Learning Group Extensions for diaspora*

Steffen Brauer

Project 2 Report

Steffen Brauer

Project 2 Report

Submitted at: 31. March 2015

Contents

1 Introduction

In the last years, qualification trends in society have shifted responsibility for learning and education more and more into the hands of individuals. The new technologies and the open Internet have granted easy ways to access content, communication, and collaboration in learning. Online Social Networks (OSN) play an important role in this context. They stimulate their users to socialize with friends and communicate to each other. Discussions in groups are user-triggered and do not need a moderator or facilitator. Our work tries to open the learning process and the building of learning groups to become part of social Internet eco system.

This report covers the extension of the open source social network diaspora* with an eLearning environment, including a content network and a group formation component. In previous work, we characterized our approach of an eLearning environment as an Online Social Network (OSN), which is extended with eLearning features [1]. Such an eLearning-enabled OSN allows users to self-pace learning on topics of personal interest and teams of personal choice. To follow the non-hierarchical paradigm of the social web, we remove any kind of instructor from our platform. This removal leads to questions in the design of such an eLearning system:

1. How to stimulate a team building process that is effective for learners?
2. How to provide access to the relevant content for a learning group?
3. How to facilitate a consistent learning progress, include feedback and corrective actions?

An approach to answer the first question is given in [2]. Here we introduce an algorithm to form groups from the social graph based on learning style, knowledge and intergroup structure. Preliminary work on the second question is based on the Learning Content Management System (LCMS) Hypermedia Learning Object System (hylOs) [3]. hylOs is an adaptive eLearning content management system and runtime environment, built upon an information object meta-model [4] tailored from the IEEE LOM (Learning Object Metadata) standard. The work of With[5] focus on the reasoner of the hylOs system and aims to integrate the content-network in our eLearning environment. To combine the social graph required for team building and the content-graph, we develop the concept of an eLearning-enabled OSN [6]. We analysed the concept of Personal Learning Networks (PLN) that describe how Internet users organize their learning contacts and map this concept to a formal graph structure consisting of learning relevant entities as vertices and semantic relations as edges.

Based on this conceptional background, we report on the implementation of two components of our eLearning-enabled OSN. We take advantage of the open source social network diaspora* that provides the typical features of an OSN and can be easily extended. We first implement the group learning component that covers all features, which are needed to enable the communication within and management of the learning groups. The second component adds a graph representation to the learning network. Learning entities are represented by vertices and edges correspond to different relations between them. This graph implements the concept of an eLearning-enabled OSN by combining the social and content network and provides an analytical data source for our team building algorithm. The remainder of this report is structured as follows: In the next section, we introduce the open source social network diaspora*. Sections 3 and 4 outline the concepts and implementations of the two components introduced above. A brief evaluation of the components is given in section 5. The last section summarises the achievements and presents the next steps in the development of our learning network.

2 diaspora*

diaspora* [1] is an open source implementation of a distributed Online Social Network. It is written using the Ruby on Rails Framework[2], which applies a 3-tier architecture to the application including recent Web technologies. It is also easily extendible using rails engines, that are loosely interconnected to the main application, but have full access to its code. This enables our extensions to reuse some parts of diaspora* and to deeply integrate it into the existing system, but the code is clearly separated. This is an advantage of extending diaspora* in contrary to using an API of a large commercial social network with limited data access and little possibilities of integration.

The social graph of diaspora* is distributed over a network of independent federated servers—so called pods—that are administrated by individual so called podmins [7]. When joining the network, a user has to choose a pod. The personal data are only stored within the database of the chosen pod. There is no centralized indexing instance, which has access to the whole social graph and personal data of all users. To find users in diaspora* an ID is assigned to each user during the registration process. The scheme

[1]https://joindiaspora.com/
[2]http://rubyonrails.org/

of the ID follows the WebFinger protocol, defined in RFC 7033 [8]. It includes the username and the pod separated by a @ (e.g. bob@bob.diaspora.example.com). The WebFinger protocol

> "is used to discover information about people or other entities on the Internet that are identified by a URI using standard Hypertext Transfer Protocol (HTTP) methods over a secure transport. A WebFinger resource returns a JavaScript Object Notation (JSON) object describing the entity that is queried. The JSON object is referred to as the JSON Resource Descriptor (JRD)." [8]

The communication between users, who are not on the same pod is handled via encrypted messages exchanged between the pods. The encryption is handled by asymmetric encryption using a key pair that is generated and assigned to the user during registration. The exchanged messages follow the Salmon protocol[3] that was designed to migrate social media data between different sources and also provides its own signature handling. When a user signs a post as public, the post is sent to all people sharing with the user, even if she is not sharing with them. This update process is handled by the Active Stream protocol[4] and a PubSubHubbub Server[5]. The following features are supported by the diaspora* federation:

- User A initiates a relation to user B,
- User A wants to cancel the relation to user B,
- User A posts something,
- User A comments on a post,
- User A marks a post with a "like",
- User A sends a private message to user B,
- User A requests profile information of user B,
- User A retracts a post,
- User A retracts a like or comment.

[3]http://www.salmon-protocol.org
[4]http://activitystrea.ms/
[5]https://code.google.com/p/pubsubhubbub/

While we aim to apply the federation between the pods to the group learning component, we implement the learning network as a central social network in the first version and concentrate on the key features of the learning components.

2.1 Features

diaspora* supports the core features of a typical OSN like sharing content with others, private messaging and user discovery. In the following, we want to introduce the features, that distinguish diaspora* from other OSNs.

Contacts A user can start *sharing* with another user. The semantic of this relation is that the posts of user A are delivered to user B. This relation is directed and needs not to be acknowledged by user B. But user B receives a notification when user A started sharing and can also start sharing with user A. With these directed edges, the social graph of diaspora* can be described as a directed push network between users. The persons user A is sharing with are called *contacts*. The sharing relation is not restricted to a pod, but can be established between all registered diaspora* users by using the WebFinger protocol. An in-depth discussion of how users connect in diaspora* can be found in section 4.

Aspects When user A starts sharing with user B, A has to attribute an *aspect* to B. This feature is designed to support a context-dependent publication behaviour. In different contexts like work, family, or friends, a person can act differently according to appropriate norms and accepted conventions. These so called facets [9] or foci [10] describe different social aspects in the life of a person and provide the theoretical background of selective sharing features. Besides the selective sharing of posts, diaspora*s aspect feature implements a selective reception of posts generated by the contacts in each aspect.

This grouping of contacts is an important feature in an eLearning context, because a user categorizes her contacts according to learning relevant attributes. While we aim to add a group feature to diaspora* we are interested in the impact of groups in the social graph and how these communities distinguish from groupings created by a selective sharing feature. In [11] we analysed the selective sharing feature of Google+. Here the groupings of contacts are called circles. While in traditional communities users join on their own will, the circles are created by a peer user from her own ego-network. We tried to explore the effects of these different building mechanisms on the social networks and on the processes of selective sharing therein. Based on a Google+ data set [12] with shared circles, we characterized the structures of circles that are embedded in the Google+ social graph. We started with the classification of the used data set by comparing it characteristics to other Google+ data sets. Comparing two social networks of circular structures with two data sets that are built from traditional communities, we could show that i) circles form pronounced community-like structures in Google+, and ii) circles attain an individual structural signature. In particular, circles are significantly less separated from the remaining network than classical communities. Selective sharing in Google+ is thus more diffuse and less confined.

Tags During the registration process, a new diaspora* user can select different *tags* to follow. The followed tags have individual streams, a collection of posts, that contain all posts visible to the user that are marked with the tag.

In social networks, tags are used to mark posts and other content with keywords to describe its context [13]. Marked with a '#', tags are interpreted by the network and posts and content marked with the same tag can be found by clicking on the tag.

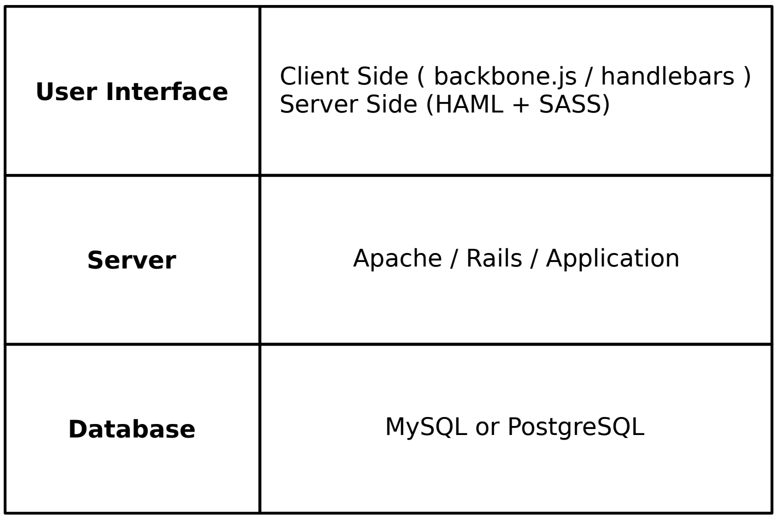
The scope of tags in diaspora* is limited by the visibility of posts. When a post is not visible to the user, because she does not share with the author or the post is not public, the tag connection is hidden. Another limitation is that only posts can be found, which are stored in the local pod. Posts are stored on the senders and recipients pods, which leads to an imbalance of public information on different pods. This is a problem of the current diaspora* implementation, which could lead to a development against the distributed nature towards to only few large pods.

2.2 Pod Architecture

Diaspora* consists of a network of distributed pods. All pods are equal peers in the network and use the same source code. The architecture of a pod follows a classic 3-tier architecture: Database, Server and User Interface (Figure 1).

Database The database stores the models created in diaspora*. While all migrations are handled by the Ruby on Rails framework, the database needs less configuration and maintenance. Possible implementations are MySQL[6] or PostgreSQL[7].

[6]http://www.mysql.com/
[7]http://www.postgresql.org/



User Interface	Client Side (backbone.js / handlebars) Server Side (HAML + SASS)
Server	Apache / Rails / Application
Database	MySQL or PostgreSQL

Figure 1: diaspora* Architecture Overview

Server The Server handles the incoming HTTP requests and passes them to the through the Rails framework to the application code. Requests can come from users or other pods. The incoming requests are mapped to a corresponding controller following the REST paradigm [14]. The inner of diaspora* is designed according to the Model View Controller Pattern. Here the model represents an entity and handles the storage in the database. A view generates a presentation of a model using predefined templates. The manipulation of models and rendering of views is handled by the controller. Besides the Web interface the server component of diaspora* handles the messages exchanged between the pods using the Salmon protocol.

User Interface The User Interface (UI) is implemented on the server and the client side. Static content is rendered at the server and sent as HTML code. Posts and messages are queried as JSON Objects by javascript using backbone.js [8] and handlebars [9]. Both approaches are not clearly separated from each other.

3 Learning Groups

The learning groups are in the center of the eLearning extensions for diaspora*. We aim to create a learning environment that enables users to learn on topics of personal interest

[8]http://backbonejs.org/
[9]http://handlebarsjs.com/

without any instructor [1]. Our concept is based on Personal Learning Environments. A PLE consists of a set of several web services, which can be used to consume content from social web services like Twitter, Blogs or discussion forums. Besides the passive content consumption, communication services take an important role. In contrary to Learning Management Systems (LMS) Personal Learning Environments focus on the user and the personal characteristics and commit the control to the user [15]. A new term in the area of PLEs are Personal Learning Networks (PLN). They describe the social relationships, which are created during interaction through the PLE. Couros [16] uses the definition that,"personal learning networks are the sum of all social capital and connections that result in the development and facilitation of a personal learning environment". Warlick [17] stresses the importance of aggregation, to see what is going on inside the PLN, but he although mentioned the importance of a variety of information sources. The objective of our platform is to provide an environment for learners to maintain their Personal Learning Network (PLN) combined with formal open learning groups. Learners browse the network, initiate connections to others with the same interests and can start groups to work on collaborative task. These tasks, called topics, are created by other learners and cover any kind of activity like creating a piece of content or exchange their knowledge in discussions. To describe the context of a task, the creator has to assign tags to it that allow its categorization and make it searchable.

3.1 Concept

A learning group can be open or closed. If the group is open, other users can join in the active state at any time and the group can be found by all other users of the learning network. If the group is closed, other users have to be invited to join. Only members are able to see the group in the system. During the learning process, members can join a group according to its visibility or leave it. The group members define, when the goal of collaboration is reached by voting for it. The final result should be any kind of artefact. A learning group can have three different states (see Figure 2): The group does not exist, or it is active, or it is frozen. When a user creates a group, a name and a description need to be assigned. In addition, the creator has to decide, whether the group is open or closed. Now the state of the group changes from *no group* to *active*. In the active state, content can be shared within the group and members can discuss about the topic. It is possible that new members can join according to the visibility of the group. To terminate the active state, all members have to agree that the task of the topic is completed. In this case, the group freezes and changes to the *frozen* state.

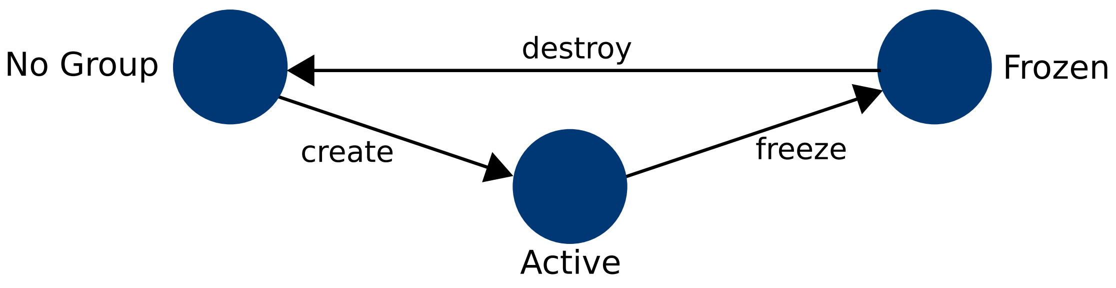


Figure 2: Life cycle of a learning group

Now it is not longer possible to join or post something in the group. But the group is still visible to its members and can be used as an archive. Members of a frozen group can request to delete it. When all members agree, the group is deleted and the state is again *no group*.

3.2 Specification

Based on the concept of the learning groups, we develop a specification, which describes the features that have to be added to diaspora* to enable a group learning experience. Besides the learner, *topic* and *group* are the main entities. A topic consists of a title, a description (including a task), a goal and several tags, which characterize the context of it. It is possible to create a group on a topic or select it in the group creation process. A group has a name, a description, a topic the members are working on and a state. These two entities are created and modified by the learners as follows:

1. A user, who is logged in, can create a topic.
2. A user, who is logged in, can create a group. The user has also to select whether the group is open or closed.
3. At any time in the active state, new members can be invited to join by other members. A text can be added to the invitation. The invited user becomes a member, when she accepts it.
4. Each member of a learning group can create a post in the group. All other group members receive a notification about it.
5. Posts from a group can be reshared as normal posts in diaspora*.

6. Group members do not need to share with each other. If a member of a group wants to share with the other group members, she can create an aspect consisting of the group members.
7. When all group members tag the topic goal as reached, the group is closed and other users are no longer able to join.
8. Recent posts of the groups the current user is a member of are aggregated to a separated stream.
9. A Member of a frozen group can request to delete it. The other members have to acknowledge the request.

3.3 Implementation

To extend diaspora* with learning groups, we chose to create a rails engine, so there is a clear separation of diaspora* and our learning extension. Rails engines have a own MVC structure and run in the same web server environment as the original application.

We aim to reduce the changes in the diaspora* code, to simplify the installation and update process. We achieve this requirement by adding hooks in the loading routine of the engine, which add the migration and locales paths to the main application. In addition, we encapsulate the learning related behaviour of the user model in a learner module. It includes the associations to the group entities and some helper methods. To enter the learning engine, there is a learning sidebar, that has to be included in the `main_stream` file of diaspora*, to make the features of the engine visible to the user.

Another objective was to adopt the existing style of diaspora*. We reuse the standard layout and the existing style sheets.

The data model of the group learning engine shown in Figure 3 contains the entities we need to implement the group management. All entities are added to the database except for the Learner entity, which extends the User model of diaspora*. The migrations of the engine are run in the diaspora* environment and stored in the same database. Tagging is implemented using the ActAsTaggableOn [10] Gem. It handles the storage and parsing of the tags in posts and enables to store taggings on different models.

In Figure 4, we show the models, views and controllers implemented for the learning group management. The entities from the data model can be found with corresponding models, views and controllers. In addition, there are two more categories of model:

[10]https://github.com/mbleigh/acts-as-taggable-on

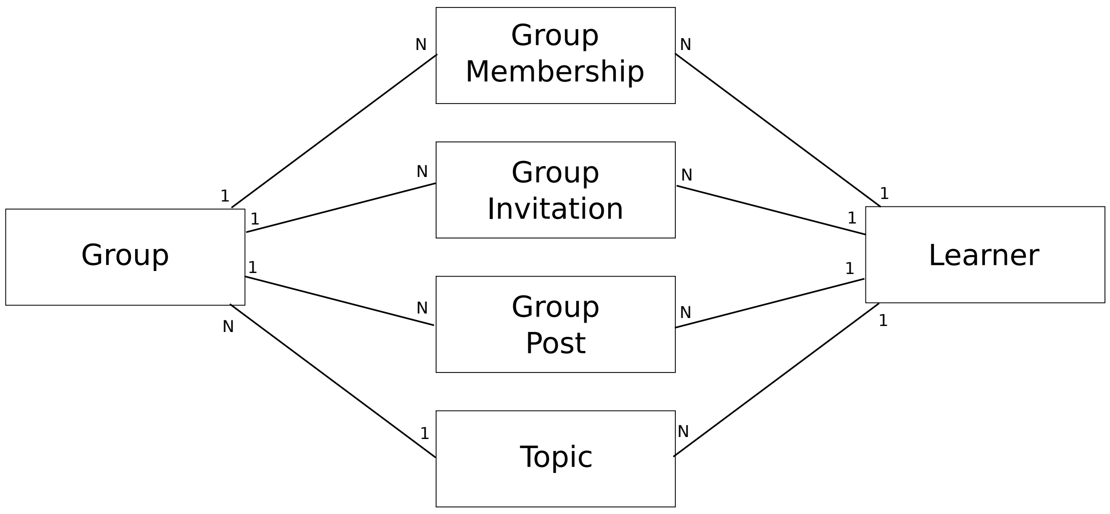


Figure 3: Data model of the group learning engine

notifications and observers that stand for several models, which handle the notifications for new group posts or invitation. The observers are described in detail in section 4.2. Besides the additional models, there are two controllers with views. The ReshareGroup-PostsController implements the feature of converting a group post to a regular one and aggregation of information from the group learning component is handled by the LearningController.

While users can post content in the learning groups, we aim to integrate this posting in the diaspora* implementation. Advantages of a deep integration are the reuse of features and code of diaspora* and a small difference in the usability of the social network and the eLearning system. The post model is tightly bounded to streams, aspects and the visibility management. Posts can be public either or belong to an aspect, which does not map well to group posts. Here the members do not have to share with each other. One possibility in existing systems is that all group posts are public posts in the social network. But this contradicts the closed group concept. So it is necessary to implement group specific posts that are bound to groups, but can be reshared by members to make them available in the network. diaspora* has a notification system that, for instance, informs the user when a new message is received or a post is liked. We integrate notifications for an invitation to a group and new group posts. Even if the notification system is implemented very generically using different classes to represent the notifications, the method which creates the notifications uses a switch case statement that limits the possibility of new notifications. The statement distinguishes between the

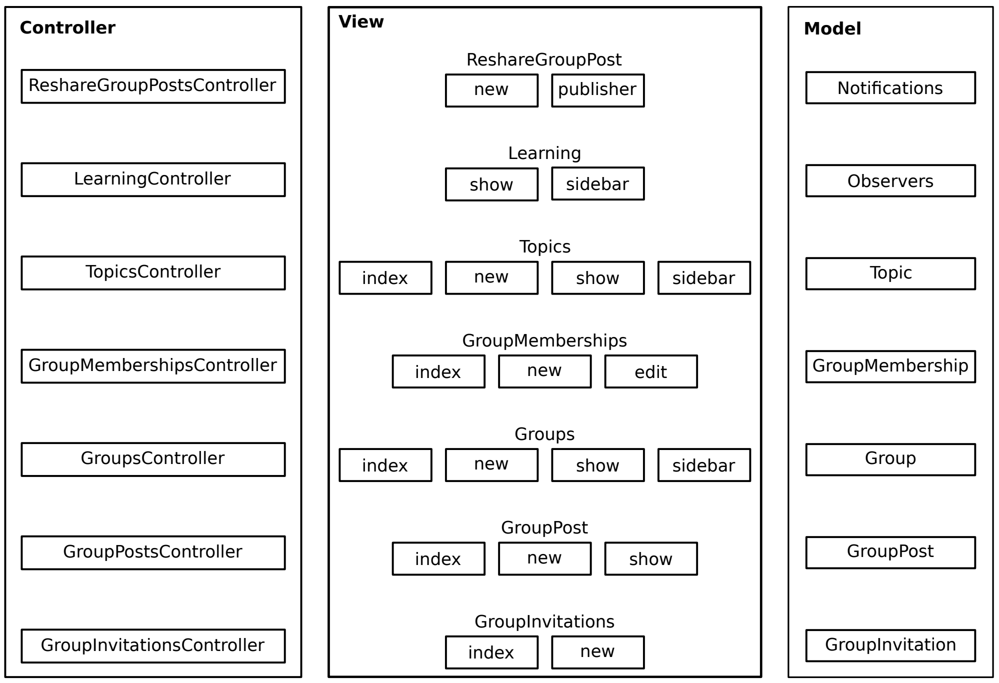


Figure 4: Overview of the Models, Views and Controllers implemented.

targets of the notification (e.g. Comment, Like or Reshare), to select the object, which should be linked to and calls model specific methods. We can avoid this restriction by overwriting the `make_notification` method in the new notifications.

4 Analytical Graph Database

Diaspora* uses a MySql or PostgreSQL database to store its models. The scheme used here is normalized and optimized for read and insert operations, but lacks usability and performance by analytical queries. The term analytical database derives from the field of Business Intelligence. Here an analytical database has an optimized scheme for querying historical aggregated business data to support the controlling. While analytical database often has a multidimensional scheme, we choose a graph scheme, so we can easily map our eLearning-enabled OSN concept to the database and are able to traverse the graph without large join tables by just following the edges. To query the graph database, we do not use SQL, which is not suitable for graph related task. Instead, we use the Gremlin[11] graph traversal language. To enable an easy-to-use and fast analysis

[11] https://github.com/tinkerpop/gremlin/wiki

of the social network in diaspora* and our extensions we set up an analytical graph database. Its task is to hold a copy of the entities of the eLearning-enabled OSN graph structure to analyse the social graph and serve as a data source for recommendation algorithms.

Besides the direct connections of learners via edges from one to another, the learners are connected via indirect connections, for instance by editing the same content or participating in discussions. These indirect connections bear a high potential for learner supporting features. The direct neighbours of a learner describe her context in the network. This context extents the profile of the leaner exclusive information. An extended view of the context of a learner is her ego-network. The ego network of a vertex in a social network is the sub-graph including all neighbours and the edges between them [12]. This surrounding of a learner in the network have a high potential for discovering new learning partners and interesting content, which can be discovered by simply browsing the network.

4.1 Formal Model

The formal model of our eLearning-enabled OSN is introduced in [6]. It was developed from an analytical point of view on environments and social networks Internet users maintain in a learning context. While these networks are distributed over several online services, they provide a rich set of information how learning takes place in a social media environments but are hard to analyse. To enable an analysis of these Personal Learning Networks, we map our eLearning OSN to a formal graph structure. This graph is modelled by a directed graph with vertices of different types (see Figure 5). These types cover all relation kinds between content objects, user profiles, groups and topics. The (technical) links are typed accordingly. This unified approach (cf., [18]) adds many implicit relations to the network. In this way, it is possible to find users, who have vertices relations in common, but no personal interconnect. It also enables algorithms to measure the strength in connectivity of two vertices by accounting for shared neighbours or distinct paths that connect them.

To represent the actual users of the platform, they hold a profile with common OSN attributes. As the learner vertices represent the active roles in the network, they create or edit other vertices and connect to them. Besides learners, content is mapped to vertices. Content vertices can refer to resources outside the platform or content objects in the content network. While learners and content form edges by editing or consuming relations, their contextual relations are mapped indirect using tag vertices. Topic vertices

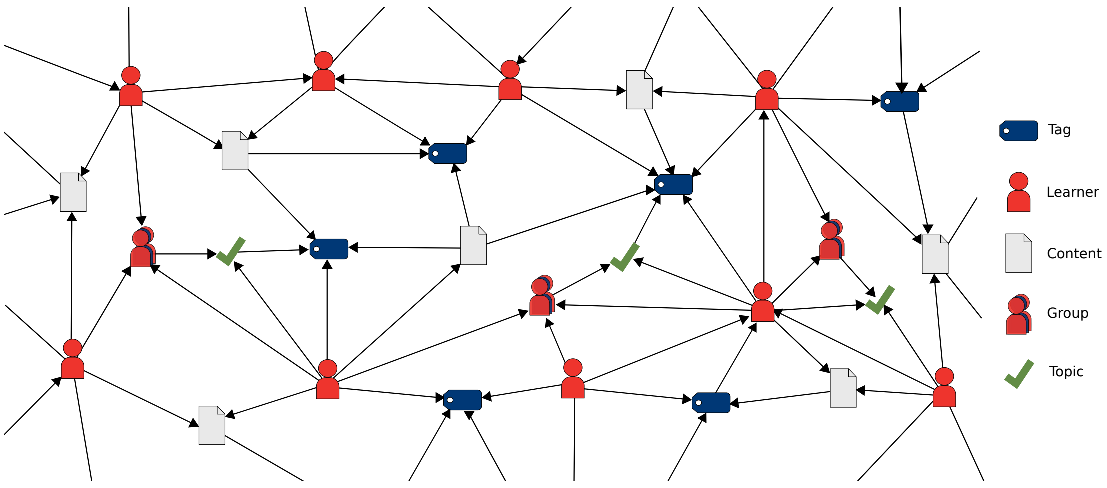


Figure 5: eLearning-enabled OSN with multiple vertices types connected by directed edges

can be chosen for collaborative group learning. If a group is successfully created, a group vertex is created and all members connect themselves to it. To distinguish the relevance of different edges in the network, we introduce weights. The calculation of the weights is determined by the context of the connected vertices. A in detail list of the different weights can be found in section 4.2.

4.2 Graph Database Engine

We chose the Rexster graph server for a graph database engine. It is part of the TinkerPop2 [12] graph computing framework and provides a REST API to different graph database implementations via the Blueprint interface.

The existing ruby client for Rexster does not fulfill our requirements, because it only returns the results of the queries as maps and does not maintain a graph structure. Therefore, we create our own client, which is able to query and create vertices and edges via the REST interface by representing the database entries by vertex and edge objects.

From the analytical perspective, we are interested in the graph structure and neglect other attributes of the entities. To map the diaspora* learning entities to vertices and edges in the graph, we have to ensure, that the identifier in both systems is unique. diaspora* creates automatically a primary key on each table, this leads to a uniqueness of the object in each table. While we map different models to the graph database, the

[12]http://www.tinkerpop.com/

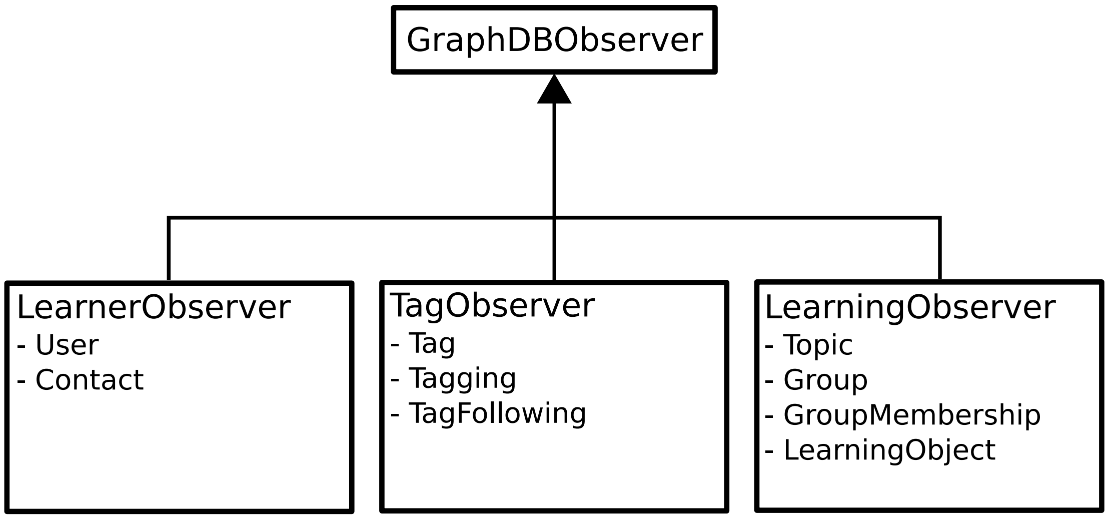


Figure 6: Inheritance structure and mapping of the observer

identification only by id is not possible. Therefore, we generate a new identifier, that consists of the entity name and the key concatenated with a '_'. Following this scheme the group with the id 1 has the identifier `group_1`.

To map the learning objects from the diaspora* database to Rexster, we implemented observers, that are executed whenever a create or update action happens on a learning model. The observer checks, if the vertex or edge exists and updates or creates it. We designed the observers based on their concerns and coupling. Figure 6 shows the Inheritance structure of the implemented observers and the diaspora* models they observe. The database connections and methods for update or create actions are handled by the `GraphDBObserver`. While the objects of the group learning component can be handled by one observer (`LearningObserver`), the observers for tags and learners are much more complex. This results in a separate `LearnerObserver` and `TagObserver`. The mapping of vertex type to the class in diaspora* is shown in table 4.2.

Vertex type	Diaspora class
Learner	`User`
Tag	`ActsAsTaggableOn::Tag`
Group	`GroupLearning::Group`
Topic	`GroupLearning::Topic`
Content	`ContentRepo::LearningObject`

Table 1: Mapping of vertex types to diaspora classes

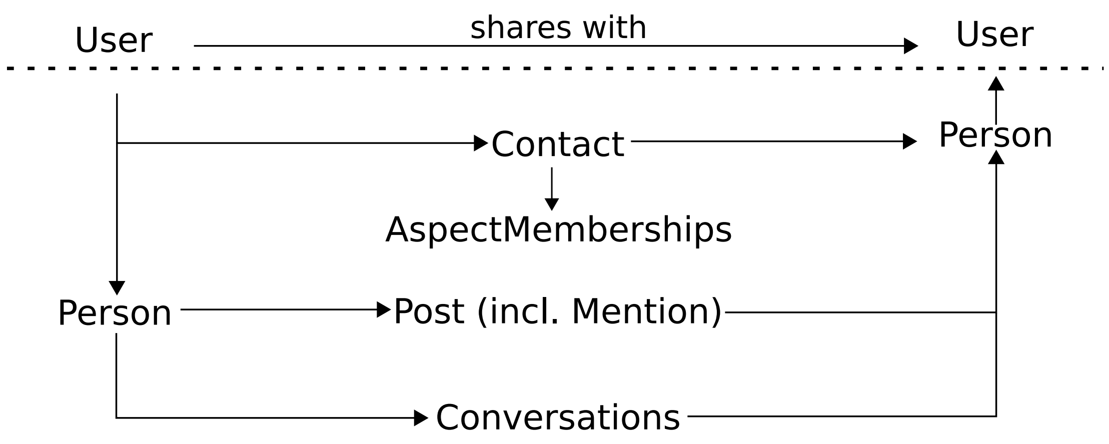


Figure 7: diaspora* entities included in a relation between two user

Edges in the graph database refer to associations in the learning engine. The identification is done by its in- and out-going vertices. The mapping is also triggered by the observers. While the creation is comparable to the vertices, the weights at the edges have to be updated. In the following we state the specific requirements for each existing edge type in the eLearning-enabled OSN [6]. The weights on the edges to and from the content vertices are defined and implemented in the work of With.

Learner $\rightarrow$ Learner We define the weight of the edge from learner v to u by the communication objects send from v to u $M_{v \rightarrow u}$. In diaspora*, an edge between v and u represents that v *shares* with u. This means, that content created by v is pushed to u according to the membership of u in the aspects of v. This perspective on a relation between users is different from the common interpretation of social relations, where users *follow* each other (u subscribes to the content of v).

An overview of the entities, which represent the sharing relation on the implementation level, is shown in Figure 7. In general, a relation is established to a user's `Person` object. The `Person` object is used to represent a user independent from the pod in the communication features. The relation between two diasproa* users is maintained via the `Contact` model. It is created when a user starts sharing with another user. The creation of edges between learners in the graph database is triggered by it. The `Contact` model also holds the memberships of u in the aspects of v. `Post` and `Conversations` (private messages) are associated with `Person` objects on the side of v and u. A `Post` in diaspora* has to direct connection to the receiver, this is not the case if it includes a `Mention` of another user.

We implement the weight of the edge as the sum of the `AspectMebership`, `Mention` and `Conversation` objects learner v and u share. `Conversation` objects have authors and participants, because of the directed edges, we only sum the edges from the author to the participants and not from all participants to all other participants. A special case is, that users in diaspora* are able to block other users, if it happens, the edge value is set to -1 and can not be updated.

Learner → Tag To weight the edges between a learner and a tag, we introduced an Activity Index [2]. It represents the relevance of a user per tag. To update or create this edge, the `TagObserver` monitors the `ActAsTaggable::Tagging` model. It is triggered, when a tag is assigned to a taggable model. To track the learner, who assigned the tag, we have to set it explicitly in the controller layer, because the option to track the tagger is not set in diaspora* by default. Every tagging increments the weight of the edge by one.

Learner → Topic The direct relation between a learner and a topic is defined by a creation or manipulation action. Such an action increments the weight by one.

Learner → Group The edges between a learner and a group vertex is weighted by the participation of the leaner in the collaborative work. This is measured by the number of posts in the learning group. The implementation of this edge weight is done by observing the `GroupPost` model. The edge is created after a user has posted something to the group and increment by any ongoing post.

Topic → Tag The weights of tags per topic are assigned by the creator or learner, who edit the topic. Each tagging increments the weight by one.

Group → Topic A group vertex always has one outgoing edge to a topic, so its weight is always 1.

4.3 Viewer

The observers that update the analytical graph are executed by specific events. This even-driven nature of the observers together with a lack of suitable database viewer make the debugging a time consuming task. To overcome this we implement a viewer for graph database using the d3.js JavaScript library[13]. It uses a data driven approach to animate data using HTML, SVG and CSS. The file containing the viewer is in the public folder of the diaspora* implementation. The included JavaScript code queries the REST API of Rexster and loads all edges and vertices of the eLearning graph. The loaded data is visualized using the force layout and colors the vertices according to the model. Besides the key in the graph database, the edge weight is shown. With this viewer it is possible to monitor the graph database and debug the observers.

5 Evaluation

Since this report covers the first building block of our eLearning-enabled OSN, the goal of this evaluation is to show that the implementation of learning groups in diaspora* full fills its specification and the analytical graph database tracks the creation of the network. In addition, we introduce a feedback group, which create a persist way to get feedback from test learners.

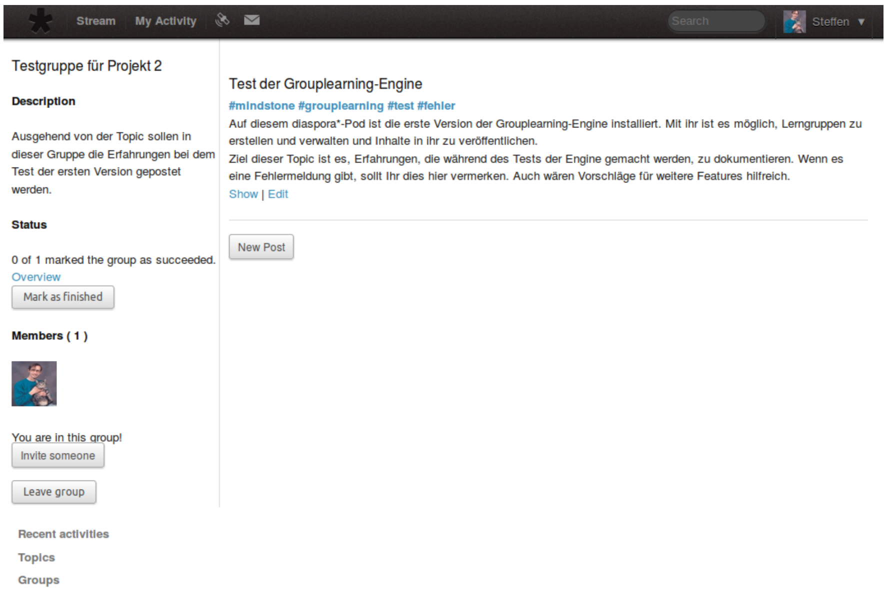
To set up the evaluation environment, we use Ubuntu 14.04.1 on a virtual machine. Following the guide in the diaspora* wiki[14]. We install diaspora* 0.4.2.1 with a MySQL database configuration. To store the analytical graph, we use Rexster in version 2.6.0 and create an empty graph. Then we integrate the group learning engine by mounting it in diaspora* and add the learning sidebar to the main view. To store the models created by the engine, we run the migrations to create the corresponding tables in the diaspora* database.

5.1 Feedback Group

We create a feedback group to establish a feedback channel, which persist from the start of this evaluation till the other two building blocks (content network and group formation) are implemented in integrated into the test environment. Its topic is to discuss about the concept and implementation of the group learning feature and our platform in general.

This group establishes a fast feedback loop for the test users. During this first evaluation, the feedback group helped to find bugs, which could be fixed immediately and also was used to discuss the usability and possible next features.

[13]http://d3js.org/
[14]https://wiki.diasporafoundation.org/Installation/Ubuntu/Trusty



Testgruppe für Projekt 2

Description

Ausgehend von der Topic sollen in dieser Gruppe die Erfahrungen bei dem Test der ersten Version gepostet werden.

Status

0 of 1 marked the group as succeeded.
Overview
Mark as finished

Members (1)

You are in this group!
Invite someone
Leave group

Recent activities
Topics
Groups

Test der Grouplearning-Engine
#mindstone #grouplearning #test #fehler
Auf diesem diaspora*-Pod ist die erste Version der Grouplearning-Engine installiert. Mit ihr ist es möglich, Lerngruppen zu erstellen und verwalten und Inhalte in ihr zu veröffentlichen.
Ziel dieser Topic ist es, Erfahrungen, die während des Tests der Engine gemacht werden, zu dokumentieren. Wenn es eine Fehlermeldung gibt, sollt Ihr dies hier vermerken. Auch wären Vorschläge für weitere Features hilfreich.
Show | Edit

New Post

Figure 8: Screenshot of the feedback group

5.2 Group Management

Figure 8 displays a screenshot of the feedback group. On the left side is a sidebar containing the group name and description. Below that is the current status of the group. Here the group members can mark when the group goal is reached. The sidebar also contains a list of the group members represented by their profile pictures. When a user is a group member, the sidebar states it and contains two buttons for inviting other contacts or leaving the group. When the current user is not a member of the group, the sidebar contains a join button. The bottom of the sidebar always contains links to the recent activities, topics, groups and content pages. The topic the groups is working on is shown on the right. Besides the name and description, the tags of the topic are listed. Below the topic are the group posts visualized. At the beginning of the feedback group, no post was created. In the following we want to demonstrate the other implemented features for the group management.

Group Goal Each member of the group has to mark the group goal as reached to change the state from active to frozen, as we state in section 3.1. An overview of the current status is given in the sidebar of a group. There is also a button that shows a detail view of the markings. A screen shot of this view can be found in Figure 9.

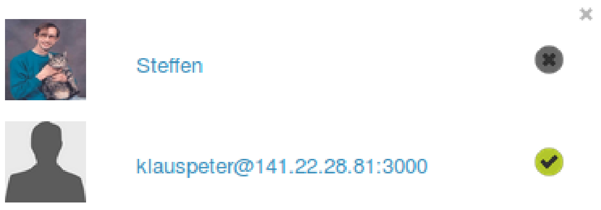
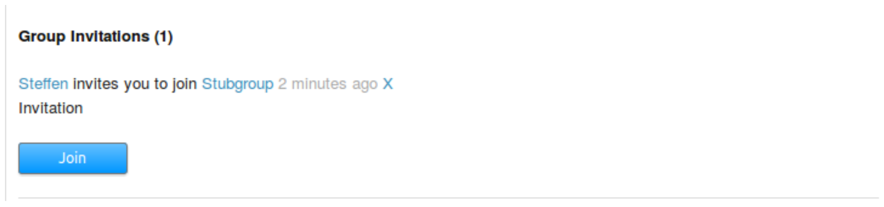


Figure 9: Screenshot of the group opinion whether the goal is reached


Group Invitations (1)

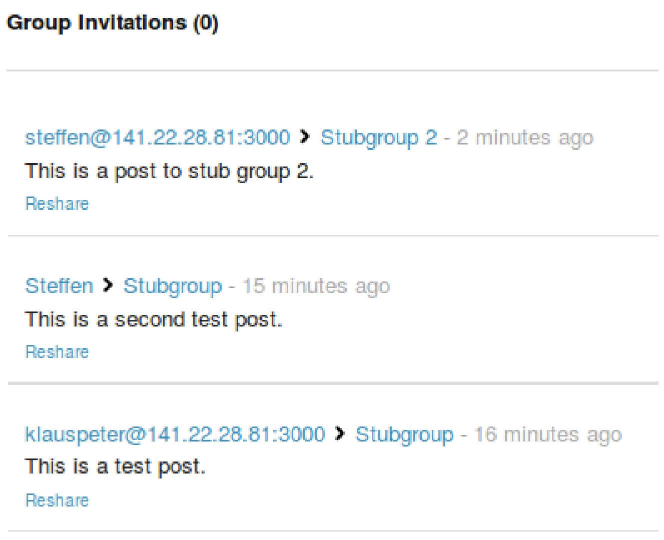
Steffen invites you to join Stubgroup 2 minutes ago X
Invitation

Figure 10: Screenshot of the group invitation

Group Invitations A group in our eLearning system can be open or closed. If the group is open any user can join it. But if it is closed a new member have to be invited out of the contacts of present group members. Figure 10 shows a screenshot of partial view that is included in the group overview and on the recent activity page. If there are no group invitations for the user, the view indicates zero invitations, but if the user receives invitations, a list of the invitations can be expanded by clicking on the number of open invitations.

Recent Activities In section 3.2, we specified a stream of group posts that are created in the groups of the current user. Figure 11 shows this stream. Each post contains the user, who made it, the group it is posted to and time span since its creation as well as the content of it. Here we also see that there are currently no group invitations.

Resharing of Group Posts Because we were not able to integrate the group posts in diaspora*, we decided to implement a resharing feature that is able to convert group posts to default diaspora* posts. We keep the group member who created the post and the name of the group and links to them (see Figure 12). An open question is, how the visibility of the reshared posts is handled. In the current implementation any group



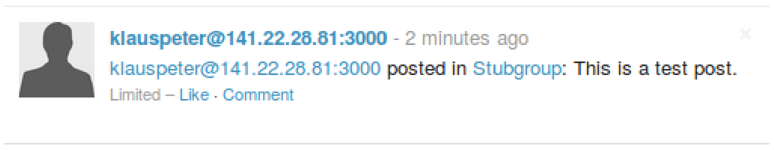
Group Invitations (0)

steffen@141.22.28.81:3000 › Stubgroup 2 - 2 minutes ago
This is a post to stub group 2.
Reshare

Steffen › Stubgroup - 15 minutes ago
This is a second test post.
Reshare

klauspeter@141.22.28.81:3000 › Stubgroup - 16 minutes ago
This is a test post.
Reshare

Figure 11: Screenshot of the recent activities in the learning groups the current user is part of



klauspeter@141.22.28.81:3000 - 2 minutes ago
klauspeter@141.22.28.81:3000 posted in Stubgroup: This is a test post.
Limited – Like · Comment

Figure 12: Screenshot of group post that was reshared to a default diaspora* post

member can reshare any post in open as well as closed groups. This contradicts the privacy model of diaspora*.

5.3 Analytical Database

To evaluate the analytical database, we use the viewer introduced in section 4.3. Figure 13 shows the graph after the creation of the feedback group. It displays that the mapping from our concept of an eLearning-enabled OSN to the graph database full fills its specification. The only registered user is `user_4`, who follows `tag_2`. This user than created the evaluation topic `topic_2` and assign the tags `tag_3`, `tag_4`, `tag_5` and `tag_6`. The weight of the edges between the `user_4` and the tags are incremented by 1. There is also a `group_1` that is created by `user_4` and works on `topic_2`.

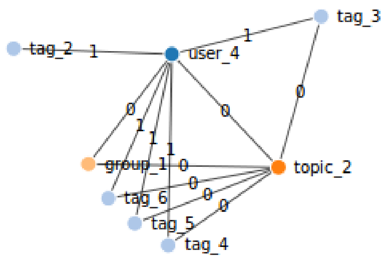


Figure 13: Analytical graph after the creation of the feedback group

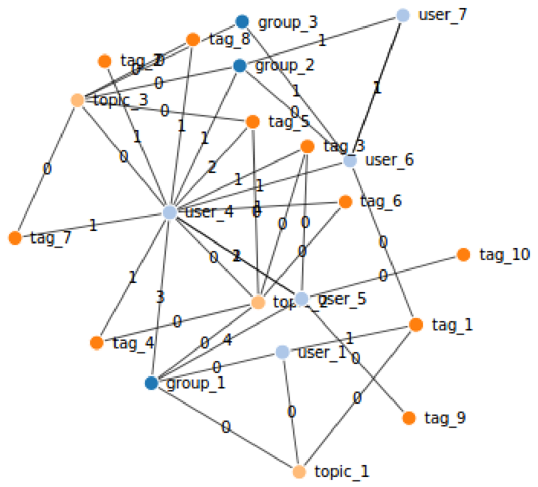


Figure 14: Analytical graph after the creation of the feedback group

Figure 14 shows the graph of the analytical database after the test of all specified features. The main test user was `user_4`. Its vertex has the highest degree and shows an emphasized role in graph. Even if the graph is very small, the visualization makes it hard to get the details of the specific edges. A clear visualization is one of our next steps do support the debugging of our recommendation algorithms. In the current version of the viewer, all vertices and edges are selected from the database each time, the site is loaded. This works fine for this evaluation with just a few vertices and edges, but will be not possible for bigger graphs. So our objective is to implement a fast and easy way to select the relevant parts of the network and visualize it with a high amount of information.

6 Conclusion and Outlook

In this paper, we reported about the first steps in the implementation of our eLearning-enabled Online Social Network. The network uses the open source OSN diaspora* to serve the main social network features. Diaspora* overcomes some of the privacy issues, commercial OSN have by splitting the private data to several pods. We extend the network with a group learning component and an analytical database. The group learning component adds formal group learning to diaspora*. It includes the management of learning topics, posting content to groups and invitation of members. The analytical database stores the eLearning relevant data in a graph database and make it easy accessible. In the evaluation, we show that the developed components fulfills their requirements.

In our future work we will improve the two components by adding more features and maintain the code. The mapping of the edges to and from the content objects was not possible yet. This is the next step, we will focus on. During the current implementation, the functionally was in focus which lead to a lack in usability. Its improvement will be a continuing task in our future work. A important building block in our future is the implementation of the group formation engine that we introduced in [2]. While the evaluation in this report covers only the proof of full filling the specification, we aim to create test learning groups with real users and also perform evaluations on our recommendation algorithm with large test data.

References

[1] H. Roreger and T. C. Schmidt, “Socialize Online Learning: Why we should Integrate Learning Content Management with Online Social Networks,” in *Proc. of IEEE Intern. Conf. on Pervasive Computing and Communication (PerCom), Workshop PerEL*. Piscataway, NJ, USA: IEEE Press, March 2012, pp. 685–690.

[2] S. Brauer and T. C. Schmidt, “Group Formation in eLearning-enabled Online Social Networks,” in *Proc. of the International Conference Interactive Computer aided Learning (ICL’12)*, M. E. Auer, Ed. Piscataway, NJ, USA: IEEE Press, Sep. 2012.

[3] M. Engelhardt, A. Hildebrand, D. Lange, and T. C. Schmidt, “Semantic Overlays in Educational Content Networks – The hylOs Approach,” *Campus-Wide Information Systems*, vol. 23, no. 4, pp. 254–267, September 2006. [Online]. Available: http://www.emeraldinsight.com/10.1108/10650740610704126

[4] B. Feustel, A. Karparti, T. Rack, and T. C. Schmidt, “An Environment for Processing Compound Media Streams,” *Informatica*, vol. 25, no. 2, pp. 201 – 209, July 2001.

[5] N. With, “Modernisierung des hylOs-Reasoners,” HAW Hamburg, Tech. Rep., 2013.

[6] S. Brauer, T. C. Schmidt, and A. Winschu, “Personal Learning Networks with Open Learning Groups - a Formal Approach,” in *Proc. of the International Conference Interactive Computer aided Learning (ICL’13)*, M. E. Auer, Ed. Piscataway, NJ, USA: IEEE Press, Sep. 2013.

[7] A. Bielenberg, L. Helm, A. Gentilucci, D. Stefanescu, and H. Zhang, “The growth of diaspora-a decentralized online social network in the wild,” in *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*. IEEE, 2012, pp. 13–18.

[8] P. Jones, G. Salgueiro, M. Jones, and J. Smarr, “WebFinger,” IETF, RFC 7033, September 2013.

[9] S. D. Farnham and E. F. Churchill, “Faceted identity, faceted lives: social and technical issues with being yourself online,” in *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, ser. CSCW ’11. New York, NY, USA: ACM, 2011, pp. 359–368.

[10] S. L. Feld, "The focused organization of social ties," *American journal of sociology*, pp. 1015–1035, 1981.

[11] S. Brauer and T. C. Schmidt, "Are Circles Communities? A Comparative Analysis of Selective Sharing in Google+," in *Proc. of 34th Int. Conf. Dist. Comp. Systems ICDCS – WS HotPost.* Piscataway, NJ, USA: IEEE Press, June 2014, pp. 8–15.

[12] J. McAuley and J. Leskovec, "Learning to discover social circles in ego networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 548–556.

[13] J. Vassileva, "Toward Social Learning Environments," *Learning Technologies, IEEE Transactions on*, vol. 1, no. 4, pp. 199 –214, oct.-dec. 2008.

[14] R. Fielding, "Representational state transfer," *Architectural Styles and the Design of Netowork-based Software Architecture*, pp. 76–85, 2000.

[15] N. Dabbagh and A. Kitsantas, "Personal Learning Environments, social media, and self-regulated learning: A natural formula for connecting formal and informal learning," *The Internet and Higher Education*, vol. 15, no. 1, pp. 3–8, 2012.

[16] A. Couros, *Developing Personal Learning Networks for Open and Social Learning.* Athabasca University Press, 2010, no. 6, pp. 109–128.

[17] D. Warlick, "Grow Your Personal Learning Network," *Learning & Leading with Technology*, vol. March/April, pp. 12–16, Mar. 2009.

[18] E. Amitay, D. Carmel, N. Har'El, S. Ofek-Koifman, A. Soffer, S. Yogev, and N. Golbandi, "Social search and discovery using a unified approach," in *Proceedings of the 20th ACM conference on Hypertext and hypermedia*, ser. HT '09. New York, NY, USA: ACM, 2009, pp. 199–208.