SONAR: Towards User-Centric Social Network Analysis and Visualization

Christian Hütter*, Björn-Oliver Hartmann*, Klemens Böhm*, Till Heistermann*,
Kevin-Simon Kohlmeyer*, Reno Reckling*, Martin Reiche* and David Soria Parra†
*Institute for Program Structures and Data Organization
Karlsruhe Institute of Technology, Germany
Email: christian.huetter@kit.edu
†MAYFLOWER GmbH
München, Germany
Email: david.soria_parra@mayflower.de

***Abstract*—Social network analysis (SNA) has attracted a lot of attention over the past years. Existing tools for SNA do not allow a user-centric analysis of the *social neighborhood*, i.e., the subgraph of the user's friends and friends of a friend. In this paper, we introduce SONAR, an open source Web application for user-centric SNA. Its extensible architecture and flexible data model allows developers to embed SONAR directly into social networking websites. A performance evaluation shows that our application scales well with the number of users and adds only minimal overhead to the SNA algorithms.**

I. Introduction

In recent years, social networks (SN) have gained great importance, which has spurred the research on social network analysis (SNA). One important technique in SNA is *centrality measures (CM)*, which determine the importance of the individuals within a network [1]. Each CM is designed for a specific context and satisfies specific information needs. In this paper, we consider the following application scenario:

A social networking service for business contacts wants to gain a competitive advantage by allowing its users to analyze their contact networks with techniques from SNA. This lets users identify 'central' contacts which they might want to intensify relations with. Since users have individual information needs, they must be able to select different analysis techniques. This results in the usage of different CM. To address many users, the analysis and visualization must be embedded into the social networking website.

Regarding the Web-based analysis and visualization of SN, the following problems arise: (P_1) The state-of-the-art of SNA is under constant development. So far, dozens of CM have been proposed [1], and we can expect more in the future. Thus, the architecture of SNA tools has to be extensible to allow the addition of novel CM. (P_2) Users have different analysis needs than network analysts. Analysts typically examine the network as a whole. In contrast, users are interested in their *social neighborhood*, i.e., the subgraph of nodes that are only a few hops away from them. Today, no major SN service supports user-centric analysis.

Answering these problems is difficult because of the following technical challenges: (C_1) SN typically are very large and SN services have to serve many requests. Thus, SNA tools must be scalable. (C_2) The underlying data models vary between the services. Thus, the internal data model of the tool has to be flexible. (C_3) Dedicated tools are hard to deploy to SN services. Most existing tools for SNA are standalone desktop applications [2]. A Web-based analysis and visualization tool requires a complete redesign of the software.

We meet these problems and challenges by introducing SONAR, a Web-based tool for SNA. SONAR has an extensible architecture by providing a plug-in API. While SONAR already offers a set of commonly used CM, the plug-in API allows developers to add new CM on the fly. SONAR features a user-centric view that allows each user to analyze his individual SN. To cope with the difficulties of large networks, i.e., expensive computations, SONAR computes the CM on dedicated servers. The client Web browser visualizes the results of the analysis. SONAR provides a flexible data model which handles arbitrary relational representations of the data. We achieve this flexibility by a configurable metamodel for the data. Our Web-based architecture allows social networking services to embed SONAR directly into their websites.

In this paper, we present the architecture of SONAR as well as a performance evaluation. We have developed an operational implementation which can be deployed to any Java Servlet container. SONAR is freely available under the GPL license. The source code as well as a demo are available on the project website http://projectsonar.org/. The results of the evaluation show that our application scales well with the number of users and adds only minimal overhead to the CM computation.

II. Related Work

SNA models social relationships between individuals as a graph. Nodes represent the individuals, and edges represent relationships between the individuals. The *centrality* of a node describes its importance relative to other nodes within the network. Research has proposed many CM that can be used to gain knowledge of the network structure. For lack of space, we limit this section to a short overview and refer to the standard text by Wasserman and Faust [1] for more details.

Brandes and Erlebach [3] distinguish three categories of CM: local, distance-based, and eigenvector-based. Local measures such as In- and Outdegree take into account only

direct neighbors of a node. Distance-based measures such as Betweenness [4] and Closeness [5] compute the shortest paths between all nodes. In eigenvector-based measures such as Google's PageRank [6], nodes may 'pass' their score to their neighbors. While local measures have linear computational complexity, eigenvector-based measures have quadratic complexity and distance-based measures even have cubic complexity. This is challenging for SNA which is typically applied to large graphs.

As SNA is obtaining more and more attention, developers have created a multitude of software tools, both free and commercial. Huisman and Van Duijn [2] have compiled an overview of the most important applications. Two popular research tools are UCINET [7], which is quite comprehensive, and Pajek [8], which focuses on the visualization of large data. NetMiner [9], a commercial tool, is specialized on visual analysis. However, all of these applications are desktop-based. The only other Web-based tool we are aware of is NetVis [10], which is specialized on the visual exploration of online surveys. In contrast, SONAR focuses on the user-centric analysis and visualization of SN services.

III. Requirements

Considering the application scenario from the introduction, there are functional and technical requirements which must be fulfilled. From problems P_1 and P_2, the following functional requirements arise:

R_E *Extensibility:* The techniques for SNA are improved continually, and new CM keep coming up. Thus, the application must allow both the modification of the CM implemented and the addition of novel CM.

R_U *User Centricity:* Users might only be interested in their social neighborhood. Further, users have individual information needs and want to specify which CM are used for the analysis. Hence, the application must support user centricity by letting users make customized analyses based on their social neighborhood.

Challenges C_1 to C_3 lead to the following technical requirements:

R_S *Scalability:* Scalability has two aspects: (1) Large sets of data require a tool that adds only minimal overhead to the calculation of the CM to be visualized. (2) Users might not be able to perform the analysis on their machines due to lack of resources. Thus, computation and visualization should be separated.

R_F *Flexible Data Model:* Different SN sites have different data models. To be suitable for multiple environments, the application must allow a mapping of the given data model to its internal representation of the SN.

R_I *Integrability:* In order to make SONAR accessible for many users, SN services must be able to embed SONAR directly into their websites.

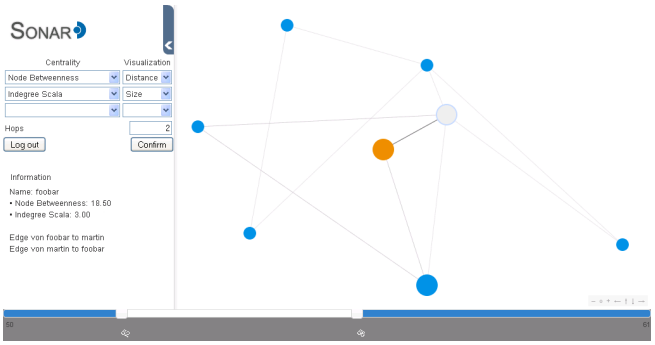

Figure 1. User interface of SONAR showing a user-centric graph

IV. SONAR

In this section, we explain how SONAR meets the requirements and describe its architecture. Figure 1 shows a graph resulting from the following use case:

User Martin wants to analyze his contacts by using the measures Indegree and Betweenness. He specifies that Indegree is visualized as size of the nodes and Betweenness as distance to the center of the graph. The analysis should consider frame $t = [52, 56]$ *and* 2 *hops of the network.*

A. Features

Plug-in API: In order to fulfill R_E, we have developed a plug-in API which simplifies the addition of new CM. Minimal knowledge of the inner workings of SONAR is needed to write new CM. A developer only has to implement the actual algorithm. By inheriting SONAR's base plug-in class, the CM has access to the graph representing the SN. The results of the CM are returned as annotations of the edges or nodes. As soon as the new CM class is accessible to the Java classloader, SONAR deploys it on the fly. SONAR runs on a Java Servlet container, so CM can be written in Java as well as in Java-based functional languages like Scala or Clojure. We have included a set of commonly used CM plug-ins into SONAR: In- and Outdegree, Betweenness [4], Closeness [5] and PageRank [6].

User-Centric View: SONAR features two different views: (1) Network analysts are able to calculate CM based on the *global graph*, i.e., the entire SN consisting of all users and all relationships. They can combine different CM by specifying how the individual measures are visualized as color, size or distance of the nodes and edges. To handle large graphs, analysts can also specify the time frame of their analysis as well as a number n of nodes to be displayed. Then, the CM calculation takes into account nodes and edges that existed within the time frame, and displays only the n most central nodes. (2) What we introduce as *User-Centric View* is the ability to limit the analysis to the social neighborhood of a single user. Users define their social neighborhood by specifying the number of hops they want to analyze, i.e., their friends (1 hop), also the friends of their friends (2 hops), and so on. Except for the limited graph, users have

the same analysis features as analysts. SONAR also supports the exploration of the network by displaying information about each node and edge (see Figure 1). In our example, user foobar has a Betweenness of 18.5 and an Indegree of 3.0. The user-centric view fulfills R_U, and we believe that it motivates further engagement of the users in the platform.

Caching and Load Balancing: To handle large graphs and to serve many requests, SONAR has been developed with scalability aspects in mind. SONAR features the caching of the network graph as well as of the centrality calculation. SONAR executes costly calculations on dedicated servers, while all visualizations are done by the clients using a JavaScript-based rendering approach. Since centrality calculation in SONAR is stateless, it is possible to set up a load balancer which distributes the requests among the available servers. The caching and load balancing satisfy R_S.

Flexible Data Model: SONAR features a *directed multigraph* with annotatable nodes and edges as a universal representation of the SN. We decided to separate the annotations of nodes and edges from the actual structure of the graph. This allows each node and edge to have several annotations (i.e., centrality values), which the individual plug-ins compute. We assume that SN services store their graphs in a relational database management system. An important design decision of ours is to use an object-relational mapping (ORM) between the relational graph database and SONAR's multigraph. We chose Hibernate as ORM library because it enables configuring the database mapping via XML files. This lets us support arbitrary relational representations of SN and thus conform to R_F.

Customizability: The flexible data model and the JavaScript GUI provides easy customizability and thus allows embedding SONAR into existing Web applications. Further, SONAR is open source software: If necessary, developers can adjust every component to conform to an existing user interface. We conclude that SONAR meets R_I.

B. Architecture

An important design decision has been to implement SONAR as Web application using the Google Web Toolkit (GWT). We chose GWT because it provides a powerful infrastructure for high performance Web applications. As illustrated in Figure 2, the architecture of SONAR is split into three parts resembling the Model-View-Presenter (MVP) [11] design pattern. We favor MVP over the classic Model-View-Controller because it features a strong separation between the data model and calculation logic on one hand and the user interface on the other.

Model (Server): The model contains the data to be displayed in the user interface. The server keeps track of all queries, caches incoming requests and responses, communicates with the database and loads and runs the CM plug-ins. The server processes requests as follows: (1) The *Remote Procedure Call (RPC)* system reacts to client requests by dispatching the requests to the centrality calculation unit. (2) The server fetches the specified subgraph S using the database interface. In our use case, the server traverses the graph starting with Martin. All nodes N that are at most 2 hops away from Martin and that existed during time frame t are part of subgraph S. All edges between nodes N from the original graph that existed during time frame t are part of S as well. (3) The *CM Plug-in System* loads the specified plug-ins, i.e., Betweenness and Indegree, in our use case. (4) The server provides access to subgraph S to all plug-ins, which calculate the CM. (5) The RPC system returns the annotated subgraph to the client.

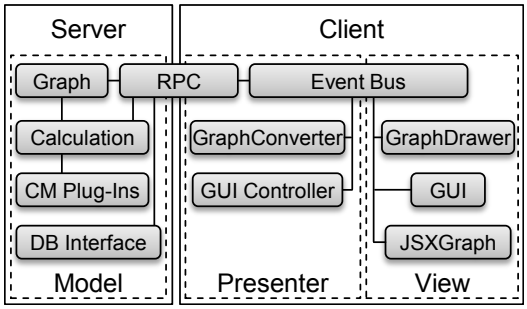

Figure 2. Software architecture of SONAR

Presenter (Client): The presenter retrieves data from the model and prepares it for the view component. It runs as JavaScript executable within the user's Web browser. The *RPC Handler* submits asynchronous RPC requests to the server. It receives the annotated subgraph from the server and forwards it to the Graph Converter. Events are dispatched through a central *Event Bus*, implementing the author-subscriber design pattern. We decided to use an event bus to avoid dependencies between the modules and to decouple the layers. This separation of duties enables easy unit testing of the presenter and the view. To abstract from asynchronous server requests, all inter-module communication is done via the event bus. The *GraphConverter* transforms the subgraph into a visualizable structure. This data structure contains specific information needed to display the network graph, such as the position of each node, its size and its color. As this information is bound to the specific implementation of the client, it is kept away from the server. In our use case, the Graph Converter maps the Indegree of a node to its size and the Betweenness of a node to its distance to the center of the graph. The *GUI Controller* module coordinates the functionality of the client.

View (Client): The view displays data from the presenter and forwards user events to it. It is responsible for displaying GUI elements and for drawing the subgraph within the user's Web browser. We decided to encapsulate the *GraphDrawer* module to make it interchangeable, enabling different kinds of graph renderings. As graph renderer we chose *JSXGraph*, which provides a powerful API combined with an excellent rendering performance.

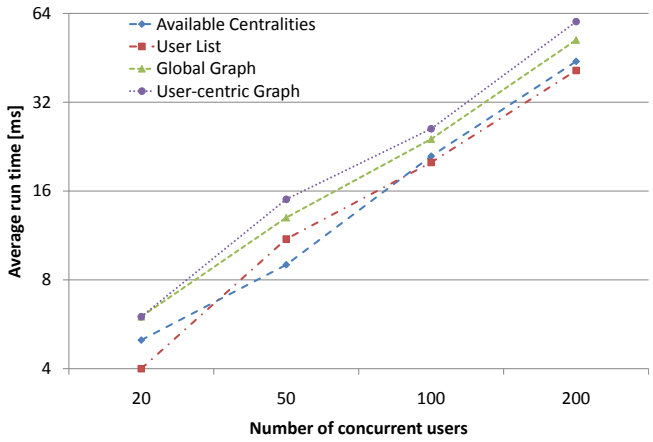

Figure 3. Results of the performance evaluation

V. Evaluation

As a Web application that can be embedded into existing social networking sites, SONAR has to serve requests to a large number of users at the same time. It is critical to ensure short response times of the application as well as a high level of scalability. We have used common performance testing techniques to evaluate the runtime behavior and scalability of SONAR under heavy load.

As server we have used a Sun Microsystem M 24 Workstation with an Intel Quad Core 3 GHz processor, 12 MB L2 cache and 4 GB RAM. To generate the load, we have used two ordinary computers connected to the server via 1 GBit switched Ethernet. We have simulated 20 to 200 concurrent users and their click path through the application using Apache JMeter. We have added the users one by one before we recorded the results so that both our application and the server were able to fill up their caches. We have repeated each test run 30 times to exclude random effects.

The performance test simulated a user who gets (1) the list of available CM as well as (2) the list of users, and calculates two different graphs, (3) a global graph with the 30 most central nodes according to the Indegree measure as well as (4) a user-centric graph representing the friends and friends of a friend (2 hops) of a random user according to the PageRank measure.

Figure 3 shows the average run times of the test cases. The run times increase linearly from 5 ms to 60 ms. We notice that SONAR scales well with the number of users. The server reached its capacity at 200 concurrent users. The cost for getting the available CM and users (i.e., the overhead of our application) is dominated by the cost for calculating the CM on the graphs. Thus, we conclude that our application adds only minimal overhead to the computation of the CM.

VI. Conclusion

We believe that a user-centric analysis and visualization is a promising feature and a competitive advantage for social networking services. Still, a tool that enables a Web-based analysis and visualization has to overcome some problems and challenges. Research in SNA leads to the emergence of novel CM. Thus, CM extensibility is essential. Since CM typically involve complex computations, CM computation and visualization should be separated. Different data models as well as the evolution of data models call for flexible data handling. To support different kinds of online platforms, the tool has to be customizable and in line with Internet standards such as JavaScript.

To overcome these challenges, we propose a novel approach for SNA. Our application – dubbed SONAR – is entirely Web-based, freely available, and can directly be embedded into existing social networking sites. We use a flexible data model that can be adapted to different relational representations without touching the application's code. The architecture is easy to extend by means of a plug-in API for CM. In a performance analysis, we have shown the scalability of SONAR.

We will use our application to conduct SN field studies with immediate network analysis and visualization. Therefore, we are working on an adapter for the OpenSocial REST protocol to query the friends and friends of a friend of the users. This allows us to embed SONAR as social application into OpenSocial networks such as MySpace or hi5.

References

[1] S. Wassermann and K. Faust, *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[2] M. Huisman and M. A. J. Van Duijn, "Software for Social Network Analysis," in *Models and Methods in Social Network Analysis*. Cambridge University Press, 2005.

[3] U. Brandes and T. Erlebach, *Network Analysis: Methodological Foundations*. Springer, 2005.

[4] L. C. Freeman, "A Set of Measures of Centrality Based on Betweenness," *Sociometry*, 1977.

[5] G. Sabidussi, "The centrality index of a graph," *Psychometrika*, 1966.

[6] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," Stanford University, Technical Report, 1998.

[7] S. Borgatti, M. Everett, and L. Freeman, "UCINET: Software for Social Network Analysis," Analytic Technologies, 2002.

[8] V. Batagelj and A. Mrvar, "Pajek: Package for Large Networks," University of Ljubljana, 2003.

[9] Cyram, "NetMiner II," Cyram Co., Ltd, 2003.

[10] J. N. Cummings, "NetVis Module - Dynamic Visualization of Social Networks," MIT, 2003.

[11] M. Potel, "MVP: Model-View-Presenter. The Taligent Programming Model for C++ and Java," Taligent, Inc., Tech. Rep., 1996.