# ANEXOS

## ANEXOS

## ANEXO A.1: CÓDIGO FUENTE RTPS

### Behavior

#### FakeRtpsReader.cs

using log4net;

using Doopec.Rtps.RtpsTransport;

using Doopec.Rtps.SharedMem;

using Rtps.Behavior;

using Rtps.Structure;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

using Rtps.Structure.Types;

namespace Doopec.Rtps.Behavior

{

public class FakeRtpsReader<T> : StatefulReader<T>, IDisposable

{

private static readonly ILog log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

private IList<Writer<T>> writers = new List<Writer<T>>();

public FakeRtpsReader(GUID guid)

: base(guid)

{

IRtpsDiscovery discoveryModule = RtpsEngineFactory.Instance.DiscoveryModule;

discoveryModule.RegisterEndpoint(this);

discoveryModule.EndpointDiscovery += OnDiscoveryEndpoints;

AddWriters(discoveryModule);

}

public void Dispose()

{

RemoveAllWriters();

IRtpsDiscovery discoveryModule = RtpsEngineFactory.Instance.DiscoveryModule;

discoveryModule.UnregisterEndpoint(this);

discoveryModule.EndpointDiscovery -= OnDiscoveryEndpoints;

}

private void OnDiscoveryEndpoints(object sender, DiscoveryEventArgs e)

{

Writer<T> writer = e.EventData as Writer<T>;

if (writer == null)

return;

if (e.Reason == EventReason.NEW\_ENDPOINT)

writers.Add(writer);

else if (e.Reason == EventReason.NEW\_ENDPOINT)

writers.Remove(writer);

}

private void AddWriter(Writer<T> writer)

{

//TODO

//WriterProxy<T> writerProxy = new WriterProxy<T>();

//this.MatchedWriterAdd(writerProxy);

writers.Add(writer);

writer.HistoryCache.Changed += OnChangedHistoryCache;

}

private void AddWriters(IRtpsDiscovery discoveryModule)

{

foreach (var endpoint in discoveryModule.Endpoints)

{

if (endpoint is Writer<T>)

AddWriter(endpoint as Writer<T>);

}

}

private void RemoveAllWriters()

{

foreach(var writer in writers)

writer.HistoryCache.Changed -= OnChangedHistoryCache;

writers.Clear();

}

private void OnChangedHistoryCache(object sender, EventArgs e)

{

log.Debug("A new change has been detected");

HistoryCache<T> whc = sender as HistoryCache<T>;

if (whc != null)

{

CacheChange<T> change = whc.GetChange();

ReaderCache.AddChange(change);

whc.RemoveChange(change);

}

}

}

}

#### FakeRtpsWriter.cs

using Doopec.Rtps.RtpsTransport;

using Doopec.Rtps.SharedMem;

using Rtps.Behavior;

using Rtps.Structure;

using Rtps.Structure.Types;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Doopec.Rtps.Behavior

{

public class FakeRtpsWriter<T> : StatefulWriter<T>, IDisposable

{

private IList<Reader<T>> readers = new List<Reader<T>>();

private WriterWorker worker;

public FakeRtpsWriter(GUID guid)

: base(guid)

{

IRtpsDiscovery discoveryModule = RtpsEngineFactory.Instance.DiscoveryModule;

discoveryModule.RegisterEndpoint(this);

discoveryModule.EndpointDiscovery += OnDiscoveryEndpoints;

AddReaders(discoveryModule);

worker = new WriterWorker();

worker.Start((int)this.heartbeatPeriod.AsMillis());

}

public void Dispose()

{

readers.Clear();

IRtpsDiscovery discoveryModule = RtpsEngineFactory.Instance.DiscoveryModule;

discoveryModule.UnregisterEndpoint(this);

discoveryModule.EndpointDiscovery -= OnDiscoveryEndpoints;

worker.End();

}

private void OnDiscoveryEndpoints(object sender, DiscoveryEventArgs e)

{

Reader<T> reader = e.EventData as Reader<T>;

if (reader == null)

return;

if (e.Reason == EventReason.NEW\_ENDPOINT)

readers.Add(reader);

else if (e.Reason == EventReason.NEW\_ENDPOINT)

readers.Remove(reader);

}

private void AddReader(Reader<T> writer)

{

readers.Add(writer);

}

private void AddReaders(IRtpsDiscovery discoveryModule)

{

foreach (var endpoint in discoveryModule.Endpoints)

{

if (endpoint is Reader<T>)

AddReader(endpoint as Reader<T>);

}

}

}

}

#### RtpsStatefulReader.cs

using log4net;

using Doopec.Rtps.RtpsTransport;

using Doopec.Rtps.SharedMem;

using Doopec.Utils.Transport;

using Mina.Core.Buffer;

using Rtps.Behavior;

using Rtps.Messages;

using Rtps.Messages.Types;

using Rtps.Structure;

using Rtps.Structure.Types;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

using Data = Rtps.Messages.Submessages.Data;

using DataObj = Rtps.Structure.Types.Data;

namespace Doopec.Rtps.Behavior

{

public class RtpsStatefulReader<T> : StatefulReader<T>, IDisposable

{

private static readonly ILog log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

private IList<Writer<T>> writers = new List<Writer<T>>();

private UDPReceiver rec;

public RtpsStatefulReader(GUID guid)

: base(guid)

{

//TODO use configuration for host and port

rec = new UDPReceiver(new Uri("udp://224.0.1.111:9999"), 1024);

rec.Start();

rec.MessageReceived += NewMessage;

IRtpsDiscovery discoveryModule = RtpsEngineFactory.Instance.DiscoveryModule;

discoveryModule.RegisterEndpoint(this);

discoveryModule.EndpointDiscovery += OnDiscoveryEndpoints;

AddWriters(discoveryModule);

}

private void NewMessage(object sender, RTPSMessageEventArgs e)

{

Message msg = e.Message;

log.DebugFormat("New Message has arrived from {0}", e.Session.RemoteEndPoint);

log.DebugFormat("Message Header: {0}", msg.Header);

foreach (var submsg in msg.SubMessages)

{

switch (submsg.Kind)

{

case SubMessageKind.DATA:

Data d = submsg as Data;

log.DebugFormat("SubMessage Data: {0}", submsg.Kind);

log.DebugFormat("The KeyFlag value state is: {0}", d.HasKeyFlag);

log.DebugFormat("The DataFlag value state is: {0}", d.HasDataFlag);

log.DebugFormat("The InlineQoSFlag value state is: {0}", d.HasInlineQosFlag);

log.DebugFormat("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);

log.DebugFormat("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);

log.DebugFormat("The extraFlags value is: {0}", d.ExtraFlags.Value);

log.DebugFormat("The octetsToInlineQos value is: Aun no logro");

log.DebugFormat("The readerID is: {0}", d.ReaderId);

log.DebugFormat("The writerID is: {0}", d.WriterId);

log.DebugFormat("The writerSN is: {0}", d.WriterSN);

IoBuffer buf = IoBuffer.Wrap(d.SerializedPayload.DataEncapsulation.SerializedPayload);

buf.Order = ByteOrder.LittleEndian; //(d.Header.IsLittleEndian ? ByteOrder.LittleEndian : ByteOrder.BigEndian);

object obj = Doopec.Serializer.Serializer.Deserialize<T>(buf);

CacheChange<T> change = new CacheChange<T>(ChangeKind.ALIVE, new GUID(msg.Header.GuidPrefix, d.WriterId), d.WriterSN, new DataObj(obj), new InstanceHandle());

ReaderCache.AddChange(change);

break;

default:

log.DebugFormat("SubMessage: {0}", submsg.Kind);

break;

}

}

}

public void Dispose()

{

rec.MessageReceived -= NewMessage;

rec.Dispose();

RemoveAllWriters();

IRtpsDiscovery discoveryModule = RtpsEngineFactory.Instance.DiscoveryModule;

discoveryModule.UnregisterEndpoint(this);

discoveryModule.EndpointDiscovery -= OnDiscoveryEndpoints;

}

private void OnDiscoveryEndpoints(object sender, DiscoveryEventArgs e)

{

Writer<T> writer = e.EventData as Writer<T>;

if (writer == null)

return;

if (e.Reason == EventReason.NEW\_ENDPOINT)

writers.Add(writer);

else if (e.Reason == EventReason.NEW\_ENDPOINT)

writers.Remove(writer);

}

private void AddWriter(Writer<T> writer)

{

//TODO

//WriterProxy<T> writerProxy = new WriterProxy<T>();

//this.MatchedWriterAdd(writerProxy);

//writers.Add(writer);

//writer.HistoryCache.Changed += OnChangedHistoryCache;

}

private void AddWriters(IRtpsDiscovery discoveryModule)

{

foreach (var endpoint in discoveryModule.Endpoints)

{

if (endpoint is Writer<T>)

AddWriter(endpoint as Writer<T>);

}

}

private void RemoveAllWriters()

{

foreach (var writer in writers)

writer.HistoryCache.Changed -= OnChangedHistoryCache;

writers.Clear();

}

private void OnChangedHistoryCache(object sender, EventArgs e)

{

log.Debug("A new change has been detected");

HistoryCache<T> whc = sender as HistoryCache<T>;

if (whc != null)

{

CacheChange<T> change = whc.GetChange();

ReaderCache.AddChange(change);

whc.RemoveChange(change);

}

}

}

}

#### RtpsStatefulWriter.cs

using log4net;

using Doopec.Encoders;

using Doopec.Rtps.RtpsTransport;

using Doopec.Rtps.SharedMem;

using Doopec.Serializer;

using Doopec.Utils.Transport;

using Mina.Core.Buffer;

using Rtps.Behavior;

using Rtps.Messages;

using Rtps.Messages.Submessages;

using Rtps.Messages.Submessages.Elements;

using Rtps.Structure;

using Rtps.Structure.Types;

using System;

using System.Collections.Generic;

using System.Reflection;

using Data = Rtps.Messages.Submessages.Data;

namespace Doopec.Rtps.Behavior

{

public class RtpsStatefulWriter<T> : StatefulWriter<T>, IDisposable

{

protected static readonly ILog log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

private IList<Reader<T>> readers = new List<Reader<T>>();

private WriterWorker worker;

private UDPTransmitter trans;

public RtpsStatefulWriter(GUID guid)

: base(guid)

{

Doopec.Serializer.Serializer.Initialize(typeof(T));

IRtpsDiscovery discoveryModule = RtpsEngineFactory.Instance.DiscoveryModule;

discoveryModule.RegisterEndpoint(this);

discoveryModule.EndpointDiscovery += OnDiscoveryEndpoints;

AddReaders(discoveryModule);

//TODO Andres. Revisar esta direccion. Deberia venir de alguna configuracion

//TODO use configuration for host and port

trans = new UDPTransmitter(new Uri("udp://224.0.1.111:9999"), 256);

trans.Start();

worker = new WriterWorker(this.PeriodicWork);

worker.Start((int)this.heartbeatPeriod.AsMillis());

}

public void Dispose()

{

readers.Clear();

IRtpsDiscovery discoveryModule = RtpsEngineFactory.Instance.DiscoveryModule;

discoveryModule.UnregisterEndpoint(this);

discoveryModule.EndpointDiscovery -= OnDiscoveryEndpoints;

worker.End();

trans.Close();

}

private void OnDiscoveryEndpoints(object sender, DiscoveryEventArgs e)

{

Reader<T> reader = e.EventData as Reader<T>;

if (reader == null)

return;

if (e.Reason == EventReason.NEW\_ENDPOINT)

readers.Add(reader);

else if (e.Reason == EventReason.NEW\_ENDPOINT)

readers.Remove(reader);

}

private void AddReader(Reader<T> writer)

{

readers.Add(writer);

}

private void AddReaders(IRtpsDiscovery discoveryModule)

{

foreach (var endpoint in discoveryModule.Endpoints)

{

if (endpoint is Reader<T>)

AddReader(endpoint as Reader<T>);

}

}

private void PeriodicWork()

{

// the RTPS Writer to repeatedly announce the availability of data by sending a Heartbeat Message.

log.DebugFormat("I have to send a Heartbeat Message, at {0}", DateTime.Now);

SendHeartbeat();

if (HistoryCache.Changes.Count > 0)

{

foreach (var change in HistoryCache.Changes)

{

//SendHeartbeat();

//SendData(change);

SendDataHeartbeat(change);

}

HistoryCache.Changes.Clear(); //TODO

}

}

private void SendHeartbeat()

{

// Create a Message with Heartbeat

Message m1 = new Message();

Heartbeat heartbeat = new Heartbeat();

EntityId id1 = EntityId.ENTITYID\_UNKNOWN;

EntityId id2 = EntityId.ENTITYID\_UNKNOWN;

heartbeat.readerId = id1;

heartbeat.writerId = id2;

heartbeat.firstSN = new SequenceNumber(10);

heartbeat.lastSN = new SequenceNumber(20);

heartbeat.count = 5;

m1.SubMessages.Add(heartbeat);

SendData(m1);

}

public void SendData(CacheChange<T> change)

{

// Create a Message with InfoSource

Message msg = new Message();

EntityId readerId = EntityId.ENTITYID\_UNKNOWN;

EntityId writerId = change.WriterGuid.EntityId;

SerializedPayload payload = new SerializedPayload();

IoBuffer buff = IoBuffer.Allocate(1024);

payload.DataEncapsulation = buff.EncapsuleCDRData(change.DataValue.Value, BitConverter.IsLittleEndian ? ByteOrder.LittleEndian : ByteOrder.BigEndian);

Data data = new Data(readerId, writerId, change.SequenceNumber.LongValue, null, payload);

msg.SubMessages.Add(data);

// Write Message to bytes1 array

SendData(msg);

}

public void SendDataHeartbeat(CacheChange<T> change)

{

// Create a Message with InfoSource

Message msg = new Message();

EntityId readerId = EntityId.ENTITYID\_UNKNOWN;

EntityId writerId = change.WriterGuid.EntityId;

SerializedPayload payload = new SerializedPayload();

IoBuffer buff = IoBuffer.Allocate(1024);

payload.DataEncapsulation = buff.EncapsuleCDRData(change.DataValue.Value, BitConverter.IsLittleEndian ? ByteOrder.LittleEndian : ByteOrder.BigEndian);

Data data = new Data(readerId, writerId, change.SequenceNumber.LongValue, null, payload);

msg.SubMessages.Add(data);

Heartbeat heartbeat = new Heartbeat();

heartbeat.readerId = readerId;

heartbeat.writerId = writerId;

heartbeat.firstSN = change.SequenceNumber;

heartbeat.lastSN = change.SequenceNumber;

heartbeat.count = 1;

msg.SubMessages.Add(heartbeat);

SendData(msg);

}

/// <summary>

/// Writes a message to network

/// </summary>

/// <param name="msg"></param>

/// <returns></returns>

private void SendData(Message msg)

{

trans.SendMessage(msg);

}

}

}

#### RtpsStatelessReader.cs

using Doopec.Rtps.Messages;

using Doopec.Serializer;

using Doopec.Utils.Transport;

using log4net;

using Mina.Core.Buffer;

using Rtps.Behavior;

using Rtps.Messages;

using Rtps.Messages.Types;

using Rtps.Structure;

using Rtps.Structure.Types;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using Data = Rtps.Messages.Submessages.Data;

using DataObj = Rtps.Structure.Types.Data;

namespace Doopec.Rtps.Behavior

{

public class RtpsStatelessReader<T> : StatelessReader<T>, IDisposable where T : new()

{

protected static readonly ILog log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

protected List<UDPReceiver> UDPReceivers { get; private set; }

public RtpsStatelessReader(GUID guid)

: base(guid)

{

Doopec.Serializer.Serializer.Initialize(typeof(T));

UDPReceivers = new List<UDPReceiver>();

}

protected void InitReceivers()

{

foreach (var locator in MulticastLocatorList)

{

UDPReceiver rec = new UDPReceiver(locator, 1024);

rec.ParticipantId = this.Guid;

rec.MessageReceived += NewMessage;

UDPReceivers.Add(rec);

}

foreach (var locator in UnicastLocatorList)

{

UDPReceiver rec = new UDPReceiver(locator, 1024);

rec.ParticipantId = this.Guid;

rec.MessageReceived += NewMessage;

UDPReceivers.Add(rec);

}

}

protected void StartReceivers()

{

foreach (var rec in UDPReceivers)

{

rec.Start();

}

}

protected virtual void NewMessage(object sender, RTPSMessageEventArgs e)

{

Message msg = e.Message;

log.DebugFormat("New Message has arrived from {0}", e.Session.RemoteEndPoint);

log.DebugFormat("Message Header: {0}", msg.Header);

foreach (var submsg in msg.SubMessages)

{

switch (submsg.Kind)

{

case SubMessageKind.DATA:

Data d = submsg as Data;

log.DebugFormat("SubMessage Data: {0}", submsg.Kind);

log.DebugFormat("The KeyFlag value state is: {0}", d.HasKeyFlag);

log.DebugFormat("The DataFlag value state is: {0}", d.HasDataFlag);

log.DebugFormat("The InlineQoSFlag value state is: {0}", d.HasInlineQosFlag);

log.DebugFormat("The EndiannessFlag value state is: {0}", d.Header.Flags.IsLittleEndian);

log.DebugFormat("The octetsToNextHeader value is: {0}", d.Header.SubMessageLength);

log.DebugFormat("The extraFlags value is: {0}", d.ExtraFlags.Value);

log.DebugFormat("The octetsToInlineQos value is: Aun no logro");

log.DebugFormat("The readerID is: {0}", d.ReaderId);

log.DebugFormat("The writerID is: {0}", d.WriterId);

log.DebugFormat("The writerSN is: {0}", d.WriterSN);

IoBuffer buf = IoBuffer.Wrap(d.SerializedPayload.DataEncapsulation.SerializedPayload);

buf.Order = (d.Header.IsLittleEndian ? ByteOrder.LittleEndian : ByteOrder.BigEndian);

T obj = EncapsulationManager.Deserialize<T>(buf);

#if TODO

CacheChange<T> change = new CacheChange<T>(ChangeKind.ALIVE, new GUID(msg.Header.GuidPrefix, d.WriterId), d.WriterSN, new DataObj(obj), new InstanceHandle());

ReaderCache.AddChange(change);

#endif

break;

default:

log.DebugFormat("SubMessage: {0}", submsg.Kind);

break;

}

}

}

public void Dispose()

{

foreach (var rec in UDPReceivers)

{

rec.MessageReceived -= NewMessage;

rec.Close();

rec.Dispose();

}

UDPReceivers.Clear();

}

}

}

#### RtpsStatelessWriter.cs

using Doopec.Serializer;

using Doopec.Utils.Transport;

using Doopec.Encoders;

using log4net;

using Mina.Core.Buffer;

using Rtps.Behavior;

using Rtps.Messages;

using Rtps.Messages.Submessages;

using Rtps.Messages.Submessages.Elements;

using Rtps.Structure;

using Rtps.Structure.Types;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using Data = Rtps.Messages.Submessages.Data;

using Doopec.Rtps.Messages;

using Doopec.Serializer.Attributes;

namespace Doopec.Rtps.Behavior

{

public class RtpsStatelessWriter<T> : StatelessWriter<T>, IDisposable where T : new()

{

protected static readonly ILog log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

protected List<UDPTransmitter> UDPTransmitters { get; private set; }

private WriterWorker worker;

protected Encapsulation Scheme { get; set; }

public RtpsStatelessWriter(GUID guid)

: base(guid)

{

Doopec.Serializer.Serializer.Initialize(typeof(T));

UDPTransmitters = new List<UDPTransmitter>();

Scheme = Encapsulation.CDR\_BE;

}

protected void InitTransmitters()

{

foreach (var locator in MulticastLocatorList)

{

UDPTransmitter rec = new UDPTransmitter(locator, 1024);

rec.ParticipantId = this.Guid;

UDPTransmitters.Add(rec);

}

// TODO. Just for testing. I dont like so many messages

//foreach (var locator in UnicastLocatorList)

//{

// UDPTransmitter trans = new UDPTransmitter(locator, 1024);

// trans.ParticipantId = this.Guid;

// UDPTransmitters.Add(trans);

//}

worker = new WriterWorker(this.PeriodicWork);

}

protected void StartTransmitters()

{

foreach (var trans in UDPTransmitters)

{

trans.Start();

}

worker.Start((int)this.ResendDataPeriod.AsMillis());

}

protected virtual void PeriodicWork()

{

// the RTPS Writer to repeatedly announce the availability of data by sending a Heartbeat Message.

log.DebugFormat("I have to send a Heartbeat Message, at {0}", DateTime.Now);

SendHeartbeat();

if (HistoryCache.Changes.Count > 0)

{

foreach (var change in HistoryCache.Changes)

{

//SendHeartbeat();

//SendData(change);

SendDataHeartbeat(change);

}

HistoryCache.Changes.Clear(); //TODO

}

}

private void SendHeartbeat()

{

// Create a Message with Heartbeat

Message m1 = new Message();

Heartbeat heartbeat = new Heartbeat();

EntityId id1 = EntityId.ENTITYID\_UNKNOWN;

EntityId id2 = EntityId.ENTITYID\_PARTICIPANT;

heartbeat.readerId = id1;

heartbeat.writerId = id2;

heartbeat.firstSN = new SequenceNumber(10);

heartbeat.lastSN = new SequenceNumber(20);

heartbeat.count = 5;

m1.SubMessages.Add(heartbeat);

SendData(m1);

}

public void SendData(CacheChange<T> change)

{

// Create a Message with InfoSource

Message msg = new Message();

EntityId readerId = EntityId.ENTITYID\_UNKNOWN;

EntityId writerId = change.WriterGuid.EntityId;

SerializedPayload payload = new SerializedPayload();

IoBuffer buff = IoBuffer.Allocate(1024);

payload.DataEncapsulation = EncapsulationManager.Serialize<T>((T)change.DataValue.Value, Scheme);

Data data = new Data(readerId, writerId, change.SequenceNumber.LongValue, null, payload);

msg.SubMessages.Add(data);

// Write Message to bytes1 array

SendData(msg);

}

public void SendDataHeartbeat(CacheChange<T> change)

{

// Create a Message with InfoSource

Message msg = new Message();

EntityId readerId = EntityId.ENTITYID\_UNKNOWN;

EntityId writerId = change.WriterGuid.EntityId;

SerializedPayload payload = new SerializedPayload();

IoBuffer buff = IoBuffer.Allocate(1024);

payload.DataEncapsulation = buff.EncapsuleCDRData(change.DataValue.Value, BitConverter.IsLittleEndian ? ByteOrder.LittleEndian : ByteOrder.BigEndian);

Data data = new Data(readerId, writerId, change.SequenceNumber.LongValue, null, payload);

msg.SubMessages.Add(data);

Heartbeat heartbeat = new Heartbeat();

heartbeat.readerId = readerId;

heartbeat.writerId = writerId;

heartbeat.firstSN = change.SequenceNumber;

heartbeat.lastSN = change.SequenceNumber;

heartbeat.count = 1;

msg.SubMessages.Add(heartbeat);

SendData(msg);

}

/// <summary>

/// Writes a message to network

/// </summary>

/// <param name="msg"></param>

/// <returns></returns>

private void SendData(Message msg)

{

foreach (var trans in UDPTransmitters)

trans.SendMessage(msg);

}

public void Dispose()

{

worker.End();

foreach (var trans in UDPTransmitters)

{

trans.Close();

trans.Dispose();

}

UDPTransmitters.Clear();

}

}

}

#### WriterWorker.cs

using log4net;

using Doopec.Rtps.Utils;

using Doopec.Utils.Transport;

using Mina.Core.Buffer;

using Rtps.Messages;

using Rtps.Messages.Submessages;

using Rtps.Messages.Submessages.Elements;

using Rtps.Structure;

using Rtps.Structure.Types;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

namespace Doopec.Rtps.Behavior

{

public class WriterWorker : PeriodicWorker

{

private static readonly ILog log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

public delegate void PeriodicWorkDelegate();

private PeriodicWorkDelegate periodicWork;

public WriterWorker( )

{

}

public override void End()

{

}

public WriterWorker(PeriodicWorkDelegate periodicWork)

{

this.periodicWork = periodicWork;

}

public override void DoPeriodicWork()

{

base.DoPeriodicWork();

if (periodicWork != null)

periodicWork();

}

}

}

### RtpsTransport

#### RtpsDiscovery.cs

using Doopec.Rtps.SharedMem;

using log4net;

using Rtps.Structure;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

namespace Doopec.Rtps.RtpsTransport

{

public class RtpsDiscovery : IRtpsDiscovery

{

private static readonly ILog log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

private List<Participant> participants = new List<Participant>();

private List<Endpoint> endpoints = new List<Endpoint>();

public event DiscoveryEventHandler ParticipantDiscovery;

public event DiscoveryEventHandler EndpointDiscovery;

public IList<Participant> Participants

{

get { return participants.AsReadOnly(); }

}

public IList<Endpoint> Endpoints

{

get { return endpoints.AsReadOnly(); }

}

public void RegisterParticipant(Participant participant)

{

if (participant != null)

{

participants.Add(participant);

DiscoveryEventArgs dea = new DiscoveryEventArgs();

dea.Reason = EventReason.NEW\_PARTICIPANT;

dea.EventData = participant;

NotifyParticipantChanges(dea);

}

}

public void UnregisterParticipant(Participant participant)

{

if (participant != null)

{

participants.Remove(participant);

DiscoveryEventArgs dea = new DiscoveryEventArgs();

dea.Reason = EventReason.DELETED\_PARTICIPANT;

dea.EventData = participant;

NotifyParticipantChanges(dea);

}

}

public void RegisterEndpoint(Endpoint endpoint)

{

if (endpoint != null)

{

endpoints.Add(endpoint);

DiscoveryEventArgs dea = new DiscoveryEventArgs();

dea.Reason = EventReason.NEW\_ENDPOINT;

dea.EventData = endpoint;

NotifyEndpointsChanges(dea);

}

}

public void UnregisterEndpoint(Endpoint endpoint)

{

if (endpoint != null)

{

endpoints.Remove(endpoint);

DiscoveryEventArgs dea = new DiscoveryEventArgs();

dea.Reason = EventReason.DELETED\_ENDPOINT;

dea.EventData = endpoint;

NotifyEndpointsChanges(dea);

}

}

private void NotifyParticipantChanges(DiscoveryEventArgs dea)

{

log.Debug("The information about Participants has changed");

if (ParticipantDiscovery != null)

{

ParticipantDiscovery(this, dea);

}

}

private void NotifyEndpointsChanges(DiscoveryEventArgs dea)

{

log.Debug("The information about Endpoints has changed");

if (EndpointDiscovery != null)

{

EndpointDiscovery(this, dea);

}

}

}

}

#### RtpsEngine.cs

using Doopec.Configuration;

using System;

using System.Collections.Generic;

using System.Configuration;

namespace Doopec.Rtps.RtpsTransport

{

public static class RtpsEngineFactory

{

private static IRtpsEngine theInstance;

public static IRtpsEngine Instance

{

get

{

if (theInstance == null)

{ theInstance = RtpsEngineFactory.CreateEngine(null); }

return theInstance;

}

}

public static IRtpsEngine CreateEngine(IDictionary<string, Object> environment)

{

DDSConfigurationSection ddsConfig = Doopec.Configuration.DDSConfigurationSection.Instance;

RTPSConfigurationSection rtpsConfig = Doopec.Configuration.RTPSConfigurationSection.Instance;

string transportProfile = ddsConfig.Domains[0].TransportProfile.Name;

string className = rtpsConfig.Transports[transportProfile].Type;

if (string.IsNullOrWhiteSpace(className))

{

// no implementation class name specified

throw new ApplicationException("Please Set the RTPS engine type property in the settings.");

}

Type ctxClass = Type.GetType(className, true);

// --- Instantiate new object --- //

try

{

// First, try a constructor that will accept the environment.

object newInstance = Activator.CreateInstance(ctxClass, environment);

if (newInstance != null)

return (IRtpsEngine)newInstance;

}

catch (Exception)

{

/\* No Map constructor found; try a no-argument constructor

\* instead.

\*

\* Get the constructor and call it explicitly rather than

\* calling Class.newInstance(). The latter propagates all

\* exceptions, even checked ones, complicating error handling

\* for us and the user.

\*/

object newInstance = Activator.CreateInstance(ctxClass);

return (IRtpsEngine)newInstance;

}

throw new ApplicationException("Exception building a RTPS engine using " + className);

}

}

public class RtpsEngine : IRtpsEngine

{

protected RtpsDiscovery discoveryModule = new RtpsDiscovery();

public IRtpsDiscovery DiscoveryModule

{

get { return discoveryModule; }

}

}

}

### Encoders

#### AckNackEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Doopec.Rtps.Encoders

{

public static class AckNackEncoder

{

public static void PutAckNack(this IoBuffer buffer, AckNack obj)

{

buffer.PutEntityId(obj.ReaderId);

buffer.PutEntityId(obj.WriterId);

buffer.PutSequenceNumberSet(obj.ReaderSNState);

buffer.PutInt32(obj.Count);

}

public static AckNack GetAckNack(this IoBuffer buffer)

{

AckNack obj = new AckNack();

buffer.GetAckNack(ref obj);

return obj;

}

public static void GetAckNack(this IoBuffer buffer, ref AckNack obj)

{

obj.ReaderId = buffer.GetEntityId();

obj.WriterId = buffer.GetEntityId();

obj.ReaderSNState = buffer.GetSequenceNumberSet();

obj.Count = buffer.GetInt32();

}

}

}

#### DataFragEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages;

using System;

namespace Doopec.Rtps.Encoders

{

public static class DataFragEncoder

{

public static void PutDataFrag(this IoBuffer buffer, DataFrag obj)

{

buffer.PutInt16(obj.ExtraFlags);

short octets\_to\_inline\_qos = 4 + 4 + 8 + 4 + 2 + 2 + 4;

buffer.PutInt16(octets\_to\_inline\_qos);

buffer.PutEntityId(obj.ReaderId);

buffer.PutEntityId(obj.WriterId);

buffer.PutSequenceNumber(obj.WriterSequenceNumber);

buffer.PutInt32(obj.FragmentStartingNumber);

buffer.PutInt16((short)obj.FragmentsInSubmessage);

buffer.PutInt16((short)obj.FragmentSize);

buffer.PutInt32(obj.SampleSize);

if (obj.HasInlineQosFlag)

{

buffer.PutParameterList(obj.ParameterList);

}

buffer.Put(obj.SerializedPayload); // TODO: check this

}

public static DataFrag GetDataFrag(this IoBuffer buffer)

{

DataFrag obj = new DataFrag();

buffer.GetDataFrag(ref obj);

return obj;

}

public static void GetDataFrag(this IoBuffer buffer, ref DataFrag obj)

{

int start\_count = buffer.Position; // start of bytes Read so far from the

// beginning

obj.ExtraFlags = (short)buffer.GetInt16();

int octetsToInlineQos = buffer.GetInt16() & 0xffff;

int currentCount = buffer.Position; // count bytes to inline qos

obj.ReaderId = buffer.GetEntityId();

obj.WriterId = buffer.GetEntityId();

obj.WriterSequenceNumber = buffer.GetSequenceNumber();

obj.FragmentStartingNumber = buffer.GetInt32(); // ulong

obj.FragmentsInSubmessage = buffer.GetInt16(); // ushort

obj.FragmentSize = buffer.GetInt16(); // ushort

obj.SampleSize = buffer.GetInt32(); // ulong

int bytesRead = buffer.Position - currentCount;

int unknownOctets = octetsToInlineQos - bytesRead;

for (int i = 0; i < unknownOctets; i++)

{

buffer.Get(); // Skip unknown octets, @see 9.4.5.3.3 octetsToInlineQos

}

if (obj.HasInlineQosFlag)

{

obj.ParameterList = buffer.GetParameterList();

}

int end\_count = buffer.Position; // end of bytes Read so far from the beginning

if (obj.Header.SubMessageLength != 0)

{

obj.SerializedPayload = new byte[obj.Header.SubMessageLength - (end\_count - start\_count)];

}

else

{ // SubMessage is the last one. Rest of the bytes are Read.

// @see 8.3.3.2.3

obj.SerializedPayload = new byte[buffer.Remaining];

}

buffer.Get(obj.SerializedPayload, 0, obj.SerializedPayload.Length);

}

}

}

#### DataSubMessageEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages;

using Rtps.Messages.Submessages;

using Rtps.Messages.Submessages.Elements;

using System;

using Doopec.Utils.Network.Encoders;

using Doopec.Rtps.Messages;

namespace Doopec.Rtps.Encoders

{

public static class DataSubMessageEncoder

{

public static void PutDataSubMessage(this IoBuffer buffer, Data obj)

{

buffer.PutInt16(obj.ExtraFlags.Value);

short octetsToInlineQos = 0;

if (obj.HasInlineQosFlag)

{

octetsToInlineQos = 4 + 4 + 8;// EntityId.LENGTH + EntityId.LENGTH + SequenceNumber.LENGTH;

}

buffer.PutInt16(octetsToInlineQos);

buffer.PutEntityId(obj.ReaderId);

buffer.PutEntityId(obj.WriterId);

buffer.PutSequenceNumber(obj.WriterSN);

if (obj.HasInlineQosFlag)

{

buffer.PutParameterList(obj.InlineQos);

}

if (obj.HasDataFlag || obj.HasKeyFlag)

{

buffer.Align(4);

buffer.Put(obj.SerializedPayload.DataEncapsulation.SerializedPayload);

}

}

public static Data GetDataSubMessage(this IoBuffer buffer)

{

Data obj = new Data();

buffer.GetDataSubMessage(ref obj);

return obj;

}

public static void GetDataSubMessage(this IoBuffer buffer, ref Data obj)

{

if (obj.HasDataFlag && obj.HasKeyFlag)

{

// Should we just ignore this message instead

throw new ApplicationException(

"This version of protocol does not allow Data submessage to contain both serialized data and serialized key (9.4.5.3.1)");

}

int start\_count = buffer.Position; // start of bytes Read so far from the

// beginning

Flags flgs = new Flags();

flgs.Value = (byte)buffer.GetInt16();

obj.ExtraFlags = flgs;

int octetsToInlineQos = buffer.GetInt16() & 0xffff;

int currentCount = buffer.Position; // count bytes to inline qos

obj.ReaderId = buffer.GetEntityId();

obj.WriterId = buffer.GetEntityId();

obj.WriterSN = buffer.GetSequenceNumber();

int bytesRead = buffer.Position - currentCount;

int unknownOctets = octetsToInlineQos - bytesRead;

for (int i = 0; i < unknownOctets; i++)

{

// TODO: Instead of looping, we should do just

// newPos = bb.getBuffer.position() + unknownOctets or something

// like that

buffer.Get(); // Skip unknown octets, @see 9.4.5.3.3

// octetsToInlineQos

}

if (obj.HasInlineQosFlag)

{

obj.InlineQos = buffer.GetParameterList();

}

if (obj.HasDataFlag || obj.HasKeyFlag)

{

buffer.Align(4); // Each submessage is aligned on 32-bit boundary, @see

// 9.4.1 Overall Structure

int end\_count = buffer.Position; // end of bytes Read so far from the beginning

int length;

if (obj.Header.SubMessageLength != 0)

{

length = obj.Header.SubMessageLength - (end\_count - start\_count);

}

else

{

// SubMessage is the last one. Rest of the bytes are Read.

// @see 8.3.3.2.3

length = buffer.Remaining;

}

obj.SerializedPayload = new SerializedPayload();

obj.SerializedPayload.DataEncapsulation = EncapsulationManager.Deserialize(buffer, length);

}

}

}

}

#### EncapsulationSchemeEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages;

using Rtps.Messages.Submessages.Elements;

using Rtps.Messages.Types;

using Rtps.Structure.Types;

using System;

namespace Doopec.Rtps.Encoders

{

public static class EncapsulationSchemeEncoder

{

public static void PutEncapsulationScheme(this IoBuffer buffer, EncapsulationScheme msg)

{

buffer.Put(msg.B0);

buffer.Put(msg.B1);

buffer.Put(msg.B2);

buffer.Put(msg.B3);

}

public static EncapsulationScheme GetEncapsulationScheme(this IoBuffer buffer)

{

EncapsulationScheme obj = new EncapsulationScheme();

obj.B0 = buffer.Get();

obj.B1 = buffer.Get();

obj.B2 = buffer.Get();

obj.B3 = buffer.Get();

return obj;

}

public static void GetEncapsulationScheme(this IoBuffer buffer, ref EncapsulationScheme obj)

{

obj.B0 = buffer.Get();

obj.B1 = buffer.Get();

obj.B2 = buffer.Get();

obj.B3 = buffer.Get();

}

}

}

#### EntityIdEncoder.cs

using Doopec.Serializer;

using Mina.Core.Buffer;

using Rtps.Structure.Types;

using System.Reflection;

namespace Doopec.Rtps.Encoders

{

public static class EntityIdEncoder

{

public static void PutEntityId(this IoBuffer buffer, EntityId obj)

{

buffer.Put(obj.EntityKey);

buffer.Put((byte)obj.TypeID);

}

public static void WriteEntityId(IoBuffer buffer, EntityId obj)

{

buffer.Put(obj.EntityKey);

buffer.Put((byte)obj.TypeID);

}

public static EntityId GetEntityId(this IoBuffer buffer)

{

EntityId obj = new EntityId();

buffer.GetEntityId(ref obj);

return obj;

}

public static void GetEntityId(this IoBuffer buffer, ref EntityId obj)

{

buffer.Get(obj.EntityKey, 0, 3);

obj.TypeID = (EntityKinds)buffer.Get();

}

public static void ReadEntityId(IoBuffer buffer, ref EntityId obj)

{

if (obj == null)

obj = new EntityId();

buffer.Get(obj.EntityKey, 0, 3);

obj.TypeID = (EntityKinds)buffer.Get();

}

}

public class EntityIdSerializer : IStaticTypeSerializer

{

delegate void WriterDelegate(IoBuffer buffer, EntityId obj);

delegate void ReaderDelegate(IoBuffer buffer, ref EntityId obj);

public void GetStaticMethods(System.Type type, out MethodInfo writer, out MethodInfo reader)

{

WriterDelegate writerDelegate = EntityIdEncoder.WriteEntityId;

ReaderDelegate readerDelegate = EntityIdEncoder.ReadEntityId;

writer = writerDelegate.Method;

reader = readerDelegate.Method;

}

public bool Handles(System.Type type)

{

return type == typeof(EntityId);

}

public System.Collections.Generic.IEnumerable<System.Type> GetSubtypes(System.Type type)

{

yield break;

}

}

}

#### GapEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages;

namespace Doopec.Rtps.Encoders

{

public static class GapEncoder

{

public static void PutGap(this IoBuffer buffer, Gap obj)

{

buffer.PutEntityId(obj.ReaderId);

buffer.PutEntityId(obj.WriterId);

buffer.PutSequenceNumber(obj.GapStart);

buffer.PutSequenceNumberSet(obj.GapList);

}

public static Gap GetGap(this IoBuffer buffer)

{

Gap obj = new Gap();

buffer.GetGap(ref obj);

return obj;

}

public static void GetGap(this IoBuffer buffer, ref Gap obj)

{

obj.readerId = buffer.GetEntityId();

obj.writerId = buffer.GetEntityId();

obj.gapStart = buffer.GetSequenceNumber();

obj.gapList = buffer.GetSequenceNumberSet();

}

}

}

#### GuidEncoder.cs

using Doopec.Serializer;

using Mina.Core.Buffer;

using Rtps.Structure.Types;

using System.Reflection;

namespace Doopec.Rtps.Encoders

{

public static class GUIDEncoder

{

public static void PutGUID(this IoBuffer buffer, GUID obj)

{

buffer.PutGuidPrefix(obj.Prefix);

buffer.PutEntityId(obj.EntityId);

}

public static void WriteGUID(IoBuffer buffer, GUID obj)

{

buffer.PutGuidPrefix(obj.Prefix);

buffer.PutEntityId(obj.EntityId);

}

public static GUID GetGUID(this IoBuffer buffer)

{

GUID obj = new GUID();

obj.Prefix = buffer.GetGuidPrefix();

obj.EntityId = buffer.GetEntityId();

return obj;

}

public static void GetGUID(this IoBuffer buffer, ref GUID obj)

{

obj.Prefix = buffer.GetGuidPrefix();

obj.EntityId = buffer.GetEntityId();

}

public static void ReadGUID(IoBuffer buffer, ref GUID obj)

{

if (obj == null)

obj = new GUID();

obj.Prefix = buffer.GetGuidPrefix();

obj.EntityId = buffer.GetEntityId();

}

}

public class GUIDSerializer : IStaticTypeSerializer

{

delegate void WriterDelegate(IoBuffer buffer, GUID obj);

delegate void ReaderDelegate(IoBuffer buffer, ref GUID obj);

public void GetStaticMethods(System.Type type, out MethodInfo writer, out MethodInfo reader)

{

WriterDelegate writerDelegate = GUIDEncoder.WriteGUID;

ReaderDelegate readerDelegate = GUIDEncoder.ReadGUID;

writer = writerDelegate.Method;

reader = readerDelegate.Method;

}

public bool Handles(System.Type type)

{

return type == typeof(GUID);

}

public System.Collections.Generic.IEnumerable<System.Type> GetSubtypes(System.Type type)

{

yield break;

}

}

}

#### GuidPrefixEncoder.cs

using Doopec.Serializer;

using Mina.Core.Buffer;

using Rtps.Structure.Types;

using System.Reflection;

namespace Doopec.Rtps.Encoders

{

public static class GuidPrefixEncoder

{

public static void PutGuidPrefix(this IoBuffer buffer, GuidPrefix obj)

{

buffer.Put(obj.Prefix);

}

public static void WriteGuidPrefix(IoBuffer buffer, GuidPrefix obj)

{

buffer.Put(obj.Prefix);

}

public static GuidPrefix GetGuidPrefix(this IoBuffer buffer)

{

GuidPrefix obj = new GuidPrefix();

buffer.GetGuidPrefix(ref obj);

return obj;

}

public static void GetGuidPrefix(this IoBuffer buffer, ref GuidPrefix obj)

{

buffer.Get(obj.Prefix, 0, GuidPrefix.GUID\_PREFIX\_SIZE);

}

public static void ReadGuidPrefix(IoBuffer buffer, ref GuidPrefix obj)

{

buffer.Get(obj.Prefix, 0, GuidPrefix.GUID\_PREFIX\_SIZE);

}

}

public class GuidPrefixSerializer : IStaticTypeSerializer

{

delegate void WriterDelegate(IoBuffer buffer, GuidPrefix obj);

delegate void ReaderDelegate(IoBuffer buffer, ref GuidPrefix obj);

public void GetStaticMethods(System.Type type, out MethodInfo writer, out MethodInfo reader)

{

WriterDelegate writerDelegate = GuidPrefixEncoder.WriteGuidPrefix;

ReaderDelegate readerDelegate = GuidPrefixEncoder.ReadGuidPrefix;

writer = writerDelegate.Method;

reader = readerDelegate.Method;

}

public bool Handles(System.Type type)

{

return type == typeof(GuidPrefix);

}

public System.Collections.Generic.IEnumerable<System.Type> GetSubtypes(System.Type type)

{

yield break;

}

}

}

#### HeaderEncoder.cs

using Doopec.Serializer;

using Mina.Core.Buffer;

using Rtps.Structure.Types;

using System.Reflection;

namespace Doopec.Rtps.Encoders

{

public static class GuidPrefixEncoder

{

public static void PutGuidPrefix(this IoBuffer buffer, GuidPrefix obj)

{

buffer.Put(obj.Prefix);

}

public static void WriteGuidPrefix(IoBuffer buffer, GuidPrefix obj)

{

buffer.Put(obj.Prefix);

}

public static GuidPrefix GetGuidPrefix(this IoBuffer buffer)

{

GuidPrefix obj = new GuidPrefix();

buffer.GetGuidPrefix(ref obj);

return obj;

}

public static void GetGuidPrefix(this IoBuffer buffer, ref GuidPrefix obj)

{

buffer.Get(obj.Prefix, 0, GuidPrefix.GUID\_PREFIX\_SIZE);

}

public static void ReadGuidPrefix(IoBuffer buffer, ref GuidPrefix obj)

{

buffer.Get(obj.Prefix, 0, GuidPrefix.GUID\_PREFIX\_SIZE);

}

}

public class GuidPrefixSerializer : IStaticTypeSerializer

{

delegate void WriterDelegate(IoBuffer buffer, GuidPrefix obj);

delegate void ReaderDelegate(IoBuffer buffer, ref GuidPrefix obj);

public void GetStaticMethods(System.Type type, out MethodInfo writer, out MethodInfo reader)

{

WriterDelegate writerDelegate = GuidPrefixEncoder.WriteGuidPrefix;

ReaderDelegate readerDelegate = GuidPrefixEncoder.ReadGuidPrefix;

writer = writerDelegate.Method;

reader = readerDelegate.Method;

}

public bool Handles(System.Type type)

{

return type == typeof(GuidPrefix);

}

public System.Collections.Generic.IEnumerable<System.Type> GetSubtypes(System.Type type)

{

yield break;

}

}

}

#### HeartbeatEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages;

namespace Doopec.Rtps.Encoders

{

public static class HeartbeatEncoder

{

public static void PutHeartbeat(this IoBuffer buffer, Heartbeat obj)

{

buffer.PutEntityId(obj.readerId);

buffer.PutEntityId(obj.writerId);

buffer.PutSequenceNumber(obj.firstSN);

buffer.PutSequenceNumber(obj.lastSN);

buffer.PutInt32(obj.count);

}

public static Heartbeat GetHeartbeat(this IoBuffer buffer)

{

Heartbeat obj = new Heartbeat();

buffer.GetHeartbeat(ref obj);

return obj;

}

public static void GetHeartbeat(this IoBuffer buffer, ref Heartbeat obj)

{

obj.readerId = buffer.GetEntityId();

obj.writerId = buffer.GetEntityId();

obj.firstSN = buffer.GetSequenceNumber();

obj.lastSN = buffer.GetSequenceNumber();

obj.count = buffer.GetInt32();

}

}

}

#### HeartbeatFragEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages;

namespace Doopec.Rtps.Encoders

{

public static class HeartbeatFragEncoder

{

public static void PutHeartbeatFrag(this IoBuffer buffer, HeartbeatFrag obj)

{

buffer.PutEntityId(obj.ReaderId);

buffer.PutEntityId(obj.WriterId);

buffer.PutSequenceNumber(obj.WriterSequenceNumber);

buffer.PutInt32(obj.LastFragmentNumber);

buffer.PutInt32(obj.Count);

}

public static HeartbeatFrag GetHeartbeatFrag(this IoBuffer buffer)

{

HeartbeatFrag obj = new HeartbeatFrag();

buffer.GetHeartbeatFrag(ref obj);

return obj;

}

public static void GetHeartbeatFrag(this IoBuffer buffer, ref HeartbeatFrag obj)

{

obj.ReaderId = buffer.GetEntityId();

obj.WriterId = buffer.GetEntityId();

obj.WriterSequenceNumber = buffer.GetSequenceNumber();

obj.LastFragmentNumber = buffer.GetInt32();

obj.Count = buffer.GetInt32();

}

}

}

#### InfoDestinationEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages;

namespace Doopec.Rtps.Encoders

{

public static class InfoDestinationEncoder

{

public static void PutInfoDestination(this IoBuffer buffer, InfoDestination msg)

{

buffer.PutGuidPrefix(msg.GuidPrefix);

}

public static InfoDestination GetInfoDestination(this IoBuffer buffer)

{

InfoDestination obj = new InfoDestination();

buffer.GetInfoDestination(ref obj);

return obj;

}

public static void GetInfoDestination(this IoBuffer buffer, ref InfoDestination obj)

{

obj.GuidPrefix = buffer.GetGuidPrefix();

}

}

}

#### InfoReplyEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages;

namespace Doopec.Rtps.Encoders

{

public static class InfoDestinationEncoder

{

public static void PutInfoDestination(this IoBuffer buffer, InfoDestination msg)

{

buffer.PutGuidPrefix(msg.GuidPrefix);

}

public static InfoDestination GetInfoDestination(this IoBuffer buffer)

{

InfoDestination obj = new InfoDestination();

buffer.GetInfoDestination(ref obj);

return obj;

}

public static void GetInfoDestination(this IoBuffer buffer, ref InfoDestination obj)

{

obj.GuidPrefix = buffer.GetGuidPrefix();

}

}

}

#### InfoReplyIp4Encoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Doopec.Rtps.Encoders

{

public static class InfoReplyIp4Encoder

{

public static void PutInfoReplyIp4(this IoBuffer buffer, InfoReplyIp4 obj)

{

buffer.PutLocatorUDPv4(obj.UnicastLocator);

buffer.PutLocatorUDPv4(obj.MulticastLocator);

}

public static InfoReplyIp4 GetInfoReplyIp4(this IoBuffer buffer)

{

InfoReplyIp4 obj = new InfoReplyIp4();

buffer.GetInfoReplyIp4(ref obj);

return obj;

}

public static void GetInfoReplyIp4(this IoBuffer buffer, ref InfoReplyIp4 obj)

{

obj.UnicastLocator = buffer.GetLocatorUDPv4();

obj.MulticastLocator = buffer.GetLocatorUDPv4();

}

}

}

#### InfoSourceEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages;

namespace Doopec.Rtps.Encoders

{

public static class InfoSourceEncoder

{

public static void PutInfoSource(this IoBuffer buffer, InfoSource obj)

{

buffer.PutInt64(0);

buffer.PutProtocolVersion(obj.ProtocolVersion);

buffer.PutVendorId(obj.VendorId);

buffer.PutGuidPrefix(obj.GuidPrefix);

}

public static InfoSource GetInfoSource(this IoBuffer buffer)

{

InfoSource obj = new InfoSource();

buffer.GetInfoSource(ref obj);

return obj;

}

public static void GetInfoSource(this IoBuffer buffer, ref InfoSource obj)

{

buffer.GetInt64(); // unused

obj.ProtocolVersion = buffer.GetProtocolVersion();

obj.VendorId = buffer.GetVendorId();

obj.GuidPrefix = buffer.GetGuidPrefix();

}

}

}

#### InfoTimestampEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages;

namespace Doopec.Rtps.Encoders

{

public static class InfoTimestampEncoder

{

public static void PutInfoTimestamp(this IoBuffer buffer, InfoTimestamp obj)

{

if (!obj.HasInvalidateFlag)

{

buffer.PutTime(obj.TimeStamp);

}

}

public static InfoTimestamp GetInfoTimestamp(this IoBuffer buffer)

{

InfoTimestamp obj = new InfoTimestamp();

buffer.GetInfoTimestamp(ref obj);

return obj;

}

public static void GetInfoTimestamp(this IoBuffer buffer, ref InfoTimestamp obj)

{

if (!obj.HasInvalidateFlag)

{

obj.TimeStamp = buffer.GetTime();

}

}

}

}

#### LocatorEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages;

namespace Doopec.Rtps.Encoders

{

public static class InfoTimestampEncoder

{

public static void PutInfoTimestamp(this IoBuffer buffer, InfoTimestamp obj)

{

if (!obj.HasInvalidateFlag)

{

buffer.PutTime(obj.TimeStamp);

}

}

public static InfoTimestamp GetInfoTimestamp(this IoBuffer buffer)

{

InfoTimestamp obj = new InfoTimestamp();

buffer.GetInfoTimestamp(ref obj);

return obj;

}

public static void GetInfoTimestamp(this IoBuffer buffer, ref InfoTimestamp obj)

{

if (!obj.HasInvalidateFlag)

{

obj.TimeStamp = buffer.GetTime();

}

}

}

}

#### LocatorUDPv4Encoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages.Elements;

namespace Doopec.Rtps.Encoders

{

public static class LocatorUDPv4Encoder

{

public static void PutLocatorUDPv4(this IoBuffer buffer, LocatorUDPv4 obj)

{

buffer.PutInt32(obj.Port);

buffer.PutInt32(obj.Address);

}

public static LocatorUDPv4 GetLocatorUDPv4(this IoBuffer buffer)

{

LocatorUDPv4 obj = new LocatorUDPv4();

buffer.GetLocatorUDPv4(ref obj);

return obj;

}

public static void GetLocatorUDPv4(this IoBuffer buffer, ref LocatorUDPv4 obj)

{

obj.Port = buffer.GetInt32();

obj.Address = buffer.GetInt32();

}

}

}

#### MessageEncoder.cs

using Mina.Core.Buffer;

using Mina.Core.Session;

using Mina.Filter.Codec;

using Rtps.Messages;

using System;

namespace Doopec.Rtps.Encoders

{

public class MessageEncoder : IProtocolEncoder

{

public void Encode(IoSession session, object message, IProtocolEncoderOutput output)

{

Message msg = (Message)message;

IoBuffer buffer = IoBuffer.Allocate(1024);

buffer.AutoExpand = true;

buffer.PutMessage(msg);

buffer.Flip();

output.Write(buffer);

}

public void Dispose(IoSession session)

{

// nothing to Dispose

}

}

public class MessageDecoder : CumulativeProtocolDecoder

{

protected override Boolean DoDecode(IoSession session, IoBuffer input, IProtocolDecoderOutput output)

{

if (input.Remaining >= 4)

{

Message request = input.GetMessage();

output.Write(request);

return true;

}

else

{

return false;

}

}

}

public class MessageCodecFactory : IProtocolCodecFactory

{

public static MessageEncoder encoder = new MessageEncoder();

public static MessageDecoder decoder = new MessageDecoder();

public IProtocolEncoder GetEncoder(IoSession session)

{

return encoder;

}

public IProtocolDecoder GetDecoder(IoSession session)

{

return decoder;

}

}

}

#### MessageStaticEncoder

using Mina.Core.Buffer;

using Rtps.Messages;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Doopec.Rtps.Encoders

{

public static class MessageStaticEncoder

{

public static void PutMessage(this IoBuffer buffer, Message msg)

{

buffer.PutHeader(msg.Header);

int position = 0;

int subMessageCount = 0;

foreach (SubMessage submsg in msg.SubMessages)

{

int subMsgStartPosition = buffer.Position;

position = buffer.PutSubMessage(submsg);

subMessageCount++;

}

// Length of last submessage is 0, @see 8.3.3.2.3 submessageLength

if (subMessageCount > 0)

buffer.PutInt16(position - 2, (short)0);

}

public static Message GetMessage(this IoBuffer buffer)

{

Message obj = new Message();

buffer.GetMessage(ref obj);

return obj;

}

public static void GetMessage(this IoBuffer buffer, ref Message obj)

{

ByteOrder byteOrder = buffer.Order;

buffer.Order = ByteOrder.LittleEndian;

obj.Header = buffer.GetHeader();

while (buffer.HasRemaining)

{

SubMessage submsg = buffer.GetSubMessage();

obj.SubMessages.Add(submsg);

}

buffer.Order = byteOrder;

}

}

}

#### NackFragEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages;

namespace Doopec.Rtps.Encoders

{

public static class NackFragEncoder

{

public static void PutNackFrag(this IoBuffer buffer, NackFrag obj)

{

buffer.PutEntityId(obj.ReaderId);

buffer.PutEntityId(obj.WriterId);

buffer.PutSequenceNumber(obj.WriterSequenceNumber);

buffer.PutSequenceNumberSet(obj.FragmentNumberState);

buffer.PutInt32(obj.Count);

}

public static NackFrag GetNackFrag(this IoBuffer buffer)

{

NackFrag obj = new NackFrag();

buffer.GetNackFrag(ref obj);

return obj;

}

public static void GetNackFrag(this IoBuffer buffer, ref NackFrag obj)

{

obj.ReaderId = buffer.GetEntityId();

obj.WriterId = buffer.GetEntityId();

obj.WriterSequenceNumber = buffer.GetSequenceNumber();

obj.FragmentNumberState = buffer.GetSequenceNumberSet();

obj.Count = buffer.GetInt32();

}

}

}

#### PadEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages;

namespace Doopec.Rtps.Encoders

{

public static class PadEncoder

{

public static void PutPad(this IoBuffer buffer, Pad obj)

{

buffer.Put(obj.Bytes);

}

public static Pad GetPad(this IoBuffer buffer)

{

Pad obj = new Pad();

buffer.GetPad(ref obj);

return obj;

}

public static void GetPad(this IoBuffer buffer, ref Pad obj)

{

obj.Bytes = new byte[buffer.Remaining];

buffer.Get(obj.Bytes, 0, obj.Bytes.Length);

}

}

}

#### ParameterEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages.Elements;

using Rtps.Messages.Types;

using Doopec.Utils.Network.Encoders;

namespace Doopec.Rtps.Encoders

{

public static class ParameterEncoder

{

public static void PutParameter(this IoBuffer buffer, Parameter obj)

{

buffer.PutInt16((short)obj.ParameterId);

buffer.PutInt16(0); // length will be calculated

int pos = buffer.Position;

buffer.Put(obj.Bytes);

buffer.Align(4); // Make sure length is multiple of 4 & align for

// next param

int paramLength = buffer.Position - pos;

buffer.PutInt16(pos - 2, (short)paramLength);

}

public static Parameter GetParameter(this IoBuffer buffer)

{

Parameter obj = new Parameter();

buffer.GetParameter(ref obj);

return obj;

}

public static void GetParameter(this IoBuffer buffer, ref Parameter obj)

{

obj.ParameterId = (ParameterId)buffer.GetInt16();

int length = buffer.GetInt16();

obj.Bytes = new byte[length];

buffer.Get(obj.Bytes, 0, length);

}

}

}

#### ParameterListEncoder.cs

using log4net;

using Mina.Core.Buffer;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

using Rtps.Messages.Submessages.Elements;

using Rtps.Messages.Types;

using Doopec.Encoders.RTPS;

using Doopec.Utils.Network.Encoders;

namespace Doopec.Rtps.Encoders

{

public static class ParameterListEncoder

{

private static readonly ILog log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

public static void PutParameterList(this IoBuffer buffer, ParameterList obj)

{

buffer.Align(4); // @see 9.4.2.11

obj.Value.Add(Sentinel.Instance); // Sentinel must be the last Parameter

foreach (Parameter param in obj.Value)

{

buffer.PutParameter(param);

}

}

public static ParameterList GetParameterList(this IoBuffer buffer)

{

ParameterList obj = new ParameterList();

buffer.GetParameterList(ref obj);

return obj;

}

public static void GetParameterList(this IoBuffer buffer, ref ParameterList obj)

{

log.Debug("Reading ParameterList from buffer");

while (true)

{

int pos1 = buffer.Position;

Parameter param = buffer.GetParameter();

obj.Value.Add(param);

int length = buffer.Position - pos1;

//log.DebugFormat("Read Parameter {0}, length {1} from position {2}", param, length, pos1);

if (param.ParameterId == ParameterId.PID\_SENTINEL)

{

break;

}

}

}

}

}

#### ProtocolIdEncoder.cs

using Doopec.Serializer;

using Mina.Core.Buffer;

using Rtps.Messages.Types;

using System.Reflection;

namespace Doopec.Rtps.Encoders

{

public static class ProtocolIdEncoder

{

public static void PutProtocolId(this IoBuffer buffer, ProtocolId obj)

{

buffer.Put(obj.Id);

}

public static void WriteProtocolId(IoBuffer buffer, ProtocolId obj)

{

buffer.Put(obj.Id);

}

public static ProtocolId GetProtocolId(this IoBuffer buffer)

{

ProtocolId obj = new ProtocolId();

buffer.GetProtocolId(ref obj);

return obj;

}

public static void GetProtocolId(this IoBuffer buffer, ref ProtocolId obj)

{

buffer.Get(obj.Id, 0, ProtocolId.PROTOID\_SIZE);

}

public static void ReadProtocolId( IoBuffer buffer, ref ProtocolId obj)

{

if (obj == null)

obj = new ProtocolId();

buffer.Get(obj.Id, 0, ProtocolId.PROTOID\_SIZE);

}

}

public class ProtocolIdSerializer : IStaticTypeSerializer

{

delegate void WriterDelegate(IoBuffer buffer, ProtocolId obj);

delegate void ReaderDelegate(IoBuffer buffer, ref ProtocolId obj);

public void GetStaticMethods(System.Type type, out MethodInfo writer, out MethodInfo reader)

{

WriterDelegate writerDelegate = ProtocolIdEncoder.WriteProtocolId;

ReaderDelegate readerDelegate = ProtocolIdEncoder.ReadProtocolId;

writer = writerDelegate.Method;

reader = readerDelegate.Method;

}

public bool Handles(System.Type type)

{

return type == typeof(ProtocolId);

}

public System.Collections.Generic.IEnumerable<System.Type> GetSubtypes(System.Type type)

{

yield break;

}

}

}

#### ProtocolVersionEncoder.cs

using Doopec.Serializer;

using Mina.Core.Buffer;

using Rtps.Structure.Types;

using System.Reflection;

namespace Doopec.Rtps.Encoders

{

public static class ProtocolVersionEncoder

{

public static void PutProtocolVersion(this IoBuffer buffer, ProtocolVersion obj)

{

buffer.Put(obj.Major);

buffer.Put(obj.Minor);

}

public static void WriteProtocolVersion(IoBuffer buffer, ProtocolVersion obj)

{

buffer.Put(obj.Major);

buffer.Put(obj.Minor);

}

public static ProtocolVersion GetProtocolVersion(this IoBuffer buffer)

{

ProtocolVersion obj = new ProtocolVersion();

buffer.GetProtocolVersion(ref obj);

return obj;

}

public static void GetProtocolVersion(this IoBuffer buffer, ref ProtocolVersion obj)

{

obj.Major = buffer.Get();

obj.Minor = buffer.Get();

}

public static void ReadProtocolVersion( IoBuffer buffer, ref ProtocolVersion obj)

{

if (obj == null)

obj = new ProtocolVersion();

obj.Major = buffer.Get();

obj.Minor = buffer.Get();

}

}

public class ProtocolVersionSerializer : IStaticTypeSerializer

{

delegate void WriterDelegate(IoBuffer buffer, ProtocolVersion obj);

delegate void ReaderDelegate(IoBuffer buffer, ref ProtocolVersion obj);

public void GetStaticMethods(System.Type type, out MethodInfo writer, out MethodInfo reader)

{

WriterDelegate writerDelegate = ProtocolVersionEncoder.WriteProtocolVersion;

ReaderDelegate readerDelegate = ProtocolVersionEncoder.ReadProtocolVersion;

writer = writerDelegate.Method;

reader = readerDelegate.Method;

}

public bool Handles(System.Type type)

{

return type == typeof(ProtocolVersion);

}

public System.Collections.Generic.IEnumerable<System.Type> GetSubtypes(System.Type type)

{

yield break;

}

}

}

#### Sentinel.cs

using Rtps.Messages.Submessages.Elements;

using Rtps.Messages.Types;

namespace Doopec.Encoders.RTPS

{

public class Sentinel : Parameter

{

private static Sentinel instance = new Sentinel();

private Sentinel()

: base(ParameterId.PID\_SENTINEL)

{

this.Bytes = new byte[0];

}

public static Sentinel Instance { get { return instance; } }

}

}

#### SequenceNumberEncoder.cs

using Mina.Core.Buffer;

using Rtps.Structure.Types;

namespace Doopec.Rtps.Encoders

{

public static class SequenceNumberEncoder

{

public static void PutSequenceNumber(this IoBuffer buffer, SequenceNumber obj)

{

buffer.PutInt32(obj.High);

buffer.PutInt32((int)obj.Low);

}

public static SequenceNumber GetSequenceNumber(this IoBuffer buffer)

{

SequenceNumber obj = new SequenceNumber();

buffer.GetSequenceNumber(ref obj);

return obj;

}

public static void GetSequenceNumber(this IoBuffer buffer, ref SequenceNumber obj)

{

obj.High = buffer.GetInt32();

obj.Low = (uint)buffer.GetInt32(); ;

}

}

}

#### SequenceNumberSetEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages.Elements;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Doopec.Rtps.Encoders

{

public static class SequenceNumberSetEncoder

{

public static void PutSequenceNumberSet(this IoBuffer buffer, SequenceNumberSet obj)

{

buffer.PutSequenceNumber(obj.BitmapBase);

// buffer.write\_long(bitmaps.length);

// buffer.write\_long(bitmaps.length \* 32);

buffer.PutInt32(obj.NumBits);

for (int i = 0; i < obj.Bitmaps.Length; i++)

{

buffer.PutInt32(obj.Bitmaps[i]);

}

}

public static SequenceNumberSet GetSequenceNumberSet(this IoBuffer buffer)

{

SequenceNumberSet obj = new SequenceNumberSet();

buffer.GetSequenceNumberSet(ref obj);

return obj;

}

public static void GetSequenceNumberSet(this IoBuffer buffer, ref SequenceNumberSet obj)

{

obj.BitmapBase = buffer.GetSequenceNumber();

obj.NumBits = buffer.GetInt32();

int count = (obj.NumBits + 31) / 32;

obj.Bitmaps = new int[count];

for (int i = 0; i < obj.Bitmaps.Length; i++)

{

obj.Bitmaps[i] = buffer.GetInt32();

}

}

}

}

#### StatusInfoEncoder.cs

using Mina.Core.Buffer;

using rtps.messages.elements;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Doopec.Rtps.Encoders

{

public static class StatusInfoEncoder

{

public static void PutStatusInfo(this IoBuffer buffer, StatusInfo obj)

{

buffer.Put(obj.Bytes);

}

public static StatusInfo GetStatusInfo(this IoBuffer buffer)

{

StatusInfo obj = new StatusInfo();

buffer.GetStatusInfo(ref obj);

return obj;

}

public static void GetStatusInfo(this IoBuffer buffer, ref StatusInfo obj)

{

buffer.Get(obj.Bytes, 0, 4);

}

}

}

#### SubMessageEncoder.cs

using Mina.Core.Buffer;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Reflection;

using log4net;

using Rtps.Messages;

using Rtps.Messages.Types;

using Rtps.Messages.Submessages;

using Doopec.Utils.Network.Encoders;

namespace Doopec.Rtps.Encoders

{

public static class SubMessageEncoder

{

private static readonly ILog log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

public static int PutSubMessage(this IoBuffer buffer, SubMessage msg)

{

// The PSM aligns each Submessage on a 32-bit boundary with respect to the start of the Message (page 159).

buffer.Align(4);

buffer.Order = (msg.Header.IsLittleEndian ? ByteOrder.LittleEndian : ByteOrder.BigEndian); // Set the endianess

buffer.PutSubMessageHeader(msg.Header);

int position = buffer.Position;

switch (msg.Kind)

{

case SubMessageKind.PAD:

buffer.PutPad((Pad)msg);

break;

case SubMessageKind.ACKNACK:

buffer.PutAckNack((AckNack)msg);

break;

case SubMessageKind.HEARTBEAT:

buffer.PutHeartbeat((Heartbeat)msg);

break;

case SubMessageKind.GAP:

buffer.PutGap((Gap)msg);

break;

case SubMessageKind.INFO\_TS:

buffer.PutInfoTimestamp((InfoTimestamp)msg);

break;

case SubMessageKind.INFO\_SRC:

buffer.PutInfoSource((InfoSource)msg);

break;

case SubMessageKind.INFO\_REPLY\_IP4:

buffer.PutInfoReplyIp4((InfoReplyIp4)msg);

break;

case SubMessageKind.INFO\_DST:

buffer.PutInfoDestination((InfoDestination)msg);

break;

case SubMessageKind.INFO\_REPLY:

buffer.PutInfoReply((InfoReply)msg);

break;

case SubMessageKind.NACK\_FRAG:

buffer.PutNackFrag((NackFrag)msg);

break;

case SubMessageKind.HEARTBEAT\_FRAG:

buffer.PutHeartbeatFrag((HeartbeatFrag)msg);

break;

case SubMessageKind.DATA:

buffer.PutDataSubMessage((Data)msg);

break;

case SubMessageKind.DATA\_FRAG:

buffer.PutDataFrag((DataFrag)msg);

break;

default:

break;

}

buffer.Align(4);

int subMessageLength = buffer.Position - position;

// Position to 'submessageLength' -2 is for short (2 bytes)

// buffers current position is not changed

buffer.PutInt16(position - 2, (short)subMessageLength);

return position;

}

public static SubMessage GetSubMessage(this IoBuffer buffer)

{

SubMessage obj = new SubMessage();

buffer.GetSubMessage(ref obj);

return obj;

}

public static void GetSubMessage(this IoBuffer buffer, ref SubMessage obj)

{

buffer.Align(4);

int smhPosition = buffer.Position;

SubMessageHeader header = buffer.GetSubMessageHeader();

int smStart = buffer.Position;

switch (header.SubMessageKind)

{ // @see 9.4.5.1.1

case SubMessageKind.PAD:

Pad smPad = new Pad();

smPad.Header = header;

buffer.GetPad(ref smPad);

obj = smPad;

break;

case SubMessageKind.ACKNACK:

AckNack smAckNack = new AckNack();

smAckNack.Header = header;

buffer.GetAckNack(ref smAckNack);

obj = smAckNack;

break;

case SubMessageKind.HEARTBEAT:

Heartbeat smHeartbeat = new Heartbeat();

smHeartbeat.Header = header;

buffer.GetHeartbeat(ref smHeartbeat);

obj = smHeartbeat;

break;

case SubMessageKind.GAP:

Gap smgap = new Gap();

smgap.Header = header;

buffer.GetGap(ref smgap);

obj = smgap;

break;

case SubMessageKind.INFO\_TS:

InfoTimestamp sminfots = new InfoTimestamp();

sminfots.Header = header;

buffer.GetInfoTimestamp(ref sminfots);

obj = sminfots;

break;

case SubMessageKind.INFO\_SRC:

InfoSource smInfoSource = new InfoSource();

smInfoSource.Header = header;

buffer.GetInfoSource(ref smInfoSource);

obj = smInfoSource;

break;

case SubMessageKind.INFO\_REPLY\_IP4:

InfoReplyIp4 smInfoReplyIp4 = new InfoReplyIp4();

smInfoReplyIp4.Header = header;

buffer.GetInfoReplyIp4(ref smInfoReplyIp4);

obj = smInfoReplyIp4;

break;

case SubMessageKind.INFO\_DST:

InfoDestination smInfoDestination = new InfoDestination();

smInfoDestination.Header = header;

buffer.GetInfoDestination(ref smInfoDestination);

obj = smInfoDestination;

break;

case SubMessageKind.INFO\_REPLY:

InfoReply smInfoReply = new InfoReply();

smInfoReply.Header = header;

buffer.GetInfoReply(ref smInfoReply);

obj = smInfoReply;

break;

case SubMessageKind.NACK\_FRAG:

NackFrag smNackFrag = new NackFrag();

smNackFrag.Header = header;

buffer.GetNackFrag(ref smNackFrag);

obj = smNackFrag;

break;

case SubMessageKind.HEARTBEAT\_FRAG:

HeartbeatFrag smHeartbeatFrag = new HeartbeatFrag();

smHeartbeatFrag.Header = header;

buffer.GetHeartbeatFrag(ref smHeartbeatFrag);

obj = smHeartbeatFrag;

break;

case SubMessageKind.DATA:

Data smdata = new Data();

smdata.Header = header;

buffer.GetDataSubMessage(ref smdata);

obj = smdata;

break;

case SubMessageKind.DATA\_FRAG:

DataFrag smdDataFrag = new DataFrag();

smdDataFrag.Header = header;

buffer.GetDataFrag(ref smdDataFrag);

obj = smdDataFrag;

break;

default:

throw new NotSupportedException();

}

int smEnd = buffer.Position;

int smLength = smEnd - smStart;

if (smLength != header.SubMessageLength && header.SubMessageLength != 0)

{

log.WarnFormat("SubMessage length differs for {0} != {1} for {2}", smLength, header.SubMessageLength, obj);

if (smLength < header.SubMessageLength)

{

byte[] unknownBytes = new byte[header.SubMessageLength - smLength];

log.DebugFormat("Trying to skip {0} bytes", unknownBytes.Length);

buffer.Get(unknownBytes, 0, unknownBytes.Length);

}

}

log.DebugFormat("SubMsg in: {0}", obj);

}

}

}

#### SubmessageHeaderEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages;

using Rtps.Messages.Types;

namespace Doopec.Rtps.Encoders

{

public static class SubMessageHeaderEncoder

{

public static void PutSubMessageHeader(this IoBuffer buffer, SubMessageHeader obj)

{

buffer.Put((byte)obj.SubMessageKind);

buffer.Put((byte)obj.FlagsValue);

buffer.PutInt16((short)obj.SubMessageLength);

}

public static SubMessageHeader GetSubMessageHeader(this IoBuffer buffer)

{

SubMessageHeader obj = new SubMessageHeader((SubMessageKind)0, 0);

buffer.GetSubMessageHeader(ref obj);

return obj;

}

public static void GetSubMessageHeader(this IoBuffer buffer, ref SubMessageHeader obj)

{

obj.SubMessageKind = (SubMessageKind)buffer.Get();

obj.FlagsValue = buffer.Get();

buffer.Order = (obj.IsLittleEndian ? ByteOrder.LittleEndian : ByteOrder.BigEndian); // Set the endianess

obj.SubMessageLength = (ushort)buffer.GetInt16();

}

}

}

#### TimeEncoder.cs

using Mina.Core.Buffer;

using Rtps.Messages.Types;

namespace Doopec.Rtps.Encoders

{

public static class TimeEncoder

{

public static void PutTime(this IoBuffer buffer, Time obj)

{

buffer.PutInt32(obj.Seconds);

buffer.PutInt32((int)obj.Fraction);

}

public static Time GetTime(this IoBuffer buffer)

{

Time obj = new Time();

buffer.GetTime(ref obj);

return obj;

}

public static void GetTime(this IoBuffer buffer, ref Time obj)

{

obj.Seconds = buffer.GetInt32();

obj.Fraction = (uint)buffer.GetInt32(); ;

}

}

}

#### VendorIdEncoder.cs

using Doopec.Serializer;

using Mina.Core.Buffer;

using Rtps.Structure.Types;

using System.Reflection;

namespace Doopec.Rtps.Encoders

{

public static class VendorIdEncoder

{

public static void PutVendorId(this IoBuffer buffer, VendorId obj)

{

buffer.Put(obj.ToBytes());

}

public static void WriteVendorId(IoBuffer buffer, VendorId obj)

{

buffer.Put(obj.ToBytes());

}

public static VendorId GetVendorId(this IoBuffer buffer)

{

VendorId obj = new VendorId();

buffer.GetVendorId(ref obj);

return obj;

}

public static void GetVendorId(this IoBuffer buffer, ref VendorId obj)

{

obj.Byte0 = buffer.Get();

obj.Byte1 = buffer.Get();

}

public static void ReadVendorId(IoBuffer buffer, ref VendorId obj)

{

if (obj == null)

obj = new VendorId();

obj.Byte0 = buffer.Get();

obj.Byte1 = buffer.Get();

}

}

public class VendorIdSerializer : IStaticTypeSerializer

{

delegate void WriterDelegate(IoBuffer buffer, VendorId obj);

delegate void ReaderDelegate(IoBuffer buffer, ref VendorId obj);

public void GetStaticMethods(System.Type type, out MethodInfo writer, out MethodInfo reader)

{

WriterDelegate writerDelegate = VendorIdEncoder.WriteVendorId;

ReaderDelegate readerDelegate = VendorIdEncoder.ReadVendorId;

writer = writerDelegate.Method;

reader = readerDelegate.Method;

}

public bool Handles(System.Type type)

{

return type == typeof(VendorId);

}

public System.Collections.Generic.IEnumerable<System.Type> GetSubtypes(System.Type type)

{

yield break;

}

}

}

### Messages

#### CDREncapsulation.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages.Elements;

using System;

using Doopec.Rtps.Encoders;

using System.Diagnostics;

namespace Doopec.Rtps.Messages

{

/// <summary>

/// CDREncapsulation is a general purpose binary DataEncapsulation. It holds a IoBuffer,

/// which can be used by marshallers.

/// In addition to the encapsulation identifier, the OMG CDR encapsulation specifies the length of the data followed by the

/// data encapsulated using CDR. The same encapsulation scheme is used for both the length and serialized data.

/// 0...2...........8...............16..............24..............32

/// +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

/// | CDR\_BE | ushort options |

/// +---------------+---------------+---------------+---------------+

/// | Data Length |

/// +---------------+---------------+---------------+---------------+

/// ~ Serialized Data (CDR Big Endian) ~

/// | |

/// +---------------+---------------+---------------+---------------+

///

/// 0...2...........8...............16..............24..............32

/// +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

/// | CDR\_LE | ushort options |

/// +---------------+---------------+---------------+---------------+

/// | Data Length |

/// +---------------+---------------+---------------+---------------+

/// ~ Serialized Data (CDR Little Endian) ~

/// | |

/// +---------------+---------------+---------------+---------------+

/// </summary>

public class CDREncapsulation : DataEncapsulation

{

private ByteOrder order;

private byte[] data;

public CDREncapsulation()

{ }

public CDREncapsulation(IoBuffer buffer, object dataObj, ByteOrder order)

{

int initialPos = buffer.Position;

this.order = order;

buffer.Order = this.order;

if (order == ByteOrder.LittleEndian)

buffer.PutEncapsulationScheme(CDR\_LE\_HEADER);

else

buffer.PutEncapsulationScheme(CDR\_BE\_HEADER);

Doopec.Serializer.Serializer.Serialize(buffer, dataObj);

var serializedData = new byte[buffer.Position - initialPos];

buffer.Position = initialPos;

buffer.Get(serializedData, 0, serializedData.Length);

data = serializedData;

}

public static void Serialize(IoBuffer buffer, object dataObj, ByteOrder order)

{

buffer.Order = order;

if (order == ByteOrder.LittleEndian)

buffer.PutEncapsulationScheme(CDR\_LE\_HEADER);

else

buffer.PutEncapsulationScheme(CDR\_BE\_HEADER);

Doopec.Serializer.Serializer.Serialize(buffer, dataObj);

}

public static T Deserialize<T>(IoBuffer buffer)

{

EncapsulationScheme scheme = buffer.GetEncapsulationScheme();

if (scheme.Equals(DataEncapsulation.CDR\_BE\_HEADER))

{

buffer.Order = ByteOrder.BigEndian;

}

else if (scheme.Equals(DataEncapsulation.CDR\_LE\_HEADER))

{

buffer.Order = ByteOrder.LittleEndian;

}

else

{

throw new NotImplementedException();

}

T rst = Doopec.Serializer.Serializer.Deserialize<T>(buffer);

return rst;

}

public static CDREncapsulation Deserialize(IoBuffer buffer, int length)

{

int initialPos = buffer.Position;

EncapsulationScheme scheme = buffer.GetEncapsulationScheme();

ByteOrder order;

if (scheme.Equals(DataEncapsulation.CDR\_BE\_HEADER))

{

order = buffer.Order = ByteOrder.BigEndian;

}

else if (scheme.Equals(DataEncapsulation.CDR\_LE\_HEADER))

{

order = buffer.Order = ByteOrder.LittleEndian;

}

else

{

throw new NotImplementedException();

}

byte[] data = new byte[length - 4];

buffer.Get(data, 0, length - 4);

Debug.Assert(buffer.Position == initialPos + length);

CDREncapsulation rst = new CDREncapsulation(data, order);

return rst;

}

public CDREncapsulation(byte[] serializeData, ByteOrder order)

{

this.order = order;

data = serializeData;

}

public override byte[] SerializedPayload

{

get

{

return data;

}

}

/// <summary>

/// Gets a IoBuffer, which can be used to marshall/unmarshall data.

/// </summary>

public IoBuffer Buffer

{

get

{

IoBuffer buff = IoBuffer.Allocate(data.Length + 4);

buff.Order = this.order;

if (order == ByteOrder.LittleEndian)

buff.PutEncapsulationScheme(CDR\_LE\_HEADER);

else

buff.PutEncapsulationScheme(CDR\_BE\_HEADER);

buff.Put(data);

return buff;

}

}

public override bool ContainsData()

{

return (data != null);

}

}

}

#### EncapsulationManager.cs

using Mina.Core.Buffer;

using Rtps.Messages.Submessages.Elements;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Doopec.Rtps.Encoders;

using Doopec.Serializer.Attributes;

using Doopec.Encoders;

namespace Doopec.Rtps.Messages

{

public class EncapsulationManager

{

public static DataEncapsulation Serialize<T>(T obj, Encapsulation scheme = Encapsulation.CDR\_BE)

{

DataEncapsulation rst;

if (scheme == Encapsulation.UNKNOWN)

{

PacketAttribute packetAttr = PacketAttribute.GetAttribute(typeof(T));

if (packetAttr != null)

scheme = packetAttr.EncapsulationScheme;

}

IoBuffer buff = IoBuffer.Allocate(1024);

buff.AutoExpand = true;

switch (scheme)

{

default:

case Encapsulation.CDR\_BE:

rst = buff.EncapsuleCDRData(obj, ByteOrder.BigEndian);

break;

case Encapsulation.CDR\_LE:

rst = buff.EncapsuleCDRData(obj, ByteOrder.LittleEndian);

break;

case Encapsulation.PL\_CDR\_BE:

rst = buff.EncapsuleParameterListData(obj, ByteOrder.BigEndian);

break;

case Encapsulation.PL\_CDR\_LE:

rst = buff.EncapsuleParameterListData(obj, ByteOrder.LittleEndian);

break;

}

return rst;

}

public static DataEncapsulation Deserialize(IoBuffer buffer, int length)

{

int pos = buffer.Position;

EncapsulationScheme scheme = buffer.GetEncapsulationScheme();

buffer.Position = pos;

if (scheme.Equals(DataEncapsulation.CDR\_BE\_HEADER) || scheme.Equals(DataEncapsulation.CDR\_LE\_HEADER))

{

return CDREncapsulation.Deserialize(buffer, length);

}

else if (scheme.Equals(DataEncapsulation.PL\_CDR\_BE\_HEADER) || scheme.Equals(DataEncapsulation.PL\_CDR\_LE\_HEADER))

{

return ParameterListEncapsulation.Deserialize(buffer, length);

}

else

throw new ApplicationException("Unkonw scheme encapsulation " + scheme);

}

public static T Deserialize<T>(IoBuffer buffer) where T : new()

{

int pos = buffer.Position;

EncapsulationScheme scheme = buffer.GetEncapsulationScheme();

buffer.Position = pos;

if (scheme.Equals(DataEncapsulation.CDR\_BE\_HEADER) || scheme.Equals(DataEncapsulation.CDR\_LE\_HEADER))

{

return CDREncapsulation.Deserialize<T>(buffer);

}

else if (scheme.Equals(DataEncapsulation.PL\_CDR\_BE\_HEADER) || scheme.Equals(DataEncapsulation.PL\_CDR\_LE\_HEADER))

{

return ParameterListEncapsulation.Deserialize<T>(buffer);

}

else

throw new ApplicationException("Unkonw scheme encapsulation " + scheme);

}

}

}

#### ParameterListEncapsulation.cs

using Doopec.Rtps.Encoders;

using Mina.Core.Buffer;

using Rtps.Messages.Submessages.Elements;

using System.Reflection;

using System;

using System.Linq;

using org.omg.dds.type;

using Rtps.Messages.Types;

using System.Diagnostics;

using Doopec.XTypes;

using org.omg.dds.type.typeobject;

namespace Doopec.Rtps.Messages

{

/// <summary>

/// ParameterListEncapsulation is a specialization of DataEncapsulation.

/// In addition to the encapsulation identifier, the ParameterList encapsulation specifies the length of the data followed by the

/// data encapsulated using a ParameterList. The same CDR encoding is used for both the length and the parameter list.

/// 0...2...........8...............16..............24..............32

/// +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

/// | PL\_CDR\_BE | ushort options |

/// +---------------+---------------+---------------+---------------+

/// | |

/// ~ Serialized Data (ParameterList CDR Big Endian) ~

/// | |

/// +---------------+---------------+---------------+---------------+

/// 0...2...........8...............16..............24..............32

/// +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

/// | PL\_CDR\_LE | ushort options |

/// +---------------+---------------+---------------+---------------+

/// | |

/// ~ Serialized Data (ParameterList CDR Little Endian) ~

/// | |

/// +---------------+---------------+---------------+---------------+

///

/// And Serialized Data is:

/// ....2...........8...............16.............24...............32

/// +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

/// | short parameterId\_1 | short length\_1 |

/// +---------------+---------------+---------------+---------------+

/// | |

/// ~ octet[length\_1] value\_1 ~

/// | |

/// +---------------+---------------+---------------+---------------+

/// | short parameterId\_2 | short length\_2 |

/// +---------------+---------------+---------------+---------------+

/// | |

/// ~ octet[length\_2] value\_2 ~

/// | |

/// +---------------+---------------+---------------+---------------+

/// | |

/// ~ ... ~

/// | |

/// | |

/// +---------------+---------------+---------------+---------------+

/// | PID\_SENTINEL | ignored |

/// +---------------+---------------+---------------+---------------+

///

/// This encapsulation is used by IsDiscovery.

/// </summary>

public class ParameterListEncapsulation : DataEncapsulation

{

private ParameterList parameters;

private byte[] data;

private ByteOrder order;

public ParameterListEncapsulation(IoBuffer buffer, object dataObj, ByteOrder order)

{

int initialPos = buffer.Position;

this.order = order;

if (order == ByteOrder.LittleEndian)

buffer.PutEncapsulationScheme(PL\_CDR\_LE\_HEADER);

else

buffer.PutEncapsulationScheme(PL\_CDR\_BE\_HEADER);

buffer.Order = this.order;

ParameterList parameters = BuildParameters(dataObj, order);

buffer.PutParameterList(parameters);

data = new byte[buffer.Position - initialPos];

buffer.Position = initialPos;

buffer.Get(data, 0, data.Length);

}

internal ParameterListEncapsulation(IoBuffer buffer, ByteOrder order, int length)

{

this.order = order;

this.data = new byte[length];

buffer.Get(data, 0, data.Length);

}

public static void Serialize(IoBuffer buffer, object dataObj, ByteOrder order)

{

buffer.Order = order;

ParameterList parameters = BuildParameters(dataObj, order);

Serialize(buffer, parameters, order);

}

public static void Serialize(IoBuffer buffer, ParameterList parameters, ByteOrder order)

{

if (order == ByteOrder.LittleEndian)

buffer.PutEncapsulationScheme(PL\_CDR\_LE\_HEADER);

else

buffer.PutEncapsulationScheme(PL\_CDR\_BE\_HEADER);

buffer.Order = order;

int initialPos = buffer.Position;

buffer.Position += 4;

buffer.PutParameterList(parameters);

int finalPos = buffer.Position;

buffer.Position = initialPos;

buffer.PutInt32(finalPos - initialPos - 4);

buffer.Position = finalPos;

}

private static ParameterList BuildParameters(object obj, ByteOrder order)

{

var type = TypeExplorer.ExploreType(obj.GetType());

//var fields = obj.GetType().GetFields(BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.DeclaredOnly)

// .Where(fi => (fi.Attributes & FieldAttributes.NotSerialized) == 0);

StructureType structType = type as StructureType;

ParameterList parameters = new ParameterList();

foreach (var member in structType.GetMember())

{

Parameter parameter = new Parameter();

//uint id = field.GetProperty().MemberId;

parameter.ParameterId = (ParameterId)member.GetProperty().MemberId; ;

IoBuffer buffer = ByteBufferAllocator.Instance.Allocate(64);

buffer.Order = order;

buffer.AutoExpand = true;

if (member.GetProperty().IsProperty)

{

var field = obj.GetType().GetProperty(member.GetProperty().Name);

object data = field.GetValue(obj);

if (data == null)

continue;

Doopec.Serializer.Serializer.Serialize(buffer, data);

}

else

{

var field = obj.GetType().GetField(member.GetProperty().Name);

Doopec.Serializer.Serializer.Serialize(buffer, field.GetValue(obj));

}

int length = buffer.Position;

parameter.Bytes = new byte[length];

buffer.Rewind();

buffer.Get(parameter.Bytes, 0, length);

parameters.Value.Add(parameter);

}

return parameters;

}

public static T Deserialize<T>(IoBuffer buffer) where T : new()

{

EncapsulationScheme scheme = buffer.GetEncapsulationScheme();

if (scheme.Equals(DataEncapsulation.PL\_CDR\_BE\_HEADER))

{

buffer.Order = ByteOrder.BigEndian;

}

else if (scheme.Equals(DataEncapsulation.PL\_CDR\_LE\_HEADER))

{

buffer.Order = ByteOrder.LittleEndian;

}

else

{

throw new NotImplementedException();

}

int initialPos = buffer.Position;

ParameterList parameters = buffer.GetParameterList();

return BuildObject<T>(parameters, buffer.Order);

}

public static ParameterListEncapsulation Deserialize(IoBuffer buffer, int length)

{

int initialPos = buffer.Position;

EncapsulationScheme scheme = buffer.GetEncapsulationScheme();

if (scheme.Equals(DataEncapsulation.PL\_CDR\_BE\_HEADER))

{

buffer.Order = ByteOrder.BigEndian;

}

else if (scheme.Equals(DataEncapsulation.PL\_CDR\_LE\_HEADER))

{

buffer.Order = ByteOrder.LittleEndian;

}

else

{

throw new NotImplementedException();

}

buffer.Position = initialPos;

return new ParameterListEncapsulation(buffer, buffer.Order, length);

}

private static T BuildObject<T>(ParameterList parameters, ByteOrder order) where T : new()

{

var type = TypeExplorer.ExploreType(typeof(T));

T obj = new T();

int cnt = 0;

StructureType structType = type as StructureType;

foreach (var member in structType.GetMember())

{

Parameter parameter = parameters.Value.Where(p => (uint)p.ParameterId == member.GetProperty().MemberId).FirstOrDefault();

if (parameter == null)

continue;

IoBuffer buffer = IoBuffer.Wrap(parameter.Bytes);

buffer.Order = order;

MethodInfo method = typeof(Doopec.Serializer.Serializer).GetMethods().Where(x => x.Name == "Deserialize" && x.IsGenericMethod).SingleOrDefault(); ;

if (member.GetProperty().IsProperty)

{

var field = obj.GetType().GetProperty(member.GetProperty().Name);

MethodInfo generic = method.MakeGenericMethod(field.PropertyType);

object val = generic.Invoke(null, new object[] { buffer });

field.SetValue(obj, val);

}

else

{

var field = obj.GetType().GetField(member.GetProperty().Name);

MethodInfo generic = method.MakeGenericMethod(field.FieldType);

object val = generic.Invoke(null, new object[] { buffer });

field.SetValue(obj, val);

}

}

return obj;

}

public ParameterListEncapsulation(ParameterList parameters, ByteOrder order)

{

this.parameters = parameters;

this.order = order;

}

public ParameterList GetParameterList()

{

return parameters;

}

public override bool ContainsData()

{

return (data != null); // TODO: how do we represent key in serialized payload

}

public override byte[] SerializedPayload

{

get

{

return data;

}

}

}

}

### Discovery

#### DiscoveryReaderData.cs

using org.omg.dds.topic;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Doopec.Rtps.Discovery

{

public class DiscoveredReaderData : SubscriptionBuiltinTopicData

{

public override BuiltinTopicKey Key

{

get

{

throw new NotImplementedException();

}

}

public override BuiltinTopicKey ParticipantKey

{

get

{

throw new NotImplementedException();

}

}

public override string TopicName

{

get

{

throw new NotImplementedException();

}

}

public override string TypeName

{

get

{

throw new NotImplementedException();

}

}

public override List<string> EquivalentTypeName

{

get

{

throw new NotImplementedException();

}

}

public override List<string> BaseTypeName

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.type.typeobject.TypeObject Type

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.DurabilityQosPolicy Durability

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.DeadlineQosPolicy Deadline

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.LatencyBudgetQosPolicy LatencyBudget

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.LivelinessQosPolicy Liveliness

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.ReliabilityQosPolicy Reliability

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.OwnershipQosPolicy Ownership

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.DestinationOrderQosPolicy DestinationOrder

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.UserDataQosPolicy UserData

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.TimeBasedFilterQosPolicy TimeBasedFilter

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.PresentationQosPolicy Presentation

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.PartitionQosPolicy Partition

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.TopicDataQosPolicy TopicData

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.GroupDataQosPolicy GroupData

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.DataRepresentationQosPolicy Representation

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.TypeConsistencyEnforcementQosPolicy TypeConsistency

{

get

{

throw new NotImplementedException();

}

}

public override SubscriptionBuiltinTopicData CopyFrom(SubscriptionBuiltinTopicData other)

{

throw new NotImplementedException();

}

public override SubscriptionBuiltinTopicData FinishModification()

{

throw new NotImplementedException();

}

public override SubscriptionBuiltinTopicData Modify()

{

throw new NotImplementedException();

}

public override org.omg.dds.core.Bootstrap GetBootstrap()

{

throw new NotImplementedException();

}

}

}

#### DiscoveryTopicData.cs

using org.omg.dds.topic;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Doopec.Rtps.Discovery

{

public class DiscoveredTopicData : TopicBuiltinTopicData

{

public override BuiltinTopicKey Key

{

get

{

throw new NotImplementedException();

}

}

public override string Name

{

get

{

throw new NotImplementedException();

}

}

public override string TypeName

{

get

{

throw new NotImplementedException();

}

}

public override List<string> EquivalentTypeName

{

get

{

throw new NotImplementedException();

}

}

public override List<string> BaseTypeName

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.type.typeobject.TypeObject Type

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.DurabilityQosPolicy Durability

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.DurabilityServiceQosPolicy DurabilityService

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.DeadlineQosPolicy Deadline

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.LatencyBudgetQosPolicy LatencyBudget

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.LivelinessQosPolicy Liveliness

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.ReliabilityQosPolicy Reliability

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.TransportPriorityQosPolicy TransportPriority

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.LifespanQosPolicy Lifespan

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.DestinationOrderQosPolicy DestinationOrder

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.HistoryQosPolicy History

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.ResourceLimitsQosPolicy ResourceLimits

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.OwnershipQosPolicy Ownership

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.TopicDataQosPolicy TopicData

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.DataRepresentationQosPolicy Representation

{

get

{

throw new NotImplementedException();

}

}

public override org.omg.dds.core.policy.TypeConsistencyEnforcementQosPolicy TypeConsistency

{

get

{

throw new NotImplementedException();

}

}

public override TopicBuiltinTopicData CopyFrom(TopicBuiltinTopicData other)

{

throw new NotImplementedException();

}

public override TopicBuiltinTopicData FinishModification()

{

throw new NotImplementedException();

}

public override TopicBuiltinTopicData Modify()

{

throw new NotImplementedException();

}

public override org.omg.dds.core.Bootstrap GetBootstrap()

{

throw new NotImplementedException();

}

}

}

#### DiscoveryWriterData.cs

using org.omg.dds.topic;

namespace Doopec.Rtps.Discovery

{

public class DiscoveredWriterData : PublicationBuiltinTopicData

{

public override BuiltinTopicKey Key

{

get

{

throw new System.NotImplementedException();

}

}

public override BuiltinTopicKey ParticipantKey

{

get

{

throw new System.NotImplementedException();

}

}

public override string TopicName

{

get

{

throw new System.NotImplementedException();

}

}

public override string TypeName

{

get

{

throw new System.NotImplementedException();

}

}

public override System.Collections.Generic.List<string> EquivalentTypeName

{

get

{

throw new System.NotImplementedException();

}

}

public override System.Collections.Generic.List<string> BaseTypeName

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.type.typeobject.TypeObject Type

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.DurabilityQosPolicy Durability

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.DurabilityServiceQosPolicy DurabilityService

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.DeadlineQosPolicy Deadline

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.LatencyBudgetQosPolicy LatencyBudget

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.LivelinessQosPolicy Liveliness

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.ReliabilityQosPolicy Reliability

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.LifespanQosPolicy Lifespan

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.UserDataQosPolicy UserData

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.OwnershipQosPolicy Ownership

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.OwnershipStrengthQosPolicy OwnershipStrength

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.DestinationOrderQosPolicy DestinationOrder

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.PresentationQosPolicy Presentation

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.PartitionQosPolicy Partition

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.TopicDataQosPolicy TopicData

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.GroupDataQosPolicy GroupData

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.DataRepresentationQosPolicy Representation

{

get

{

throw new System.NotImplementedException();

}

}

public override org.omg.dds.core.policy.TypeConsistencyEnforcementQosPolicy TypeConsistency

{

get

{

throw new System.NotImplementedException();

}

}

public override PublicationBuiltinTopicData CopyFrom(PublicationBuiltinTopicData other)

{

throw new System.NotImplementedException();

}

public override PublicationBuiltinTopicData FinishModification()

{

throw new System.NotImplementedException();

}

public override PublicationBuiltinTopicData Clone()

{

throw new System.NotImplementedException();

}

public override PublicationBuiltinTopicData Modify()

{

throw new System.NotImplementedException();

}

public override org.omg.dds.core.Bootstrap GetBootstrap()

{

throw new System.NotImplementedException();

}

}

}

#### DiscoveryImpl.cs

using Doopec.Rtps.Utils;

using org.omg.dds.domain;

using Rtps.Structure.Types;

using System;

using System.Collections.Generic;

namespace Doopec.Rtps.Discovery

{

/// <summary>

/// Discovery Strategy class that implements RTPS IsDiscovery

/// This class implements the Discovery interface for Rtps-based

/// IsDiscovery.

/// </summary>

public class DiscoveryImpl : IDisposable

{

GuidGenerator generator = new GuidGenerator();

public void StartDiscovery()

{

}

public void CloseDiscovery()

{

}

// Participant operations:

public virtual bool AttachParticipant(int domainId, int participantId)

{

throw new NotImplementedException();

}

internal virtual AddDomainStatus AddDomainParticipant(int domain, DomainParticipantQos qos)

{

lock (this)

{

AddDomainStatus ads = new AddDomainStatus() { id = new GUID(), federated = false };

generator.Populate(ref ads.id);

ads.id.EntityId = EntityId.ENTITYID\_PARTICIPANT;

try

{

if (participants\_.ContainsKey(domain) && participants\_[domain] != null)

{

participants\_[domain][ads.id] = new Spdp(domain, ads.id, qos, this);

}

else

{

participants\_[domain] = new Dictionary<GUID, Spdp>();

participants\_[domain][ads.id] = new Spdp(domain, ads.id, qos, this);

}

}

catch (Exception e)

{

ads.id = GUID.GUID\_UNKNOWN;

// ACE\_ERROR((LM\_ERROR, "(%P|%t) RtpsDiscovery::add\_domain\_participant() - "

// "failed to initialize RTPS Simple Participant Discovery Protocol: %C\n",

// e.what()));

}

return ads;

}

}

public virtual bool RemoveDomainParticipant(int domainId, int participantId)

{

throw new NotImplementedException();

}

public virtual bool IgnoreDomainParticipant(int domainId, int myParticipantId, int ignoreId)

{

throw new NotImplementedException();

}

public virtual bool UpdateDomainParticipantQos(int domain, int articipantId, DomainParticipantQos qos)

{

throw new NotImplementedException();

}

private IDictionary<int, IDictionary<GUID, Spdp>> participants\_ = new Dictionary<int, IDictionary<GUID, Spdp>>();

public void Dispose()

{

this.CloseDiscovery();

}

}

// Information returned from call to add\_domain\_participant()

internal class AddDomainStatus

{

// These are unique across a domain

// They are also the InstanceHandle\_t in Sample\_Info for built-in Topics

public GUID id;

public bool federated;

}

}

#### Sedp.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Doopec.Rtps.Discovery

{

public class Sedp

{

}

}

#### SEDPbuiltinPublicationsReader.cs

using Rtps.Behavior;

using Rtps.Structure;

using Rtps.Structure.Types;

namespace Doopec.Rtps.Discovery

{

public class SEDPbuiltinPublicationsReader : StatefulReader<DiscoveredWriterData>

{

public SEDPbuiltinPublicationsReader(GUID guid)

: base(guid)

{

this.guid = new GUID(guid.Prefix, EntityId.ENTITYID\_SEDP\_BUILTIN\_PUBLICATIONS\_READER);

}

}

}

#### SEDPbuiltinPublicationsWriter.cs

using Rtps.Behavior;

using Rtps.Structure;

using Rtps.Structure.Types;

namespace Doopec.Rtps.Discovery

{

public class SEDPbuiltinPublicationsWriter : StatefulWriter<DiscoveredWriterData>

{

public SEDPbuiltinPublicationsWriter(GUID guid)

: base(guid)

{

this.guid = new GUID(guid.Prefix, EntityId.ENTITYID\_SEDP\_BUILTIN\_PUBLICATIONS\_WRITER);

}

}

}

#### SEDPbuiltinSubscriptionsReader.cs

using Rtps.Behavior;

using Rtps.Structure;

using Rtps.Structure.Types;

namespace Doopec.Rtps.Discovery

{

public class SEDPbuiltinSubscriptionsReader : StatefulReader<DiscoveredReaderData>

{

public SEDPbuiltinSubscriptionsReader(GUID guid)

: base(guid)

{

this.guid = new GUID(guid.Prefix, EntityId.ENTITYID\_SEDP\_BUILTIN\_SUBSCRIPTIONS\_READER);

}

}

}

#### SEDPbuiltinSubscriptionsWriter.cs

using Rtps.Behavior;

using Rtps.Structure;

using Rtps.Structure.Types;

namespace Doopec.Rtps.Discovery

{

public class SEDPbuiltinSubscriptionsWriter : StatefulWriter<DiscoveredReaderData>

{

public SEDPbuiltinSubscriptionsWriter(GUID guid)

: base(guid)

{

this.guid = new GUID(guid.Prefix, EntityId.ENTITYID\_SEDP\_BUILTIN\_SUBSCRIPTIONS\_WRITER);

}

}

}

#### SEDPbuiltinTopicReader.cs

using Rtps.Behavior;

using Rtps.Structure;

using Rtps.Structure.Types;

namespace Doopec.Rtps.Discovery

{

public class SEDPbuiltinTopicsReader : StatefulReader<DiscoveredTopicData>

{

public SEDPbuiltinTopicsReader(GUID guid)

: base(guid)

{

this.guid = new GUID(guid.Prefix, EntityId.ENTITYID\_SEDP\_BUILTIN\_TOPIC\_READER);

}

}

}

#### SEDPbuiltinTopicWriter.cs

using Rtps.Behavior;

using Rtps.Structure;

using Rtps.Structure.Types;

namespace Doopec.Rtps.Discovery

{

public class SEDPbuiltinTopicsWriter : StatefulWriter<DiscoveredTopicData>

{

public SEDPbuiltinTopicsWriter(GUID guid)

: base(guid)

{

this.guid = new GUID(guid.Prefix, EntityId.ENTITYID\_SEDP\_BUILTIN\_TOPIC\_WRITER);

}

}

}

#### Spdp.cs

using org.omg.dds.domain;

using Rtps.Structure.Types;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Doopec.Rtps.Discovery

{

/// <summary>

/// Each instance of class Spdp represents the implementation of the RTPS

/// Simple Participant Discovery Protocol for a single local DomainParticipant.

/// </summary>

public class Spdp

{

public Spdp(int domain, GUID id, DomainParticipantQos qos, DiscoveryImpl disco)

{ }

}

}

#### SPDPbuiltinParticipantReaderImpl.cs

using Doopec.Configuration;

using Doopec.Configuration.Rtps;

using Doopec.Rtps.Behavior;

using Doopec.Rtps.Structure;

using Rtps.Behavior.Types;

using Rtps.Discovery.Spdp;

using Rtps.Structure;

using Rtps.Structure.Types;

using System;

using System.Collections.Generic;

using System.Net;

namespace Doopec.Rtps.Discovery

{

public class SPDPbuiltinParticipantReaderImpl : RtpsStatelessReader<SPDPdiscoveredParticipantData>

{

public SPDPbuiltinParticipantReaderImpl(Transport transportconfig, ParticipantImpl participant)

: base(participant.Guid)

{

this.TopicKind = TopicKind.WITH\_KEY;

this.ReliabilityLevel = ReliabilityKind.BEST\_EFFORT;

this.ExpectsInlineQos = false;

//The following timing-related values are used as the defaults in order to facilitate

// ‘out-of-the-box’ interoperability between implementations.

this.HeartbeatResponseDelay = new Duration(transportconfig.RtpsReader.HeartbeatResponseDelay.Val); // default is 500 milliseconds

this.HeartbeatSuppressionDuration = new Duration(transportconfig.RtpsReader.HeartbeatSuppressionDuration.Val);// default is 0 milliseconds

SetLocatorListFromConfig(transportconfig, participant);

InitReceivers();

foreach (var rec in this.UDPReceivers)

{

rec.IsDiscovery = true;

}

}

public void Start()

{

StartReceivers();

}

protected void SetLocatorListFromConfig(Transport transport, ParticipantImpl participant)

{

IList<Locator> unicastLocatorList = new List<Locator>();

IList<Locator> multicastLocatorList = new List<Locator>();

int PB = transport.Discovery.PortBase.Val;

int DG = transport.Discovery.DomainGain.Val;

int PG = transport.Discovery.ParticipantGain.Val;

int d0 = transport.Discovery.OffsetMetatrafficMulticast.Val;

int d1 = transport.Discovery.OffsetMetatrafficUnicast.Val;

string[] unicastStr = transport.Discovery.MetatrafficUnicastLocatorList.Val.Split(new char[] { ',', ';' }, StringSplitOptions.RemoveEmptyEntries);

foreach (string addr in unicastStr)

{

string[] parts = addr.Split(':');

IPAddress[] addresses = System.Net.Dns.GetHostAddresses(parts[0]);

IPAddress ipaddr = null;

if (addresses != null)

{

foreach (var ipa in addresses)

if (ipa.AddressFamily == System.Net.Sockets.AddressFamily.InterNetwork)

{

ipaddr = ipa;

break;

}

}

else

throw new ArgumentException("Invalid unicast address " + parts[0]);

int port;

if (parts.Length <= 1)

{

port = PB + DG \* participant.DomainId + d1 + PG \* participant.ParticipantId;

}

else

port = int.Parse(parts[1]);

if (ipaddr != null)

{

Locator locator = new Locator(ipaddr, port);

log.DebugFormat("Using unicast Addr:{0} and Port:{0} for SPDP Discovery", ipaddr, port);

unicastLocatorList.Add(locator);

}

}

string[] multicastStr = transport.Discovery.MetatrafficMulticastLocatorList.Val.Split(new char[] { ',', ';' }, StringSplitOptions.RemoveEmptyEntries);

foreach (string addr in multicastStr)

{

string[] parts = addr.Split(':');

IPAddress[] addresses = System.Net.Dns.GetHostAddresses(parts[0]);

IPAddress ipaddr = null;

if (addresses != null)

{

foreach (var ipa in addresses)

if (ipa.AddressFamily == System.Net.Sockets.AddressFamily.InterNetwork)

{

ipaddr = ipa;

break;

}

}

else

throw new ArgumentException("Invalid multicast address " + parts[0]);

int port;

if (parts.Length <= 1)

{

port = PB + DG \* participant.DomainId + d0;

}

else

port = int.Parse(parts[1]);

if (ipaddr != null)

{

Locator locator = new Locator(ipaddr, port);

log.DebugFormat("Using multicast Addr:{0} and Port:{0} for SPDP Discovery", ipaddr, port);

multicastLocatorList.Add(locator);

}

}

this.UnicastLocatorList = unicastLocatorList;

this.MulticastLocatorList = multicastLocatorList;

}

}

}

#### SPDPbuiltinParticipantWriterImpl.cs

using Doopec.Configuration.Rtps;

using Doopec.Rtps.Behavior;

using Doopec.Rtps.Structure;

using Doopec.Serializer.Attributes;

using Rtps.Behavior;

using Rtps.Behavior.Types;

using Rtps.Discovery.Sedp;

using Rtps.Discovery.Spdp;

using Rtps.Structure;

using Rtps.Structure.Types;

using System;

using System.Collections.Generic;

using System.Net;

namespace Doopec.Rtps.Discovery

{

/// <summary>

/// For each Participant, the SPDP creates two RTPS built-in Endpoints: the

/// SPDPbuiltinParticipantWriter and the SPDPbuiltinParticipantReader.

/// The SPDPbuiltinParticipantWriter is an RTPS Best-Effort StatelessWriter. The HistoryCache of the

/// SPDPbuiltinParticipantWriter contains a single data-object of type SPDPdiscoveredParticipantData.

/// The Value of this data-object is Set from the attributes in the Participant.

/// If the attributes change, the data-object is replaced.

/// </summary>

public class SPDPbuiltinParticipantWriterImpl : RtpsStatelessWriter<SPDPdiscoveredParticipantData>

{

private WriterWorker worker;

public SPDPbuiltinParticipantWriterImpl(Transport transportconfig, ParticipantImpl participant)

: base(participant.Guid)

{

SetLocatorListFromConfig(transportconfig, participant);

participant.DefaultMulticastLocatorList = this.MulticastLocatorList as List<Locator>;

participant.DefaultUnicastLocatorList = this.UnicastLocatorList as List<Locator>;

SPDPdiscoveredParticipantData data = new SPDPdiscoveredParticipantData(participant);

// TODO Assign UserData from configuration

CacheChange<SPDPdiscoveredParticipantData> change = this.NewChange(ChangeKind.ALIVE, new Data(data), null);

this.HistoryCache.AddChange(change);

this.TopicKind = TopicKind.WITH\_KEY;

this.ReliabilityLevel = ReliabilityKind.BEST\_EFFORT;

this.ResendDataPeriod = new Duration(transportconfig.Discovery.ResendPeriod.Val);

this.heartbeatPeriod = new Duration(transportconfig.RtpsWriter.HeartbeatPeriod.Val);

//The following timing-related values are used as the defaults in order to facilitate

// ‘out-of-the-box’ interoperability between implementations.

this.nackResponseDelay = new Duration(transportconfig.RtpsWriter.NackResponseDelay.Val); //200 milliseconds

this.nackSuppressionDuration = new Duration(transportconfig.RtpsWriter.NackSuppressionDuration.Val);

this.pushMode = transportconfig.RtpsWriter.PushMode.Val;

InitTransmitters();

foreach (var trans in this.UDPTransmitters)

{

trans.IsDiscovery = true;

}

this.Scheme = Encapsulation.PL\_CDR\_BE;

worker = new WriterWorker(this.PeriodicWork);

}

public void Start()

{

StartTransmitters();

worker.Start((int)this.ResendDataPeriod.AsMillis());

}

/// <summary>

/// The SPDPbuiltinParticipantWriter periodically sends this data-object to a pre-configured list of

/// locators to announce the Participant’s presence on the network. This is achieved by periodically

/// calling StatelessWriter::unsent\_changes\_reset, which causes the StatelessWriter to resend all

/// changes present in its HistoryCache to all locators. The periodic rate at which the

/// SPDPbuiltinParticipantWriter sends out the SPDPdiscoveredParticipantData defaults to a PSM specified

/// Value. This period should be smaller than the leaseDuration specified in the SPDPdiscoveredParticipantData

/// </summary>

protected override void PeriodicWork()

{

foreach (var change in HistoryCache.Changes)

{

SendData(change);

}

this.UnsentChangesReset();

}

protected void SetLocatorListFromConfig(Transport transport, ParticipantImpl participant)

{

IList<Locator> unicastLocatorList = new List<Locator>();

IList<Locator> multicastLocatorList = new List<Locator>();

int PB = transport.Discovery.PortBase.Val;

int DG = transport.Discovery.DomainGain.Val;

int PG = transport.Discovery.ParticipantGain.Val;

int d0 = transport.Discovery.OffsetMetatrafficMulticast.Val;

int d1 = transport.Discovery.OffsetMetatrafficUnicast.Val;

string[] unicastStr = transport.Discovery.MetatrafficUnicastLocatorList.Val.Split(new char[] { ',', ';' }, StringSplitOptions.RemoveEmptyEntries);

foreach (string addr in unicastStr)

{

string[] parts = addr.Split(':');

IPAddress[] addresses = System.Net.Dns.GetHostAddresses(parts[0]);

IPAddress ipaddr = null;

if (addresses != null)

{

foreach (var ipa in addresses)

if (ipa.AddressFamily == System.Net.Sockets.AddressFamily.InterNetwork)

{

ipaddr = ipa;

break;

}

}

else

throw new ArgumentException("Invalid unicast address " + parts[0]);

int port;

if (parts.Length <= 1)

{

port = PB + DG \* participant.DomainId + d1 + PG \* participant.ParticipantId;

}

else

port = int.Parse(parts[1]);

if (ipaddr != null)

{

Locator locator = new Locator(ipaddr, port);

log.DebugFormat("Using unicast Addr:{0} and Port:{0} for SPDP Discovery", ipaddr, port);

unicastLocatorList.Add(locator);

}

}

string[] multicastStr = transport.Discovery.MetatrafficMulticastLocatorList.Val.Split(new char[] { ',', ';' }, StringSplitOptions.RemoveEmptyEntries);

foreach (string addr in multicastStr)

{

string[] parts = addr.Split(':');

IPAddress[] addresses = System.Net.Dns.GetHostAddresses(parts[0]);

IPAddress ipaddr = null;

if (addresses != null)

{

foreach (var ipa in addresses)

if (ipa.AddressFamily == System.Net.Sockets.AddressFamily.InterNetwork)

{

ipaddr = ipa;

break;

}

}

else

throw new ArgumentException("Invalid multicast address " + parts[0]);

int port;

if (parts.Length <= 1)

{

port = PB + DG \* participant.DomainId + d0;

}

else

port = int.Parse(parts[1]);

if (ipaddr != null)

{

Locator locator = new Locator(ipaddr, port);

log.DebugFormat("Using multicast Addr:{0} and Port:{0} for SPDP Discovery", ipaddr, port);

multicastLocatorList.Add(locator);

}

}

this.UnicastLocatorList = unicastLocatorList;

this.MulticastLocatorList = multicastLocatorList;

}

}

}

#### SPDPPublicationBuiltinTopicData.cs

using Doopec.Dds.Topic;

namespace Doopec.Rtps.Discovery

{

public class SPDPPublicationBuiltinTopicData : PublicationBuiltinTopicDataImpl

{

public const string PUBLICATION\_TOPIC = "DCPSPublication,";

public SPDPPublicationBuiltinTopicData()

{

this.topicName = PUBLICATION\_TOPIC;

}

}

}

#### SPDPSubscriptionBuiltinTopicData.cs

using Doopec.Dds.Topic;

namespace Doopec.Rtps.Discovery

{

public class SPDPSubscriptionBuiltinTopicData : SubscriptionBuiltinTopicDataImpl

{

public const string SUBSCRIPTION\_TOPIC = "DCPSSubscription";

public SPDPSubscriptionBuiltinTopicData()

{

this.topicName = SUBSCRIPTION\_TOPIC;

}

}

}

#### SPDPTopicBuiltinTopicData.cs

using Doopec.Dds.Topic;

namespace Doopec.Rtps.Discovery

{

public class SPDPTopicBuiltinTopicData : TopicBuiltinTopicDataImpl

{

public const string TOPIC\_TOPIC = "DCPSTopic,";

public SPDPTopicBuiltinTopicData()

{

this.topicName = TOPIC\_TOPIC;

}

}

}

## ANEXO A.2: CÓDIGO FUENTE TRANSPORTE UDP

### IReceiver.cs

using Rtps.Structure.Types;

namespace Doopec.Utils.Transport

{

/// <summary>

/// Receiver will be used to receive packets from the source. Typically, source is from the network, but

/// it can be anything. Like memory, file etc.

/// </summary>

public interface IReceiver

{

/// <summary>

/// Gets the locator associated with this Receiver. This locator will be transmitted

/// to remote participants.

/// </summary>

/// <returns></returns>

Locator Locator { get; }

/// <summary>

/// Gets the participantId associated with this receiver. During creation of receiver,

/// participantId may be given as -1, indicating that provider should generate one.

/// This method returns the Value assigned by the provider.

/// </summary>

GUID ParticipantId { get; }

void Start();

/// <summary>

/// Close this Receiver

/// </summary>

void Close();

}

}

### ITransmitter.cs

using Rtps.Messages;

using Rtps.Structure.Types;

namespace Doopec.Utils.Transport

{

/// <summary>

/// Transmitter is used to deliver messages to destination.

/// </summary>

public interface ITransmitter

{

/// <summary>

/// Gets the locator associated with this Receiver. This locator will be transmitted

/// to remote participants.

/// </summary>

/// <returns></returns>

Locator Locator { get; }

/// <summary>

/// Gets the participantId associated with this receiver. During creation of receiver,

/// participantId may be given as -1, indicating that provider should generate one.

/// This method returns the Value assigned by the provider.

/// </summary>

GUID ParticipantId { get; }

/// <summary>

/// Sends a Message to destination.

/// </summary>

/// <param name="msg">Message to send</param>

/// <returns>true, if an overflow occured.</returns>

void SendMessage(Message msg);

void Start();

/// <summary>

/// Close this Writer

/// </summary>

void Close();

}

}

### UDPReceiver.cs

using Doopec.Rtps.Encoders;

using log4net;

using Mina.Core.Session;

using Mina.Filter.Codec;

using Mina.Transport.Socket;

using Rtps.Messages;

using Rtps.Structure.Types;

using System;

using System.Net;

using System.Net.Sockets;

using System.Reflection;

namespace Doopec.Utils.Transport

{

/// <summary>

/// Provides data for <see cref="IoSession"/>'s message receive/sent event.

/// </summary>

public class RTPSMessageEventArgs : IoSessionEventArgs

{

private readonly Message \_message;

/// <summary>

/// </summary>

public RTPSMessageEventArgs(IoSession session, Message message)

: base(session)

{

\_message = message;

}

/// <summary>

/// Gets the associated message.

/// </summary>

public Message Message

{

get { return \_message; }

}

}

/// <summary>

/// This class receives UDP packets from the network.

/// </summary>

public class UDPReceiver : IReceiver, IDisposable

{

private static readonly ILog log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

private readonly int bufferSize;

private readonly Locator locator;

private GUID participantId;

private AsyncDatagramAcceptor acceptor;

public event EventHandler<RTPSMessageEventArgs> MessageReceived;

public bool IsDiscovery { get; set; }

public UDPReceiver(Uri uri, int bufferSize)

{

this.bufferSize = bufferSize;

var addresses = System.Net.Dns.GetHostAddresses(uri.Host);

int port = (uri.Port < 0 ? 0 : uri.Port);

if (addresses != null && addresses.Length >= 1)

this.locator = new Locator(addresses[0], port);

}

public UDPReceiver(Locator locator, int bufferSize)

{

this.bufferSize = bufferSize;

this.locator = locator;

}

public Locator Locator

{

get { return locator; }

}

public GUID ParticipantId

{

get { return participantId; }

set { participantId = value; }

}

public void Start()

{

if (locator == null)

throw new ApplicationException();

IPEndPoint ep = new IPEndPoint(locator.SocketAddress, locator.Port);

bool isMultiCastAddr;

if (ep.AddressFamily == AddressFamily.InterNetwork) //IP v4

{

byte byteIp = ep.Address.GetAddressBytes()[0];

isMultiCastAddr = (byteIp >= 224 && byteIp < 240) ? true : false;

}

else if (ep.AddressFamily == AddressFamily.InterNetworkV6)

{

isMultiCastAddr = ep.Address.IsIPv6Multicast;

}

else

{

throw new NotImplementedException("Address family not supported yet: " + ep.AddressFamily);

}

if (isMultiCastAddr)

{

acceptor = new AsyncDatagramAcceptor();

// Define a MulticastOption object specifying the multicast group

// address and the local IPAddress.

// The multicast group address is the same as the address used by the client.

MulticastOption mcastOption = new MulticastOption(locator.SocketAddress, IPAddress.Any);

acceptor.SessionConfig.MulticastOption = mcastOption;

acceptor.SessionConfig.ExclusiveAddressUse = false;

acceptor.SessionConfig.ReuseAddress = true;

}

else

acceptor = new AsyncDatagramAcceptor();

//acceptor.FilterChain.AddLast("logger", new LoggingFilter());

acceptor.FilterChain.AddLast("RTPS", new ProtocolCodecFilter(new MessageCodecFactory()));

acceptor.SessionConfig.EnableBroadcast = true;

acceptor.ExceptionCaught += (s, e) =>

{

Console.WriteLine(e.Exception);

e.Session.Close(true);

};

acceptor.MessageReceived += (s, e) =>

{

Message msg = e.Message as Message;

if (MessageReceived != null)

MessageReceived(s, new RTPSMessageEventArgs(e.Session, msg));

//if (log.IsDebugEnabled)

//{

// log.DebugFormat("New Message has arrived from {0}", e.Session.RemoteEndPoint);

// log.DebugFormat("Message Header: {0}", msg.Header);

// foreach (var submsg in msg.SubMessages)

// {

// log.DebugFormat("SubMessage: {0}", submsg);

// if (submsg is Data)

// {

// Data d = submsg as Data;

// foreach (var par in d.InlineQos.Value)

// log.DebugFormat("InlineQos: {0}", par);

// }

// }

//}

};

acceptor.SessionCreated += (s, e) =>

{

log.Debug("Session created...");

};

acceptor.SessionOpened += (s, e) =>

{

log.Debug("Session opened...");

};

acceptor.SessionClosed += (s, e) =>

{

log.Debug("Session closed...");

};

acceptor.SessionIdle += (s, e) =>

{

log.Debug("Session idle...");

};

if (isMultiCastAddr)

acceptor.Bind(new IPEndPoint(IPAddress.Any, locator.Port));

else

acceptor.Bind(new IPEndPoint(locator.SocketAddress, locator.Port));

log.DebugFormat("Listening on udp://{0}:{1} for {2}", locator.SocketAddress, locator.Port, IsDiscovery ? "IsDiscovery traffic" : "user traffic");

}

public void Close()

{

if (acceptor != null)

{

log.DebugFormat("Closing {0}", acceptor.LocalEndPoint);

acceptor.Dispose();

}

}

public void Dispose()

{

if (acceptor != null)

acceptor.Dispose();

}

}

}

### UDPTransmitter.cs

using Doopec.Rtps.Encoders;

using log4net;

using Mina.Core.Future;

using Mina.Core.Service;

using Mina.Core.Session;

using Mina.Filter.Codec;

using Mina.Transport.Socket;

using Rtps.Messages;

using Rtps.Structure.Types;

using System;

using System.Net;

using System.Net.Sockets;

using System.Reflection;

namespace Doopec.Utils.Transport

{

public class UDPTransmitter : ITransmitter, IDisposable

{

private static readonly ILog log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

private readonly Locator locator;

private GUID participantId;

private AsyncDatagramConnector connector;

private int bufferSize;

private IoSession session;

public bool IsDiscovery { get; set; }

public Locator Locator

{

get { return locator; }

}

public GUID ParticipantId

{

get { return participantId; }

set { participantId = value; }

}

public UDPTransmitter(Uri uri, int bufferSize)

{

var addresses = System.Net.Dns.GetHostAddresses(uri.Host);

int port = (uri.Port < 0 ? 0 : uri.Port);

if (addresses != null && addresses.Length >= 1)

this.locator = new Locator(addresses[0], port);

}

/// <summary>

/// Constructor for UDPTransmitter.

/// </summary>

/// <param name="locator">Locator where the messages will be sent.</param>

/// <param name="bufferSize">Size of the buffer that will be used to Write messages.</param>

public UDPTransmitter(Locator locator, int bufferSize)

{

this.locator = locator;

this.bufferSize = bufferSize;

}

public void Start()

{

IPEndPoint ep = new IPEndPoint(locator.SocketAddress, locator.Port);

bool isMultiCastAddr;

if (ep.AddressFamily == AddressFamily.InterNetwork) //IP v4

{

byte byteIp = ep.Address.GetAddressBytes()[0];

isMultiCastAddr = (byteIp >= 224 && byteIp < 240) ? true : false;

}

else if (ep.AddressFamily == AddressFamily.InterNetworkV6)

{

isMultiCastAddr = ep.Address.IsIPv6Multicast;

}

else

{

throw new NotImplementedException("Address family not supported yet: " + ep.AddressFamily);

}

connector = new AsyncDatagramConnector();

connector.FilterChain.AddLast("RTPS", new ProtocolCodecFilter(new MessageCodecFactory()));

if (isMultiCastAddr)

{

// Set the local IP address used by the listener and the sender to

// exchange multicast messages.

connector.DefaultLocalEndPoint = new IPEndPoint(IPAddress.Any, 0);

// Define a MulticastOption object specifying the multicast group

// address and the local IP address.

// The multicast group address is the same as the address used by the listener.

MulticastOption mcastOption = new MulticastOption(locator.SocketAddress, IPAddress.Any);

connector.SessionConfig.MulticastOption = mcastOption;

// Call Connect() to force binding to the local IP address,

// and get the associated multicast session.

IoSession session = connector.Connect(ep).Await().Session;

}

connector.ExceptionCaught += (s, e) =>

{

log.Error(e.Exception);

};

connector.MessageReceived += (s, e) =>

{

log.Debug("Session recv...");

};

connector.MessageSent += (s, e) =>

{

log.Debug("Session sent...");

};

connector.SessionCreated += (s, e) =>

{

log.Debug("Session created...");

};

connector.SessionOpened += (s, e) =>

{

log.Debug("Session opened...");

};

connector.SessionClosed += (s, e) =>

{

log.Debug("Session closed...");

};

connector.SessionIdle += (s, e) =>

{

log.Debug("Session idle...");

};

IConnectFuture connFuture = connector.Connect(new IPEndPoint(locator.SocketAddress, locator.Port));

connFuture.Await();

connFuture.Complete += (s, e) =>

{

IConnectFuture f = (IConnectFuture)e.Future;

if (f.Connected)

{

log.Debug("...connected");

session = f.Session;

}

else

{

log.Warn("Not connected...exiting");

}

};

}

/\*\*

\* Sends a Message to a Locator of this UDPWriter.

\* If an overflow occurs during writing of Message, only submessages that

\* were succesfully written will be sent.

\*

\* @param m Message to send

\* @return true, if Message did not fully fit into buffer of this UDPWriter

\*/

public void SendMessage(Message m)

{

if (session != null)

{

session.Write(m);

}

}

public void Close()

{

session.Close(false);

}

public void Dispose()

{

this.Close();

}

}

}

## ANEXO A.3: CÓDIGO FUENTE DDS (INTERACCIÓN CON RTPS)