FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Exploiting Interaction in Dynamic Graph Visualization

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Riccardo Scalisi
Matrikelnummer 0626036

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao. Univ.-Prof. Mag. Dr. Silvia Miksch
Mitwirkung: Paolo Federico MSc

Wien, 03.12.2013 Riccardo Scalisi

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Exploiting Interaction in Dynamic Graph Visualization

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Riccardo Scalisi
Registration Number 0626036

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao. Univ.-Prof. Mag. Dr. Silvia Miksch
Assistance: Paolo Federico MSc

Vienna, 03.12.2013

Riccardo Scalisi
(Signature of Author)

(Signature of Advisor)

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Abstract

Dynamic networks are networks that can change over time. Their visualization plays an important role in helping users gain information and make discoveries in what is becoming more and more complex data. While approaches for static networks have been extensively researched, this happened for the unique challenges of dynamic networks to a lesser extent so far. In this thesis I provide an overview of the state-of-the-art of dynamic social networks and review some of the theoretical backgrounds involved. Further, I describe a software prototype based on the work done by Federico et al., implemented using the prefuse toolkit. The aim of the application is to present a set of features used in visualizing dynamic networks and enable others to evaluate these findings, using an existing evaluation framework. Example implementations for evaluating different kinds of user goals are shown. The prototype includes an interpolated layout with a stable mode and a consistent mode, that are useful for either preserving the mental map of the user or showing each time frame independently and with as few edge-crossings as possible. Further, users are able to highlight nodes and get context-sensitive information with each time frame for tracking groups and tracking the relationship of individual nodes.

Table of Contents

Abstract i

1. Introduction 1
1.1 Problem Description....1

2. Visualizing Dynamic Networks – State of the Art Report 2
2.1 Introduction....2
2.2 Related Work....3
2.3 Method....3
2.4 Results....4
2.4.1 Graph Drawing....4
2.4.2 Approaches to Dynamic Networks....5
2.4.3 The Mental Map....6
2.4.4 Visualizing Change....6
2.4.5 Force-directed Layout....7
2.4.6 Qualitative Evaluation....8
2.5 Discussion....9
2.6 Conclusions....9

3. Results 10
3.1 Overview....10
3.2 Visualization....16
3.2.1 Importing Data....17
3.2.2 Graph Layout....19
3.2.3 Highlighting and Color Scheme....20
3.3 Evaluation Framework....22

4. Conclusion 24

References 25

Appendix 27

1. Introduction

To gain insight into and understand social networks, visualization plays an important role. [12] In order to further facilitate such understanding I have implemented a software prototype based on the concepts and solutions Federico et al. applied to this problem. This prototype serves the purpose of being able to evaluate those solutions. In addition, an overview of the field of social networks itself is provided.

1.1 Problem Description

In their paper "A Visual Analytics Approach to Dynamic Social Networks" Federico et al. proposed and implemented a set of visualization techniques for dynamic networks as part of a framework, and integrated visual analytics methods. [11] Such networks are represented as node-link diagrams and are interactively manipulable by the user in regards to the graph layout and the highlighting of nodes and their neighbors. They have taken special care to support different kinds of data and tasks by providing multiple views and transitions between them. Thus they enable users of the framework to compare distinct time slices of the network.

This thesis focuses on certain aspects of the work done by Federico et al. with the goal of enabling a quantitative evaluation and, if possible, finding and developing further refinements. One such feature is the automatic generation of a graph layout suited for the challenges that arise through a dynamic network that changes over time. [4] Depending on the task a different approach is needed. The primary trade-off lies between the overall consistency of the graph and the stability of individual nodes. Being able to track certain participants of the social network across multiple time slices requires that they change their position very little so the mental map of the relations can be preserved. [9] On the other hand this also means that changes over time cannot optimally be reflected in the layout of the graph with regards to its desired aesthetic criteria. These include for example avoiding edge crossings and being able to identify communities within the network. In order to support the users intentions, the layout must therefor be configurable to be either as stable or as consistent as possible.

Since comparing time slices by looking at separate graphs would be cumbersome, the prototype positions multiple network graphs next to each other, thus providing a juxtaposition view as proposed by Federico et al. Another feature addressed by the prototype is the highlighting of nodes and their neighbors in said view. For example if the user is interested in a particular set of participants of the social network that are all connected, he or she might want to observe how their relationships change over time. To that end the same nodes need to be highlighted in each time slice. If the user only wants to track a specific node, that one node and its neighbors in each respective time slice should be highlighted.

In chapter 2 I detail past and current findings on the topic of dynamic social network in a state-of-the-art report. In chapter 3 I will expand on the software prototype, implementation details, and the integration of the evaluation framework. Finally, I provide my conclusions and address possible ways of expanding on this work in chapter 4.

2. Visualizing Dynamic Networks – State of the Art Report

Dynamic networks are networks that can change over time. Their visualization plays an important role in helping users gain information and make discoveries in what is becoming more and more complex data. While approaches for static networks have been extensively researched, this happened for the unique challenges of dynamic networks to a lesser extent so far. This chapter provides an overview of some of these challenges and approaches and discusses them in the context of social networks in particular.

2.1 Introduction

Many forms of data are best represented as networks, be they human relationships, business organizations or telecommunications networks. For such structures the complexity and amount of detail can grow rapidly with each additional participant and each relationship. Therefor in order to enable users to perform basic perceptual tasks in an efficient manner, so they can gain insight into a network, suitable representation are needed. Visualizations with the help of graphs can solve this problem.

So far the majority of the proposed solutions have been centered around static graphs [13]. The introduction of time as a variable poses an additional challenge, though. Dynamic networks consist of multiple time slices, meaning that nodes, edges and attributes can change over time. The visualizations used for networks with such characteristics not only have to meet certain aesthetic criteria for each individual slice, they also have to highlight change while preserving the user's mental map of the graph layout in between slices – a task that can be contradictory at times. While for example the removal of a node and it's incoming and outgoing connections during time n can require a substantial change of the visualization's layout for time n + 1, such a requirement is often directly opposed to the need of stability. Suitable trade-offs need to be found.

In this chapter I aim to give an overview of the state of the art of visualizing dynamic networks. First I present related work, that has already been done in this field of study, in section 2. In section 3 I outline how the results from section 4 have been gathered. The results span approaches from different areas including graph drawing, the challenges of dynamic networks described above, visualization of change and algorithms for graphs. Finally I discuss these findings in section 5 and provide some possible conclusions in section 6.

2.2 Related Work

For social networks in particular, Freeman [12] gives an in-depth historical overview of the evolution of network illustrations from hand-drawn images, to the use of computation, to the inclusion of the web.

Correa [8] provides information about different ways of representing a social network, including structural visualizations like node-link diagrams and matrices, as well as higher level representations, that focus on the semantics instead of the details. Further he details methods to analyze networks in an interactive way.

2.3 Method

There were multiple factors contributing to the results presented in the next section. To begin with I started out with a few suggestions given to me by Paolo Federico. These publications were a useful source for getting started and cover topics like the user's mental map and animations in graphs. From there I looked at the works that were referenced in the papers I had already read, paying special attention to those that were cited in multiple sources.

To branch out from that body of work, I used a number of keywords, determined by my own prior knowledge of the topic and general searches and information gathered from not peer-reviewed sources, like blogs and Wikipedia. I arrived at a handful of keywords, including among others:

graph drawing, static aesthetics, node-link diagram, adjacency matrix, dynamic networks, mental map, animation & small multiples, visual analytics, social network analysis, interaction techniques, graph drawing algorithms, force-directed algorithms

I used these keywords mostly in conjunction with each other and applied them to search the digital libraries with the most sources available to me. These are the IEEE Xplore Digital Library[1], the ACM Digital Library[2], SpringerLink[3] and ScienceDirect[4], where the last one mostly had the most results. Following that step I compared the results to gain an overview of the different areas, that are covered in this report and compiled the most important approaches in the results section. Again, at this stage, I made sure to follow citations of similar literature to gain further insight into a specific topic.

1 www.ieeexplore.ieee.org/Xplore/home.jsp
2 www.dl.acm.org
3 www.link.springer.com
4 www.sciencedirect.com

2.4 Results

In order to produce effective visualizations, that illustrate social networks, multiple areas of problems need to be solved.

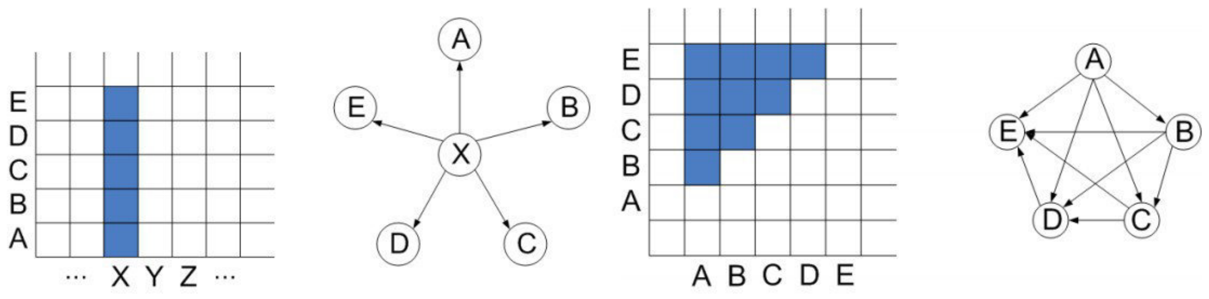
2.4.1 Graph Drawing

Going back to the first sociogram in 1934 drawing graphs has been the most widely used method of representing social networks so far [12]. Illustrating actors as nodes and relationships as edges provides an intuitive overview and lets users easily trace paths between nodes. Edges can be directed or undirected and there can be a number of attributes associated with each element, including relationship status (qualitative or quantitative), group affiliations, etc.

As node-link graphs become more complex, finding a suitable layout becomes more and more difficult, though. In order to ensure that graphs do not become cluttered and stay readable, Purchase [19] has identified seven metrics, that all contribute to the overall aesthetic quality of a graph. These criteria are:

- minimizing edge crossings,
- minimizing edge bends,
- maximizing symmetry,
- maximizing the minimum angle between edges leaving a node,
- maximizing edge orthogonality,
- maximizing node orthogonality,
- maximizing consistent flow direction (directed graphs only).

Alternatively the use of adjacency matrices is possible as well [8]. In a matrix both columns and rows list the actors while the respective entries are either binary (connected or not connected) or a value that characterizes the relation between the actors. This form of representation is less common, because of certain inherent disadvantages. The network's structures and groups are not as easily recognizable. Paths between actors have to manually be traced from cell to cell. Further, matrices do not scale as easily as point and line graphs. Each additional actor requires an added row and column, meaning that the boundaries of the illustration steadily grow as well.

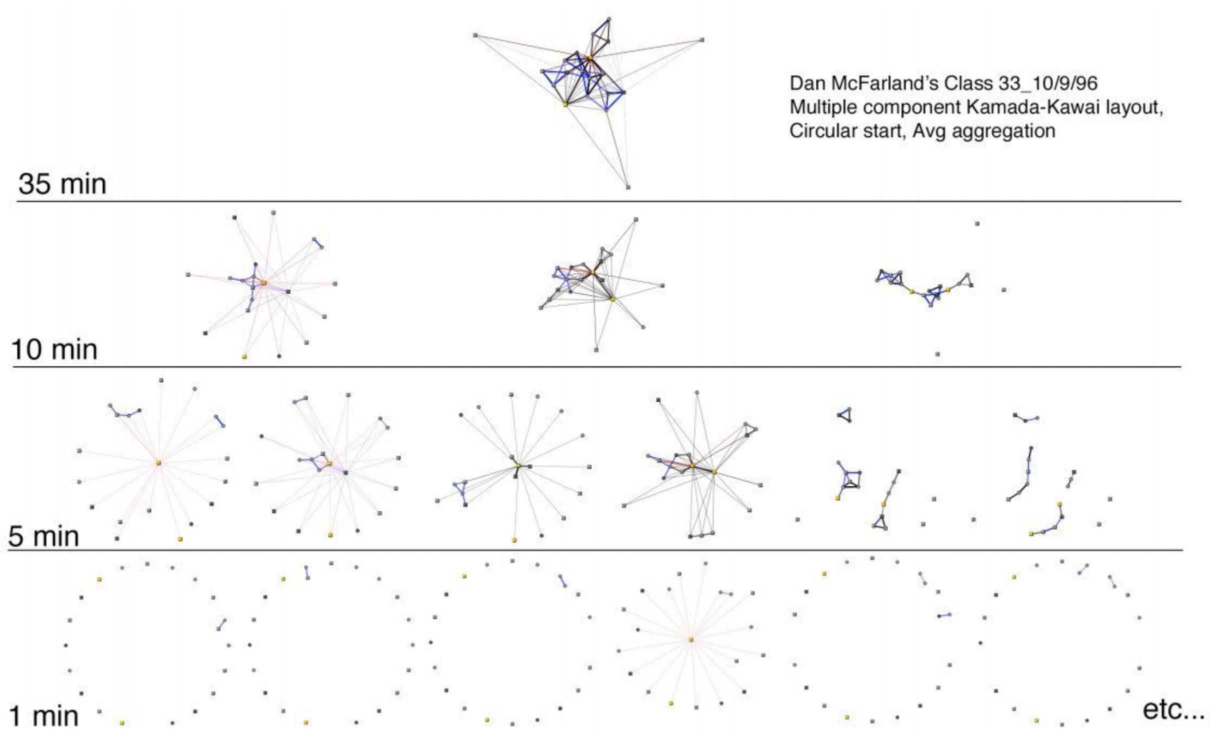


(figure 1, illustrating adjacency matrices on the left and the corresponding node-link diagram on the right. source: [21])

2.4.2 Approaches to Dynamic Networks

As stated in the introduction, dynamic networks pose additional challenges compared to static networks. When introducing change over time into a graph, users need not only to be able to track an actor, a relationship or a path through the network, but also through the course of its respective development.

Before being able to visualize the data, the means of collecting and organizing it have to be determined, though. Continuous processes in the real world, that have an effect on social networks, need to be sampled in discrete points in time, in order to be processable by computers [4]. This mostly happens in waves. Of course the interval between these waves plays an important role in interpreting the data. When using a high temporal resolution otherwise minor events can cause the resulting graph's layout to overly fluctuate between time slices and obscure the bigger picture. For example in a human relationship there might be events measured in daily intervals, that have a strong influence on the feelings of person A towards person B, like a quarrel or a shared emotional experience. These change the data for the specific day they were registered, but have only an average effect when otherwise observed in the context of a whole week. Therefore, depending on the specific time interval the user is interested in, data has to be filtered and reorganized accordingly.

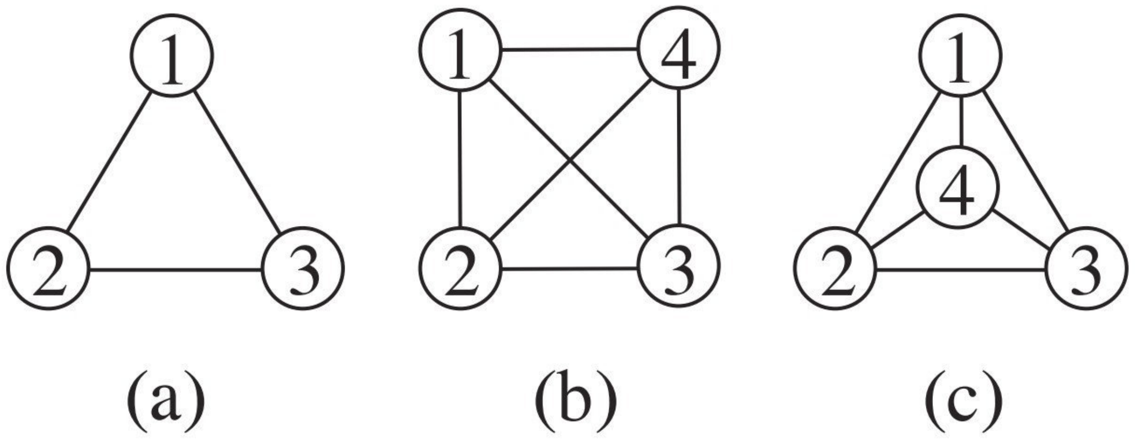

Dan McFarland's Class 33_10/9/96
Multiple component Kamada-Kawai layout,
Circular start, Avg aggregation
35 min
10 min
5 min
1 min
etc...

(figure 2, illustrating the same group interactions collected at different intervals, source: [4])

2.4.3 The Mental Map

The “mental map” of a user has been defined as the cognitive model that a user applies to a graph's layout. Since the layout of a dynamic network changes over time, this model has to adapt as well. How effectively the mental map is being preserved between time slices can for example be measured as the movement of nodes as the layout changes [20]. The aim of this approach is to better support perceptual tasks like tracking actors, groups and paths, and being able to recognize change in a local area by keeping unaffected areas otherwise stable.

Limitations to this approach have been identified, however. When requiring stability of node positions over time, a high number of subtractions and additions of nodes can lead to a layout, that no longer meets certain aesthetic criteria. In other words, stability and consistency are at odds with each other when dynamic networks with a high degree of change over time is concerned [20].

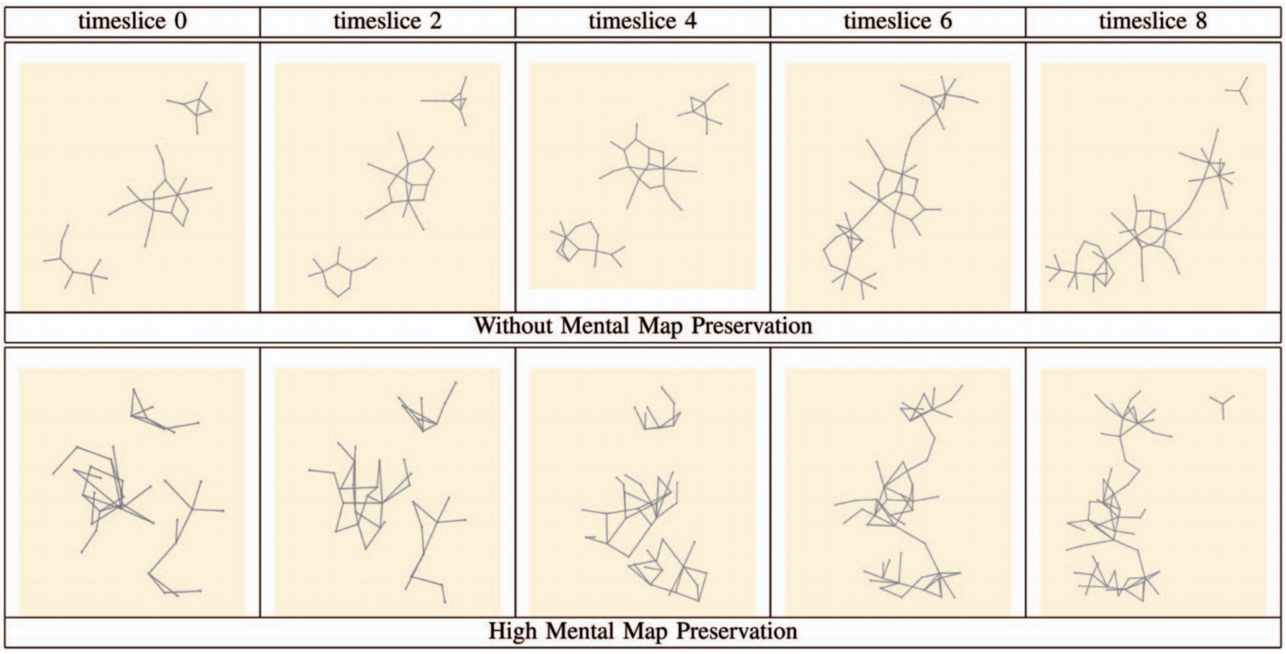


(figure 3, illustrating the change in a graph (a) with no mental map preservation (b) and mental map preservation (c), source: [17])

2.4.4 Visualizing Change

There are multiple approaches of visualizing change in a graph. Since the state of a network is divided into separate slices, that each correlate to a point in time, each slice can be viewed as a single image. Like in other media, images with a temporal connection can be strung together different ways. For example when interpolating between slices, the result is animation. An animated graph can be useful to track actors or groups across their evolution in the network. Studies have shown, that users can track up to five independently moving targets [3]. Therefor the limitations of animation are clear. When the number of moving nodes exceeds the abilities of the users to usefully track them simultaneously, alternative approaches might be more efficient.

Small multiples are a series of images next to each other in chronological order [24]. They all depict the same graph during different points in time and enable the user to visually compare their state. Surveys performed by Purchase et al. [2] have shown, that participants can answer questions about graphs faster when using small multiples than when using animations. Animations on the other hand produce less errors.

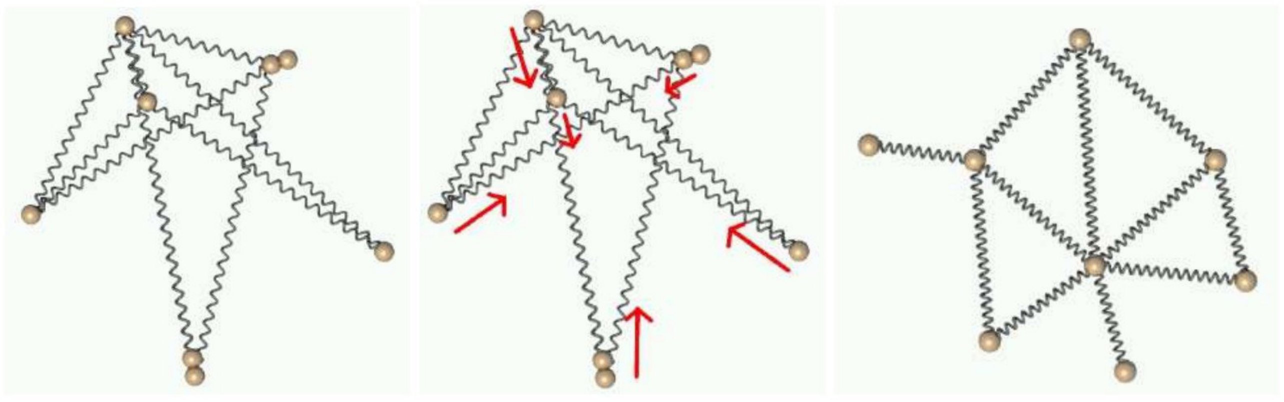


(figure 4, illustrating small multiples of time slices, source: [2]

2.4.5 Force-directed Layout

Force-directed graph drawing algorithms or spring embedders are algorithms that use a physical metaphor to produce graph layouts with high symmetry and edges of similar length to each other. Edges work like physical springs, meaning they cause connected nodes to attract each other and prevent them from moving too far apart. Unconnected nodes meanwhile repel each other and cause the graph to find an equilibrium of a minimal energy state [16].

Force-directed algorithms work in iterations and often end up in a local minimum instead of the global minimum. Approaches have been developed, that work in layers and therefor are more effective for higher numbers of nodes. Local structures are calculated first and then used as elements for larger structures. This way a more even layout can be found [16].



(figure 5, illustrating the physical metaphor of a force-directed layout algorithm, source: [16])

In particular I will highlight three different approaches to producing dynamic layouts with multiple related graphs as input. [5] The aim for these is to obtain a sequence of layouts, that retains stability as well as readability. Approaches can be differentiated by whether they are usable offline, meaning that all the information is available at the beginning of the algorithm, or online, with data being introduced one step at a time, resulting in an iterative processing of input.

- **Aggregation**: The input graphs are used to compute an aggregate graph, that serves as a combination of all available node positions and relations. Recurring connections are thereby emphasized more strongly than those found only infrequently. Finally, overall stability is maximized by using these fixed positions for all layouts.
- **Anchoring**: Nodes for the layout are “anchored” to a fixed position, meaning that they can only deviate from this position by a certain amount. This results in a layout that shows a higher resistance to local changes from one graph to the next. The anchors can either be obtained through an aggregated layout (see above), requiring an offline processing of data with all graphs available, or taken from the node positions of the previous graph's layout. Further, initial node positions do play a role as well. They are obtained either again from the preceding graph, or computed independently with multidimensional scaling.
- **Linking**: All available graphs are used simultaneously, making Linking an offline scenario. Nodes are connected to their predecessors as well as successors and influence each other, increasing stability not only in relation to the previous graph, but also to the following one. The length of the links can be adapted to influence the overall amount of deviation between graphs and the effect of local changes.

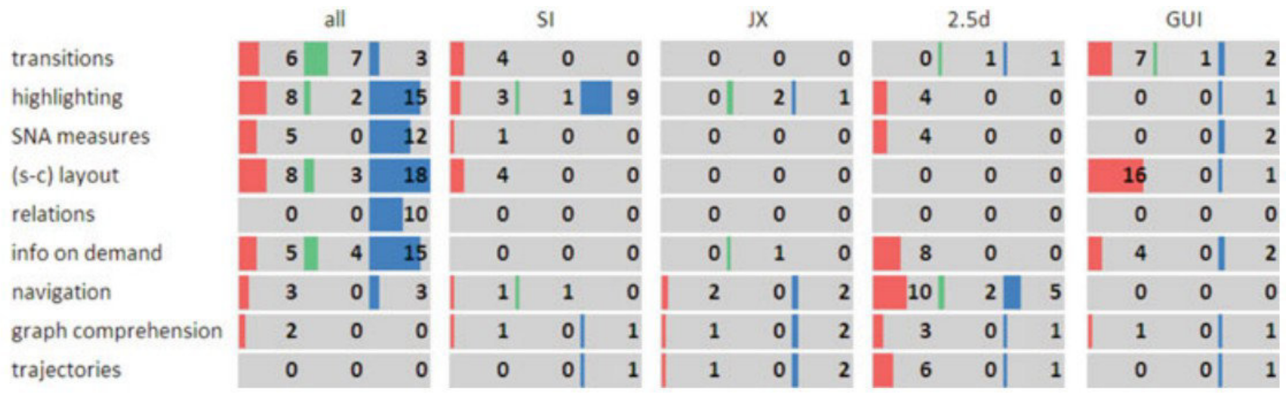
Brandes and Mader conclude, that of the three approaches detailed above, linking compares the most favorable. [5] Also worthy of note, is that Federico et al. introduced a metric called “change centrality”, that expresses the change a node has experienced between two points in time. [10] Using this metric, the linking and the anchoring approach can be enhanced to better reflect the evolution of the network by dynamically altering spring lengths and coefficients. The change centrality metric is computed by taking the ratios of added, removed and remaining links into account. The influences of neighboring nodes reachable in n steps are combined, whereby nodes farther away have less of an impact. This is especially useful for a direct superimposition of layout slices, where change is expressed as node displacement. Apart from layout techniques, change centrality can also be encoded visually through node size and color.

2.4.6 Qualitative Evaluation

For their original prototype, Federico et al. conducted a mock-up study during development, using non-interactive sketches. [22] These showed that users were able to understand dynamic networks with the use of their graph visualizations. While the juxtaposition view turned out to be the most comprehensible, the comet plot did often require additional explanation.

The finished prototype was then evaluated through a qualitative user study with nine expert-level participants. [22] They were asked to explore the visualization and give feedback. In a second phase they solved a set of tasks, varying in openness. Overall the participants solved 89% of the tasks correctly, while most of the incorrect answers, as well as the least confident answers were given for task 1, which was the most open and least prescriptive.

Figure 6 shows an overview of the gathered feedback. Overall all users rated the prototype remarkably positive. Again, the juxtaposition view tested the to be the most comprehensive among participants. The Superimposition view was observed to have problems with too slow transitions and visual clutter for some. For the 2.5D view participant reported issues, because of visual distortion and missing responsiveness for interaction methods like panning, zooming and rotating. The GUI overall was partially problematic, because of the choice of certain labels like “Superimposition” and “Juxtaposition”. While experts might be familiar with these terms, it seems that they are not immediately self-evident for the layman. This is a factor that needs to be considered in regards to usability.



	all			SI			JX			2.5d			GUI		
transitions	6	7	3	4	0	0	0	0	0	0	1	1	7	1	2
highlighting	8	2	15	3	1	9	0	2	1	4	0	0	0	0	1
SNA measures	5	0	12	1	0	0	0	0	0	4	0	0	0	0	2
(s-c) layout	8	3	18	4	0	0	0	0	0	0	0	0	16	0	1
relations	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0
info on demand	5	4	15	0	0	0	0	1	0	8	0	0	4	0	2
navigation	3	0	3	1	1	0	2	0	2	10	2	5	0	0	0
graph comprehension	2	0	0	1	0	1	1	0	2	3	0	1	1	0	1
trajectories	0	0	0	0	0	1	1	0	2	6	0	1	0	0	1

(figure 6, user feedback for problems (red), positive observations(green) and ideas for improvement (blue) for the different views of the prototype, source: [22])

2.5 Discussion

Some of the approaches can be combined, while others can not. When determining the basic representation of the network either a matrix-representation or a node-link diagram can be used. The later has turned out to be a lot more common [8]. When considering the challenges of change over time, preserving the user's mental map has proven to be useful only for networks with certain characteristics. Otherwise the trade-off between consistency and stability might prove to be undesirable and violate aesthetic criteria. Approaches for communicating change to the user are for example animation and small multiples, both having advantages and disadvantages, the former producing results with less error and the later enabling users to answer questions faster [2].

2.6 Conclusions

The results presented in the previous section are only part of the approaches, that are needed to visualize dynamic social networks. Others are actively being developed and researched. This includes the field of Visual Analytics, that combines approaches from multiple disciplines relevant to dynamic graphs like Information Analytics, Interaction and Knowledge Discovery [15].

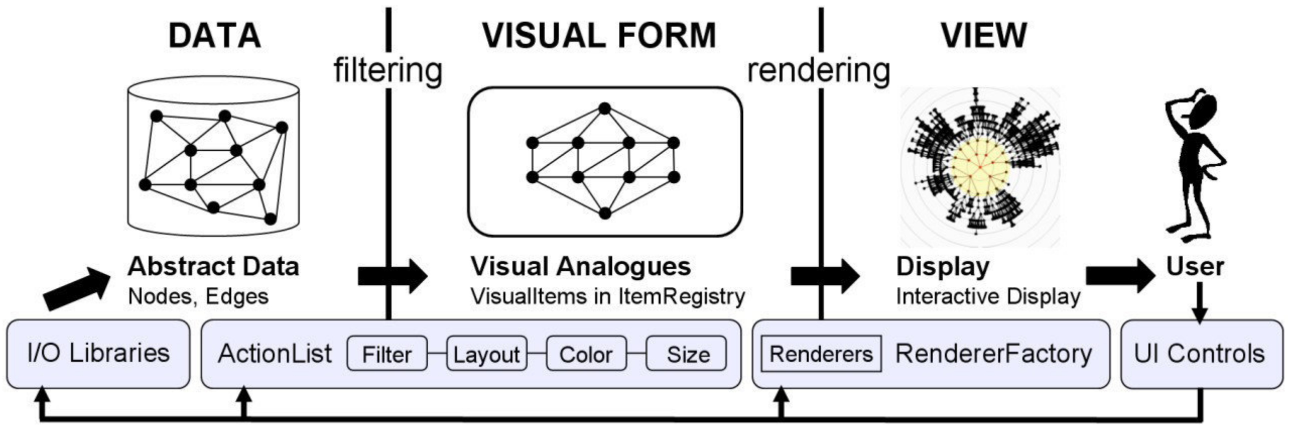
3. Results

In this chapter I will describe the software prototype that was the result of the investigation defined in the problem description above.

3.1 Overview

The prototype is written in the Java programming language and uses the prefuse toolkit[5] to help display the visualization. prefuse has proven itself to be a capable framework for multiple types of interactive visualizations. [18] It is especially useful for implementing node-link diagrams with a force-directed layout, because of already existing components, that are designed to aid in this task. Further, prefuse is able to output the same dataset to multiple displays with different filtering criteria.

During the development of the visualization I followed the workflow methodology, that prefuse is based on: a typical Information Visualization pipeline. [14] The first stage of the pipeline is data acquisition, which is generally accomplished by importing source data into a form readable by the framework, i.e. tables. Often the data then needs to be transformed or simplified in some way. This includes the detection and removal of faulty values or the consolidation of complex data into a more convenient structure. Once these internal representations are ready to be used, the data is mapped to a visual form. The appearance of a visual item depends on the visualization, that it is used for, and can be predefined with a list of properties, including position, color, size, labels, etc., as well as whether the item is currently part of the visualization (visible) or to be used later (invisible). After determining the visual qualities of an item, it is rendered to the appropriate display by a render factory. The display is part of the user interface and can be interactively manipulated by the user. In order to react to input from the user, the prefuse framework is able to loop the pipeline from any point, be it continuously or on demand only, and redraw the data accordingly.



(figure 7, the InfoVis pipeline implemented by prefuse: customizable action lists filter the raw abstract data into a visual form. These visual items are then rendered and presented to the user. Finally, the visualization can be interactively manipulated. source: [14])

5 www.prefuse.org

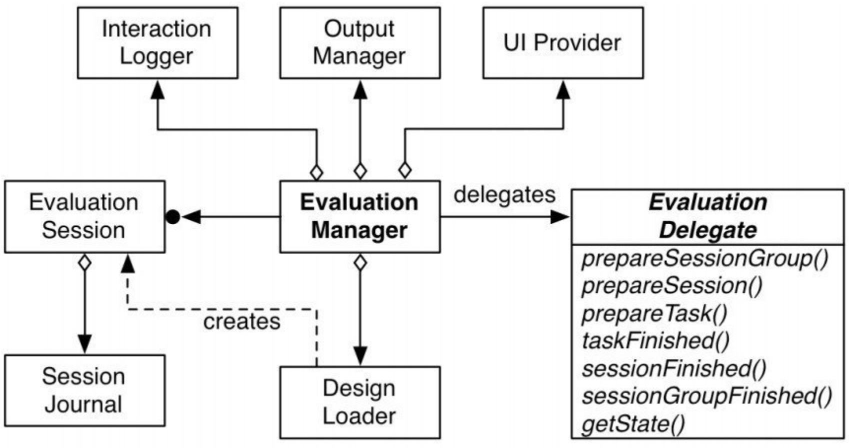
Although a community of users and contributors exists for the framework, it is currently not in active development anymore. One downside of this fact is that the official documentation has partially stayed incomplete, which has resulted in a somewhat steeper learning curve for me. It also led to additional code refactoring in the beginning of the project, that could have otherwise been avoided.

Integrating the prefuse library itself into the prototype was relatively easy, though, since it has been made available as open source software. I made sure to leave the library unaltered, which means that the prototype can be distributed without the library included and is theoretically compatible with future prefuse releases. As an additional benefit, future work and testing are limited to a smaller amount of classes.

Since the prototype is written in Java, I used Swing for the graphical user interface part of the visualization. The Swing code is contained within the main visualization class (BAVisualization.java) and serves as the presentational layer of the application.

Further, I integrated the EvalBench[6] library into the prototype, so the software can be used to evaluate its current feature set and possible future additions with ease.

EvalBench was created to support researchers in implementing their own methods of evaluation more easily, be they qualitative or quantitative. On the one hand solutions for gathering task completion times and error rates are provided, as well as for managing user study transcripts and feedback. The software is open source and designed to be used with as many and diverse kinds of visualizations as possible. This requires EvalBench to be highly customizable, since most evaluation efforts connected to visual artifacts have unique requirements, specific to their subject matter. Integrating the library is accomplished via loose coupling, through the implementation of the EvaluationDelegate class. This class provides the hooks to control the evaluation session and present an application-specific interface to the user.

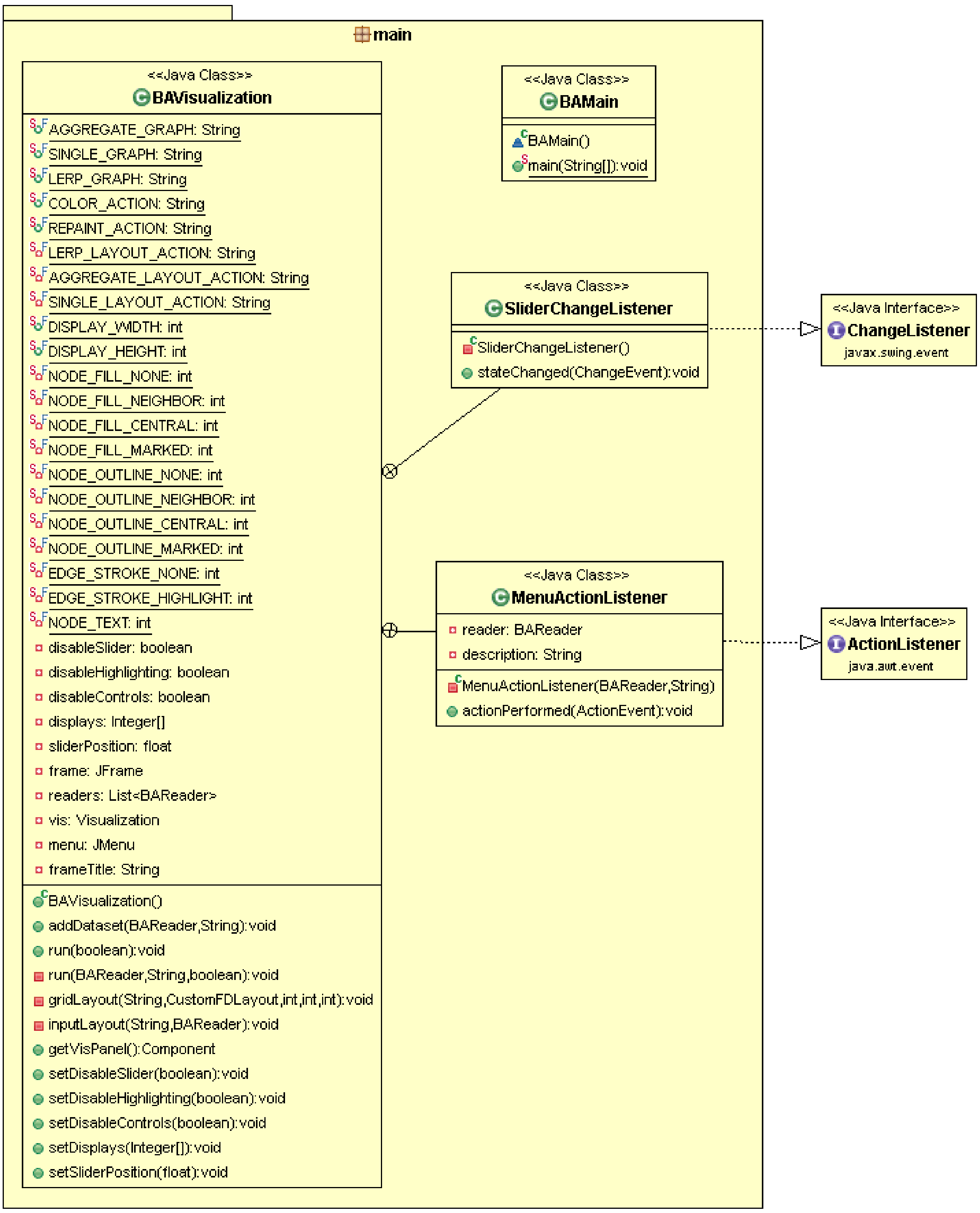


(figure 8, the EvaluationDelegate class is used to integrate EvalBench into the prototype)

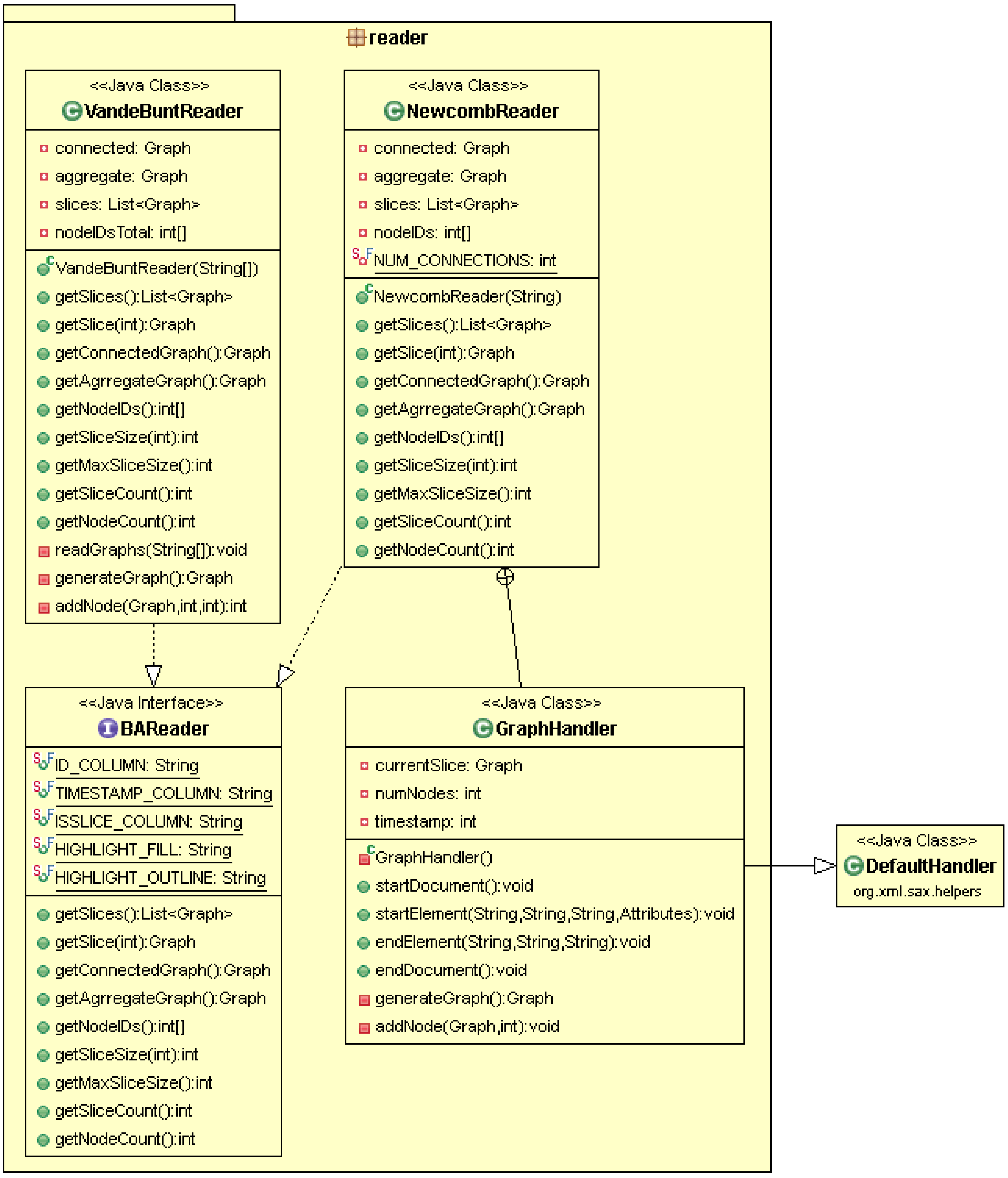
A few sample tasks are provided in the Appendix and source code to demonstrate the customization capabilities and range of diverse potential evaluation tasks.

In the coming sections of this chapter I will in more detail explain the prototype's functionality, that can be thought of as divided into five packages. (main, reader, layout, controls and evalBench)

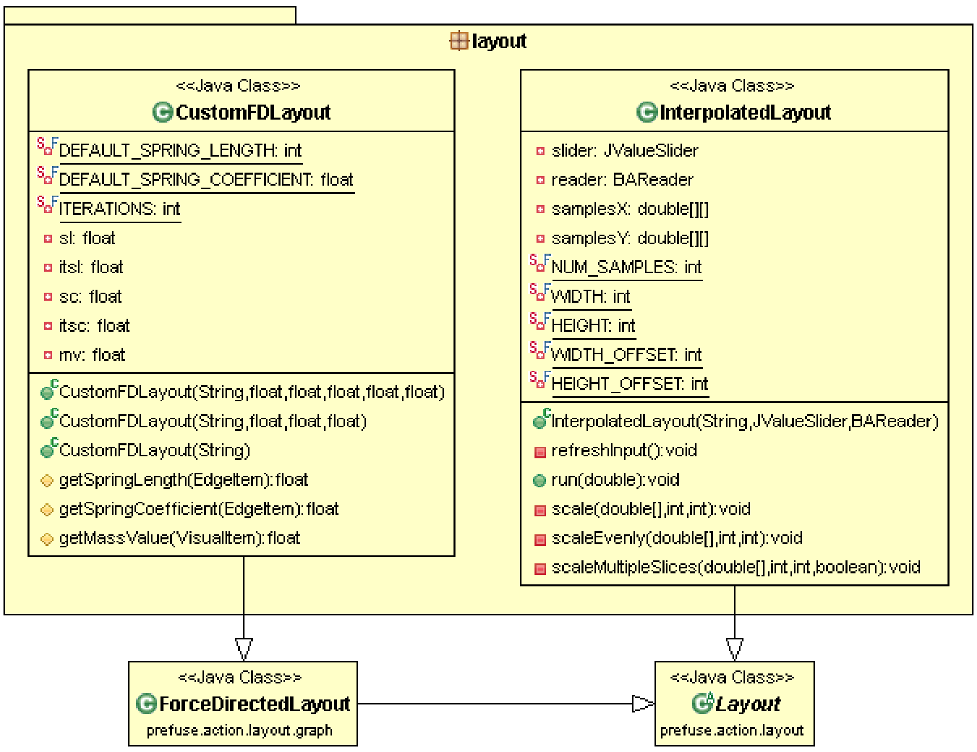
6 www.evalbench.org



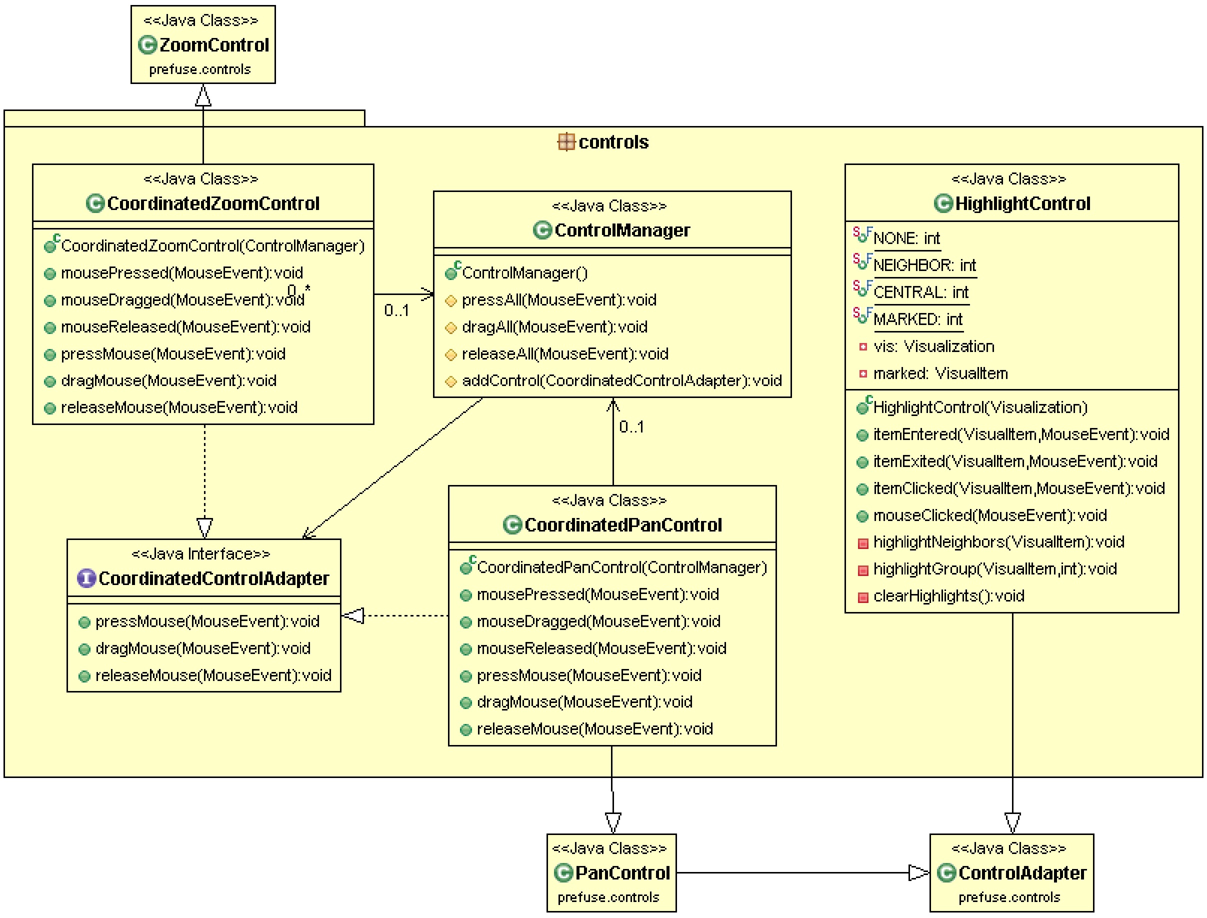
*(figure 9, UML diagram of the **main** package)*



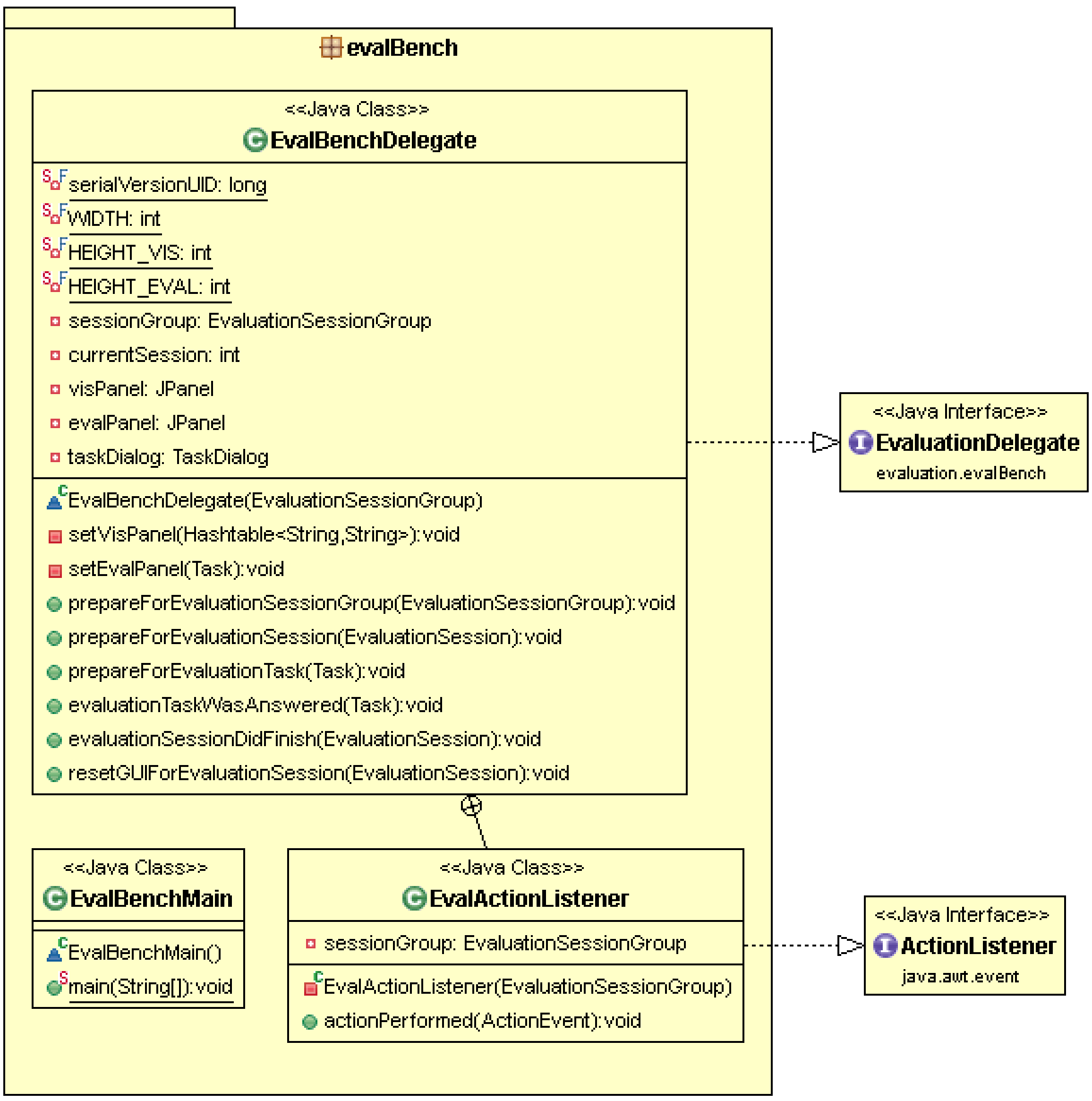
*(figure 10, UML diagram of the **reader** package)*


layout
<<Java Class>>
CustomFDLayout
DEFAULT_SPRING_LENGTH: int
DEFAULT_SPRING_COEFFICIENT: float
ITERATIONS: int
sl: float
itsl: float
sc: float
itsc: float
mv: float
CustomFDLayout(String,float,float,float,float,float)
CustomFDLayout(String,float,float,float)
CustomFDLayout(String)
getSpringLength(EdgeItem):float
getSpringCoefficient(EdgeItem):float
getMassValue(VisualItem):float
<<Java Class>>
InterpolatedLayout
slider: JValueSlider
reader: BAReader
samplesX: double[][]
samplesY: double[][]
NUM_SAMPLES: int
WIDTH: int
HEIGHT: int
WIDTH_OFFSET: int
HEIGHT_OFFSET: int
InterpolatedLayout(String,JValueSlider,BAReader)
refreshInput():void
run(double):void
scale(double[],int,int):void
scaleEvenly(double[],int,int):void
scaleMultipleSlices(double[],int,int,boolean):void
<<Java Class>>
ForceDirectedLayout
prefuse.action.layout.graph
<<Java Class>>
Layout
prefuse.action.layout

(figure 11, UML diagram of the **layout** package)


<<Java Class>>
ZoomControl
prefuse.controls
controls
<<Java Class>>
CoordinatedZoomControl
CoordinatedZoomControl(ControlManager)
mousePressed(MouseEvent):void
mouseDragged(MouseEvent):void
mouseReleased(MouseEvent):void
pressMouse(MouseEvent):void
dragMouse(MouseEvent):void
releaseMouse(MouseEvent):void
0..*
0..1
<<Java Class>>
ControlManager
ControlManager()
pressAll(MouseEvent):void
dragAll(MouseEvent):void
releaseAll(MouseEvent):void
addControl(CoordinatedControlAdapter):void
0..1
<<Java Class>>
HighlightControl
NONE: int
NEIGHBOR: int
CENTRAL: int
MARKED: int
vis: Visualization
marked: VisualItem
HighlightControl(Visualization)
itemEntered(VisualItem,MouseEvent):void
itemExited(VisualItem,MouseEvent):void
itemClicked(VisualItem,MouseEvent):void
mouseClicked(MouseEvent):void
highlightNeighbors(VisualItem):void
highlightGroup(VisualItem,int):void
clearHighlights():void
<<Java Interface>>
CoordinatedControlAdapter
pressMouse(MouseEvent):void
dragMouse(MouseEvent):void
releaseMouse(MouseEvent):void
<<Java Class>>
CoordinatedPanControl
CoordinatedPanControl(ControlManager)
mousePressed(MouseEvent):void
mouseDragged(MouseEvent):void
mouseReleased(MouseEvent):void
pressMouse(MouseEvent):void
dragMouse(MouseEvent):void
releaseMouse(MouseEvent):void
<<Java Class>>
PanControl
prefuse.controls
<<Java Class>>
ControlAdapter
prefuse.controls

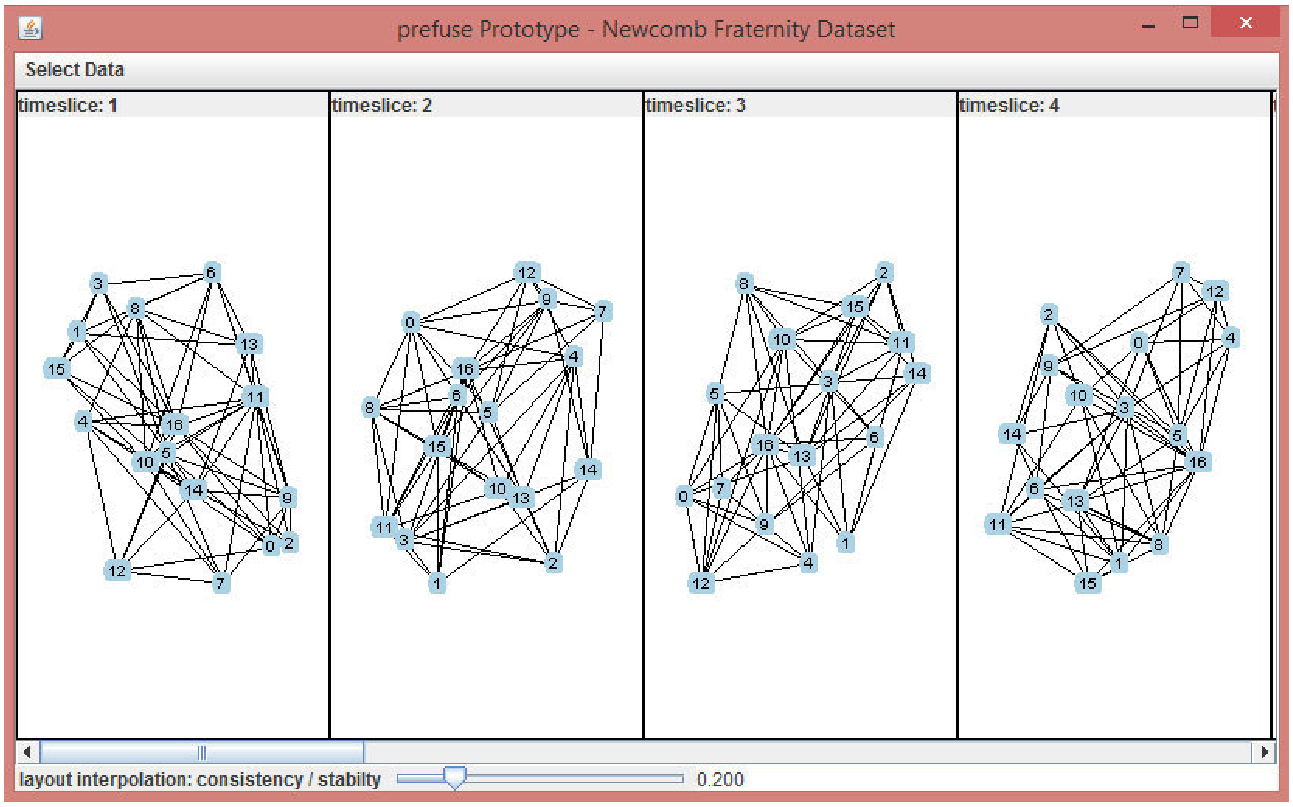
(figure 12, UML diagram of the **controls** package)


evalBench
<<Java Class>>
EvalBenchDelegate
serialVersionUID: long
WIDTH: int
HEIGHT_VIS: int
HEIGHT_EVAL: int
sessionGroup: EvaluationSessionGroup
currentSession: int
visPanel: JPanel
evalPanel: JPanel
taskDialog: TaskDialog
EvalBenchDelegate(EvaluationSessionGroup)
setVisPanel(Hashtable<String,String>):void
setEvalPanel(Task):void
prepareForEvaluationSessionGroup(EvaluationSessionGroup):void
prepareForEvaluationSession(EvaluationSession):void
prepareForEvaluationTask(Task):void
evaluationTaskWasAnswered(Task):void
evaluationSessionDidFinish(EvaluationSession):void
resetGUIForEvaluationSession(EvaluationSession):void
<<Java Interface>>
EvaluationDelegate
evaluation.evalBench
<<Java Class>>
EvalBenchMain
EvalBenchMain()
main(String[]):void
<<Java Class>>
EvalActionListener
sessionGroup: EvaluationSessionGroup
EvalActionListener(EvaluationSessionGroup)
actionPerformed(ActionEvent):void
<<Java Interface>>
ActionListener
java.awt.event

*(figure 13, UML diagram of the **evalBench** package)*

3.2 Visualization

The aim of the visualization is to give the user an overview of a dynamic social network and let him or her interactively explore nodes and relationships with the desired level of detail. The following screenshot shows the main window of the application.


prefuse Prototype - Newcomb Fraternity Dataset
Select Data
timeslice: 1
timeslice: 2
timeslice: 3
timeslice: 4
layout interpolation: consistency / stabilty
0.200

(figure 14, multiple time slices of a dataset are arranged next to each other. The layout-slider is set to favor consistency and therefor avoid edge crossings.)

Timeslices are represented as displays and positioned next to each other, accessible through a scrollbar. Nodes are labeled with an id-number, unique per timeslice. Edges are always bidirectional and unweighted.

3.2.1 Importing Data

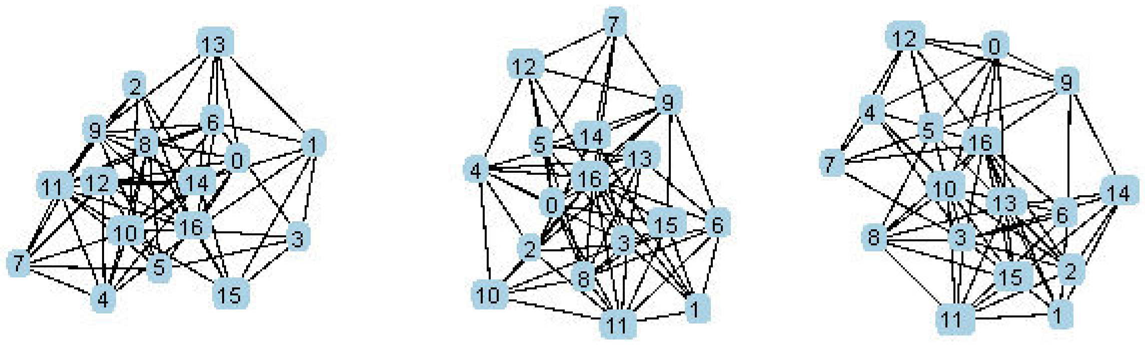
The prototype supports different kinds of data including XML and comma-separated values (CSV). Currently two datasets are included: the Newcomb Fraternity dataset [6] (social rankings between 17 participants from the University of Michigan, recorded in 1956) and the van de Bunt dataset [23] (recorded by Gerhard van de Bunt among students in 1999). Since there is no guarantee for any dataset to follow a specific structure (syntactically or semantically), a certain amount of customization in the importing process is necessary for both of them. That will also be true for further datasets added to the prototype, should the need arise.

The Newcomb dataset consists of an XML file and is read via SAXParser. Since the visualization is modeled without weighted edges only a subset of the social rankings is used. Those are a person's top 5 preferences, so a change in student relationships over time can be displayed. Each node will therefor have at least 5 connections.

The van de Bunt dataset is comprised of multiple text files (one for each timeslice of the network), each containing an adjacency matrix with qualitative values. The data is read with the help of a prefuse function and then interpreted through simple String operations. As opposed to the Newcomb dataset, nodes can be connected to any number of other nodes in the network (including none). Participants with no ratings (and therefor no relationship in the network) were omitted. This resulted in a somewhat more complex process of fitting the data into a format acceptable to the prefuse toolkit.

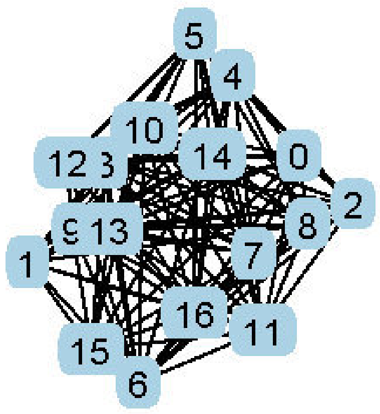
In order to efficiently display the dataset all necessary calculations for importing the data happen at the startup of the application. This results in multiple instances of prefuse's *Graph* class:

1. **Single slice graphs** represent exactly one timeslice of the network. They are used to calculate the "consistency"-side of the layout slider, since here the aim is to give the user an internally consistent view for each point in time. There is one independent graph for each slice.



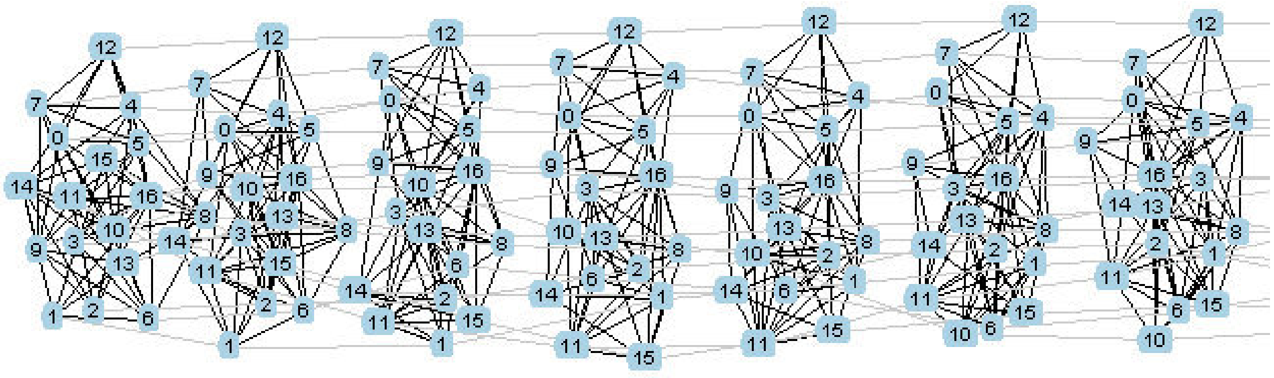
(figure 15, a single slice represent exactly one point in time. Each node is only connected to other nodes in the the same timeslice.)

2. The **aggregate graph** contains each node of the network exactly once and includes all connections from all timeslices. Two nodes can therefor be connected to each other with multiple edges. The graph is used for the “stability”-end of the layout slider and will position nodes with a relationship over a longer period of time closer to each other.



(figure 16, in the aggregate graph all edges of the network are present at once)

3. The **linked graph** is used for convenience and stores all the nodes and edges in the way they are finally displayed to the user within a single graph instance. There exist edges between the timeslices, so each node is connected to its predecessor and successor in the other timeslices. Those edges are not visible within the visualization. They are however used to traverse the graph for purposes of highlighting. (see **Highlighting** section)



(figure 17, the linked graph is closest to the final result of the visualization. Lighter edges are not visible to the user and are only included for clarity.)

Potential additions to the available datasets will have to at least provide these three kinds of graphs in order to be used by the prototype. This is accomplished by implementing the BAReader.

3.2.2 Graph Layout

As mentioned in the problem description in chapter 1, one of the properties the software prototype aims to investigate is the ability to interactively change the layout of the network graph. From the beginning a few important criteria were set:

- The consistent layout would display timeslices independently from each other and in an aesthetically consistent configuration. This way information can be displayed more efficiently without regard to the progression from one timeslice to the next.
- The stable layout would change node positions between timeslices as little as possible and thus prevent the user from having to reorient themselves while viewing different parts of the visualization.
- Both modes (consistency and stability) need to be computed at the launch of the application. The final layout displayed to the user is then calculated as a linear interpolation with the slider position taken into account.
- Every setting of the slider should in itself be useful and result in a viable layout, be it closer to one of the extremes or right in the middle. The interpolation between those states needs to be as seamless as possible.

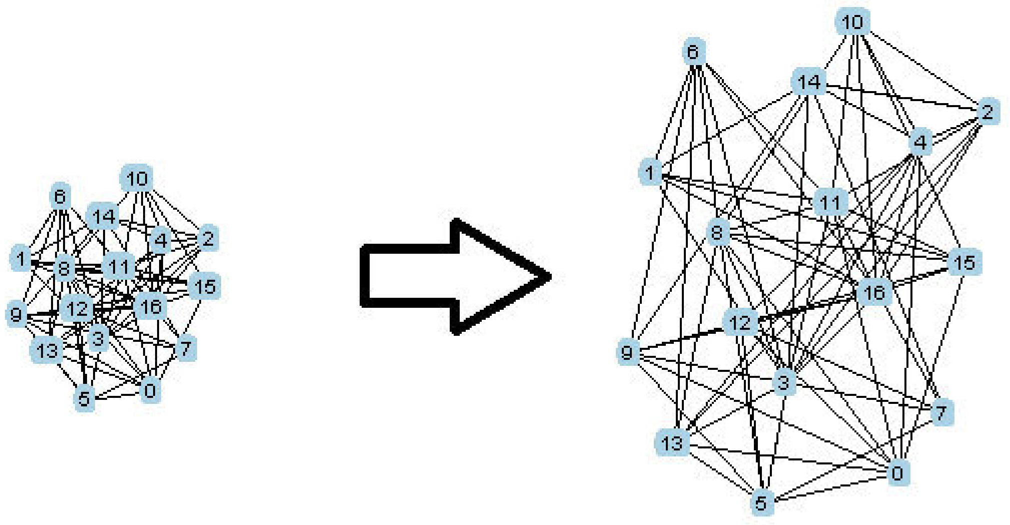
Since the node positions of the stable layout do not change from timeslice to timeslice, computing this part of the layout is straightforward. The aggregate graph (see section **Importing Data** above) is used as the input for a force-directed layout algorithm provided by prefuse. This algorithm positions the nodes according to a physical model of forces and springs, resulting in a layout with as few edge-crossings as possible. (see **State of the Art** chapter) It therefor fulfills our requirement of an aesthetically consistent layout. I chose an evenly-spaced grid as starting position for this algorithm to increase the chances of finding an optimal configuration earlier and to stop the prefuse implementation from acting non-deterministically.

For the consistent layout it turned out that individually computing the timeslices in the same way as the stable layout did not yield results good enough to meet our proposed criteria. Although each timeslice needs to be independent from the rest, it is still necessary to avoid small local changes from resulting in widely different outcomes by comparison. This would manifest in the force-directed layout rotating the graph as a whole or moving groups of nodes between graphs where only minor differences could otherwise detected. I solved this problem by again finding a common starting position for the nodes before applying the prefuse algorithm. Instead of a grid I used the stable layout as input, since at this point it is already calculated and represents a viable configuration for the force-directed layout to end up in. Individual changes for each timeslice are then applied by the physical model, though keeping unwanted spinning, etc. to a minimum.

With both the consistent and the stable mode in place, the only part missing is the interpolation according to the UI slider. The exact implementation of this step, however, went through some iterations before I settled on its eventual approach. Simply applying the formula

$$c = a \cdot t + b \cdot (1 - t) \mid 0 \leq t \leq 1$$

with a and b representing node positions for both layout modes and t being the slider position was not enough to produce a useful visualization for each value of t. For intermediate slider-settings most nodes tended to cluster in the middle of the display and reduce visibility. The solution to this problem was to spread out the nodes in equal distances in the x and y directions of the screen after interpolating. This way the graphical elements of the visualization would always fill as much of the space provided as possible and graphical elements did not overlap each other anymore. This could be seen as the affine transformation of an image, that is stretched to fill a predefined 2D space. With this final step the desired layout is finished.



(figure 18, the result of an interpolated layout is scaled to be better readable)

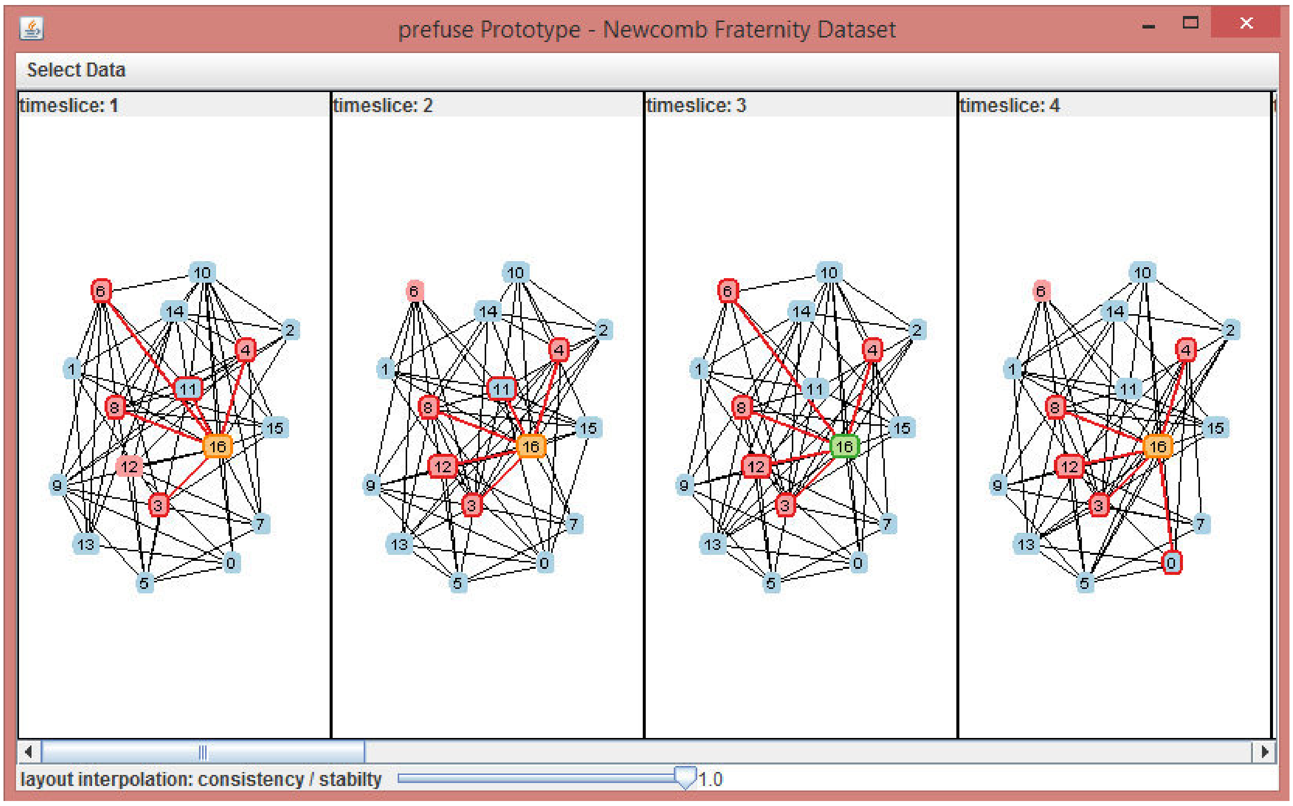
As an additional way to interact with the prototype, I extended prefuse's built-in controls for zooming and panning to apply to all displays at once. This coordinated manipulation lets the user configure an optimal viewpoint in as few steps as necessary.

3.2.3 Highlighting and Color Scheme

Another important feature for providing the information the user is seeking out without cluttering the visualization, is being able to highlight individual nodes and getting information about their evolution over time and their neighbors. This is happening in two different modes whenever the mouse pointer is hovered over a node:

1. trajectories-first: For this set of highlights the selected node is traversed along its trajectories to the other timeslices first. (see “linked graph” in the **Importing Data** section) This way all of its predecessors and successors are found and highlighted. They are visualized through an orange fill color and a slightly darker orange outline. From there edges within the same timeslice are traversed, thus finding all neighbors of the selected node. The edges are colored red and slightly increased in thickness. In addition, neighboring nodes have a red outline. This is useful for tracking a single participant in a social network. Their relationships can be observed at any point in time and we can quickly tell whether they now have a different number of neighbors compared to earlier or later in the evolution of network.

2. edges-first: The second portion of highlights is calculated in the opposite order. The edges of the selected node in its current timeslices are traversed to find its neighbors. Those neighbors are given a light red fill color. Then the same is done along the trajectories to other timeslices of this exact group. The second highlighting mode can be used to track an entire group of participants at once and answer questions like, *"When did this group form?"* and *"How long did they stay connected?"*.


prefuse Prototype - Newcomb Fraternity Dataset
Select Data
timeslice: 1
timeslice: 2
timeslice: 3
timeslice: 4
layout interpolation: consistency / stabilty
1.0

(figure 19, nodes are highlighted by hovering the mouse cursor over them and are temporarily marked by clicking)

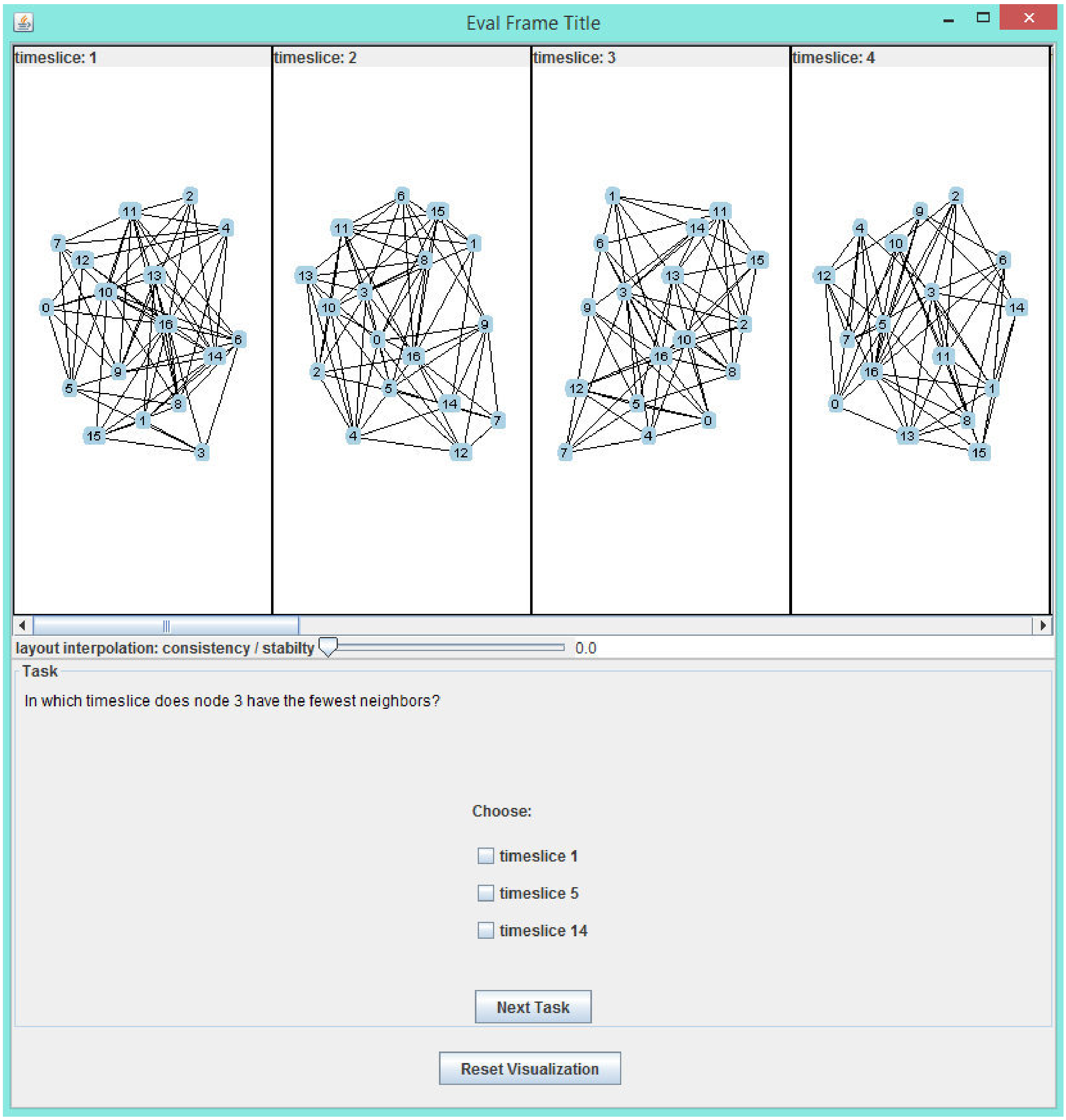
A selected node can be toggled (marked / hovering) by clicking the mouse. Marked nodes are highlighted with a green fill color and a slightly darker shade of green for the outline. This way the mouse can be used for for other things while keeping the highlighting active.

When choosing the color scheme for the visualization and the highlighting in particular I used Colorbrewer[7], a tool for selecting appropriate colors for maps and other graphics. I used the colors depicted above to make each of their individual meanings distinct from each other and to increase readability. [7]

7 www.colorbrewer2.org

3.3 Evaluation Framework

Besides building a software prototype with the features detailed in chapter 1, the second goal of this thesis is to support a way to evaluate all of the implemented features in a quantitative way. The EvalBench framework does exactly that by providing functionality for questionnaires and collecting user feedback, i.e. correctness of the answers and task completion times. [1] I chose a range of properties to customize the associated evaluation tasks, so that future research including this prototype can be done with a diverse set of goals in mind.


Eval Frame Title

timeslice: 1 | timeslice: 2 | timeslice: 3 | timeslice: 4

layout interpolation: consistency / stabilty 0.0

Task

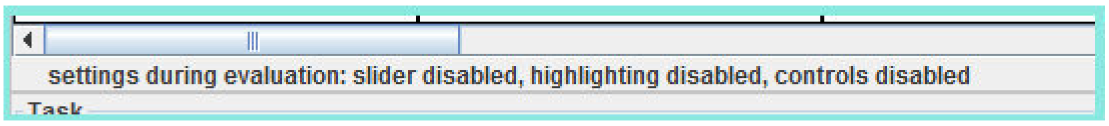
In which timeslice does node 3 have the fewest neighbors?

Choose:

- timeslice 1
- timeslice 5
- timeslice 14

(figure 20, the evaluation window shows part of the visualization at the top and a task description at the bottom)

As part of the evaluation view only a subset of the actual visualization is shown. Multiple attributes are configurable to be included in the evaluation task. These can either be defined as part of an EvaluationSession object's configuration (see EvalBench documentation) or through an XML file, and are displayed as a label at the bottom of the screen.


settings during evaluation: slider disabled, highlighting disabled, controls disabled

(figure 21, an information panel shows the attributes of this evaluation session)

A list of attributes follows:

- *"taskFileName"*: specifies the path to the session's task list. This way an XML file is used to define the evaluation tasks.
- *"training"*: either true or false, specifies whether feedback should be presented to the user after each task. With the training options a few example tasks can be presented before doing the actual evaluation.
- *"dataset"*: specifies which dataset is to be used for the session. Currently the datasets *"newcomb"* and *"vandebunt"* are supported. (see **Importing Data** section).
- *"displays"*: specifies which timeslices (displays) are to be part of the visualization. Allowed are integers separated with a comma. (for example: *"1, 2, 5, 7"*)
- *"sliderPosition"*: sets the initial position of the layout slider to any floating point value between 0 and 1.
- *"disableSlider"*: either true or false, disables the slider from being moved by the user. Note that the initial slider position can still be set.
- *"disableHighlighting"*: either true or false, disables the highlighting functionality
- *"disableControls"*: either true or false, disables the coordinated zooming and panning functionality

A set of sample tasks in XML format can be found in the Appendix and as part of the prototype.

4. Conclusion

I documented my approach to an interactive visualization for dynamic social networks. The resultant prefuse prototype implements the Visual Analytics approach described by Federico et al. [11] The network is displayed as a series of timeslices, that show its evolution over time. The changes from one slice to next are customizable by the user through a layout slider, that interpolates between two states. In the "consistency" state the node positions for each display are individually computed and observe common graph drawing criteria like minimizing edge-crossings and edge-length. The "stability" state minimizes changes between timeslices and serves to preserve the user's mental map. The layout-slider functions as a basic (linear) interpolation parameter. In addition, the layout is adapted to make use of as much of its allotted space as possible by stretching the nodes away from the center and spacing them in equal distances.

To further enable users to visually analyze the data I implemented a dual-highlighting mode. When hovered over with the mouse or selected by clicking, a node is visually emphasized through its fill color, outline and connecting edge stroke-thickness. This happens along two vectors: all of the node's predecessors and successors (nodes with the same id label in other timeslices) are marked, as well as the neighbors of this group. In addition to that, the group of adjacent nodes in the timeslices of the original node are changed everywhere else in hue as well. This way groups as well as individual participants of a network can be tracked and compared across the changing timeslices.

I have detailed how data is imported and what kind of graph structures need to be extracted to be displayed through the prototype. New datasets can be added by extending the code's BAReader interface.

Finally, the software prototype has an evaluation framework built in, that lets interested researchers evaluate its features with a variety of user goals. These are customizable through a range of attributes. Examples for tasks are provided below, as well as with the source code.

In order to further expand on the prototype, more data could be supported, as described above. Additional features for the visualization could include a more sophisticated graph structure with, for example, weighted, unidirectional edges and color-coded nodes. This way more information could be encoded at once and be available on a glance. To improve the GUI, different display arrangements could be included, like an overview, that shows a grid of timeslices or a thumbnail-view, that zooms in on a range of timeslices, showing information on demand.

References

[1] Aigner, W.; Hoffmann, S. and Rind, A., EvalBench: A Software Library for Visualization Evaluation, Computer Graphics Forum, vol. 32, no. 3, p. 41-50, 2013.

[2] Archambault, D., Purchase H.C., and Pinaud B. 2011. “Animation, Small Multiples, and the Effect of Mental Map Preservation in Dynamic Graphs.” In Visualization and Computer Graphics, IEEE Transactions on (Volume:17 , Issue: 4)

[3] Archambault, D., and Purchase H.C. 2013. “Mental Map Preservation Helps User Orientation in Dynamic Graphs.” In Graph Drawing, Lecture Notes in Computer Science Volume 7704, 2013, 475-486

[4] Bender-deMoll, S. and McFarland, D. A. (2006) “The Art and Science of Dynamic Network Visualization.” In Journal of Social Structure. Volume 7, Number 2

[5] Brandes, U., Mader, M., 2011. A quantitative comparison of stress-minimisation approaches for offline dynamic graph drawing. In: Proceedings of Graph Drawing (GD 2011), vol. 7034 of LNCS, pp. 99–113.

[6] www.casos.cs.cmu.edu/computational_tools/datasets/external/newfrat/index2.html

[7] www.colorbrewer2.org/index.php?type=qualitative&scheme=Paired&n=8

[8] Correa, C. D. and Ma, K.-L. 2011. “Visualizing social networks”. In Social Network Data Analytics. C. Aggarwal Ed., Springer, New York, NY, USA.

[9] Eades, P. and Lai, W. 1991. Preserving the mental map of a diagram. In Proceedings of the 1st International Conference on Computational Graphics and Visualization Techniques (Sesimbra, Portugal, September 1991). COMPUGRAPHICS '91. Elsevier, 24 – 33.

[10] Federico, P., Pfeffer, J., Aigner, W., Miksch, S., Zenk, L.: "Visual Analysis of Dynamic Networks using Change Centrality"; in: "Proceedings of the International Conference on Advances in Social Networks Analysis and Mining (ASONAM)", IEEE, (2012), ISBN: 978-1-4673-2497-7; 179 – 183.

[11] Federico, P., Aigner, W., Miksch, S., Windhager, F., Smuc, M.: "Vertigo zoom: combining relational and temporal perspectives on dynamic networks"; in: "Proc. of the Int. Working Conf. on Advanced Visual Interfaces (AVI)", ACM , (2012), ISBN: 978-1-4503-1287-5; 437-440.

[12] Freeman, L. C. 2000. Visualizing social networks. Journal of Social Structure. 1 (1).

[13] Ghani, S., Elmqvist N. and Yi J. S., 2012. “Perception of Animated Node-Link Diagrams for Dynamic Graphs.” Computer Graphics Forum Volume 31, Issue 3pt3, June 2012, 1205-1214

[14] Heer, J., Card, S. K., & Landay, J. A. (2005, April). Prefuse: a toolkit for interactive information visualization. In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 421-430). ACM.

[15] Keim, D A., Mansmann F., Schneidewind J., Thomas J., and Ziegler H. 2008. “Visual Analytics: Scope and Challenges.” In Visual Data Mining, Lecture Notes in Computer Science 4404, 2008, 76–90.

[16] Kobourov, S. G. 2012. “Spring Embedders and Force Directed Graph Drawing Algorithms.” arXiv:1201.3011 (January 14). http://arxiv.org/abs/1201.3011.

[17] Lin, C-C., Lee, Y-Y. and Yen, H-C. 2011. “Mental Map Preserving Graph Drawing Using Simulated Annealing.” In Information Sciences Volume 181, Issue 19, 1 October 2011, 4253–4272

[18] www.prefuse.org/doc/faq

[19] Purchase, H.C. 2002. “Metrics for Graph Drawing Aesthetics.” In Journal of Visual Languages & Computing, Volume 13, Issue 5, October 2002, 501–516

[20] Purchase, H.C., and Samra, A. 2008. “Extremes Are Better: Investigating Mental Map Preservation in Dynamic Graphs.” In Proceedings of the 5th International Conference on Diagrammatic Representation and Inference, 60–73.

[21] Shen, Z. and K-L Ma. 2007. “Path Visualization for Adjacency Matrices”. In Proceedings of the 9th Joint Eurographics / IEEE VGTC Conference on Visualization, 83–90.

[22] Smuc, M., Federico, P., Windhager, F., Aigner, W.,Zenk, L.,Miksch, S: "How do you connect moving dots? Insights from user studies on Dynamic Network Visualizations"; in: "Handbook of Human Centric Visualization", W. Huang (ed.); Springer, New York, USA, 2013, ISBN: 978-1-4614-7484-5, 623 - 650.

[23] www.stats.ox.ac.uk/~snijders/siena/vdBunt_data.htm

[24] Tufte, Edward R. 1983. The Visual Display of Quantitative Information. Graphics Press.

Appendix

What follows is an example XML-file for the EvalBench framework, used to define a task list. Entries in the `<configuration>` tag are attributes described in chapter 3.3.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<taskList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <task id="task1">
            <configuration/>
            <task-description>In which timeslice does node 3 have the fewest
                  neighbors?</task-description>
            <task-instruction>Choose the correct answer.</task-instruction>
            <task-type>TaskType1</task-type>
            <questions>
                  <choice-selection id="q1">
                        <question-text>Choose: </question-text>
                        <correctAnswers>
                              <correctAnswer>timeslice 1</correctAnswer>
                              <correctAnswer>timeslice 5</correctAnswer>
                              <correctAnswer>timeslice 14</correctAnswer>
                        </correctAnswers>
                        <maxChoices>3</maxChoices>
                        <possibleAnswers>
                              <possibleAnswer label="timeslice 1"/>
                              <possibleAnswer label="timeslice 5"/>
                              <possibleAnswer label="timeslice 14"/>
                        </possibleAnswers>
                  </choice-selection>
            </questions>
      </task>
      <task id="task2">
            <configuration>
                  <entry>
                        <key>displays</key>
                        <value>1, 2, 3</value>
                  </entry>
            </configuration>
            <task-description>How many unique neighbors does node 8 have in the
                  first 3 timeslices?</task-description>
            <task-instruction>Choose the correct answer.</task-instruction>
            <task-type>TaskType2</task-type>
            <questions>
            <quantitative id="q1">
                  <question-text>Choose: </question-text>
                  <correctValue>7</correctValue>
                  <is-integer>true</is-integer>
                  <maximum>20</maximum>
                  <maximumString>20</maximumString>
                  <minimum>0</minimum>
                  <minimumString>0</minimumString>
                  <step-size>1</step-size>
                  <tolerance>0</tolerance>
                  <use-spinner>true</use-spinner>
                  </quantitative>
            </questions>
      </task>
</taskList>
```