Rapid Serial Visual Presentation in Dynamic Graph Visualization

Fabian Beck*, Michael Burch†, Corinna Vehlow†, Stephan Diehl* and Daniel Weiskopf†
*University of Trier, Germany
Email: {beckf,diehl}@uni-trier.de
†VISUS, University of Stuttgart, Germany
Email: {michael.burch,corinna.vehlow,daniel.weiskopf}@visus.uni-stuttgart.de

***Abstract*—Rapid Serial Visual Presentation is an effective approach for browsing and searching large amounts of data. By presenting subsequent images at high frequency, we utilize the perceptual abilities of the human visual system to rapidly process certain visual features. While this concept is successfully used in video and image browsing, we demonstrate how it can be applied to dynamic graph visualization. In this paper, we introduce a visualization technique for time-varying graphs that is scalable with respect to the number of time steps. The graph visualization is based on the Parallel Edge Splatting technique, which employs a space-efficient display of a sequence of dynamically changing graphs. To illustrate the usefulness of our approach we analyzed method call graphs recorded during the execution of the open source software system JHotDraw. Furthermore, we studied a time-varying social network representing researchers and their dynamic communication structure while attending the ACM Hypertext 2009 conference.**

I. Introduction

Many application domains deal with graph data that evolves over time, i.e., either the structure itself changes by adding or removing vertices and edges, or the attributes such as the weights of the edges change. An explorative analysis promises interesting insights into the evolution of such data. Software developers, for instance, might be interested in the execution behavior of their software manifested in the dynamically recorded method calls. They could use this information to find possible performance weaknesses or bugs. A social network may also change dynamically: people meet unknown people, people lose track of each other, or cliques expand slowly. Social network analysts can use this information to predict future changes or to investigate the spreading of information.

In general, it is challenging to visualize large static graph datasets using node-link diagrams. Therefore, layout algorithms try to optimize a variety of aesthetic criteria describing a good graph layout, aiming at the minimization of link crossings or the maximization of symmetries [2], [33].

When a graph structure changes over time, the problem of generating a readable node-link diagram becomes even harder. Many approaches use animated diagrams to show the changes (e.g., [10], [13]). In these approaches, the single layouts of the graph need to be optimized to preserve the viewer's mental image of the graph, the so-called *mental map* [32], during animation [10]. Although those animated node-link diagrams are quite accessible, they mainly suffer from scalability issues. Depending on the size of the graph and number of changes, it might not be possible to find a good graph layout for each time step when trying to preserve the mental map. Moreover, the viewer may only keep track of the most recent changes and cannot readily analyze longer sequences.

In this paper, we introduce a dynamic graph visualization technique that is able to display time-varying graph datasets in a scalable way, in the vertex and edge dimension, and as the major contribution of this work, also in the time dimension. To this end, we combine the concepts of *Parallel Edge Splatting* [7] and *Rapid Serial Visual Presentation (RSVP)* [3], [37]. The representation of a set of graphs based on Parallel Edge Splatting already creates scalable and space-efficient graph diagrams in a static layout [7]. This work adds the following new contributions: Animating the sequence of graphs by rapidly scrolling through a long list of diagrams increases the scalability with respect to the time dimension. Moreover, an adaptive slow-down mechanism automatically controls the serial presentation, different modes aggregate long sequences of evolving graphs, and clustering of nodes improves the node layout. Finally, two case studies showing the scalability and practical usefulness of the approach. A video illustrating the approach can be found online: http://www.st.uni-trier.de/vlhcc12/.

II. Related Work

Beck et al. [1] generalize and extend aesthetic criteria for drawing static graphs—those that do not change over time—to dynamic graphs. Among other things, they point out that three dimensions of scalability have to be considered: the number of vertices, the number of edges, as well as the number of graphs (a dynamic graph is usually modeled as a sequence of static graphs). Various approaches for visualizing dynamic graph data have been proposed, but suffer from scalability issues in at least one of these dimensions:

Animated node-link diagrams exploit a natural time-to-time mapping to display the sequence of graphs. The graphs are presented one after the other while animating the transition steps (e.g. [13], [10]). Those diagrams are limited in their abilities to support the analysis of dynamic changes [1]: Remembering states of previous graphs and following the movement of nodes require high cognitive loads. Moreover, the scalability for the single static graphs is as restricted as

for usual node-link diagrams (see [16] for a discussion of the scalability of node-link diagrams).

Reitz et al. [34] proposed an approach that automatically focuses on the changing parts of a dynamic graph by collapsing the stable parts of the graph. This increases certain aspects of the scalability but requires slowly evolving datasets.

Timeline-based diagrams map the time dimension to a space dimension in a non-animated diagram. The TimeArcTrees technique, e.g., uses a sequence of linearized node-link diagrams [17] where many parallel and crossing edges, however, limit the readability. The TimeRadarTrees approach [5] and its enhancement, called Layered TimeRadarTrees [6], are based on matrix representations using radial graphical elements—the circle circumference constrains the number of vertices. Recently, Parallel Edge Splatting was introduced by Burch et al. [7]. Similar to TimeArcTrees, a sequence of graphs is shown in a linearized layout side-by-side, but the readability is improved by splitting the vertices and drawing the edges from left to right. Moreover, based on an edge splatting technique edges are represented as density fields to reduce the clutter in dense diagrams (see [30], [38]). Parallel Edge Splatting is also related to (continuous) parallel coordinates plots [21], [18], [19]. Due to a restricted number of graphs that fit onto the screen side-by-side, Parallel Edge Splatting is still limited with respect to the time dimension.

Rapid Serial Visual Presentation (RSVP) [3], [37], the technique that we apply to tackle the scalability limitation of Parallel Edge Splatting, describes the process of quickly flipping through a set of images, diagrams, or other visual representations. This approach is often applied to video or image browsing [11], [39], [41]. Two subconcepts of RSVP can be distinguished [4]. In *static* RSVP, the displayed entities are stacked exactly on top of each other such that only one entity is visible at a time. Animated node-link diagrams can be considered instances of this approach. In contrast to that, *dynamic* RSVP shows more than a single entity, but subsequences of entities at the same time.

Related but interpreted differently, RSVP is also known as a technique for displaying text word by word [31]. For details on the cognitive processes behind RSVP we refer to Coltheart [8].

III. Visualization Technique

Our approach is built on Parallel Edge Splatting [7] and integrates RSVP. It is thus a combined animated and timeline-based diagram for visualizing dynamic graphs, trying to incorporate the advantages of both paradigms. After specifying the graph data model for the visualization precisely, we explain the graph layout and edge splatting in detail. Then, we show how RSVP is leveraged in the visualization technique and describe interactive features that help exploring the data.

A. Graph Data Model

Adding and removing edges is a relevant aspect of dynamic graphs. On a continuous time axis, edges can be added or removed at any time step. Furthermore, edge weights might change in weighted graphs. When edges point from a source to a target, we talk about directed graphs. The presented visualization approach takes *directed and weighted dynamic graphs* as input and generates a sequence of images to visualize them.

Formally, such graphs can be defined as follows. Let

$$G = (V_A, E_A, \mu)$$

be a directed weighted graph, where V_A denotes the set of n vertices and $E_A \subseteq V_A \times V_A$ the set of directed adjacency edges between the vertices. A weight is assigned to each adjacency edge $e \in E_A$ by a function

$$\mu_G : E_A \to \mathbb{R}^+.$$

A sequence of such graphs

$$\mathcal{G} := (G_1, G_2, \ldots, G_k)$$

with $k \geq 2$ is called *dynamic graph*. Moreover, each graph

$$G_i = (V_{A_i}, E_{A_i}),\ 1 \leq i \leq k$$

of the sequence $\mathcal{G}$ has a timestamp defined by a function

$$t : \mathbb{G} \to \mathbb{R}^+$$

where $\mathbb{G}$ is the set of all possible directed graphs and

$$t(G_i) < t(G_{i+1})\ \forall i,\ \ 1 \leq i < k.$$

Modeling a dynamic graph as a sequence of static graphs as introduced above requires the use of a discrete instead of a continuous time dimension. Nevertheless, the flexible definition of the timestamp function t allows arbitrary temporal differences between two subsequent static graphs, formally, $t(G_{i+1}) - t(G_i)$ is not necessarily constant for all $1 \leq i < k$. Hence, continuous temporal information can be sampled with a fixed or flexible sampling rate to match the data model. In case of non-weighted graphs or discrete changes of edge weights, sampling can be implemented without loss of information because all changes are discrete.

B. Graph Layout

A single graph $G = (V_A, E_A)$ is typically laid out in the two-dimensional plane with the goal to meet certain aesthetic graph drawing criteria. Since our approach aims at showing many graphs side-by-side, a graph layout is needed that is space-efficient and that clearly reflects the graph and link structures at the same time. For this reason, the traditional 2D layout is transformed to a 1D layout, i.e., graph vertices are mapped to one-dimensional vertical lines and edge splatting is also applied.

To achieve the aforementioned goal, the directed graph $G = (V_A, E_A)$ has to be transformed to a bipartite directed graph $G' = (V_A \cup V'_A, E'_A)$ where for each $v \in V_A$ a corresponding $v' \in V'_A$ is introduced. Each directed adjacency edge $e := (v, u) \in E_A$ is transformed to a directed edge $(v, u') = e' \in E'_A$ where $v \in V_A$ and $u' \in V'_A$.

Figure 1 demonstrates how a graph can be displayed in a space-efficient 1D layout by making it bipartite and then mapping the directed edges as straight links from left to right following the reading direction in Western countries.

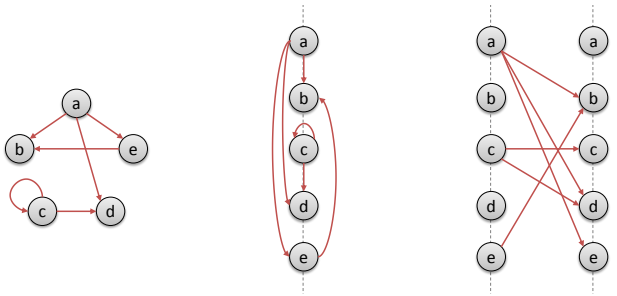

Fig. 1. A graph is transformed to a bipartite graph by copying the vertex set V_A to V'_A and only allow edges between V_A and V'_A. Each vertex group is mapped equidistantly on one axis of two parallel vertical axes and edges as straight links between the axes.

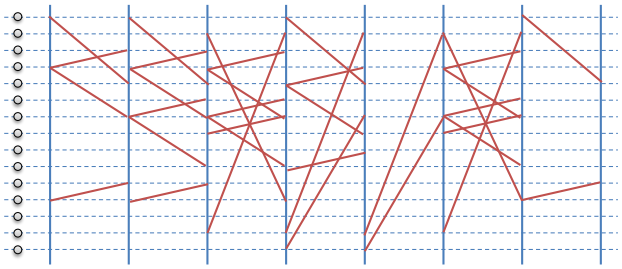

Fig. 2. A dynamic graph is mapped on a sequence of parallel vertical lines with fixed vertical vertex positions in all of the graphs.

A visual representation of edges as straight links between parallel vertical lines has the great benefit that we obtain a space-efficient layout for a directed weighted graph, though visual clutter will be increased initially by the restriction of using 1D instead of 2D. The advantage of this space-efficiency can be exploited for representing several graphs side-by-side—a visual metaphor that is very similar to parallel coordinates. Since the representatives of each vertex are all located on the same horizontal line, there are no layout changes throughout the graph sequence and hence, the mental map of a viewer is easily preserved. Furthermore, viewers can easily explore a graph sequence for dynamic patterns such as trends, countertrends, periodicities, temporal shifts, stabilities, or anomalies because they have the flexibility of looking at a region of interest within the static picture. The single elements of the dynamic data are presented next to each other and can thus be analyzed by visual comparison of components placed side-by-side. In contrast, when using graph animation a viewer has to remember the single elements and the comparison has to be done in his mind relying on short term memory skills. Figure 2 shows a dynamic graph consisting of a sequence of seven static graphs with 15 vertices and 34 edges in a side-by-side static representation.

Another benefit of a static diagram for dynamic graphs is the fact that an additional hierarchical organization of the graph vertices can easily be attached to the diagram. This hierarchical structure is visualized as a layered icicle representation [28] and can be used to orientate and to explore the relational

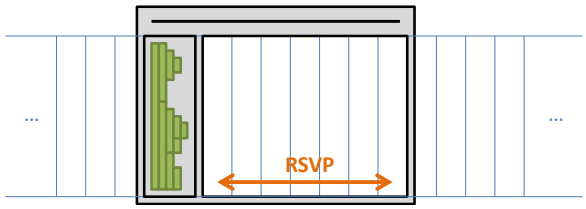

Fig. 3. The concept of RSVP is applied by showing a subsequence of a dynamic graph at a time, using the idea of a sliding time window.

data on different levels of granularity by allowing interactive expanding and collapsing of subhierarchies. Details on the hierarchical structure follow in Section IV-B.

To benefit from the layout restrictions and to perceive the graph and link structure, the concept of Parallel Edge Splatting [7] is applied. In particular, density fields of the link information are computed and the quantitative data points are shown by color coding in a heatmap representation, which makes the graph and link structures more apparent. This also addresses the issue of increased clutter caused by the 1D layout.

C. Rapid Serial Visual Presentation (RSVP)

Though the exploration task for dynamic graph data benefits from static diagrams, the static visualization approach only scales to a very limited number of subsequent graphs. The main contribution of this work is to significantly improve scalability with respect to the time dimension. To this end, we apply the concept of RSVP traditionally used for video browsing. Following this idea, we use the concept of a sliding time window and always represent a subsequence of a longer dynamic graph dataset. In our approach, the concept of dynamic RSVP is applied, i.e., the single graphs of a sequence are not represented at the same location on the display and replaced by the next one after a given delay. Instead, many graphs are shown side-by-side, all contained in the same time window, which can be moved by a given delay, see Figure 3 for an illustration of this concept.

D. Interactive Features

Our visualization approach supports different modes of RSVP, i.e., ways to move the visible part of the static visualization as illustrated in Figure 3. It is possible to navigate through the static view manually using a slider. Furthermore, the RSVP can be performed automatically by clicking a play button or by quickly nudging the slider. For the former, a default speed is used to move the view, whereas for the latter the speed is determined by the strength of the push, i.e., depending on how much and how far the slider was moved. Buttons can be used to regulate, i.e., to slow down or speed up the RSVP.

The speed can also be regulated automatically if the respective feature is enabled. In this case, the animation automatically slows down as soon as two consecutive graphs come into view that differ strongly. As soon as these graphs have reached the center of the view, the animation is accelerated again. The

similarity of two graphs is determined referring to the number and overlay of edges. First, the number of edges within both graphs is evaluated: if the number increases or decreases at a minimal rate defined by a threshold, the difference between these graphs is considered to be strong. But even if the number of edges is similar, the graphs might differ. Hence, the overlap of edges of the two graphs is determined and compared to the union of edges. If this overlap is smaller than a given percentage of the union, the difference between these graphs is also considered to be strong.

Another interactive feature targets the adaption of granularity of displayed information. Therefore, it is possible to collapse internal nodes of the hierarchy, thereby aggregating the edges for all leaves under this internal node, to increase the available space along the vertical lines for the subhierarchies of interest. Furthermore, consecutive graphs can be aggregated, thereby summarizing graphs for certain time intervals and decreasing the number of graphs to be displayed.

IV. Data Transformations

We distinguish between the weighted adjacency edges that may change over time and the static inclusion edges given by a hierarchical organization of the graph vertices. First, we discuss ways to aggregate graph edges. Second, we formalize the hierarchical organization and present a clustering approach to automatically create such a hierarchy.

A. Graph Aggregation

The discrete sequence of static graphs may possibly consist of many graphs. Applying RSVP allows us to show the full sequence, but aggregating the sequence might be still useful for interactively hiding unnecessary details. For instance, in a graph where each edge is added separately at different points in time, aggregating certain frames of time may simplify the analysis. We propose a set of aggregation modes, which reduce the number of graphs or transform the graphs in the sequence. This aggregation can be used as a preprocessing step for visualization.

In our formal model, we transform the sequence of graphs $\mathcal{G} = (G_1, G_2, \ldots, G_k)$ to a sequence $\overline{\mathcal{G}} = (\overline{G}_1, \overline{G}_2, \ldots, \overline{G}_l)$, where $l \leq k$. For the purpose of aggregation, we define a union operation on graphs by

$$G_1 \cup G_2 := (V_{A_1} \cup V_{A_2}, E_{A_1} \cup E_{A_2}, \mu') = G'$$

$$\mu'(e) = \begin{cases} \mu_1(e) + \mu_2(e) & \text{if } e \in E_{A_1} \cap E_{A_2} \\ \mu_1(e) & \text{if } e \in E_{A_1} - E_{A_2} \\ \mu_2(e) & \text{if } e \in E_{A_2} - E_{A_1} \end{cases}$$

- **Semantic Aggregation:** A simple, yet often very useful form of simplification is a semantic-based aggregation. An external source provides the information about what graphs should be merged. For instance, using a calendar, aggregating the graphs on a monthly basis is possible.
  The necessary information can be described as an ordered list of unique indices $I = (i_1, i_2, \ldots i_{l+1})$ where $i_1 = 1$ and $i_{l+1} = k$. These indices mark the points where the sequence of graphs is to be split and aggregated. Hence, the aggregation can be defined as follows:
  $$\overline{G}_j = \bigcup_{i \in [i_j, i_{j+1}-1]} G_i$$
- **Aggregation Frame:** An alternative to the semantic-based aggregation is to divide the time axis into frames of fixed length and to merge the graphs according to these frames. This simplifies the dynamic information independent of some semantic information that might be available.
  For a step size $c \leq \frac{t(G_k)-t(G_1)}{k-1}$ we get $l \leq k$ transformed graphs by aggregation defined as follows:
  $$\overline{G}_j = \bigcup_{t_j \leq t(G_i) < t_j+c} G_i$$
  where $t_j := t(G_1) + (j-1) \cdot c$ is the start of the jth frame. Please note that some graphs $\overline{G}_j$ could be empty because there were no original graphs in the respective time frame.
- **Moving Aggregation Frame:** Instead of dividing the time axis into frames of time, a frame of time with a fixed length can also be moved stepwise along the time axis. The graphs in the time frame are aggregated in each step. This aggregation strategy will lead to overlapping frames and could be helpful if, for instance, each of the original graphs only consists of a few edges.
  We reuse parameter c (see above) for representing the step size guaranteeing $l \leq k$. The size of the time frame is denoted as c', where $c' \geq c$. The aggregation strategy then is defined as follows:
  $$\overline{G}_j = \bigcup_{t_j \leq t(G_i) < t_j+c'} G_i$$
  In case $c = c'$, the moving aggregation frame mode is equivalent to the normal aggregation frame mode.

In case no explicit time information is provided or only the order of events is of interest, we set $t(G_i) = i$. This transforms the two aggregation modes based on time frames into event-based modes—one with a disjoint event frame and one with a moving frame.

B. Hierarchy Model

Hierarchically ordering the vertices of a large graph could provide a better overview of the graph because similar vertices can be grouped together or aggregated. When linearly arranging the vertices as in our visualization, a hierarchy helps finding a meaningful linear order. In the following, we extend our data model towards representing such a hierarchy and discuss an approach to automatically generate the hierarchical structure if necessary.

The hierarchy is defined as a tree structure $H = (V_I, E_I)$. The set $V_I \supset V_A$ denotes a set of vertices and $E_I \subset V_I \times V_I$ a set of directed edges. These edges describe the inclusion

relation between vertices of the tree structure and should not be confused with the adjacency edges E_A. One vertex is designated the root vertex, which has no outgoing edges. In the context of this work, we use a constant hierarchy, although the dynamic graph may change frequently over time. In other words, the inclusion edges stay the same while the adjacency edges may change over time.

If the vertices of a dynamic graph are not ordered, it could be difficult to derive any dynamic pattern from the visualization: Due to the chaotic vertex order, many link crossings clutter the visualization. A hierarchy that groups frequently linked vertices together would automatically reduce the length of the links and the number of edge crossings.

To generate some kind of meaningful one-dimensional vertex ordering we first apply the concept of agglomerative hierarchical clustering proposed by Kaufman and Rousseeuw [27]. To achieve a runtime complexity of $\mathcal{O}(n^2)$, where n denotes the number of all vertices to be clustered, we apply the complete linkage approach of Defays [9]. Complete linkage benefits from the fact that compact clusters of equal diameters can be found as shown by Everitt et al. [12].

The clustering algorithm starts by aggregating the weights μ_i of all adjacency edges of all k graphs into a correlation matrix $C = (c_{ij}),\ 1 \leq i, j, \leq n$ where

$$c_{ij} := \sum_{l=1}^{k} \sum_{(v_i,v_j)=:e \in E_{A_l}} \mu_l(e) \ .$$

This correlation matrix serves as the distance metric of our hierarchical complete linkage clustering algorithm, i.e., we use global distance information to generate a clustering. The algorithm terminates after generating a hierarchical structure $H_{clust} = (V_{clust}, E_{clust})$ containing all vertices of the dynamic adjacency graphs as leaf vertices besides newly introduced inner vertices, i.e.,

$$\bigcup_{1 \leq l \leq k} V_{A_l} \subset V_{clust}$$

and an additional inclusion relation E_{clust}.

V. Case Studies

The goal of these case studies is to demonstrate the scalability of the introduced visualization technique in two application scenarios from different domains. We argued that the technique is well-suited for datasets with a fine-grained temporal resolution and long evolution. Hence, the scenarios were selected with respect to this idea. In particular, the first scenario investigates dynamic method calls observed during the execution of a software system. The second one analyzes a dynamic social network encoding interpersonal contacts during a conference.

A. Dynamic Call Graph Analysis

Software systems are executed at a speed of billions of instructions per second. As one of the basic building blocks, those instructions are grouped into methods. Invoking each other, methods reflect the main flow of information during execution. Analyzing those dynamic method calls instead of static calls retrieved from the source code of the system bears the advantage of observing the real program behavior and not an over-estimated set of theoretic dependencies.

As an example, we study the dynamic method calls of an open source Java system called *JHotDraw*, a graphics framework that is also intended to serve as a benchmark system for studying software design [25]. To get an executable system, we chose the *JavaDrawApp* sample, a simple graphics editor, which is distributed together with JHotDraw. In particular, our test scenario was to start the program, create a new file, draw a rectangle, draw a circle, and write a text into the circle. For recording the dynamic method calls during execution, we employed the *Java Interactive Profiler (JIP)* [26]. It stores the calls in an XML file, which we converted and imported in our visualization tool. The weights of the edges encode the execution times of the called methods.

The actual dataset is large with respect to all three dimensions: It contains 982 vertices (methods and hierarchy vertices) and 32 259 weighted edges (method calls). The Java system divided into two threads, a main thread and a supporting thread, which ran in parallel during execution. These two threads are themselves subdivided into interactions, connected sequences of executed method calls—1 077 in total. The methods are organized hierarchically by the class and package structure of the system. In the following we first aggregate all calls at interaction level and study the difference between those interactions. The next analysis addresses the main thread of the program and investigates its method calls on a finer level of granularity.

1) Overview: To get an overview of the dataset, we want to browse through the call graphs at a high level of abstraction. Here, the semantic aggregation (Section IV-A) provides the possibility to reduce the length of the sequence to a manageable number of graphs. We analyze the dynamic graph at the level of interactions (1 077 graphs) and propose the following approach: First, use the animation at high speed to get a first impression on the dataset in seconds. Next, repeat the animation activating the adaptive slow down feature and make (mental) notes of interesting phenomena. Third and last, study the interesting parts by scrolling to particular time steps manually and comparing the consecutive graphs to the sequence using the scrollbar.

a) Outliers: Among the sequence of graphs on thread level, the first graph representing the main thread is the densest (Figure 4 a). The main thread only consists of one interaction because it is never interrupted or stopped. Analyzing this thread at the level of interaction aggregates all dynamic information and hence is only of limited interest. We study the dynamic behavior of the main thread in detail in Section V-A2.

The other 1 076 graphs of the sequence are interactions belonging to the supporting thread. Many of those interactions are instances of only a few graph patterns. But among those recurring graphs, we also find some other outliers (e.g., Figure 4 b-d). Two of those (Figure 4 b and c) are quite dense

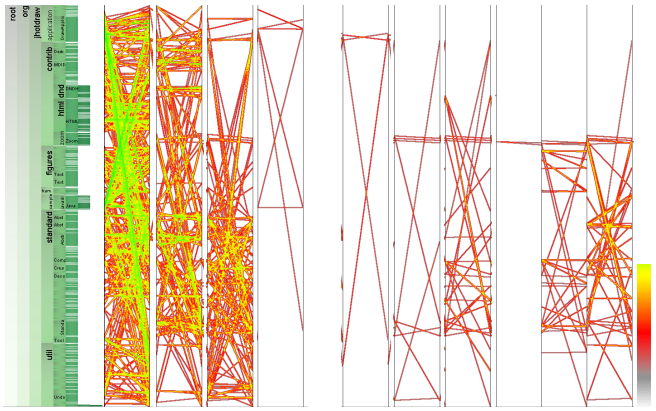

Fig. 4. Outliers (a-d) and recurring patterns (e-h) on the level of interactions in the dynamic method call graph of JHotDraw.

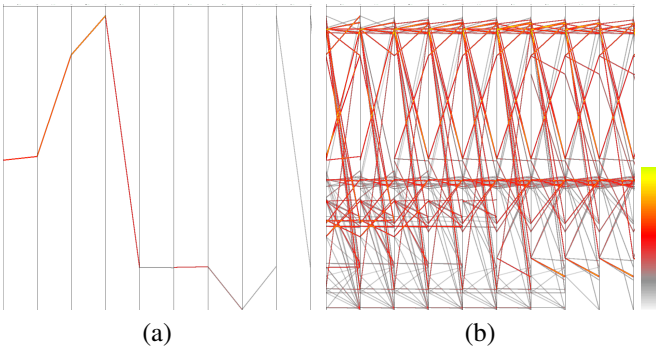

Fig. 5. Method calls in the main thread of JHotDraw; (a) single calls; (b) aggregated calls.

and, as they are outliers, also differ from the previous graphs. Hence, the adaptive animation automatically slows down at these points, which makes these outliers easy to detect, even when browsing through the sequence rapidly.

Figure 4 d, in contrast, shows an outlier that is sparse and only consists of seven method calls. It is the last graph of the sequence where the program shuts down as a reaction to the closing event. Sparse outliers like this graph are harder to detect in the call graph example because most of the recurring patterns are also sparse. But assuming that a particular sparse graph is an outlier, this assumption can be quickly checked by repeating another fast inspection of the whole sequence in a few seconds.

b) Recurring Patterns: While following the animation and scrolling through the graphs, the user can observe recurring visual patterns: For patterns presented side-by-side, the similarity of the graph is obvious and the adaptive animation can move quite fast across these diagrams. If they are gaped, however, by other diagrams in between, recurring patterns are harder to detect, but they are automatically presented at a slower animation speed. We discern patterns that only consist of a single graph and those that are short sequences of graphs. As already mentioned, most of the interaction graphs in the analyzed sequence are repetitions of a few such patterns.

For instance, at the beginning of the whole sequence, there is a single graph pattern that occurs more than a hundred times in subsequent graphs only with few interruptions (Figure 4 e). This sparse pattern consisting only of four method calls reappears again for a longer sequence in the remainder of the dynamic graph. As the recorded raw data tells, this pattern represents the reaction to a mouse movement on the toolbar and a subsequent update of the status bar.

There are similar sparse, but frequently recurring patterns based on single graphs, for example, the two patterns in Figure 4 f and g. Both patterns look similar because some of the edges depicted in Figure 4 f can also be found in Figure 4 g. While the the pattern of Figure 4 f is occurring all over the execution of the program, the pattern of Figure 4 g only occurs in the second half of execution. The explanation is that both patterns show the reaction to a mouse movement on the drawing area: while the first pattern has no consequences for the drawn objects (the mouse is probably over a whitespace area), the second pattern interacts with those objects (a hovering effect; the objects are added during program execution).

Finally, Figure 4 h shows a recurring pattern consisting of more than one graph. It is repeated over 140 times in two subsequences. Comparing this pattern to the patterns of Figure 4 f and g, we see some similarities that might indicate that Figure 4 h is also related to the drawing area. The raw data confirms this assumption: In particular, the pattern reflects a drag operation. At the end of the subsequences repeating the patterns, we find a dense outlier (Figure 4 c), which can be identified as the respective drop operation.

2) Details: In our recorded dataset the main thread included 1 757 method calls in a single interaction. This is the dataset that we want to analyze in more detail. We look at the raw data in form of the separate method calls as well as an aggregated dataset created by applying a moving aggregation frame.

a) Single Method Calls: To preserve the complete information on the execution order of the methods, each of the subsequent method calls has to be included into a single graph. The result is a sequence of 1 757 graphs, each consisting of only one weighted edge. The first nine of such graphs are shown in Figure 5 a. While linear sequences of calls are represented by connected lines over several graphs, gaps in the line indicate returning calls. The color of the edges points to expensive method executions. Browsing through the sequence, the user may search for the most expensive calls or observe the activity in different parts of the program. Moreover, recurring patterns can be retrieved as already demonstrated for the call graphs on interaction level. In the sequence of method calls in the main thread, we, however, note only few such patterns.

b) Aggregated Calls: For aggregating the method calls, we chose a moving frame of size 100 with a step size of 10. The resulting dynamic graph consists of 167 static graphs each consisting of 100 edges and overlapping 90% of the previous/next graph. Figure 5 b depicts the first nine of those graphs. The overall activity in the different parts of the program can be better observed using this aggregated graph: In

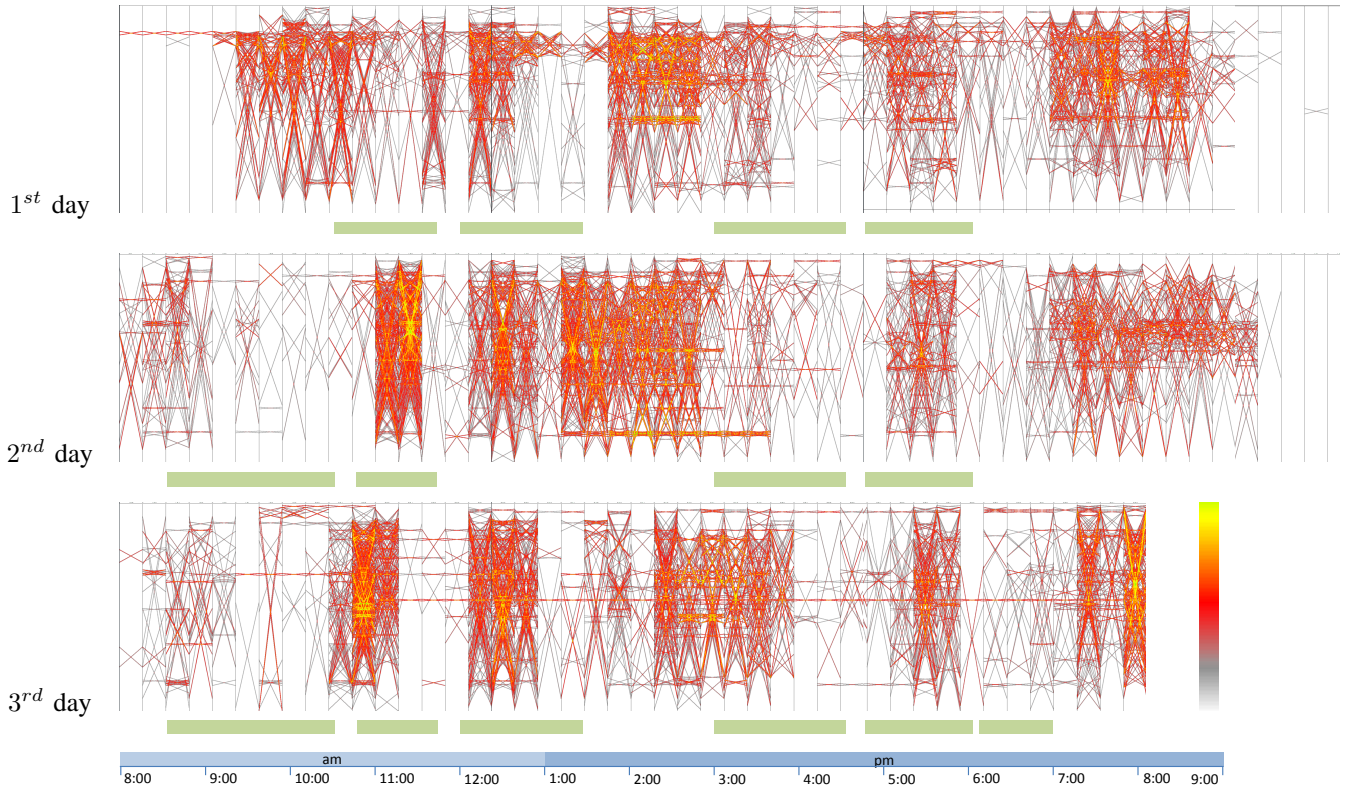

Fig. 6. Face-to-face contacts of the ACM Hypertext 2009 conference attendees (green bars: sessions).

Figure 5 b, for instance, we register high activity for a group of methods near the top border of the diagram as well as for a group in the middle. The moving aggregation, in contrast to a non-moving aggregation, prevents misinterpretations because a non-moving aggregation leads to artificial cuts in the sequence. As Figure 5 b shows, the sequence for the moving aggregation frame is slowly evolving instead.

B. Social Network Analysis

As a second application example for our visualization tool, we analyze a dynamic social network collected during the ACM Hypertext 2009 conference, a dataset that is freely available as a part of the SocioPatterns project [22], [23]. Face-to-face proximity of volunteering conference attendees was monitored by RFID badges. A contact between two attendees was only recorded if they really faced each other in close range over an interval of 20 s. The acquired dataset contains the dynamic network of 20 818 face-to-face proximities of 113 conference participants over a time period of three conference days. The dataset was anonymized, i.e., neither personal data nor other metadata is available.

To retrieve a linear order for the unstructured vertices in this dataset, we applied a hierarchical clustering algorithm as described in Section IV-B. We analyze the dataset in intervals of 15 minutes (frame aggregation mode), which results in a sequence of 236 graphs. Figure 6 provides an overview of this sequence showing 148 graphs (excluding those at night). The interpretation of the figure is supported by a timeline and green bars indicating the sessions of the conference (plenum sessions only; retrieved from the conference program [24]). The compressed image, however, can only give a first impression; the interactive visualization tool is needed to retrieve details at full-screen size.

Social Activity: The visualization reveals simple insights on the general social activity during the conference. For instance, the sessions clearly relate to a significantly lower number of face-to-face contacts. This effect is weaker on the first day, a preconference day with workshop sessions. Temporally shifted gaps in the graph density hint at delayed sessions (e.g., the third session on the third day). The most active intervals seem to be coffee breaks in the morning (second and third day) as well as after the closing session of the conference.

Temporal Patterns: Our visualization could be of particular use when analyzing more complex phenomena like graph patterns that are stable across several intervals. We observe such stable patterns around 1pm at each day, probably during lunch: It shows a quite dense graph only slightly changing across 3–4 intervals (45–60 min). In other intervals, for instance at the end of the first day, some parts of the graph are stable while other parts change. These could be situations where some of the conference attendees are sitting and discussing while other attendees move around and communicate with many different people.

Clusters: Conference attendees who communicate frequently are grouped into clusters and positioned near to each

other in the visualization. In Figure 6, we observe small clusters of attendees that are stable across several hours especially on the third day of the conference (long sequences of edge clusters)—contacts seem to become less volatile towards the end of the conference. A group of four attendees, represented as a cluster near the bottom of the diagram, is quite actively communicating also on the other days of the conference, in particular around noon at the second day. Less active cluster, but more cross-cluster contacts can be observed at the beginning of the conference (before the first session starts) as well as in the morning coffee breaks (second and third day). This might be the premier points to meet new people.

VI. Conclusion

In this paper we demonstrated how the concept of RSVP can be applied to dynamic graph visualization. Parallel Edge Splatting provides a compact side-by-side representation of a sequence of graphs. The presented visualization approach is a hybrid between animated and timeline-based graph diagrams. The major contribution of our work is the improved scalability in the dimension of time. In two application scenarios, we illustrate this benefit by analyzing large graphs consisting of more than thousand time steps. The different aggregation modes particularly helped analyzing the execution behavior of JHotDraw. In a dynamic social network showing face-to-face contacts during a conference, clustering provided a meaningful order of the conference attendees and revealed clusters of edges in the dynamic graph.

References

[1] F. Beck and M. Burch and S. Diehl, Towards an Aesthetic Dimensions Framework for Dynamic Graph Visualizations, In Proc. of the International Conference on Information Visualisation, pages 592-597, 2009.

[2] C. Bennett and J. Ryall and L. Spalteholz and A. Gooch, The Aesthetics of Graph Visualization, In Proc. of Computational Aesthetics, pages 57-64, 2007.

[3] O. Bruijn and R. Spence, Rapid Serial Visual Presentation: A Space-Time Tradeoff in Information Presentation, In Proc. of Advanced Visual Interfaces, pages 189-192, 2000.

[4] O. Bruijn and R. Spence, Rapid Serial Visual Presentation, http://www.iis.ee.ic.ac.uk/ o.debruijn/rsvp.pdf, 2000.

[5] M. Burch and S. Diehl, TimeRadarTrees: Visualizing Dynamic Compound Digraphs, Computer Graphics Forum, 27(3), pages 823-830, 2008.

[6] M. Burch and M. Höferlin and D. Weiskopf, Layered TimeRadarTrees, In Proc. of International Conference on Information Visualization (IV), pages 18-25, 2011.

[7] M. Burch and C. Vehlow and F. Beck and S. Diehl and D. Weiskopf, Parallel Edge Splatting for Scalable Dynamic Graph Visualization, IEEE Trans. on Visualization and Computer Graphics, 17(12), pages 2344-2353, 2011.

[8] V. Coltheart, Fleeting Memories: Cognition of Brief Visual Stimuli (Cognitive Psychology), The MIT Press, 1999.

[9] D. Defays, An Efficient Algorithm for a Complete Link Method, The Computer Journal (British Computer Society), 20(4), pages 364-366, 1977.

[10] S. Diehl and C. Görg, Graphs, They Are Changing, In Proc. of International Symposium on Graph Drawing, pages 23-30, 2002.

[11] E. Eddie and G. Davenport, Video Streamer, In Conference Companion on Human Factors in Computing Systems, pages 65-68, 1994.

[12] B. Everitt and S. Landau and M. Leese, Cluster Analysis, Hodder Arnold Publications, 2001.

[13] Y. Frishman and A. Tal, Online Dynamic Graph Drawing, IEEE Trans. on Visualization and Computer Graphics, 14(4), pages 727-740, 2008.

[14] T.M.J. Fruchterman and E.M. Reingold, Graph Drawing by Force-Directed Placement, Software: Practice and Experience, 21(11), pages 1129-1164, 1991.

[15] M. R. Garey and David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.

[16] M. Ghoniem and J. D. Fekete and P. Castagliola, A Comparison of the Readability of Graphs Using Node-Link and Matrix-Based Representations, In Proc. of IEEE Symposium on Information Visualization, pages 17-24, 2004.

[17] M. Greilich and M. Burch and S. Diehl, Visualizing the Evolution of Compound Digraphs with TimeArcTrees, Computer Graphics Forum, 28(3), pages 975-982, 2009.

[18] J. Heinrich and D. Weiskopf, Continuous Parallel Coordinates, IEEE Trans. on Visualization and Computer Graphics, 15(6), 1531-1538, 2009.

[19] J. Heinrich and S. Bachthaler and D. Weiskopf, Progressive Splatting of Continuous Scatterplots and Parallel Coordinates, Computer Graphics Forum, 30(3), 653-662, 2011.

[20] D. Holten, Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data, IEEE Trans. on Visualization and Computer Graphics, 12(5), 741-748, 2006.

[21] A. Inselberg and B. Dimsdale, Parallel Coordinates: A Tool for Visualizing Multi-Dimensional Geometry, In Proc. of the IEEE Visualization Conference, pages 361-378, 1990.

[22] L. Isella and J. Stehle and A. Barrat and C. Cattuto and J.-F. Pinton and W. van den Broeck, What's in a Crowd? Analysis of Face-to-Face Behavioral Networks, Journal of Theoretical Biology, 271(166), 2011.

[23] http://www.sociopatterns.org.

[24] http://www.ht2009.org/program.php.

[25] http://www.jhotdraw.org/.

[26] http://jiprof.sourceforge.net/.

[27] L. Kaufman and P.J. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis, John Wiley & Sons, Inc., 1990.

[28] J. Kruskal and J. Landwehr, Icicle Plots: Better Displays for Hierarchical Clustering, The American Statistician, 37, 162-168, 1983.

[29] K. Lam and R. Spence, Image Browsing: A Space-Time Trade-Off, Human-Computer Interaction, pages 611-612, 1997.

[30] R. van Liere and W.C. de Leeuw, GraphSplatting: Visualizing Graphs as Continuous Fields, IEEE Trans. on Visualization and Computer Graphics, 9(2), 206-212, 2003.

[31] C.B. Mills and L.J. Weldon, Reading Text from Computer Screens, ACM Computer Surveys, 19(4), pages 329-357, 1987.

[32] K. Misue and P. Eades and W. Lai and K. Sugiyama, Layout Adjustment and the Mental Map, Journal of Visual Languages and Computing, 6(2), pages 183-210, 1995.

[33] H. C. Purchase, Metrics for Graph Drawing Aesthetics, Journal of Visual Languages and Computing, 13(5), pages 501-516, 2002.

[34] F. Reitz and M. Pohl and S. Diehl, Focused Animation of Dynamic Compound Graphs, In Proc. of the International Symposium on Information Visualisation, pages 679-684, 2009.

[35] R. Rosenholtz and Y. Li and J. Mansfield and Z. Jin, Feature Congestion: A Measure of Display Clutter, In Proc. of SIGCHI Conference on Human Factors in Computing Systems, pages 761-770, 2005.

[36] R. Sibson, SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method, The Computer Journal (British Computer Society), 16(1), pages 30-34, 1973.

[37] R. Spence, Rapid, Serial and Visual: A Presentation Technique with Potential, Information Visualization, 1(1), pages 13-19, 2002.

[38] A. Telea and O. Ersoy, Image-Based Edge Bundles: Simplified Visualization of Large Graphs, Computer Graphics Forum, 29(3), pages 843-852, 2010.

[39] T. Tse and G. Marchionini and W. Ding and L. Slaughter and A. Komlodi, Dynamic Key Frame Presentation Techniques for Augmenting Video Browsing, In Proc. of the Working Conference on Advanced Visual Interfaces, pages 185-194, 1998.

[40] B. Tversky and J. Bauer Morrison and M. Bétrancourt, Animation: Can it Facilitate?, International Journal on Human-Computer Studies, 57(4), 247-262, 2002.

[41] K. Wittenburg and W. Ali-Ahmed and D. LaLiberte and T. Lanning, Rapid-Fire Image Previews for Information Navigation, In Proc. of Advanced Visual Interfaces, pages 76-82, 1998.