УТВЕРЖДЕН
RU.17701729.503200-01 12 01-1-ЛУ

ПРОГРАММА СИНТЕЗА ГИБРИДНЫХ UML ДИАГРАММ ПО ЖУРНАЛАМ СОБЫТИЙ

Текст программы

RU.17701729.504900-01 12 01-1

Листов 172

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.504900-01 12				

2017

СОДЕРЖАНИЕ

1. ТЕКСТ ПРОГРАММЫ

Программа состоит из 77 классов.

Текст программы на исходном языке находится в директории Документация/Текст программы.pdf на носителе информации типа компакт-диск в связи с большим количеством строк кода.

1.1. Модуль EventLogOM

1.1.1. EventLogLib

EventAttrFilterType.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Xidv.EventLogLib
{
    /// <summary>
    /// Enumeration for event attribute filter types
    /// </summary>
    public enum EventAttrFilterType
    {
        All, InListOnly, NotInListOnly
    }
}
```

EventsOrdering.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Xidv.EventLogLib
{
    /// <summary>
    /// Enumeration of order types
    /// </summary>
    public enum EventsOrdering
    {
        /// <summary>
        /// No order
        /// </summary>
        None=0,
        /// <summary>
        /// Ascending order
        /// </summary>
        Asc=1,
        /// <summary>
        /// Descending order
        /// </summary>
        Dsc=2
    }
}
```

IAttribute.cs

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Xidv.EventLogLib
{
    /// <summary>
    /// Declaration of event attribute
    /// </summary>
    public interface IAttribute
    {
        bool Equals(System.Object obj);
        string ToString();
    }
}
```

IAttributes.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Xidv.EventLogLib
{
    /// <summary>
    /// Declaration of event log attributes
    /// </summary>
    public interface IAttributes : IEnumerable<IAttribute>, IList<IAttribute>
    {
    }
}
```

IEvent.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Xidv.EventLogLib
{
    /// <summary>
    /// Declaration of event
    /// </summary>
    public interface IEvent
    {
        /// <summary>
        /// Event attributes
        /// </summary>
        IAttributes Attributes
        {
            get;
            set;
        }
```

```
    }
}
```

IEvents.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Xidv.EventLogLib
{
    /// <summary>
    /// Declaration of log events
    /// </summary>
    public interface IEvents : IEnumerable<IEvent>, IList<IEvent>
    {
    }
}
```

ILog.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Xidv.EventLogLib
{
    /// <summary>
    /// Declare methods for log
    /// </summary>
    public interface ILog
    {
        /// <summary>
        /// Attributes of log
        /// </summary>
        IAttributes Attributes
        {
            get;
            set;
        }
        /// <summary>
        /// The order of events in log
        /// </summary>
        EventsOrdering AutoEventsOrder
        {
            get;
            set;
        }
        /// <summary>
        /// Case attributes name
        /// </summary>
        string CaseAttrName
        {
```

```
            get;
            set;
        }
        /// <summary>
        /// Determines which attributes of events should be extracted: All, only those which is
in a list, only those which is not in a list
        /// </summary>
        EventAttrFilterType EventAttributesFilter
        {
            get;
            set;
        }
        /// <summary>
        /// Determines a list of event attributes used in extracting procedure.
        /// </summary>
        List<string> EventAttributesFilterList
        {
            get;
            set;
        }
        /// <summary>
        /// Describes the name of an attribute playing a role of Event ID for the whole event log
        /// </summary>
        string ActivityAttrName
        {
            get;
            set;
        }
        /// <summary>
        /// Save whether log is prepared
        /// </summary>
        bool Prepared
        {
            get;
            set;
        }
        /// <summary>
        /// Timestamp attributes name
        /// </summary>
        string TimestampAttrName
        {
            get;
            set;
        }
        /// <summary>
        /// Traces of log
        /// </summary>
        ITraces Traces
        {
            get;
            set;
        }
        /// <summary>
        /// Prepare log
        /// </summary>
        void Prepare();
    }
}
```

ITrace.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Xidv.EventLogLib
{
    /// <summary>
    /// Declaration of trace
    /// </summary>
    public interface ITrace
    {
        /// <summary>
        /// Attributes of trace
        /// </summary>
        IAttributes Attributes
        {
            get;
            set;
        }
        /// <summary>
        /// Case ID of trace
        /// </summary>
        IAttribute CaseID
        {
            get;
            set;
        }
        /// <summary>
        /// Events of trace
        /// </summary>
        IEvents Events
        {
            get;
            set;
        }
        /// <summary>
        /// Events order in trace
        /// </summary>
        EventsOrdering EventsOrder
        {
            get;
            set;
        }
    }
}
```

ITraces.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;

namespace Xidv.EventLogLib
{
    /// <summary>
    /// Declaration of traces
    /// </summary>
    public interface ITraces : IEnumerable<ITrace>, IList<ITrace>
    {
    }
}
```

ListAttributes.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Xidv.EventLogLib;

namespace Xidv.EventLogLib
{
    public class ListAttributes : IAttributes
    {
        private List<IAttribute> attributes;

        public ListAttributes()
        {
            attributes = new List<IAttribute>();
        }

        public IEnumerator<IAttribute> GetEnumerator()
        {
            return attributes.GetEnumerator();
        }

        private IEnumerator GetEnumerator1()
        {
            return this.GetEnumerator();
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            return GetEnumerator1();
        }

        public int IndexOf(IAttribute item)
        {
            return attributes.IndexOf(item);
            //throw new NotImplementedException();
        }

        public void Insert(int index, IAttribute item)
        {
            attributes.Insert(index, item);
            //throw new NotImplementedException();
```

```
}

public void RemoveAt(int index)
{
    attributes.RemoveAt(index);
    //throw new NotImplementedException();
}

public IAttribute this[int index]
{
    get
    {
        return attributes[index];
        //throw new NotImplementedException();
    }
    set
    {
        attributes[index] = value;
        //throw new NotImplementedException();
    }
}

public void Add(IAttribute item)
{
    attributes.Add(item);
    //throw new NotImplementedException();
}

public void Clear()
{
    attributes.Clear();
    //throw new NotImplementedException();
}

public bool Contains(IAttribute item)
{
    return attributes.Contains(item);
    //throw new NotImplementedException();
}

public void CopyTo(IAttribute[] array, int arrayIndex)
{
    attributes.CopyTo(array, arrayIndex);
    //throw new NotImplementedException();
}

public int Count
{
    get
    {
        return attributes.Count;
        //throw new NotImplementedException();
    }
}

public bool Remove(IAttribute item)
{
    return attributes.Remove(item);
```

```
            //throw new NotImplementedException();
        }

        public bool IsReadOnly
        {
            get
            {
                return false;
            }
        }
    }
}
```

ListEvents.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Xidv.EventLogLib;

namespace Xidv.EventLogLib
{
    public class ListEvents : IEvents
    {
        private List<IEvent> events;

        public ListEvents()
        {
            events = new List<IEvent>();
        }

        public IEnumerator<IEvent> GetEnumerator()
        {
            return events.GetEnumerator();
        }

        private IEnumerator GetEnumerator1()
        {
            return this.GetEnumerator();
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            return GetEnumerator1();
        }

        public int IndexOf(IEvent item)
        {
            return events.IndexOf(item);
            //throw new NotImplementedException();
        }

        public void Insert(int index, IEvent item)
        {
            events.Insert(index, item);
```

```
        //throw new NotImplementedException();
    }

    public void RemoveAt(int index)
    {
        events.RemoveAt(index);
        //throw new NotImplementedException();
    }

    public IEvent this[int index]
    {
        get
        {
            return events[index];
            //throw new NotImplementedException();
        }
        set
        {
            events[index] = value;
            //throw new NotImplementedException();
        }
    }

    public void Add(IEvent item)
    {
        events.Add(item);
        //throw new NotImplementedException();
    }

    public void Clear()
    {
        events.Clear();
        //throw new NotImplementedException();
    }

    public bool Contains(IEvent item)
    {
        return events.Contains(item);
        //throw new NotImplementedException();
    }

    public void CopyTo(IEvent[] array, int arrayIndex)
    {
        events.CopyTo(array, arrayIndex);
        //throw new NotImplementedException();
    }

    public int Count
    {
        get
        {
            return events.Count;
            //throw new NotImplementedException();
        }
    }

    public bool Remove(IEvent item)
    {
```

```
            return events.Remove(item);
            //throw new NotImplementedException();
        }

        public bool IsReadOnly
        {
            get
            {
                return false;
            }
        }
    }
}
```

ListTraces.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Xidv.EventLogLib;

namespace Xidv.EventLogLib
{
    public class ListTraces : ITraces
    {
        private List<ITrace> traces;

        public ListTraces()
        {
            traces = new List<ITrace>();
        }

        public IEnumerator<ITrace> GetEnumerator()
        {
            return traces.GetEnumerator();
        }

        private IEnumerator GetEnumerator1()
        {
            return this.GetEnumerator();
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            return GetEnumerator1();
        }

        public int IndexOf(ITrace item)
        {
            return traces.IndexOf(item);
            //throw new NotImplementedException();
        }

        public void Insert(int index, ITrace item)
        {
```

```
        traces.Insert(index, item);
        //throw new NotImplementedException();
    }

    public void RemoveAt(int index)
    {
        traces.RemoveAt(index);
        //throw new NotImplementedException();
    }

    public ITrace this[int index]
    {
        get
        {
            return traces[index];
            //throw new NotImplementedException();
        }
        set
        {
            traces[index] = value;
            //throw new NotImplementedException();
        }
    }

    public void Add(ITrace item)
    {
        traces.Add(item);
        //throw new NotImplementedException();
    }

    public void Clear()
    {
        traces.Clear();
        //throw new NotImplementedException();
    }

    public bool Contains(ITrace item)
    {
        return traces.Contains(item);
        //throw new NotImplementedException();
    }

    public void CopyTo(ITrace[] array, int arrayIndex)
    {
        traces.CopyTo(array, arrayIndex);
        //throw new NotImplementedException();
    }

    public int Count
    {
        get
        {
            return traces.Count;
            //throw new NotImplementedException();
        }
    }

    public bool Remove(ITrace item)
```

```
        {
            return traces.Remove(item);
            //throw new NotImplementedException();
        }

        public bool IsReadOnly
        {
            get
            {
                return false;
            }
        }
    }
}
```

ListBooleanAttr.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Xidv.EventLogLib
{
    /// <summary>
    /// Adapter class for bool with realisation of IAttribute
    /// Bool attribute
    /// </summary>
    public class BooleanAttr : IAttribute
    {
        private bool boolean;

        public bool Boolean
        {
            get { return boolean; }
            set { boolean = value; }
        }

        public BooleanAttr(bool _boolean)
        {
            boolean = _boolean;
        }

        public override string ToString()
        {
            return boolean.ToString();
        }

        public override bool Equals(System.Object obj)
        {
            // If parameter is null return false.
            if (obj == null)
            {
                return false;
            }
```

```
            BooleanAttr otherAttr = obj as BooleanAttr;
            if (boolean == otherAttr.Boolean)
                return true;
            else
                return false;
        }

        public static bool operator ==(BooleanAttr a, BooleanAttr b)
        {
            // If both are null, or both are same instance, return true.
            if (System.Object.ReferenceEquals(a, b))
            {
                return true;
            }

            // If one is null, but not both, return false.
            if (((object)a == null) || ((object)b == null))
            {
                return false;
            }

            // Return true if the fields match:
            return a.Boolean == b.Boolean;
        }

        public static bool operator !=(BooleanAttr a, BooleanAttr b)
        {
            return !(a == b);
        }
    }
}
```

DoubleAttr.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Xidv.EventLogLib
{
    /// <summary>
    /// Adapter class for double with realisation of IAttribute
    /// Double attribute
    /// </summary>
    public class DoubleAttr : IAttribute
    {
        private double num;

        public double Num
        {
            get { return num; }
            set { num = value; }
        }

        public DoubleAttr(double _num)
```

```
        {
            num = _num;
        }

        public override string ToString()
        {
            return num.ToString();
        }

        public override bool Equals(System.Object obj)
        {
            // If parameter is null return false.
            if (obj == null)
            {
                return false;
            }

            DoubleAttr otherAttr = obj as DoubleAttr;
            if (num == otherAttr.Num)
                return true;
            else
                return false;
        }

        public static bool operator ==(DoubleAttr a, DoubleAttr b)
        {
            // If both are null, or both are same instance, return true.
            if (System.Object.ReferenceEquals(a, b))
            {
                return true;
            }

            // If one is null, but not both, return false.
            if (((object)a == null) || ((object)b == null))
            {
                return false;
            }

            // Return true if the fields match:
            return a.Num == b.Num;
        }

        public static bool operator !=(DoubleAttr a, DoubleAttr b)
        {
            return !(a == b);
        }
    }
}
```

IntAttr.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace Xidv.EventLogLib
{
    /// <summary>
    /// Adapter class for int with realisation of IAttribute
    /// Int attribute
    /// </summary>
    public class IntAttr : IAttribute
    {
        private int num;

        public int Num
        {
            get { return num; }
            set { num = value; }
        }

        public IntAttr(int _num)
        {
            num = _num;
        }

        public override string ToString()
        {
            return num.ToString();
        }

        public override bool Equals(System.Object obj)
        {
            // If parameter is null return false.
            if (obj == null)
            {
                return false;
            }

            IntAttr otherAttr = obj as IntAttr;
            if (num == otherAttr.Num)
                return true;
            else
                return false;
        }

        public static bool operator ==(IntAttr a, IntAttr b)
        {
            // If both are null, or both are same instance, return true.
            if (System.Object.ReferenceEquals(a, b))
            {
                return true;
            }

            // If one is null, but not both, return false.
            if (((object)a == null) || ((object)b == null))
            {
                return false;
            }

            // Return true if the fields match:
            return a.Num == b.Num;
        }
```

```
        public static bool operator !=(IntAttr a, IntAttr b)
        {
            return !(a == b);
        }
    }
}
```

StringAttr.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Xidv.EventLogLib
{
    /// <summary>
    /// Adapter class for string with realisation of IAttribute
    /// String attribute
    /// </summary>
    public class StringAttr : IAttribute
    {
        private string str;

        public string Str
        {
            get { return str; }
            set { str = value; }
        }

        public StringAttr(string _str)
        {
            str = _str;
        }

        public override string ToString()
        {
            return str;
        }

        public override bool Equals(System.Object obj)
        {
            // If parameter is null return false.
            if (obj == null)
            {
                return false;
            }

            StringAttr otherAttr = obj as StringAttr;
            if (str == otherAttr.Str)
                return true;
            else
                return false;
        }
```

```
        public static bool operator ==(StringAttr a, StringAttr b)
        {
            // If both are null, or both are same instance, return true.
            if (System.Object.ReferenceEquals(a, b))
            {
                return true;
            }

            // If one is null, but not both, return false.
            if (((object)a == null) || ((object)b == null))
            {
                return false;
            }

            // Return true if the fields match:
            return a.Str == b.Str;
        }

        public static bool operator !=(StringAttr a, StringAttr b)
        {
            return !(a == b);
        }
    }
}
```

TimestampAttr.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Xidv.EventLogLib
{
    /// <summary>
    /// Adapter class for timestamp with realisation of IAttribute
    /// Timestamp attribute
    /// </summary>
    public class TimestampAttr : IAttribute
    {
        private TimeSpan timestamp;

        public TimeSpan Timestamp
        {
            get { return timestamp; }
            set { timestamp = value; }
        }

        public TimestampAttr(TimeSpan _timestamp)
        {
            timestamp = _timestamp;
        }

        public override string ToString()
        {
            return timestamp.ToString();
```

```
        }

        public override bool Equals(System.Object obj)
        {
            // If parameter is null return false.
            if (obj == null)
            {
                return false;
            }

            TimestampAttr otherAttr = obj as TimestampAttr;
            if (timestamp == otherAttr.Timestamp)
                return true;
            else
                return false;
        }

        public static bool operator ==(TimestampAttr a, TimestampAttr b)
        {
            // If both are null, or both are same instance, return true.
            if (System.Object.ReferenceEquals(a, b))
            {
                return true;
            }

            // If one is null, but not both, return false.
            if (((object)a == null) || ((object)b == null))
            {
                return false;
            }

            // Return true if the fields match:
            return a.Timestamp == b.Timestamp;
        }

        public static bool operator !=(TimestampAttr a, TimestampAttr b)
        {
            return !(a == b);
        }
    }
}
```

1.1.2. CSVEventLog

CSVLog.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xidv.EventLogLib;
using Microsoft.VisualBasic.FileIO;
using System.IO;

namespace Xidv.EventLogLib.Csv
{
    public class CSVLog : ILog
    {
        /// <summary>
        /// Determines if events should be cached after first extracting from test string of
corresponding line of the CSV file.
        /// If it is true, the extracted Event is stored by its ref in CSVLogTraceDescriptor
structure after first usage.
        /// Otherwise, the event will be loosed.
        /// </summary>
        private bool cacheEvents;
        /// <summary>
        /// Collection of attribute names.
        /// If IsHeadlinePresent property is true, the collection is retrived from first line of
log normally.
        /// Otherwise, a user has to determine all the names manually.
        /// If the elements of the collection are extracted automatically, the user is still
possible to rename every attribute name.
        /// </summary>
        private string[] eventAttrNames;
        /// <summary>
        /// Global at log level collection of map "line number" -> "Event".
        /// Used to store extracted events isolated from containing traces.
        /// The collection is being filled only if CacheEvents flag is set.
        /// Can be useful if either case id or event id attribute is changed. It leads to
recreating all traces,
        /// but instead of repeated extracting of events, one can use globally stored events.
        /// </summary>
        private IEvent[] eventsLineNumsMap;
        private Dictionary<IAttribute, List<CSVLogEventDescriptor>> descriptorMap;
        /// <summary>
        /// Determines if a headline is presented in log file.
        /// Is so, calling Prepare() extracts first line and parse it as a collection of
attribute names.
        /// </summary>
        private bool isHeadlinePresent;
        /// <summary>
        /// Determines if all events should be extracted during log preparation process.
        /// If the flag is set, it leads to setting CacheEvents on too.
        /// Otherwise, every event is extract only at first usage.
        /// </summary>
        private bool prefetchEvents;
        /// <summary>
```

```
        /// Attribute separator
        /// </summary>
        private char separator;
        private IAttributes attributes;
        private EventsOrdering autoEventsOrder;
        private string caseAttrName;
        private EventAttrFilterType eventAttributesFilter;
        private List<string> eventAttributesFilterList;
        /// <summary>
        /// Describes the name of an attribute playing a role of Event ID for the whole event
log.
        /// </summary>
        private string activityAttrName;
        private bool prepared;
        private string timestampAttrName;
        private ITraces traces;
        /// <summary>
        /// Designated a Stream that representing CSV log as a collection of string lines, one
event per line.
        /// </summary>
        private string fileName;
        /// <summary>
        /// Indexes of attributes from eventAttributesFilterList.
        /// </summary>
        private int[] eventAttributesFilterIndexes;

        public Dictionary<IAttribute, List<CSVLogEventDescriptor>> DescriptorMap
        {
            get { return descriptorMap; }
            set { descriptorMap = value; }
        }

        protected string FileName
        {
            get { return fileName; }
            set { fileName = value; }
        }

        public IAttributes Attributes
        {
            get { return attributes; }
            set { attributes = value; }
        }

        public EventsOrdering AutoEventsOrder
        {
            get { return autoEventsOrder; }
            set { autoEventsOrder = value; }
        }

        public string CaseAttrName
        {
            get { return caseAttrName; }
            set { caseAttrName = value; }
        }

        public EventAttrFilterType EventAttributesFilter
        {
```

```
        get { return eventAttributesFilter; }
        set { eventAttributesFilter = value; }
    }

    public List<string> EventAttributesFilterList
    {
        get { return eventAttributesFilterList; }
        set { eventAttributesFilterList = value; }
    }

    public string ActivityAttrName
    {
        get { return activityAttrName; }
        set { activityAttrName = value; }
    }

    public bool Prepared
    {
        get { return prepared; }
        set { prepared = value; }
    }

    public string TimestampAttrName
    {
        get { return timestampAttrName; }
        set { timestampAttrName = value; }
    }

    public ITraces Traces
    {
        get
        {
            ExtractTraces();
            return traces;
        }
        set { traces = value; }
    }

    public bool CacheEvents
    {
        get { return cacheEvents; }
        set { cacheEvents = value; }
    }

    public string[] EventAttrNames
    {
        get { return eventAttrNames; }
        set { eventAttrNames = value; }
    }

    public IEvent[] EventsLineNumsMap
    {
        get { return eventsLineNumsMap; }
        set { eventsLineNumsMap = value; }
    }

    public bool IsHeadlinePresent
    {
```

```
        get { return isHeadlinePresent; }
        set { isHeadlinePresent = value; }
    }

    public bool PrefetchEvents
    {
        get { return prefetchEvents; }
        set { prefetchEvents = value; }
    }

    public char Separator
    {
        get { return separator; }
        set { separator = value; }
    }

    /// Constructors:

    /// <summary>
    /// Constructor, accept file name as an argument to extract log from csv file
    /// Set separator as ";"
    /// </summary>
    /// <param name="fileName"></param>
    public CSVLog(string _fileName)
    {
        fileName = _fileName;
        separator = ';';
        autoEventsOrder = EventsOrdering.None;
        eventAttributesFilter = EventAttrFilterType.All;
        eventAttributesFilterList = new List<string>();
    }
    /// <summary>
    /// Constructor, accept file name as an argument to extract log from csv file
    /// and separator for csv file
    /// </summary>
    /// <param name="fileName"></param>
    /// <param name="_separator"></param>
    public CSVLog(string _fileName, char _separator)
    {
        fileName = _fileName;
        separator = _separator;
        autoEventsOrder = EventsOrdering.None;
        eventAttributesFilter = EventAttrFilterType.All;
        eventAttributesFilterList = new List<string>();
    }

    public void Prepare()
    {
        if (CaseAttrName == "")
            throw new InvalidOperationException("Case attribute name is not set");
        if (ActivityAttrName == "")
            throw new InvalidOperationException("Event attribute name is not set");
        if (TimestampAttrName == "")
            throw new InvalidOperationException("Timestamp attribute name is not set");
        if (!IsHeadlinePresent)
            ExtractAttrNamesFromHeadline();

        if (eventAttributesFilter == EventAttrFilterType.All)
```

```
{
    eventAttributesFilterIndexes = new int[eventAttrNames.Length];
    for (int i = 0; i < eventAttrNames.Length; i++)
    {
        eventAttributesFilterIndexes[i] = i;
    }
}
if (eventAttributesFilter == EventAttrFilterType.InListOnly)
{
    eventAttributesFilterIndexes = new int[eventAttributesFilterList.Count];
    for (int i = 0, j = 0; i < eventAttrNames.Length; i++)
    {
        if (eventAttributesFilterList.Contains(eventAttrNames[i]))
        {
            eventAttributesFilterIndexes[j] = i;
            j++;
        }
    }
}
if (eventAttributesFilter == EventAttrFilterType.NotInListOnly)
{
    eventAttributesFilterIndexes = new int[eventAttributesFilterList.Count];
    for (int i = 0, j = 0; i < eventAttrNames.Length; i++)
    {
        if (!eventAttributesFilterList.Contains(eventAttrNames[i]))
        {
            eventAttributesFilterIndexes[j] = i;
            j++;
        }
    }
}

//traces = new ListTraces();
descriptorMap = new Dictionary<IAttribute, List<CSVLogEventDescriptor>>();
if (cacheEvents)
{
    int num = 0;
    using (TextFieldParser parser = new TextFieldParser(fileName))
    {
        parser.TextFieldType = FieldType.Delimited;
        parser.SetDelimiters(Separator.ToString());
        while (!parser.EndOfData)
        {
            parser.ReadFields();
            num++;
        }
    }
    eventsLineNumsMap = new IEvent[num];
}

using (TextFieldParser parser = new TextFieldParser(fileName))
{
    parser.TextFieldType = FieldType.Delimited;
    parser.SetDelimiters(Separator.ToString());
    parser.ReadFields();
    while (!parser.EndOfData)
    {
        string[] fields = parser.ReadFields();
```

```
                    MessageBox.Show("Select element in the attribute list", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                else
                    MessageBox.Show("Необходимо выбрать элемент в списке атрибутов", "Ошибка
добавления", MessageBoxButtons.OK, MessageBoxIcon.Error);
            else
            {
                allAttrListBox.Items.Add(el);
                lifelinesAttrListBox.Items.Remove(el);
            }
        }

        private void OkButton_Click(object sender, EventArgs e)
        {
            if (CheckAllData())
            {
                _log.Prepare();

                try
                {
                    UMLSDMiner umlsdminer = new UMLSDMiner(_log,
lifelinesAttrListBox.Items.Cast<String>().ToList(),
paramsAttrListBox.Items.Cast<String>().ToList(), true, false, _rqRsIndex, rqTextBox.Text,
rsTextBox.Text);
                    _diagram = umlsdminer.Mine();
                    if (isEnglish)
                        MessageBox.Show("Model is build. You can save it to XMI-file", "Message",
MessageBoxButtons.OK);
                    else
                        MessageBox.Show("Модель построена, вы можете ее сохранить в XMI-файл",
"Сообщение", MessageBoxButtons.OK);
                }
                catch (Exception ex)
                {
                    if (isEnglish)
                        MessageBox.Show("The program cannot build a diagram for your log.
Message: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    else
                        MessageBox.Show("Программа не смогла построить диаграмму для вашего лога.
Сообщение: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }

            }
        }

        private bool CheckAllData()
        {
            if (lifelinesAttrListBox.Items.Count == 0 && paramsAttrListBox.Items.Count == 0)
            {
                if (isEnglish)
                    MessageBox.Show("Add attributes to lists of lifeline and message parameter
attributes", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                else
                    MessageBox.Show("Необходимо добавить атрибуты в списки атрибутов линий жизни
и параметров сообщений", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return false;
            }
            if (lifelinesAttrListBox.Items.Count == 0)
```

```
            {
                if (isEnglish)
                    MessageBox.Show("Add attributes to lists of lifeline attributes", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                else
                    MessageBox.Show("Необходимо добавить атрибуты в список атрибутов линий
жизни", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return false;
            }
            if (paramsAttrListBox.Items.Count == 0)
            {
                if (isEnglish)
                    MessageBox.Show("Add attributes to lists of message parameter attributes",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                else
                    MessageBox.Show("Необходимо добавить атрибуты в список атрибутов параметров
сообщений", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return false;
            }
            if (_rqRsIndex == -1)
            {
                if (isEnglish)
                    MessageBox.Show("Select an attribute which sets a value of call/response
(REQ/RES)", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                else
                    MessageBox.Show("Необходимо выбрать атрибут, задающий значение
вызова/возврата (REQ/RES)", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return false;
            }
            if (rqTextBox.Text == "")
            {
                if (isEnglish)
                    MessageBox.Show("Set an attribute value for call", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                else
                    MessageBox.Show("Необходимо установить значение атрибута для вызова",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return false;
            }
            if (rsTextBox.Text == "")
            {
                if (isEnglish)
                    MessageBox.Show("Set an attribute value for response", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                else
                    MessageBox.Show("Необходимо установить значение атрибута для возврата",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return false;
            }
            return true;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            saveFileDialog1 = new SaveFileDialog();
            saveFileDialog1.Filter = "XMI File|*.xml";
            saveFileDialog1.Title = "Сохранить модель";
            saveFileDialog1.FileName = "diagram1";
```

```
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                string filename = saveFileDialog1.FileName;
                try
                {
                    CoordUMLSDBuilder builder = new CoordUMLSDBuilder(_diagram);
                    builder.ExportToXMI();
                    builder.WriteToFile(filename);
                }
                catch (Exception ex)
                {
                    if (isEnglish)
                        MessageBox.Show("The program cannot export your diagram to XMI-file.
Message: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    else
                        MessageBox.Show("Программа не смогла экспортировать диаграмму в XMI-
формат. Сообщение: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }

        private void rqRsComboBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            _rqRsIndex = rqRsComboBox.SelectedIndex;
        }

        private CSVLog _log;
        private UMLSD _diagram;
        private int _rqRsIndex = -1;


        private void nextButton_Click(object sender, EventArgs e)
        {
            if (CheckAllData())
            {
                _log.Prepare();
                UMLSDMiner umlsdminer = new UMLSDMiner(_log,
lifelinesAttrListBox.Items.Cast<String>().ToList(),
paramsAttrListBox.Items.Cast<String>().ToList(), true, false, _rqRsIndex, rqTextBox.Text,
rsTextBox.Text);
                Form form = new ChooseDiagrTypeForm(_log, umlsdminer);
                form.FormClosed += new FormClosedEventHandler(form_FormClosed);
                form.Show();
                this.Hide();
            }
        }

        private void form_FormClosed(object sender, FormClosedEventArgs e)
        {
            this.Close();
        }

    }
}
```

ChooseDiagrTypeForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Xidv.EventLogLib;
using Xidv.EventLogLib.Csv;
using Xidv.UmlSdMiner;
using Xidv.UmlBehaviorDiagrams.UmlSd;
using Xidv.UmlBehaviorDiagramsExporter;
using Xidv.HybridUMLModelsMiner;
using Xidv.UmlAdMiner;
using Xidv.UmlBehaviorDiagrams.UmlAd;

namespace UMLModelsMiner
{
    public partial class ChooseDiagrTypeForm : Form
    {
        bool isEnglish = false;
        public ChooseDiagrTypeForm(CSVLog log, UMLSDMiner miner)
        {
            InitializeComponent();

            if (isEnglish)
            {
                this.Text = "UML Models Miner";
                this.label5.Text = "Attribute:";
                this.label6.Text = "Regular expression:";
                this.label2.Text = "Choose a trace";
                this.groupBox1.Text = "Diagram type";
                this.usualSDradioButton.Text = "Simple sequence diagram";
                this.hierarchicalRadioButton.Text = "Hierarchical";
                this.hybridRadioButton.Text = "Hybrid";
                this.umlADradioButton.Text = "Activity diagrams";
                this.buildButton.Text = "Build";
                this.nextButton.Text = "Next";
                this.saveButton.Text = "Save";
            }

            _log = log;
            _miner = miner;

            foreach (string attr in _log.EventAttrNames)
            {
                if (attr != _log.TimestampAttrName && attr != _log.CaseAttrName)
                {
                    attrComboBox.Items.Add(attr);
                }
            }
            attrComboBox.SelectedIndex = 0;

            if (isEnglish)
                tracesComboBox.Items.Add("All traces");
```

```
            else
                tracesComboBox.Items.Add("Все трассы");
            foreach (IAttribute attr in _log.DescriptorMap.Keys)
            {
                tracesComboBox.Items.Add(attr.ToString());
            }
            tracesComboBox.SelectedIndex = 0;
        }

        private void addButton_Click(object sender, EventArgs e)
        {
            if (regExprTextBox.Text != "")
            {
                string newRegExpr = attrComboBox.SelectedItem.ToString() + "=" +
regExprTextBox.Text;
                regExprListBox.Items.Add(newRegExpr);
                regExprTextBox.Text = "";
            }
            else
                if (isEnglish)
                    MessageBox.Show("Set a regular epression", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                else
                    MessageBox.Show("Необходимо заполнить поле для регулярного выражения",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

        private void delButton_Click(object sender, EventArgs e)
        {
            //int index = ;
            if (regExprListBox.SelectedIndex < 0)
                if (isEnglish)
                    MessageBox.Show("Select a regular expression to remove", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                else
                    MessageBox.Show("Необходимо выбрать регулярное выражение для удаления",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            else
                regExprListBox.Items.RemoveAt(regExprListBox.SelectedIndex);
        }

        private void buildButton_Click(object sender, EventArgs e)
        {
            if (!usualSDradioButton.Checked && !hierarchicalRadioButton.Checked &&
!hybridRadioButton.Checked && !umlADradioButton.Checked)
            {
                if (isEnglish)
                    MessageBox.Show("Select a diagram type", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                else
                    MessageBox.Show("Выберите тип диаграммы", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                return;
            }
            if (tracesComboBox.SelectedIndex > 0)
            {
                IAttribute caseId = null;
                foreach (IAttribute attr in _log.DescriptorMap.Keys)
```

```
                {
                    if (attr.ToString() == tracesComboBox.SelectedItem.ToString())
                        caseId = attr;
                }
                _miner.SingleCaseId = caseId;
                _miner.IsAlt = false;
            }
            _miner.RegularExpressions = regExprListBox.Items.Cast<String>().ToList();
            if (usualSDradioButton.Checked)
            {
                //try
                //{
                    _diagram = _miner.Mine();
                    _isReady = true;
                    if (isEnglish)
                        MessageBox.Show("Model is built. You can save it to XMI-file", "Message",
MessageBoxButtons.OK);
                    else
                        MessageBox.Show("Модель построена, вы можете ее сохранить в XMI-файл",
"Сообщение", MessageBoxButtons.OK);
                //}
                //catch (Exception ex)
                //{
                //    if (isEnglish)
                //        MessageBox.Show("The program cannot built a diagram for your log.
Message: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                //    else
                //        MessageBox.Show("Программа не смогла построить диаграмму для вашего
лога. Сообщение: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                //}
            }
            if (umlADradioButton.Checked)
            {
                _hybridMiner = new HybridUMLModelsMiner(_log, _miner);
                Dictionary<string, List<List<string>>> logsForADs = _hybridMiner.GetLogsForADs();
                try
                {
                    _adDiagrams = new List<UmlAdImpl>();
                    foreach (string l in logsForADs.Keys)
                    {
                        AlphaAlgorithm adMiner = new AlphaAlgorithm(logsForADs[l]);
                        UmlAdImpl adDiagram = adMiner.Mine();
                        adDiagram.DiagramName = l;
                        _adDiagrams.Add(adDiagram);
                    }
                    _isReady = true;
                    if (isEnglish)
                        MessageBox.Show("Models are built. You can save it to XMI-file",
"Message", MessageBoxButtons.OK);
                    else
                        MessageBox.Show("Модели построены, вы можете сохранить их в XMI-файл",
"Сообщение", MessageBoxButtons.OK);
                }
                catch (Exception ex)
                {
                    if (isEnglish)
                        MessageBox.Show("The program cannot built diagrams for your log. Message:
" + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
                else
                    MessageBox.Show("Программа не смогла построить диаграммы для вашего лога.
Сообщение: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
            if (hybridRadioButton.Checked)
            {
                _hybridMiner = new HybridUMLModelsMiner(_log, _miner);
                //_hybridMiner.PrepareLogs();
                UMLSD sdDiagram = _hybridMiner.Mine();

                if (isEnglish)
                    MessageBox.Show("Does not work yet");
                else
                    MessageBox.Show("Пока не работает");
            }

        }

        private void radioButton2_CheckedChanged(object sender, EventArgs e)
        {
            if (hierarchicalRadioButton.Checked)
            {
                nextButton.Visible = true;
                buildButton.Visible = false;
                saveButton.Visible = false;
            }
            else
            {
                nextButton.Visible = false;
                buildButton.Visible = true;
                saveButton.Visible = true;
            }
        }

        private void nextButton_Click(object sender, EventArgs e)
        {
            if (tracesComboBox.SelectedIndex > 0)
            {
                IAttribute caseId = null;
                foreach (IAttribute attr in _log.DescriptorMap.Keys)
                {
                    if (attr.ToString() == tracesComboBox.SelectedItem.ToString())
                        caseId = attr;
                }
                _miner.SingleCaseId = caseId;
                _miner.IsAlt = false;
            }
            else
            {
                if (isEnglish)
                    MessageBox.Show("Hierarchical diagram can be built only for one trace of the
log. Select a trace", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                else
                    MessageBox.Show("Иерархическую диаграмму можно построить по одной трассе.
Выберите одну трассу", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
```

```
            _miner.RegularExpressions = regExprListBox.Items.Cast<String>().ToList();
            _miner.IsHierarchical = true;

            Form form = new SelectHierarchicalElementForm(_log, _miner);
            form.FormClosed += new FormClosedEventHandler(form_FormClosed);
            form.Show();
            this.Hide();
        }

        private void form_FormClosed(object sender, FormClosedEventArgs e)
        {
            this.Close();
        }

        private void saveButton_Click(object sender, EventArgs e)
        {
            if (!_isReady)
            {
                if (isEnglish)
                    MessageBox.Show("Build model at first", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                else
                    MessageBox.Show("Сохранение недоступно. Сначала постройте модель", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            saveFileDialog1 = new SaveFileDialog();
            saveFileDialog1.Filter = "XMI File|*.xml";
            saveFileDialog1.Title = "Сохранить модель";
            saveFileDialog1.FileName = "diagram1";

            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                string filename = saveFileDialog1.FileName;
                if (usualSDradioButton.Checked)
                {
                    try
                    {
                        CoordUMLSDBuilder builder = new CoordUMLSDBuilder(_diagram);
                        builder.ExportToXMI();
                        builder.WriteToFile(filename);
                    }
                    catch (Exception ex)
                    {
                        if (isEnglish)
                            MessageBox.Show("The program cannot export your diagram to XMI-file.
Message: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                        else
                            MessageBox.Show("Программа не смогла экспортировать диаграмму в XMI-
формат. Сообщение: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    }
                }
                if (umlADradioButton.Checked)
                {
                    try
                    {
                        UmlAdExporter exporter = new UmlAdExporter(_adDiagrams);
                        exporter.ExportToXMI();
```

```
                    exporter.WriteToFile(filename);
                }
                catch (Exception ex)
                {
                    if (isEnglish)
                        MessageBox.Show("The program cannot export your diagram to XMI-file.
Message: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    else
                        MessageBox.Show("Программа не смогла экспортировать диаграмму в XMI-
формат. Сообщение: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }
    }

    private void tracesComboBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (tracesComboBox.SelectedIndex == 0)
        {
            _miner.IsAlt = true;
            _miner.SingleCaseId = null;
        }
        else
        {
            _miner.IsAlt = false;
            _miner.SingleCaseId =
_log.DescriptorMap.Keys.ElementAt(tracesComboBox.SelectedIndex - 1);
        }
    }

    private CSVLog _log;
    private UMLSDMiner _miner;
    private UMLSD _diagram;
    private bool _isReady = false;
    private HybridUMLModelsMiner _hybridMiner;
    private List<UmlAdImpl> _adDiagrams;
}
}
```

MainForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Xidv.EventLogLib.Csv;

namespace UMLModelsMiner
{
    public partial class MainForm : Form
    {
        bool isEnglish = false;
```

```
        public MainForm()
        {
            InitializeComponent();

            if (isEnglish)
            {
                this.Text = "UML Models Miner";
                this.groupBox1.Text = "Select file";
                this.viewButton.Text = "View...";
                this.groupBox2.Text = "Select attributes";
                this.nextButton.Text = "Next";
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            //openFileDialog1.InitialDirectory = "c:\\";
            openFileDialog1.Filter = "CSV files (*.csv)|*.csv";
            //openFileDialog1.RestoreDirectory = true;

            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                try
                {
                    fileNameTextBox.Text = openFileDialog1.FileName;

                    _log = new CSVLog(openFileDialog1.FileName, ';');
                    _log.ExtractAttrNamesFromHeadline();
                    this.Height = 430;

                    groupBox2.Visible = true;
                    nextButton.Visible = true;

                    caseIdComboBox.Items.Clear();
                    activityComboBox.Items.Clear();
                    timestampComboBox.Items.Clear();

                    caseIdComboBox.Items.AddRange(_log.EventAttrNames);
                    activityComboBox.Items.AddRange(_log.EventAttrNames);
                    timestampComboBox.Items.AddRange(_log.EventAttrNames);
                    caseIdComboBox.SelectedIndex = 0;
                    activityComboBox.SelectedIndex = 0;
                    timestampComboBox.SelectedIndex = 0;
                }
                catch (Exception ex)
                {
                    if (isEnglish)
                        MessageBox.Show("The program cannot read the file. Message: " +
ex.Message);
                    else
                        MessageBox.Show("Невозможно прочитать файл. Код ошибки: " + ex.Message);
                }
            }
        }

        private void caseIdComboBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            _log.CaseAttrName = caseIdComboBox.Text;
```

```
        }

        private void activityComboBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            _log.ActivityAttrName = activityComboBox.Text;
        }

        private void timestampComboBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            _log.TimestampAttrName = timestampComboBox.Text;
        }

        private void nextButton_Click(object sender, EventArgs e)
        {
            Form form = new ChooseAttrsForm(_log);
            form.FormClosed += new FormClosedEventHandler(form_FormClosed);
            form.Show();
            this.Hide();
        }

        private void form_FormClosed(object sender, FormClosedEventArgs e)
        {
            this.Close();
        }

        private CSVLog _log;

    }
}
```

SelectHierarchicalElementForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Xidv.EventLogLib;
using Xidv.EventLogLib.Csv;
using Xidv.UmlSdMiner;
using Xidv.UmlBehaviorDiagrams;
using Xidv.UmlBehaviorDiagrams.UmlSd;
using Xidv.UmlBehaviorDiagramsExporter;

namespace UMLModelsMiner
{
    public partial class SelectHierarchicalElementForm : Form
    {
        bool isEnglish = false;
        public SelectHierarchicalElementForm(CSVLog log, UMLSDMiner miner)
        {
            InitializeComponent();
```

```
            if (isEnglish)
            {
                Text = "UML Models Miner";
                label1.Text = "Hierarchies";
                this.buildButton.Text = "Build";
                this.saveButton.Text = "Save";
            }

            _log = log;
            _miner = miner;

            HashSet<string> hierarchies = _miner.ExtractHierarchicalElements();
            foreach (string h in hierarchies)
            {
                listBox1.Items.Add(h);
            }
        }

        private void buildButton_Click(object sender, EventArgs e)
        {
            if (listBox1.SelectedItem == null)
                if (isEnglish)
                    MessageBox.Show("Select one hierarchy.", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                else
                    MessageBox.Show("Необходимо выбрать одну иерархию.", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            else
            {
                try
                {
                    string hierarchy = listBox1.SelectedItem.ToString();
                    _miner.Hierarchy = hierarchy;
                    _diagram = _miner.Mine();
                    _isReady = true;
                    if (isEnglish)
                        MessageBox.Show("Model is build. You can save it to XMI-file", "Message",
MessageBoxButtons.OK);
                    else
                        MessageBox.Show("Модель построена, вы можете ее сохранить в XMI-файл",
"Сообщение", MessageBoxButtons.OK);
                }
                catch (Exception ex)
                {
                    if (isEnglish)
                        MessageBox.Show("The program cannot build a diagram for your log.
Message: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    else
                        MessageBox.Show("Программа не смогла построить диаграмму для вашего лога.
Сообщение: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }

        private void saveButton_Click(object sender, EventArgs e)
        {
            if (!_isReady)
            {
```

```
                if (isEnglish)
                    MessageBox.Show("Build model at first", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                else
                    MessageBox.Show("Сохранение недоступно. Сначала постройте модель", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            saveFileDialog1 = new SaveFileDialog();
            saveFileDialog1.Filter = "XMI File|*.xml";
            saveFileDialog1.Title = "Сохранить модель";
            saveFileDialog1.FileName = "diagram1";

            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                string filename = saveFileDialog1.FileName;
                try
                {
                    CoordUMLSDBuilder builder = new CoordUMLSDBuilder(_diagram);
                    builder.ExportToXMI();
                    builder.WriteToFile(filename);
                }
                catch (Exception ex)
                {
                    if (isEnglish)
                        MessageBox.Show("The program cannot export your diagram to XMI-file.
Message: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    else
                        MessageBox.Show("Программа не смогла экспортировать диаграмму в XMI-
формат. Сообщение: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }

        private CSVLog _log;
        private UMLSDMiner _miner;
        private UMLSD _diagram;
        private bool _isReady = false;
    }
}
```

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Изм.	Номера листов (страниц)				Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа и дата	Подпись	Дата
	измененных	замененных	новых	Аннулированных					

Изм.	Лист	№ докум.	Подп.	Дата
RU. 17701729.504900-01 12				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата