ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

СОГЛАСОВАНО

Доцент департамента программной инженерии факультета компьютерных наук

_____ Л. В. Дворянский
«_____» _____ 2020 г.

УТВЕРЖДАЮ

Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук

_____ В. В. Шилов
«_____» _____ 2020 г.

АНАЛИЗ ПОВЕДЕНИЯ ВРЕМЕННЫХ СИСТЕМ С ПОМОЩЬЮ ДИНАМИЧЕСКИХ МЕТРИЧЕСКИХ ГРАФОВ.

Текст программы

Лист Утверждения

RU.17701729.04.01-01 12 01-1-ЛУ

Исполнитель: Студент группы БПИ172
_____ А. А. Измайлов
«_____» _____ 2020 г.

Москва 2020

Утверждён
RU.17701729.04.01-01 12 01-1-ЛУ

АНАЛИЗ ПОВЕДЕНИЯ ВРЕМЕННЫХ СИСТЕМ С ПОМОЩЬЮ ДИНАМИЧЕСКИХ МЕТРИЧЕСКИХ ГРАФОВ.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

Текст программы

RU.17701729.04.01-01 12 01-1

Листов 77

Москва 2020

Содержание

1. Текст программы

1.1. TapaalHeadlessService.java

```
package com.github.zinoviy23.tapaal;

import java.awt.*;
import java.util.function.Supplier;

public class TapaalHeadlessService {
    public static boolean isHeadless() {
        return GraphicsEnvironment.isHeadless();
    }

    public static void handleHeadless(Runnable runnable) {
        if (!isHeadless()) {
            runnable.run();
        }
    }

    public static <T> T computeIfNotHeadless(Supplier<T> supplier) {
        if (!isHeadless()) {
            return supplier.get();
        }
        return null;
    }
}
```

1.2. ExtensionException.java

```
package com.github.zinoviy23.tapaal.extenstions;

public class ExtensionException extends RuntimeException {
  public ExtensionException(String message) {
    super(message);
  }

  public ExtensionException(String message, Throwable cause) {
    super(message, cause);
  }
}
```

1.3. ExtensionPoint.java

```
package com.github.zinoviy23.tapaal.extenstions;

import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.util.*;
import java.util.stream.Collectors;
```

```
public final class ExtensionPoint<T> {
  private final Class<T> interfaceType;
  private final List<Class<?>> extensionClasses = new ArrayList<>();

  private final Map<Class<?>, Object> extensionCache = new HashMap<>();

  public ExtensionPoint(Class<T> interfaceType) {
    this.interfaceType = interfaceType;
  }

  public Class<T> getInterfaceType() {
    return interfaceType;
  }

  public synchronized List<T> getExtensions() {
    return extensionClasses.stream()
        .map(ext -> {
          @SuppressWarnings("unchecked")
          T instance = (T) extensionCache.computeIfAbsent(ext, ExtensionPoint::instantiate)
              ;
          return instance;
        })
        .collect(Collectors.toList());
  }

  public synchronized void addExtension(Class<?> extension) {
    if (!interfaceType.isAssignableFrom(extension)) {
      throw new ExtensionException("Adding␣extension␣of␣wrong␣type.␣Expected␣" +
          interfaceType.getName());
    }

    extensionClasses.add(extension);
  }

  private static Object instantiate(Class<?> clazz) {
    try {
      Constructor<?> constructor = clazz.getConstructor();
      return constructor.newInstance();
    } catch (NoSuchMethodException | IllegalAccessException | InstantiationException |
        InvocationTargetException e) {
      throw new ExtensionException("Extension␣must␣have␣default␣constructor", e);
    }
  }
}
```

1.4. ExtensionManager.java

```
package com.github.zinoviy23.tapaal.extenstions;

import java.util.HashMap;
import java.util.Map;
import java.util.Objects;
```

```
public class ExtensionManager {
  private static ExtensionManager instance;

  private final Map<String, ExtensionPoint<?>> extensionPoints = new HashMap<>();

  private ExtensionManager() {
  }

  public static ExtensionManager getInstance() {
    if (instance == null) {
      synchronized (ExtensionManager.class) {
        if (instance == null) {
          instance = new ExtensionManager();
        }
      }
    }
    return instance;
  }

  public synchronized <T> void registerExtensionPoint(String name, Class<T> interfaceType)
      {
    Objects.requireNonNull(name, "name");
    Objects.requireNonNull(interfaceType, "interfaceType");

    if (extensionPoints.containsKey(name)) {
      throw new ExtensionException("Already␣contains␣extensionPoint␣" + name);
    }
    extensionPoints.put(name, new ExtensionPoint<>(interfaceType));
  }

  public synchronized void registerExtension(String name, Class<?> extensionImpl) {
    Objects.requireNonNull(name, "name");
    Objects.requireNonNull(extensionImpl, "extension");
    if (!extensionPoints.containsKey(name)) {
      throw new ExtensionException("There␣isn't␣any␣extension␣point␣" + name);
    }

    extensionPoints.get(name).addExtension(extensionImpl);
  }

  public synchronized <T> ExtensionPoint<T> getExtensionPoint(String name) {
    //noinspection unchecked
    return (ExtensionPoint<T>) extensionPoints.get(name);
  }
}
```

1.5. TapnFilesTest.java

```
package com.github.zinoviy23;

import dk.aau.cs.io.LoadedModel;
```

```
import dk.aau.cs.io.TapnXmlLoader;
import dk.aau.cs.io.TimedArcPetriNetNetworkWriter;
import dk.aau.cs.model.tapn.*;
import dk.aau.cs.util.FormatException;
import org.junit.Test;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Collections;

import static org.assertj.core.api.Assertions.assertThat;

public class TapnFilesTest {
    @Test
    public void readFileWebserver() throws FormatException {
        InputStream resource = getClass().getClassLoader().getResourceAsStream("Example␣
            nets/webserver.tapn");
        assertThat(resource).isNotNull();

        assertWebserver(resource);
    }

    @Test
    public void writeFileWebserver() throws FormatException, ParserConfigurationException,
         TransformerException, IOException {
        TapnXmlLoader loader = new TapnXmlLoader();
        InputStream resource = getClass().getClassLoader().getResourceAsStream("Example␣
            nets/webserver.tapn");
        assertThat(resource).isNotNull();

        LoadedModel model = loader.load(resource);
        TimedArcPetriNetNetwork network = model.network();

        TimedArcPetriNetNetworkWriter writer = new TimedArcPetriNetNetworkWriter(network,
            model.templates(), Collections.emptyList(), network.constants());
        File file = File.createTempFile("myFile", ".xml");
        writer.savePNML(file);

        try (InputStream stream = new FileInputStream(file)) {
            assertWebserver(stream);
        }

        assertThat(file.delete()).isTrue();
    }

    private void assertWebserver(InputStream resource) throws FormatException {
        TapnXmlLoader loader = new TapnXmlLoader();
```

```
        LoadedModel model = loader.load(resource);
        TimedArcPetriNetNetwork network = model.network();
        TimedArcPetriNet webServer = network.getTAPNByName("WebServer");
        assertThat(webServer).isNotNull();

        TimedPlace userA = webServer.getPlaceByName("UserA");
        TimedPlace userB = webServer.getPlaceByName("UserB");
        TimedTransition drop = webServer.getTransitionByName("Drop");
        assertThat(userA).isNotNull();
        assertThat(userB).isNotNull();
        assertThat(drop).isNotNull();
    }
}
```

1.6. ConversionException.java

```
package com.github.zinoviy23.tapnToMg.converters;

public class ConversionException extends RuntimeException {
  public ConversionException(Throwable cause) {
    super(cause);
  }
}
```

1.7. Converter.java

```
package com.github.zinoviy23.tapnToMg.converters;

import org.jetbrains.annotations.NotNull;

import java.util.function.Function;

public interface Converter<TFrom, TTo> extends Function<TFrom, TTo> {
  @NotNull TTo convert(@NotNull TFrom from);

  @Override
  default TTo apply(TFrom from) {
    if (from == null) {
      return null;
    }
    return convert(from);
  }
}
```

1.8. ConvertersFactory.java

```
package com.github.zinoviy23.tapnToMg.converters;

import com.github.zinoviy23.metricGraphs.MetricGraph;
import com.github.zinoviy23.metricGraphs.io.MetricGraphReadingException;
import com.github.zinoviy23.metricGraphs.io.ValidatedMetricGraphJsonReader;
import com.github.zinoviy23.metricGraphs.util.ContainerUtil;
```

```
import com.github.zinoviy23.tapnToMg.converters.mgToTapn.ConvertedTimedArcPetriNet;
import com.github.zinoviy23.tapnToMg.converters.mgToTapn.
   MetricGraphToTimedArcPetriNetConverter;
import dk.aau.cs.io.ModelLoader;
import dk.aau.cs.io.TimedArcPetriNetNetworkWriter;
import dk.aau.cs.model.tapn.TimedArcPetriNet;
import org.jetbrains.annotations.NotNull;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;
import java.io.File;
import java.io.IOException;
import java.util.Collections;
import java.util.List;
import java.util.function.Function;

public class ConvertersFactory {
  public static @NotNull Function<File, File> createMetricGraphToTimedArcPetriNetConverter
      () {
    return createFileToMetricGraphConverter()
        .andThen(new MetricGraphToTimedArcPetriNetConverter())
        .andThen(createTimedArcPetriNetToFileConverter());
  }

  public static @NotNull Function<File, MetricGraph> createFileToMetricGraphConverter() {
    return file -> {
      try (var reader = new ValidatedMetricGraphJsonReader(file).getReader()) {
        return reader.read();
      } catch (IOException | MetricGraphReadingException e) {
        throw new ConversionException(e);
      }
    };
  }

  public static @NotNull Function<ConvertedTimedArcPetriNet, File>
      createTimedArcPetriNetToFileConverter() {
    return petriNet -> {
      var writer = new TimedArcPetriNetNetworkWriter(
          petriNet.getNetwork(),
          List.of(petriNet.getTemplate()),
          Collections.emptyList(),
          petriNet.getNetwork().constants()
      );
      try {
        File file = File.createTempFile("convertedTapn", ".tapn");
        writer.savePNML(file);
        return file;
      } catch (TransformerException | IOException | ParserConfigurationException e) {
        throw new ConversionException(e);
      }
    };
```

```
  }

  public static @NotNull Function<File, TimedArcPetriNet>
      createFileToTimedArcPetriNetConverter() {
    return file -> {
      ModelLoader loader = new ModelLoader();
      try {
        var load = loader.load(file);
        var network = load.network();
        var net = ContainerUtil.first(network.allTemplates());
        if (net == null) {
          throw new RuntimeException("Network␣must␣have␣at␣least␣one␣template");
        }
        return net;
      } catch (Exception e) {
        throw new ConversionException(e);
      }
    };
  }

  private ConvertersFactory() {
  }
}
```

1.9. MgMetadataUtil.java

```
package com.github.zinoviy23.tapnToMg.converters.tapnToMg;

import com.github.zinoviy23.metricGraphs.MetricGraph;
import com.github.zinoviy23.metricGraphs.util.Ref;
import dk.aau.cs.model.tapn.*;
import org.jetbrains.annotations.NotNull;
import org.jgrapht.alg.util.Pair;
import org.jgrapht.alg.util.Triple;

final class MgMetadataUtil {
  private MgMetadataUtil() {
  }

  static @NotNull String getNodeName(@NotNull TimedTransition transition) {
    return "n_" + transition.name();
  }

  static @NotNull String getNodeComment(@NotNull TimedTransition transition) {
    return "Generated␣from␣transition␣" + transition.name();
  }

  static @NotNull String getArcName(@NotNull TimedTransition transitionSource,
                                    @NotNull TimedPlace place,
                                    @NotNull TimedTransition transitionTarget) {
    return "a_" + transitionSource.name() + "_" + place.name() + "_" + transitionTarget.
        name();
```

```
}

static @NotNull String getNameForReversal(@NotNull String arcName) {
  return "rev_" + arcName;
}

static @NotNull String getTokenName(@NotNull Ref<Integer> currentId) {
  return "t_" + currentId.update(i -> i + 1);
}

static @NotNull String getLeadName(@NotNull TimedOutputArc outputArc) {
  return "l_" + outputArc.source().name() + "_" + outputArc.destination().name();
}

static @NotNull String getLeadName(@NotNull TimedInputArc inputArc) {
  return "l_" + inputArc.source().name() + "_" + inputArc.destination().name();
}

static @NotNull String getInfName(@NotNull TimedInputArc inputArc) {
  return "inf_" + inputArc.source().name() + "_" + inputArc.destination().name();
}

static @NotNull String getInfName(@NotNull TimedOutputArc outputArc) {
  return "inf_" + outputArc.source().name() + "_" + outputArc.destination().name();
}

static @NotNull String getGraphName(@NotNull TimedArcPetriNet tapn) {
  return "mg_" + tapn.name();
}

static @NotNull String getMultiedgeHandlerNodeName(@NotNull String name) {
  return "br_n_" + name;
}

static @NotNull Pair<String, String> getMultiedgeHandlerArcsNames(@NotNull String name)
    {
  return Pair.of("br_" + name + "_part1", "br_" + name + "_part2");
}

static @NotNull Triple<String, String, String> getSelfLoopHandleArcsNames(@NotNull
    String name) {
  return Triple.of("br_" + name + "_part1", "br_" + name + "_part2", "br_" + name + "
      _part3");
}

static @NotNull String getComment(@NotNull String name, @NotNull TimedArcPetriNet
    petriNet) {
  return String.format("Metric␣graph␣%s␣generated␣from␣corresponding␣Timed␣Arc␣Petri␣Net
      ␣%s.\n" +
          "n_*␣-␣nodes,␣generated␣from␣transitions.\n" +
          "a_*_*_*␣-␣arc,␣generated␣from␣transition,␣place,␣and␣transition.\n" +
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.01-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
            "rev_* - reversal arc. Arc + reversal arc = edge in metric graph.\n" +
            "t_* - point generated from token.\n" +
            "l_* - lead generated from place, which has only incoming or only outgoing arcs
                .\n" +
            "inf_* - node representing infinity node, which isn't exist.\n" +
            "br_* - additional node, to break self loop or multiple edges between to nodes.
                ",
        name, petriNet.name()
    );
  }
}
```

1.10. TapnPlacesArcs.java

```
package com.github.zinoviy23.tapnToMg.converters.tapnToMg;

import dk.aau.cs.model.tapn.*;
import org.jetbrains.annotations.NotNull;

import java.util.*;

final class TapnPlacesArcs {
  private final Map<TimedPlace, List<TimedInputArc>> outputArcs = new HashMap<>();
  private final Map<TimedPlace, List<TimedOutputArc>> inputArcs = new HashMap<>();

  public TapnPlacesArcs(@NotNull TimedArcPetriNet petriNet) {
    for (TimedTransition transition : petriNet.transitions()) {
      for (TimedOutputArc outputArc : transition.getOutputArcs()) {
        inputArcs.computeIfAbsent(outputArc.destination(), __ -> new ArrayList<>()).add(
            outputArc);
      }
      for (TimedInputArc inputArc : transition.getInputArcs()) {
        assertInputArc(inputArc);
        outputArcs.computeIfAbsent(inputArc.source(), __ -> new ArrayList<>()).add(inputArc
            );
      }
    }
  }

  public List<TimedInputArc> getOutputArcs(@NotNull TimedPlace place) {
    var list = outputArcs.get(place);
    return list != null ? Collections.unmodifiableList(list) : Collections.emptyList();
  }

  public List<TimedOutputArc> getInputArcs(@NotNull TimedPlace place) {
    var list = inputArcs.get(place);
    return list != null ? Collections.unmodifiableList(list) : Collections.emptyList();
  }

  private static void assertInputArc(@NotNull TimedInputArc arc) {
    var interval = arc.interval();
    if (!interval.IsLowerBoundNonStrict() || !interval.IsUpperBoundNonStrict()) {
```

```
      throw new TimedArcPetriNetToMetricGraphConversionException(
          String.format("Time␣interval␣%s␣in␣%s␣must␣be␣closed!", interval, arc)
      );
    }

    if (interval.lowerBound().value() != interval.upperBound().value()) {
      throw new TimedArcPetriNetToMetricGraphConversionException(
          String.format("Time␣interval␣%s␣in␣%s␣must␣be␣zero␣length!", interval, arc)
      );
    }
  }
}
```

1.11. TimedArcPetriNetToMetricGraphConversionException.java

```
package com.github.zinoviy23.tapnToMg.converters.tapnToMg;

public final class TimedArcPetriNetToMetricGraphConversionException extends
    RuntimeException {
  public TimedArcPetriNetToMetricGraphConversionException(String message) {
    super(message);
  }
}
```

1.12. TimedArcPetriNetToMetricGraphConverter.java

```
package com.github.zinoviy23.tapnToMg.converters.tapnToMg;

import com.github.zinoviy23.metricGraphs.Arc;
import com.github.zinoviy23.metricGraphs.MetricGraph;
import com.github.zinoviy23.metricGraphs.MovingPoint;
import com.github.zinoviy23.metricGraphs.Node;
import com.github.zinoviy23.metricGraphs.util.DoubleUtil;
import com.github.zinoviy23.metricGraphs.util.Ref;
import com.github.zinoviy23.tapnToMg.converters.Converter;
import dk.aau.cs.model.tapn.*;
import org.jetbrains.annotations.NotNull;
import org.jgrapht.alg.util.Pair;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public final class TimedArcPetriNetToMetricGraphConverter implements Converter<
    TimedArcPetriNet, MetricGraph> {
  @Override
  public @NotNull MetricGraph convert(@NotNull TimedArcPetriNet petriNet) {
    var graphName = MgMetadataUtil.getGraphName(petriNet);
    var graphBuilder = MetricGraph.createBuilder()
        .setId(graphName)
        .setComment(MgMetadataUtil.getComment(graphName, petriNet));
```

```
  var transitionNodes = addNodes(graphBuilder, petriNet);
  var placesArcs = new TapnPlacesArcs(petriNet);
  var tokenId = addEdges(graphBuilder, petriNet, transitionNodes, placesArcs);
  addLeads(graphBuilder, petriNet, transitionNodes, placesArcs, tokenId);

  return graphBuilder.buildGraph();
}

private @NotNull Map<TimedTransition, Node> addNodes(@NotNull MetricGraph.
    MetricGraphBuilder builder,
                                                     @NotNull TimedArcPetriNet petriNet) {
  var result = new HashMap<TimedTransition, Node>();

  for (TimedTransition transition : petriNet.transitions()) {
    assertTransition(transition);

    var id = MgMetadataUtil.getNodeName(transition);
    var comment = MgMetadataUtil.getNodeComment(transition);
    var node = Node.createNode(id, id, comment);
    builder.addNode(node);
    result.put(transition, node);
  }

  return result;
}

private Ref<Integer> addEdges(@NotNull MetricGraph.MetricGraphBuilder builder,
                              @NotNull TimedArcPetriNet petriNet,
                              @NotNull Map<TimedTransition, Node> transitionNodes,
                              @NotNull TapnPlacesArcs arcs) {
  var tokenId = new Ref<>(0);
  for (TimedTransition transition : petriNet.transitions()) {
    var source = transitionNodes.get(transition);
    for (TimedOutputArc outputArc : transition.getOutputArcs()) {
      var place = outputArc.destination();
      for (TimedInputArc arc : arcs.getOutputArcs(place)) {
        var target = transitionNodes.get(arc.destination());
        String name = addEdge(builder, transition, source, place, arc, target);

        var points = getPointsFromPlace(tokenId, place);
        builder.addPoints(name, points);
      }
    }
  }
  return tokenId;
}

private @NotNull String addEdge(@NotNull MetricGraph.MetricGraphBuilder builder,
                                @NotNull TimedTransition transition,
                                @NotNull Node source,
```

```
                            @NotNull TimedPlace place,
                            @NotNull TimedInputArc arc,
                            @NotNull Node target) {
String name = MgMetadataUtil.getArcName(transition, place, arc.destination());
if (!builder.containsEdge(source, target) && !source.equals(target)) {
  var resultArc = Arc.createBuilder()
      .setSource(source)
      .setTarget(target)
      .setLength(fixLength(arc.interval().upperBound().value()))
      .setId(name);

  builder.addArc(resultArc).withReversal(MgMetadataUtil.getNameForReversal(name));
} else if (!source.equals(target)) {
  var handlerNode = Node.createMultiEdgeHandler(MgMetadataUtil.
     getMultiedgeHandlerNodeName(name), source, target);
  var edgesNames = MgMetadataUtil.getMultiedgeHandlerArcsNames(name);
  var length = fixLength(arc.interval().upperBound().value()) / 2.0;
  builder
      .addNode(handlerNode)
      .addArc(Arc.createBuilder()
          .setSource(source)
          .setTarget(handlerNode)
          .setId(edgesNames.getFirst())
          .setLength(length)
      ).withReversal(MgMetadataUtil.getNameForReversal(edgesNames.getFirst()))
      .addArc(Arc.createBuilder()
          .setSource(handlerNode)
          .setTarget(target)
          .setId(edgesNames.getSecond())
          .setLength(length)
      ).withReversal(MgMetadataUtil.getNameForReversal(edgesNames.getSecond()));

  name = edgesNames.getFirst();
} else {
  var edgesNames = MgMetadataUtil.getSelfLoopHandleArcsNames(name);
  var loopFixes = fixSelfLoop(name, source, target);
  var length = fixLength(arc.interval().upperBound().value()) / 3.0;
  builder
      .addNode(loopFixes.getFirst())
      .addNode(loopFixes.getSecond())
      .addArc(Arc.createBuilder()
          .setSource(source)
          .setTarget(loopFixes.getFirst())
          .setId(edgesNames.getFirst())
          .setLength(length)
      ).withReversal(MgMetadataUtil.getNameForReversal(edgesNames.getFirst()))
      .addArc(Arc.createBuilder()
          .setSource(loopFixes.getFirst())
          .setTarget(loopFixes.getSecond())
          .setId(edgesNames.getSecond())
          .setLength(length)
```

```
        ).withReversal(MgMetadataUtil.getNameForReversal(edgesNames.getSecond()))
        .addArc(Arc.createBuilder()
            .setSource(loopFixes.getSecond())
            .setTarget(target)
            .setId(edgesNames.getThird())
            .setLength(length)
        ).withReversal(MgMetadataUtil.getNameForReversal(edgesNames.getThird()));
    name = edgesNames.getFirst();
  }

  return name;
}

private double fixLength(int length) {
  return length != 0 ? length : DoubleUtil.DELTA;
}

private Pair<Node, Node> fixSelfLoop(@NotNull String name, @NotNull Node source,
    @NotNull Node target) {
  var handlerNode1 = Node.createMultiEdgeHandler(MgMetadataUtil.
      getMultiedgeHandlerNodeName(name) + "_1", source, target);
  var handlerNode2 = Node.createMultiEdgeHandler(MgMetadataUtil.
      getMultiedgeHandlerNodeName(name) + "_2", source, target);
  return Pair.of(handlerNode1, handlerNode2);
}

private @NotNull List<MovingPoint> getPointsFromPlace(Ref<Integer> tokenId, TimedPlace
    place) {
  // TODO: in paper points should be at the beginning of arc, but it isn't intuitive
  return place.tokens().stream()
      .map(token -> new MovingPoint(MgMetadataUtil.getTokenName(tokenId), 0))
      .collect(Collectors.toList());
}

private void addLeads(@NotNull MetricGraph.MetricGraphBuilder builder,
                      @NotNull TimedArcPetriNet petriNet,
                      @NotNull Map<TimedTransition, Node> transitionNodes,
                      @NotNull TapnPlacesArcs arcs,
                      @NotNull Ref<Integer> tokenId) {
  for (TimedPlace place : petriNet.places()) {
    var outputArcs = arcs.getOutputArcs(place);
    var inputArcs = arcs.getInputArcs(place);
    if (!inputArcs.isEmpty() && outputArcs.isEmpty()) {
      for (TimedOutputArc inputArc : inputArcs) {
        var node = transitionNodes.get(inputArc.source());
        var leadName = MgMetadataUtil.getLeadName(inputArc);
        var infinity = Node.createInfinity(MgMetadataUtil.getInfName(inputArc));
        builder
            .addNode(infinity)
            .addArc(Arc.createBuilder()
                .setSource(node)
```

```
                .setTarget(infinity)
                .setLength(Double.POSITIVE_INFINITY)
                .setId(leadName)
                .createArc()
            )
            .withReversal(MgMetadataUtil.getNameForReversal(leadName));
      }
    }
    if (inputArcs.isEmpty() && !outputArcs.isEmpty()) {
      for (TimedInputArc outputArc : outputArcs) {
        var node = transitionNodes.get(outputArc.destination());
        var leadName = MgMetadataUtil.getLeadName(outputArc);
        var infinity = Node.createInfinity(MgMetadataUtil.getInfName(outputArc));
        builder
            .addNode(infinity)
            .addArc(Arc.createBuilder()
                .setId(leadName)
                .setSource(infinity)
                .setTarget(node)
                .setLength(Double.POSITIVE_INFINITY)
                .setPoints(getPointsFromInputArc(tokenId, outputArc))
                .createArc()
            )
            .withReversal(MgMetadataUtil.getNameForReversal(leadName));
      }
    }
  }
}

private @NotNull List<MovingPoint> getPointsFromInputArc(Ref<Integer> tokenId,
    TimedInputArc arc) {
  return arc.source().tokens().stream()
      .map(token -> new MovingPoint(MgMetadataUtil.getTokenName(tokenId), arc.interval().
          upperBound().value()))
      .collect(Collectors.toList());
}

private static void assertTransition(@NotNull TimedTransition transition) {
  if (transition.isUrgent()) {
    throw new TimedArcPetriNetToMetricGraphConversionException(
        String.format("Transition␣%s␣must␣be␣non-urgent", transition.name())
    );
  }
}
}
```

1.13. MinMax.java

```
package com.github.zinoviy23.tapnToMg.converters.mgToTapn;

final class MinMax {
  static final MinMax NEUTRAL = new MinMax(Double.POSITIVE_INFINITY, Double.
```

```
      NEGATIVE_INFINITY);
  double min;
  double max;

  MinMax(double min, double max) {
    this.min = min;
    this.max = max;
  }

  static MinMax reduce(MinMax minMax, double d) {
    return new MinMax(Math.min(minMax.min, d), Math.max(minMax.max, d));
  }

  static MinMax combine(MinMax a, MinMax b) {
    return new MinMax(Math.min(a.min, b.min), Math.max(a.max, b.max));
  }

  double avg() {
    return (min + max) / 2;
  }
}
```

1.14. MetricGraphToTimedArcPetriNetConverter.java

```
package com.github.zinoviy23.tapnToMg.converters.mgToTapn;

import com.github.zinoviy23.metricGraphs.Arc;
import com.github.zinoviy23.metricGraphs.MetricGraph;
import com.github.zinoviy23.metricGraphs.MovingPoint;
import com.github.zinoviy23.metricGraphs.Node;
import com.github.zinoviy23.tapnToMg.converters.Converter;
import dk.aau.cs.model.tapn.*;
import org.jetbrains.annotations.NotNull;
import org.jgrapht.Graph;
import org.jgrapht.graph.SimpleDirectedWeightedGraph;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

public final class MetricGraphToTimedArcPetriNetConverter implements Converter<MetricGraph
    , ConvertedTimedArcPetriNet> {
  @Override
  public @NotNull ConvertedTimedArcPetriNet convert(@NotNull MetricGraph graph) {
    var petriNet = new TimedArcPetriNet(TapnNamingUtil.getTapnName(graph.getId()));

    var layoutGraph = new SimpleDirectedWeightedGraph<>(null, null);
    var mapping = addPlacesForEachArc(graph, petriNet, layoutGraph);

    for (Arc arc : graph.getGraph().edgeSet()) {
      processArc(graph, arc, petriNet, mapping, layoutGraph);
```

```
  }

  var result = new TimedArcPetriNetNetwork();
  result.add(petriNet);

  var layoutMaker = new TapnLayoutMaker(graph.getId(), layoutGraph);

  return new ConvertedTimedArcPetriNet(result, petriNet, layoutMaker.createDataLayer());
}

private @NotNull Map<Arc, TimedPlace> addPlacesForEachArc(@NotNull MetricGraph graph,
                                                          @NotNull TimedArcPetriNet petriNet,
                                                          @NotNull Graph<Object, Object>
                                                              layoutGraph) {
  var arcToPlaceMapping = new HashMap<Arc, TimedPlace>();
  for (Arc arc : graph.getGraph().edgeSet()) {
    if (Node.isInfinity(arc.getSource())) continue;

    var placeSourceTarget = new LocalTimedPlace(TapnNamingUtil.nameForPlace(arc));
    arcToPlaceMapping.put(arc, placeSourceTarget);
    petriNet.add(placeSourceTarget);
    layoutGraph.addVertex(placeSourceTarget);

    //TODO: tokens cannot be serialized
    arc.getPoints().stream()
        .map(point -> new TimedToken(placeSourceTarget, BigDecimal.valueOf(point.
            getPosition())))
        .forEach(placeSourceTarget::addToken);
  }

  return arcToPlaceMapping;
}

private void processArc(@NotNull MetricGraph graph,
                        @NotNull Arc arc,
                        @NotNull TimedArcPetriNet petriNet,
                        @NotNull Map<Arc, TimedPlace> mapping,
                        @NotNull SimpleDirectedWeightedGraph<Object, Object> graphToDraw) {
  if (Node.isInfinity(arc.getSource())) {
    handleInfinity(graph, arc, petriNet, mapping, graphToDraw);
    return;
  }

  var placeSourceTarget = mapping.get(arc);

  //TODO make rational
  var timedTransition = new TimedTransition(TapnNamingUtil.nameForTransition(arc), false
      );
  var arcLength = new IntBound(Double.isInfinite(arc.getLength()) ? 0 : (int) arc.
      getLength());
  var timedInputArc = new TimedInputArc(placeSourceTarget, timedTransition,
```

```
      new TimeInterval(
          true, arcLength, arcLength.copy(), true
      )
  );
  petriNet.add(timedTransition);
  petriNet.add(timedInputArc);

  graphToDraw.addVertex(timedTransition);
  graphToDraw.addEdge(placeSourceTarget, timedTransition, timedInputArc);

  addArcForNeighbours(graph, arc, petriNet, mapping, graphToDraw, timedTransition);

  var collapsingTransition = new TimedTransition(TapnNamingUtil.
      nameForCollapsingTransition(arc), false);
  var collapsingInputArc = new TimedInputArc(placeSourceTarget, collapsingTransition,
      new TimeInterval(
          true, new IntBound(0), new IntBound(0), true
      ),
      new IntWeight(2)
  );

  var collapsingOutputArc = new TimedOutputArc(collapsingTransition, placeSourceTarget);

  petriNet.add(collapsingTransition);
  petriNet.add(collapsingInputArc);
  petriNet.add(collapsingOutputArc);

  graphToDraw.addVertex(collapsingTransition);
  graphToDraw.addEdge(placeSourceTarget, collapsingTransition, collapsingInputArc);
  graphToDraw.setEdgeWeight(collapsingInputArc, 2);
  graphToDraw.addEdge(collapsingTransition, placeSourceTarget, collapsingOutputArc);
}

private void addArcForNeighbours(@NotNull MetricGraph graph,
                                 @NotNull Arc arc,
                                 @NotNull TimedArcPetriNet petriNet,
                                 @NotNull Map<Arc, TimedPlace> mapping,
                                 @NotNull SimpleDirectedWeightedGraph<Object, Object>
                                     graphToDraw,
                                 @NotNull TimedTransition timedTransition) {
  if (!Node.isInfinity(arc.getTarget())) {
    for (Arc outgoingArc : graph.getGraph().outgoingEdgesOf(arc.getTarget())) {
      var destination = mapping.get(outgoingArc);
      var timedOutputArc = new TimedOutputArc(timedTransition, destination);
      petriNet.add(timedOutputArc);
      graphToDraw.addEdge(timedTransition, destination, timedOutputArc);
    }
  }
}

private void handleInfinity(@NotNull MetricGraph graph,
```

```
                            @NotNull Arc arc,
                            @NotNull TimedArcPetriNet petriNet,
                            @NotNull Map<Arc, TimedPlace> mapping,
                            @NotNull SimpleDirectedWeightedGraph<Object, Object>
                                graphToDraw) {
    if (arc.getPoints().isEmpty()) {
      return;
    }

    var timedTransition = new TimedTransition(TapnNamingUtil.nameForTransition(arc), false
        );
    petriNet.add(timedTransition);
    var name = TapnNamingUtil.nameForPlace(arc);
    var arcs = new ArrayList<TimedInputArc>();
    for (MovingPoint point : arc.getPoints()) {
      var placeName = TapnNamingUtil.nameForInfinitePlace(name, point);
      var place = new LocalTimedPlace(placeName);
      var length = (int) point.getPosition();
      var inputArc = new TimedInputArc(place, timedTransition, new TimeInterval(
        true, new IntBound(length), new IntBound(length), true
      ));
      petriNet.add(place);
      var token = new TimedToken(place);
      place.addToken(token);
      petriNet.add(inputArc);
      graphToDraw.addVertex(place);
      arcs.add(inputArc);
    }

    graphToDraw.addVertex(timedTransition);
    for (TimedInputArc timedInputArc : arcs) {
      graphToDraw.addEdge(timedInputArc.source(), timedInputArc.destination(),
          timedInputArc);
    }

    addArcForNeighbours(graph, arc, petriNet, mapping, graphToDraw, timedTransition);
  }
}
```

1.15. TapnNamingUtil.java

```
package com.github.zinoviy23.tapnToMg.converters.mgToTapn;

import com.github.zinoviy23.metricGraphs.Arc;
import com.github.zinoviy23.metricGraphs.MovingPoint;
import org.jetbrains.annotations.NotNull;

final class TapnNamingUtil {
  private TapnNamingUtil() {
  }

  static @NotNull String getTapnName(@NotNull String id) {
```

```
    return "TAPN_" + id;
  }

  static @NotNull String nameForPlace(@NotNull Arc arc) {
    return "p_" + arc.getId();
  }

  static @NotNull String nameForTransition(@NotNull Arc arc) {
    return "t_" + arc.getId();
  }

  static @NotNull String nameForCollapsingTransition(@NotNull Arc arc) {
    return "ct_" + arc.getId();
  }

  static @NotNull String nameForInfinitePlace(@NotNull String name, @NotNull MovingPoint
      point) {
    return name + "_" + point.getId();
  }
}
```

1.16. ConvertedTimedArcPetriNet.java

```
package com.github.zinoviy23.tapnToMg.converters.mgToTapn;

import dk.aau.cs.model.tapn.TimedArcPetriNet;
import dk.aau.cs.model.tapn.TimedArcPetriNetNetwork;
import org.jetbrains.annotations.NotNull;
import pipe.dataLayer.DataLayer;
import pipe.dataLayer.Template;

public final class ConvertedTimedArcPetriNet {
  private final TimedArcPetriNetNetwork network;
  private final Template template;

  public ConvertedTimedArcPetriNet(@NotNull TimedArcPetriNetNetwork network,
                                   @NotNull TimedArcPetriNet petriNet,
                                   @NotNull DataLayer dataLayer) {
    this.network = network;
    this.template = new Template(petriNet, dataLayer, null);
  }

  public TimedArcPetriNetNetwork getNetwork() {
    return network;
  }

  public Template getTemplate() {
    return template;
  }
}
```

1.17. TapnLayoutMaker.java

```
package com.github.zinoviy23.tapnToMg.converters.mgToTapn;

import com.github.zinoviy23.metricGraphs.util.GraphLayout;
import dk.aau.cs.model.tapn.TimedInputArc;
import dk.aau.cs.model.tapn.TimedOutputArc;
import dk.aau.cs.model.tapn.TimedPlace;
import dk.aau.cs.model.tapn.TimedTransition;
import org.jetbrains.annotations.NotNull;
import org.jgrapht.Graph;
import org.jgrapht.alg.drawing.FRLayoutAlgorithm2D;
import org.jgrapht.alg.drawing.IndexedFRLayoutAlgorithm2D;
import org.jgrapht.alg.drawing.model.Box2D;
import org.jgrapht.alg.drawing.model.LayoutModel2D;
import org.jgrapht.alg.drawing.model.MapLayoutModel2D;
import pipe.dataLayer.DataLayer;
import pipe.gui.graphicElements.AnnotationNote;
import pipe.gui.graphicElements.PetriNetObject;
import pipe.gui.graphicElements.PlaceTransitionObject;
import pipe.gui.graphicElements.tapn.TimedInputArcComponent;
import pipe.gui.graphicElements.tapn.TimedOutputArcComponent;
import pipe.gui.graphicElements.tapn.TimedPlaceComponent;
import pipe.gui.graphicElements.tapn.TimedTransitionComponent;

import java.util.HashMap;
import java.util.Map;
import java.util.Random;

final class TapnLayoutMaker {
  public static final int ITERATIONS = 200;
  private static final double X_OFFSET = 100;
  private static final double Y_OFFSET = 100;
  private static final int NOTE_OFFSET = 50;
  private static final String HEADER = "'%s' - Timed Arc Petri Net automatically generated
      from Dynamic Metric Graph '%s'";
  private static final String DESCRIPTION_TEXT = "Places p_* - generated from
      corresponding arc in metric graph\n" +
      "Transitions t_* - generated from corresponding arc in metric graph\n" +
      "Transitions ct_* - generated for simulated collapsing of two points on corresponding
          arc.";
  private static final int NOTE_WIDTH = 500;
  private static final int NODE_HEIGHT = 200;
  private static final int VERTEX_SIZE = 130;
  private static final int SIZE_OFFSET = 300;

  private final String graphId;
  private final Graph<Object, Object> graph;
  private final LayoutModel2D<Object> layoutModel;

  private final Map<Object, PlaceTransitionObject> visualComponents = new HashMap<>();
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.01-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
TapnLayoutMaker(@NotNull String graphId, @NotNull Graph<Object, Object> graph) {
  this.graphId = graphId;
  this.graph = graph;
  layoutModel = new GraphLayout().createLayout(graph, VERTEX_SIZE, SIZE_OFFSET);
}

private static String getTextFromId(@NotNull String id) {
  return String.format(HEADER, TapnNamingUtil.getTapnName(id), id) + "\n" +
      DESCRIPTION_TEXT;
}

public @NotNull DataLayer createDataLayer() {
  var dataLayer = new DataLayer();

  for (Object vertex : graph.vertexSet()) {
    dataLayer.addPetriNetObject(createPetriNetObject(vertex, layoutModel));
  }

  for (Object edge : graph.edgeSet()) {
    dataLayer.addPetriNetObject(createPetriNetObject(edge, layoutModel));
  }

  AnnotationNote note = createNote();
  dataLayer.addPetriNetObject(note);

  return dataLayer;
}

private @NotNull AnnotationNote createNote() {
  var maxX = (int) visualComponents.values().stream()
      .mapToDouble(PlaceTransitionObject::getPositionX)
      .max()
      .orElse(0) + NOTE_OFFSET;
  var maxY = (int) visualComponents.values().stream()
      .map(PlaceTransitionObject::getPositionY)
      .reduce(MinMax.NEUTRAL, MinMax::reduce, MinMax::combine)
      .avg();
  return new AnnotationNote(
      getTextFromId(graphId),
      maxX,
      maxY,
      NOTE_WIDTH,
      NODE_HEIGHT,
      true,
      true
  );
}

private @NotNull PetriNetObject createPetriNetObject(@NotNull Object object,
                                                     @NotNull LayoutModel2D<Object> model) {
```

```
if (object instanceof TimedPlace) {
  var point2D = model.get(object);
  var timedPlaceComponent = new TimedPlaceComponent(
      point2D.getX() + X_OFFSET,
      point2D.getY() + Y_OFFSET,
      (TimedPlace) object
  );
  visualComponents.put(object, timedPlaceComponent);
  return timedPlaceComponent;
}

if (object instanceof TimedTransition) {
  var point2D = model.get(object);
  var timedTransitionComponent = new TimedTransitionComponent(
      point2D.getX() + X_OFFSET,
      point2D.getY() + Y_OFFSET,
      (TimedTransition) object
  );
  visualComponents.put(object, timedTransitionComponent);
  return timedTransitionComponent;
}

if (object instanceof TimedInputArc) {
  var source = visualComponents.get(((TimedInputArc) object).source());
  var target = visualComponents.get(((TimedInputArc) object).destination());

  var arc = new TimedOutputArcComponent(
      source.getPositionX() + X_OFFSET,
      source.getPositionY() + Y_OFFSET,
      target.getPositionX() + X_OFFSET,
      target.getPositionY() + Y_OFFSET,
      source,
      target,
      0,
      "arc",
      false
  );
  var resultArc = new TimedInputArcComponent(arc);
  resultArc.setUnderlyingArc((TimedInputArc) object);

  return resultArc;
}

if (object instanceof TimedOutputArc) {
  var source = visualComponents.get(((TimedOutputArc) object).source());
  var target = visualComponents.get(((TimedOutputArc) object).destination());

  var arc = new TimedOutputArcComponent(
      source.getPositionX() + X_OFFSET,
      source.getPositionY() + Y_OFFSET,
      target.getPositionX() + X_OFFSET,
```

```
        target.getPositionY() + Y_OFFSET,
        source,
        target,
        0,
        "arc",
        false
      );
      arc.setUnderlyingArc((TimedOutputArc) object);

      return arc;
    }

    throw new AssertionError("Cannot␣be␣here");
  }
}
```

1.18. ExtendedTapaal.java

```
package com.github.zinoviy23.tapaal.extended;

import com.github.zinoviy23.tapaal.extenstions.ExtensionManager;
import net.tapaal.TAPAAL;

import java.io.IOException;
import java.io.InputStream;
import java.util.Map;
import java.util.Properties;
import java.util.regex.Pattern;

public class ExtendedTapaal {
  private static final Pattern extensionPattern = Pattern.compile("(.+)\\.\\d+");

  public static void main(String[] args) {
    initializeExtensionPoints();
    initializeExtensions();
    TAPAAL.main(args);
  }

  private static void initializeExtensionPoints() {
    try (InputStream config = TAPAAL.class.getResourceAsStream("/resources/Plugins/tapaal.
        ext.properties")) {
      Properties properties = new Properties();
      properties.load(config);
      addExtensionPoints(properties);
    } catch (IOException | ClassNotFoundException e) {
      e.printStackTrace();
    }
  }

  private static void addExtensionPoints(Properties properties) throws
      ClassNotFoundException {
    ExtensionManager instance = ExtensionManager.getInstance();
```

```
    for (Map.Entry<Object, Object> extensionPoint : properties.entrySet()) {
      Class<?> extensionInterface = Class.forName((String) extensionPoint.getValue());
      instance.registerExtensionPoint((String) extensionPoint.getKey(), extensionInterface)
          ;
    }
  }

  private static void initializeExtensions() {
    try (InputStream config = TAPAAL.class.getResourceAsStream("/extensions.properties"))
        {
      Properties properties = new Properties();
      properties.load(config);
      addExtensions(properties);
    } catch (IOException | ClassNotFoundException e) {
      e.printStackTrace();
    }
  }

  private static void addExtensions(Properties properties) throws ClassNotFoundException {
    ExtensionManager instance = ExtensionManager.getInstance();
    for (Map.Entry<Object, Object> extension : properties.entrySet()) {
      Class<?> extensionInterface = Class.forName((String) extension.getValue());
      String key;
      var matcher = extensionPattern.matcher((String) extension.getKey());
      if (matcher.matches()) {
        key = matcher.group(1);
      } else {
        key = (String) extension.getKey();
      }
      instance.registerExtension(key, extensionInterface);
    }
  }
}
```

1.19. MyGuiAction.java

```
package com.github.zinoviy23.tapaal.extended.extensions;

import pipe.gui.action.GuiAction;

import java.awt.event.ActionEvent;

public class MyGuiAction extends GuiAction {
  public MyGuiAction() {
    super("kek", "lol");
  }

  @Override
  public void actionPerformed(ActionEvent e) {
    System.out.println("clicked");
  }
}
```

1.20. ConvertTapnToJsonMgAction.java

```
package com.github.zinoviy23.tapaal.extended.extensions;

import com.github.zinoviy23.metricGraphs.MetricGraph;
import com.github.zinoviy23.metricGraphs.io.MetricGraphJsonWriter;
import pipe.gui.CreateGui;

import javax.swing.*;
import java.io.FileWriter;
import java.io.IOException;

public class ConvertTapnToJsonMgAction extends ConvertTapnToMgAction {
  public ConvertTapnToJsonMgAction() {
    super("Convert␣to␣Json␣Metric␣Graph", "Convert␣current␣Timed␣Arc␣Petri␣Net␣to␣Metric␣
        Graph␣in␣JSON␣format");
  }

  @Override
  protected String getFileExt() {
    return "json";
  }

  @Override
  protected boolean save(MetricGraph graph, String path) {
    try (var writer = new FileWriter(path); MetricGraphJsonWriter gw = new
        MetricGraphJsonWriter(writer, true)) {
      gw.write(graph);
      return true;
    } catch (IOException e) {
      JOptionPane.showMessageDialog(CreateGui.getAppGui(), e.getMessage(), "Error",
          JOptionPane.ERROR_MESSAGE);
    }

    return false;
  }
}
```

1.21. ConvertTapnToYedMgAction.java

```
package com.github.zinoviy23.tapaal.extended.extensions;

import com.github.zinoviy23.metricGraphs.MetricGraph;
import com.github.zinoviy23.metricGraphs.io.graphml.MetricGraphYedWriter;
import pipe.gui.CreateGui;

import javax.swing.*;
import java.io.FileWriter;
import java.io.IOException;

public class ConvertTapnToYedMgAction extends ConvertTapnToMgAction {
  public ConvertTapnToYedMgAction() {
```

```
    super("Convert␣to␣yEd␣Metric␣Graph", "Convert␣current␣Timed␣Arc␣Petri␣Net␣to␣Metric␣
        Graph␣in␣yEd␣format");
  }

  @Override
  protected String getFileExt() {
    return "graphml";
  }

  @Override
  protected boolean save(MetricGraph graph, String path) {
    try (var fileWriter = new FileWriter(path); var graphWriter = new MetricGraphYedWriter
        (fileWriter, true)) {
      graphWriter.write(graph);
      return true;
    } catch (IOException e) {
      JOptionPane.showMessageDialog(CreateGui.getAppGui(), e.getMessage(), "Error",
          JOptionPane.ERROR_MESSAGE);
    }
    return false;
  }
}
```

1.22. ConvertTapnToMgAction.java

```
package com.github.zinoviy23.tapaal.extended.extensions;

import com.github.zinoviy23.metricGraphs.MetricGraph;
import com.github.zinoviy23.metricGraphs.util.ContainerUtil;
import com.github.zinoviy23.tapnToMg.converters.tapnToMg.
    TimedArcPetriNetToMetricGraphConversionException;
import com.github.zinoviy23.tapnToMg.converters.tapnToMg.
    TimedArcPetriNetToMetricGraphConverter;
import pipe.gui.CreateGui;
import pipe.gui.action.GuiAction;
import pipe.gui.widgets.filebrowser.FileBrowser;

import javax.swing.*;
import java.awt.event.ActionEvent;

public abstract class ConvertTapnToMgAction extends GuiAction {
  protected ConvertTapnToMgAction(String name, String tooltip) {
    super(name, tooltip);
  }

  protected abstract String getFileExt();

  @Override
  public void actionPerformed(ActionEvent e) {
    if (CreateGui.getAppGui().getSelectedTabIndex() < 0) {
      return;
    }
```

```
    var currentTab = CreateGui.getCurrentTab();

    var path = FileBrowser.constructor("Metric␣Graph", getFileExt(), FileBrowser.userPath)
        .saveFile("graph");

    SwingWorker<Void, Void> worker = new SwingWorker<>() {
      @Override
      protected Void doInBackground() {
        var network = currentTab.network();
        if (network.allTemplates().size() > 1) {
          JOptionPane.showMessageDialog(CreateGui.getAppGui(), "There␣is␣too␣may␣templates.
              ␣Will␣use␣only␣first", "Warning", JOptionPane.WARNING_MESSAGE);
        }
        var net = ContainerUtil.first(network.allTemplates());
        if (net == null) {
          JOptionPane.showMessageDialog(CreateGui.getAppGui(), "There␣isn't␣any␣template␣in
              ␣network!", "Error", JOptionPane.ERROR_MESSAGE);
          return null;
        }

        var converter = new TimedArcPetriNetToMetricGraphConverter();
        try {
          var graph = converter.convert(net);
          if (save(graph, path)) {
            JOptionPane.showMessageDialog(CreateGui.getAppGui(), "Graph␣saved!");
          }
        } catch (TimedArcPetriNetToMetricGraphConversionException e) {
          JOptionPane.showMessageDialog(CreateGui.getAppGui(), e.getMessage(), "Error",
              JOptionPane.ERROR_MESSAGE);
          return null;
        }
        return null;
      }
    };

    worker.execute();
  }

  protected abstract boolean save(MetricGraph graph, String path);
}
```

1.23. ConvertMgToTapnAction.java

```
package com.github.zinoviy23.tapaal.extended.extensions;

import com.github.zinoviy23.tapnToMg.converters.ConversionException;
import com.github.zinoviy23.tapnToMg.converters.ConvertersFactory;
import pipe.gui.CreateGui;
import pipe.gui.action.GuiAction;
import pipe.gui.widgets.filebrowser.FileBrowser;

import javax.swing.*;
```

```
import java.awt.*;
import java.awt.event.ActionEvent;

public class ConvertMgToTapnAction extends GuiAction {
  public ConvertMgToTapnAction() {
    super("Convert␣from␣Metric␣Graph", "Open␣Metric␣Graph␣file␣as␣Timed␣Arc␣Petri␣Net");
  }

  @Override
  public void actionPerformed(ActionEvent event) {
    var mgFile = FileBrowser.constructor("Metric␣Graph", "json", FileBrowser.userPath).
        openFile();
    var converter = ConvertersFactory.createMetricGraphToTimedArcPetriNetConverter();

    CreateGui.getAppGui().setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
    SwingWorker<Void, Void> worker = new SwingWorker<>() {
      @Override
      protected Void doInBackground() throws Exception {
        try {
          var resultFile = converter.apply(mgFile);
          FileBrowser.userPath = mgFile.getParent();
          CreateGui.getAppGui().createNewTabFromFile(resultFile);
        } catch (ConversionException e) {
          JOptionPane.showMessageDialog(CreateGui.getAppGui(), e.getCause().getMessage(), "
              Error", JOptionPane.ERROR_MESSAGE);
        }
        return null;
      }

      @Override
      protected void done() {
        try {
          CreateGui.getAppGui().setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR)
              );
          get();
          JOptionPane.showMessageDialog(CreateGui.getAppGui(), "For␣save␣TAPN␣use␣Save␣as␣
              action");
        } catch (Exception e) {
          JOptionPane.showMessageDialog(CreateGui.getAppGui(),
              e.getMessage(),
              "Error␣loading␣file",
              JOptionPane.ERROR_MESSAGE
          );
        }
      }
    };

    worker.execute();
  }
}
```

1.24. Node.java

```
package com.github.zinoviy23.metricGraphs;

import com.github.zinoviy23.metricGraphs.api.Identity;
import com.github.zinoviy23.metricGraphs.api.ObjectWithComment;
import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

import java.util.Objects;

public abstract class Node implements Identity, ObjectWithComment {
  public static final String INFINITY_NODE_LABEL = "Infinity␣node";
  private final String id;
  private final String label;
  private final String comment;

  private Node(@NotNull String id) {
    this(id, id);
  }

  private Node(@NotNull String id, @Nullable String label) {
    this(id, label, null);
  }

  private Node(@NotNull String id, @Nullable String label, @Nullable String comment) {
    this.id = Objects.requireNonNull(id, "id");
    this.label = Objects.requireNonNull(label, "label");
    this.comment = comment;
  }

  @Contract("_␣->␣new")
  public static @NotNull Node createNode(@NotNull String id) {
    return new NodeImpl(id);
  }

  @Contract("_,␣_␣->␣new")
  public static @NotNull Node createNode(@NotNull String id, @Nullable String label) {
    return new NodeImpl(id, label);
  }

  @Contract("_,␣_,␣_␣->␣new")
  public static @NotNull Node createNode(@NotNull String id, @Nullable String label,
      @Nullable String comment) {
    return new NodeImpl(id, label, comment);
  }

  @Contract("_␣->␣new")
  public static @NotNull Node createInfinity(@NotNull String id) {
    return new InfinityNode(id);
```

```
}

public static @NotNull Node createMultiEdgeHandler(@NotNull String id, @NotNull Node
    source, @NotNull Node target) {
  return new MultiEdgeHandleNode(id, source, target);
}
@Override
public @NotNull String getId() {
  return id;
}

public @NotNull String getLabel() {
  return label;
}

@Override
public @Nullable String getComment() {
  return comment;
}

@Override
public boolean equals(Object o) {
  if (this == o) return true;
  if (o == null || getClass() != o.getClass()) return false;
  Node node = (Node) o;
  return id.equals(node.id);
}

@Override
public int hashCode() {
  return Objects.hash(id);
}

@Override
public String toString() {
  return "Node{" +
      "id='" + id + '\'' +
      ",␣label='" + label + '\'' +
      '}';
}

public static boolean isInfinity(@Nullable Node node) {
  return node instanceof InfinityNode;
}

static class NodeImpl extends Node {
  private NodeImpl(@NotNull String id) {
    super(id);
  }

  private NodeImpl(@NotNull String id, @Nullable String label) {
```

```
      super(id, label);
    }

    private NodeImpl(@NotNull String id, @Nullable String label, @Nullable String comment)
        {
      super(id, label, comment);
    }
  }

  static class InfinityNode extends Node {
    private InfinityNode(@NotNull String id) {
      super(id, INFINITY_NODE_LABEL, "This␣node␣placed␣on␣infinity␣distance␣from␣graph.");
    }
  }

  static class MultiEdgeHandleNode extends Node {
    private MultiEdgeHandleNode(@NotNull String id, @NotNull Node source, @NotNull Node
        target) {
      super(id, id, "Handles␣multiedges␣between␣" + source.getId() + "␣and␣" + target.getId
          ());
    }
  }
}
```

1.25. MetricGraphStructureException.java

```
package com.github.zinoviy23.metricGraphs;

public class MetricGraphStructureException extends RuntimeException {
  public MetricGraphStructureException(String message) {
    super(message);
  }

  public MetricGraphStructureException(String message, Throwable cause) {
    super(message, cause);
  }
}
```

1.26. MovingPoint.java

```
package com.github.zinoviy23.metricGraphs;

import com.github.zinoviy23.metricGraphs.api.Identity;
import com.github.zinoviy23.metricGraphs.api.ObjectWithComment;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

import java.util.Objects;

public final class MovingPoint implements Identity, ObjectWithComment {
  private final String id;
  private final double position;
```

```
  private final String comment;

  public MovingPoint(@NotNull String id, double position) {
    this(id, position, null);
  }

  public MovingPoint(@NotNull String id, double position, @Nullable String comment) {
    this.id = Objects.requireNonNull(id, "id");
    this.position = position;
    this.comment = comment;
  }

  @Override
  public @NotNull String getId() {
    return id;
  }

  public double getPosition() {
    return position;
  }

  @Override
  public @Nullable String getComment() {
    return comment;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    MovingPoint movingPoint = (MovingPoint) o;
    return id.equals(movingPoint.id);
  }

  @Override
  public int hashCode() {
    return Objects.hash(id);
  }

  @Override
  public String toString() {
    return "MovingPoint{" +
        "id='" + id + '\'' +
        ",␣position=" + position +
        '}';
  }
}
```

1.27. Arc.java

```
package com.github.zinoviy23.metricGraphs;
```

```
import com.github.zinoviy23.metricGraphs.api.Identity;
import com.github.zinoviy23.metricGraphs.api.ObjectWithComment;
import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Objects;

import static com.github.zinoviy23.metricGraphs.util.ObjectUtil.*;

public final class Arc implements Identity, ObjectWithComment {
  private static final String POINT_MUST_BE_ON_ARC_ERROR = "Point␣%s␣isn't␣on␣arc.␣[0,␣%f]
      ␣is␣not␣contains␣it";
  private static final String POINT_ID_ALREADY_ASSIGNED_TO_NODE = "Points␣id␣%s␣already␣
      assigned␣to␣node";

  private final String id;
  private final String label;
  private final Node source;
  private final Node target;
  private final String comment;
  private final double length;
  private final List<MovingPoint> points;
  private final boolean distanceToTarget;

  private Arc(@NotNull String id,
              @Nullable String label,
              @NotNull Node source,
              @NotNull Node target,
              @Nullable String comment,
              double length,
              @NotNull List<MovingPoint> points) {
    this.id = Objects.requireNonNull(id, "id");
    this.label = Objects.requireNonNullElse(label, id);
    this.source = Objects.requireNonNull(source, "source");
    this.target = Objects.requireNonNull(target, "target");
    this.comment = comment;
    this.length = length;
    this.points = List.copyOf(checkPointsOnArc(length, Objects.requireNonNull(points, "
        points")));

    if (Node.isInfinity(source) && Node.isInfinity(target)) {
      throw new MetricGraphStructureException(String.format("%s␣and␣%s␣both␣are␣infinity␣
          nodes", source, target));
    }

    distanceToTarget = Node.isInfinity(source);
  }
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.01-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
@Contract(value = "␣->␣new", pure = true)
public static @NotNull ArcBuilder createBuilder() {
  return new ArcBuilderImpl();
}

private static double verifyLength(double length) {
  if (length <= 0) throw new MetricGraphStructureException("length␣must␣be␣greater␣than␣
      0");
  return length;
}

static @NotNull List<MovingPoint> checkPointsOnArc(double arcLength, @NotNull List<
    MovingPoint> points) {
  points.stream()
      .filter(point -> point.getPosition() < 0 || point.getPosition() > arcLength)
      .findAny()
      .ifPresent(point -> {
        throw new MetricGraphStructureException(String.format(POINT_MUST_BE_ON_ARC_ERROR,
            point, arcLength));
      });
  return points;
}

@Override
public @NotNull String getId() {
  return id;
}

public @NotNull String getLabel() {
  return label;
}

public @NotNull Node getSource() {
  return source;
}

public @NotNull Node getTarget() {
  return target;
}

@Override
public @Nullable String getComment() {
  return comment;
}

public double getLength() {
  return length;
}

public @NotNull List<MovingPoint> getPoints() {
```

```
  return points;
}

/**
 * Direction for points movement. <br>
 * If this is true, it means that points positions are relative to arc's target, not
    source. <br>
 * If this is false, which is regular situation, it means, that points positions are
    relative to arc's source.
 * @return true, only if this arc is lead, with infinity source, otherwise false
 */
public boolean isDistanceToTarget() {
  return distanceToTarget;
}

@Override
public boolean equals(Object o) {
  if (this == o) return true;
  if (o == null || getClass() != o.getClass()) return false;
  Arc arc = (Arc) o;
  return id.equals(arc.id);
}

@Override
public int hashCode() {
  return Objects.hash(id);
}

@Override
public String toString() {
  return "Arc{" +
      "id='" + id + '\'' +
      ",␣label='" + label + '\'' +
      ",␣length=" + length +
      '}';
}

/**
 * Too support lazy creation of graphs and allow adding built arcs to graph builder
 * @return wrapper builder, that "builds" this arc
 */
@Contract("␣->␣new")
@NotNull ArcBuilder toBuilder() {
  return new ExactArcBuilder(this);
}

/**
 * Class, because currently java do not support sealed interfaces
 */
public abstract static class ArcBuilder implements Identity {
  private ArcBuilder() {
```

```
}

public abstract @NotNull ArcBuilder setSource(@NotNull Node source);

public abstract @Nullable Node getSource();

public abstract @NotNull ArcBuilder setTarget(@NotNull Node target);

public abstract @Nullable Node getTarget();

public abstract @NotNull ArcBuilder setLength(double length);

public abstract @Nullable Double getLength();

public abstract @NotNull ArcBuilder addPoint(@NotNull MovingPoint point);

public abstract @NotNull Arc createArc();

public abstract @NotNull String getId();

public abstract @NotNull ArcBuilder setId(@NotNull String id);

public abstract @NotNull List<MovingPoint> getPoints();

public abstract @NotNull ArcBuilder setPoints(@NotNull List<MovingPoint> points);

public abstract @Nullable String getLabel();

public abstract @NotNull ArcBuilder setLabel(@Nullable String label);

public abstract @Nullable String getComment();

public abstract @NotNull ArcBuilder setComment(@Nullable String comment);

public abstract @NotNull ArcBuilder copy();

@Override
public final boolean equals(Object o) {
  if (this == o) return true;
  if (!(o instanceof ArcBuilder)) return false;
  ArcBuilder that = (ArcBuilder) o;
  return Objects.equals(getId(), that.getId());
}

@Override
public final int hashCode() {
  return Objects.hash(getId());
}

protected static @NotNull String makeString(@Nullable String id, @Nullable String
    label, @Nullable Double length) {
```

```
    var beginning = "Raw␣Arc{";
    StringBuilder sb = new StringBuilder(beginning);
    if (id != null) {
      sb.append("id='").append(id).append("',␣");
    }
    if (label != null) {
      sb.append("label='").append(label).append("',␣");
    }
    if (length != null) {
      sb.append("length=").append(length).append(",␣");
    }
    if (sb.length() != beginning.length()) {
      sb.delete(sb.length() - 2, sb.length());
    }

    return sb.append("}").toString();
  }
}

private static final class ArcBuilderImpl extends ArcBuilder {
  private String id;
  private String label;
  private Node source;
  private Node target;
  private String comment;
  private Double length;
  private List<MovingPoint> points = new ArrayList<>();

  private ArcBuilderImpl() {
  }

  public @NotNull ArcBuilderImpl setSource(@NotNull Node source) {
    this.source = Objects.requireNonNull(source, "source");
    return this;
  }

  @Override
  public @Nullable Node getSource() {
    return source;
  }

  public @NotNull ArcBuilderImpl setTarget(@NotNull Node target) {
    this.target = Objects.requireNonNull(target, "target");
    return this;
  }

  @Override
  public @Nullable Node getTarget() {
    return target;
  }
```

```
public @NotNull ArcBuilderImpl setLength(double length) {
  this.length = verifyLength(length);
  return this;
}

@Override
public @Nullable Double getLength() {
  return length;
}

public @NotNull ArcBuilderImpl addPoint(@NotNull MovingPoint point) {
  points.add(verifyPointId(Objects.requireNonNull(point, "point"), id));
  return this;
}

@Contract("␣->␣new")
public @NotNull Arc createArc() {
  return new Arc(id, label, source, target, comment, Objects.requireNonNull(length, "
      length"), points);
}

private @NotNull MovingPoint verifyPointId(@NotNull MovingPoint movingPoint, @NotNull
    String id) {
  if (movingPoint.getId().equals(id)) {
    throw new MetricGraphStructureException(String.format(
        POINT_ID_ALREADY_ASSIGNED_TO_NODE, id));
  }
  return movingPoint;
}

public @NotNull String getId() {
  if (id != null) {
    return id;
  }

  throw new IllegalStateException("ArcBuilder␣has␣not␣id");
}

public @NotNull ArcBuilderImpl setId(@NotNull String id) {
  Objects.requireNonNull(id, "id");
  for (var point : points) {
    verifyPointId(point, id);
  }
  this.id = id;
  return this;
}

public @NotNull List<MovingPoint> getPoints() {
  return Collections.unmodifiableList(points);
}
```

```
  public @NotNull ArcBuilderImpl setPoints(@NotNull List<MovingPoint> points) {
    for (var point : Objects.requireNonNull(points, "points")) {
      verifyPointId(Objects.requireNonNull(point, "point␣in␣points"), id);
    }
    this.points = new ArrayList<>(points);
    return this;
  }

  public @Nullable String getLabel() {
    return label;
  }

  public @NotNull ArcBuilderImpl setLabel(@Nullable String label) {
    this.label = label;
    return this;
  }

  public @Nullable String getComment() {
    return comment;
  }

  public @NotNull ArcBuilderImpl setComment(@Nullable String comment) {
    this.comment = comment;
    return this;
  }

  @Override
  public @NotNull ArcBuilder copy() {
    var builder = createBuilder();
    doIfNotNull(id, builder::setId);
    doIfNotNull(label, builder::setLabel);
    doIfNotNull(source, builder::setSource);
    doIfNotNull(target, builder::setTarget);
    doIfNotNull(length, builder::setLength);
    builder.setComment(comment);
    builder.setPoints(points);
    return builder;
  }

  @Override
  public String toString() {
    return makeString(id, label, length);
  }
}

private static final class ExactArcBuilder extends ArcBuilder {
  private final Arc arc;

  private ExactArcBuilder(Arc arc) {
    this.arc = arc;
  }
```

```
@Override
public @NotNull ArcBuilder setSource(@NotNull Node source) {
  throw new UnsupportedOperationException();
}

@Override
public @NotNull Node getSource() {
  return arc.getSource();
}

@Override
public @NotNull ArcBuilder setTarget(@NotNull Node target) {
  throw new UnsupportedOperationException();
}

@Override
public @NotNull Node getTarget() {
  return arc.getTarget();
}

@Override
public @NotNull ArcBuilder setLength(double length) {
  throw new UnsupportedOperationException();
}

@Override
public @NotNull Double getLength() {
  return arc.getLength();
}

@Override
public @NotNull ArcBuilder addPoint(@NotNull MovingPoint point) {
  throw new UnsupportedOperationException();
}

@Override
public @NotNull Arc createArc() {
  return arc;
}

@Override
public @NotNull String getId() {
  return arc.getId();
}

@Override
public @NotNull ArcBuilder setId(@NotNull String id) {
  throw new UnsupportedOperationException();
}
```

```
    @Override
    public @NotNull List<MovingPoint> getPoints() {
      return arc.getPoints();
    }

    @Override
    public @NotNull ArcBuilder setPoints(@NotNull List<MovingPoint> points) {
      throw new UnsupportedOperationException();
    }

    @Override
    public @NotNull String getLabel() {
      return arc.getLabel();
    }

    @Override
    public @NotNull ArcBuilder setLabel(@Nullable String label) {
      throw new UnsupportedOperationException();
    }

    @Override
    public @Nullable String getComment() {
      return arc.getComment();
    }

    @Override
    public @NotNull ArcBuilder setComment(@Nullable String comment) {
      throw new UnsupportedOperationException();
    }

    /**
     * Do not create any new object, because it is immutable, so it can be shared
     * @return {@code this} instance
     */
    @Override
    public @NotNull ArcBuilder copy() {
      return this;
    }

    @Override
    public String toString() {
      return makeString(arc.getId(), arc.getLabel(), arc.getLength());
    }
  }
}
```

1.28. MetricGraph.java

```
package com.github.zinoviy23.metricGraphs;

import com.github.zinoviy23.metricGraphs.api.Identity;
import com.github.zinoviy23.metricGraphs.api.ObjectWithComment;
```

```
import com.github.zinoviy23.metricGraphs.util.DoubleUtil;
import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;
import org.jgrapht.Graph;
import org.jgrapht.graph.AsUnmodifiableGraph;
import org.jgrapht.graph.SimpleDirectedWeightedGraph;

import java.util.*;

public final class MetricGraph implements Identity, ObjectWithComment {
  private final String id;
  private final String label;
  private final String comment;
  private final Graph<Node, Arc> graph;
  private final Map<String, Identity> ids;

  private MetricGraph(@NotNull String id,
                      @Nullable String label,
                      @Nullable String comment,
                      @NotNull Graph<Node, Arc> graph,
                      @NotNull Map<String, Identity> ids) {
    this.id = Objects.requireNonNull(id, "id");
    this.comment = comment;
    this.label = Objects.requireNonNullElse(label, id);
    this.graph = Objects.requireNonNull(graph, "graph");
    this.ids = new HashMap<>(ids);
  }

  @Contract("␣->␣new")
  public static @NotNull MetricGraphBuilder createBuilder() {
    return new MetricGraphBuilder();
  }

  @Override
  public @NotNull String getId() {
    return id;
  }

  public @NotNull Graph<Node, Arc> getGraph() {
    return graph;
  }

  public @NotNull String getLabel() {
    return label;
  }

  @Override
  public @Nullable String getComment() {
    return comment;
  }
```

```
public @Nullable Arc getReversal(@NotNull Arc arc) {
  Objects.requireNonNull(arc, "arc");

  return graph.getEdge(arc.getTarget(), arc.getSource());
}

public @Nullable Node getNode(@NotNull String id) {
  Objects.requireNonNull(id);

  var identity = ids.get(id);
  return identity instanceof Node ? ((Node) identity) : null;
}

public @Nullable Arc getArc(@NotNull String id) {
  Objects.requireNonNull(id);

  var identity = ids.get(id);
  return identity instanceof Arc ? ((Arc) identity) : null;
}

@Override
public boolean equals(Object o) {
  if (this == o) return true;
  if (o == null || getClass() != o.getClass()) return false;
  MetricGraph that = (MetricGraph) o;
  return id.equals(that.id);
}

@Override
public int hashCode() {
  return Objects.hash(id);
}

@Override
public String toString() {
  return "MetricGraph{" +
      "id='" + id + '\'' +
      '}';
}

public static final class MetricGraphBuilder {
  private static final String ALREADY_EXISTS_IN_GRAPH_MESSAGE = "␣already␣exists␣in␣
      graph␣";
  private static final String ID_ALREADY_EXISTS_MESSAGE = "Id␣%s␣already␣assigned␣to␣%s"
      ;

  private final Graph<Node, Arc.ArcBuilder> graph = new SimpleDirectedWeightedGraph<>(
      null, null);

  private final Map<MovingPoint, Arc.ArcBuilder> containingPoints = new HashMap<>();
```

```
private final Map<String, Identity> ids = new HashMap<>();

private String id;
private String label;
private String comment;

private MetricGraphBuilder() {
}

@Contract(value = "␣->␣new", pure = true)
public @NotNull MetricGraph buildGraph() {
  var resultGraph = new SimpleDirectedWeightedGraph<Node, Arc>(null, null);
  for (Node node : graph.vertexSet()) {
    resultGraph.addVertex(node);
  }
  Map<String, Arc> arcIds = new HashMap<>();
  graph.edgeSet().stream()
      .map(Arc.ArcBuilder::createArc)
      .forEach(arc -> {
        resultGraph.addEdge(arc.getSource(), arc.getTarget(), arc);
        resultGraph.setEdgeWeight(arc, arc.getLength());
        arcIds.put(arc.getId(), arc);
      });
  var newIds = new HashMap<>(ids);
  newIds.putAll(arcIds);
  return new MetricGraph(id, label, comment, new AsUnmodifiableGraph<>(resultGraph),
      newIds);
}

public @NotNull MetricGraphBuilder addNode(@NotNull Node node) {
  verifyId(node);
  if (!graph.addVertex(Objects.requireNonNull(node, "node"))) {
    failAlreadyExists(node);
  }
  addId(node);
  return this;
}

public @NotNull ArcWithReversalBuilder addArc(@NotNull Arc.ArcBuilder arcBuilder) {
  Objects.requireNonNull(arcBuilder, "arcBuilder");
  return new ArcWithReversalBuilder(arcBuilder.copy());
}

public @NotNull ArcWithReversalBuilder addArc(@NotNull Arc arc) {
  Objects.requireNonNull(arc, "arc");
  return new ArcWithReversalBuilder(arc.toBuilder());
}

public @NotNull MetricGraphBuilder setLabel(@Nullable String label) {
  this.label = label;
  return this;
```

```
}

public @NotNull MetricGraphBuilder setComment(@Nullable String comment) {
  this.comment = comment;
  return this;
}

public @NotNull MetricGraphBuilder setId(@NotNull String id) {
  Identity identity = new Identity() {
    @Override
    public @NotNull String getId() {
      return id;
    }

    @Override
    public String toString() {
      return "CURRENT␣GRAPH";
    }
  };
  verifyId(identity);

  this.id = id;
  ids.put(id, identity);
  return this;
}

public @NotNull MetricGraphBuilder addPoints(@NotNull String arcId, @NotNull List<
    MovingPoint> points) {
  var arc = ids.get(arcId);
  if (!(arc instanceof Arc.ArcBuilder)) {
    throw new MetricGraphStructureException("Graph␣is␣not␣contain␣arc␣with␣id=" + arcId
        );
  }
  verifyPoints(((Arc.ArcBuilder) arc), points);
  //noinspection ConstantConditions verified while adding arc
  Arc.checkPointsOnArc(((Arc.ArcBuilder) arc).getLength(), points);
  for (MovingPoint point : points) {
    ((Arc.ArcBuilder) arc).addPoint(point);
  }
  addPoints(((Arc.ArcBuilder) arc), points);
  return this;
}

public boolean containsEdge(Node source, Node target) {
  return graph.containsEdge(source, target);
}

private void verifyPoints(@NotNull Arc.ArcBuilder arc, @NotNull List<MovingPoint>
    points) {
  for (var point : points) {
    verifyId(point);
```

```
    if (containingPoints.containsKey(point)) {
      throw new MetricGraphStructureException(point + ALREADY_EXISTS_IN_GRAPH_MESSAGE +
          id + "␣in␣node␣" + arc);
    }
  }
}

private void addPoints(@NotNull Arc.ArcBuilder arc, @NotNull List<MovingPoint> points)
    {
  for (var point : points) {
    addId(point);
    containingPoints.put(point, arc);
  }
}

private void verifyId(@NotNull Identity identity) {
  if (ids.containsKey(identity.getId())) {
    throw new MetricGraphStructureException(
        String.format(ID_ALREADY_EXISTS_MESSAGE, identity.getId(), ids.get(identity.
            getId()))
    );
  }
}

private void addId(@NotNull Identity identity) {
  ids.put(identity.getId(), identity);
}

@Contract("_␣->␣fail")
private void failAlreadyExists(Object o) {
  throw new MetricGraphStructureException(o + ALREADY_EXISTS_IN_GRAPH_MESSAGE + id);
}

public class ArcWithReversalBuilder {
  private final Arc.ArcBuilder currentArc;

  private String comment;
  private String label;

  private ArcWithReversalBuilder(@NotNull Arc.ArcBuilder currentArc) {
    this.currentArc = currentArc;
    verifyArcBuilder(currentArc, "currentArc");
  }

  public @NotNull ArcWithReversalBuilder setReversalComment(@Nullable String comment) {
    this.comment = comment;
    return this;
  }

  public @NotNull ArcWithReversalBuilder setReversalLabel(@Nullable String label) {
    this.label = label;
```

```
  return this;
}

public @NotNull MetricGraphBuilder withReversal(@NotNull String id, MovingPoint
    @NotNull ... points) {
  return withReversal(id, List.of(points));
}

public @NotNull MetricGraphBuilder withReversal(@NotNull String id, @NotNull List<
    MovingPoint> points) {
  //noinspection ConstantConditions verified in class ctor
  return internalWithReversal(Arc.createBuilder()
      .setId(id)
      .setLength(currentArc.getLength())
      .setSource(currentArc.getTarget())
      .setTarget(currentArc.getSource())
      .setLabel(label)
      .setComment(comment)
      .setPoints(points)
  );
}

public @NotNull MetricGraphBuilder withReversal(@NotNull Arc reversalArc) {
  return withReversal(reversalArc.toBuilder());
}

public @NotNull MetricGraphBuilder withReversal(@NotNull Arc.ArcBuilder reversalArc)
    {
  return internalWithReversal(reversalArc.copy());
}

private @NotNull MetricGraphBuilder internalWithReversal(@NotNull Arc.ArcBuilder
    reversalArc) {
  verifyArcBuilder(reversalArc, "reversalArc");

  //noinspection ConstantConditions verified in verifyArcBuilder and ctor
  if (!reversalArc.getTarget().equals(currentArc.getSource()) ||
      !reversalArc.getSource().equals(currentArc.getTarget())) {
    throw new MetricGraphStructureException(String.format("Reversal␣of␣%s␣must␣have␣
        wrong␣source=%s␣and␣target=%s",
        currentArc,
        reversalArc.getSource(),
        reversalArc.getTarget()
    ));
  }

  verifyId(currentArc);
  verifyPoints(currentArc, currentArc.getPoints());
  verifyId(reversalArc);
  verifyPoints(reversalArc, reversalArc.getPoints());
```

```
        //noinspection ConstantConditions verified in verifyArcBuilder and ctor
        if (!DoubleUtil.equals(currentArc.getLength(), reversalArc.getLength())) {
          throw new MetricGraphStructureException(
              String.format("Reversal␣edge␣%s␣of␣%s␣must␣have␣length␣%f", reversalArc,
                  currentArc, currentArc.getLength())
          );
        }

        if (currentArc.getId().equals(reversalArc.getId())) {
          throw new MetricGraphStructureException(String.format(ID_ALREADY_EXISTS_MESSAGE,
              currentArc.getId(), currentArc));
        }

        if (graph.containsEdge(currentArc) || graph.containsEdge(currentArc.getSource(),
            currentArc.getTarget())) {
          failAlreadyExists(currentArc);
        }
        if (graph.containsEdge(reversalArc) || graph.containsEdge(reversalArc.getSource(),
            reversalArc.getTarget())) {
          failAlreadyExists(reversalArc);
        }
        graph.addEdge(currentArc.getSource(), currentArc.getTarget(), currentArc);
        graph.setEdgeWeight(currentArc, currentArc.getLength());
        graph.addEdge(reversalArc.getSource(), reversalArc.getTarget(), reversalArc);
        graph.setEdgeWeight(reversalArc, reversalArc.getLength());

        addId(currentArc);
        addPoints(currentArc, currentArc.getPoints());
        addId(reversalArc);
        addPoints(reversalArc, reversalArc.getPoints());

        return MetricGraphBuilder.this;
      }

      private void verifyArcBuilder(Arc.@NotNull ArcBuilder arcBuilder, final String
          fieldName) {
        Objects.requireNonNull(arcBuilder, fieldName);
        Objects.requireNonNull(arcBuilder.getSource(), fieldName + "␣must␣have␣source");
        Objects.requireNonNull(arcBuilder.getTarget(), fieldName + "␣must␣have␣target");
        Objects.requireNonNull(arcBuilder.getLength(), fieldName + "␣must␣have␣length");
      }
    }
  }
}
```

1.29. GraphLayout.java

```
package com.github.zinoviy23.metricGraphs.util;

import org.jetbrains.annotations.NotNull;
import org.jgrapht.Graph;
import org.jgrapht.alg.drawing.FRLayoutAlgorithm2D;
```

```
import org.jgrapht.alg.drawing.IndexedFRLayoutAlgorithm2D;
import org.jgrapht.alg.drawing.model.Box2D;
import org.jgrapht.alg.drawing.model.LayoutModel2D;
import org.jgrapht.alg.drawing.model.MapLayoutModel2D;

import java.util.Random;

public class GraphLayout {
  private final long randomSeed;
  private final int iterations;
  private final double thetaFactor;
  private final double normalizationFactor;

  public GraphLayout() {
    this(11);
  }

  public GraphLayout(long randomSeed) {
    this(randomSeed, 200);
  }

  public GraphLayout(long randomSeed, int iterations) {
    this(randomSeed, iterations, IndexedFRLayoutAlgorithm2D.DEFAULT_THETA_FACTOR);
  }

  public GraphLayout(long randomSeed, int iterations, double thetaFactor) {
    this(randomSeed, iterations, thetaFactor, FRLayoutAlgorithm2D.
        DEFAULT_NORMALIZATION_FACTOR);
  }

  public GraphLayout(long randomSeed, int iterations, double thetaFactor, double
      normalizationFactor) {
    this.randomSeed = randomSeed;
    this.iterations = iterations;
    this.thetaFactor = thetaFactor;
    this.normalizationFactor = normalizationFactor;
  }

  public <V, E> @NotNull LayoutModel2D<V> createLayout(Graph<V, E> graph, int vertexSize,
      int sizeOffset) {
    Random random = new Random(randomSeed);

    var layoutAlgorithm2D = new IndexedFRLayoutAlgorithm2D<V, E>(
        iterations,
        thetaFactor,
        normalizationFactor,
        random
    );

    var size = calcSize(graph.vertexSet().size(), vertexSize, sizeOffset);
    MapLayoutModel2D<V> layoutModel2D = new MapLayoutModel2D<>(new Box2D(size, size));
```

```
    layoutAlgorithm2D.layout(graph, layoutModel2D);

    return layoutModel2D;
  }

  private static int calcSize(int vertexCount, int vertexSize, int sizeOffset) {
    return (int) Math.ceil(Math.sqrt(vertexCount) * vertexSize) + sizeOffset;
  }
}
```

1.30. ContainerUtil.java

```
package com.github.zinoviy23.metricGraphs.util;

import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

import java.util.Iterator;
import java.util.Map;
import java.util.stream.Stream;

public class ContainerUtil {
  public static <T> @Nullable T first(@NotNull Iterable<T> iterable) {
    Iterator<T> iterator = iterable.iterator();
    if (iterator.hasNext()) {
      return iterator.next();
    }
    return null;
  }

  public static <T> @Nullable T second(@NotNull Iterable<T> iterable) {
    Iterator<T> iterator = iterable.iterator();
    if (!iterator.hasNext()) {
      return null;
    }
    iterator.next();
    if (!iterator.hasNext()) {
      return null;
    }
    return iterator.next();
  }

  public static <T1, T2, T3> @Nullable T3 getFromTable(Map<T1, Map<T2, T3>> table, T1 row,
      T2 column) {
    var tmpRow = table.get(row);
    if (tmpRow == null) return null;

    return tmpRow.get(column);
  }

  /**
   * Should be used once. You should close given stream by your own, after iteration
```

```
   *
   * @return iterable, wrapping this stream.
   */
  public static <T> @NotNull Iterable<T> iterate(Stream<T> stream) {
    return stream::iterator;
  }
}
```

1.31. Ref.java

```
package com.github.zinoviy23.metricGraphs.util;

import org.jetbrains.annotations.NotNull;

import java.util.function.UnaryOperator;

public class Ref<T> {
  private T data;

  public Ref(T data) {
    this.data = data;
  }

  public Ref() {
  }

  public T getData() {
    return data;
  }

  public void setData(T data) {
    this.data = data;
  }

  /**
   * Updates containing value
   * @param updateOperator operator for updating value
   * @return previous value
   */
  public T update(@NotNull UnaryOperator<T> updateOperator) {
    T prev = data;
    data = updateOperator.apply(data);
    return prev;
  }
}
```

1.32. ObjectUtil.java

```
package com.github.zinoviy23.metricGraphs.util;

import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;
```

```
import java.util.function.Consumer;

public class ObjectUtil {
  private ObjectUtil() {
  }

  public static <T> void doIfNotNull(@Nullable T t, Consumer<@NotNull T> consumer) {
    if (t != null) {
      consumer.accept(t);
    }
  }
}
```

1.33. DoubleUtil.java

```
package com.github.zinoviy23.metricGraphs.util;

public class DoubleUtil {
  public static final double DELTA = 1e-5;

  public static boolean equals(double a, double b) {
    // Check infinities
    if (Double.isInfinite(a) && Double.isInfinite(b) && a > 0 && b > 0) return true;

    return Math.abs(a - b) < DELTA;
  }
}
```

1.34. ThrowableBiConsumer.java

```
package com.github.zinoviy23.metricGraphs.util;

@FunctionalInterface
public interface ThrowableBiConsumer<T, E, TExp extends Throwable> {
  void consume(T t, E e) throws TExp;
}
```

1.35. MetricGraphWriter.java

```
package com.github.zinoviy23.metricGraphs.io;

import com.github.zinoviy23.metricGraphs.MetricGraph;
import org.jetbrains.annotations.NotNull;

import java.io.Closeable;

public interface MetricGraphWriter<T extends Throwable> extends AutoCloseable, Closeable {
  void write(@NotNull MetricGraph graph) throws T;
}
```

1.36. IoUtils.java

```
package com.github.zinoviy23.metricGraphs.io;

import com.fasterxml.jackson.core.JsonFactory;

final class IoUtils {
  static final JsonFactory factory = new JsonFactory();

  static final String LENGTH = "length";
  static final String DIRECTED = "directed";
  static final String ID = "id";
  static final String GRAPH = "graph";
  static final String ALREADY_WRITE_GRAPH_TO_STREAM_MESSAGE = "Already␣write␣graph␣to␣
      stream!";
  static final String ALREADY_READ_GRAPH_FROM_STREAM_MESSAGE = "Already␣read␣graph␣from␣
      stream";
  static final String TYPE = "type";
  static final String METRIC_GRAPH = "metric␣graph";
  static final String LABEL = "label";
  static final String NODES = "nodes";
  static final String EDGES = "edges";
  static final String SOURCE = "source";
  static final String TARGET = "target";
  static final String RELATION = "relation";
  static final String NODES_CONNECTION = "nodes␣connection";
  static final String METADATA = "metadata";
  static final String POINTS = "points";
  static final String POSITION = "position";
  static final String COMMENT = "comment";

  private IoUtils() {
  }
}
```

1.37. MetricGraphJsonReader.java

```
package com.github.zinoviy23.metricGraphs.io;

import com.fasterxml.jackson.core.JsonLocation;
import com.fasterxml.jackson.core.JsonParseException;
import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.core.JsonToken;
import com.github.zinoviy23.metricGraphs.Arc;
import com.github.zinoviy23.metricGraphs.MetricGraph;
import com.github.zinoviy23.metricGraphs.MovingPoint;
import com.github.zinoviy23.metricGraphs.Node;
import com.github.zinoviy23.metricGraphs.util.ContainerUtil;
import com.github.zinoviy23.metricGraphs.util.Ref;
import com.github.zinoviy23.metricGraphs.util.ThrowableBiConsumer;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;
```

```
import org.jgrapht.alg.util.Pair;
import org.jgrapht.alg.util.Triple;

import java.io.*;
import java.util.*;

public final class MetricGraphJsonReader implements AutoCloseable, Closeable {
  private static final Logger LOG = LogManager.getLogger(MetricGraphJsonReader.class);

  private static final String GRAPH_HAS_EDGES_BUT_HASN_T_NODES_MESSAGE = "Graph␣has␣edges,
      ␣but␣hasn't␣nodes";
  private static final String HASN_T_ANY_NODES_WITH_ID_MESSAGE = "Hasn't␣any␣nodes␣with␣id
      =";
  private static final String ARC_S_MUST_HAVE_REVERSAL_EDGE = "Arc␣%s␣must␣have␣reversal␣
      edge!";

  private final JsonParser parser;
  private boolean isRead;
  private boolean errorOccurred;

  public MetricGraphJsonReader(@NotNull Reader reader) throws IOException {
    parser = IoUtils.factory.createParser(reader);
  }

  public MetricGraphJsonReader(@NotNull InputStream inputStream) throws IOException {
    parser = IoUtils.factory.createParser(inputStream);
  }

  public MetricGraphJsonReader(@NotNull File file) throws IOException {
    parser = IoUtils.factory.createParser(file);
  }

  public MetricGraphJsonReader(@NotNull String source) throws IOException {
    parser = IoUtils.factory.createParser(source);
  }

  public @Nullable MetricGraph read() throws IOException {
    if (isRead || errorOccurred) {
      throw new IllegalStateException(IoUtils.ALREADY_READ_GRAPH_FROM_STREAM_MESSAGE);
    }

    try {
      var graph = internalRead();
      isRead = true;
      return graph;
    } catch (JsonParseException e) {
      errorOccurred = true;
      throw new MetricGraphReadingException(e.getMessage());
    } catch (IOException e) {
      errorOccurred = true;
      throw new IOException(e);
```

```
  }
}

private @Nullable MetricGraph internalRead() throws IOException {
  while (parser.nextToken() != JsonToken.END_OBJECT) {
    String currentName = parser.getCurrentName();
    if (IoUtils.GRAPH.equals(currentName)) {
      return readGraph();
    }
  }

  return null;
}

private @NotNull MetricGraph readGraph() throws IOException {
  var graphBuilder = MetricGraph.createBuilder();
  Map<String, Node> nodes = null;
  Map<String, Map<String, Arc.ArcBuilder>> edgesInfo = null;
  while (parser.nextToken() != JsonToken.END_OBJECT) {
    var currentName = parser.getCurrentName();
    if (IoUtils.ID.equals(currentName)) {
      parser.nextToken();
      graphBuilder.setId(parser.getValueAsString());
    }

    if (IoUtils.LABEL.equals(currentName)) {
      parser.nextToken();
      graphBuilder.setLabel(parser.getValueAsString());
    }

    if (IoUtils.NODES.equals(currentName)) {
      parser.nextToken();
      nodes = readNodes(graphBuilder);
    }

    if (IoUtils.EDGES.equals(currentName)) {
      parser.nextToken();
      edgesInfo = readEdges();
    }

    if (IoUtils.METADATA.equals(currentName)) {
      fetchMetadata((name, balance) -> {
        if (IoUtils.COMMENT.equals(name) && balance == 1) {
          parser.nextToken();
          graphBuilder.setComment(parser.getValueAsString());
        }
      });
    }
  }

  if (edgesInfo != null && nodes != null) {
```

```
    Set<String> processedArcs = new HashSet<>();

    var edgesStream = edgesInfo.entrySet().stream()
        .flatMap(mapEntry ->
           mapEntry.getValue().entrySet().stream()
               .map(targetAndBuilder -> Triple.of(mapEntry.getKey(),
                   targetAndBuilder.getKey(),
                   targetAndBuilder.getValue()
                   )
               )
        );

    for (var triple : ContainerUtil.iterate(edgesStream)) {
      if (processedArcs.contains(triple.getThird().getId())) continue;

      var stringStringPair = addArcAndReversal(graphBuilder, nodes, edgesInfo, triple);
      processedArcs.add(stringStringPair.getFirst());
      processedArcs.add(stringStringPair.getSecond());
    }
  } else if (edgesInfo != null) {
    throw new MetricGraphReadingException(GRAPH_HAS_EDGES_BUT_HASN_T_NODES_MESSAGE);
  }

  return graphBuilder.buildGraph();
}

private @NotNull Pair<String, String> addArcAndReversal(@NotNull MetricGraph.
    MetricGraphBuilder graphBuilder,
                                                        @NotNull Map<String, Node> finalNodes,
                                                        @NotNull Map<String, Map<String, Arc.
                                                            ArcBuilder>> finalEdgesInfo,
                                                        @NotNull Triple<String, String, Arc.
                                                            ArcBuilder> triple) {
  var source = finalNodes.get(triple.getFirst());
  var target = finalNodes.get(triple.getSecond());
  if (source == null) {
    throw new MetricGraphReadingException(HASN_T_ANY_NODES_WITH_ID_MESSAGE + triple.
        getFirst());
  }
  if (target == null) {
    throw new MetricGraphReadingException(HASN_T_ANY_NODES_WITH_ID_MESSAGE + triple.
        getSecond());
  }

  var arcBuilder = triple.getThird();
  var reversal = ContainerUtil.getFromTable(finalEdgesInfo, triple.getSecond(), triple.
      getFirst());
  if (reversal == null) {
    throw new MetricGraphReadingException(String.format(ARC_S_MUST_HAVE_REVERSAL_EDGE,
        arcBuilder.getId()));
  }
```

```
  var arc = arcBuilder
      .setSource(source)
      .setTarget(target)
      .createArc();
  graphBuilder
      .addArc(arc)
      .setReversalComment(reversal.getComment())
      .setReversalLabel(reversal.getLabel())
      .withReversal(reversal.getId(), reversal.getPoints());

  return Pair.of(arc.getId(), reversal.getId());
}

private @NotNull Map<String, Map<String, Arc.ArcBuilder>> readEdges() throws IOException
    {
  var result = new HashMap<String, Map<String, Arc.ArcBuilder>>();
  while (parser.nextToken() != JsonToken.END_ARRAY) {
    if (parser.currentToken() == JsonToken.START_OBJECT) {
      var edgeInfo = readEdge();
      result.computeIfAbsent(edgeInfo.getFirst(), (k) -> new HashMap<>())
          .put(edgeInfo.getSecond(), edgeInfo.getThird());
    }
  }
  return result;
}

private @NotNull Triple<String, String, Arc.ArcBuilder> readEdge() throws IOException {
  var arcBuilder = Arc.createBuilder();
  String source = null;
  String target = null;

  while (parser.nextToken() != JsonToken.END_OBJECT) {
    var currentName = parser.getCurrentName();
    if (IoUtils.LABEL.equals(currentName)) {
      parser.nextToken();
      arcBuilder.setLabel(parser.getValueAsString());
    }
    if (IoUtils.SOURCE.equals(currentName)) {
      parser.nextToken();
      source = parser.getValueAsString();
    }
    if (IoUtils.TARGET.equals(currentName)) {
      parser.nextToken();
      target = parser.getValueAsString();
    }
    if (IoUtils.ID.equals(currentName)) {
      parser.nextToken();
      var valueAsString = parser.getValueAsString();
      arcBuilder.setId(valueAsString);
      LOG.debug(() -> "read␣edge␣" + valueAsString);
```

```
    }
    if (IoUtils.METADATA.equals(currentName)) {
      readArcMetadata(arcBuilder);
    }
  }

  return Triple.of(source, target, arcBuilder);
}

private void readArcMetadata(@NotNull Arc.ArcBuilder arcBuilder) throws IOException {
  fetchMetadata((name, balance) -> {
    if (IoUtils.COMMENT.equals(name) && balance == 1) {
      parser.nextToken();
      arcBuilder.setComment(parser.getValueAsString());
    }
    if (IoUtils.POINTS.equals(name) && balance == 1) {
      parser.nextToken();
      readPoints(arcBuilder);
    }
    if (IoUtils.LENGTH.equals(name) && balance == 1) {
      parser.nextToken();
      if (parser.getCurrentToken().isNumeric()) {
        arcBuilder.setLength(parser.getNumberValue().doubleValue());
      } else if ("Infinity".equals(parser.getValueAsString())) {
        arcBuilder.setLength(Double.POSITIVE_INFINITY);
      } else {
        throw new MetricGraphReadingException("Unexpected␣value:␣" + parser.
            getValueAsString());
      }
    }
  });
}

private void readPoints(@NotNull Arc.ArcBuilder arcBuilder) throws IOException {
  while (parser.nextToken() != JsonToken.END_ARRAY) {
    if (parser.getCurrentToken() == JsonToken.START_OBJECT) {
      arcBuilder.addPoint(readPoint());
    }
  }
}

private MovingPoint readPoint() throws IOException {
  String id = null;
  Double position = null;
  Ref<String> comment = new Ref<>();

  while (parser.nextToken() != JsonToken.END_OBJECT) {
    var currentName = parser.getCurrentName();
    if (IoUtils.ID.equals(currentName)) {
      parser.nextToken();
      id = parser.getValueAsString();
```

```
      String finalId = id;
      LOG.debug(() -> "read point " + finalId);
    }
    if (IoUtils.POSITION.equals(currentName)) {
      parser.nextToken();
      position = parser.getNumberValue().doubleValue();
    }
    if (IoUtils.METADATA.equals(currentName)) {
      fetchMetadata((name, balance) -> {
        // comment must be property of metadata obj, not its child
        if (IoUtils.COMMENT.equals(name) && balance == 1) {
          parser.nextToken();
          comment.setData(parser.getValueAsString());
        }
      });
    }
  }

  //noinspection ConstantConditions
  return new MovingPoint(id, position, comment.getData());
}

private @NotNull Map<String, Node> readNodes(@NotNull MetricGraph.MetricGraphBuilder
    graphBuilder) throws IOException {
  Map<String, Node> nodes = new HashMap<>();
  while (parser.nextToken() != JsonToken.END_OBJECT) {
    LOG.debug(() -> "read nodes");
    String id = parser.getCurrentName();
    if (id != null) {
      var node = readNode(id);
      graphBuilder.addNode(node);
      nodes.put(id, node);
    }
  }
  return nodes;
}

private @NotNull Node readNode(@NotNull String id) throws IOException {
  String label = null;
  Ref<String> comment = new Ref<>();
  while (parser.nextToken() != JsonToken.END_OBJECT) {
    if (IoUtils.LABEL.equals(parser.getCurrentName())) {
      LOG.debug(() -> "read node " + id);
      parser.nextToken();
      label = parser.getValueAsString();
    }
    if (IoUtils.METADATA.equals(parser.getCurrentName())) {
      fetchMetadata((name, balance) -> {
        // comment must be property of metadata obj, not its child
        if (IoUtils.COMMENT.equals(name) && balance == 1) {
          parser.nextToken();
```

```
          comment.setData(parser.getValueAsString());
        }
      });
    }
  }
  if (Node.INFINITY_NODE_LABEL.equals(label)) {
    return Node.createInfinity(id);
  }
  return Node.createNode(id, label, comment.getData());
}

private void fetchMetadata(ThrowableBiConsumer<String, Integer, IOException> fetcher)
    throws IOException {
  parser.nextToken();
  if (parser.currentToken() != JsonToken.START_OBJECT) {
    parser.nextToken();
    return;
  }
  int balance = 1;
  while (balance != 0) {
    parser.nextToken();
    if (parser.currentToken() == JsonToken.START_OBJECT) {
      balance++;
    }
    if (parser.currentToken() == JsonToken.END_OBJECT) {
      balance--;
    }
    fetcher.consume(parser.getCurrentName(), balance);
  }
}

public @NotNull JsonLocation lastLocation() {
  return parser.getCurrentLocation();
}

@Override
public void close() throws IOException {
  parser.close();
}
}
```

1.38. MetricGraphReadingException.java

```
package com.github.zinoviy23.metricGraphs.io;

public class MetricGraphReadingException extends RuntimeException {
  public MetricGraphReadingException(String message) {
    super(message);
  }

  public MetricGraphReadingException(String message, Throwable cause) {
    super(message, cause);
```

```
  }
}
```

1.39. ValidatedMetricGraphJsonReader.java

```
package com.github.zinoviy23.metricGraphs.io;

import org.everit.json.schema.ValidationException;
import org.everit.json.schema.loader.SchemaLoader;
import org.jetbrains.annotations.NotNull;
import org.json.JSONObject;
import org.json.JSONTokener;

import java.io.*;

public class ValidatedMetricGraphJsonReader {
  public static final String JSON_GRAPH_SCHEMA_JSON = "/json-graph-schema.json";
  private final MetricGraphJsonReader reader;

  public ValidatedMetricGraphJsonReader(@NotNull File file) throws IOException {
    try (var bufferedReader = new BufferedReader(new FileReader(file))) {
      validate(bufferedReader);
    } catch (ValidationException e) {
      throw new IOException(String.join("\n", e.getAllMessages()));
    }
    reader = new MetricGraphJsonReader(file);
  }

  public ValidatedMetricGraphJsonReader(@NotNull String string) throws IOException {
    validate(new StringReader(string));
    reader = new MetricGraphJsonReader(string);
  }

  private static void validate(@NotNull Reader reader) {
    JSONObject jsonSchema = new JSONObject(new JSONTokener(
        ValidatedMetricGraphJsonReader.class.getResourceAsStream(JSON_GRAPH_SCHEMA_JSON)
    ));
    JSONObject jsonGraph = new JSONObject(new JSONTokener(reader));
    var schema = SchemaLoader.load(jsonSchema);
    schema.validate(jsonGraph);
  }

  public @NotNull MetricGraphJsonReader getReader() {
    return reader;
  }
}
```

1.40. MetricGraphJsonWriter.java

```
package com.github.zinoviy23.metricGraphs.io;

import com.fasterxml.jackson.core.JsonGenerator;
```

```
import com.github.zinoviy23.metricGraphs.Arc;
import com.github.zinoviy23.metricGraphs.MetricGraph;
import com.github.zinoviy23.metricGraphs.api.ObjectWithComment;
import org.jetbrains.annotations.NotNull;

import java.io.IOException;
import java.io.OutputStream;
import java.io.Writer;

public final class MetricGraphJsonWriter implements MetricGraphWriter<IOException> {

  private final JsonGenerator generator;
  private boolean isWritten;
  private boolean errorOccurred;

  public MetricGraphJsonWriter(@NotNull OutputStream outputStream, boolean
      usePrettyPrinter) throws IOException {
    this(IoUtils.factory.createGenerator(outputStream), usePrettyPrinter);
  }

  public MetricGraphJsonWriter(@NotNull Writer writer, boolean usePrettyPrinter) throws
      IOException {
    this(IoUtils.factory.createGenerator(writer), usePrettyPrinter);
  }

  private MetricGraphJsonWriter(@NotNull JsonGenerator generator, boolean usePrettyPrint)
      {
    this.generator = generator;
    if (usePrettyPrint) {
      this.generator.useDefaultPrettyPrinter();
    }
  }

  public void write(@NotNull MetricGraph graph) throws IOException {
    if (isWritten || errorOccurred) throw new IllegalStateException(IoUtils.
        ALREADY_WRITE_GRAPH_TO_STREAM_MESSAGE);

    try {
      internalWrite(graph);
    } catch (IOException e) {
      errorOccurred = true;
      throw new IOException(e);
    }

    isWritten = true;
  }

  private void internalWrite(@NotNull MetricGraph graph) throws IOException {
    generator.writeStartObject();
    generator.writeObjectFieldStart(IoUtils.GRAPH);
```

```
  writeGraphInfo(graph);
  writeNodes(graph);
  writeEdges(graph);
  writeComment(graph);

  generator.writeEndObject();
  generator.writeEndObject();
}

private void writeGraphInfo(@NotNull MetricGraph graph) throws IOException {
  generator.writeBooleanField(IoUtils.DIRECTED, true);
  generator.writeStringField(IoUtils.ID, graph.getId());
  generator.writeStringField(IoUtils.TYPE, IoUtils.METRIC_GRAPH);
  generator.writeStringField(IoUtils.LABEL, graph.getLabel());
}

private void writeNodes(@NotNull MetricGraph graph) throws IOException {
  generator.writeObjectFieldStart(IoUtils.NODES);

  for (var node : graph.getGraph().vertexSet()) {
    generator.writeObjectFieldStart(node.getId());
    generator.writeStringField(IoUtils.LABEL, node.getLabel());
    writeComment(node);
    generator.writeEndObject();
  }
  generator.writeEndObject();
}

private void writeEdges(@NotNull MetricGraph graph) throws IOException {
  generator.writeArrayFieldStart(IoUtils.EDGES);

  for (var arc : graph.getGraph().edgeSet()) {
    generator.writeStartObject();
    generator.writeStringField(IoUtils.ID, arc.getId());
    generator.writeStringField(IoUtils.SOURCE, arc.getSource().getId());
    generator.writeStringField(IoUtils.TARGET, arc.getTarget().getId());
    generator.writeStringField(IoUtils.RELATION, IoUtils.NODES_CONNECTION);
    generator.writeStringField(IoUtils.LABEL, arc.getLabel());
    writeArcMetadata(arc);
    generator.writeEndObject();
  }

  generator.writeEndArray();
}

private void writeArcMetadata(@NotNull Arc arc) throws IOException {
  generator.writeObjectFieldStart(IoUtils.METADATA);

  generator.writeArrayFieldStart(IoUtils.POINTS);
  for (var point : arc.getPoints()) {
    generator.writeStartObject();
```

```
      generator.writeStringField(IoUtils.ID, point.getId());
      generator.writeNumberField(IoUtils.POSITION, point.getPosition());
      writeComment(point);
      generator.writeEndObject();
    }
    generator.writeEndArray();

    if (arc.getComment() != null) {
      generator.writeStringField(IoUtils.COMMENT, arc.getComment());
    }

    generator.writeNumberField(IoUtils.LENGTH, arc.getLength());

    generator.writeEndObject();
  }

  private void writeComment(@NotNull ObjectWithComment objectWithComment) throws
      IOException {
    if (objectWithComment.getComment() == null) return;

    generator.writeObjectFieldStart(IoUtils.METADATA);
    generator.writeStringField(IoUtils.COMMENT, objectWithComment.getComment());

    generator.writeEndObject();
  }


  @Override
  public void close() throws IOException {
    generator.close();
  }
}
```

1.41. XmlUtils.java

```
package com.github.zinoviy23.metricGraphs.io.graphml;

import javax.xml.stream.XMLOutputFactory;
import javax.xml.transform.TransformerFactory;

class XmlUtils {
  static final XMLOutputFactory xmlOutputFactory = XMLOutputFactory.newFactory();
  static final TransformerFactory transformerFactory = TransformerFactory.newInstance();

  static final String ALREADY_WRITE_GRAPH_TO_STREAM_MESSAGE = "Already␣write␣graph␣to␣
      stream!";
}
```

1.42. MetricGraphYedWriter.java

```
package com.github.zinoviy23.metricGraphs.io.graphml;
```

```
import com.github.zinoviy23.metricGraphs.Arc;
import com.github.zinoviy23.metricGraphs.MetricGraph;
import com.github.zinoviy23.metricGraphs.MovingPoint;
import com.github.zinoviy23.metricGraphs.Node;
import com.github.zinoviy23.metricGraphs.io.MetricGraphWriter;
import com.github.zinoviy23.metricGraphs.util.GraphLayout;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;
import org.jgrapht.alg.drawing.model.LayoutModel2D;

import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.TransformerException;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;
import java.io.*;
import java.util.HashSet;
import java.util.Set;

public final class MetricGraphYedWriter implements MetricGraphWriter<IOException> {
  private boolean isWritten;
  private boolean errorOccurred;
  private final Writer resultWriter;
  private final File currentFile;
  private final boolean applyLayout;

  public MetricGraphYedWriter(@NotNull Writer writer, boolean applyLayout) throws
      IOException {
    currentFile = File.createTempFile("result", ".xml");
    resultWriter = writer;
    this.applyLayout = applyLayout;
  }

  public void write(@NotNull MetricGraph graph) throws IOException {
    if (isWritten || errorOccurred) {
      throw new IllegalStateException(XmlUtils.ALREADY_WRITE_GRAPH_TO_STREAM_MESSAGE);
    }

    try (var fileWriter = new FileWriter(currentFile); var writer = new YedWriter(
        fileWriter, applyLayout)) {
      writer.write(graph);
    } catch (XMLStreamException e) {
      errorOccurred = true;
      throw new IOException(e);
    }
    try (var fileReader = new FileReader(currentFile)) {
      addIndents(fileReader);
    } catch (TransformerException e) {
      errorOccurred = true;
      throw new IOException(e);
```

```
  }
  isWritten = true;
}

private void addIndents(@NotNull Reader reader) throws TransformerException {
  var transformer = XmlUtils.transformerFactory.newTransformer();
  transformer.setOutputProperty(OutputKeys.INDENT, "yes");
  transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "2");
  transformer.transform(new StreamSource(reader), new StreamResult(resultWriter));
}

@Override
public void close() throws IOException {
  if (!currentFile.delete()) {
    throw new IOException("Cannot␣delete␣cache␣file!");
  }
}

private static final class YedWriter implements MetricGraphWriter<XMLStreamException> {
  public static final String EDGE_POINTS = "d12";
  private static final String GRAPH_DESCR = "d0";
  private static final String NODE_DESCR = "d5";
  private static final String EDGE_DESCR = "d9";
  private static final String NODE_GRAPHICS = "d6";
  private static final String Y = "http://www.yworks.com/xml/graphml";
  private static final String EDGE_GRAPHICS = "d10";
  public static final String EDGE_LENGTH = "d11";

  private final XMLStreamWriter writer;
  private final boolean applyLayout;

  private final Set<Arc> drownArc = new HashSet<>();

  private YedWriter(Writer writer, boolean applyLayout) throws XMLStreamException {
    this.writer = XmlUtils.xmlOutputFactory.createXMLStreamWriter(writer);
    this.applyLayout = applyLayout;
  }

  public void write(@NotNull MetricGraph graph) throws XMLStreamException {
    writer.writeStartDocument();
    writer.writeStartElement("graphml");
    writeYedSchemes();
    writeDataElements();
    writeGraph(graph);
    writer.writeEndDocument();
  }

  private void writeYedSchemes() throws XMLStreamException {
    writer.writeNamespace("xmlns", "http://graphml.graphdrawing.org/xmlns");
    writer.writeNamespace( "java", "http://www.yworks.com/xml/yfiles-common/1.0/java");
    writer.writeNamespace( "sys", "http://www.yworks.com/xml/yfiles-common/markup/
```

```
      primitives/2.0");
  writer.writeNamespace( "x", "http://www.yworks.com/xml/yfiles-common/markup/2.0");
  var xsi = "http://www.w3.org/2001/XMLSchema-instance";
  writer.writeNamespace( "xsi", xsi);
  writer.writeNamespace( "y", Y);
  writer.writeNamespace( "yed", "http://www.yworks.com/xml/yed/3");
  writer.writeAttribute(xsi, "schemaLocation", "http://graphml.graphdrawing.org/xmlns␣
      http://www.yworks.com/xml/schema/graphml/1.1/ygraphml.xsd");
}

private void writeDataElements() throws XMLStreamException {
  writeDataDef(GRAPH_DESCR, "graph", "Description", "string", null);
  writeDataDef("d1", "port", null, null, "portgraphics");
  writeDataDef("d2", "port", null, null, "portgeometry");
  writeDataDef("d3", "port", null, null, "portuserdata");
  writeDataDef("d4", "node", "url", "string", null);
  writeDataDef(NODE_DESCR, "node", "description", "string", null);
  writeDataDef(NODE_GRAPHICS, "node", null, null, "nodegraphics");
  writeDataDef("d7", "graphml", null, null, "resources");
  writeDataDef("d8", "edge", "url", "string", null);
  writeDataDef(EDGE_DESCR, "edge", "description", "string", null);
  writeDataDef(EDGE_GRAPHICS, "edge", null, null, "edgegraphics");
  writeDataDef(EDGE_LENGTH, "edge", "length", "double", null);
  writeDataDef(EDGE_POINTS, "edge", "points", null, null);
}

private void writeDataDef(@NotNull String id,
                          @NotNull String forAttr,
                          @Nullable String name,
                          @Nullable String type,
                          @Nullable String yfilesType) throws XMLStreamException {
  writer.writeStartElement("key");
  writer.writeAttribute("id", id);
  writer.writeAttribute("for", forAttr);
  if (name != null) {
    writer.writeAttribute("attr.name", name);
  }
  if (type != null) {
    writer.writeAttribute("attr.type", type);
  }
  if (yfilesType != null) {
    writer.writeAttribute("yfiles.type", yfilesType);
  }
  writer.writeEndElement();
}

private void writeGraph(MetricGraph graph) throws XMLStreamException {
  writer.writeStartElement("graph");
  writer.writeAttribute("edgedefault", "directed");
  writer.writeAttribute("id", graph.getId());
  writer.writeStartElement("data");
```

```
  writer.writeAttribute("key", GRAPH_DESCR);
  writer.writeCharacters(graph.getComment());
  writer.writeEndElement();

  LayoutModel2D<Node> layoutModel2D = applyLayout
      ? new GraphLayout().createLayout(graph.getGraph(), 200, 300)
      : null;

  for (Node node : graph.getGraph().vertexSet()) {
    writeNode(node, layoutModel2D);
  }

  for (Arc arc : graph.getGraph().edgeSet()) {
    writeArc(arc, graph);
  }
}

private void writeArc(@NotNull Arc arc, @NotNull MetricGraph graph) throws
    XMLStreamException {
  writer.writeStartElement("edge");
  writer.writeAttribute("id", arc.getId());
  writer.writeAttribute("source", arc.getSource().getId());
  writer.writeAttribute("target", arc.getTarget().getId());

  writeComment(arc.getComment(), EDGE_DESCR);

  writeArcGraphics(arc, drownArc.contains(graph.getReversal(arc)));

  writer.writeStartElement("data");
  writer.writeAttribute("key", EDGE_LENGTH);
  writer.writeCharacters(Double.toString(arc.getLength()));
  writer.writeEndElement();

  writer.writeStartElement("data");
  writer.writeAttribute("key", EDGE_POINTS);
  for (MovingPoint point : arc.getPoints()) {
    writePoint(point);
  }
  writer.writeEndElement();

  writer.writeEndElement();

  drownArc.add(arc);
}

private void writePoint(@NotNull MovingPoint point) throws XMLStreamException {
  writer.writeStartElement("point");
  writer.writeAttribute("id", point.getId());
  writer.writeAttribute("position", Double.toString(point.getPosition()));
  if (point.getComment() != null) {
    writer.writeCharacters(point.getComment());
```

```
  }
  writer.writeEndElement();
}

private void writeArcGraphics(@NotNull Arc arc, boolean hide) throws
    XMLStreamException {
  writer.writeStartElement("data");
  writer.writeAttribute("key", EDGE_GRAPHICS);
  writer.writeStartElement(Y, "PolyLineEdge");

  writer.writeStartElement(Y, "EdgeLabel");
  if (hide) {
    writer.writeAttribute("visible", "false");
  }
  writer.writeCharacters(arc.getLabel() + ",␣" + arc.getLength());
  writer.writeEndElement();

  writer.writeStartElement(Y, "Path");
  writer.writeAttribute("sx", !hide ? "0.0" : "10.0");
  writer.writeAttribute("sy", "0.0");
  writer.writeAttribute("tx", !hide ? "0.0" : "10.0");
  writer.writeAttribute("ty", "0.0");
  writer.writeEndElement();

  writer.writeStartElement(Y, "LineStyle");
  writer.writeAttribute("color", hide ? "#00000000" : "#000000");
  writer.writeAttribute("width", "1.0");
  writer.writeEndElement();

  writer.writeStartElement(Y, "Arrows");
  writer.writeAttribute("source", "none");
  writer.writeAttribute("target", "none");
  writer.writeEndElement();

  writer.writeStartElement(Y, "BendElement");
  writer.writeAttribute("smoothed", "false");
  writer.writeEndElement();

  writer.writeEndElement();
  writer.writeEndElement();
}

private void writeComment(String comment, String descr) throws XMLStreamException {
  if (comment != null) {
    writer.writeStartElement("data");
    writer.writeAttribute("key", descr);
    writer.writeCharacters(comment);
    writer.writeEndElement();
  }
}
```

```
  private void writeNode(@NotNull Node node, @Nullable LayoutModel2D<Node> layoutModel2D
      ) throws XMLStreamException {
    writer.writeStartElement("node");
    writer.writeAttribute("id", node.getId());
    writeComment(node.getComment(), NODE_DESCR);
    writeNodeGraphics(node, layoutModel2D);
    writer.writeEndElement();
  }

  private void writeNodeGraphics(@NotNull Node node, @Nullable LayoutModel2D<Node>
      layoutModel2D) throws XMLStreamException {
    writer.writeStartElement("data");
    writer.writeAttribute("key", NODE_GRAPHICS);
    writer.writeStartElement(Y, "ShapeNode");

    writer.writeStartElement(Y, "NodeLabel");
    writer.writeCharacters(node.getLabel());
    writer.writeEndElement();

    writer.writeStartElement(Y, "Geometry");
    writer.writeAttribute("height", "30");
    writer.writeAttribute("width", "30");
    if (layoutModel2D != null) {
      var point = layoutModel2D.get(node);
      writer.writeAttribute("x", Double.toString(point.getX()));
      writer.writeAttribute("y", Double.toString(point.getY()));
    }
    writer.writeEndElement();

    writer.writeStartElement(Y, "Fill");
    writer.writeAttribute("color", "#FFCC00");
    writer.writeAttribute("transparent", "false");
    writer.writeEndElement();

    writer.writeStartElement(Y, "Shape");
    writer.writeAttribute("type", "ellipse");
    writer.writeEndElement();

    writer.writeEndElement();
    writer.writeEndElement();
  }

  @Override
  public void close() throws IOException {
    try {
      writer.close();
    } catch (XMLStreamException e) {
      throw new IOException(e);
    }
  }
}
```

```
}
```

1.43. ObjectWithComment.java

```
package com.github.zinoviy23.metricGraphs.api;

import org.jetbrains.annotations.Nullable;

public interface ObjectWithComment {
  @Nullable String getComment();
}
```

1.44. Identity.java

```
package com.github.zinoviy23.metricGraphs.api;

import org.jetbrains.annotations.NotNull;

public interface Identity {
  @NotNull String getId();
}
```

1.45. TapnToMgApp.java

```
package com.github.zinoviy23;

import com.github.zinoviy23.metricGraphs.io.MetricGraphJsonWriter;
import com.github.zinoviy23.metricGraphs.io.graphml.MetricGraphYedWriter;
import com.github.zinoviy23.tapaal.TapaalHeadlessService;
import com.github.zinoviy23.tapaal.extended.ExtendedTapaal;
import com.github.zinoviy23.tapnToMg.converters.ConversionException;
import com.github.zinoviy23.tapnToMg.converters.ConvertersFactory;
import com.github.zinoviy23.tapnToMg.converters.mgToTapn.
    MetricGraphToTimedArcPetriNetConverter;
import com.github.zinoviy23.tapnToMg.converters.tapnToMg.
    TimedArcPetriNetToMetricGraphConverter;
import dk.aau.cs.io.TimedArcPetriNetNetworkWriter;
import picocli.CommandLine;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.Callable;

@CommandLine.Command(
    name = "tapn-to-mg",
    description = "Converts␣Timed␣Arc␣Petri␣Net␣to␣Metric␣Graph␣and␣vice␣versa.",
    version = "tapn-to-mg␣0.0"
)
public class TapnToMgApp implements Callable<Integer> {
```

```
@SuppressWarnings("unused")
@CommandLine.Option(names = {"-f", "--from"}, description = "Source␣file")
private File sourceFile;

@SuppressWarnings("unused")
@CommandLine.Option(names = "--to-mg", description = "Target␣metric␣graph␣file")
private File mgFile;

@SuppressWarnings({"FieldMayBeFinal", "FieldCanBeLocal"})
@CommandLine.Option(names = "--format", description = "JSON,␣yEd")
private String mgFileFormat = "json";

@SuppressWarnings("unused")
@CommandLine.Option(names = "--to-tapn", description = "Target␣Timed␣Arc␣Petri␣Net")
private File tapnFile;

@SuppressWarnings("unused")
@CommandLine.Option(names = {"-g", "--gui"})
private boolean useGui;

private static Runnable resultRunnable;

public static void main(String[] args) {
  var exitCode = new CommandLine(new TapnToMgApp()).execute(args);
  if (exitCode == 0 && resultRunnable != null) {
    resultRunnable.run();
  } else {
    System.exit(exitCode);
  }
}

@Override
public Integer call() throws Exception {
  if (useGui && TapaalHeadlessService.isHeadless()) {
    System.err.println("Cannot␣use␣GUI␣in␣headless␣mode");
    return -1;
  } else if (useGui) {
    String[] args;
    if (sourceFile != null) {
      args = new String[]{sourceFile.getAbsolutePath()};
    } else {
      args = new String[0];
    }
    resultRunnable = () -> ExtendedTapaal.main(args);
    return 0;
  }

  if (mgFile != null && tapnFile != null) {
    System.err.println("Provided␣both␣mg␣and␣tapn␣destinations");
    return -1;
  }
```

```
if (sourceFile == null || !sourceFile.exists()) {
  System.err.println("Please␣provide␣existing␣source␣file");
  return -1;
}

if (mgFile != null) {
  if (!"json".equals(mgFileFormat.toLowerCase()) && !"yed".equals(mgFileFormat.
      toLowerCase())) {
    System.err.println("Unknown␣format␣" + mgFileFormat);
    return -1;
  }

  var converter = ConvertersFactory.createFileToTimedArcPetriNetConverter()
      .andThen(new TimedArcPetriNetToMetricGraphConverter());

  try {
    var graph = converter.apply(sourceFile);
    try (var fileWriter = new FileWriter(mgFile)) {
      try (var graphWriter = "yed".equals(mgFileFormat.toLowerCase())
          ? new MetricGraphYedWriter(fileWriter, true)
          : new MetricGraphJsonWriter(fileWriter, true)
      ) {
        graphWriter.write(graph);
      }
      return 0;
    } catch (IOException e) {
      if (e.getCause() != null) {
        System.err.println(e.getCause().getMessage());
      } else {
        System.err.println(e.getMessage());
      }
      return -1;
    }
  } catch (ConversionException e) {
    System.err.println(e.getCause().getMessage());
    return -1;
  }
}

if (tapnFile != null) {
  var converter = ConvertersFactory.createFileToMetricGraphConverter()
      .andThen(new MetricGraphToTimedArcPetriNetConverter());

  try {
    var network = converter.apply(sourceFile);
    TimedArcPetriNetNetworkWriter writer = new TimedArcPetriNetNetworkWriter(
        network.getNetwork(),
        List.of(network.getTemplate()),
        Collections.emptyList(),
        network.getNetwork().constants()
```

```
        );
        writer.savePNML(tapnFile);
        return 0;
      } catch (ConversionException e) {
        System.err.println(e.getCause().getMessage());
        return -1;
      } catch (IOException e) {
        System.err.println(e.getMessage());
        return -1;
      }
    }

    return 0;
  }
}
```

Лист регистрации изменений

Изм.	Номера листов				Всего листов в документе	№ документа	Входящий № сопроводит. докум. и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					