WeKnowIt

Emerging, Collective Intelligence for Personal, Organisational and Social Use

FP7-215453

D4.1.1

Initial community analysis tool

Dissemination level	<Confidential>
Contractual date of delivery	Month 12, 31.03.2009
Actual date of delivery	Month 12, 27.03.2009
Workpackage	WP4 Social Intelligence
Task	T4.3 Community analysis tool
Type	Prototype
Approval Status	< PMB Final Draft >
Version	1.0
Number of pages	47
Filename	D4.1.1_2009-04-02_v01_EMKA_Initial_tool_documentation.odt

Abstract

This is the documentation of the Initial Community Analysis Tool (CAT). This document describes the what, why and how of the CAT. Together with this higher-level description it contains the source code, Javadoc source code documentation and the PostgreSQL DDL statements of the creation of the required database.

co-funded by the European Union

History

Version	Date	Reason	Revised by
0.1	20.02.09	First draft	Michael Ovelgönne
0.2	04/03/09	First "release candidate"; language and style, class descriptions	Michael Ovelgönne, Martin Stein
1.0	26/03/09	Final version	Martin Stein

Author list

Organization	Name	Contact Information
EMKA	Michael Ovelgönne	michael.ovelgoenne@kit.edu
EMKA	Martin Stein	martin.stein@em.uni-karlsruhe.de

Executive Summary

The initial community analysis tool (ICAT) is a software library for the analysis of social networks. A social network can be defined as a set of users and the various relationships among them depending on the use case.

The objective of the initial community analysis tool is to provide social network analysis both for the other work packages and for end users. Other work packages should be able to benefit from the ICAT by improving their results with the provided social knowledge.

The ICAT is based on JUNG (Java Universal Network/Graph Framework). This free open-source package has been chosen because of its speed and easy extensibility. Furthermore, JUNG can be used together with Prefuse, which is a powerful tool-kit to build interactive visualization components. Creating innovative network visualization solutions is the objective of T4.4 (visualization of community information).

The initial community analysis tool consists of two parts. First, a framework for the implementation of analysis functionality consisting of the necessary data structures to represent social networks. Second, algorithms to analyse these networks. Additional functionality to cache (intermediate) data is provided as the ICAT will deal with large amounts of data and complex algorithms. Caching prevents recalculation and allows for a better user experience in terms of response time.

This documentation will describe the functional and technical aspects of the community analysis tool. We will describe how we implemented the ICAT and why we made specific design decisions.

The ICAT will be extended to CAT by D4.1.2.

Abbreviations and Acronyms

SNA	Social Network Analysis
SNS	Social Network Site
HCC	Huge Connected Component
LycosIQ	Lycos Web 2.0 product, Question and Answer Community

Table of Contents

List of Figures

List of Tables

1. State of the Art of Community Analysis

This section summarizes the state of the art in community analysis. A description of the techniques used by popular social network sites is complemented by further methods gathered in a literature review.

Social Network sites (SNS) as defined by Boyd and Ellison [9] are web-based services with three properties.

1. SNS allow users to create a profile within a bounded system.
2. Users can create a list of somehow related users.
3. Users can traverse these lists of connections.

Social network sites can be classified by their main usage purpose or usage scenario. Social network sites in the narrower sense of the word are web-based services with the main purpose to connect people. Boyd and Ellison call this type of SNS profile-centric sites as their focus element is the personal profile of every member. As a second group we define in this paper ressource-sharing sites as web-based services which users predominantly use to get and to give access to some kind of ressource (videos, images, hyperlinks, etc).

We will give an overview of popular SNSs where we put an emphasis on the relationships the sites are able to map. The sites we have inspected are LinkedIn, Xing, MySpace, Facebook, Rummble, Twitter, LiveJournal, Flickr, YouTube, Library Thing, Digg, LycosIQ and Amazon.com.

Our sample of SNS should present an overview of the different kinds of social network sites available to the public so far. As online social networks became popular many existing web-based services integrated social network functionality to their sites. This led to a huge amount of social network sites according to the above definition. Nevertheless, the sample of SNS presented below covers all major kinds of social network sites. It should be kept in mind that SNS are just a fraction of all online social networks. With e.g. email lists, newsgroups and bulletin boards or websites like Wikipedia there are lots of other social networks based on computer-mediated-communication that do not fit in the category of SNS we want to discuss in this paper.

We have a look at those social network sites to determine the social network analysis (SNA) techniques used by those sites. SNA for social network sites can technically make use of every method developed for the analysis of offline networks as both can be represented in the form of graphs. Basically there are two issues when selecting SNA methods for SNS: First, does this method provide meaningful results for the data we have? Second, can we calculate the method in an acceptable amount of time? And third, does it add value in the user perspective?

To answer the first question we need to address the specialities of data from social network services. Social network data can be gathered from various sources. Social scientists often collect data by interviews or questionnaires. As a result of this data source the amount of actors taken into account is quite small but the information gathered can be very extensive. Collecting friendship relation data through interviews will probably result in a pretty complete dataset. However, forgetting some friends or revealing not all friends because of some memory biases will be common sources of error. The kind of relation to be considered as a friendship can be explained by an interviewer to an interviewee. Using data from social network sites means that members differ in who they regard as 'friends' respectively according to which criteria they add somebody to their friendship list. SNS members are aware of the social impact of public friendship lists. Some users "collect" friendship links because of the impression on others they expect to gain from a high number of friends [10]. Furthermore, offline friendships can be modeled in a SNS only if both friends are members of the same SNS. Possible different meanings of SNA measures in online and offline networks have to be considered. What meaning has betweenness in SNS when users can view the personal networks of their direct friends and directly contact a friend of a friend? Likewise, drawing conclusions from the analysis of social networks in SNS for any other environment than the particular SNS is dangerous.

The second problem when analysing SNS measures for their suitability for the analysis of SNS we mentioned is their computational costs. SNS can have several million users. Facebook claims to have reached the 100 million users mark [11]. Myspace is reported to have more than 180 million members in mid 2007 [12] and will probably have passed the 200 million users mark by now. The size of SNS leads to computation problems for complex algorithms. Algorithms that operate in exponential time and/or need exponential memory will most likely not be able to handle very large numbers of actors. All in all the question which of the developed social network analysis techniques can be applied to large social network sites and meet live operation requirements has to be answered.

The third question is whether a given measure is of any value in the user perspective. The users of a social network site are on the one hand the so-called members, the people who create profiles and connect to other members, on the other hand are also those people administering the SNS users. While the first group of users, the members, are interested in features that support them in creating connections, communicating and so on, the second group of users, the administrators, are interested in features that help them foster the SNS and its subcommunities.

When it comes to the analysis of social network data of SNS there are - at least to some extent - different research questions of interest than when dealing with other networks. For example communicating by means of SNS is less expensive in terms of time and money than it is for communicating face-to-face. Additionally, SNS provide more anonymity

and it is easier to pose as somebody else. Therefore, problems usually unknown or insignificant in offline networks like the detection of spammers are of great interest.

1.1.Popular social network sites

This section should give an overview of popular social network sites to show their span and in particular to summarize the different kind of social networks they support. We will regularly refer to these SNS when we describe the state of the art of social network analysis methods for social network sites in the subsequent sections.

1.1.1.Profile-centric sites

The dominant type of webpages on profile-centric SNSs are the user profiles. The members establish them to present themselves to others. The most important use cases are creating connections to other users and communicating with friends or in discussion groups. As profile-centric SNS usually ask users to display many personal data on their profile pages and the web services are of no use without access to member profiles it is mostly mandatory to sign up and log in to these SNSs.

LinkedIn (http://www.linkedin.com) is a SNS intended for business users. Accordingly user profiles have fields to describe one's current and past job positions, one's education and industry experience. LinkedIn members can add tagged connections to other members. Connection tags mark the kind of relationship two users share: colleague, classmate, friend, etc. Beside simple connections users can give other members publicly visible recommendations which include their relationship type and a text field for a short kind of a letter of recommendation. LinkedIn members can create and join groups which do not offer discussion boards but rather serve as a public sign of group membership and allow users to contact other group members. Usually LinkedIn members can only contact first degree contacts (direct contacts) or 2nd and 3rd degree contacts (contacts of contacts respectively contacts of contacts of contacts) via introductions of mutual contacts.

Xing (http://www.xing.com) focuses on the same user group as LinkedIn and offers similar functionality. Connections at Xing may not be tagged and no recommendation feature is available but besides that both SNS are quite similar. Xing allows every premium member (paid membership) to send messages to every other member. Groups at Xing serve in contrast to groups at LinkedIn as discussion platforms. Groups can have several forums for discussions and they provide features to support organizing group meetings.

Facebook (http://www.facebook.com) started as a social network for Harvard students. Later on Facebook opened first to all university students, then to highschool students, then to people in corporate networks and in the end abolished any restrictions. Friendship connections are reciprocal at Facebook and need the confirmation of the target of a friendship request. One of the built-in features of Facebook is that members can upload photos and videos and that they can link users featured in some media item to this item. In addition there are many third-party applications which provide further functionality and create additional networks like e.g. personal attribute rating networks provided by the application ScoreMe or a network of virtual drink servings provided by the application Happy Hour.

MySpace (http://www.myspace.com) is a profile-centric SNS like LinkedIn and Facebook. But as a site for private/leisure use it gives the user greater possibilities to shape his or her profile page. The profile appearance is not as uniform as the profiles of the above described sites. Users can use HTML code to change the text style and the background as well as add pictures, songs and videos to it. The users can also organize their pictures and videos in albums. Profile visitors can post comments and therein include pictures and videos. Relations to other members are called friendship at MySpace which are reciprocal and, therefore, need approval. Favourites are unilateral links to other users. These relations are not publicly visible.

Rummble (http://www.rummble.com/) is a social network site with a focus on locations called rummbles. Members can enter locations of any kind (restaurant, park, beach) with their address and can comment, rate and tag them. Other users can have a look at these locations when they search for a city name or by browsing for locations on a map. Each user has a profile page and pages for past and future trips, a blog and photos. Rummble provides the usual friend lists of SNS but lets the user rate his or her trust to each of his or her friends "on things to do and places to go". In addition users can label the relation and add a comment. Personal attributes (age, sex, religion, etc.) have only a minor role on profile pages. The so-called rummbles take centre stage. However, these locations are not as predominant for the service as the resources of resource-sharing sites.

Twitter (http://www.twitter.com) is a social network site with a very limited usage scenario. The idea is to let people keep track of what others do. Therefore, Twitter users should regulary update the question "What are you doing?" (called update). All other Twitter users who have chosen to become a follower of a specific other user will be informed whenever this user changes his or her answer to the above mentioned question. The profile pages show beside the dominant update history and the list of friends only username, location, a 160 character self description and some statistics. Twitter users can update their own status and receive the status of other users via mobile phone (SMS), instant messaging and the twitter web site. By default every user can follow every other user.

Members who want to maintain control of who gets informed of their updates can choose that following them need their explicit approval.

LiveJournal (http://www.livejournal.com) is a blog service. As a result the most important pages for this service are the users' journal pages were they post blog entries. However, every user has a seperate profile page. This page is less structured than profiles at LinkedIn or Facebook which ask the user to input a lot of information. LiveJournal users can describe themselves mainly by filling in two free text forms for their interests and their biography. Additionally they can pick the schools they attended or still attend from a list. LiveJournal members can add any other user to their friend list. An approval is not required, however, the user profiles seperately display one-sided and two-sided friendships. Additional links can be created to single journal entries of other users to add them to a memory list.

1.1.2.Resource-sharing sites

The resource-sharing sites we investigated are Flickr (images), YouTube (video), LibraryThing (book libraries), Digg (bookmarks) and LycosIQ (knowledge). The main purpose of these web sites is not to connect to current contacts or to find new ones but to share ressources. These websites are open to every web user without prior registration.

Flickr (http://www.flickr.com) is a photo storing and sharing web-based service. Flickr users can upload their photos, annotate and tag the images and arrange them in so-called collections. Other users can post comments to these pictures. Storing and sharing photos is still the main focus of flickr and users can make use of flickr without regarding the social network. Flickr has an unusual history and evolved from an online game with photo uploading functionality. It added SNS features when online social networks became popular. Flickr allows its members to add any other user to their contact list. Links can be marked *as friend* and/or *family.* To add a link no approval of the target is needed. However, a user can block other members to delete all existing links between them, i.e. users can remove themselves from the friendship lists of others (opt-out).

YouTube (http://www.youtube.com) is a kind of equivalent to flickr but for video sharing. YouTube offers two types of relations to other users. First, a user can create a list of friends. Friendship is designed as a reciprocal relationship and an undirected tie between two users is created after following a request and approval process. Besides friendship a YouTube user can create directed relationships called subscription. This type of link needs no confirmation of the target user and is intended to allow users keep track of the videos of favourite co-members. Subscription links can be traversed forward (subscriptions) and backward (subscribers). Beside links to other users YouTube members can also create links to videos by adding them to their favourites list.

LibraryThing (http://www.librarything.com) is a site for publishing one's personal libraries catalogue. User profiles have very sparse information as the site focuses on the users catalogue. Personal catalogues include book ratings and comments. LibraryThing users can maintain three lists of links to other users: reciprocal and confirmed friendships, a public list of interesting libraries and a private watchlist. Adding users to the interesting libraries list or the private watch list does not need their approval but interesting libraries targets will get a notification. Browsing through the user space respectively their libraries can be done by following *friendship* or *interesting libraries* links or via book detail pages as mediators. Most other social network sites just have one dominant page type which is the user profile page. LibraryThing has beside that the book detail pages which show among others some quick facts about a book, member reviews, ratings and recommendations to other somehow related books. Because of this Library Thing has the most prominently featured affiliation network of all in this work considered social network sites.

Digg (http://www.digg.com) is an online service for sharing bookmarks. Users upload interesting URLs to share them. Other users can rate these URL positively *(digging)* or negatively *(burying).* Digg users can add any other user to their friend list. However, unless the target of the friendship does not return the friendship, the connection is marked as *fan* instead of *mutual friend.* Rather than having seperated list for reciprocal friendships and another one-way relation (favourite, fan, etc.) as some other SNSs, Digg uses a single annotated list of connections.

LycosIQ (http://iq.lycos.co.uk) is a web-based question and answer service. Users can ask questions and receive answers from other users. The original intent was to provide an alternative or addition to search engines and let humans instead of computers fulfill the information needs of web users. Users can ask other users to create a reciprocal friendship link. There are also two onesided relations between users at LycosIQ: observing and blacklisting. However, the most dominant network is of course the who-answers-whose-question network.

1.1.3.Other web-based services

A web-based service which does not fit into both before mentioned categories, but lately integrated social network site features, is Amazon.com (http://www.amazon.com). The online shop provides more and more community features. For a long time amazon.com customers could benefit from product reviews of other customers.

amazon.com supports two relations between customers: a reciprocal friendship and an *interesting people* link. Users can write product reviews which can be rated as helpful or not helpful by other customers. For every

user the number of all helpful ratings minus the number of all not helpful ratings is calculated and used to rank the users.

With reviews, item tagging, discussions and lists amazon.com is not very different from resource-sharing sites. Actually if you compare the detail pages ofamazon.com and e.g. YouTube (product page resp. video page) with respect to the page components / features the similarity is amazingly high. Two web sites with completely different origins converge to a point where almost only the offered product differentiates them.

1.1.4.Roundup of SNS

In table 2 and table 1 the properties of the considered SNS are consolidated. The first table shows what kind of relations are supported. The first two columns indicate whether reciprocal or non-reciprocal links between users can be created. Reciprocal links are usually called contacts or friends and are created in an invitation/confirmation process. Non-reciprocal links are usually called favourites, bookmarks or memories and in most cases can be created without an approval of the link target. The last column shows which explicit affiliation networks are supported. This could be group memberships, book ownership and so on. Additionally, various implicit affiliation networks can be detected in SNS data, e.g. tag usage.

		reciprocal	unilateral	affiliation
Profile-centric	LinkedIn	x- labeled	x	groups
	Xing	x		groups
	MySpace	x	x	
	Facebook	x	(3rd party appl.)	groups
	Rummble	x - weighted, labeled		
	Twitter		x	
	LiveJournal		x- labeled	groups
Resource-sharing	Flickr	x- labeled		groups
	YouTube	x	x	groups, favourites
	Library Thing	x		books
	Digg	x	x	
	LycosIQ	x	x	
other	amazon.com	x	x	products

Table 1: Relations in popular social network sites

1.1.6.Levels of analysis of social networks

Analysing social networks can be done in different ways. SNA methods can work with a snapshot of a network at some point in time or explicitly analyse the dynamic in a network, i.e. the change of actors and ties.

We differentiate analysis methods by their view on the network. Actor level methods look at all actors seperately and assign values to them e.g. to rank them by their prestige or their influence. Subnetwork level methods look at groups of actors and decompose the network into subnetworks of cohesive actors or sets of actors with similar attribute properties. Finally, network level methods examine the network as a whole and provide results that describe the entire network. Table 2 gives an overview of SNA methods described in the following sections.

Actor level	Sub-network level	Network level
neighbours	cohesive subgroups	density
centrality	equivalence classes	characteristic path length
behaviour-based measures	attribute-based grouping	diameter
trust	common neighbours	
degree of bi-directionality	co-attendance	
clustering coefficient		

Table 2 Analysis methods by view on the network

		User provided information	system provided information
Profile-centric	LinkedIn	education, work	#contacts, #recommendations
	Xing MySpace	education, work private	#contacts, join date, #profile visits, activity index #profile visitis, last login, #friends,
	Facebook	private, education, work	#friends, #friends grouped by 'net-work', #uploaded photos/videos, date last profile update, #received wall messages
	Rummble	private	#friends, #trips, #blog entries, #pho-tos, #rummbles
	Twitter	rudimental	#following, #followers, #favourites, #direct messages, #updates
	LiveJournal	private, education	#friends, #mutual friends, #groups, interests, #memories, date created, date last update, #journal entries, #comments posted, #comments received, #virtual gifts received, account type
Resource-sharing	Flickr	private	#photos, account type, (#contacts, #image sets)
	YouTube	private, education	join date, last sign in, videos watched, subscribers, channel views, #friends, #uploaded videos, #bulletins, #received comments
	LibraryThing	personal	#books in library, join date, account type, members with same
	Digg	personal	join date, #photos, #friends, must dugg topics, #duggs, #comments, #submitted links, #links made pop-ular, ratio popular links, #invited friends, #profile views
	LycosIQ	self-description	join date, #questions, #answers, %helpful answers, #friends, #track-ers, #bookmarks, rank, iQ
other	amazon.com	personal	reviewer rank, #reviews, #product lists, #uploaded images, #friend, #interesting people

Table 3: Profile information in popular social network sites

1.2.Actor level SNA methods

1.2.1.Neighbours

For every user a social network site has to provide by definition a list of other users with whom they share a connection. Social network sites

that use directed links (non-reciprocal relationships) like Twitter and Flickr usually show a list of all incoming links in addition. So a user can traverse his or her neighbourhood forward and backward.

For privacy reasons, browsing the network is restricted at some sites. At Facebook users can only visit the profiles of their direct neighbours (friends) and view their friends friendship lists. The profile pages of the second order friends (friends of friends) are not accessible.

1.2.2.Shortest path and common contacts

Another usual feature of many social networking platforms is a (visual) display of the users shortest path(s) between a selected other user and himself/herself. Among others Xing also calculates for each two users which are not directly connected the set of common contacts. Users viewing the profile of other members can see if and which persons they share as friends.

1.2.3.Trust measures

Measuring trust in social networks is a very interesting issue as it could be applied to various problems, e.g. fighting spam and generating recommendations.

At Rummble trust is defined as the trust in someones opinion regarding "thing to do and places to go". For each of his/her friends at Rummble a user sets a trust value between 0% and 100% which shall indicate whether a user has no trust at all in a specific contact or respectively completly. For any other reachable user the trust is calculated. The trust propagation algorithm is undocumented. The trust network paths displayed at Rummble indicate that all paths to another user are taken into consideration for propagating trust and not just the shortest paths. Walter et al. give an overview on literature dealing with question whether trust is transitive and propose a model for a trust-based recommendation system on a social network [13]. Further trust propagation models are discussed in [14].

1.2.4.Co-attendance

If we study affiliation networks the co-attendance of two users is the number of events they both have a tie to. In a SNS setting typical events are groups, tags or media items and the corresponding relations are e.g. *member of* or *has as favourite.* Co-attendance can be interpreted as a measure for similarity of e.g. social affinity or taste.

The book community Library Thing displays on each user's profile a list of members with the highest number of books in their library that are also part of the library of the user in question. If we consider books as events in the terminology of affiliation networks then books and community

members build an affiliation network and the mentioned value is called co-attendance. Library Thing shows on profile pages a list of members with libraries similar to the one of the profile owner. The lists are either sorted by the absolute number of shared books or the fraction of shared book to the size of the library or by the fraction of shared books added to the library in the last two weeks. Library Things also calculates two values they call median, respectively mean book "obscurity". This values do not compare the libraries of two users but compares a user's library to all other libraries. Medium and mean book obscurity are the median respectively the mean number of users who own a book that is part of one user's library. Thereby the book obscurity shows how unique one user's library is.

1.2.5.Centrality

Centrality denotes the involvement of a member in a community. Centrality measures are the most widely used measures in social network analysis. They are for example used to estimate prominence, accessibility, influence, or power of an actor.

The most simple centrality measure is the degree centrality. In an undirected network the degree of a node is the number of its neighbours. The degree could be interpreted in several ways according to the meaning of the underlying relation. For a friendship network degree could be interpreted as a measure for popularity.

LycosIQ supports beside the friendship relation a blacklist relation which is a kind of opposite to friendship. Blacklisting a member prevents incoming communication of any kind. For the directed blacklist network the degree centrality is obviously no measure for popularity.

The degree of a user in the friendship / contact network is a common measure social networking sites provide for their users. The social network Xing also provides figures for the number of people in a distance of two steps (sum of distinct contacts of the contacts of a specific user) and three steps. Livejournal and Twitter provide in-degree and out-degree as their relationships are not reciprocal.

Closeness centrality should measure the distances of an actor to the other actors in a network. The most widely used definition is the average geodesic distance from an actor to all other actors. Because the closeness value should increase the closer an actor is located to other actors the inverse of the distances is used. For friendship networks closeness centrality could be interpreted as the ability to get access to information. The lower the average distance of one actor to the other actors is the faster information passed from one actor to the next one will reach him or her.

Betweeness centrality measures the ratio of shortest paths between any two nodes that pass a specific other node. The measure is e.g. used to

assess the influence of an actor on information flow. The more information paths rely on a specific actor the more influence he or she has.

Eigenvector centrality values the centrality of an actor by the centrality values of his/her neighbours. Hence, the centrality of neighbouring actors is reciprocally dependent. In matrix notation the eigensystem $Ax = \lambda x$ has to be solved to get the centrality values. The eigenvector centrality of the i-th actor is the i-th component of the eigenvector of the maximal eigenvalue. The interpretation of eigenvector centrality in friendship or contact networks is e.g. importance. An actor is the more important himself/herself as he/she is connected to other important actors.

1.2.6.Clustering Coefficient

The clustering coefficient measures the degree of connectivity between direct neighbours of a specific node in a graph. A subgraph of a graph G with three nodes and two edges is called triple *at x if x* is incident to both edges. A complete subgraph with three nodes is called triangle. If a node x has at least one triple the clustering coefficient c(v) is defined as c(v) = (number of triangles of v)/(number of triples at v). For instance, this measure has been proposed to detect spammers [15]. If SNS users need to be friends to enable message sending, spammers randomly ask for friendships to be able to send their spam messages afterwards. Random friendship requests lead to unusual low clustering coefficients.

1.2.7.Profile visits

Xing shows by default on every user profile the number of profile visits by other users. This value lets Xing members estimate their own popularity respectively the popularity of other users (especially when compared to signup date). As the Xing search enables the users to search for what other users offer or what they are interested in, the popularity could be both the popularitywithin one's social network (contacts, contacts of contacts) or the attractiveness of one's offerings and interests. Most social network sites display on a users start page (which is different from his own profile) all recent visitors. Other sites just show those recent visitor that did not opt to remain unseen when visiting profiles.

If we consider a multigraph where an arc between a node (user) A and a node B exists for every visit of A to B's profile than B's indegree is the number of profile visits. If we are interested in the number of distinct visitors we can look at the directed graph of profile visits where an arc denotes at least one visit to the profile of the target of the arc by the source of the arc. For public profiles (profiles that can be viewed without prior login to the according website) distinct profile visits cannot be counted. Because of unknown visitors profile visits will usually be calculated by increasing a corresponding counter every time some users

visits a profile page. As this information does not rely on the network profile, visits are not considered as a centrality measure. However, it still could be interpreted as a measure for popularity. Web robots crawling the web are a problem as they can distort the results.

1.2.8.Behaviour-based measures

Xing assigns an activity value to every user according to his or her usage frequency of the social network site. Twitter displays the number of times a user has updated his or her current activity. Other sites like MySpace display the last login of a user on his or her profile. Rummble shows the bygone time since the last profile update. Digg displays the number of submitted URLS, number of comments, number and percentage of submitted URLs which became popular.

With activity figures and last login dates users are able to make a (rough) estimate of the involvement of their fellow community members in that particular SNS whereas profile update dates let visitors of a profile estimate the actuality of the presented information. Digg's popularity values show the quality of submitted resources. Other figures are: number of people who followed an invitation or joined a particular community.

1.3.Subnetwork level SNA methods

1.3.1.Network Decomposition

Clique, k-plex and k-core are measurements for subgroups of a graph for total (clique), nearly-total (k-plex) or lower-bound (k-core) connected subgroups. A clique is a totally connected subgraph that means a subset of nodes of a graph G where each node is connected to every other node. A k-plex is a less restricted subgraph than a clique. Each node does not have to be connected to every other node but just to a minumum of all except k nodes. A k-core is a subgraph where all nodes have to be connected to k other nodes.

Identifying dense groups can e.g. be applied to detect fraudulent behaviour at online auction platforms where different fake users give feedback to fake auctions to obtain a good feedback history by fraud. The buying behaviour at online auction platforms can be represented as a network of users with a directed link from one user to an other when the first one bought at least one item from the latter. The expectable cohesiveness in such a selling network is very low as buyers usually do not buy from just a few sellers and sellers do not sell just to a small group of buyers.

Detecting communities in social networks is another topic in this group of network decomposition problems. Numerous algorithms have been developed to tackle this problem, e.g. by Tyler et al. [16] and Clauset et al. [17].

1.3.2.Attribute-based grouping

With no or just indirect exploitation of the network structure grouping can be achieved by applying standard clustering algorithms to user profile elements (e.g. age, sex, location), behaviour statistics (e.g. number of media uploads, activity value) or network measures (e.g. indegree, outdegree, clustering coefficient).

The most simple way of clustering users is to use a single attribute. For nominal measures the categorization is trivial. Social networking sites often group one user's contacts according to properties of their user profiles. LinkedIn groups contacts according to the industries they work in or the region they live in. Facebook groups friends by their networks (which could be the university they attend, their employer or their residence). By the number of contacts in a specific group users you can for example estimate the local embeddedness of a user.

1.3.3.Manual grouping

MySpace chooses another way of grouping contacts than most SNS and lets user group their contacts by self-choosen categories. MySpace members can add each of their friends to an arbitrary number of categories. The categorized users can see how their friends have categorized them.

1.4.Network level SNA methods

Network level SNA methods describe a network as a whole like the general network-wide figures network size (number of community members), density or diameter.

The density is the ratio of present ties to all possible ties. If a network has a directed relation the number of possible ties is n(n-1) whereas undirected relations result in n(n -1)/2 possible ties.

The diameter of a network is the maximal geodesic distance between any two nodes while the characteristic path length is the average geodesic distance between each pair of nodes in a graph. In other words the diameter is the maximum and the characteristic path length the average length of all shortest paths.

2. Requirements

The requirements analysis for the community analysis tool consists of the contribution of WP4 to the deliverable D7.1 "CSG/ER use case initial requirements". For the initial version of the CAT the following functional and non-functional requirements were derived.

2.1.Notation

Before describing the requirements for social network analysis measures we need to introduce the notation we will use for all equations in this document. The notation is as follows:

G = (V, E) is a graph representing a network where V is the set of actors, E the set of ties of a specific relation, x, y $\in$ V are single actors and (x, y) $\in$ E and {x, y} $\in$ E are directed, respectively undirected ties. N is the number of vertices and M the number of edges in G. Furthermore, d_{xy} is the geodesic distance (length of the shortest path(s)) between the actors x and y, p_{xy} is the number of shortest paths between x and y and $p_{xy}(z)$ is the number of shortest paths between x and y that include node z.

2.2.Functional requirements

The functional requirements describe the scope of services the community analysis tool should provide. The requirements in detail are:

- analyse community structures to estimate popularity, information control power and reachability of actors
- find cohesive subgroups of users

The following set of centrality measures have been identified as requirements for the initial CAT as they are needed for the case study implementations and cover a broad range of analysis demands.

Centrality Measures

- Degree Centrality: measures the number of connections of an actor

$$C_d(x) = \sum_{y \in V \vee (x,y) \in E} 1$$

- Closeness Centrality: the total length of the shortest paths to all other nodes

$$C_c(x) = [\sum_{y \in V} d_{xy}]^{-1}$$

- Inverse Distance Closeness Centrality: similar to closeness centrality but also defined for unconnected graphs

$$C_{idc}(x) = \sum_{y \in V} d_{xy}^{-1}$$

- Betweenness Centrality: measures the number of (shortest) paths between all other vertices that pass a specific vertex x

$$C_b(x) = \sum_{y<z \in V} p_{yz}(x) / p_{yz}$$

Graph Decomposition

- Clustering: detection of cohesive subgroups (high density of connections within a group, few connections between different groups)

The purpose of the initial community analysis tool is to cover a broad range of use with a limited set of tools. Therefore, the measures above were chosen. For other analytic needs the community analysis tool must have a modular structure to allow for extensibility.

2.3.Non-functional requirements

2.3.1.Supported network size

The initial tool has to support graphs of varying size but with the special characteristics of social networks (e.g. very sparse graph, power-law degree distribution). It must be capable of dealing with network sizes at least as large as the LycosIQ dataset. Hence, the analysis of graphs with 100,000 vertices and one million edges must be supported.

2.3.2.Speed

The community analysis tool should provide its results fast enough to enable smooth, undelayed user experience of WeKnowIt systems. Calculations should take less than 1 to 5 seconds, depending on how frequently users call those functions. Caching mechanisms must be supported for systems where the calculation times above cannot be met.

3. Technical realisation

3.1.Overview

The initial community analysis tool is a modular set of classes. An application engineer can choose which modules to use in a specific application context while omitting the other ones. Figure 1 shows an example how ICAT modules can be connected. (The semantic interpreter is planned to be a part of the final CAT version D4.1.2).

The community analysis tool is based on the JUNG framework (Java Universal Network/Graph Framework)[2]. JUNG is free and open-source software and an ongoing project. It has been chosen because of its speed and easy extensibility. This is a crucial issue as there will be several further components to be developed in other tasks of WP4. Furthermore, JUNG can be used together with Prefuse, which is a powerful tool-kit to build interactive visualization components. As of now, we plan to employ Prefuse in the T4.4 as it seems to be the visualization library best fitting the expected requirements.

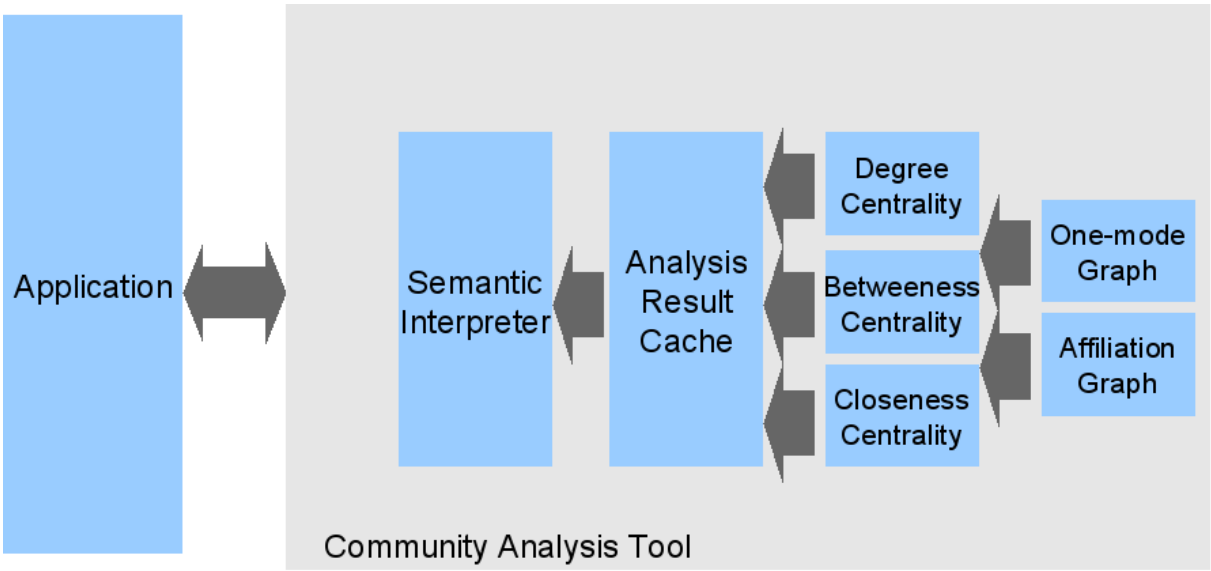

Figure 1: Possible CAT usage scenario

3.2.Data structures

Figure 2 shows a UML class diagram of the graph data structures. The grey classes and interfaces are part of the JUNG library.

The ICAT provides five graph data structures:

- UndirectedSNAGraph

- DirectedSNAGraph
- UndirectedSNAAffiliationGraph
- DBUndirectedSNAGraph
- DBDirectedSNAGraph

UndirectedSNAGraph and DirectedSNAGraph are datastructures storing (un)directed graphs in RAM. DBUndirectedSNAGraph and DBDirectedSNAGraph are their equivalents storing graphs in a database. Furthermore, UndirectedSNAAffiliationGraph stores bipartite affiliation networks in RAM.

SNAGraph is the basic interface for all of our graph classes. This interface ensures that we have vertices and edges from type SNAVertex and SNAEdge. Furthermore, this interface provides the essential methods to access (strongly) connected components.

The two database data structures implement the interface DBStorable which is needed for all classes that store graphs or vertex properties in a database. DBStorable defines the method getPersistentUID() which returns an unique integer value. This unique identifier is necessary to assign vertices and edges to a specific graph. The vertices for all graphs are stored in one table. The same is true for the edges. DBSNAGraph further restricts the permitted type for the vertex identifiers to Integer as supporting arbitrary types of identifiers is not efficiently possible when using a database.

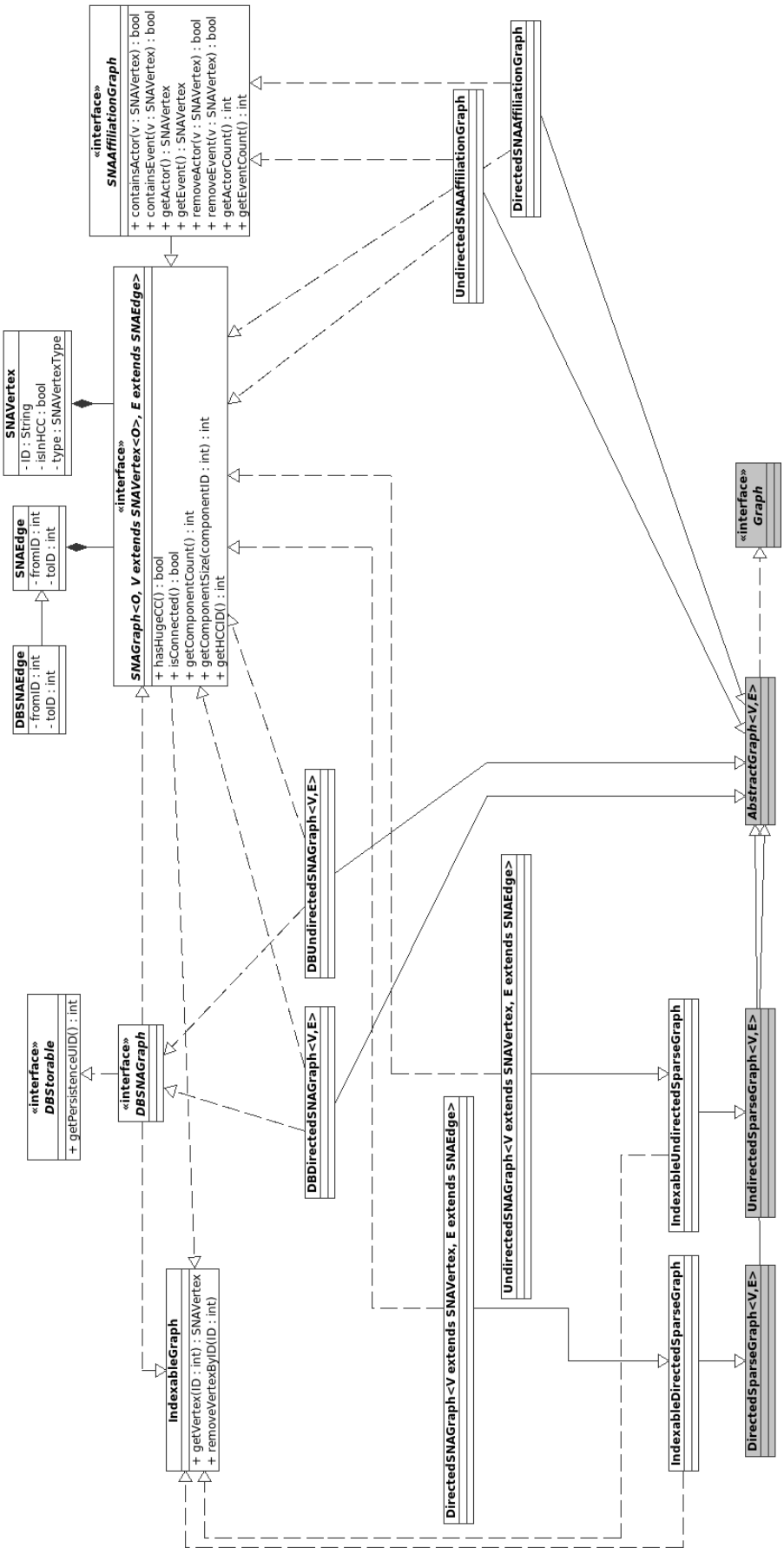

Figure 2: UML class diagram of the graph data structures

3.2.1.Upcoming integration of different layers of intelligence

In the later project phases different layers of intelligence will be integrated. Data exchange is a critical part in any complex system environment. WP3 is the other WP within WeKnowIt analysing data represented as graphs. Therefore, we have collaborated with WP3 to develop a common interface to graph data structures.

The package eu.weknowit.common.graph consists of all classes and interfaces which will be used by different work-packages (DBStorable, IndexableGraph, IndexableUndirectedSparseGraph, IndexableDirectedSparseGraph). The package eu.weknowit.social.cat.graph contains the extensions of the common graph base specific to the the work-package.

3.2.2.RAM data structures

The classes UndirectedSNAGraph and DirectedSNAGraph are RAM data structures for undirected respectively directed graphs. As these data structures store vertices and edges in hash tables in the main memory all basic operations (add, delete, find) are very fast and have a computational complexity of O(1).

Vertices are identified by unique identifiers. Edges are identified by their incident vertices. An additional hash table maps ids to vertex objects. Thus, we can access vertices by their id in O(1). As we are dealing with large and very sparse graphs we store edges in adjacency lists. For each vertex a separate hash table stores all adjacent vertices and the corresponding edges.

Edges may have positive or negative weights. However, distance based measures are only defined for unweighted graphs or for positive edge weights. For the initial community analysis tool the implemented algorithms therefore require non-negative edge weights. As a consequence, they are faster than algorithms capable of dealing with negative edge-weights, since finding shortest-paths when allowing negative edge weights (without further restrictions on paths) is NP-hard.

3.2.3.Database data structures

For huge graphs that do not fit into the main memory the classes DBUndirectedSNAGraph and DBDirectedSNAGraph are the database equivalents for the above mentioned RAM data structures. The advantage of storing edges and vertices in a database is the nearly unlimited

available storage space. The disadvantage is of course the significantly higher access time.

Therefore, the database data structures should only be used when there is not enough main memory available and the time savings gained by RAM storage are not worth the additional costs. For access speed up optimized algorithms and intelligent RAM caching concepts could be used. For the database design we took data query times into account: We decided to denormalize the database schema in order to improve read times. As a result, degree centrality can be queried from the database without needing a UNION statement, which is a costly operation. For further measures relevant subgraphs should be created, stored in RAM data structures and calculated separately. The research and implementation of I/O-efficient algorithms is out of scope of the initial community analysis tool.

3.3. Algorithms

Base algorithms are those algorithms used by other analysis algorithms. The most important base algorithm classes calculate shortest paths and distances. This information is the base of all distance-related algorithms like closeness centrality or betweenness centrality and many graph statistics like diameter or average distance. We use Dijkstra's algorithm [4] to calculate the single-source shortest paths to all other reachable vertices. When the path information is not needed and only distances are of interest, then a distance class (DijkstraDistance, DBDijkstraDistance) can be used instead. The complexity of our implementation is O(|V| log |E|). The shortest path / distance calculation is separated in extra classes to easily reuse the results for the calculation of different measures. This can result in a significant speed-up as a complex and time consuming part does not need to be calculated several times.

The functionality available to the toolbox users is made up of different analysis algorithms. The supported algorithms calculate the network-level measures density, diameter and average degree. At the subnetwork level there are algorithms for connected component decomposition and the clustering algorithm of Newman[5]. At the actor-level the centrality measures degree centrality, closeness centrality and betweenness centrality can be calculated.

Initially we support this set of multi-purpose analysis algorithms. Since we use the JUNG framework as a basis of our tool-kit, further algorithms of JUNG are available, but no adaptation for our specific requirements

(particularly dealing with unconnected graphs) have been made. With sufficient care they can nevertheless be employed for analysis.

Figure 3 provides an overview of the classes in the eu.weknowit.social.cat.algorithms packages (including sub- packages). The classes shown in grey are classes of JUNG. The interface Distance is central, since all distance-based measures need a class implementing Distance. Currently DijkstraDistance and DBDijkstraDistance provide geodesic distance information. While DijkstraDistance optionally caches calculated distances in RAM, DBDijkstraDistance stores the calculated distances in a database.

AllPairsShortestPath also implements the Distance interface and provides distance information. Additionally, AllPairsSPData provides access to the shortest paths. The path information is required by SNABetweennessCentrality which calculates the betweenness centrality.

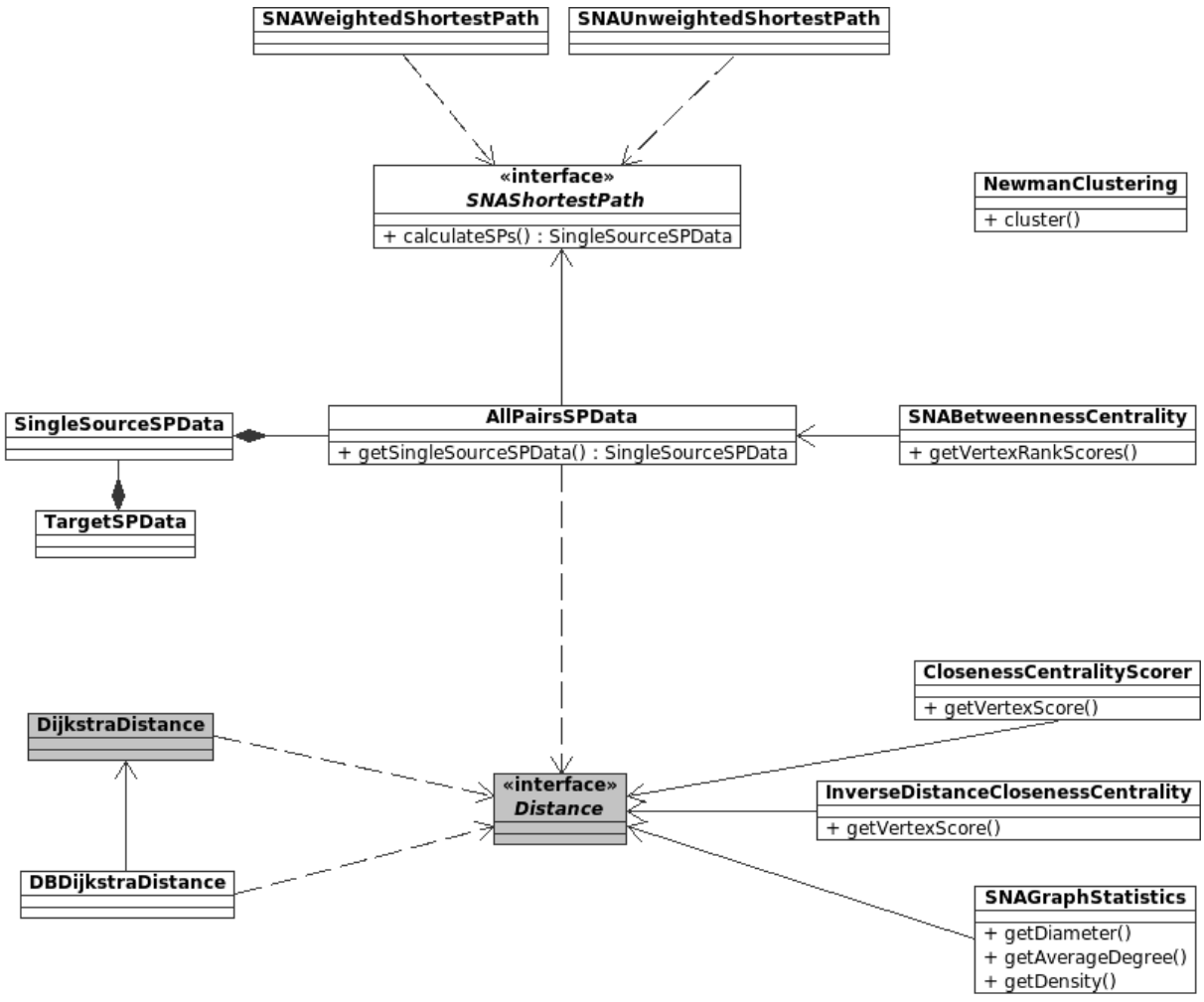

Figure 3: UML class diagram of the algorithm part of the CTA

3.3.1.Dealing with insufficient connectivity

Some of the analysis measures the initial community analysis tool provides are not defined for unconnected graphs or result in undesirable, skewed results. From several analyses of social networks it is known that social networks usually have one huge connected component consisting of most of the vertices. All other components are usually small. For a review on these findings see Mislove et al.[1] and Newman[7] .

The friendship network of the LycosIQ Germany dataset consists of about 80.000 registered users but only about 33.000 links. This results in one large component with approximately 7800 vertices, 72.000 isolated nodes and several very small components. The friendship network of the LycosIQ dataset does not show the usual structure of friendship networks in SNS analysed in [1]. Only very few members made use of the possibility to be connected to other users. The reason seems to be that a significant amount of users sign up at LycosIQ to ask a single question and never return to the site. Nevertheless, the network structure consists of one connected component that is several times larger (more than two orders of magnitude) than the second largest component.

Because of this distribution of connected components, vertices that are not found in the huge connected components (hcc) will be omitted in calculations that are undefined for unconnected graphs. For example, the ICAT defines closeness centrality of non-hcc vertices as zero. As those vertices are isolated from the central component this follows a natural perception of their centrality.

The problem of insufficient connectivity occurs on the network level analysis as well. Since the diameter is defined as the maximal length of a shortest-path between any pair of vertices, the diameter is infinite for unconnected graphs. Adding just a single isolated node to a complete graph would result in a diameter change from one to infinity. For the purposes of community analysis such a result is undesirable. The objective is to provide a stable measure which helps to estimate the structure of a social network. So, if a graph has a hcc, we define the network diameter as the diameter of the largest component of the network.

3.3.2.Network level algorithms

So far, the supported analysis measures on the network level are diameter and density. The diameter algorithm can make use of the same shortest-path and distance classes that can be used for the calculation of all shortest-path / distance based centrality measures. Then, the diameter algorithm just needs to compare the provided distance values to find the maximum. This calculation has a complexity of $O(|V|^2+SP)$ with SP as the complexity of the all-pairs shortest path algorithm executed once for the whole network.

3.3.3.Subnetwork level algorithms

On the subnetwork level the initial community analysis tool provides graph decomposition into connected components (undirected graphs) and strongly connected components (directed graphs). The decomposition for undirected graphs is achieved by a depth-first search. For directed graphs we implemented Tarjan's strongly connected components algorithm[3]. The algorithm runs fast even for huge graphs as its complexity is O(|V|+|E|).

We employ this algorithm to determine the (strongly) connected components and their sizes. These components are very important for actor-level algorithms that are only defined for connected graphs (e.g. closeness centrality).

For the detection of cohesive subgroups we have implemented Newman's community identification algorithm[5]. This hierarchical agglomerative algorithm is fast enough for ad-hoc clustering of medium-sized graphs (tested up to about 1,500 vertices) in an interactive user interface. The complexity for sparse networks is $O(|V|^2)$.

3.3.4.Actor level algorithms

So far, on the actor-level we provide four centrality measures: degree centrality, closeness centrality, inverse distance closeness centrality and betweenness centrality.

Degree centrality measures the number of neighbours of a vertex. For directed graphs we can distinguish inbound and outbound degree centrality, which is the number of predecessors and respectively the number of successors of a vertex. The calculation is trivial and fast: For the RAM graph we store the adjacency list of a vertex in a HashMap. The HashMap stores its size (number of entries) in RAM so that the complexity for retrieving the vertex degree is O(1). For database stored graphs the number of vertices can be acquired by just one simple query on the edge table. As we store undirected edges as two directed edges we buy the fast query functionality with some extra demand in storage space.

The calculation of closeness centrality uses the Distance interface of the package edu.uci.ics.jung.algorithms.shortestpath. Therefore, either DijkstraDistance from edu.uci.ics.jung.algorithms.shortestpath or DB-cached DBDijkstraDistance and AllPairsSPData from eu.weknowit.social.cat.algorithms.shortestpath can be used to provide the distances required by the closeness centrality algorithms.

Inverse distance closeness centrality (id closeness) provides similar results as closeness centrality but has the advantage of being defined on unconnected graphs. The interpretation is not as straightforward as for

closeness centrality. Therefore, we recommend to use closeness centrality for large social networks that will have a huge connected component. If for some reason (e.g. smaller graph) a graph is not connected and neither has a hcc, the inverse distance closeness centrality should be used. The calculation of the id closeness centrality is similar to closeness centrality. The main difference is that id closeness centrality does not necessitate checking for connectivity or strategies to deal with disconnected components.

To calculate BetweennessCentrality we employ the algorithm of Brandes [6]. However, we execute the calculation of shortest paths in a separate class to reuse the results for other distance-based measures such as closeness centrality or diameter. Resulting from the complexity of the shortest path implementation, betweenness centrality runs in O(|V||E|) for unweighted and O(|V||E| log |V|) for weighted networks.

3.4.Interface Overview

The ICAT uses interfaces for many of its data structures and algorithm related classes in order to keep the actual implementations easily interchangeable. Below is an overview of the most important interfaces. A textual description is only given for methods where the name and signature are not self-explanatory. Classes of interest are described as well. Further documentation can be found in the Javadoc part of the appendix and in the source code itself.

Interface SNAGraph<O,V extends SNAVertex<O>,E extends SNAEdge> extends Graph<V,E>, IndexableGraph<O,V,E>, DBStorable	
Interface for all graphs that provide the essential methods to determine and access (strongly) connected components	
Return value	Method signature and description
int	getComponentCount() Returns the number of (strongly) connected components
int	getComponentSize(int componentID) Returns the size of the given component, if the parameter is invalid (out of range) returns -1
int	getHCCID() Returns the ID of the huge connected component. If the graph is connected returns the ID of its single component (which is always 0). If graph has no HCC returns -1
java.util.Collection<V>	getHCCVertices() Returns a Collection of all vertices in the graph that belong to the huge connected component.
boolean	hasHugeCC() Returns true if graph has a HCC or is (strongly) connected, otherwise false
boolean	isConnected() Returns true if the graph is (strongly) connected, otherwise false

Interface SNAAffiliationGraph<O,V extends SNAVertex<O>, E extends SNAEdge> extends SNAGraph<O,V,E>	
Affiliation graph interface. An affiliation network is a bipartite network with two distinct sets of nodes: actors (person, social group, organisation etc.) and events (real event, tag, club etc.). The methods in this interface are similar to those of one-mode graphs, but operate either on actors or events.	
Return value	Method signature and description
boolean	containsActor(V vertex)
boolean	containsEvent(V vertex)
V	getActor(O ID)
V	getEvent(O ID)
boolean	removeActor(V vertex)
boolean	removeEvent(V vertex)
int	getActorCount()
int	getEventCount()

Class SNAVertex<O> extends IndexableVertex<O> implements Cloneable	
The class SNAVertex represents a vertex in a social network. The ID of the vertex is set in the constructor and cannot be changed.	
Return value	Method signature and description
none (constructor)	SNAVertex(O ID) Creates a new vertex with the given identifier ID and the default value for the vertex type, which is SNAVertexType.Actor
none (constructor)	SNAVertex(O ID, SNAVertexType type) Creates a new SNAVertex with the given identifier ID and type.

none (constructor)	SNAVertex(O ID, int componentID, SNAVertexType type) This constructor also sets the component id of the vertex. Usually, this is only necessary if a graph has been created, the connected components have been calculated, then the graph was stored in database and gets recreated.
SNAVertex-Type	getType()
int	getComponentID()
void	setComponentID(int componentID)
Object	clone() Assumes that the index O of SNAVertex is immutable. Returns a copy of this vertex with the same index O and the same type but with componentID reset to -1.

Class SNAEdge implements Cloneable	
The class SNAEdge represents an edge in a social network.	
Return value	Method signature and description
none (constructor)	SNAEdge() Creates a new SNAEdge-instance with its weight set to 0 (unweighted). A weight-value can still be set afterwards via the setWeight-method.
none (constructor)	SNAEdge(double weight) Creates a new SNAEdge-instance with the given weight.
double	getWeight()
void	setWeight(double weight)
void	setComponentID(int componentID)
Object	clone() Returns a copy of this edge.

Interface SNAShortestPath<V,E>	
Interface for algorithms which implements the calculation of the single-source shortest path problem in a given graph.	
Return value	Method signature and description
SingleSourceSPData<V>	calculateSPs(Graph<V,E>graphV source) Calculates the shortest paths in the graph based on the given source vertex.

Class SNAGraphStatistics	
SNAGraphStatistics provides functions to calculate values that describe the graph as a whole.	
Return value	Method signature and description
double	getDensity(SNAGraph<O,V,E> graph) Calculates the density of the given graph or returns -1 if a calculation error occurs.
double	getDiameter(SNAGraph<O,V,E> graph, Distance<V> distance) Calculates the diameter (length of the longest of all shortest paths) of the graph. If the graph is unconnected and has a hcc, it returns the diameter of the hcc. For unconnected graphs without a hcc, returns Double.POSITIVE_INFINITY. The diameter can alco be calculated based on AllPairsSPData instead of a Distance object.
double	getDiameter(SNAGraph<O,V,E> graph, Transformer<E,Double> transformer) Calculates the diameter of the graph. Unlike the other getDiameter-methods, this one calculates the path distances internally instead of getting the distance information from a parameter object. Therefore, only use this method if you have not precomputed the distances yet and if you do not plan to reuse those values.
double	getAverageDegree(SNAGraph<O,V,E> graph, Transformer<E,Double> transformer) Calculates the average degree (for undirected graphs) or the average indegree = average outdegree (for directed

	graphs).
Class NewmanClustering<O,V extends SNAVertex<O>, E extends SNAEdge>	
This class delivers the functionality to cluster the vertices of an unweighted, undirected graph into tightly knit groups (many intra-group edges, few inter-group edges). It implements the algorithm proposed by Newman[5]. It is assumed that we operate on a simple, sparse graph.	
Return value	Method signature and description
List<List<V>>	cluster(SNAGraph<O,V,E> graph) Clusters all vertices in the given graph into cohesive subgroups. If the graph is not connected, each connected component is clustered separately.

3.5.Demo Application

Beside the SNA toolbox we developed a web-based sample application to demonstrate the capabilities of the initial community analysis tool (Figure 4). The demo shows pages called "profiles" which provide information about a profile's owner. The demo assumes to have a logged in user whose name is shown in a box in the upper right.

The sample application uses the LycosIQ dataset and presents data in four categories. The upper-left region shows basic member information. The lower-left region shows the ego-friendship-network of the profile owner. The friend list is grouped by clusters which have been found employing Newman's algorithm on the subgraph induced by all friends of the profile owner.

The upper right area shows network ratings for the friendship network of the profile owner. Popularity has been measured by degree centrality, reachability by closeness centrality and information control by betweenness centrality. These are the standard interpretations for the mentioned measures when applied to friendship networks [8].

Every rating is shown twice: normalized for the connected component a user belongs to (a user's extended network = all users reachable by some path) and normalized for the whole network. This separation demonstrates that members that are less popular or reachable in the whole network can still be popular and reachable within their restricted group.

The lower right area displays results of the analysis of the connection between the profile owner and the logged in user who views this profile page. First, the path that connects both in the friendship network is shown. As there might be several shortest paths in the network connecting the two members their number is shown below the randomly selected displayed path.

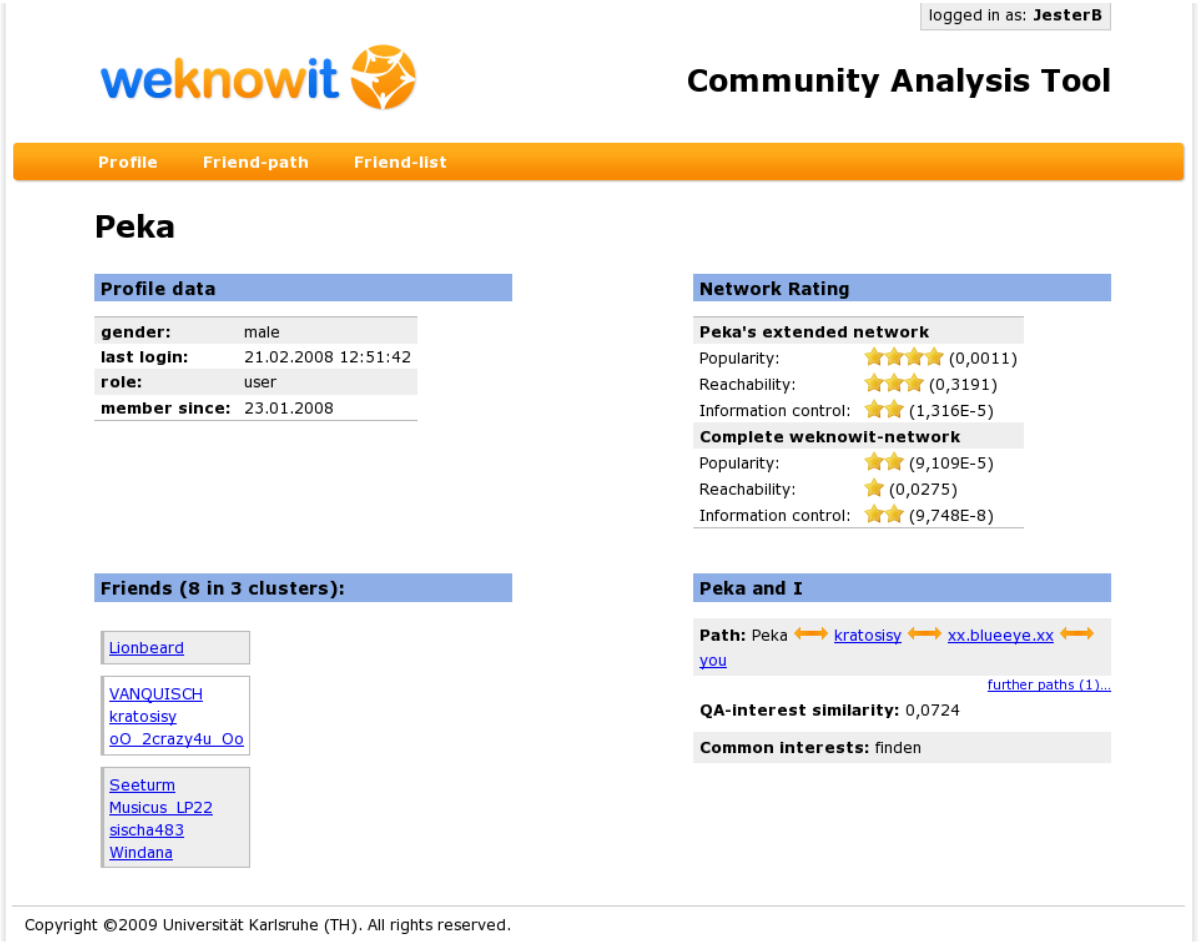

Figure 4: Screenshot of the demo application

As a demonstration for an affiliation network we created a member – tag network from the LycosIQ dataset. A member has a link to a tag whenever he or she posted a question that is tagged with that term or when he or she answered to one of these questions. This affiliation network is used to determine the Question&Answer-interest similarity (measured by the cosine similarity) and the common interests. For this purpose we regard tags that are connected to both considered users as a common interest.

4. Conclusions

With the Initial Community Analysis Tool we provide a fast and scalable toolbox for social network analysis. Analysis methods are provided on all three network levels: actor level, sub-network level and network level.

To demonstrate the capabilities of the CAT a web demo has been developed. This sample application shows that the developed toolbox is able to provide fast and reliable calculations of the described social network analysis functionality for different networks that could be extracted from the LycosIQ dataset.

The initial community analysis tool provides a sufficient range of functionality for a basic social network analysis. Task 4.3 is not finished with this deliverable but will continue to improve the CAT. The final version of the community analysis tool will be delivered in D4.1.2.

The initial version of the CAT now available to all other work packages. The functionality can be directly provided to the user or it can be used to improve other parts. Further within the project the functionality of the CAT will be employed in the community browser of task T4.4, in the integration task T2.3 and T4.1 and for the case study implementations in tasks T7.1.2 and T7.2.2.

5. References

[1]Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi and Peter Druschel, "Measurement and analysis of online social networks", Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, p. 29-42, 2007

[2]Java Universal Network/Graph Framework JUNG, http://jung.sourceforge.net/

[3]Robert Tarjan, "Depth-first search and linear graph algorithms", SIAM Journal on Computing, 1(2), p. 146-160, 1972

[4]E. W. Dijkstra, "A note on two problems in connexion with graphs." *Numerische Mathematik*, 1, p. 269–271,1959

[5]M. E. J. Newman, "Fast algorithm for detecting community structure in networks", Physical Review E, 69, p. 066133-1 – 066133-5, 2004

[6]Ulrik Brandes, "A faster algorithm for betweenness centrality", Journal of Mathematical Sociology, 25 (2), p. 163-177, 2001

[7]M. E. J. Newman, "The structure und Function of Complex Networks", SIAM Review, 45 (2), p. 167-256, 2003

[8]Stenley Wassermann and Katherine Faust, "Social Network Analysis", Cambridge University Press, 1994

[9]Danah M. Boyd and Nicole B. Ellison, "Social network sites: Definition, history, and scholarship", *Journal of Computer-Mediated Communication,* 13, p. 210-230, 2008

[10] Stephanie Tom Tong, Brandon Van Der Heide, Lindsey Langwell, and Joseph B. Walther "Too much of a good thing? the relationship between number of friends and interpersonal impressions on facebook", *Journal of Computer-Mediated Communication,* 13(3), p. 531-549, 2008.

[11] Mark Zuckerberg, "Our first 100 million", The Facebook Blog, August 2008. Available online at http://blog.facebook.com/blog.php?post=28111272130, visited September 22nd 2008.

[12] N.N., "Victim's parents want action against online predators", ABC News, 2007. Available online at http://abcnews.go.com/WN/Technology/story?id=3447710&page=1, visited September 22nd 2008

[13] Frank Edward Walter, Stefano Battiston, and Frank Schweitzer "A model of a trust-based recommendation system on a social network", *Autonomous Agents and Multi-Agent Systems,* 16(1), p. 57-74, February 2008.

[14] Cai-Nicolas Ziegler and Georg Lausen, "Propagation models for trust and distrust in social networks", *Information Systems Frontiers,* 7(4/5), p. 337-358, 2005

[15] P. Oscar Boykin and Vwani P. Roychowdhury, "Leveraging social networks to fight spam", *Computer,* 38(4), p. 61-68, April 2005

[16] Joshua R. Tyler, Dennis M. Wilkinson, and Bernardo A. Huberman, "Email as spectroscopy: automated discovery of community structure within organizations", p. 81-96, 2003.

[17] Aaron Clauset, M. E. J. Newman, and Cristopher Moore, "Finding community structure in very large networks", *Phys. Rev. E,* 70(6), Dec 2004.

A. Appendix

A.1.Javadoc source documentation

see folder javadoc/

A.2.Java source files

see folder java/

A.3.Database tables

The community analysis tool uses a PostgreSQL database to store graphs (vertices, edges) and analysis results. All tables are intended to store data for more than one graph and therefore a unique graph_id identifies to which graph a vertex, edge or analysis result belongs to.

A.3.1Table graphs

This table is used for an overview of all graphs existing in the database. It is used to assure new graphs receive a unique graph_id.

```
CREATE TABLE graphs
(
  graph_id serial NOT NULL,
  description character varying(80),
  diameter double precision,
  CONSTRAINT graphs_pkey PRIMARY KEY (graph_id)
)
WITHOUT OIDS;
ALTER TABLE graphs OWNER TO weknowit;
```

A.3.2Table vertices

The table vertices stores the vertices of database-stored graphs (DBUndirectedSNAGraph, DBDirectedSNAGraph) as well as related analysis results. Vertices of RAM graphs (UndirectedSNAGraph, DirectedSNAGraph) can be stored, too, if centrality values need to be stored.

```
CREATE TABLE vertices
(
  graph_id integer NOT NULL,
  vertex_id integer NOT NULL,
```

```
  component_id integer,
  actor boolean, -- true: vertex is an actor...
  closeness_centrality double precision,
  betweenness_centrality double precision,
  inverse_distance_closeness_centrality double precision,
  CONSTRAINT vertices_pkey PRIMARY KEY (graph_id, vertex_id)
)
WITHOUT OIDS;
ALTER TABLE vertices OWNER TO weknowit;
COMMENT ON COLUMN vertices.actor IS 'true: vertex is an actor
false: vertex is an event';
```

A.3.3Table edges

The table edges stores the edges for directed as well as undirected graphs. Undirected edges will be stored twice – (v1,v2) and (v2,v1) - so that the order of the vertices doesn't matter when querying. This is not identical with storing undirected edges as two directed edges. We know from the graph an edge belongs to, whether or not the edges are directed or not. Mixed edges are not permitted in our graphs. Among other things, this approach prevents having to use a UNION statement when querying for the neighbours of a vertex in undirected graphs.

```
CREATE TABLE edges
(
  graph_id integer NOT NULL,
  from_id integer NOT NULL,
  to_id integer NOT NULL,
  weight double precision,
  CONSTRAINT edges_pkey PRIMARY KEY (graph_id, from_id, to_id)
)
WITHOUT OIDS;
ALTER TABLE edges OWNER TO weknowit;
```

A.3.1Table vertex_data_meta

The table vertex_data_meta is used to store information about which analysis results have been stored in the table vertices. For all stored analysis results the date of the last update is stored, too. This value can be used to decide whether to use the stored analysis results or recalculate them.

```
CREATE TABLE vertex_data_meta
(
  graph_id integer NOT NULL,
  data_type character(40) NOT NULL,
  last_update timestamp without time zone,
  CONSTRAINT vertex_data_meta_pkey PRIMARY KEY (graph_id, data_type)
)
WITHOUT OIDS;
ALTER TABLE vertex_data_meta OWNER TO weknowit;
```

A.3.2Table distances

The calculation of the geodesic distances between all pairs of vertices is complex and expensive. Therefore, the results can be stored in this table.

```
CREATE TABLE distances
(
  graph_id integer NOT NULL,
  source_id integer NOT NULL,
  target_id integer NOT NULL,
  distance double precision NOT NULL,
  CONSTRAINT distances_pkey PRIMARY KEY (graph_id, source_id, target_id)
)
WITHOUT OIDS;
ALTER TABLE distances OWNER TO weknowit;
```

A.3.3Table maprelationaldb

In order to achieve flexibility, data sources are not hard-coded in the source code but will be read from a database. The table maprelationaldb maps identifiers to connections.

```
CREATE TABLE maprelationaldb
(
  identifier character(255) NOT NULL,
  "table" character varying(100),
  connectionid bigint NOT NULL,
  CONSTRAINT pk_maprelationaldb PRIMARY KEY (identifier)
)
WITH OIDS;
ALTER TABLE maprelationaldb OWNER TO weknowit;
```

A.3.1Table relationaldbconnection

The table relationaldbconnection stores all information required to create connections to the available database servers. These connections are referenced in the table maprelationaldb above.

```
CREATE TABLE relationaldbconnection
(
  id bigint NOT NULL,
  server character varying(255) NOT NULL,
  dbtype character varying(20) NOT NULL,
  db character varying(100) NOT NULL,
  username character varying(255),
  "password" character varying(255),
  ssl boolean NOT NULL,
  CONSTRAINT pk_relationaldbconnection PRIMARY KEY (id)
)
WITH OIDS;
ALTER TABLE relationaldbconnection OWNER TO weknowit;
```