*)	Adds a linkable component to the composition
RemoveModel (*ILinkableComponent*)	Removes a linkable component from the composition
AddLink (*ILink*)	Adds a link to the composition and to the linkable components specified in the link
RemoveLink (*link id*)	Removes a link from the composition and from the linkable components specified in the link
RemoveLink (*ILink*)	Removes a link from the composition and from the linkable components specified in the link
TimeStepping *: TimeStepping*	Definition of start, step and end time of the composition
Validate () *: string*	Validates whether the composition can be run. Returns an empty string if okay, otherwise error message. The following checks are performed: 1. Trigger has been set 2. Trigger is part of the list of linkable components 3. The trigger has at least one output exchange item (otherwise it cannot be asked for values to initiate the computation) 4. The component id and model id are set for each linkable component 5. The time stepping information has been set 6. The time step has a value unequal to zero 7. All links have an id and they are all different. 8. All links have a source and target linkable component, source and target element set and source and target quantity and they all have an id unequal to null. 9. The units in the quantities in the links have been set. 10. The data operations in the links have an id. 11. The Validate methods of the linkable components are invoked

The LinkCollection is a collection of links used in the composition. Adding or removing a link from this collection adds / removes that link also to / from the linkable components specified in the link.

Method	**Notes**
LinkCollection ()	Empty constructor
Add (*object*) *: int*	Adds a link to the collection and adds the link to the specified linkable components in the link. Ignores an object not implementing ILink. Throws an exception if the ID of the link is not unique within this collection. Returns position in collection, -1 if not added
Remove (*object*)	Removes a link from the collection and removes the link from the linkable components specified in the link. Ignores an object not implementing ILink.

The TimeStepping class defines the period over which a composition performs a computation

Method / Property	**Notes**
TimeStepping ()	Default constructor
Finalize ()	Default Finalize (takes no action)
Dispose ()	Default Dispose (takes no action)
Start *: double*	Start date and time of the computation. The date and time is expressed in modified julian date (number of days since November 17, 1858)
Step *: double*	Time step in seconds
End *: double*	End date and time of the computation. The date and time is expressed in modified julian date (number of days since November 17, 1858)
ToString() *: string*	Representation in various win controls. Returns string containing start, end and step values

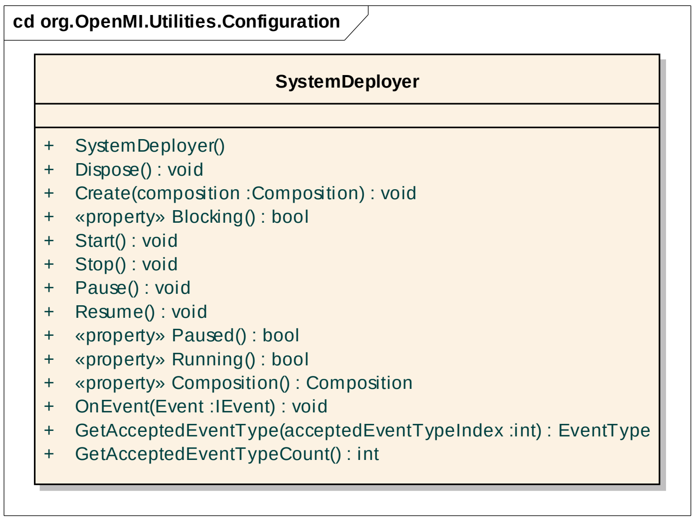

cd org.OpenMI.Utilities.Configuration
SystemDeployer
+ SystemDeployer()
+ Dispose() : void
+ Create(composition :Composition) : void
+ «property» Blocking() : bool
+ Start() : void
+ Stop() : void
+ Pause() : void
+ Resume() : void
+ «property» Paused() : bool
+ «property» Running() : bool
+ «property» Composition() : Composition
+ OnEvent(Event :IEvent) : void
+ GetAcceptedEventType(acceptedEventTypeIndex :int) : EventType
+ GetAcceptedEventTypeCount() : int

Figure 7-2 System Deployer

The SystemDeployer is a class for running a Composition. Therefore there is a method (Create) to accept a Composition. The Composition can be run in a separate thread if desired. In that case it can be paused, resumed and killed.

Method / Property	**Notes**
SystemDeployer ()	Default constructor
Finalize ()	Default finalizer
Dispose ()	Default dispose
Create (*composition*)	Prepares the controller for running a composition. Performs validation and throws an exception if not okay. Adds a dummy linkable component for invoking the trigger linkable component in the composition.
Blocking *: bool*	Indicates whether the calling application will be blocked during the execution. A blocked application is an application that doesn't respond to any requests, either programmatically or via a user interface. If blocking is false, the composition run is executed in a separate thread. Recommended value: false for user interface applications, true for console applications.
Start ()	Starts the execution of the composition run
Stop ()	Immediately stops the execution of the composition run. Ignored if blocking is true or not running.
Pause ()	Pauses the execution of the composition run. Ignored if there is no run.
Resume ()	Continues the execution of the composition run after it has been paused, otherwise ignored.
Paused *: bool*	Indicates whether the execution is paused at the moment.
Running *: bool*	Indicates whether the execution is running at the moment. If paused, this value returns true.
Composition *: composition*	The composition the deployer is configured for.

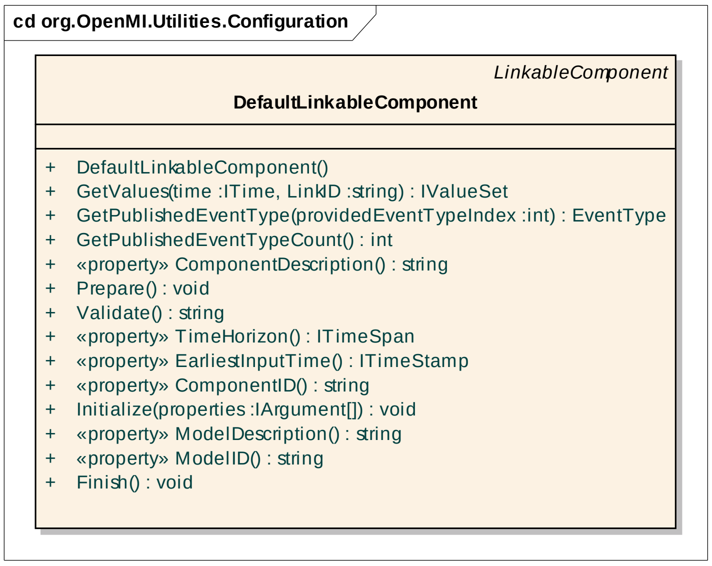

cd org.OpenMI.Utilities.Configuration
LinkableComponent
DefaultLinkableComponent
+ DefaultLinkableComponent()
+ GetValues(time :ITime, LinkID :string) : IValueSet
+ GetPublishedEventType(providedEventTypeIndex :int) : EventType
+ GetPublishedEventTypeCount() : int
+ «property» ComponentDescription() : string
+ Prepare() : void
+ Validate() : string
+ «property» TimeHorizon() : ITimeSpan
+ «property» EarliestInputTime() : ITimeStamp
+ «property» ComponentID() : string
+ Initialize(properties :IArgument[]) : void
+ «property» ModelDescription() : string
+ «property» ModelID() : string
+ Finish() : void

Figure 7-3 Default Linkable Component

The default linkable component is a component which is used as a dummy linkable component. It is instantiated by the deployer in order to be able to ask the trigger component for values.

Method	**Notes**
DefaultLinkableComponent ()	Empty constructor
GetValues (*time, link id*) : *IValueSet*	Empty implementation, returns null. Sets the current time to the requested time.
GetPublishedEventType (*int*) : *EventType*	Empty implementation, raises exception when called
GetPublishedEventTypeCount () *: int*	Gets number of published event types, being zero
ComponentDescription *: string*	Gets component description (empty string)
Prepare ()	Prepares for execution (takes no action)
Validate () *: string*	Validates established links (takes no action)
TimeHorizon ()	Dummy implementation for time horizon (returns null)
EarliestInputTime (): *time*	Returns current time
ComponentID *: string*	Component ID (empty string)
Initialize (*arguments*)	Default implementation of Initialize (takes no action)
ModelDescription *: string*	Model description (empty string)
ModelID *: string*	Model ID (empty string)
Finish ()	Finishes computation (takes no action)

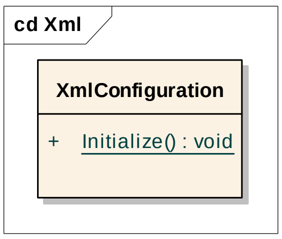


Figure 7-4 XmlConfiguration

The class XmlConfiguration holds all MetaInfo needed for proper functioning of XmlFile. In this case proper functioning means that it can read and write xml files for linkable components and compositions.

Method	**Notes**
Initialize ()	Configures XmlFile for reading and writing compositions and linkable components. This method must be called before the first read or write action with XmlFile.

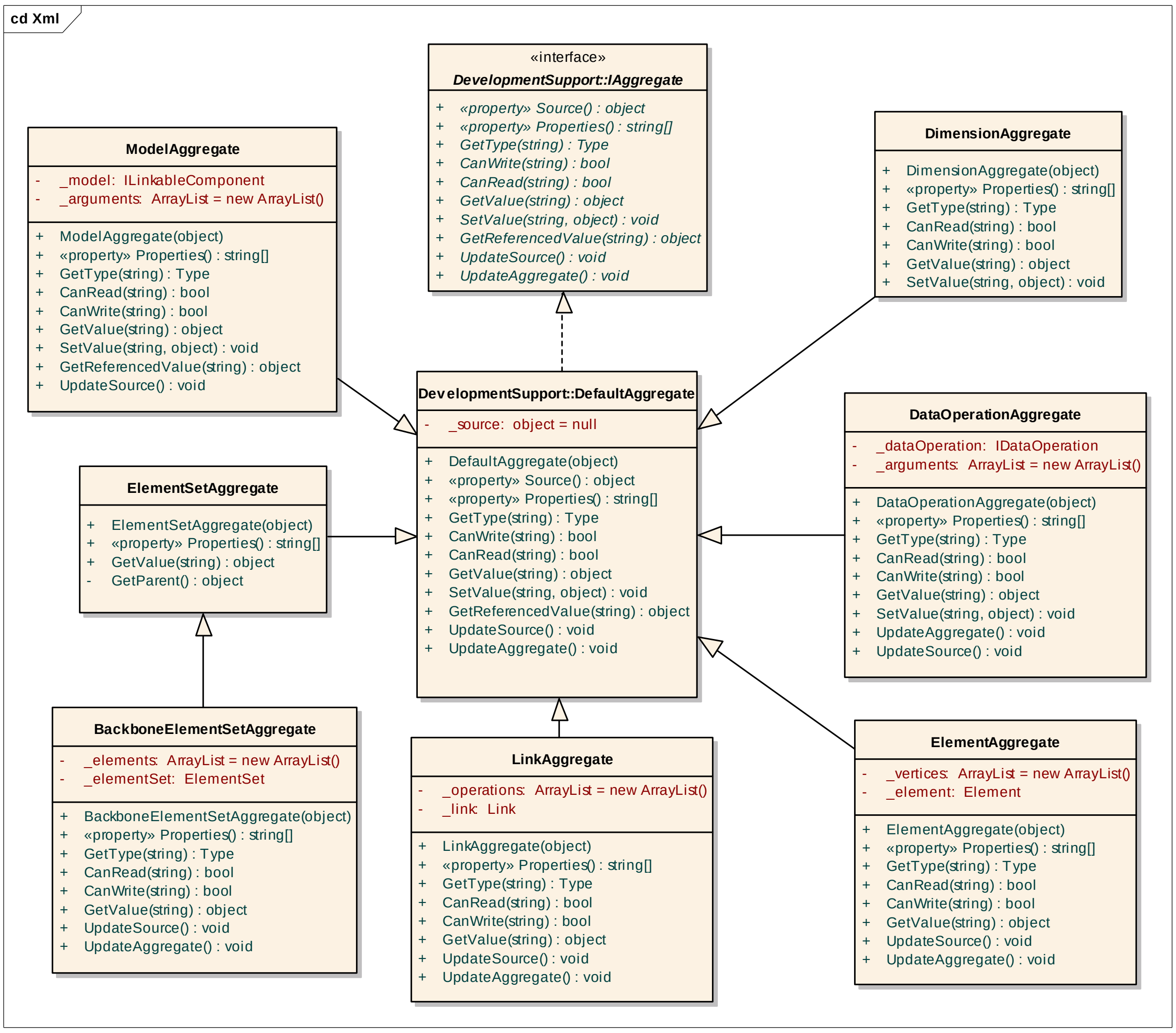

cd Xml
«interface»
DevelopmentSupport::IAggregate
+ «property» Source() : object
+ «property» Properties() : string[]
+ GetType(string) : Type
+ CanWrite(string) : bool
+ CanRead(string) : bool
+ GetValue(string) : object
+ SetValue(string, object) : void
+ GetReferencedValue(string) : object
+ UpdateSource() : void
+ UpdateAggregate() : void
ModelAggregate
- _model: ILinkableComponent
- _arguments: ArrayList = new ArrayList()
+ ModelAggregate(object)
+ «property» Properties() : string[]
+ GetType(string) : Type
+ CanRead(string) : bool
+ CanWrite(string) : bool
+ GetValue(string) : object
+ SetValue(string, object) : void
+ GetReferencedValue(string) : object
+ UpdateSource() : void
DimensionAggregate
+ DimensionAggregate(object)
+ «property» Properties() : string[]
+ GetType(string) : Type
+ CanRead(string) : bool
+ CanWrite(string) : bool
+ GetValue(string) : object
+ SetValue(string, object) : void
DevelopmentSupport::DefaultAggregate
- _source: object = null
+ DefaultAggregate(object)
+ «property» Source() : object
+ «property» Properties() : string[]
+ GetType(string) : Type
+ CanWrite(string) : bool
+ CanRead(string) : bool
+ GetValue(string) : object
+ SetValue(string, object) : void
+ GetReferencedValue(string) : object
+ UpdateSource() : void
+ UpdateAggregate() : void
DataOperationAggregate
- _dataOperation: IDataOperation
- _arguments: ArrayList = new ArrayList()
+ DataOperationAggregate(object)
+ «property» Properties() : string[]
+ GetType(string) : Type
+ CanRead(string) : bool
+ CanWrite(string) : bool
+ GetValue(string) : object
+ SetValue(string, object) : void
+ UpdateAggregate() : void
+ UpdateSource() : void
ElementSetAggregate
+ ElementSetAggregate(object)
+ «property» Properties() : string[]
+ GetValue(string) : object
- GetParent() : object
BackboneElementSetAggregate
- _elements: ArrayList = new ArrayList()
- _elementSet: ElementSet
+ BackboneElementSetAggregate(object)
+ «property» Properties() : string[]
+ GetType(string) : Type
+ CanRead(string) : bool
+ CanWrite(string) : bool
+ GetValue(string) : object
+ UpdateSource() : void
+ UpdateAggregate() : void
LinkAggregate
- _operations: ArrayList = new ArrayList()
- _link: Link
+ LinkAggregate(object)
+ «property» Properties() : string[]
+ GetType(string) : Type
+ CanRead(string) : bool
+ CanWrite(string) : bool
+ GetValue(string) : object
+ UpdateSource() : void
+ UpdateAggregate() : void
ElementAggregate
- _vertices: ArrayList = new ArrayList()
- _element: Element
+ ElementAggregate(object)
+ «property» Properties() : string[]
+ GetType(string) : Type
+ CanRead(string) : bool
+ CanWrite(string) : bool
+ GetValue(string) : object
+ UpdateSource() : void
+ UpdateAggregate() : void

Figure 7-5 Implementations of IAggregate

A number of implementations exist of the interface IAggregate. They are needed as a bridge between XmlFile (in org.OpenMI.DevelopmentSupport) and some classes which can't expose their properties in a generic way. For example, a dimension (IDimension) doesn't have properties all possible dimensions (length, time, etc.), but exposes this information through other methods (in this case GetPower). However, the xml file we want to read and write does have xml elements for length, time, etc. The aggregate DimensionAggregate acts as a bridge between these xml elements and the methods of IDimension.

ModelAggregate : IAggregate for linkable components

Method / Property	**Notes**
ModelAggregate (*object*)	Default constructor, taking the underlying linkable component as argument
Properties *: string[]*	Gets the properties of the linkable component, being only "Arguments"
GetType (*property*) *: type*	Gets the type of the property
CanRead (*property*) : *bool*	Tells whether a property can be read. Always true, since only calls are expected for property "Arguments"

CanWrite (*property*) : *bool*	Tells whether a property can be written. Always false
GetValue (*property*) *: object*	Gets the value of a property, i.e. the list of arguments
SetValue (*property, object*)	Sets the value of a property. Takes no action.
GetReferencedValue (*reference*)	Gets a referenced value. If the reference equals "<TypeName>";"<ID>", where the type name denotes a class implementing IElementSet, the element set with the specified ID is searched in all exchange items of the linkable component. This is useful because element sets are not stored in xml files, but produced by the linkable component (which probably reads them somewhere from its own files). However, the element sets can be referenced in xml files (especially in links in compositions). When reading such an xml file, an element set must be assigned to the link. That element set is retrieved from the linkable component by this method. In all other cases the call is delegated to XmlFile.GetRegisteredTarget
UpdateSource ()	Updates the underlying linkable component by calling its Initialize method with the arguments

LinkAggregate : IAggregate for links (defined in the backbone)

Method / Property	Notes
LinkAggregate (*object*)	Default constructor, taking the underlying link as argument
Properties *: string[]*	Gets a list of properties which are accessed in a generic way.
GetType (*property*) *: type*	Class type of a property
CanRead (*property*) : *bool*	Tells whether a property can be read
CanWrite (*property*) : *bool*	Tells whether a property can be written
GetValue (*property*) *: object*	Gets the value of a property
UpdateSource ()	Updates the underlying link by adding data operations to it
UpdateAggregate ()	Gets the arguments from the underlying link so that they can be accessed with GetValue calls

ElementSetAggregate : IAggregate for element sets

Method / Property	Notes
ElementSetAggregate (*object*)	Default constructor, taking the underlying element set as argument
Properties *: string[]*	Gets a list of properties which are accessed in a generic way.
GetValue (*property*) *: object*	Gets the value of a property. If the property is “Parent”, The linkable component which owns the element set is returned. To find this linkable component, all exchange items of linkable components registered in XmlFile are examined. In XmlConfiguration is specified that the XmlParent of IElementSet is “Parent”. See XmlFile documentation how the tag XmlParent is processed.

BackboneElementSetAggregate: IAggregate for element sets (defined in the backbone)

Method / Property	**Notes**
BackboneElementSetAggregate (*object*)	Default constructor, taking the underlying object as argument
Properties *: string[]*	List of properties which can be accessed in a generic way
GetType (*property*) *: type*	Class type of a property
CanRead (*property*) *: bool*	Tells whether a property can be read
CanWrite (*property*) *: bool*	Tells whether a property can be written
GetValue (*property*) *: object*	Gets the value of a property
UpdateSource ()	Updates the underlying element set by adding element to it with the AddElement method
UpdateAggregate ()	Updates the aggregate by retrieving elements from the element set

ElementAggregate : IAggregate for elements in the backbone

Method / Property	**Notes**
ElementAggregate (*object*)	Default constructor, taking the underlying element as argument
Properties *: string[]*	Gets a list of properties which can be accessed in a generic way
GetType (*property*) *: type*	Class type of a property
CanRead (*property*) *: bool*	Tells whether a property can be read
CanWrite (*property*) *: bool*	Tells whether a property can be written
GetValue (*property*) *: object*	Gets the value of a property
UpdateSource ()	Updates the underlying element by adding vertices to it with the AddVertex method
UpdateAggregate ()	Updates the aggregate by retrieving vertices from the element

DimensionAggregate : IAggregate for dimensions (defined in the backbone)

Method / Property	**Notes**
DimensionAggregate (*object*)	Default constructor, taking the underlying element as argument
Properties *: string[]*	Gets a list of all dimension bases
GetType (*property*) *: type*	Class type of a dimension value, always int
CanRead (*property*) *: bool*	Tells whether a dimension value can be read, always true
CanWrite (*property*) *: bool*	Tells whether a dimension value can be written, always true
GetValue (*property*) *: object*	Gets the value of a dimension
GetValue (*property*) *: object*	Sets the value of a dimension

DataOperationAggregate : IAggregate for data operations (defined in the backbone)

Method / Property	**Notes**
DataOperationAggregate (*object*)	Default constructor, taking the underlying object as argument
Properties *: string[]*	List of properties which can be accessed in a generic way
GetType (*property*) *: type*	Class type of a property

CanRead (*property*) *: bool*	Tells whether a property can be read
CanWrite (*property*) *: bool*	Tells whether a property can be written
GetValue (*property*) *: object*	Gets the value of a property
SetValue (*property, value*)	Sets the value of a property
UpdateAggregate ()	Gets the arguments from the underlying data operation so that they can be accessed with GetValue calls
UpdateSource ()	Updates the underlying data operation by calling its Initialize method with the arguments

7.3 XML Specification

This paragraph describes how the xml files of compositions should be built up. For validation of composition xml files, a schema is used: composition.xsd. Therefore describing the xml format of composition xml files comes down to describing the xsd file, since the xml parser of composition files accept all xml files which meet the composition.xsd file.

A graphical representation of this file on a high level is given below.

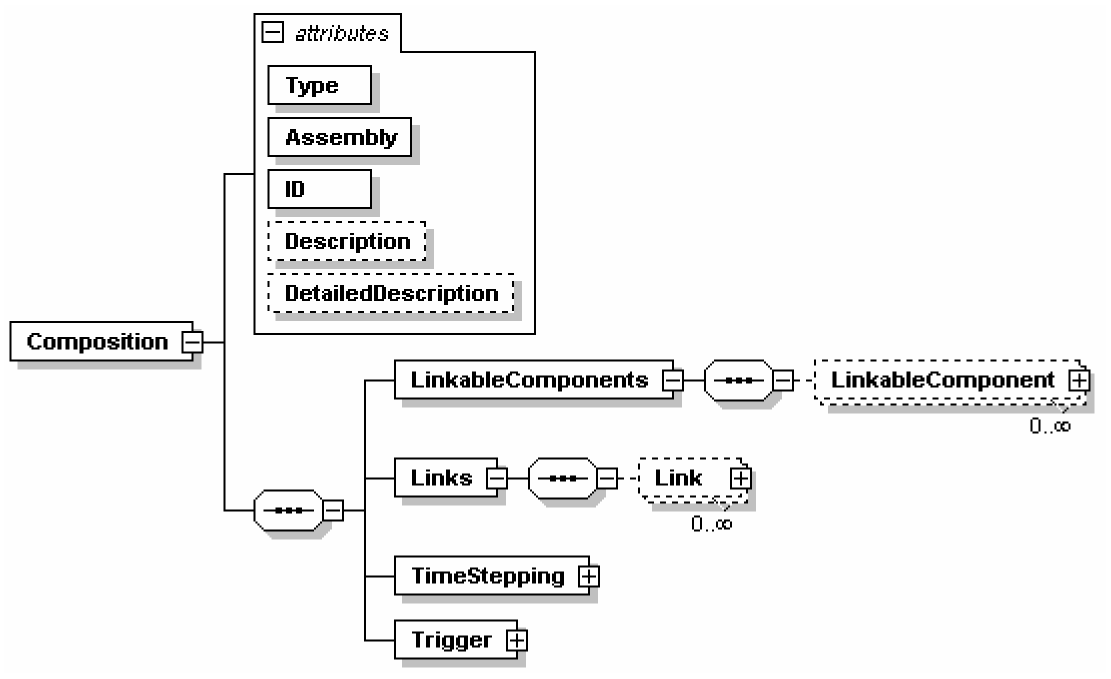


Figure 7-6 Composition.xsd overview

The attributes ID and Assembly of the composition specify which class to instantiate (being org.OpenMI.Utilities.Configuration.Composition or a subclass of it). The assembly is a full path to an assembly file or a full or partial name of an assembly in the GAC. See also XmlFile in org.OpenMI.DevelopmentSupport how XmlFile can parse xml files.

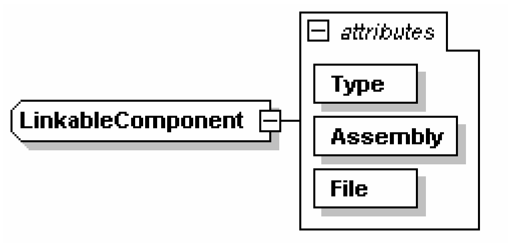


Figure 7-7 Composition.xsd linkable component definition

The linkable component only has three attributes, Type, Assembly and File. They are interpreted by XmlFile in org.OpenMI.DevelopmentSupport in its standard way: Type defines the type class of the linkable component to instantiate, Assembly defines the assembly where this type resides and file is a relative path to a file where subsequent properties of the linkable component are defined. This file is also known as the OMI file (see org.OpenMI.Standard).

The trigger element is expanded as a linkable component too

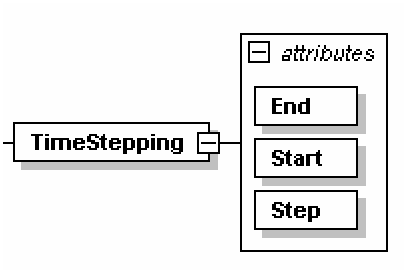


Figure 7-8 Composition.xsd time stepping definition

The begin, end and timestep are required and defined in the xml element TimeStepping.

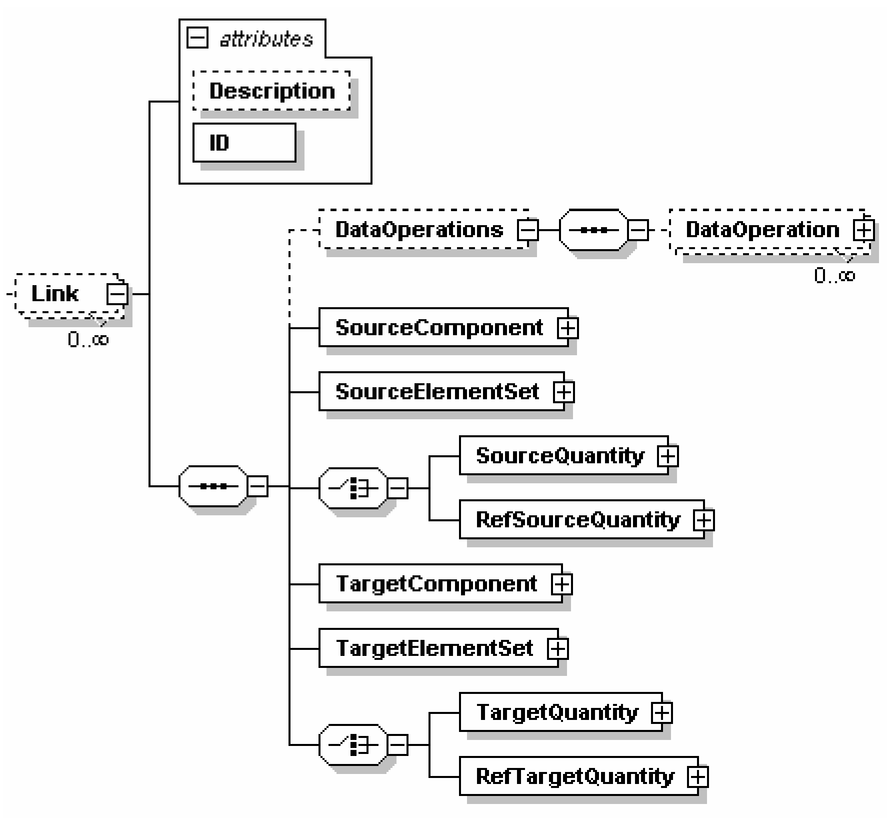


Figure 7-9 Composition.xsd link definition

The link structure in xml is similar to the interface ILink (see org.OpenMI.Standard), so it consists of an ID and description as attribute and a number of data operations, a source and target element set, quantity and attached linkable components.

A quantity is either defined as type Quantity or RefQuantity, depending on whether they have been written out fully or referencing an already defined quantity. SourceComponent and TargetComponent are both defined as LinkableComponents, SourceElementSet and TargetElementSet as ElementSets (linkable components and element sets are always defined as references). There is an optional list of data operations.

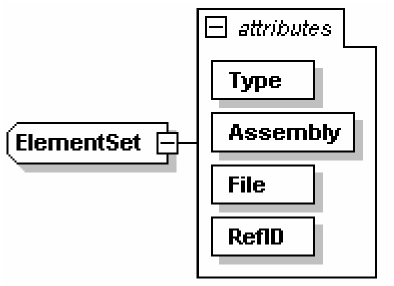


Figure 7-10 Composition.xsd element set definition

The element set is defined as an identifier in combination with a reference to the linkable component file (the OMI file). Since the linkable component is responsible for populating the element set and this is done often by the linkable component itself, it is most convenient to refer to the element set which has been created by the linkable component.

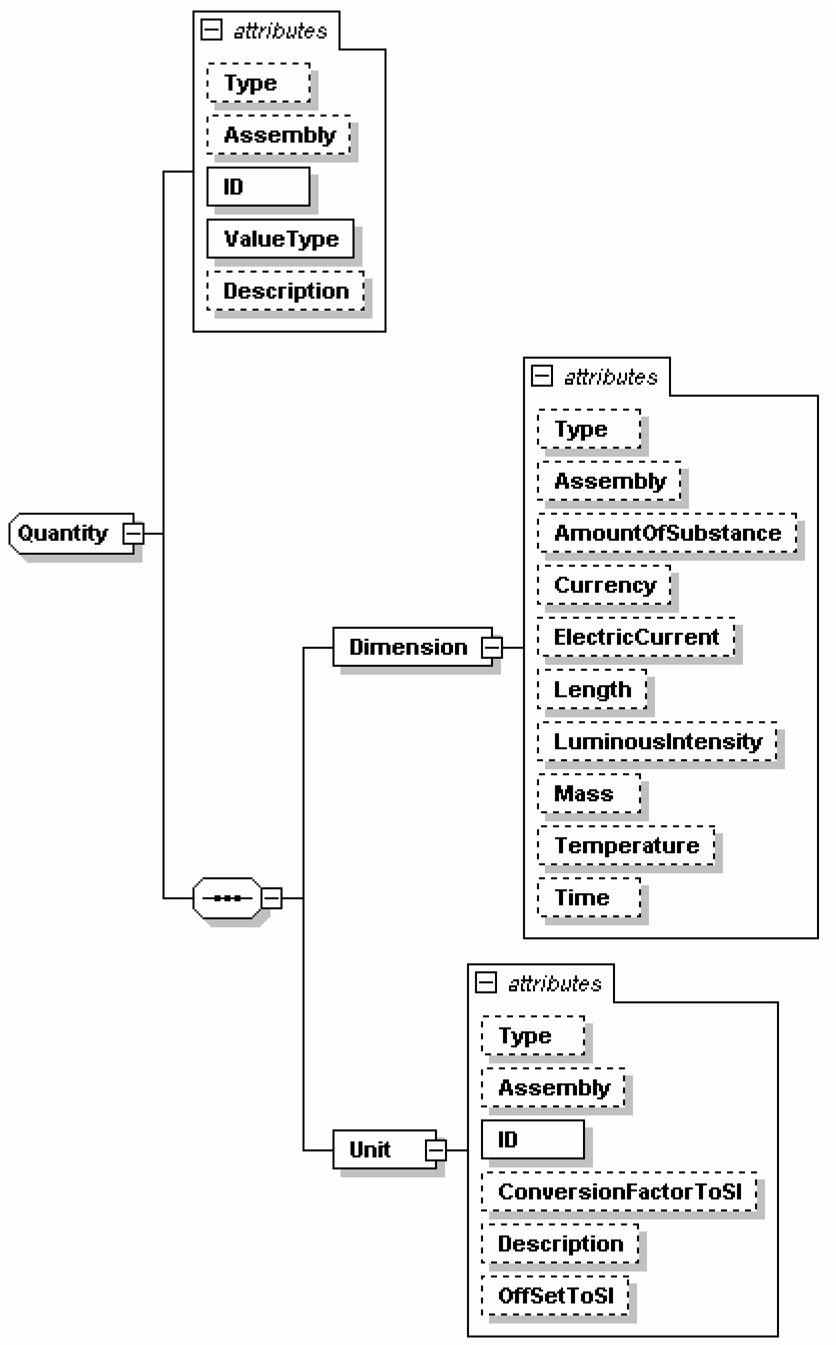


Figure 7-11 Composition.xsd quantity definition

The quantity consists of some attributes defining the type and the identifier and tow xml elements defining the dimension and unit. If attributes are omitted, their default value is assumed (zero for all dimensions and OffsetToSI, one for ConversionFactorToSI).

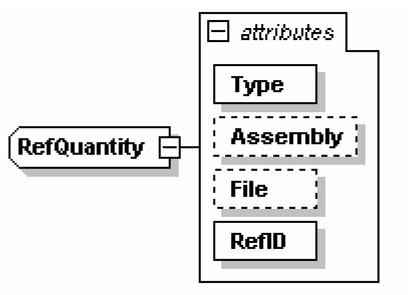


Figure 7-12 Composition.xsd quantity as reference definition

As a quantity can also be defined as a reference to an already known quantity, the xml element for the quantity can also consist of the usual xml attributes for a referenced object. These are the type, assembly, file and identifier of the quantity.

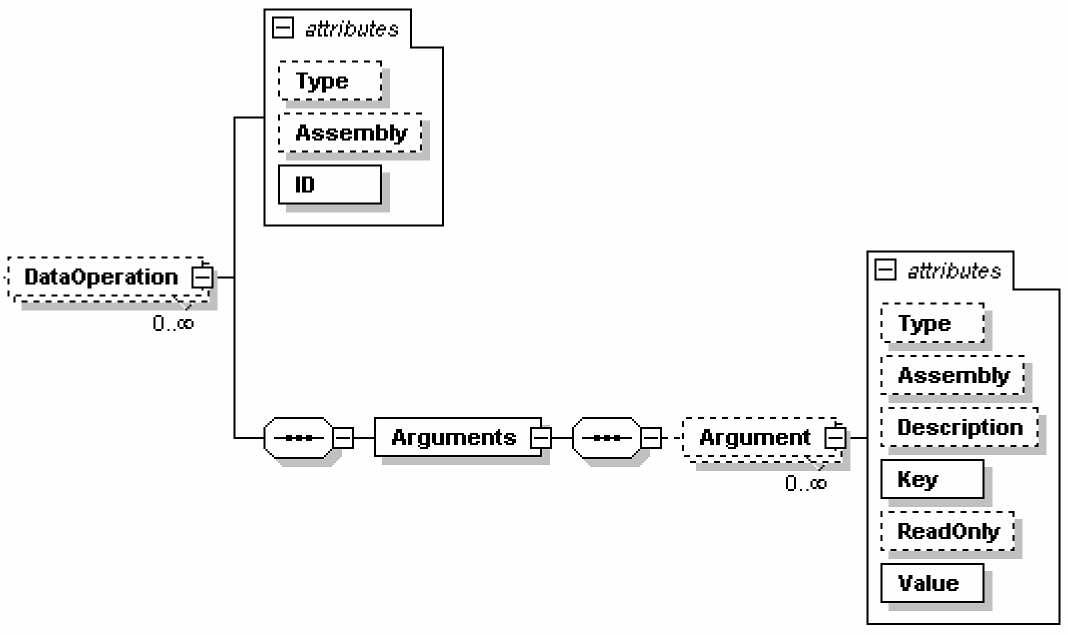


Figure 7-13 Composition.xsd data operation definition

A data operation consists of an identifier and a list of arguments. Except for the type identifying attributes (Type and Assembly) the attributes resemble the properties of the corresponding classes.

7.4 Implementation remarks

7.4.1 C#-implementation

The C# implementation utilizes no specific .NET features. The classes require the System and the System.Collections assembly. The correct implementation of all methods has been tested using dedicated unit tests in combination with the NUnit framework for testing.

The org.OpenMI.Utilities.Buffer package is available as open source under LGPL licence on Source Forge (http://sourceforge.net/projects/openmi)

7.4.2 Java implementation

Not yet available.

Annex I Package Details

Annex I-A org.OpenMI.Utilities.Buffer package

Data:

The content of the *SmartBuffer* are lists of corresponding *times* and *ValueSets*, where times can be *TimeStamps* or *TimeSpans* and the *ValueSets* can be *ScalarSets* or *VectorSets*.

SmartBuffer objects may not contain mixtures of *TimeSpans* and *TimeStamps* and may not contain mixtures of *ScalarSets* and *VectorSets*.

The number of Times (*TimeSpans* or *TimeStamps*) must equal the number of *ValueSets* (*ScalarSets* or *VectorSets*) in the SmartBuffer.

The data structures can be expressed in mathematical terms as follows:

List of *TimeStamps*:

$$\underline{t_b} = \left\{ t_b^0, t_b^1, t_b^2, ..., t_b^{n-1}, t_b^n, t_b^{n+1}, ..., t_b^{N-2}, t_b^{N-1} \right\} \qquad \text{(A1.A.1)}$$

where $\underline{t_b}$ is a list of buffered *TimeStamps*, t_b^n is the nth *TimeStamp* in the list, and *N* is the number of *TimeStamps* in the list of *TimeStamps*.

It is required that:

$$t_b^n < t_b^{n+1} \Big|_{n=0...N-2} \qquad \text{(A1.A.2)}$$

List of *TimeSpans*:

$$\begin{aligned} \underline{T_b} &= \left\{ T_b^0, T_b^1, T_b^2, ..., T_b^{n-1}, T_b^n, T_b^{n+1}, ..., T_b^{N-2}, T_b^{N-1} \right\} \\ &= \left\{ \begin{pmatrix} t_{b,b} \\ t_{b,e} \end{pmatrix}^0, \begin{pmatrix} t_{b,b} \\ t_{b,e} \end{pmatrix}^1, ..., \begin{pmatrix} t_{b,b} \\ t_{b,e} \end{pmatrix}^{n-1}, \begin{pmatrix} t_{b,b} \\ t_{b,e} \end{pmatrix}^n, \begin{pmatrix} t_{b,b} \\ t_{b,e} \end{pmatrix}^{n+1}, ..., \begin{pmatrix} t_{b,b} \\ t_{b,e} \end{pmatrix}^{N-2}, \begin{pmatrix} t_{b,b} \\ t_{b,e} \end{pmatrix}^{N-1} \right\} \end{aligned} \qquad \text{(A1.A.3)}$$

where $\underline{T_b}$ is a list of buffered *TimeSpans*, T_b^n is the nth *TimeSpan*, $t_{b,b}$ is the *TimeStamp* that holds the begin time of the *TimeSpan*, $t_{b,e}$ is the *TimeStamp* that holds the end time of the TimeSpan, and *N* is the number of *TimeSpans* in the list of *TimeSpans*.

It is required that:

$$t_{b,e}^n = t_{b,b}^{n+1} \Big|_{n=0...N-2} \qquad \text{(A1.A.4)}$$

List of *ScalarSets*:

$$\underline{\underline{s_b}} = \begin{Bmatrix} \underline{s_b}^{0} \\ \underline{s_b}^{1} \\ \cdot \\ \underline{s_b}^{n-1} \\ \underline{s_b}^{n} \\ \underline{s_b}^{n+1} \\ \cdot \\ \underline{s_b}^{N-2} \\ \underline{s_b}^{N-1} \end{Bmatrix} = \begin{Bmatrix} \left\{ s_{b,0}^{0}, \right. & s_{b,1}^{0}, & \ldots, & s_{b,i-1}^{0}, & s_{b,i}^{0}, & s_{b,i+1}^{0}, & \ldots, & s_{b,M-2}^{0}, & \left. s_{b,M-1}^{0} \right\} \\ \left\{ s_{b,0}^{1}, \right. & s_{b,1}^{1}, & \ldots, & s_{b,i-1}^{1}, & s_{b,i}^{1}, & s_{b,i+1}^{1}, & \ldots, & s_{b,M-2}^{1}, & \left. s_{b,M-1}^{1} \right\} \\ \cdot & \cdot & \ldots, & \cdot & \cdot & \cdot & \ldots, & \cdot & \cdot \\ \left\{ s_{b,0}^{n-1}, \right. & s_{b,1}^{n-1}, & \ldots, & s_{b,i-1}^{n-1}, & s_{b,i}^{n-1}, & s_{b,i+1}^{n-1}, & \ldots, & s_{b,M-2}^{n-1}, & \left. s_{b,M-1}^{n-1} \right\} \\ \left\{ s_{b,0}^{n}, \right. & s_{b,1}^{n}, & \ldots, & s_{b,i-1}^{n}, & s_{b,i}^{n}, & s_{b,i+1}^{n}, & \ldots, & s_{b,M-2}^{n}, & \left. s_{b,M-1}^{n} \right\} \\ \left\{ s_{b,0}^{n+1}, \right. & s_{b,1}^{n+1}, & \ldots, & s_{b,i-1}^{n+1}, & s_{b,i}^{n+1}, & s_{b,i+1}^{n+1}, & \ldots, & s_{b,M-2}^{n+1}, & \left. s_{b,M-1}^{n+1} \right\} \\ \cdot & \cdot & \ldots, & \cdot & \cdot & \cdot & \ldots, & \cdot & \cdot \\ \left\{ s_{b,0}^{N-2}, \right. & s_{b,1}^{N-2}, & \ldots, & s_{b,i-1}^{N-2}, & s_{b,i}^{N-2}, & s_{b,i+1}^{N-2}, & \ldots, & s_{b,M-2}^{N-2}, & \left. s_{b,M-1}^{N-2} \right\} \\ \left\{ s_{b,0}^{N-1}, \right. & s_{b,1}^{N-1}, & \ldots, & s_{b,i-1}^{N-1}, & s_{b,i}^{N-1}, & s_{b,i+1}^{N-1}, & \ldots, & s_{b,M-2}^{N-1}, & \left. s_{b,M-1}^{N-1} \right\} \end{Bmatrix} \qquad \text{(A1.A.5)}$$

where $\underline{\underline{s_b}}$ is a list of buffered *ScalarSets*, $\underline{s_b}^{n}$ is the nth *ScalarSet* in the buffered list, $s_{b,i}^{n}$ is the ith scalar (double) in the nth *ScalarSet*, *M* is the number of scalars in each *ScalarSet*, and *N* is the number of buffered *ScalarSets*.

<u>List of *VectorSets*:</u>

$$\left|\underline{\underline{v_b}}\right| = \begin{Bmatrix} \underline{\underline{v_b}}^{0} \\ \underline{\underline{v_b}}^{1} \\ \cdot \\ \underline{\underline{v_b}}^{n-1} \\ \underline{\underline{v_b}}^{n} \\ \underline{\underline{v_b}}^{n+1} \\ \cdot \\ \underline{\underline{v_b}}^{N-2} \\ \underline{\underline{v_b}}^{N-1} \end{Bmatrix} = \begin{Bmatrix} \left\{ \underline{v}_{b,0}^{0}, \right. & \underline{v}_{b,1}^{0}, & \ldots, & \underline{v}_{b,i-1}^{0}, & \underline{v}_{b,i}^{0}, & \underline{v}_{b,i+1}^{0}, & \ldots, & \underline{v}_{b,M-2}^{0}, & \left. \underline{v}_{b,M-1}^{0} \right\} \\ \left\{ \underline{v}_{b,0}^{1}, \right. & \underline{v}_{b,1}^{1}, & \ldots, & \underline{v}_{b,i-1}^{1}, & \underline{v}_{b,i}^{1}, & \underline{v}_{b,i+1}^{1}, & \ldots, & \underline{v}_{b,M-2}^{1}, & \left. \underline{v}_{b,M-1}^{1} \right\} \\ \cdot & \cdot & \ldots, & \cdot & \cdot & \cdot & \ldots, & \cdot & \cdot \\ \left\{ \underline{v}_{b,0}^{n-1}, \right. & \underline{v}_{b,1}^{n-1}, & \ldots, & \underline{v}_{b,i-1}^{n-1}, & \underline{v}_{b,i}^{n-1}, & \underline{v}_{b,i+1}^{n-1}, & \ldots, & \underline{v}_{b,M-2}^{n-1}, & \left. \underline{v}_{b,M-1}^{n-1} \right\} \\ \left\{ \underline{v}_{b,0}^{n}, \right. & \underline{v}_{b,1}^{n}, & \ldots, & \underline{v}_{b,i-1}^{n}, & \underline{v}_{b,i}^{n}, & \underline{v}_{b,i+1}^{n}, & \ldots, & \underline{v}_{b,M-2}^{n}, & \left. \underline{v}_{b,M-1}^{n} \right\} \\ \left\{ \underline{v}_{b,0}^{n+1}, \right. & \underline{v}_{b,1}^{n+1}, & \ldots, & \underline{v}_{b,i-1}^{n+1}, & \underline{v}_{b,i}^{n+1}, & \underline{v}_{b,i+1}^{n+1}, & \ldots, & \underline{v}_{b,M-2}^{n+1}, & \left. \underline{v}_{b,M-1}^{n+1} \right\} \\ \cdot & \cdot & \ldots, & \cdot & \cdot & \cdot & \ldots, & \cdot & \cdot \\ \left\{ \underline{v}_{b,0}^{N-2}, \right. & \underline{v}_{b,1}^{N-2}, & \ldots, & \underline{v}_{b,i-1}^{N-2}, & \underline{v}_{b,i}^{N-2}, & \underline{v}_{b,i+1}^{N-2}, & \ldots, & \underline{v}_{b,M-2}^{N-2}, & \left. \underline{v}_{b,M-1}^{N-2} \right\} \\ \left\{ \underline{v}_{b,0}^{N-1}, \right. & \underline{v}_{b,1}^{N-1}, & \ldots, & \underline{v}_{b,i-1}^{N-1}, & \underline{v}_{b,i}^{N-1}, & \underline{v}_{b,i+1}^{N-1}, & \ldots, & \underline{v}_{b,M-2}^{N-1}, & \left. \underline{v}_{b,M-1}^{N-1} \right\} \end{Bmatrix} \qquad \text{(A1.A.6)}$$

where $|\underline{\underline{v_b}}|$is the buffered list of *VectorSets*, $\underline{\underline{v_b}}^{n}$ is the nth *VectorSet* in the buffered list, $\underline{v}_{b,i}^{n}$ is the ith *Vector* in the nth *VectorSet*, M is the number of Vectors in each *VectorSet*, and N is the number of buffered *VectorSets*.

Each Vector is represented by three co-ordinates (doubles):

$$\underline{v}_{b,i}^{n} = \left\{ v_{b,i,1}^{n}, v_{b,i,2}^{n}, v_{b,i,3}^{n} \right\} \qquad \text{(A1.A.7)}$$

where $v_{b,i,1}^{n}, v_{b,i,2}^{n}$, and $v_{b,i,3}^{n}$ are values (doubles) for the x, y, and z directions, respectively.

Methods:

org.OpenMI.Utilities.SmartBuffer.SmartBuffer

Constructor method for the SmartBuffer class.

org.OpenMI.Utilities.SmartBuffer.AddValues(time : ITime, values: IValueSet) : void Public

The AddValues method will add data to the SmartBuffer object.

Parameters:

- time : object that implements the ITime interface and either the ITimeStamp or the ITimeSpan interface. The times in the time list of the SmartBuffer must all implement ITimeStamp or all implement the ITimeSpan. Mixed “interface types” are not allowed.
- values : object that implements the IValueSet interface and either the IScalarSet or the IVectorSet interface. The values in the value list of the SmartBuffer must all implement IScalarSet or all implement the IVectorSet. Mixed “interface types” are not allowed.

Return values:

none

Exceptions:

- time parameter implemenets neither ITimeStamp nor ITimeSpan.
- valueSet parameter implements neither IScalarSet nor IVectorSet.
- AddValues failed to add time and valueSet to the SmartBuffer

org.OpenMI.Utilities.SmartBuffer.GetValues(time: ITime) : IValueSet Public

The returned object that implements IValueSet is calculated by the GetValues( ) method using interpolation, aggregation, or extrapolation in order to make the IValuesSet correspond to the requested time. The time parameter must either implement ITimeStamp or ITimeSpan and thereby also ITime. The returned object will either be of type ScalarSet or VectorSet depending on the interface implemented on the values added to the buffer.

Parameters:

- time : object that implements the ITime interface and either the ITimeStamp or the ITimeSpan interface. The times in the time list of the SmartBuffer must all implement ITimeStamp or all implement the ITimeSpan. Mixed “interface types” are not allowed.

Return value:

ScalarSet or VectorSet found by interpolation, extrapolation or aggregation. I.e. the returned object implements IValueSet.

Exceptions:

- Requested TimeMapping not available.
- GetValues failed to get value from the SmartBuffer.

org.OpenMI.Utilities.SmartBuffer.Clear (timeStamp : TimeStamp) : void Public

Removes corresponding sets of objects that implements ITime and IValuesSets from the SmartBuffer for which the time falls within the time span defined by timeStamp.

Parameters:

- timeStamp: ITimeStamp

Return value:

- none

Exceptions:

- none

org.OpenMI.Utilities.SmartBuffer.CheckBuffer ( ) : void Public

Checks consistency of the stored times and valueSets.

Parameters:

- none

Return value:

- none

Exceptions:

- Different numbers of values and times in buffer.
- Buffer is empty
- Illegal data type for time in buffer.
- Illegal data type for values in buffer.
- BeginTime is larger than or equal to EndTime in timespan.
- EndTime is not equal to StartTime for the following time step.
- Timestamps are not encreasing in buffer.

GetTimeAt (index : int) : ITime Public

Returns the time at the index location.

Parameters:

- index : int

Return value:

The index´th time from the time list is returned. The type depends on the type originally added. The returned object implemets the ITime interface.

Exceptions:

- Negative index not allowed.
- Index exceeds the contents of the SmartBuffer.
- GetTimeAt failed to return time from SmartBuffer.

GetValuesAt (index : int) : IValueSet Public

Returns the values at location index.

Parameters:

- index : int

Return value:

The index´th valueSet from the values list is returned. The type depends on the type originally added. The returned object implemets the IValueSet interface and is, by the way, identical (in regard of values) to the object originally added using the AddValues mehtod.

Exceptions:

- Negative index not allowed.
- Index exceeds the contents of the SmartBuffer.
- GetValuesAt failed to return ValueSet from SmartBuffer.

MapTimeStampsToTimeStamp(tr : ITimeStamp) : IValueSet Private

MapTimeStampsToTimeStamp is a private method that is called from the public GetValues method, if the buffered times are objects that implements ITimeStamp and if the requested time is an object that implements ITimeStamp. Depending on the contents of the SmartBuffer a ScalarSet or a ValuesSet will be returned. Either way the returned object will be an object that implements IValueSet.

Parameters:

- t_r : ITimeStamp

Detailed description:

The buffered objects are objects that either implement IScalarSet or IVectorSet. The algorithms used for objects with IScalarSet interface are shown below. For objects with IVectorSet the same algorithms are used for the three components in each Vector.

The ScalarSet to return ($x_{r,i}$) is calculated as described in equations (A1.A.8), (A1.A.9), (A1.A.10), and (A1.A.11)

if (N = 1)

$$x_{r,i} = s_{b,i}^{0}\Big|_{For\ i=0\ to\ i=M-1} \qquad \text{(A1.A.8)}$$

else if ($t_r < t_b^0$)

$$x_{r,i} = \left(\frac{s_{b,i}^{0} - s_{b,i}^{1}}{t_b^0 - t_b^1}\right)\left(t_r - t_b^0\right)\left(1-\alpha\right) + s_{b,i}^{0}\Big|_{For\ i=0\ to\ i=M-1} \qquad \text{(A1.A.9)}$$

else if ($t_r > t_b^{N-1}$)

$$x_{r,i} = \left(\frac{s_{b,i}^{N-1} - s_{b,i}^{N-2}}{t_b^{N-1} - t_b^{N-2}}\right)\left(t_r - t_b^{N-1}\right)\left(1-\alpha\right) + s_{b,i}^{N-1}\Bigg|_{For\ i=0\ to\ i=M-1} \qquad \text{(A1.A.10)}$$

else if ($t_b^0 \le t_r \le t_b^{N-1}$)

Find n for which $t_b^n < t_r < t_b^{n+1}$ and $x_{r,i}$ is calculated as follows:

$$x_{r,i} = \left(\frac{s_{b,i}^{n+1} - s_{b,i}^{n}}{t_b^{n+1} - t_b^{n}} \right) \left(t_r - t_b^n \right) + s_{b,i}^n \Big|_{For\ i=M-2\ to\ i=0} \qquad \text{(A1.A.11)}$$

where t_r is the TimeStamp passed as input parameter to this method, α is the relaxation parameter (property of the SmartBuffer class). $\alpha \in [0,1]$. The remaining symbols are as described in eq. (A1.A.1) through (A1.A.7).

Exceptions:

- none

MapTimeSpansToTimeSpan (tr : ITimeSpan) : IValueSet Private

MapTimeSpansToTimeSpan is a private method that is called from the public GetValues method, if the buffered times are objects that implements ITimeSpan and if the requested time is an object that implements ITimeSpan. Depending on the contents of the SmartBuffer a ScalarSet or a ValuesSet will be returned. Either way the returned object will be an object that implements IValueSet.

Parameters:

- t_r : ITimeSpan

Detailed description:

The buffered objects are objects that either implement IScalarSet or IVectorSet. The algorithms used for objects with IScalarSet interface are shown below. For objects with IVectorSet the same algorithms are used for the three components in each Vector.

if (N = 1): Only one time step in the buffer

$$x_{r,i} = s_{b,i}^0 \Big|_{for\ i=0\ to\ i=M-1}$$

if (N > 1): Calculate using step 1 through step 3 as described below:

Step 1: Initialise the result *ScalarSet*

$$x_{r,i} = 0 \Big|_{for\ i=0\ to\ i=M-1}$$

Step 2: For each time step in the buffered list of *TimeSpans* determine how big a fraction of the requested *TimeSpan* that overlaps this time step. Then add the corresponding contribution to $x_{r,i}$

$$\left. \begin{array}{ll} if\ \left(\left(t_{r,b} \le t_{b,b}^{n}\right) \wedge \left(t_{r,e} \ge t_{b,e}^{n}\right)\right) & \left\{ x_{r,i} = x_{r,i} + s_{b,i}^{n} \left(\frac{t_{b,e}^{n} - t_{b,b}^{n}}{t_{r,e} - t_{r,b}} \right) \Bigg|_{for\, i=0\, to\, i=M-1} \right\} \\ else\ if\ \left(t_{b,b}^{n} \le t_{r,b} \wedge t_{r,e} \le t_{b,e}^{n} \right) & \left\{ x_{r,i} = x_{r,i} + s_{b,i}^{n} \Big|_{for\, i=0\, to\, i=M-1} \right\} \\ else\ if\ \left(t_{b,b}^{n} < t_{r,b} \wedge t_{r,b} < t_{b,e}^{n} \wedge t_{r,e} > t_{b,e}^{n} \right) & \left\{ x_{r,i} = x_{r,i} + s_{b,i}^{n} \left(\frac{t_{b,e}^{n} - t_{r,b}}{t_{r,e} - t_{r,b}} \right) \Bigg|_{for\, i=0\, to\, i=M-1} \right\} \\ else\ if\ \left(t_{r,b} < t_{b,b}^{n} \wedge t_{r,e} > t_{b,b}^{n} \wedge t_{r,e} < t_{b,e}^{n} \right) & \left\{ x_{r,i} = x_{r,i} + s_{b,i}^{n} \left(\frac{t_{r,e} - t_{b,b}^{n}}{t_{r,e} - t_{r,b}} \right) \Bigg|_{for\, i=0\, to\, i=M-1} \right\} \end{array} \right|_{for\, n=0\, to\, n=N-1}$$

(A1.A.12)

Step 3: For requested *TimeSpans* that partially lays outside the buffered values extrapolated contributions must be added:

$$if\left(\left(t_{r,b} < t_{b,b}^{0}\right) \wedge \left(t_{r,e} > t_{b,b}^{0}\right)\right) \quad \left\{ x_{r,i} = x_{r,i} + \left(\frac{t_{b,b}^{0} - t_{r,b}}{t_{r,e} - t_{r,b}} \right) \left(s_{b,i}^{0} - (1-\alpha) \left(\frac{\left(t_{b,b}^{0} - t_{r,b}\right)\left(s_{b,i}^{1} - s_{b,i}^{0}\right)}{t_{b,e}^{1} - t_{b,e}^{0}} \right) \right) \Bigg|_{for\, i=0\, to\, i=M-1} \right\}$$

(A1.A.13)

$$if\left(\left(t_{r,e} > t_{b,e}^{N-1}\right) \wedge \left(t_{r,b} < t_{b,e}^{N-1}\right)\right) \quad \left\{ x_{r,i} = x_{r,i} + \left(\frac{t_{r,e} - t_{b,e}^{N-1}}{t_{r,e} - t_{r,b}} \right) \left(s_{b,i}^{N-1} + (1-\alpha) \left(\frac{\left(t_{r,e} - t_{b,b}^{N-1}\right)\left(s_{b,i}^{N-1} - s_{b,i}^{N-2}\right)}{t_{b,e}^{N-1} - t_{b,b}^{N-2}} \right) \right) \Bigg|_{for\, i=0\, to\, i=M-1} \right\}$$

(A1.A.14)

Step 4: For requested *TimeSpans* that our fully outside the buffered values

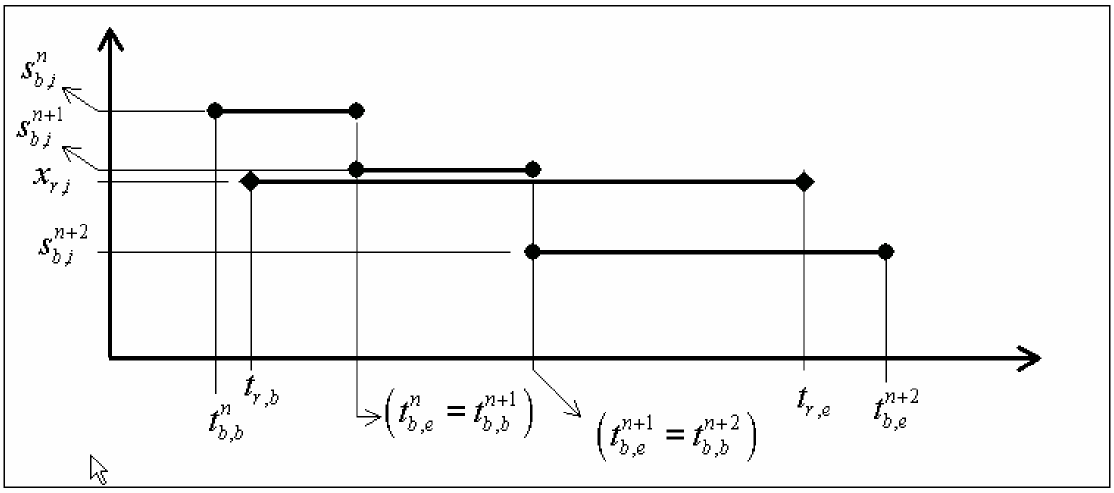
$$if\ \left(t_{r,b} \ge t_{b,e}^{N-1}\right) \quad \left\{ x_{r,i} = s_{b,i}^{N-1} + (1-\alpha) \left(\frac{s_{b,i}^{N-1} - s_{b,i}^{N-2}}{t_{b,e}^{N-1} - t_{b,e}^{N-2}} \right) \left(\frac{\left(t_{r,e} - t_{r,b}\right) - \left(t_{b,e}^{N-1} - t_{b,b}^{N-1}\right)}{2} \right) \Bigg|_{for\, i=0\, to\, i=M-1} \right\}$$

(A1.A.15)

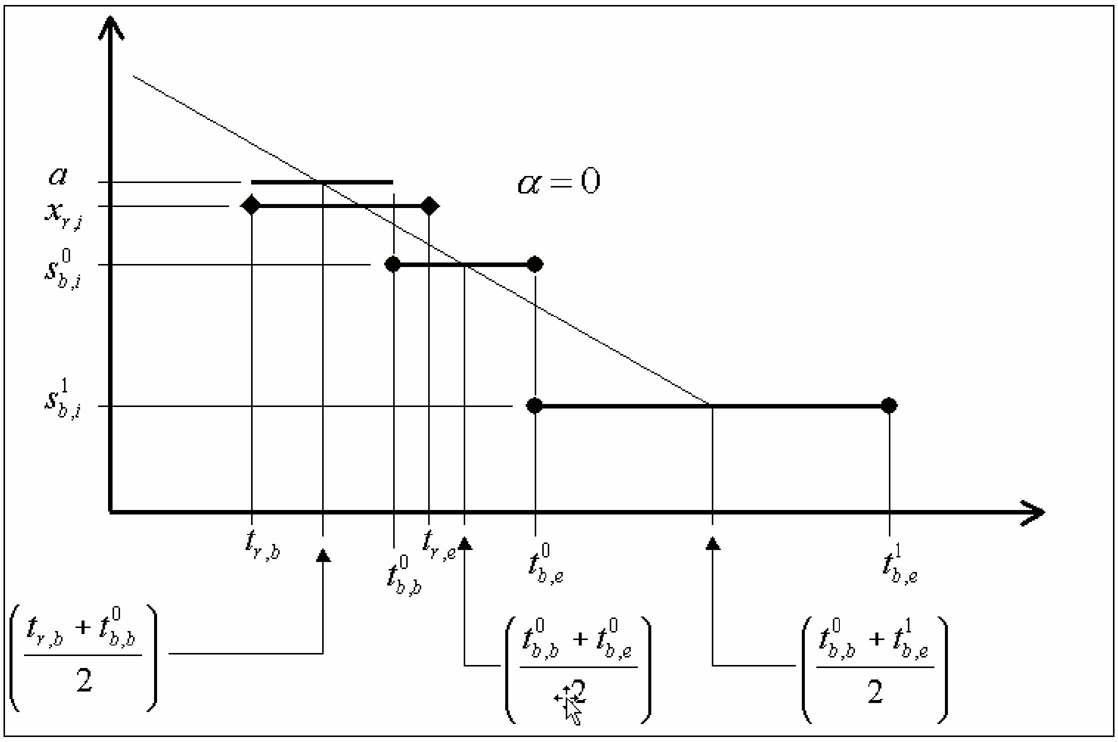
$$if\ \left(t_{r,e} \le t_{b,b}^{0}\right) \quad \left\{ s_{b,i}^{0} - (1-\alpha) \left(\frac{s_{b,i}^{0} - s_{b,i}^{1}}{t_{b,e}^{0} - t_{b,e}^{1}} \right) \left(\frac{\left(t_{r,e} - t_{r,b}\right) - \left(t_{b,e}^{0} - t_{b,b}^{0}\right)}{2} \right) \Bigg|_{for\, i=0\, to\, i=M-1} \right\} \qquad \text{(A1.A.16)}$$

Algorithm explanation:

Below you will find two specific examples of the algorithm.



$$x_{r,i} = s^n_{b,i}\left(\frac{t_{r,b} - t^n_{b,b}}{t_{r,e} - t_{r,b}}\right) + s^{n+1}_{b,i}\left(\frac{t^{n+1}_{b,e} - t^{n+1}_{b,b}}{t_{r,e} - t_{r,b}}\right) + s^{n+2}_{b,i}\left(\frac{t^{n+2}_{b,e} - t^{n+2}_{b,b}}{t_{r,e} - t_{r,b}}\right)$$

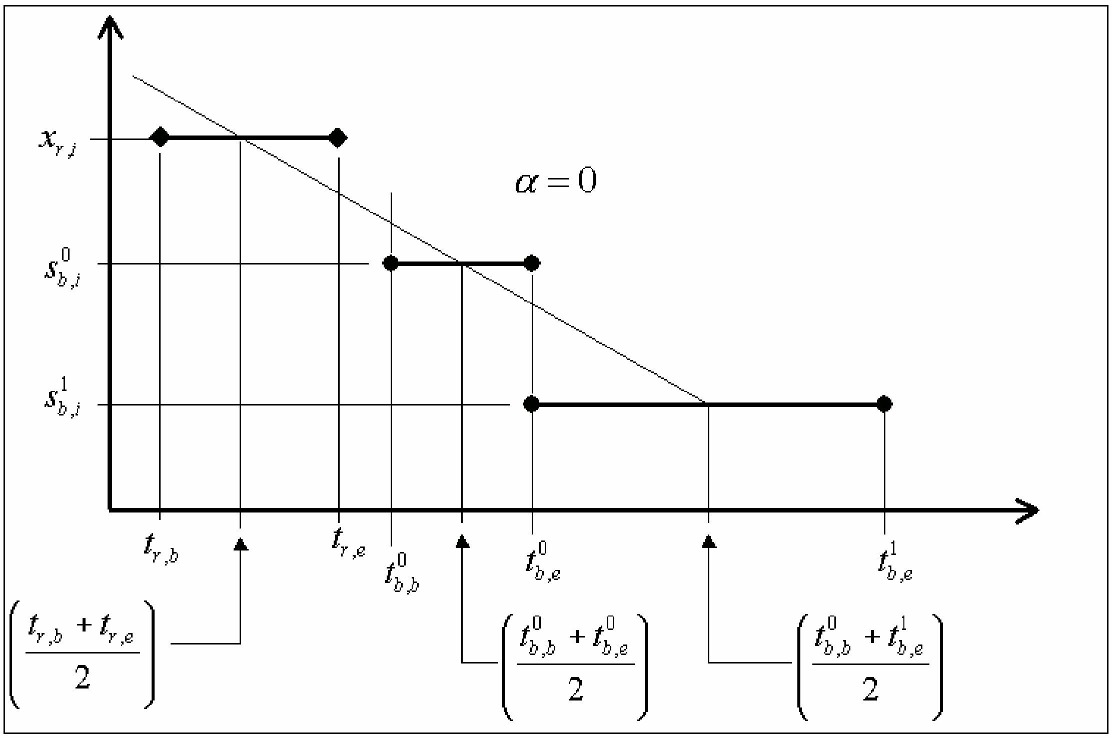

a
$x_{r,i}$
$s^0_{b,i}$
$s^1_{b,i}$
$\alpha = 0$
$t_{r,b}$
$t^0_{b,b}$
$t_{r,e}$
$t^0_{b,e}$
$t^1_{b,e}$
$\left(\frac{t_{r,b} + t^0_{b,b}}{2}\right)$
$\left(\frac{t^0_{b,b} + t^0_{b,e}}{2}\right)$
$\left(\frac{t^0_{b,b} + t^1_{b,e}}{2}\right)$

$$a = s_{b,i}^0 - (1-\alpha)\left(\frac{s_{b,i}^1 - s_{b,i}^0}{\frac{t_{b,b}^0 + t_{b,e}^1}{2} - \frac{t_{b,b}^0 + t_{b,e}^0}{2}}\right)\left(\frac{t_{b,b}^0 + t_{b,e}^0}{2} - \frac{t_{r,b} + t_{b,e}^0}{2}\right)$$

$$\Rightarrow a = s_{b,i}^0 - (1-\alpha)\left(\frac{\left(t_{b,b}^0 - t_{r,b}\right)\left(s_{b,i}^1 - s_{b,i}^0\right)}{t_{b,e}^1 - t_{b,e}^0}\right)$$

$$x_{r,i} = \left(\frac{t_{b,b}^0 - t_{r,b}}{t_{r,e} - t_{r,b}}\right)\left(s_{b,i}^0 - (1-\alpha)\left(\frac{\left(t_{b,b}^0 - t_{r,b}\right)\left(s_{b,i}^1 - s_{b,i}^0\right)}{t_{b,e}^1 - t_{b,e}^0}\right)\right) + s_{b,i}^0\left(\frac{t_{r,e} - t_{b,b}^0}{t_{r,e} - t_{r,b}}\right)$$

where the first term is handled in step 3 and the last term is handled in step 2.


$x_{r,i}$
$\alpha = 0$
$s_{b,i}^0$
$s_{b,i}^1$
$t_{r,b}$
$t_{r,e}$
$t_{b,b}^0$
$t_{b,e}^0$
$t_{b,e}^1$
$\left(\frac{t_{r,b} + t_{r,e}}{2}\right)$
$\left(\frac{t_{b,b}^0 + t_{b,e}^0}{2}\right)$
$\left(\frac{t_{b,b}^0 + t_{b,e}^1}{2}\right)$

$$x_{r,i} = s_{b,i}^0 - (1-\alpha)\left(\frac{s_{b,i}^1 - s_{b,i}^0}{\frac{t_{b,b}^0 + t_{b,e}^1}{2} - \frac{t_{b,b}^0 + t_{b,e}^0}{2}}\right)\left(\frac{t_{b,b}^0 + t_{b,e}^0}{2} - \frac{t_{r,b} + t_{r,e}}{2}\right)$$

$$. \qquad \Rightarrow x_{r,i} = s_{b,i}^{0} - (1-\alpha)\left(\frac{\left(s_{b,i}^{1} - s_{b,i}^{0}\right)\left(t_{b,b}^{0} + t_{b,e}^{0} - t_{r,b} - t_{r,e}\right)}{t_{b,e}^{1} - t_{b,e}^{0}}\right)$$

Exceptions:

- none

MapTimeStampsToTimeSpan (tr : ITimeSpan) : IValuesSet Private

MapTimeStampsToTimeSpan is a private method that is called from the public GetValues method, if the buffered times are objects that implements ITimeStamp and if the requested time is an object that implements ITimeSpan. Depending on the contents of the SmartBuffer a ScalarSet or a ValuesSet will be returned. Either way the returned object will be an object that implements IValueSet.

Parameters:

t_r : ITimeSpan

Detailed description:

The buffered objects are objects that either implement IScalarSet or IVectorSet. The algorithms used for objects with IScalarSet interface are shown below. For objects with IVectorSet the same algorithms are used for the three components in each Vector.

if (N = 1): Only one time step in the buffer

$$x_{r,i} = s_{b,i}^{0}\Big|_{for\, i=0\, to\, i=M-1}$$

if (N > 1): Calculate using step 1 through step 3 as described below:

Step 1: Initialise the result ScalarSet

$$x_{r,i} = 0\Big|_{for\, i=0\, to\, i=M-1}$$

Step 2:

$$
\left.\begin{array}{l}
\text{if } \left(t_{r,b} \le t_b^n \wedge t_{r,e} \ge t_b^{n+1}\right) \\
\qquad \left\{ x_{r,i} = x_{r,i} + \left(\dfrac{s_{b,i}^n + s_{b,i}^{n+1}}{2}\right)\left(\dfrac{t_b^{n+1} - t_b^n}{t_{r,e} - t_{r,b}}\right)\Bigg|_{i=0\ldots M-1} \right\} \\[2ex]
\text{else if } \left(t_b^n \le t_{r,b} \wedge t_{r,e} \le t_b^{n+1}\right) \\
\qquad \left\{ x_{r,i} = x_{r,i} + s_{b,i}^n + \left(\dfrac{s_{b,i}^{n+1} - s_{b,i}^n}{t_b^{n+1} - t_b^n}\right)\left(\dfrac{t_{r,e} + t_{r,b}}{2} - t_b^n\right)\Bigg|_{i=0\ldots M-1} \right\} \\[2ex]
\text{else if } \left(t_b^n < t_{r,b} \wedge t_{r,b} < t_b^{n+1} \wedge t_{r,e} > t_b^{n+1}\right) \\
\qquad \left\{ x_{r,i} = x_{r,i} + s_{b,i}^n - \left(\dfrac{s_{b,i}^{n+1} - s_{b,i}^n}{t_b^{n+1} - t_b^n}\right)\left(\dfrac{t_b^{n+1} - t_{r,b}}{2}\right)\left(\dfrac{t_b^{n+1} - t_{r,b}}{t_{r,e} - t_{r,b}}\right)\Bigg|_{i=0\ldots M-1} \right\} \\[2ex]
\text{else if } \left(t_{r,b} < t_b^n \wedge t_{r,e} > t_b^n \wedge t_{r,e} < t_b^{n+1}\right) \\
\qquad \left\{ x_{r,i} = x_{r,i} + s_{b,i}^n + \left(\dfrac{s_{b,i}^{n+1} - s_{b,i}^n}{t_b^{n+1} - t_b^n}\right)\left(\dfrac{t_{r,e} - t_n^n}{2}\right)\left(\dfrac{t_{r,e} - t_b^n}{t_{r,e} - t_{r,b}}\right)\Bigg|_{i=0\ldots M-1} \right\}
\end{array}\right|_{n=0\ldots N-2} \qquad (5.2.10.3)
$$

Step 3: For requested *TimeSpans* that partially lays outside the buffered values extrapolated contributions must be added:

$$
\mathit{if} \left(t_{r,b} < t_b^0 \wedge t_{r,e} > t_b^0\right)
$$

$$
\left\{ x_{r,i} = x_{r,i} + \left(\frac{t_b^0 - t_{r,b}}{t_{r,e} - t_{r,b}}\right)\left(s_{b,i}^0 - (1-\alpha)\left(\frac{\left(t_b^0 - t_{r,b}\right)\left(s_{b,i}^1 - s_{b,i}^0\right)}{2\left(t_b^1 - t_b^0\right)}\right)\right)\Bigg|_{i=0\ldots M-1} \right\}
$$

$$
\mathit{if} \left(t_{r,e} > t_b^{N-1} \wedge t_{r,b} < t_b^{N-1}\right)
$$

$$
\left\{ x_{r,i} = x_{r,i} + \left(\frac{t_{r,e} - t_b^{N-1}}{t_{r,e} - t_{r,b}}\right)\left(s_{b,i}^{N-1} + (1-\alpha)\left(\frac{\left(t_{r,e} - t_b^{N-1}\right)\left(s_{b,i}^{N-1} - s_{b,i}^{N-2}\right)}{2\left(t_b^{N-1} - t_b^{N-2}\right)}\right)\right)\Bigg|_{\hat{\imath}=0\ldots M-1} \right\}
$$

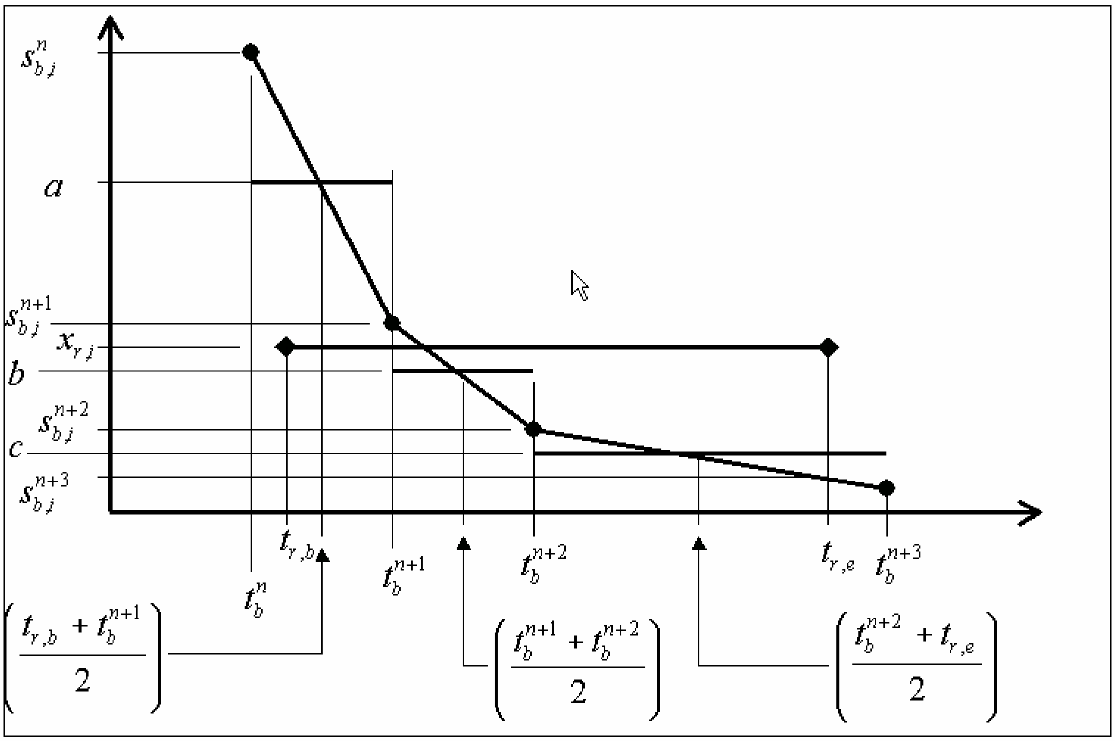
$$
\mathit{if} \left(t_{r,b} \ge t_b^{N-1}\right)
$$

$$
\left\{ x_{r,i} = s_{b,i}^{N-1} + (1-\alpha)\left(\frac{s_{b,i}^{N-1} - s_{b,i}^{N-2}}{t_b^{N-1} - t_b^{N-2}}\right)\left(\frac{t_{r,b} + t_{r,e}}{2} - t_n^{N-1}\right)\Bigg|_{\hat{\imath}=0\ldots M-1} \right\}
$$

$$if\left(t_{r,e} \le t_b^0\right)$$

$$\left\{ x_{r,i} = s_{b,i}^0 - (1-\alpha)\left(\frac{s_{b,i}^1 - s_{b,i}^0}{t_b^1 - t_b^0}\right)\left(t_b^0 - \frac{(t_{r,e}+t_{r,b})}{2}\right)\Bigg|_{\hat{\imath}=0\ldots M-1} \right\}$$

Algorithm explanation:
Below more details of how the algorithms are deducted shown.



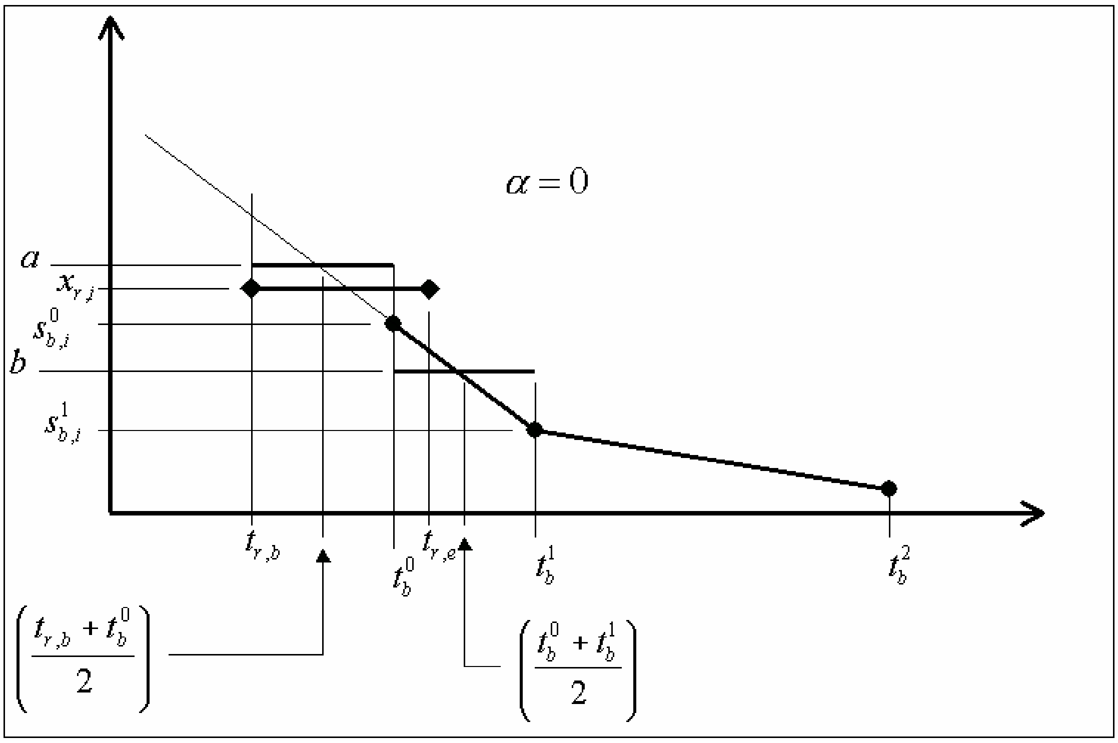
$$a = s_{b,i}^{n+1} - \left(\frac{s_{b,i}^{n+1} - s_{b,i}^n}{t_b^{n+1} - t_b^n}\right)\left(t_b^{n+1} - \frac{t_{r,b} - t_b^{n+1}}{2}\right) = s_{b,i}^{n+1} - \left(\frac{s_{b,i}^{n+1} - s_{b,i}^n}{t_b^{n+1} - t_b^n}\right)\left(\frac{t_b^{n+1} - t_{r,b}}{2}\right)$$

$$b = \frac{s_{b,i}^{n+1} + s_{b,i}^{n+2}}{2}$$

$$c = s_{b,i}^{n+2} - \left(\frac{s_{b,i}^{n+3} - s_{b,i}^{n+2}}{t_b^{n+3} - t_b^{n+2}}\right)\left(\frac{t_b^{n+2} + t_{r,e}}{2} - t_b^{n+2}\right) = s_{b,i}^{n+2} - \left(\frac{s_{b,i}^{n+3} - s_{b,i}^{n+2}}{t_b^{n+3} - t_b^{n+2}}\right)\left(\frac{t_{r,e} - t_b^{n+2}}{2}\right)$$

$$x_{r,i} = \frac{t_b^{n+1} - t_{r,b}}{t_{r,e} - t_{r,b}}a + \frac{t_b^{n+2} - t_b^{n+1}}{t_{r,e} - t_{r,b}}b + \frac{t_{r,e} - t_b^{n+2}}{t_{r,e} - t_{r,b}}c$$

$$
\begin{aligned}
x_{r,i} = &\left(\frac{t_b^{n+1} - t_{r,b}}{t_{r,e} - t_{r,b}}\right)\left(s_{b,i}^{n+1} - \left(\frac{s_{b,i}^{n+1} - s_{b,i}^{n}}{t_b^{n+1} - t_b^{n}}\right)\left(\frac{t_b^{n+1} - t_{r,b}}{2}\right)\right) \\
&+\left(\frac{t_b^{n+2} - t_b^{n+1}}{t_{r,e} - t_{r,b}}\right)\left(\frac{s_{b,i}^{n+1} + s_{b,i}^{n+2}}{2}\right) \\
&+\left(\frac{t_{r,e} - t_b^{n+2}}{t_{r,e} - t_{r,b}}\right)\left(s_{b,i}^{n+2} - \left(\frac{s_{b,i}^{n+3} - s_{b,i}^{n+2}}{t_b^{n+3} - t_b^{n+2}}\right)\left(\frac{t_{r,e} - t_b^{n+2}}{2}\right)\right)
\end{aligned}
$$



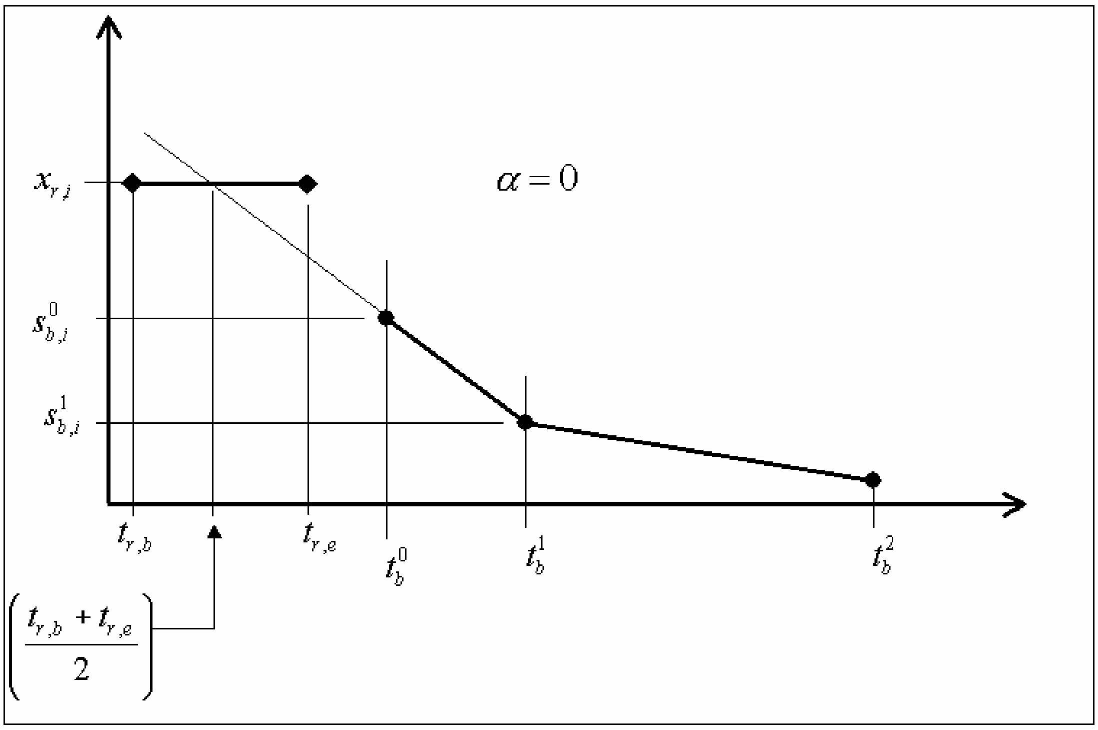
$$
a = s_{b,i}^0 - (1-\alpha)\left(\frac{s_{b,i}^1 - s_{b,i}^0}{t_b^1 - t_b^0}\right)\left(t_b^0 - \frac{t_{r,b} + t_b^0}{2}\right) = s_{b,i}^0 - (1-\alpha)\left(\frac{s_{b,i}^1 - s_{b,i}^0}{t_b^1 - t_b^0}\right)\left(\frac{t_b^0 - t_{r,b}}{2}\right)
$$

$$
b = \frac{s_{b,i}^0 + s_{b,i}^1}{2}
$$

$$
x_{r,i} = \left(\frac{t_b^0 - t_{r,b}}{t_{r,e} - t_{r,b}}\right)a + \left(\frac{t_{r,e} - t_b^0}{t_{r,e} - t_{r,b}}\right)b
$$

$$
\Rightarrow x_{r,i} = \left(\frac{t_b^0 - t_{r,b}}{t_{r,e} - t_{r,b}}\right)\left(s_{b,i}^0 - (1-\alpha)\left(\frac{s_{b,i}^1 - s_{b,i}^0}{t_b^1 - t_b^0}\right)\left(\frac{t_b^0 - t_{r,b}}{2}\right)\right) + \left(\frac{t_{r,e} - t_b^0}{t_{r,e} - t_{r,b}}\right)\left(\frac{s_{b,i}^0 + s_{b,i}^1}{2}\right)
$$

where the firs term is handled in step 3 and the last term is handled in step 2.



$$x_{r,i} = s_{b,i}^0 - (1-\alpha)\left(\frac{s_{b,i}^1 - s_{b,i}^0}{t_b^1 - t_b^0}\right)\left(t_b^0 - \frac{t_{r,b}+t_{r,e}}{2}\right)$$

Exceptions:

- none

MapTimeSpansToTimeStamp (tr : ITimeStamp) : IValuesSet Private

MapTimeSpansToTimeStamp is a private method that is called from the public GetValues method, if the buffered times are objects that implements ITimeSpan and the requested time is an object that implements ITimeStamp. Depending on the contents of the SmartBuffer a ScalarSet or a ValuesSet will be returned. Either way the returned object will be an object that implements IValueSet.

Parameters:

t_r : ITimeStamp

Detailed description:

The buffered objects are objects that either implement IScalarSet or IVectorSet. The algorithms used for objects with IScalarSet interface are shown below. For objects with IVectorSet the same algorithms are used for the three components in each Vector.

The ScalarSet to return ($x_{r,i}$) is calculated as described in equations (A1.A.17), (A1.A.18), (A1.A.19), and (A1.A.20)

if (N = 1)

$$x_{r,i} = s_{b,i}^{0}\Big|_{For\ i=0\ to\ i=M-1} \qquad \text{(A1.A.17)}$$

else if ($t_r \le t_{bb}^{0}$)

$$x_{r,i} = \left(\frac{s_{b,i}^{0} - s_{b,i}^{1}}{t_{bb}^{0} - t_{bb}^{1}}\right)\left(t_r - t_{bb}^{0}\right)\left(1-\alpha\right) + s_{b,i}^{0}\Big|_{For\ i=0\ to\ i=M-1} \qquad \text{(A1.A.18)}$$

else if ($t_r \le t_{b}^{N-1}$)

$$x_{r,i} = \left(\frac{s_{b,i}^{N-1} - s_{b,i}^{N-2}}{t_{b,e}^{N-1} - t_{b,e}^{N-2}}\right)\left(t_r - t_{b,e}^{N-1}\right)\left(1-\alpha\right) + s_{b,i}^{N-1}\Bigg|_{For\ i=0\ to\ i=M-1} \qquad \text{(A1.A.19)}$$

else

$$\text{if}\left(t_{b,b}^{n} \le t_r < t_{b,e}^{n}\right) \quad \left\{x_{r,i} = s_{b,i}^{n}\Big|_{i=0\ldots M-1}\right\}\Bigg|_{n=N-1\ldots0} \qquad \text{(A1.A.20)}$$

where t_r is an object with ITimeStamp passed as input parameter to this method, α is the relaxation parameter (property of the SmartBuffer class). The remaining symbols are as described in eq. (A1.A.1) through (A1.A.7).

Exceptions:

- none

Properties:

[property] RelaxationFactor : double Public

Sets or gets the relaxation factor used for time weighting in extrapolation algorithms

Get: **Return value:** double relaxationFactor

Set: **Parameter**: double relaxationFactor

[property] TimesCount : int Public

Gets the number of records in the buffer.

Get: **Return value:** int numberOfRecords

[property] ValuesCount : int Public

Gets the values in each ValueSet in the buffer.

Get: **Return value:** int numberOfValuesInEachValueSet

[property] DoExtendedDataVerification : bool Public

Gets or sets the _doExtendedDataVerification variable. When _doExtendedDataVerification is true the private Buffer.CheckBuffer( ) method will be invoked at following locations.

Get: **Return value:** int numberOfValuesInEachValueSet

Annex I-B org.OpenMI.Utilities.Spatial package

Short description:

The ElementMapper maps a ValueSet (ScalarSet or VectorSet) associated to one ElementSet (FromElements) onto a ValueSet that is associated to another ElementSet (ToElements).

$$\underline{c} = \underline{\underline{A}}\,\underline{b}$$

where

$$\underline{b} = \lfloor b_0, b_1, ...b_j, ...b_{m-1} \rfloor \text{ for ScalarSet mapping}$$

$$\underline{b} = \begin{bmatrix} b_{0,0}, b_{0,1}, ...b_{0,j}, ...b_{0,m-1} \\ b_{1,0}, b_{1,1}, ...b_{1,j}, ...b_{1,m-1} \end{bmatrix} \text{for VectorSet mapping}$$

$$\underline{\underline{A}} = \begin{bmatrix} a_{0,0} & a_{0,1} & . & . & . & . & a_{0,m-1} \\ a_{1,0} & . & . & . & . & . & . \\ . & . & a_{i,j} & . & . & . & . \\ . & . & . & . & . & . & . \\ a_{n-1,0} & . & . & . & . & . & a_{n-1,m-1} \end{bmatrix}$$

$$\underline{c} = \left[c_0, c_1, ...c_i, ...c_{n-1}\right] \text{ for ScalarSet mapping}$$

$$\underline{c} = \begin{bmatrix} c_{0,0}, c_{0,1}, ...c_{0,i}, ...c_{0,m-1} \\ c_{1,0}, c_{1,1}, ...c_{1,i}, ...c_{1,m-1} \end{bmatrix} \text{for VectorSet mapping}$$

$$n = \text{number of acceptor elements}$$

$$m = \text{number of provider elements}$$

ScalarSets maps to ScalarSets, VectorSets maps to VectorSets.

The matrix element, $a_{i,j,}$ contains the mapping of the j´th element of the InputElementSet onto the i´th element of the OutputElementSet. The mapping elements depend on the mapping method and the element geometry.

For scalar mapping the vector element, b_j, contains the j´th scalar of the ValueSet associated with the InputElementSet. For vector mapping $\underline{b}_j = (b_{0,j}, b_{1,j})$ contains the j´th vector of the ValueSet associated with the InputElementSet.

Similarly, the vector element, c_i, contains the i´th scalar of the ValueSet associated with the OutputElementSet for scalar mapping and for vector mapping $\underline{c}_i = (c_{0,i}, c_{1,i})$ contains the i´th vector of the ValueSet associated with the OutputElementSet.

The mapping is a two step procedure. First step (Initialise) sets up the mapping matrix, $\underline{\underline{A}}$, and is executed only ones. This step depends on the geometry and method choice only. Second step is the actual mapping where ElementSets with their current ValueSet are mapped onto one another by multiplication with the mapping matrix, $\underline{\underline{A}}$.

Data:

MappingMatrix: Two-dimensional double Array containing the coefficients of $\underline{\underline{A}}$.

NumberOfRows:	Integer holding the number of rows in A., which is also the number of Elements in the OutputElementSet.
NumberOfColumns:	Integer holding the number of columns in A., which is also the number of Elements in the InputElementSet.
IsInitialised:	Boolean flag telling weather MappingMatrix, NumberOfRows, NumberOfColumns and in principle also IsInitialised are initialised.

org.OpenMI.Utilities.ElementMapper:

Methods:

Initialise(string methodDescriptor, IElementSet fromElements, IElementSet toElements): void Public

The Initialize method will create a conversion matrix with the same number of rows as the number of elements in the ElementSet associated to the accepting component and the same number of columns as the number of elements in the ElementSet associated to the providing component.

Parameters:

- methodDescriptor: String specifying the mapping method to use. Could be Nearest, Inverse, Weighted Mean or one of the other method descriptors returned by GetAvaliableMethods.
- fromElements: Object that through its IElementSet interface describes the geometry that is to mapped from.
- toElements: Object that through its IElementSet interface describes the geometry that is to mapped to.

Return values:

- none

Exceptions:

- none

MapValues(IValueSet InputValues) : IValueSet Public

Maps the values in the inputValues onto a ValueSet using the mapping matrix.

Parameters:

- inputValues: Object that has the IValueSet interface and that is to be mapped using the mapping matrix.

Return values:

- Object with the IValueSet interface. InputValues implements IScalarSet a ScalarSet is returned and similarly a IVectorSet results in a VectorSet. Either way the returned object implements IValueSet.

Exceptions:

- ElementMapper objects needs to be initialised before the MapValue method can be used.
- Dimension mismatch between inputValues and mapping matrix.
- Invalid datatype used for inputValues parameter. MapValues failed.

UpdateMappingMatrix(string methodDescriptor, IElementSet fromElements, IElementSet toElements) : void Public

UpdateMappingMatrix is where the mapping matrix is processed. UpdateMappingMatrix is called from the initialize but may also be called directly in case the mapping matrix are to be updated during simulation.

Parameters:

- methodDescriptor : String specifying the mapping method to use. Could be Nearest, Inverse, Weighted Mean or one of the other method descriptors returned by GetAvaliableMethods.
- fromElements: Object that through its IElementSet interface describes the geometry that is to mapped from.
- toElements: Object that through its IElementSet interface describes the geometry that is to mapped to.

Return values:

- none

Detailed description:

The UpdateMappingMatrix method includes an algorithm choice based on fromElementType, toElementType and methodDescriptor. The available algorithms are described in the following.

XYPoint→ XYPoint:

Nearest:

The nearest method assigns the value of the point closest to the point in question. In case of more points being "closest", the value assigned will be the mean of the closest values.

In order to populate the mapping matrix, $\underline{\underline{A}}$, a matrix of point to point distances, $l_{i,j}$, is created. Next the minimum distance for each point each point, L_i, is found by inspection. Finally elements of the mapping matrix, $a_{i,j}$, are assigned with the point present relative to total number of points in polygon.

$$l_{i,j} = DistancePointToPoint(b_j, c_i)$$

$$L_i = min(l_{i,j})\Big|_{j\in[0;m-1]}$$

$$l_{i,j} = \begin{cases} 1\, if\ l_{i,j} = L_i \\ 0\, if\ l_{i,j} \neq L \end{cases}$$

$$L_i = \sum_{j=0}^{m-1} l_{i,j}$$

$$a_{i,j} = \frac{l_{i,j}}{L_i}$$

where b_j is the j´th XYPoint in the InputElementSet and c_i is the i´th XYPoint of the OutputElementSet.

Inverse:

In order to populate the mapping matrix, $\underline{\underline{A}}$, a matrix of point to point inverse distances, $l_{i,j}$, is created. Next the total distance from point i to the all points of the InputElelmentSet , L_i, is found by summation. Finally elements of the mapping matrix, $a_{i,j}$, are assigned with the inverse length relative to the summed inverse lengths.

$$l_{i,j} = \frac{1}{DistancePointToPoint(b_j, c_i)}$$

$$L_i = \sum_{j=0}^{m-1} l_{i,j}$$

$$a_{i,j} = \frac{l_{i,j}}{L_i}$$

where b_j is the j´th XYPoint in the InputElementSet and c_i is the i´th XYPoint of the OutputElementSet.

<u>XYPoint→XYLine:</u>

<u>Nearest:</u>

The nearest method assigns the value of the point closest to the XYline in question. In case of more points being “closest”, the value assigned will be the mean of the value of the closest points.

In order to populate the mapping matrix, $\underline{\underline{A}}$, a matrix of point to polyline distances, $l_{i,j}$, is created. Next the minimum distance for each line, L_i, is found by inspection. After this the matirx l may be overwritten with ones for distances being the minimum distance for each line.
Finally each row of the matrix is normalized with the row sum to deal with several points being ”closest”.

$$l_{i,j} = DistancePointToLine(b_j, c_i)$$

$$L_i = min(l_{i,j})\Big|_{j \in [0;m-1]}$$

$$l_{i,j} = \begin{cases} 1\, if\ l_{i,j} = L_i \\ 0\, if\ l_{i,j} \neq L \end{cases}$$

$$L_i = \sum_{j=0}^{m-1} l_{i,j}$$

$$a_{i,j} = \frac{l_{i,j}}{L_i}$$

where b_j is the j´th XYPoint in the InputElementSet and c_i is the i´th XYPolyLine of the OutputElementSet.

<u>Inverse:</u>

In order to populate the mapping matrix, $\underline{\underline{A}}$, a matrix of point to line reciprocal distances, $l_{i,j}$, is created. Next the total distance from line i to the all points of the InputElelmentSet , L_i, is found by summation. Finally elements of the mapping matrix, $a_{i,j}$, are assigned with the inverse length relative to the summed inverse lengths.

$$l_{i,j} = \frac{1}{DistancePointToLine(b_j, c_i)}$$

$$L_i = \sum_{j=0}^{m-1} l_{i,j}$$

$$a_{i,j} = \frac{l_{i,j}}{L_i}$$

where b_j is the j´th XYPoint in the InputElementSet and c_i is the i´th XYPolyLine of the OutputElementSet.

XYPoint → XYPolygon:

Mean:

The polygon is assigned a value found by averaging the point contributions. In this way a polygon that has no points inside is assigned value 0, a polygon enclosing one point only is assigned the value of the point and finally a polygon enclosing more than one point is assigned with the mean of the point values.

In order to populate the mapping matrix, $\underline{\underline{A}}$, a matrix of point enclosed, $l_{i,j}$, is created. $l_{i,j}$ is assigned value one if the point is enclosed and zero otherwise. Next the total number of points inside the i´th XYPolygon, L_i, is found by summation. Finally elements of the mapping matrix, $a_{i,j}$, are assigned with the point present relative to total number of points in polygon.

$$l_{i,j} = IsPointInside(b_j, c_i)$$

$$L_i = \sum_{j=0}^{m-1} l_{i,j}$$

$$a_{i,j} = \frac{l_{i,j}}{L_i}$$

where b_j is the j´th XYPoint in the InputElementSet and c_i is the i´th XYPolygon of the OutputElementSet.

Sum:

The polygon is assigned a value found by summing the point contributions. In this way a polygon that has no points inside is assigned value 0, a polygon enclosing one point only is assigned the value of the point and finally a polygon enclosing more than one point is assigned with the sum of the point values.

In order to populate the mapping matrix, $\underline{\underline{A}}$, a matrix of point enclosed, $l_{i,j}$, is created. $l_{i,j}$ is assigned value one if the point is enclosed and zero otherwise.

$$l_{i,j} = IsPointInside(b_j, c_i)$$

where b_j is the j´th XYPoint in the InputElementSet and c_i is the i´th XYPolygon of the OutputElementSet.

XYPolyLine→XYPoint:

Nearest:

The nearest method assigns the value of the XYPolyLine passing closest to the XYpoint in question. In case of more lines being “closest”, the value assigned will be the mean of the value of the closest lines.

In order to populate the mapping matrix, $\underline{\underline{A}}$, a matrix of polyline to point distances, $l_{i,j}$, is created. Next the minimum distance for each line, L_i, is found by inspection. After this the matirx, l, is overwritten with ones for distances being the minimum distance for each point.
Finally each row of the matrix is normalized with the row sum to deal with several lines being ”closest”.

$$l_{i,j} = DistancePointToLine(c_i, b_j)$$

$$L_i = min(l_{i,j})\Big|_{j\in[0;m-1]}$$

$$l_{i,j} = \begin{cases} 1\, if\ l_{i,j} = L_i \\ 0\, if\ l_{i,j} \neq L \end{cases}$$

$$L_i = \sum_{j=0}^{m-1} l_{i,j}$$

$$a_{i,j} = \frac{l_{i,j}}{L_i}$$

where b_j is the j´th XYPolyLine in the InputElementSet and c_i is the i´th XYPoint of the OutputElementSet.

XYPolyLine→XYPolyLine:

Methods for line line mapping are left out since no meaningful uses have been recognised.

XYPolyLine → XYPolygon:

Weigthed Mean:

The polygon is assigned a value found by length weighting of the line contributions. In this way a polygon that is not intersected by any line is assigned value 0, a polygon intersected with one line only is assigned the value of the line and finally a polygon intersected by more than one value is assigned a length weighted mean of the line values.

The mapping matrix, $\underline{\underline{A}}$, is populated by initially evaluating the length, $l_{i,j}$, of the j´th Polyline inside the i´th XYPolygon, $l_{i,j}$. Next the total line length inside the i´th XYPolygon, L_i, is found by summation. Finally elements of the mapping matrix, $a_{i,j}$, the line length in the polygon relative to the total line length in the polygon are calculated.

$$l_{i,j} = LengthOfPolylineInsidePolygon(b_j, c_i)$$

$$L_i = \sum_{j=0}^{m-1} l_{i,j}$$

$$a_{i,j} = \frac{l_{i,j}}{L_i}$$

where b_j is the j´th XYPolyline in the InputElementSet and c_i is the i´th XYPolygon of the OutputElementSet.

Weighted Sum:

The polygon is assigned a value found by length weighting of the line contributions. In this way a polygon that is not intersected by any line is assigned value 0, a polygon intersected with one line only is assigned the value times the length in polygon to total length ratio. A polygon intersected by more than one value is assigned the sum of values times their length in polygon to total length ratio.

The mapping matrix, $\underline{\underline{A}}$, is populated by initially evaluating the length, $l_{i,j}$, of the j´th Polyline inside the i´th XYPolygon, $l_{i,j}$ and dividing this with the length of j´th polyline.

$$l_{i,j} = LengthOfPolylineInsidePolygon(b_j, c_i)$$

$$a_{i,j} = \frac{l_{i,j}}{Length(b_j)}$$

where b_j is the j´th XYPolyline in the InputElementSet and c_i is the i´th XYPolygon of the OutputElementSet.

XYPolygon → XYPoint:

Value:

The polygon to point method is simple. Points included in a polygon are assigned the value of the polygon in which they are included

Hence the mapping matrix is populated as:

$$a_{i,j} = IsPoint\,Inside(b_j, c_i)$$

where b_j is the j´th XYPolygon in the InputElementSet and c_i is the i´th XYPoint of the OutputElementSet.

Points on edges shared between more polygons are given a mean of the polygon values.

XYPolygon → XYPolyLine:

Weighted Mean:

The polyline is assigned a value found by length weighting of the polygon contributions. In this way a line that is not in a polygon is assigned value 0, a line contained completely inside a polygon is assigned the value of the polygon and a line passing more polygons is assigned a length weighted mean of the polygon values.

The mapping matrix, $\underline{\underline{A}}$, is populated by initially evaluating the length of the i´th XYPolyline inside the j´th XYPolygon relative to the total length of the i´th XYPolyline.

$$l_{i,j} = \frac{LengthOfPolylineInsidePolygon(b_j, c_i)}{LengthOfPolyline(c_i)}$$

$$L_i = \sum_{j=0}^{m-1} l_{i,j}$$

$$a_{i,j} = \frac{l_{i,j}}{L_i}$$

where b_j is the j´th XYPolygon in the InputElementSet and c_i is the i´th XYPolyline of the OutputElementSet.

Weighted Sum:

The polyline is assigned a value found by length weighting of the polygon contributions. In this way a line that is not in a polygon is assigned value 0, a line contained completely inside a polygon is assigned the value of the polygon and a line passing more polygons is assigned a length weighted mean of the polygon values.

The mapping matrix, $\underline{\underline{A}}$, is populated by initially evaluating the length of the i´th XYPolyline inside the j´th XYPolygon relative to the total length of the i´th XYPolyline.

$$a_{i,j} = \frac{LengthOfPolylineInsidePolygon(b_j, c_i)}{LengthOfPolyline(c_i)}$$

where b_j is the j´th XYPolygon in the InputElementSet and c_i is the i´th XYPolyline of the OutputElementSet.

XYPolygon $\rightarrow$ XYPolygon:

Weigthed Mean:

Polygons are assigned values found by area weighting of polygon contributions. A polygon not at all covered by any polygon is assigned the delete value. A polygon fully or partly covered by one polygon only is assigned the value of that polygon and, finally, a polygon fully or partly covered by more than one polygon is assigned a area weighted mean of the polygon values.

The mapping matrix, $\underline{\underline{A}}$, is populated by initially evaluating the area, $l_{i,j}$, of the j´th polygon inside the i´th polygon. Next the total area covered inside the i´th XYPolygon, L_i, is found by summation. Finally elements of the mapping matrix, $a_{i,j}$, is assigned the ratio between the single area and the summed area.

$$l_{i,j} = AreaOfPolygonInsidePolygon(b_j, c_i)$$

$$L_i = \sum_{j=0}^{m-1} l_{i,j}$$

$$a_{i,j} = \frac{l_{i,j}}{L_i}$$

where b_j is the j´th XYPolygon in the InputElementSet and c_i is the i´th XYPolygon of the OutputElementSet.

Weighted Sum:

Where the Weighted Mean method is well suited for “state variables”, e.g. water level, the “Weighted Sum” method is more suitable for distributed quantities such as rain pr area. For distributed quantities the default method will tend to over estimate, whereas the distributed is more of a mass conserving method.

Polygons are assigned values found by area weighting of polygon contributions. A polygon not at all covered by any polygon is assigned the delete value. A polygon fully or partly covered by one polygon only is assigned the value of that polygon weighted with covered are over full area and, finally, a polygon fully or partly covered by more than one polygon is assigned a sum of area weighted contributions.

$$a_{i,j} = \frac{AreaOfPolygonInsidePolygon(b_j, c_i)}{Area(c_i)}$$

where b_j is the j´th XYPolygon in the InputElementSet and c_i is the i´th XYPolygon of the OutputElementSet.´

Exceptions:

- methodDescription unknown for point to point mapping
- Point to point mapping failed
- modDescription unknown for point to polyline mappingeth
- Point to polyline mapping failed
- methodDescription unknown for point to polygon mapping
- Point to polygon mapping failed
- methodDescription unknown for polyline to point mapping
- Polyline to point mapping failed
- methodDescription unknown for polygon to point mapping
- Polyline to polygon mapping failed
- methodDescription unknown for polygon to point mapping
- Polygon to point mapping failed
- methodDescription unknown for polygon to polyline mapping
- Polygon to polyline mapping failed
- methodDescription unknown for polygon to polygon mapping
- Polygon to polygon mapping failed
- Mapping of specified ElementTypes not included in ElementMapper
- UpdateMappingMatrix failed to update mapping matrix

GetAvailableMethods(fromElementType: ElementType, toElementType: ElementType): ArrayList Public

The method returns a list of method strings describing the methods available for the given combination of from- and to- ElementTypes. The method is typically to be used to fill the ExchangeModel.OutputExchangeItem´s Dataoperation list.

Parameters:

- fromElementType: ElementType of the elements in the ElementSet that is to mapped from.
- toElementType: ElementType of the elements in the ElementSet that is to mapped to.

Return values:

- List of strings describing the available mapping methods.

Exceptions:

- none

GetValueFromMappingMatrix(row: int, column: int) : double Public

Retrieves element (row, column) from the mapping matrix.

Parameters:

- row: row index. 0-based.
- column: column index. 0-based

Return values:

- Value of element (row, column).

Exceptions:

- GetValueFromMappingMatrix failed.

SetValueInMappingMatrix(Row: int, Column: int) : void Public

Sets element (row, column) in the mapping matrix.

Parameters:

- row: row index. 0-based.
- column: column index. 0-based.

Return values:

- None

Exceptions:

- SetValueInMappingMatrix failed.

org.OpenMI.Utilities.XYGeometryTools:

All methods are static methods.

Methods:

CalculateInterSectionPoint (XYLine: L_1, XYLine L_2) : XYpoint Public

Returns the intersection point for the two lines L_1 and L_2. Actual intersection must be ensured prior to use of this method by use of DoLineSegmentsIntersect. The intersection point is calculated as:

$$(x_1, x_2) = (L_1.x_1, L_1.x_2) \qquad (x_3, x_4) = (L_2.x_1, L_2.x_2)$$
$$(y_1, y_2) = (L_1.y_1, L_1.y_2) \qquad (y_3, y_4) = (L_2.y_1, L_2.y_2)$$

$$x = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & x_1 - x_2 \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & x_3 - x_4 \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 \\ x_3 - x_4 & y_3 - y_4 \end{vmatrix}} \quad and \quad y = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & y_1 - y_2 \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & y_3 - y_4 \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & y_1 - y_2 \\ x_3 - x_4 & y_3 - y_4 \end{vmatrix}}$$

Since it is, beforehand, ensured that the lines cross there is no risk of zero-division.

Parameters:

- L_1: XYLine to test.
- L_2: XYLine to test.

Return values:

- Intersection point as a XYPoint.

Exceptions:

- Attempt to calculate intersection point between non-intersecting lines. CalculateIntersectionPoint failed.

CalculateLengthOfPolylineInsidePolygon (XYPolyline: PL, XYPolygon PG) : double Public

The method calculates the length of the polyline, PL, inside the polygon PG. The method loops over the lines included in PL and adds up the length of each line in the polygon PG. The length of each line is calculated using the private method, CalculateLengthOfLineInsidePolygon. Line pieces that are shared between the polygon and the line contributes with half their length.

Parameters:

- PL: the polyline.
- PG: polygon.

Return values:

- Length of polyline, PL, inside polygon, PG.

Exceptions:

- None.

CalculateLengthOfLineInsidePolygon (XYLine: L, XYPolygon PG) : double Private

The method calculates the length a line has inside a polygon. Lines on edges only counts half.

The algorithm works with a list of XYLines, LL, which initially includes the line L only. The XYPolygon, PG has n lines. Every line of the polygon is tested for intersection with lines from LL. If intersection is found the intersected line is divided at the intersection point and the two new lines replaces the intersected line in the line collection, LL.

```
$n = number\ of\ vertices\ in\ polygon$
$m = 1$
$i = 0$
$j = 0$
$while\ (\ j < m\ )\ and\ (\ not\ \ found\ )$
    $while\ (\ i \le n\ )\ and\ (\ not\ \ found\ )$
        $if(DoLineSegmentsInter\ sec\ t(LL_j,\ PG.L_i\ ))$
            $found = true$
            $P = CalculateInter\ sec\ tionPo\ int\ (LL_j,\ PG.L_i\ )$
            $LL_j\ = (LL_j.p_1,\ P)$
            $LL_{j+1} = (P,\ LL_j.p_2\ )$
            $increase(\ m\ )$
        $else$
            $increase(\ i\ )$
    $end$
    $increase(\ j\ )$
$end$
```

The above creates a line list containing the original line chopped into pieces that are either completely inside or completely outside the polygon. The IsPointInPolygon method for the midpoint of every line in the line list decides which line segments are to be included in the length summation.

Parameters:

- L: the line.
- PG: polygon.

Return values:

- Length of line, L, inside polygon, PG.

Exceptions:

- None.

CalculatePolylineToPointDistance (XYPolyline: polyline, XYLine line2) : double Public

Finds the shortest distance between any line segment of the polyline and the point.

Parameters:

- polyline: the polyline
- line2: XYLine

Return values:

- Length between polyline and point.

Exceptions:

- None

CalculateLineToPointDistance (XYLine: L, XYPoint P) : double Private

Calculates the distance from a line, L, to a point, P, in the plane. The algorithm decides weather the point lies besides the line segment in which case the distance is the length along a line perpendicular to the line. Alternatively the distance is the smallest of the distances to either endpoint.

Algorithm:

$$a = \sqrt{(L.x_2 - P.x)^2 + (L.y_2 - P.y)^2}$$

$$b = \sqrt{(L.x_2 - L.x_1)^2 + (L.y_2 - L.y_1)^2}$$

$$c = \sqrt{(L.x_1 - P.x)^2 + (L.y_1 - P.y)^2}$$

$$\alpha = \left| a\cos\left(\frac{b^2 + c^2 - a^2}{2bc} \right) \right|$$

$$\beta = \left| a\cos\left(\frac{a^2 + b^2 - c^2}{2ab} \right) \right|$$

$$Dis\tan ce = \begin{cases} \dfrac{\left|(L.x_2 - L.x_1)(L.y_1 - P.y) - (L.x_1 - P.x)(L.y_2 - L.y_1)\right|}{\sqrt{(L.x_2 - L.x_1)^2 + (L.y_2 - L.y_1)^2}} & for\ max(\alpha, \beta) < \dfrac{\pi}{2} \\ min(\sqrt{(L.x_2 - L.x_1)^2 + (L.y_2 - L.y_1)^2}, \sqrt{(L.x_2 - L.x_1)^2 + (L.y_2 - L.y_1)^2}\ ) & otherwise \end{cases}$$

Parameters:

- L: XYLine.
- P: XYPoint.

Return values:

- Shortest distance between any point in the line and the point in question.

Exceptions:

- None.

CalculatePointToPointDistance (XYPoint: p1, XYPoint p2) : double Public

Calculates the distance between to points.

Parameters:

- P1: Point
- P2: Point

Return values:

- Length between points.

Exceptions:

- None

CalculateSharedArea (XYPolygon: polygonA, XYPolygon polygonB) : double Public

Calculates the shared area between two arbitrarily shaped polygons in the plane. Both polygons are triangulated and the shared area is calculated as the sum of the shared areas between the triangles.

Parameters:

- polygonA: XYPolygon

- polygonB: XYPolygon

Return values:

- The shared area between the two polygons.

Exceptions:

- None.

CalculateSharedLength (XYLine: lineA, XYLine lineB) : double Private

Calculates the length that two line overlap.

Parameters:

- lineA: XYLine
- lineB: XYLine

Return values:

- Length of the shared line segment.

Exceptions:

- None

DoLineSegmentsIntersect (XYLine: L_1, XYLine L_2) : bool Private

Determines whether two lines cross. The return value will be true if lines literally cross whereas two points just sharing a point will result in a false. The algorithm is:

$$(x_1, x_2) = (L_1.x_1, L_1.x_2) \qquad (x_3, x_4) = (L_2.x_1, L_2.x_2)$$

$$(y_1, y_2) = (L_1.y_1, L_1.y_2) \qquad (y_3, y_4) = (L_2.y_1, L_2.y_2)$$

$$Det_{123} = \begin{vmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{vmatrix} \qquad Det_{124} = \begin{vmatrix} x_2 - x_1 & x_4 - x_1 \\ y_2 - y_1 & y_4 - y_1 \end{vmatrix}$$

$$Det_{341} = \begin{vmatrix} x_4 - x_3 & x_1 - x_3 \\ y_4 - y_3 & y_1 - y_3 \end{vmatrix} \qquad Det_{342} = \begin{vmatrix} x_4 - x_3 & x_2 - x_3 \\ y_4 - y_3 & y_2 - y_3 \end{vmatrix}$$

Intersects if:

$$(Det_{123} \cdot Det_{124} < 0) \ \wedge \ (Det_{341} \cdot Det_{342} < 0)$$

Parameters:

- L_1: XYLine to test.
- L_2: XYLine to test.

Return values:

- Boolean indicating whether or not the lines intersects.

Exceptions:

- None.

Intersect (XYPolygon a, XYPolygon b, ref XYPoint p, ref int i, ref int j, ref XYPolygon c) : void Private

The method searches counterclockwise along polygon a for intersections with polygon b. Vertices passed in the search are added to polygon c if $j \neq -1$. The intersection vertex is added to polygon c.

Parameters:

- a: XYPolygon to search along.
- b: XYPolygon to search for intersections with.
- p: Input: XYPoint used as starting point for the search. Output: Intersection point.
- i: Input: end index for first line segment of a in the search. Output: End index for last intersected line segment of a.
- j: Input: -1 if vertices before intersection is not to be added to list. Output: End index for last intersected line segment of b.
- c: Input/Output: XYPolygon describing the intersection area between a and b.

Return values:

Intersect returns values in the reference parameters, p, i, j and c.

Exceptions:

- none.

IntersectionPoint(XYLine: lineA, XYLine: lineB, ref XYPoint: intersectionPoint) : bool Private

Checks if the lines lineA and lineB shares a point either as a real crossing point or as a shared end point or a end point of the one line being in the other line.

Parameters:

- lineA: XYLine
- lineB: XYLine

Return values:

- returns true if lineA and lineB has shared point.
- The intersectionPoint parameter holds the intersection point if any. Called by reference.

Exceptions:

- None

IsPointInLine (XYPoint: point, XYLine line) : bool Private

Determines if a point is included in a line either in the interior or as one of the end points.

Parameters:

- point: XYPoint

Return values:

- line: XYLine

Exceptions:

- None

IsPointInPolygon (XYPoint: point, XYPolygon polygon) : bool Public

Determines if a point is inside a polygon. The method is an implementation of the Winding number test, valid for convex as well as concave polygons.

Parameters:

- point: XYPoint
- polygon: XYPolygon

Return values:

- true if the point is inside the polygon, false if outside or on edge.

Exceptions:

- None

IsPointInPolygonOrOnEdge (XYPoint: point, XYPolygon polygon) : bool Private

Determines if a point is inside or on the edge of a polygon. The method is an implementation of the Winding number test, valid for convex as well as concave polygons. The method is based on a combination of IsPointInPolygon and IsPointInLine.

Parameters:

- point: XYPoint
- polygon: XYPolygon

Return values:

- true if the point is inside or on the edge of the polygon, false otherwise

Exceptions:

- None

TriangleIntersectionArea(XYPolygon: a, XYPolygon: b) : double Private

The method calculates the intersection area of triangle a and b both of type XYPolygon. Triangle a has the vertices a_0, a_1 and a_2. Vertex a_1 has coordinates $a_{1,x}$, $a_{1,y}$. Edge from vertex a_1 to a_2 is denoted a_{12}. The algorithm creates polygon c by following alternately a and b.

```
$i = 1;$
$j = -1$
$p = a_0$
$InterSect(a,b,p,i,j,c)$
$j_{stop} = j$
$InterSect(a,b,p,i,j,c)$
$InterSect(b,a,p,j,i,c)$
$if\ \ j \neq j_{stop}$
    $InterSect(a,b,p,i,j,c)$
    $InterSect(b,a,p,j,i,c)$
$TriangleInter\,sec\,tionArea = c.CalculateArea$
```

Parameters:

- a: triangle of type XYPolygon
- b: triangle of type XYPolygon

Return values:

- The shared area of the two triangles.

Exceptions:

- none.

org.OpenMI.Utilities.XYPoint:

Methods:

XYPoint (double: x, double: y): void Public

Constructor

Parameters:

- x: x-coordinate
- y: y-coordinate

Return values:

- None

Exceptions:

- None

Properties:

[property] X : double Public

Sets or gets the x-coordinate of the point.

Get: **Return value:** double x-coordinate

Set: **Parameter**: double x-coordinate

[property] Y : double Public

Sets or gets the y-coordinate of the point.

Get: **Return value:** double y-coordinate

Set: **Parameter**: double y-coordinate

org.OpenMI.Utilities.XYPolyline:

Methods:

XYPolyline (): void Public

Constructor.

Parameters:

- None

Return values:

- None

Exceptions:

- None

GetLength(): double Public

Sums the length of the line segments of the poly line.

Parameters:

- None

Return values:

- Length of the polyline.

Exceptions:

- None

GetLine(int index): XYLine Public

Extracts the index'th line (0-based) from the polyline.

Parameters:

- Index: int

Return values:

- The index'th XYLine of the polyline

Exceptions:

- None

GetX(int index): void Public

Returns the start x-coordinate of the index'th line (0-based) of the polyline.

Parameters:

- Index: int

Return values:

- X-coordinate

Exceptions:

- None

GetY (int index): void Public

Returns the start y-coordinate of the index'th line (0-based) of the polyline.

Parameters:

- Index: int

Return values:

- Y-coordinate

Exceptions:

- None

Properties

[property] Points : ArrayList Public

Gets the list of points in the line.

Get: **Return value:** ArrayList of XYPoints

org.OpenMI.Utilities.XYLine:

Methods:

XYLine(): void Public

Constructor

Parameters:

- None

Return values:

- None

Exceptions:

- None

GetLength(): double Public

Calculates the length of the line.

Parameters:

- None

Return values:

- Line Length

Exceptions:

- None

GetMidpoint(): XYPoint Public

Returns the midpoint of the line.

$$p_{midpoint} = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right)$$

Parameters:

- None

Return values:

- Midpoint: XYPoint

Exceptions:

- None

Properties

[property] P1 : XYPoint Public

Gets the one end-point of the line.

Get: **Return value:** XYPoint

[property] P2 : XYPoint Public

Gets the other end-point of the line.

Get: **Return value:** XYPoint

org.OpenMI.Utilities.XYPolygon:

Methods:

XYPolygon(): void Public

Constructor.

Parameters:

- None

Return values:

- None

Exceptions:

- None

GetArea(): double Public

The area of a polygon is found as the sum of the determinants for each vector along the polygon.

$$A = 0.5 \cdot \left(\begin{vmatrix} x_0 & x_1 \\ y_0 & y_1 \end{vmatrix} + \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} + \ldots + \begin{vmatrix} x_{n-1} & x_0 \\ y_{n-1} & y_0 \end{vmatrix} \right)$$

Parameters:

- None

Return values:

- Polygon area

Exceptions:

- None

GetLine(int index): XYLine Public

Extracts the index'th line (0-based) from the polyline**.**

Parameters:

- Index: int

Return values:

- Index'th line: XYLine

Exceptions:

- None

GetTriangulation(): ArrayList Public

Finds a triangulation of the polygon. The method returns an ArrayList of XYPolygons that are all triangles. The method is ear cutting until the remaining polygon is a triangle itself.

Parameters:

- None

Return values:

- An Arraylist of triangles of type XYPolygon

Exceptions:

none

IsConvex (int pointIndex): bool Public

Decides whether a given point in the polygon is convex or concave. The method is used during when deciding if a given point is the midpoint of an ear. Ears needs to have a midpoint that is convex.

Parameters:

- pointIndex: int

Return values:

- true if the point in the polygon is convex.

Exceptions:

- None

FindEar () : int Public

Searches along the polygon until a convex point is found. If the triangle composed of the found point, the point before and the point after constitutes a triangle that is not intersected by other lines of the polygon, the triangle is said to be an ear and the index of the point is returned.

Parameters:

- None

Return values:

- Index of the midpoint of the ear triangle.

Exceptions:

- None

IsIntersected(int i) : bool Public

Decides if the triangle formed by P_{i-1}, Pi$_{-1}$, P_{i+1} of the polygon surrounds any of the other points in the polygon.

Parameters:

- i; Index of the midpoint of the triangle.

Return values:

- true if the triangle formed by P_{i-1}, Pi$_{-1}$, P_{i+1} includes any of the other polygon points.

Exceptions:

- None

Properties

[property] Points : ArrayList Public

Gets the list of points in the line.

Get: **Return value:** ArrayList of XYPoints

Annex I-C org.OpenMI.Utilities.Wrapper package

org.OpenMI.Utilities.Wrapper.IEngineApiAccess:

The IEngineApiAccess is the interface the ModelEngine component must implement when used with the SmartWrapper.

Methods:

Create (Hashtable properties): void Public

The Create method will be invoked just after creation of the object that implements the IEngineApiAccess interfac

Parameters:

- Properties : Hashtable with the same contents as the Component arguments in the ILinkableComponent interface. Typically any information needed for initialization of the model will be included in this table. This could be path and file names for input files.

Return values:

- none

Exceptions:

- none

Initialize() : Void Public

This method will be invoked after the Create method has been invoked and before any other methods in the IEngineApiAccess interface has been invoked. Typically, the model engine will be populated when the Initialize method is invoked. E.g. by reading input files.

Parameters:

- None

Return values:

- none

Exceptions:

- None

Finalize() : Void Public

This method will be invoked after all computations are completed.

Parameters:

- none

Return values:

- none

Exceptions:

- None

Dispose() : Void Public

This method will be invoked after all computations are completed and after the method Finalize has been invoked.

Parameters:

- None

Return values:

- none

Exceptions:

- None

PerformTimeStep() : bool Public

This method will make the model engine perform one time step.

Parameters:

- None

Return values:

- Returns true if the time step was completed, otherwise it will return false

Exceptions:

- None

GetCurrentTime() : ITimeStamp Public

Get the current time of the model engine

Parameters:

- None

Return values:

- The current time for the model engine

Exceptions:

- None

GetInputTime() : ITimeStamp Public

Get the time for which the next input is needed for a specific Quantity and ElementSet combination

Parameters:

- quantityID (string):
- elementSetID (string)

Return values:

- Time for the next required values associated to a Quantiy and ElementSet combination

Exceptions:

- None

GetEarliestNeededTime () : ITime Public

Get earliest needed time, which can be used to clear the buffer. For most time stepping model engines this time will be the time for the previous time step.

Parameters:

- None
-

Return values:

- Time

Exceptions:

- None

GetValues () : IValuesSet Public

Get values from the model engine.

Parameters:

- String: QuantityID associated to the requested values
- String: ElementSetID associated to the requested values

Return values:

- The values

Exceptions:

- None

SetValues () : Void Public

Set values in the model engine.

Parameters:

- String: QuantityID associated to the values
- String: ElementSetID associated to the values
- IValuesSet: The values to set

Return values:

- void

Exceptions:

- None

Property: ID

IDstring for the model engine.

Property: Description

Description string for the model engine.