

Bijlage IX

“Ontwerp in vogelvlucht”

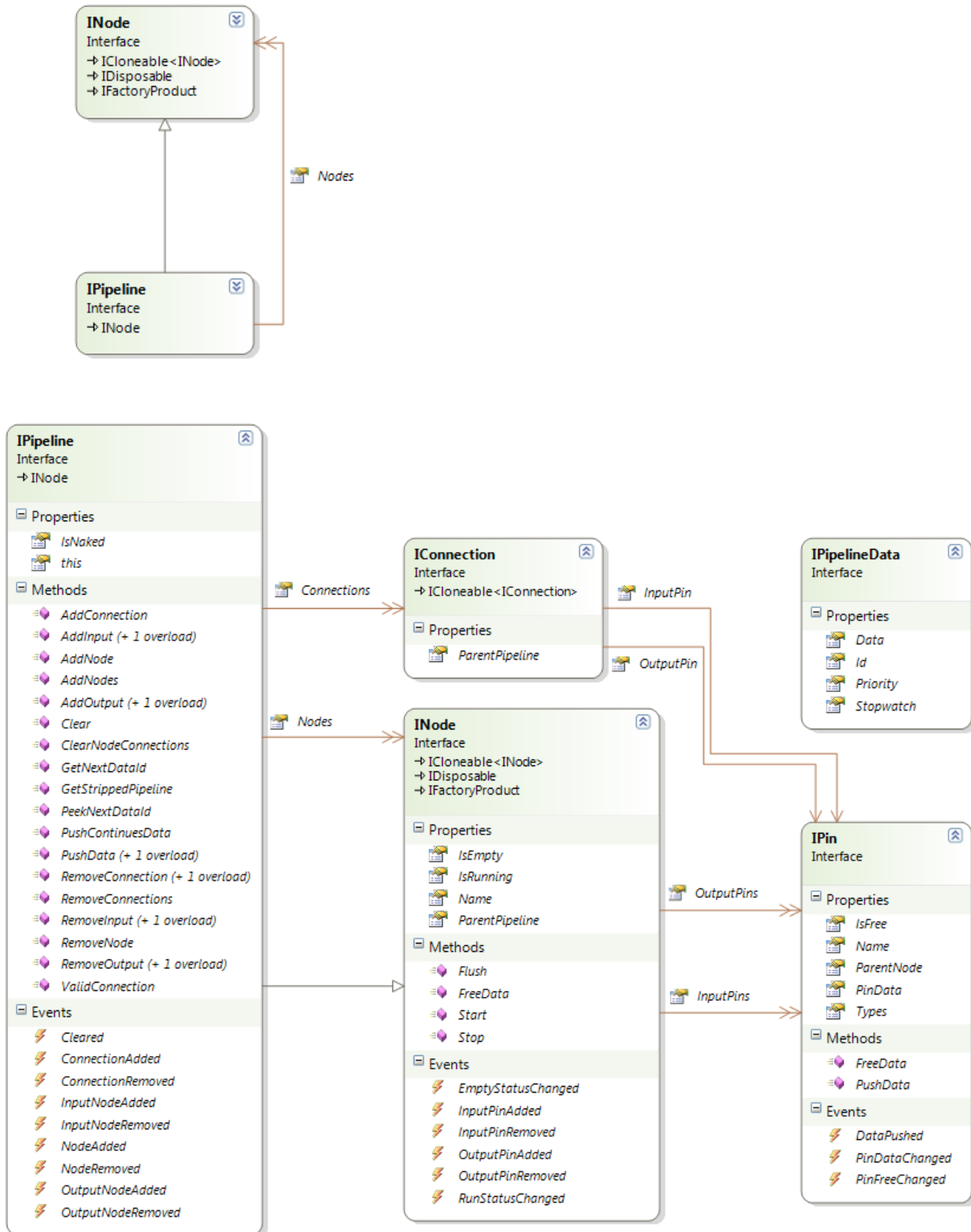
Student:	Jasper Lammers
Student nummer:	2108351
Begin studie:	2007
Afstudeer periode:	07-02-2011 t/m 24-06-2011
Opleiding:	Technische Informatica, voltijd
Bedrijf:	Memoria Media
Locatie:	Eindhoven
Bedrijfsbegeleider:	Erwin Wullems, Algemeen Directeur
Technisch begeleider:	Henk Jooren
Docentbegeleider:	Peter Lambooi
Date:	07-06-2011
Document release:	01
Document status:	Finished

Inhoudsopgave

1. Pipeline Library	2
1.1 Ontwerp	2
INode	3
IPin	3
IConnection	3
IPipeline	3
IPipelineData	3
1.2 Implementatie	4
INode	4
IPin	5
IConnection	5
IPipeline	6
IPipelineData	8
2. Cloud Framework	9
3.1 Ontwerp	10
3.2 Threading	12
3.3 Implementatie	13
ICloudInstanceInternal (Internal Service Interface)	13
ICloudInstanceExternalCallback (Callback Interface)	15
ICloudInstanceProxy (Local Service Interface)	16

1. Pipeline Library

1.1 Ontwerp



INode

Deze interface vertegenwoordigt een node. Een node wordt gebruikt om werk uit te voeren.

IPin

Elke node heeft een verzameling met input en output pinnen. De node houdt deze pinnen in de gate, en kan data van pinnen lezen en naar pinnen toe schrijven.

IConnection

Een connectie verbind pinnen met elkaar. Het is de bedoeling dat de pipeline de dataflow reguleert.

IPipeline

Deze interface vertegenwoordigt de pipeline zelf. Een pipeline heeft een verzameling nodes, connecties, input nodes, input pins, output nodes en output pins.

Connecties verbinden de pinnen van de lijst met nodes, om zo de pipeline te vormen. Input en output nodes en pinnen lopen naar buiten, en kunnen bijvoorbeeld worden gebruikt om data te pushen, te lezen door een programma of andere pipeline.

IPipelineData

Deze interface vertegenwoordigt data die door de pipeline heen gaat.

1.2 Implementatie

INode

```

/// A readonly collection containing all inputgroupcontainers of this node
ReadOnlyCollection<IPin> InputPins { get; }

/// Returns true if the node is empty of data
bool IsEmpty { get; }

/// Returns true if the node is running
bool IsRunning { get; }

/// The name of this node
string Name { get; set; }

/// A collection containing all outputpins of this node
ReadOnlyCollection<IPin> OutputPins { get; }

/// The pipeline this node is in. If this node is the most upper pipeline, this value will either
refer to itself or be null
/// This value should not be set, this will be done by the pipelines themselves
IPipeline ParentPipeline { get; set; }

/// Event that will be triggered if the empty status of this node is changed
event DataSenderDelegate<INode, bool> EmptyStatusChanged;

/// This event will be triggered if a new input pin is added to the node
event DoubleDataSenderDelegate<INode, int, IPin> InputPinAdded;

/// This event will be triggered if an input pin is removed from the node
event DoubleDataSenderDelegate<INode, int, IPin> InputPinRemoved;

/// This event will be triggered if a new output pin is added to the node
event DoubleDataSenderDelegate<INode, int, IPin> OutputPinAdded;

/// This event will be triggered if an output pin is removed from the node
event DoubleDataSenderDelegate<INode, int, IPin> OutputPinRemoved;

/// This event will be triggered if the run status of this node is changed
event DataSenderDelegate<INode, bool> RunStatusChanged;

/// Flushes the node of all data (making it empty)
void Flush();

/// Will free all the data from all the pins in this node
void FreeData();

/// Starts the node
bool Start();

/// Stops the node
bool Stop();

```

IPin

```
/// Will trigger when the data on this pin changes
event DataSenderDelegate<IPin, IPipelineData> PinDataChanged;

/// Will trigger when data is pushed to the pin
event DataSenderDelegate<IPin, IPipelineData> DataPushed;

/// Will trigger when the free status of this pin changes
event DataSenderDelegate<IPin, bool> PinFreeChanged;

/// The data currently in the pin
IPipelineData PinData { get; }

/// The name of this pin
string Name { get; }

/// This methode will push data into the pin, this can only be done if the pin is free
void PushData(IPipelineData data);

/// Will be called when a data piece was used and should be removed from the pin
void FreeData();

/// Indicates whether the pin is free to get data pushed to it
bool IsFree { get; }

/// A collection with types that are supported by this pin
ReadOnlyCollection<Type> Types { get; }

/// The IInputGroup object that contains this pin
INode ParentNode { get; set; }
```

IConnection

```
/// The input pin this connection is connecting
IPin InputPin { get; }

/// The output pin this connection is connecting
IPin OutputPin { get; }

/// The pipeline this connection belongs too
IPipeline ParentPipeline { get; }
```

IPipeline

```

/// Will return a readonly collection of all connections in this pipeline
ReadOnlyCollection<IConnection> Connections { get; }

/// False if one or more nodes in this pipeline are also pipelines
bool IsNaked { get; }

/// Will return a readonly collection of all nodes in this pipeline
ReadOnlyCollection<INode> Nodes { get; }

/// Will return the node at the given index
/// <returns>The node at a given index. Null if index is out of bounds</returns>
INode this[int index] { get; }

/// Will be triggered when the pipeline is cleared (Clear() is called)
event SenderDelegate<IPipeline> Cleared;

/// Will be triggered when a connection is added to the pipeline
event DoubleDataSenderDelegate<IPipeline, int, IConnection> ConnectionAdded;

/// Will be triggered when a connection is removed from the pipeline
event DoubleDataSenderDelegate<IPipeline, int, IConnection> ConnectionRemoved;

/// Will be triggered when an input node is added to the pipeline
event DoubleDataSenderDelegate<IPipeline, int, INode> InputNodeAdded;

/// Will be triggered when an input node is removed from the pipeline
event DoubleDataSenderDelegate<IPipeline, int, INode> InputNodeRemoved;

/// Will be triggered when a node is added to the pipeline
event DoubleDataSenderDelegate<IPipeline, int, INode> NodeAdded;

/// Will be triggered when a node is removed from the pipeline
event DoubleDataSenderDelegate<IPipeline, int, INode> NodeRemoved;

/// Will be triggered when an output node is added to the pipeline
event DoubleDataSenderDelegate<IPipeline, int, INode> OutputNodeAdded;

/// Will be triggered when an output node is removed from the pipeline
event DoubleDataSenderDelegate<IPipeline, int, INode> OutputNodeRemoved;

/// Will add a new connection, based on two pins. The node of each pin should already be present
in the pipeline
/// <returns>Returns true if successful</returns>
bool AddConnection(IPin outputPin, IPin inputPin);

/// Adds the provided node as an input node. The node should already be present in the pipeline
void AddInput(INode inputNode);

/// Adds the provided pin as an input pin. The node containing the pin should already be present
in the pipeline
void AddInput(IPin inputPin);

/// Will add the provided node to the pipeline
void AddNode(INode node);

/// Will add the provided nodes to the pipeline
void AddNodes(IEnumerable<INode> nodes);

/// Adds the provided node as an output node. The node should already be present in the pipeline
void AddOutput(INode outputNode);

```

```
/// Adds the provided pin as an output pin. The node containing the pin should already be present
in the pipeline
void AddOutput(IPin outputPin);

/// Clears all nodes, connections and references in the pipeline. It will make the pipeline empty
void Clear();

/// Removes all connections connected to the provided node
void ClearNodeConnections(INode node);

/// Will return the next data id, which will be used to id data traveling through a pipeline. This
will automatically raise the counter.
uint GetNextDataId();

/// Returns the same pipeline, but all nodes that are pipelines will be converted to nodes. This
will return the same instance if IsNaked is true.
IPipeline GetStrippedPipeline();

/// Same as GetNextDataId(), but will not raise the counter inside the IPipeline object.
uint PeekNextDataId();

/// Same as PushData(), but will keep the IPipelineData object intact. This can be used to connect
two IPipeline objects
/// <returns>Returns true if successful</returns>
bool PushContinuesData(IPin inputPin, IPipelineData pipelineData);

/// Will push provided data to the provided input pin.
/// <returns>Returns true if successful</returns>
bool PushData(IPin inputPin, object data);

/// Will push provided data to the provided input pin, with a given priority
/// <returns>Returns true if successful</returns>
bool PushData(IPin inputPin, object data, int priority);

/// Removes provided connection
/// <returns>Returns true if successful</returns>
bool RemoveConnection(IConnection connection);

/// Removes a connection, specified by the two provided pins
/// <returns>Returns true if successful</returns>
bool RemoveConnection(IPin outputPin, IPin inputPin);

/// Removes all connections connected to the provided pin
/// <returns>Returns true if successful</returns>
bool RemoveConnections(IPin pin);

/// Removes provided input node as input. This will not remove the node from the pipeline
/// <returns>Returns true if successful</returns>
bool RemoveInput(INode inputNode);

/// Removes provided input pin as input. This will not remove the pin's node from the pipeline
/// <returns>Returns true if successful</returns>
bool RemoveInput(IPin inputPin);

/// Will remove provided node from the pipeline
/// <returns>Returns true if successful</returns>
bool RemoveNode(INode node);

/// Removes provided output node as output. This will not remove the node from the pipeline
/// <returns>Returns true if successful</returns>
bool RemoveOutput(INode outputNode);
```

```
/// Removes provided output pin as output. This will not remove the pin's node from the pipeline
/// <returns>Returns true if successful</returns>
bool RemoveOutput(IPin outputPin);

/// Returns whether a connection is possible to make, given provided pins.
bool ValidConnection(IPin outputPin, IPin inputPin);
```

IPipelineData

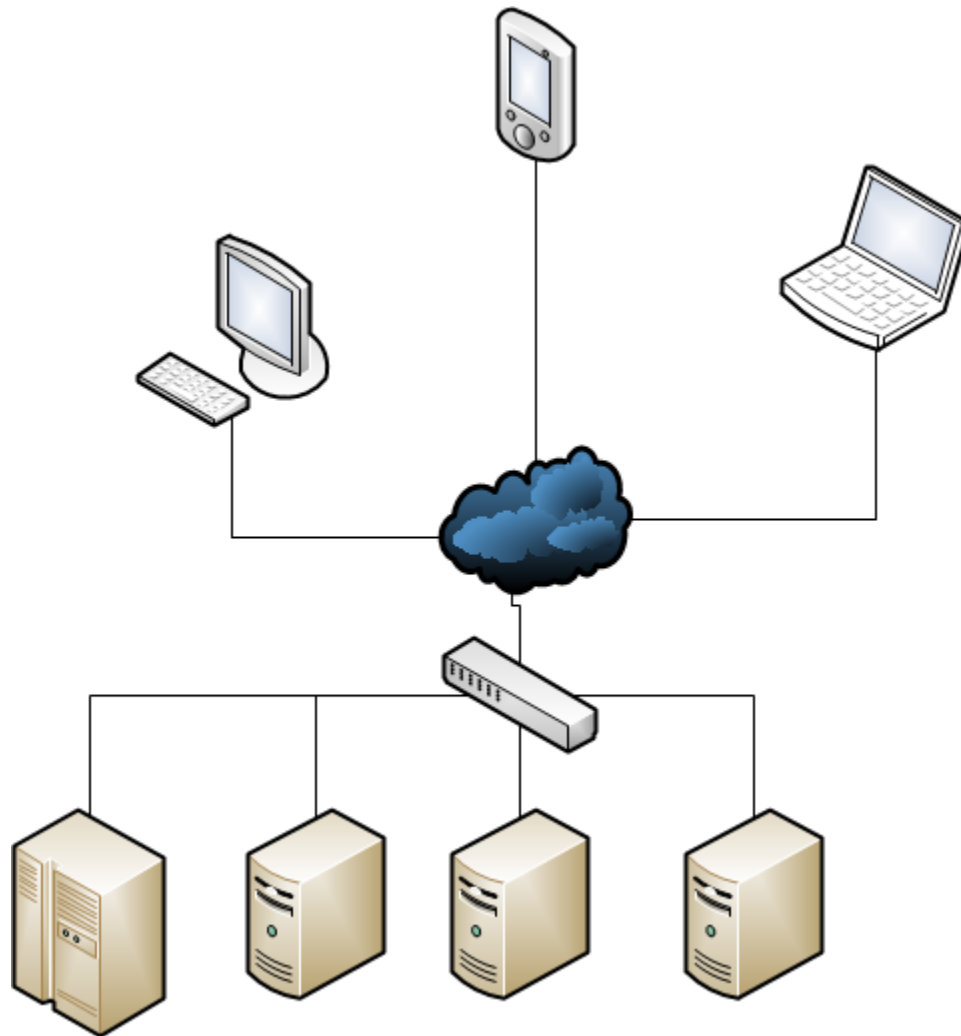
```
/// The data that is in this data item
object Data { get; }

/// The id of this data item
uint Id { get; }

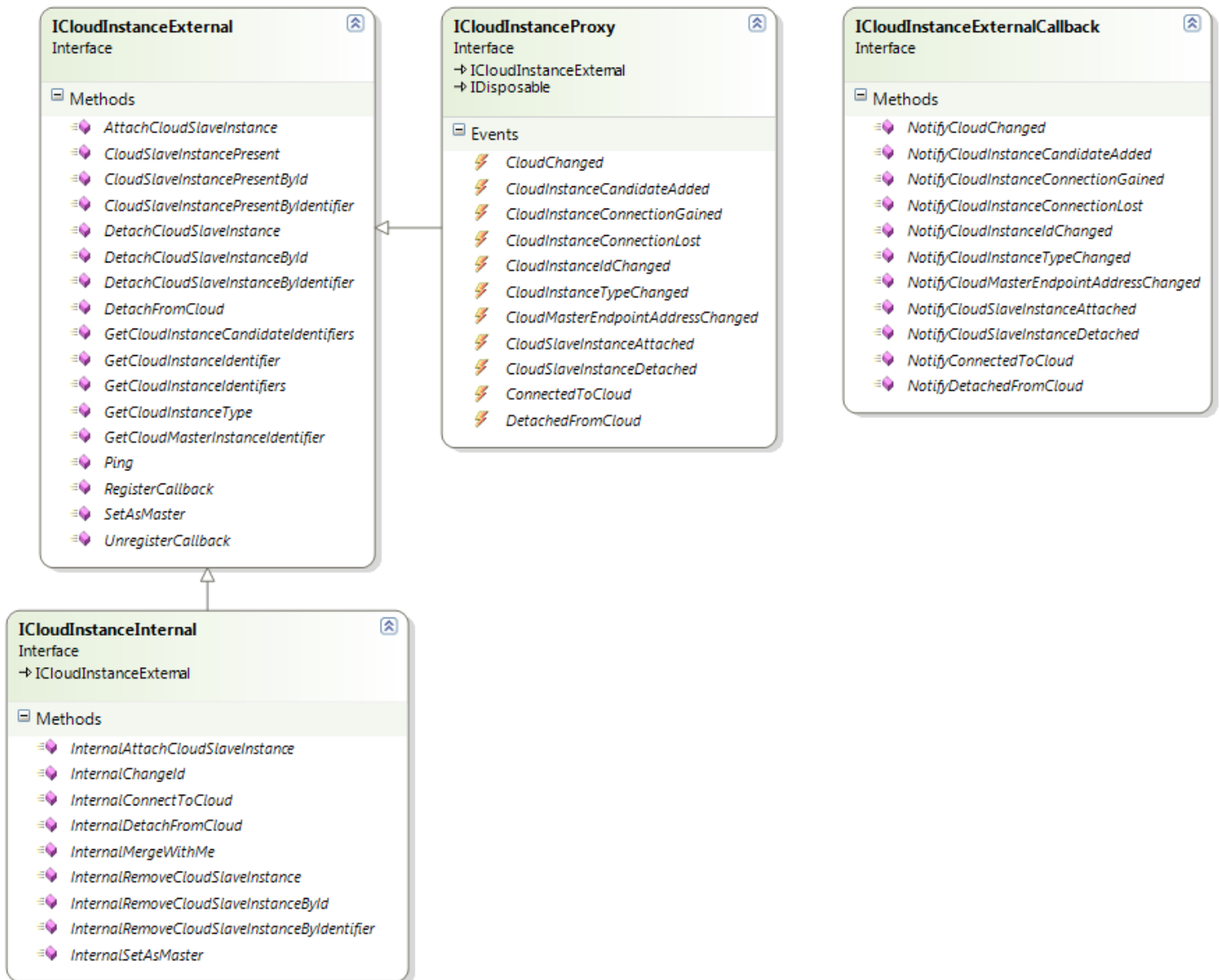
/// The priority of this data item
int Priority { get; }

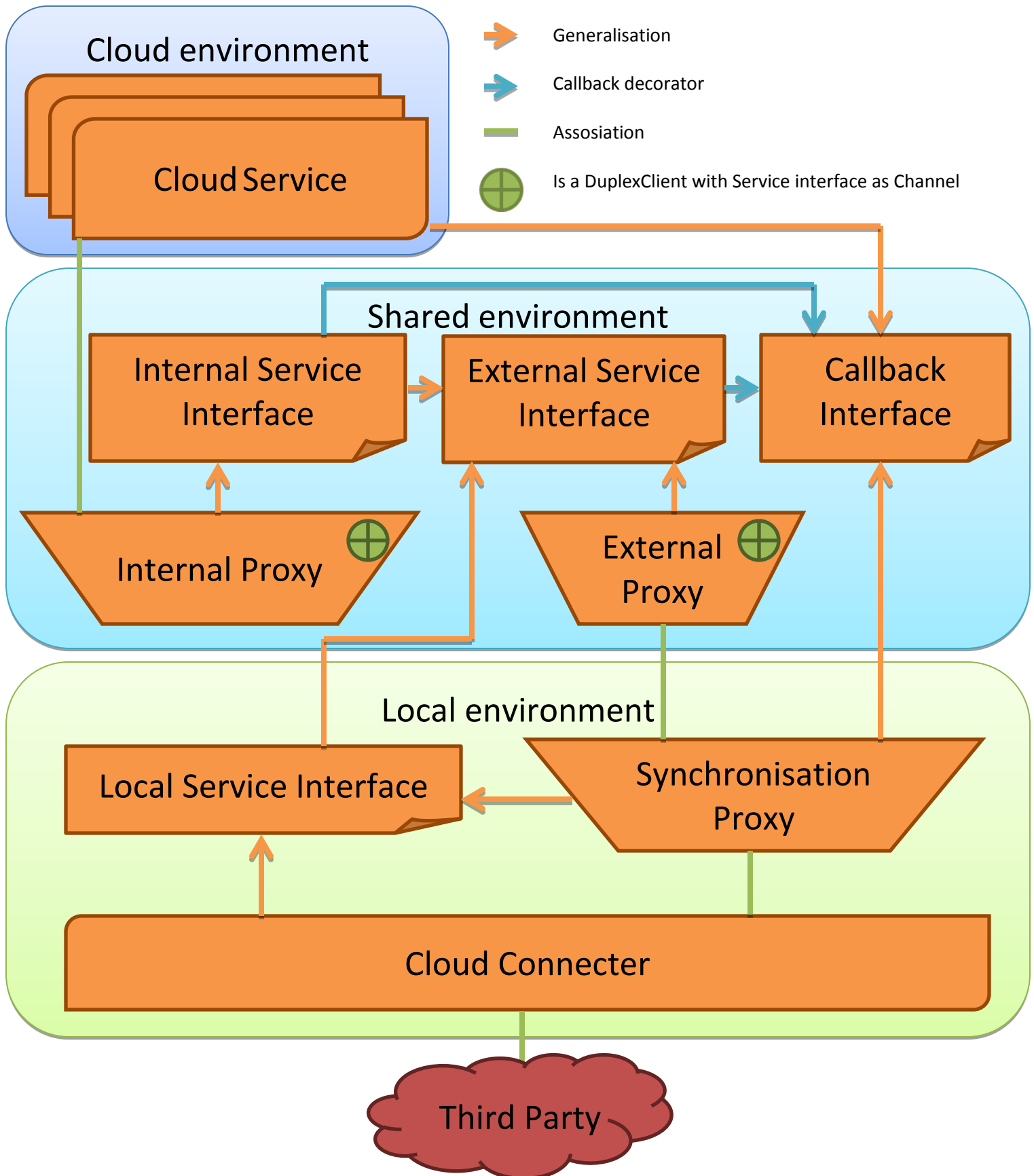
/// This stopwatch is activated by the pipeline, to track the time this datapackage travels
through the pipeline
Stopwatch Stopwatch { get; }
```

2. Cloud Framework

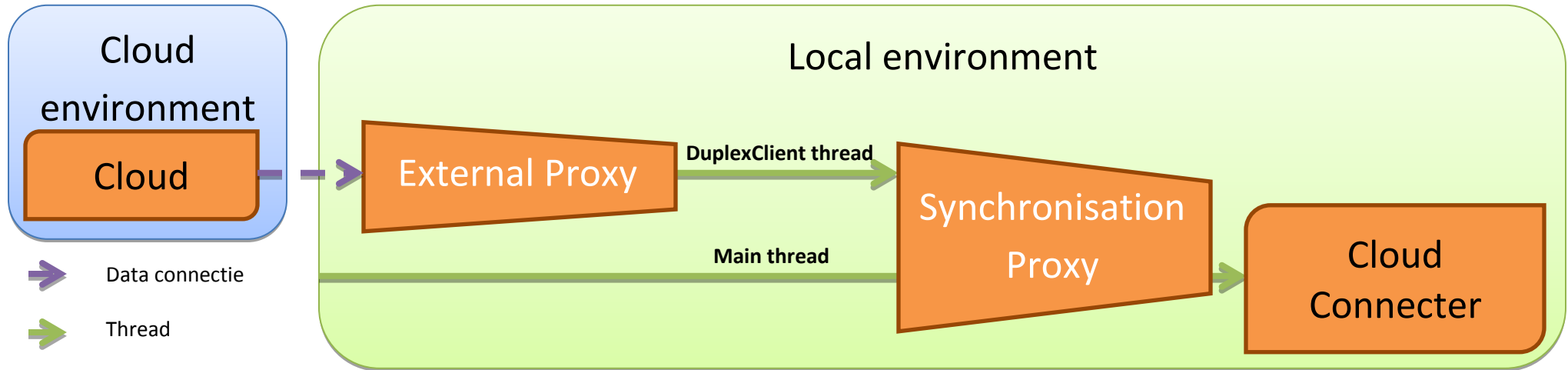


3.1 Ontwerp





3.2 Threading



Het framework handelt alle communicatie met de cloud af op een losse thread. Vervolgens zorgt de synchronisation proxy ervoor dat de data op de thread van de cloud connector komt. Andersom geldt hetzelfde: als er data naar de cloud moet, zorgt de synchronisation proxy ervoor dat deze op de goede thread gebracht wordt.

3.3 Implementatie

ICloudInstanceInternal (Internal Service Interface)

```

/// Attaches a slave to the cloud
bool InternalAttachCloudSlaveInstance(string endpointAddress, CloudInstanceGatherMethods
cloudInstanceGatherMethods, CloudInstanceGatherGuidelines cloudInstanceGatherGuidelines);

/// Will change the id of this cloud instance
/// Undefined/Master: Will not do anything
/// Slave: Will change id
void InternalChangeId(uint id);

/// Will connect this cloud instance to the caller's cloud
/// Undefined: Will connect to the cloud as a slave
/// Master/Slave: Will detach from current cloud and connect to the cloud as a slave
void InternalConnectToCloud(uint id, string endpointAddress, CloudInstanceIdentifier[]
cloudInstanceIdentifiers);

/// Will detach the slave/master from a cloud
void InternalDetachFromCloud();

/// Will tell a remote master to absorb this cloud into its own
bool InternalMergeWithMe(string endpointAddress);

/// Removes a slave from the cloud
/// Only works on master
bool InternalRemoveCloudSlaveInstance(string endpointAddress);

bool InternalRemoveCloudSlaveInstanceById(uint cloudInstanceId);

bool InternalRemoveCloudSlaveInstanceByIdentifier(CloudInstanceIdentifier
cloudInstanceIdentifier);

/// Sets this cloud instance as a master
/// Undefined: Will turn it into a master, making a new cloud
/// Slave: Will turn the slave into a master
/// Master: Will do nothing
bool InternalSetAsMaster();

```

ICloudInstanceExternal (External Service Interface)

```

/// Undefined: Will not do anything
/// Master/Slave: Will detach the instance from the cloud
void DetachFromCloud();

/// Returns all cloud candidates (cloud instances that are offline)
CloudInstanceIdentifier[] GetCloudInstanceCandidateIdentifiers();

/// Returns the cloud instance identifier of the current instance
CloudInstanceIdentifier GetCloudInstanceIdentifier();

/// Returns the entire cloud
CloudInstanceIdentifier[] GetCloudInstanceIdentifiers();

/// Returns the cloud computer type of this cloud service
CloudInstanceTypes GetCloudInstanceType();

/// Returns the cloud master identifier of the current instance
CloudInstanceIdentifier GetCloudMasterInstanceIdentifier();

/// Will not do anything. This method can be used to check the connection between services
void Ping();

/// Will register the service slave for callbacks
void RegisterCallback();

/// Sets this cloud instance as a master
/// Undefined: Will turn it into a master, making a new cloud
/// Master/Slave: Will return false
bool SetAsMaster();

/// Will unregister the service slave for callbacks
void UnregisterCallback();

/// Removes a slave from the cloud
/// Undefined: Will return false
/// Master/Slave: Will remove the cloud instance from it's cloud
bool DetachCloudSlaveInstance(string endpointAddress);

[OperationContract]
bool DetachCloudSlaveInstanceByIdentifier(CloudInstanceIdentifier cloudInstanceIdentifier);

[OperationContract]
bool DetachCloudSlaveInstanceById(uint cloudInstanceId);

/// Undefined: Will return false
/// Master/Slave: Will return if a cloud instance is present in the cloud
bool CloudSlaveInstancePresent(string endpointAddress);
bool CloudSlaveInstancePresentByIdentifier(CloudInstanceIdentifier cloudInstanceIdentifier);
bool CloudSlaveInstancePresentById(uint cloudInstanceId);

/// Attaches a slave to the cloud
/// Undefined: Will return false
/// Master/Slave: Will add the instance to the cloud, depending on the cloudInstanceGatherMethods argument
/// OnlineOnly: Will add instance to cloud if it is online. When it is not, it will return false and not
do anything
/// CandidateAble: Will add instance to cloud if it is online. When it is not, it will return false and
add the instance to the candidates list
[OperationContract]
bool AttachCloudSlaveInstance(string endpointAddress, CloudInstanceGatherMethods
cloudInstanceGatherMethods = CloudInstanceGatherMethods.Merge,
CloudInstanceGatherGuidelines cloudInstanceGatherGuidelines = CloudInstanceGatherGuidelines.OnlineOnly);

```

ICloudInstanceExternalCallback (Callback Interface)

```

/// Will be called when the cloud is changed
void NotifyCloudChanged();

/// Will be called when a slave is attached to the cloud
void NotifyCloudSlaveInstanceAttached(CloudInstanceIdentifier cloudInstanceIdentifier);

/// Will be called when a slave is attached from the cloud
void NotifyCloudSlaveInstanceDetached(CloudInstanceIdentifier cloudInstanceIdentifier);

/// Will be called when a candidate is added to the cloud
void NotifyCloudInstanceCandidateAdded(CloudInstanceIdentifier cloudInstanceIdentifier);

/// Will be called when a connection to a slave is lost
void NotifyCloudInstanceConnectionLost(CloudInstanceIdentifier cloudInstanceIdentifier);

/// Will be called when a connection to a lost slave is regained
void NotifyCloudInstanceConnectionGained(CloudInstanceIdentifier failedCloudInstanceIdentifier,
CloudInstanceIdentifier newCloudInstanceIdentifier);

/// Will be called when the id of this cloud instance is changed
void NotifyCloudInstanceIdChanged(uint id);

/// Will be called when the type of this cloud instance is changed
void NotifyCloudInstanceTypeChanged(CloudInstanceTypes cloudInstanceType);

/// Will be called when the master of this cloud has moved to another cloud instance
void NotifyCloudMasterEndpointAddressChanged(CloudInstanceIdentifier cloudMasterInstanceIdentifier);

/// Will be called when this cloud instance connected to a cloud
void NotifyConnectedToCloud(uint id, CloudInstanceTypes cloudInstanceType);

/// Will be called when this cloud instance detached/disconnected from it's cloud
void NotifyDetachedFromCloud();

```

ICloudInstanceProxy (Local Service Interface)

```

/// Will be triggered when the cloud is changed
event SenderDelegate<ICloudInstanceProxy> CloudChanged;

/// Will be triggered when a slave is attached to the cloud
event DataSenderDelegate<ICloudInstanceProxy, CloudInstanceId> CloudSlaveInstanceAttached;

/// Will be triggered when a slave is attached from the cloud
event DataSenderDelegate<ICloudInstanceProxy, CloudInstanceId> CloudSlaveInstanceDetached;

/// Will be triggered when a candidate is added to the cloud
event DataSenderDelegate<ICloudInstanceProxy, CloudInstanceId> CloudInstanceCandidateAdded;

/// Will be triggered when a connection to a lost slave is regained
event DoubleDataSenderDelegate<ICloudInstanceProxy, CloudInstanceId,
CloudInstanceId> CloudInstanceConnectionGained;

/// Will be triggered when a connection to a slave is lost
event DataSenderDelegate<ICloudInstanceProxy, CloudInstanceId> CloudInstanceConnectionLost;

/// Will be triggered when the id of this cloud instance is changed
event DataSenderDelegate<ICloudInstanceProxy, uint> CloudInstanceIdChanged;

/// Will be triggered when the type of this cloud instance is changed
event DataSenderDelegate<ICloudInstanceProxy, CloudInstanceTypes> CloudInstanceTypeChanged;

/// Will be triggered when the master of this cloud has moved to another cloud instance
event DataSenderDelegate<ICloudInstanceProxy, CloudInstanceId>
CloudMasterEndpointAddressChanged;

/// Will be triggered when this cloud instance connected to a cloud
event DoubleDataSenderDelegate<ICloudInstanceProxy, uint, CloudInstanceTypes> ConnectedToCloud;

/// Will be triggered when this cloud instance detached/disconnected from its cloud
event SenderDelegate<ICloudInstanceProxy> DetachedFromCloud;

```