FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Model-based Reverse Engineering of Social Networks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Andreas Munk
Matrikelnummer 0726826

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: O.Univ.-Prof. Mag. Dipl.-Ing. Dr.techn. Gerti Kappel
Mitwirkung: Mag. Dr. Manuel Wimmer

Wien, 30.11.2011

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Model-based Reverse Engineering of Social Networks

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Andreas Munk
Registration Number 0726826

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: O.Univ.-Prof. Mag. Dipl.-Ing. Dr.techn. Gerti Kappel
Assistance: Mag. Dr. Manuel Wimmer

Vienna, 30.11.2011

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Andreas Munk
Blauensteinerstrasse 19, 3130 Herzogenburg

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Danksagung

Im Laufe meines Studiums gab es sehr viele Höhen, aber auch wenige Tiefen. Die Anspannung vor Prüfungen oder der Stress vor wichtigen Abgaben hatte sicherlich auch Auswirkungen auf jene Menschen, die das Wichtigste in meinem Leben sind. Hiermit möchte ich mich bei meiner Familie bedanken, insbesondere bei meiner Freundin Magdalena! Ihr habt mich während der gesamten Studienzeit und während der Erstellung dieser Masterarbeit stets unterstützt und mich in so manchen schwierigen Tagen wieder aufgebaut.

Weiters gilt mein Dank meiner Hauptbetreuerin, Frau Gerti Kappel sowie meinem Nebenbetreuer Manuel Wimmer. Frau Kappel unterstützte mich durch Gespräche und ihr Feedback bei meiner Masterarbeit. Manuel Wimmer verdanke ich, dass es durch seine Ideen und sein Vertrauen zu dieser Masterarbeit und meinem Beitrag im TheHiddenU Projekt gekommen ist. Außerdem unterstützte er mich stets mit seinem Fachwissen, seinen Vorschlägen und durch seine konstruktive Kritik.

Abstract

Social networks on the Web have seen enormous growth over the past few years, leading to a truly widespread adoption. Every social network is focused on serving specific human needs. Most networkers are present in a number of different networks, which leads to scattered social content. The development of such Web-based platforms is in an early stadium, which result in short feature release cycles. The evolving data schemas and the different ways to access social data are resulting in tedious and error-prone development and maintenance processes of social applications.

In this master thesis, evolution, main characteristics and features of social networks are surveyed. Four platforms, namely Facebook, LinkedIn, Twitter and GooglePlus, are examined. Class diagrams of the data schemas, based on the official documentation, give an overview of these platforms. Each social network has its individualities of accessing the data. A widespread data authorization system is oAuth, available in two versions, which are explained in this master thesis and implemented in a social adaptor for a project called TheHiddenU, to enable an easy access to the social data. To realize this access, the information about the data schema is needed. Because of the incomplete documentation of the API, in this thesis a tool called Json2Ontology is developed for an automatic reverse engineering of the data schema offered by the social networks.

In the first step the tool uses the currently implemented REST Web services. The response of the Web service is transformed into sentences of a domain specific language of TheHiddenU (THUDSL), which represent social user profiles and enable the generation of Java classes for data access. Starting at one or more request URLs, the Json2Ontology tool analyze the Json response and search for navigation possibilities, which are represented as relationships. The goal is to extract as much information about the data structure as possible.

The Json2Ontology tool has been evaluated by comparing the information of the created class diagrams with the generated data schema. The result depends on the authorized user and the amount of personal data. In case of Facebook, a real world test user has been used to find the data produced by real social interactions. The Json2Ontology tool found 79% of the classes and 80% of the attributes. Even more important are the the newly found classes (namely 7) and attributes (namely 162). Different settings have been evaluated to find a well balanced configuration for the tool.

Kurzfassung

Soziale Netzwerke verzeichneten in den letzten Jahren ein enormes Wachstum. Die meisten dieser Netzwerke spezialisieren sich auf eine bestimmte Zielgruppe, was dazu führt, dass die Benutzer in verschiedenen Netzwerken registriert sind und dort, je nach Thema, Informationen über sich preisgeben. Die Entwicklung sozialer Netzwerke befindet sich in einem frühen Stadium und neue Funktionen oder Änderungen am Datenschema werden in kurzen Zyklen durchgeführt. Dieser Umstand und die verschiedenen Methoden um Zugriff auf die Daten zu erhalten, bedeuten große Herausforderungen für Entwickler von sozialen Anwendungen.

In dieser Masterarbeit wird die Evolution von sozialen Netzwerken beschrieben und grundlegende Funktionen und Eigenschaften definiert. Die vier Plattformen Facebook, LinkedIn, Twitter und GooglePlus werden untersucht. Mit Klassendiagrammen, basierend auf der API Dokumentation, soll ein Überblick über die zur Verfügung stehenden Daten geschaffen werden. Jedes soziale Netzwerk hat Eigenheiten, um auf die Daten zugreifen zu können. oAuth, ein weit verbreitetes und von den untersuchten sozialen Netzwerken eingesetztes Autorisierungssystem, wird in dieser Masterarbeit beschrieben. Durch die Implementierung eines Adaptors für das Projekt TheHiddenU, soll der Zugriff auf die sozialen Daten vereinfacht werden. Um dies zu ermöglichen, ist neben der Autorisierung auch die Information über das Datenschema notwendig. Auf Grund der unvollständigen Dokumentierung der API, wird in dieser Masterarbeit ein Ansatz beschrieben, um das Datenschema automatisch zu generieren.

Der erste Schritt in diesem Ansatz ist die beispielhafte Verwendung der von den sozialen Netzwerken eingesetzten REST web services und die Transformation der Json Antwort in eine domänenspezifische Sprache des TheHiddenU Projekts (THUDSL), welche das Profil eines Benutzers von sozialen Netzwerken beschreibt und für die spätere Generierung von Java Klassen für den Zugriff auf soziale Daten verwendet werden kann. Das Transformationssystem beginnt die Analyse bei einem oder mehreren Startpunkten (URLs), analysiert die entsprechenden Antworten und sucht nach Navigationsmöglichkeiten, um Verbindungen herauszufinden und neue Objekte zu erreichen.

Die Evaluierung des Systems wurde mit den untersuchten Plattformen durchgeführt. Mit einem realen Facebook Profil wurde eine Abdeckung von 79% der dokumentierten Klassen und 80% der Attribute erreicht, sowie 7 neue Klassen und 162 Attribute gefunden. Das Ergebnis ist abhängig von der Datenqualität des verwendeten Profils. Unterschiedliche Konfigurationen wurden evaluiert, um eine optimierte Basiskonfiguration des Ansatzes zu finden.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Social networks on the Web have seen enormous growth over the past few years leading to a truly widespread adoption. According to a recent report of Nielsen [4], two-thirds of the world's Internet population visit a social network (including blogging sites), accounting now for almost 10% of all Internet time. They publish personal information and connect themselves with friends, former classmates or other people, even if they do not really know them. They integrate social networks in their daily life. Facebook[1], for example, with more than 740 million users is the biggest social network in the world. 49% of the user in Austria are between 13 and 25 but in the last months, the age groups with the fastest growth are the people from 35 to 44 and seniors (64+). 2,6 million user in Austria and almost 21 million user in Germany[2] show the importance of social networks and the potential for social applications by using the enormous amount of data. This is one of the major reasons to do a master thesis about social networks and make research in how to access the data of the users to help building a social data adaptor.

Every social network is focused on serving specific human needs. Most networkers are present in a number of different networks, which leads to scattered social content. To enable and enhance personalization it is necessary to collect information of the different data sources (social networks) for data mining and profiling to build social software. Social software operate with social data: personal information about the user and interconnections between users which generate network effects.

Social software has big advantages for business. The possibility to interact with customers in social networks led to the situation, that many companies use Facebook, Twitter and other networks for marketing. The APIs of the biggest social networks help web engineers to create more personalized web-based social software systems and benefit from the core data of social

[1] `http://www.facebook.com`
[2] `http://www.checkfacebook.com`

networkers and connections between the users. Social software can cause a snowball effect. With a good concept, developer can reach users who use the software and, supported by the networking features of the social networks, also their friends read about and use the social software. From the perspective of a company there is the possibility to collect important personal information about the customer or the customers opinion via social software. In former times there had to be a strong customer relationship to get such information. One part of creating social software is how to get access to the data. The authorization techniques as well as the different fast moving APIs excite my interests.

In my thesis I will focus on the model based reverse engineering of social networks for further usage like creating recommender systems such as TheHiddenU [16], a project realized by the Department of Telecooperation and Information Systems Group (Johannes Kepler University Linz), Business Informatics Group (Vienna University of Technology) and Netural Communication[3].

1.2 Problem Statement

The enormous amount of personal data in social networks has a high potential for companies. The derived knowledge will open new possibilities to recommend products for users. To collect the maximum amount of private data about the user, a social software like a recommender system must provide the possibility to connect to different social networks, which have different ways to authenticate and retrieve the data. There exist no tools for an easy-access and so companies have to do own implementations and spend time to get in touch with the characteristics of the social network APIs. An automatic model generator, generic adaptor and step-by-step instructions to use and add new social networks will help developers to create social software.

To represent the data structure we need an up-to-date ontology for every social network. The poor documentation of the APIs and the absence of an official schema make it hard for non-developers integrate new social networks for data collection and analysis. Different social networks have different information about the user. A data schema for semantic representation of a social user profile [17] and storage of the extracted data is necessary for the further usage of the information, for example to analyze a users habits.

1.3 Aim of the Work

The aim of this master thesis is to analyze a selection of the currently most important social networks and build an adaptor prototype for TheHiddenU (cf. Figure 1.1). It should be easy to add new social networks in the future and developers of other parts of TheHiddenU should have an easy access to the extracted data. An implementation of the common authorization system oAuth, *An open protocol to allow secure API authorization*[4], should be realized. To have access to an up-to-date ontology a tool called *Json2Ontology* (cf. Figure 1.1) should be developed and

[3] `http://www.netural.com/`
[4] `http://oauth.net`

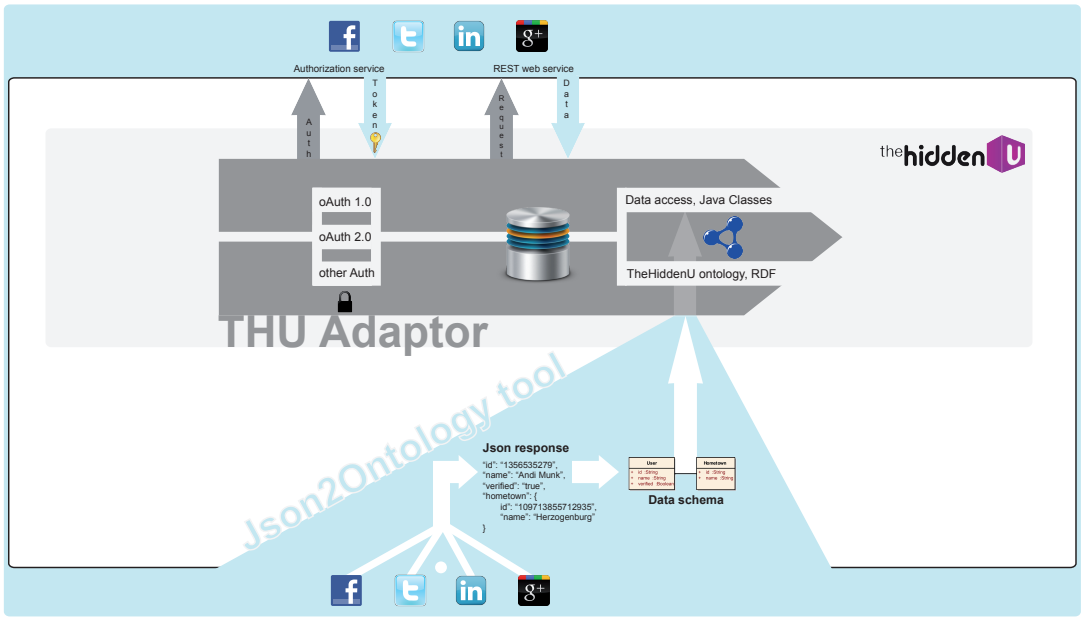


Figure 1.1: Implemetation: THU Adaptor and Json2Ontology tool

deliver a model, represented by a TheHiddenU-specific ontology code which I will explain in Section 5.1. With this ontology code we will be able to generate java classes for an easy access to the data and class diagrams help to understand the structure. The aim of the work is a detailed description and comparison of the selected social networks, a tool which automatically create a model of the social networks and an easy-to-use adaptor.

1.4 Methodological Approach

1. **Literature and documentation survey**. As the aim of the work is to build an adaptor prototype, my first step consists of surveying literature and documentations about social networks to define what is a social network and which features and categories exist. I will investigate available APIs for the future extraction of social data. The challenges, benefits, limitations and drawbacks should be determined. With help of the feature- and category list I will evaluate selected social networks and describe them to increase the understanding for the following steps.

2. **Implementation**. An adaptive prototype for TheHiddenU should be implemented. With this adaptor, future developers of TheHiddenU should access social networks and other data sources. The Json2Ontology-Tool should provide ontology code (Section 5.1) to allow to instantly get an overview of the structure of the available data (for example by generating UML class diagrams) and create the java classes for the adaptor.

3. **Evaluation and testing**. The last step should be an evaluation of the prototype to identify possible enhancements and a comparison if all elements and relations of the available data has been extracted from the Json2Ontology tool.

1.5 Structure of the Work

The remaining part of this thesis is structured as follows: in Chapter 2, a definition of Social Network and a description of features and a possible categorization is presented. Chapter 3 covers a description and an evaluation of selected social networks which will be used for the further implementation of the adaptor prototype. The retrieval of social data is outlined in Chapter 4 with details about the authorization and extraction of personal data of social networkers. In Chapter 5 I will describe the developed Json2Ontology tool, which is used for the extraction of the data schema of social networks. This the output of this tool should be evaluated in Chapter 6. Finally, in Chapter 7 I will give an overview of related work, Chapter 8 summarizes the thesis, ending with a conclusion and offering an outlook to future work.

CHAPTER 2

Social Networks

2.1 History

Since the beginnings of the public internet in the 1990s, people used tools to communicate. With e-mails there exist an easy way for communication, e-mail lists enabled the connection of people with same interests to get information about a specific topic and built the first online community structures. Discussion systems like Usenet where people can read or post news to newsgroups are predecessors of the nowadays widespread web forums with tree like directory structures. In web forums there exist more or less mandatory memberships and so first user profile features. Users can create so called threads in predefined categories which start a discussion about a thread-topic and enables users to publish their comments in posts. The era of user generated content has started with the first wiki-based system called WikiWikiWeb, developed by Ward Cunningham in 1994 [8] and has now brought to perfection with Wikipedia[1]. In wikis, every user has the possibility to create and edit articles and so collect together information and build powerful encyclopedias. Another form of online platforms are the so called online blogs (web logs) which first mentioned by Jorn Barger in 1997 [20] and enabled users to start their own news platform where they can create articles, comments or share photos to their community. People always had the desire to connect together and share information. Social networks cover this demand in specific area of interests and combine the features for users to connect, discuss, present themselves, share photos or exchange information.

2.2 Definitions

For the following explanation I will split up the term Social Network in his parts: 'Social' and 'Network'. In the web we can find hundreds of definitions of the word social:

[1] http://wikipedia.org

- „The term Social refers to a characteristic of living organisms ... It always refers to the interaction of organisms with other organisms and to their collective co-existence, irrespective of whether they are aware of it or not, and irrespective of whether the interaction is voluntary or involuntary.“[2]
- „Living together in communities “[3]
- „tending to form cooperative and interdependent relationships with others „[4]

In [7], J.S. Dolwick describe three social approaches and possible the first one, in perhaps the broadest sense will match the requirements in terms of social networks: Social means association and come from the Latin word socius, meaning a companion or associate, with the root, sequi, meaning 'to follow'. We can say that there is a connection or interaction between individuals, like animals, plants or humans.

The word network is used in different areas like electrical network, graph network or business network. They all the identical property that there exist interconnections, for example between electrical elements, graph nodes or business persons.

As mentioned before, in both of the words there exist an interpretation, that there are connections. Moreover the association and participation of the connected actors is important in social networks, which can be offline in the real world, or online in the internet.

Name	**Connection**	**Association/Participation**
sports clubs	same interests, same goal (championship, fitness)	play together in teams
workplace	same department, same tasks	work together at a company
music band	same band	make music together
City Council	voted to represent inhabitants	make decisions to improve city life

Table 2.1: Social Networks in a broader sense

Nowadays nobody thinks of such offline social networks (cf. Table 2.1), most people associates social network the online ones, where crafty enterprises develop online platforms which exactly satisfy the needs of the actors of offline social networks, such of the ones mentioned above.

My definition of the term Social Network in relation to this thesis is the following:

A Social Network is an online platform which satisfy the requirements of actors in terms of the

[2] http://en.wikipedia.org/wiki/Social
[3] http://www.thefreedictionary.com/social
[4] http://www.merriam-webster.com/dictionary/social

category of the Social Network. The main basic requirement is the interaction with other actors. A Social Network is a model of the reality in the internet.

2.3 Features and Characteristics

The following list is used in the evaluation of the social networks in Chapter 6, I defined some of the most important main and global features and characteristics of a social network.

- **Connections**: Social actors get in contact with other actors. Such connections can be friendships, kinships, same interests, geographical relationships, knowledge or prestige.
  - **Bilaterally 1:1 Connection**
    both actors must accept the connections.
    Examples: friendship, family relation
  - **Bilaterally 1:n Connection**
    one actor can connect to a group of actors and this connection must be accepted by a group administrator.
    Examples: closed groups like virtual school classes, company group
  - **Unilaterally 1:1 Connection**
    one actor can connect to another actor without his permission.
    Examples: be a fan of things like products, trademarks or humans such as celebrities, politician
  - **Unilaterally 1:n Connection**
    one actor can connect to a group of actors without any permission.
    Examples: open/public groups
- **Private User Profile**: Social networks also satisfy the desire of social actors to represent themselves. This can be in the form of private or public profiles with relevant information in the sense of the basic idea of the network. In business networks actors will publish information about their career or education, in music networks they will present their musical taste. Every social network has its individual information about the user.
- **Non-Private Profiles**: In addition to private user profiles, social networks allow companies to create social profiles for non-human things like trademarks, products and so on.
- **Communication**: Social networkers can communicate with private or public messages
- **Application Programming Interface (API)**: With an application programming interface, developers can use the data from users of the social network for own applications.
  - **Authorization Type**
    To get access to private data of the user, an authorization system had to be used.
    Examples: oAuth 1.0, oAuth 2.0, OpenID, individual authorization system

  - **Internal Applications**
    Developers can create applications which run inside the social network
  - **External Applications**
    Developers can use the API to access data to improve the social experience in external applications
  - **Single Sign On**
    Developers can benefit from single sign on features when a user connect an external application with the social network

- **Customer Relationship**

  - **Import contacts**
    Import your contacts from other social networks or webmail services.
  - **Newsletter**
    Periodically news updates from the social network.
  - **Comeback mails**
    Users which were not logged in for a while get mails where the features or the attractiveness of the social network are explaineds

CHAPTER 3

Survey on selected Social Networks

In the following sections I evaluate 4 social networks, which I examined for my master thesis. In order to satisfy the demand of the TheHiddenU project to grant access to different social data and also to select the most widespread social networks in Austria, I decided to discover **Facebook**[1], as the biggest social network in the world with an enormous amount of daily-life data and an easy-to-use API. More about Facebook and the other selected social networks in the following sections. To satisfy the demand on business data, I thought about XING.

XING[2], former named OpenBC (Open Business Club), was founded in 2003 in Germany and provide a business social network for professionals from all kinds of industries. The main ideas of XING is that users can meet up, find jobs, colleagues, new assignments, cooperation partners, experts and generate business ideas. With more than 11 million users[3] in 200 countries, XING will meet our vision of a social network, accessed in the TheHiddenU project. After searching for an API, I aborted the idea to use this social network. XING published an blog entry in their dev-blog[4] in April 2011, where they introduced in a feature called XING-connect where developers will be able to integrate XING in their own web sites. Unfortunately, XING-connect was only a project of 3 XING engineers, Christopher Blum, Lennart Koopmann and Nenad Nikolic, and in a very beta stage. The access is limited to the name, profile picture and profile url to identify users. This and the fact that there was no further dev-blog entry about XING-connect or an API, it will not justify the working hours to implement an XING adaptor.

Good alternatives for XING are the business social networks **Viadeo**[5] and the selected social network **LinkedIn**[6].

[1] http://www.facebook.com
[2] https://www.xing.com/
[3] http://corporate.xing.com/english/unternehmen/xing-ag/
[4] http://devblog.xing.com
[5] http://www.viadeo.com
[6] http://www.linkedin.com

Another selected social network with a very different type of data is the micro blogging service **Twitter**[7], where users are able to publish 140-character messages, known as *Tweets*. 200 million users use this platform, also called as the „SMS of the internet“.

The fourth selected social network is **Google+**[8], operated by Google Inc. Since the first testing phase and launch in June 2011, Google+ reached about 40 million users. After the early beta stadium, where some users have the possibility to invite up to 150 users, Google+ was opened to everyone in September 2011. I selected this social network because of the high potential to be a serious competitor of Facebook and the innovative entrance.

The selected social networks will be explained and evaluated in the following sections: Facebook, LinkedIn, Twitter and Google+.

Because of the complexity of the whole data structure, I tried to split up the whole data structure in UML [10] class diagrams to keep a clear view on the complete system. I defined some main classes, which will be illustrated in own class diagrams, bulky classes were splitted up to topic-related sub diagrams (cf. Figure 3.1)

Most of the attributes in the diagrams have speaking names, others will be described in the subsections.

3.1 Facebook

„Founded in February 2004, Facebook is a social utility that helps people communicate more efficiently with their friends, family and coworkers. The company develops technologies that facilitate the sharing of information through the social graph, the digital mapping of people's real-world social connections. Anyone can sign up for Facebook and interact with the people they know in a trusted environment. Facebook is a part of millions of people's lives all around the world. Facebook is a privately-held company and is headquartered in Palo Alto, Calif.“[9]

This is the main description of Facebook, available in 70 languages and the biggest social network in the world with more than 800 million active users. 50% of these users log on to Facebook every day and the average number of friends an user is connected to is about 130. Users can register for free and maintain their own user profiles, connect to other users and receive textual updates about their friends or pages in the home feed.Facebook Pages are the second main feature. With Pages users can represent their business, brands or themselves when they do not want to use their private profile for representation issues. With applications a Facebook Page can be upgraded to make it more individual, make a tombola or use other methods to gather more, so called, likes. When an user clicks on the like button of a Facebook Page, the user will receive status information from the page in the home feed.

Facebook use oAuth 2.0 authorization, which will be explained in Section 4.3. Depending on the

[7] `http://www.twitter.com`
[8] `https://plus.google.com`
[9] `http://www.facebook.com/press.php`

Figure 3.1: Social Networks: Package Overview

requested data, developers must define the needed permissions which were listed in the authorization screens. For some data the access token is sufficient, other fields like email, education require additional permissions. In the documentation[10] there exist a table with all possible fields and the required permission.

For re-engineering and discovering the Facebook data structure, I used a tool called „Graph API Explorer “[11]. With this tool, provided by Facebook, developers are able to request an access token with previous selected permissions. „GET “, „POST “and „DELETE “requests can be

[10] `http://developers.facebook.com/docs/reference/api/`
[11] `http://developers.facebook.com/tools/explorer`

issued against graph.facebook.com URLs. Another source of information is the official API documentation[12]. In my evaluation I noticed, that the documentation is not 100% persistent. Some enumerations are not described, or some fields, responded by the API were not listed.

Facebook provides an autocomplete list in the user interface to define associations to pages (or users). The autocomplete list show appropriate pages (matching entered letters and page category). I evaluated the user interface to create a not exhaustive list of possible page categories for several associations listed in the class diagrams. I tried to type in different letters in the profile field and examine the appearing pages to create a list of possible page categories. Some associations permit exactly one page category, others are not so strictly and permit different page categories. If the entered letters do not match, a new page with the entered name will be automatically created. The first mentioned category in the category list is defined for the new created page. You can find the tables after the class diagrams, captioned with „Autocomplete List: CLASSNAME “. In a restriction column I try to assess if there are limited, topic related categories in the autocomplete list.

Basic Features and Characteristics

The followingitems are described in Chapter 2.3.

- **Connections**:
  - **Bilaterally 1:1 Connection:** Friends, Family (cf. Figure 3.2)
  - **Bilaterally 1:n Connection:** Groups with privacy setting closed or secret (cf. Figure 3.6)
  - **Unilaterally 1:1 Connection** User can like a fanpage (cf. Figure 3.15)
  - **Unilaterally 1:n Connection** Groups with privacy setting open (cf. Figure 3.6)
- **Private User Profile**: Standard user profile after registration with lot of possibilities to share information like (cf. Figure 3.2)
- **Non-Private Profiles**: Pages for business, politics, brands, ... (cf. Figure 3.15)
- **Communication**: Write Notes, Comments, Postings, Messages or share Videos, Photos, Albums and Links (cf. Figure 3.4)
- **Application Programming Interface (API)**:
  - **Authorization Type:** Facebook Connect, oAuth 2.0
  - **Internal Applications:** Integrate with Facebooks core experience by building apps that operate within the platform. Pages can add applications to present themselves in a cooperate identity, make contests or landing pages to encourage users to click the like button on the page.

[12] `http://developers.facebook.com/docs/reference/api/`

– **External Applications:** Facebook comment box, like buttons and a lot of other social plugins can be used in external websites

– **Single Sign On:** With Facebook Connect, users can use their Facebook credentials to log on a website and use their approved social data.

- **Customer Relationship**

– **Import contacts** incredible 2590 e-mail providers are supported for importing their contacts to Facebook. This indicate the importance of such a feature for social networks.

– **Newsletter** Only for administrators of pages with weekly statistics

– **Comeback mails** Information mails about friends which are active to come back and log on to Facebook.

Facebook User Private

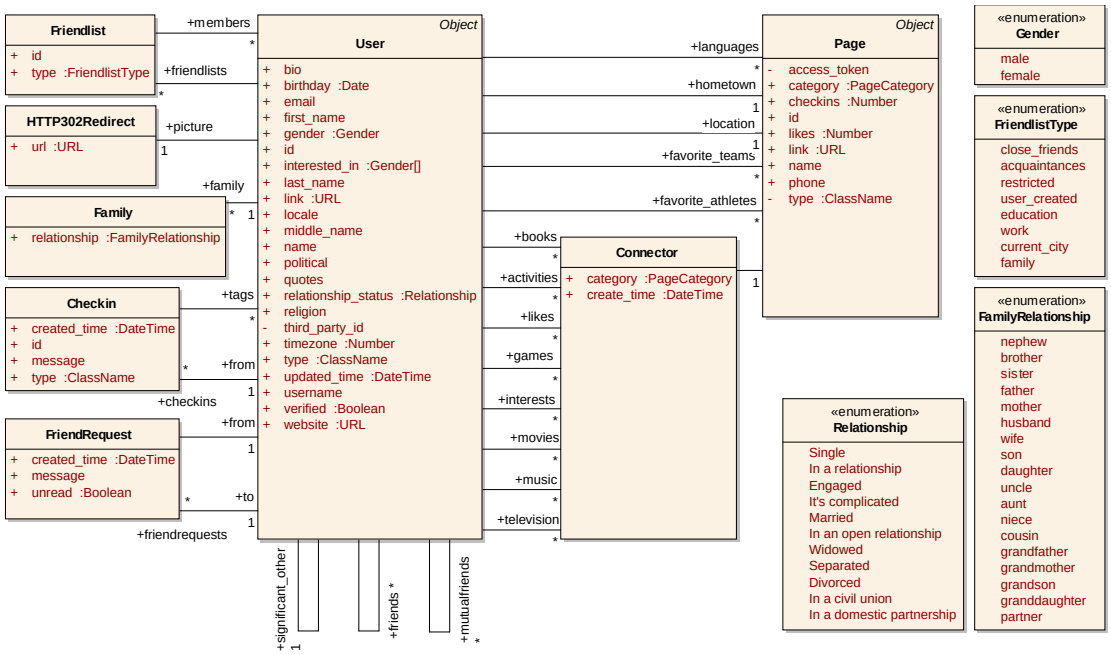


Figure 3.2: Facebook User Private

In Figure 3.2 you can see the first part of the Facebook User class diagram. In this diagram, the focus is on the private information about an individual user. Besides self speaking attributes and associations, listed in the official documentation[13], I will describe some selected elements. *HTTP302Redirect* will, as you can derive out of the name, redirect the request to, in this case, an URL where the profile picture of the user is located. Note the enumerations of the *family* relationship. With the *family* connection we can request family members including their relationship to the current user. With *Checkins* user can define online, that they are at a specific place or geo coordinates. More about the Checkin class in Figure 3.9. Facebook user can define a *relationship* status (e.g., married, enumeration Relationship) and an association to another user instance named *significant_other* (e.g. the wife or husband user account). The *mutualfriends* association describes the friends between the current authorized user and the requested user. Associations on the left side of the class diagram show a users *hometown*, *location*, *languages* and variations of *like* connections, such as *favorite teams*, *athletes*, *books*, *music*, ... All these associations target a Facebook Page (e.g., Figure 3.15), where further information about the object can be found. Some of these associations have additional information about the *category* of the page and the time, when the relation to the page had been added (*create_time*).

Data Access (GET Requests)

- User: https://graph.facebook.com/me for own user object or user id instead of 'me'
- User Connections, https://graph.facebook.com/ID/CONNECTION_NAME:
  - friendlists, family, checkins, friendrequests, books, activities, likes, games, interests, movies, music, television
- Page: https://graph.facebook.com/ID where 'ID' is the page id of the desired object

Inline Objects (Relations) of 'User'

- languages, hometown, location, favorite_teams, favorite_athletes, significant_other

[13] `http://developers.facebook.com/docs/reference/api/user/`

Association name	Page Category	Restriction
language	**Language**	yes
hometown	**City**	yes
location	**City**	yes
favourite_teams	**Professional sports team**, Amateur sports team, School sports team, Sports league, Community, Interest	yes
favorite_athletes	**Athlete**, Interest, Actor/director	yes
books	**Music**, Local business, Product/service, Community, Games/toys, University, Musician/band, Media/news/publishing, Tv network, Tv, Author, Computers/technology, Movie general, Book ...	no
activities	**Interest**, Movie genre, Literature genre, Anatomical structure, Field of study, Musical genre, ...	no
likes	all categories, no profile field	no
games	**Games/toys**, Interest, Community, Computers/internet, Field of study, Amateur sports team, ...	no
interests	**Interest**, Tv network, Movie, Musician/band, Regional, Non-profit organization, Tv show, Magazine ...	no
movies	**Movie general**, Tv show, Tv network, Interest, Movie, Movie genre, Book genre, City, Company, Entertainer, Cars, ...	no
music	**Book**, Musician/band, Professional sports team, Non-profit organization, Website, Local business, Album, ...	no
television	**Tv**, Tv network, Actor/director, Book genre, Tv channel, Tv show, City, Interest, Writer, ...	no

Table 3.2: Autocomplete List: Facebook User Private

Facebook User Business

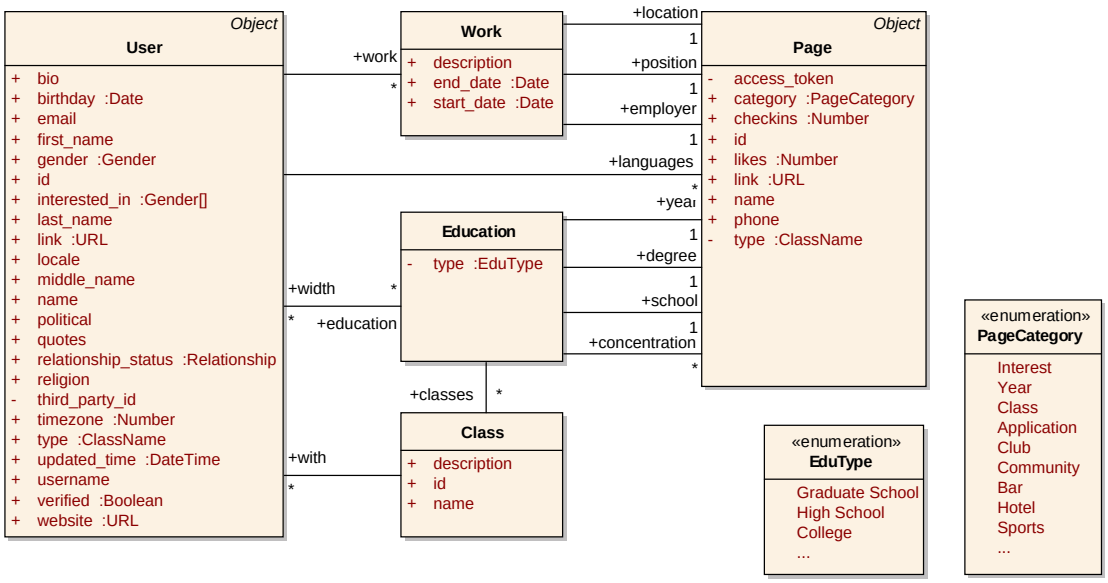


Figure 3.3: Facebook User Business

In Figure 3.3 the Business parts of a Facebook user are focused. An array of work elements can be defined, all with optional description, start- and end date elements. Relations to the Facebook page of the location, position and employer of the work object extend the information. Another part is the information about the education of a Facebook user, with an array of education elements, where classes, degree, concentration, school and year can be defined as well as other users, which also attend the same education or class.

Association name	Page Category	Restriction
position	**Work position**	yes
location	**City**	yes
concentration	**concentration or major**, field of study, interest, ...	no
degree	**degree**, Tv show, interest, ...	no
school	**school**, University, Interest, City, Company, local business, ...	no
year	**year**	yes

Table 3.3: Autocomplete List: Facebook User Business

Data Access (GET Requests)

- User: https://graph.facebook.com/me for own user object or user id instead of 'me'
- Page: https://graph.facebook.com/ID where 'ID' is the page id of the desired object

Inline Class of 'User'

- Work, Education

Inline Class of 'Education'

- Class

Inline Objects (Relations) of 'Work'

- location, position, employer

Inline Objects (Relations) of 'Education'

- year, degree, school, concentration

Inline Object (Relation) of 'Class'

- with

Facebook User Media

One of the biggest class diagram is about Facebook User Media features (cf. Figure 3.4). An user has a lot of possibilities to share information on Facebook. Messages from one user to another, collected in threads, or postings, photos, albums, videos, links which can be commented by users and where the privacy settings can be defined for each object. It is possible to publish public objects, or for all friends or just for a defined group of friends.

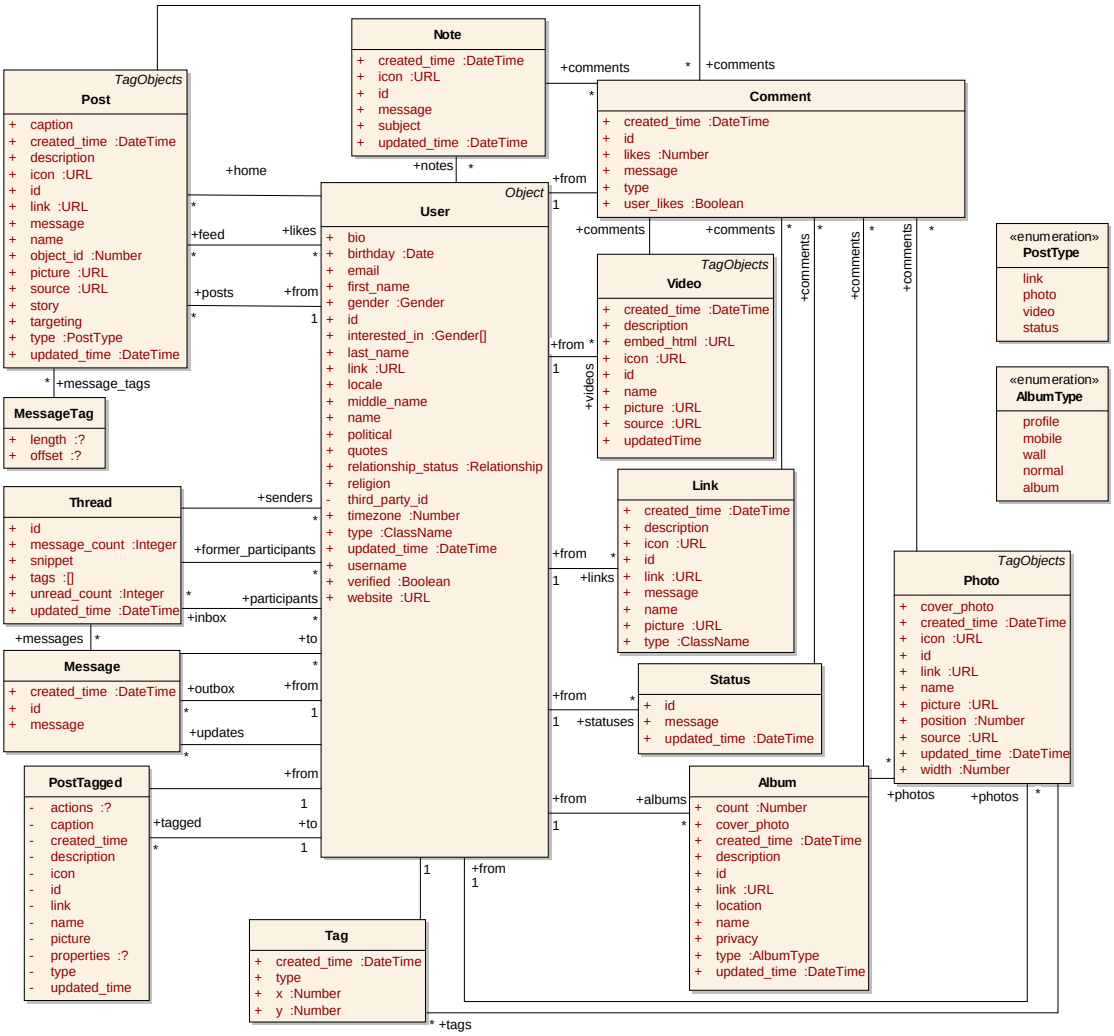


Figure 3.4: Facebook User Media

Data Access (GET Requests)

- User: https://graph.facebook.com/me for own user object or user id instead of 'me'
- User Connections, https://graph.facebook.com/ID/CONNECTION_NAME:
  - feed, posts, home, inbox, outbox, updates, tagged, albums, statuses, links, videos, notes

Facebook User System

Notifications for the user are visible in the user interface to inform the user about new friend requests, new messages and comments or likes of published postings, links, videos, photos or albums. The achievement object represents the achievement achieved by a user for a particular application like games. A very new feature are orders, where users can buy something within applications. With the account connection, the Facebook apps and pages owned by the current user are returned. These elements are described in Figure 3.5.

Data Access (GET Requests)

- User: https://graph.facebook.com/me for own user object or user id instead of 'me'
- User Connections, https://graph.facebook.com/ID/CONNECTION_NAME:
  - accounts, notifications, achievements, payments

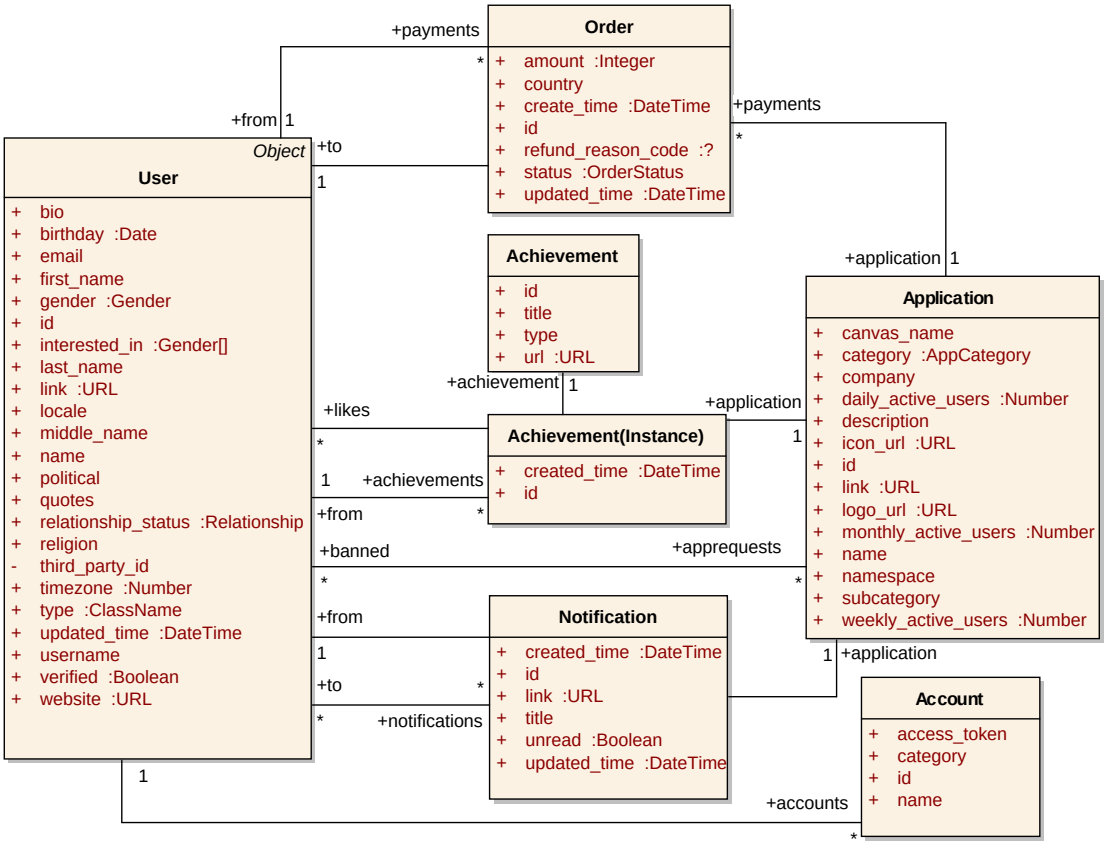


Figure 3.5: Facebook User System

Facebook User Fun (cf. Figure 3.6)

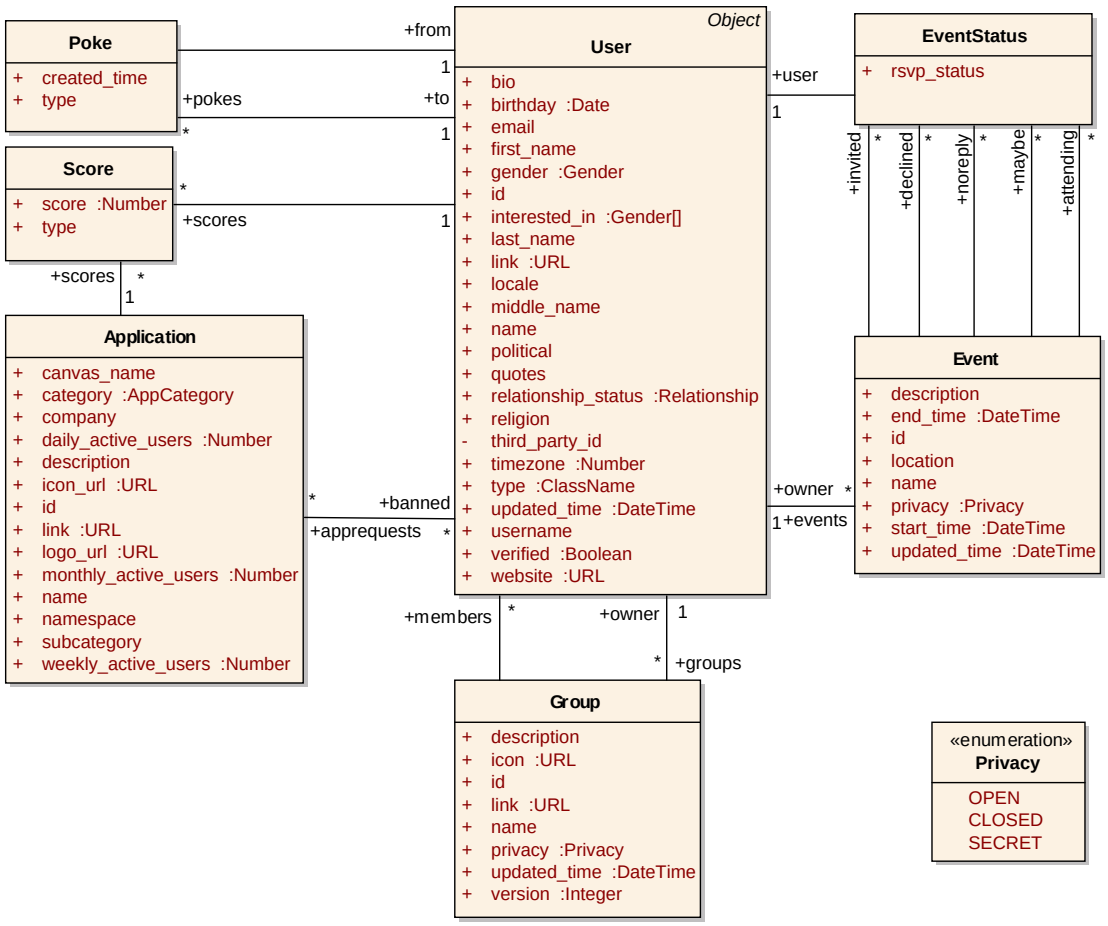


Figure 3.6: Facebook User Fun

Data Access (GET Requests)

- User: https://graph.facebook.com/me for own user object or user id instead of 'me'
- User Connections, https://graph.facebook.com/ID/CONNECTION_NAME:
  - pokes, scores, apprequests, groups, events

Facebook Album and Photo (cf. Figure 3.7)

User and page objects can create albums and upload photos.

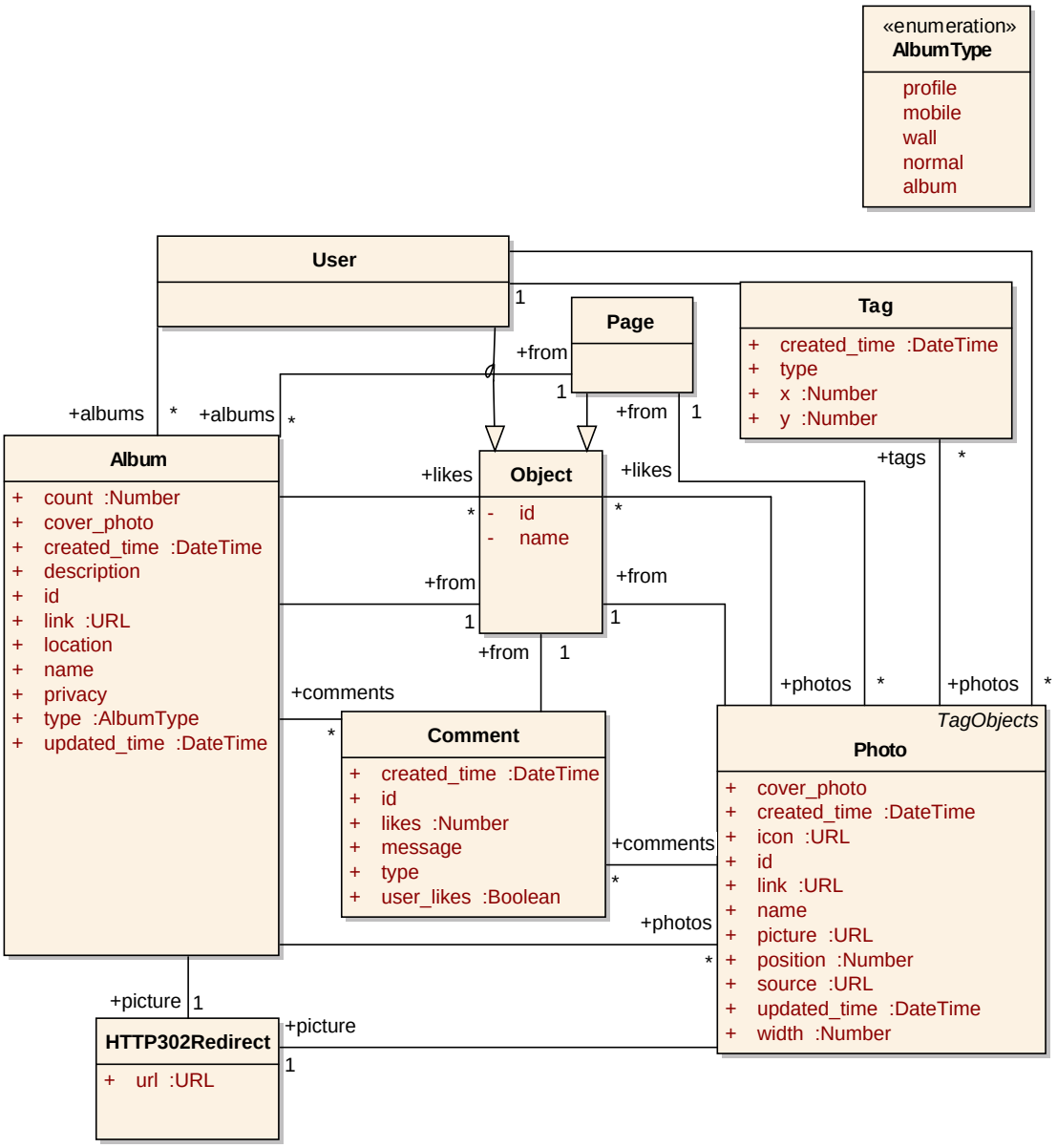


Figure 3.7: Facebook Album and Photo

Data Access (GET Requests)
Additional permissions required: user_photos, friend_photos

- Album: https://graph.facebook.com/ID where 'ID' is the album id
- Album Connections, https://graph.facebook.com/ID/CONNECTION_NAME:

  - photos, likes, comments

- Photo: https://graph.facebook.com/ID where 'ID' is the photo id

- Photo Connections, https://graph.facebook.com/ID/CONNECTION_NAME:

  - tags, likes, comments

Facebook Application

Applications (cf. Figure 3.8) on Facebook are used in different purposes.

SingleSignOn and data access. With Facebook applications, developers can implement a single sign on system on their websites. New user can register using the social data like name, e-mail address and every other information stored at Facebook. When an user is logged in on Facebook, this is recognized by the application and the user is also logged in on the website, where the application is integrated.

Extend possibilities on Pages. Social media agencies create small applications which they can add to Facebook Pages, to extend the functionalities. The owner of the pages can present themselves or their business in applications or try to get more fans with competitions or fan gates, which are these applications which tempt users to click the like button.

Data Access (GET Requests) Applications a user administers can be retrieved via the /accounts connection on the User object.

- Application: https://graph.facebook.com/ID where 'ID' is the application id

- Application Connections, https://graph.facebook.com/ID/CONNECTION_NAME:

  - accounts, albums*, banned, feed*, insights, links*, payments, picture, posts*, reviews, statuses*, subscriptions, tagged*, translations, scores, achievements, videos

* ... Deprecated. Will be removed on March 1st, 2012, because the applications profile page will be removed.

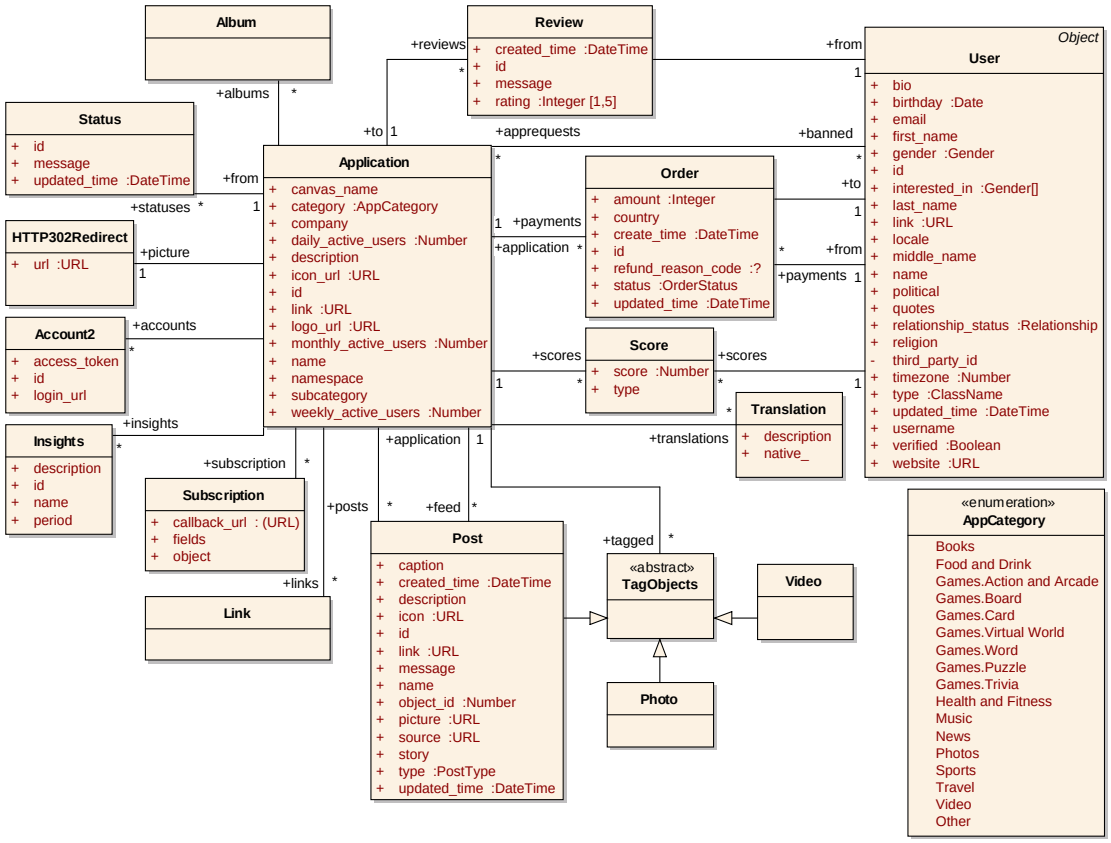


Figure 3.8: Facebook Application

Facebook Checkin

With Facebook Checkin (cf. Figure 3.9) a user can tag himself at a specific location, based on the geographic coordinates. This Checkin is displayed in the user profile and, if a page is assigned to the geographic coordinates, the Checkin is also displayed at the pages profile.

Data Access (GET Requests)

Additional permissions required: user_checkins, friends_checkins

- Checkin: https://graph.facebook.com/ID where 'ID' is the checkin id
- Checkin Connections, https://graph.facebook.com/ID/CONNECTION_NAME:
  - likes, comments

Inline Class of 'Checkin'

- Place

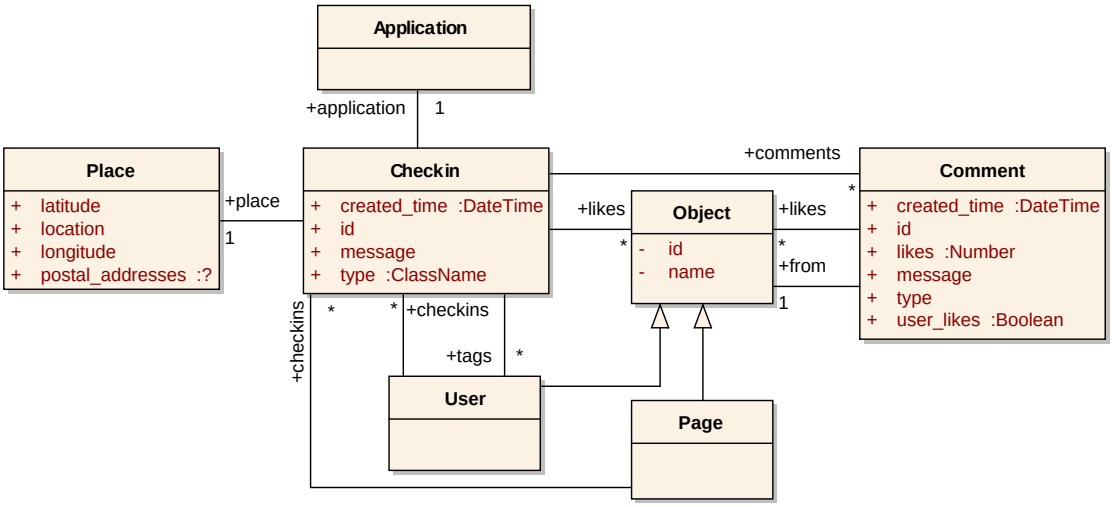


Figure 3.9: Facebook Checkin

Inline Objects (Relations) of 'Checkin'

- from, applications, tags

Facebook Event

„Organize gatherings, respond to invites, and keep up with what your friends are doing.“[14] With Facebook Events (cf. Figure 3.10), users can promote public or private events with an own page where information about the event and an event picture can be published. The two main features are the invitations and the comment box. An user can attend, decline or express that they will maybe attend an event, attending users can invite their own friends to spread the information. In the comment box users can discuss about the event, share pictures, links, comments or videos with others. Facebook allows users to set individual privacy for an event, such as open for everyone, private so that only invited user can attend or closed where only invited user can see the event page.

[14] `http://www.facebook.com/help/events`

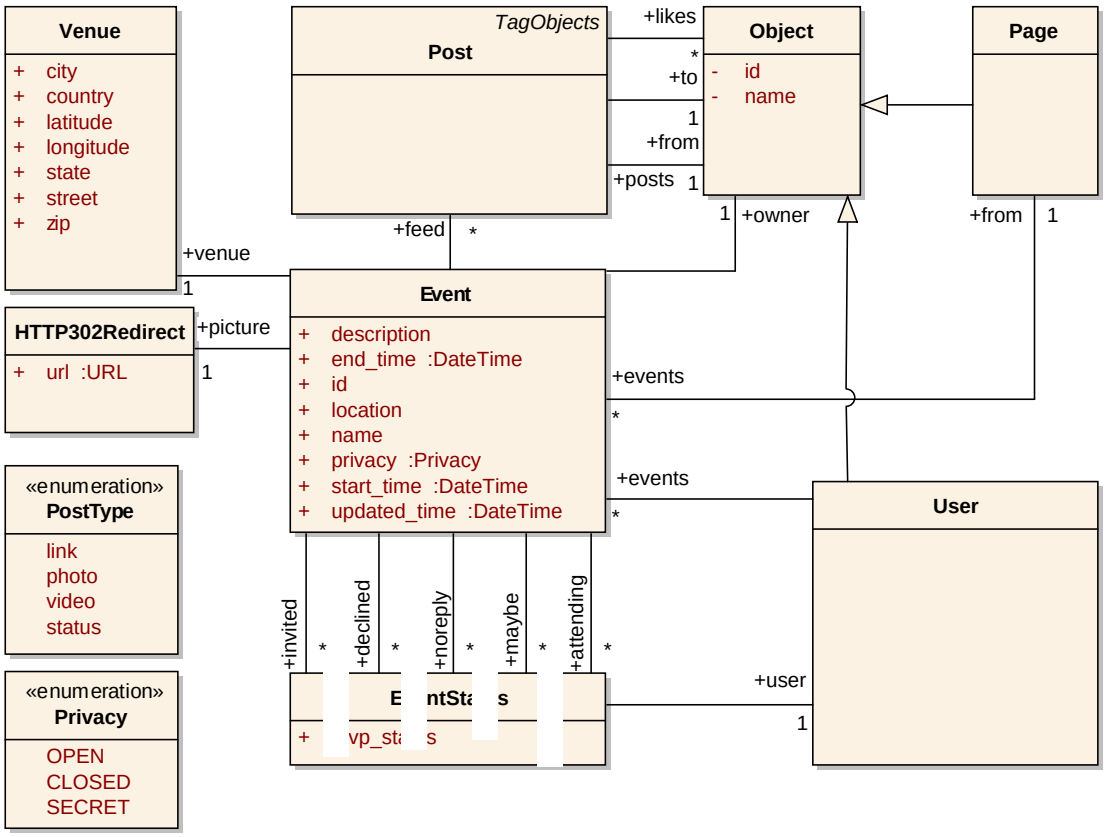


Figure 3.10: Facebook Event

Data Access (GET Requests)
Additional permissions required: user_events, friends_events for non-public events

- Event: https://graph.facebook.com/ID where 'ID' is the event id
- Event Connections, https://graph.facebook.com/ID/CONNECTION_NAME:
  - feed, noreply, invited, attending, maybe, declined, picture, videos

Inline Class of 'Event'

- Venue

Inline Object (Relation) of 'Event'

- owner

Facebook Group

A Facebook Group (cf. Figure 3.11) is a possibility for users to connect together for cases like organizing something. The advantage to Facebook Pages is, that in groups there are privacy settings like closed or secret, so that not everyone can attend this group.

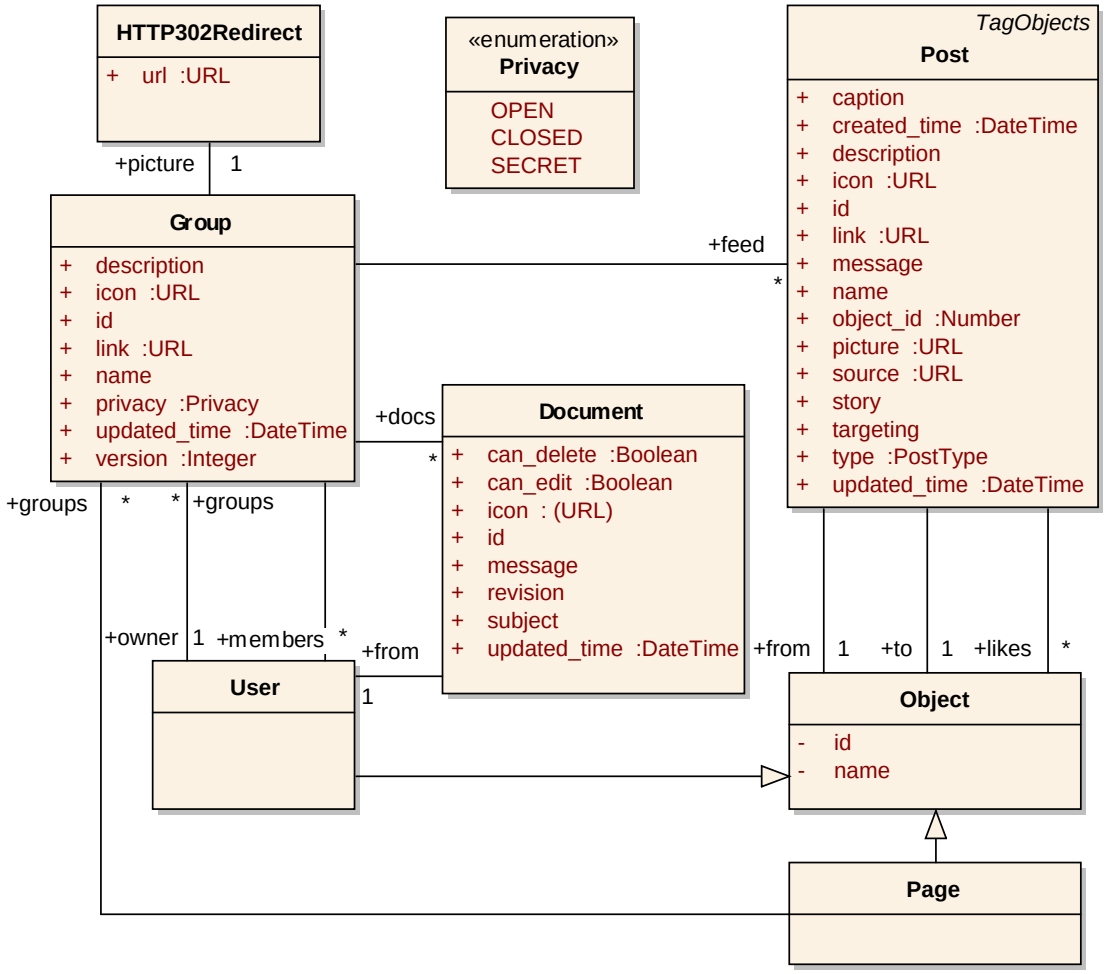


Figure 3.11: Facebook Group

Data Access (GET Requests)
Additional permissions required: user_groups, friends_groups for non-public groups

- Group: https://graph.facebook.com/ID where 'ID' is the group id
- Group Connections, https://graph.facebook.com/ID/CONNECTION_NAME:
  - feed, members, picture, docs

Inline Object (Relation) of 'Group'

- owner

Facebook Link

A link (cf. Figure 3.12) shared on a wall. The User and Page objects can publish links on walls.

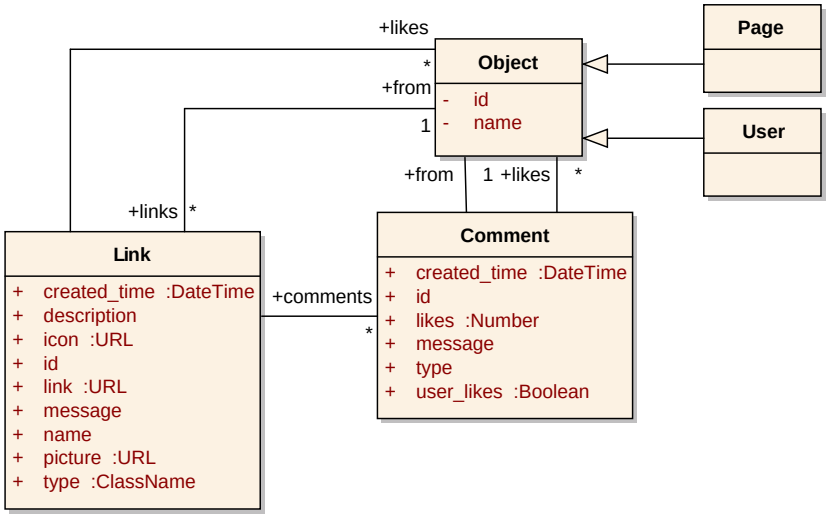


Figure 3.12: Facebook Link

Data Access (GET Requests)
Additional permissions required: read_stream if the link is not public

- Link: https://graph.facebook.com/ID where 'ID' is the link id
- Link Connections, https://graph.facebook.com/ID/CONNECTION_NAME:
  - comments, likes

Inline Object (Relation) of 'Group'

- from

Facebook Note (cf. Figure 3.13)

Although there is a connection from user to note in the documentation[15], I do not know any way to write a note with an user account. Pages have such a possibility, which is often used.

[15] `http://developers.facebook.com/docs/reference/api/note/`

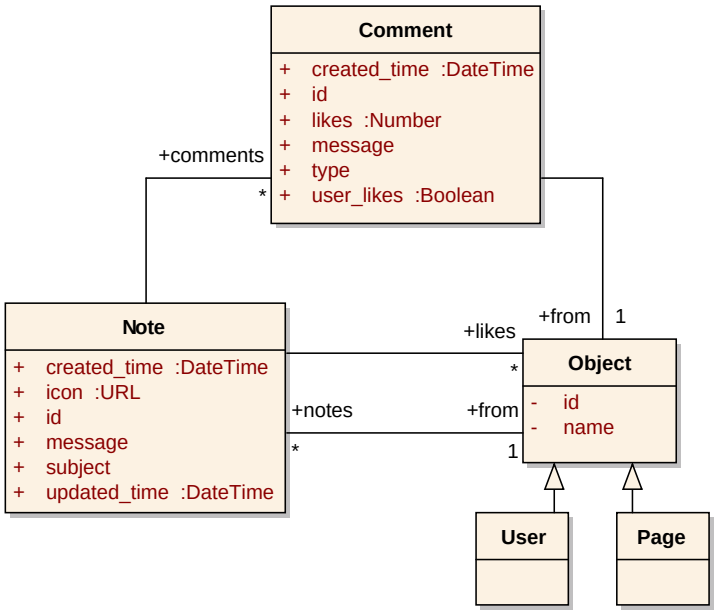


Figure 3.13: Facebook Note

Data Access (GET Requests)

Additional permissions required: user_notes, friends_notes for non-public notes

- Note: https://graph.facebook.com/ID where 'ID' is the note id
- Note Connections, https://graph.facebook.com/ID/CONNECTION_NAME:
  - comments, likes

Inline Object (Relation) of 'Note'

- from

Facebook Order

This is a very new feature, which can be accessed by the API. The order object (cf. Figure 3.14) is used to interact with orders created by an application using Facebook credits. Credits can bought by users and used to pay in applications.

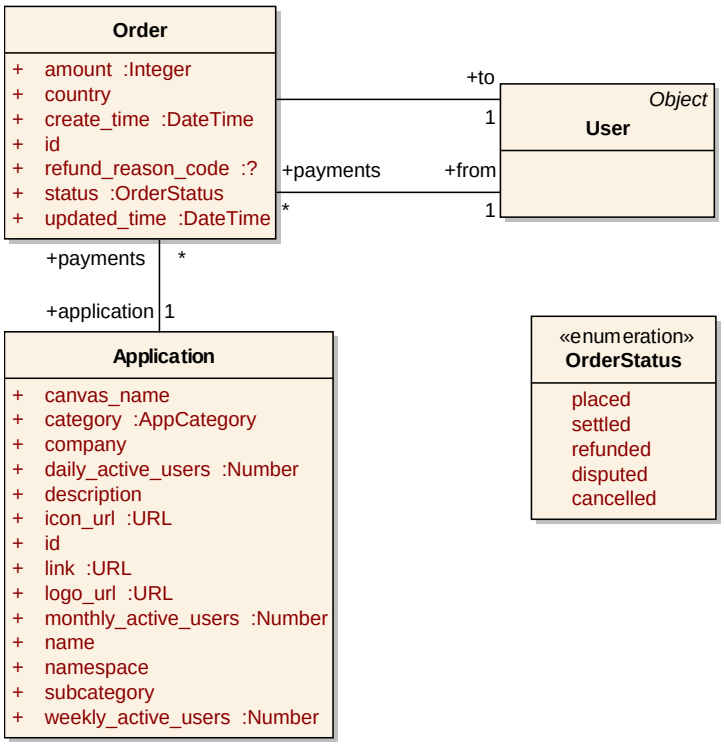


Figure 3.14: Facebook Order

Data Access (GET Requests)
Additional permissions required: application or user access token

- Order: https://graph.facebook.com/ID where 'ID' is the order id

Inline Objects (Relations) of 'Order'

- from, to, application

Facebook Page

A Facebook Page (cf. Figure 3.15) is the second main object on Facebook apart from a Facebook User. On Pages it is possible to represent something like a brand, company, product, celebrity, politician, ... A lot of companies realize the potential to get in contact with potential customers of their products or services and present themselves or their products on Facebook. The social media branch is growing very fast and agencies create strategies for their clients and advise them to use the functionality of Facebook in their business case as effectively as possible. With applications a page can be extended. Worldwide brands like Red Bull[16] with more than 23 million fans which clicked the like button on their page demonstrate the power of Facebook Pages and

[16] `http://www.facebook.com/redbull`

is often mentioned as a best practise project.

There are a lot of page categories with different optional attributes and inline classes for a lot of areas. Here are some examples: Actor/director, Amateur sports team, Application, Arts/entertainment/nightlife, Arts/humanities, Athlete, Attractions/things to do, Bank/financial institution, Bank/financial services, Bar, Business services, Business/economy, Camera/photo, Cars, Cause, City, Club, Coach, Community, Community organization, Company, Computers, Computers/internet, Concentration or major, Concert tour, Consulting/business services, Course, Degree, Diseases, Education, Energy/utility, Entertainment, Event planning/event services, Field of study, Food/beverages, Games/toys, Hotel, Interest Internet/software, Jewelry/watches, Language, Legal/law, Lifestyle, Local business, Local/travel, Magazine, Media/news/publishing, Movie, Movies/music, Musician/band, News/media, Non-profit organization, Organization, Outdoor gear/sporting goods, Product/service, Professional services, Professional sports team, Public figure, Public places, Radio station, Real estate, Reference, Regional, Restaurant/cafe, Retail and consumer merchandise, School, Small business, Software, Sport, Sports, Sports league, Sports venue, Sports/recreation/activities, Telecommunication, Transportation, Travel/leisure, Tv channel, Tv show, University, Utilities, Website, Wine/spirits, Work position, Work project and Year.

Data Access (GET Requests) No access token required for public and non-demographically restricted pages.

- Page: https://graph.facebook.com/ID where 'ID' is the page id
- Page Connections, https://graph.facebook.com/ID/CONNECTION_NAME:
  - feed, picture, settings, tagged, links, photos, groups, albums, statuses, videos, notes, posts, questions, events, checkins, admins, blocked, tabs, insights

Inline Classes of 'Page'

- Location, PaymentOptions, Parking, RestaurantSpecialities, RestaurantServices, ... depending on the page category

Facebook Post

Posts (cf. Figure 3.16) can be written from Pages, Users, Groups and Applications which have a feed connections containing post objects that represent their walls. In addition to this the User and Page objects have a connection named posts containing Posts made by the User and the Page respectively.

Data Access (GET Requests) No access token required for public and non-demographically restricted pages.

- Post: https://graph.facebook.com/ID where 'ID' is the post id
- Post Connections, https://graph.facebook.com/ID/CONNECTION_NAME:

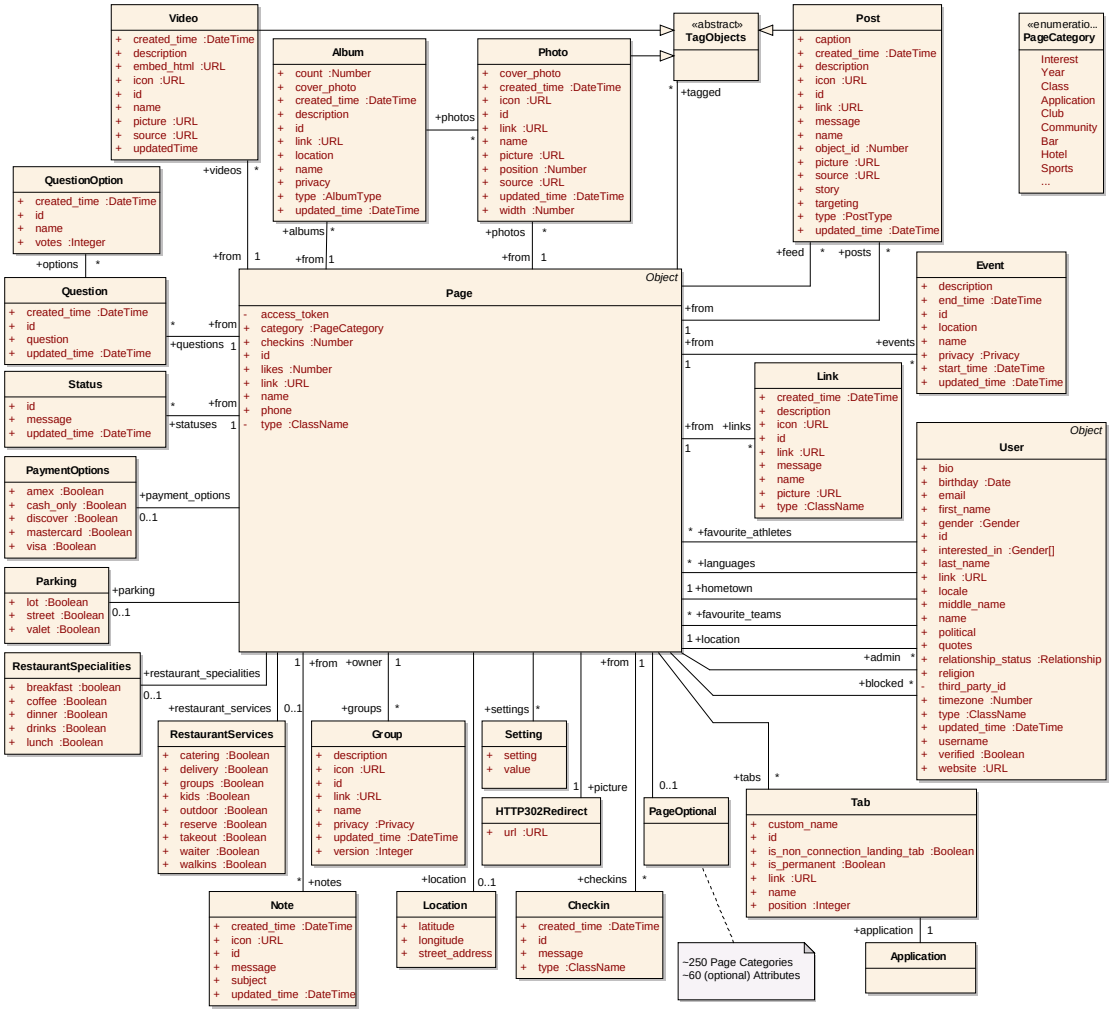


Figure 3.15: Facebook Page

- comments, likes, insights

Inline Object (Relation) of 'Post'

- from, to, application

Inline Classes of 'Post'

- Place, Property, Action, PostPrivacy, MessageTag

Facebook question

A question (cf. Figure 3.17) asked by a user or page.

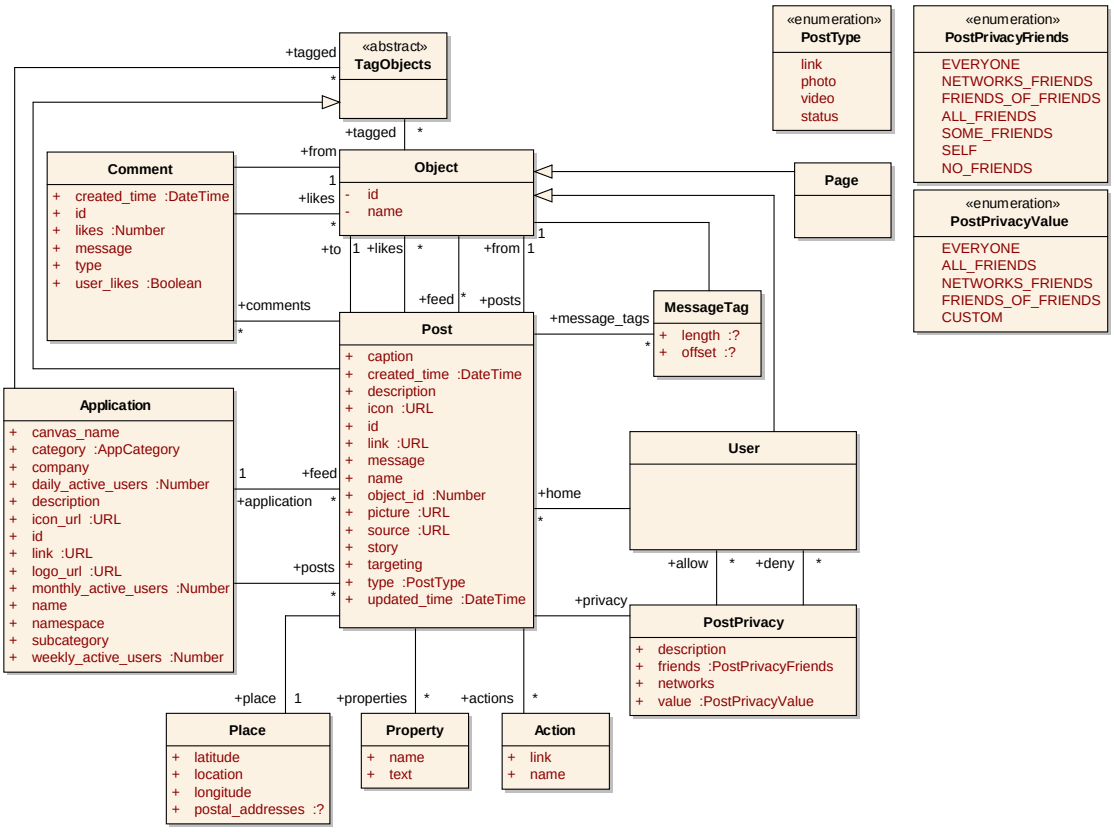


Figure 3.16: Facebook Post

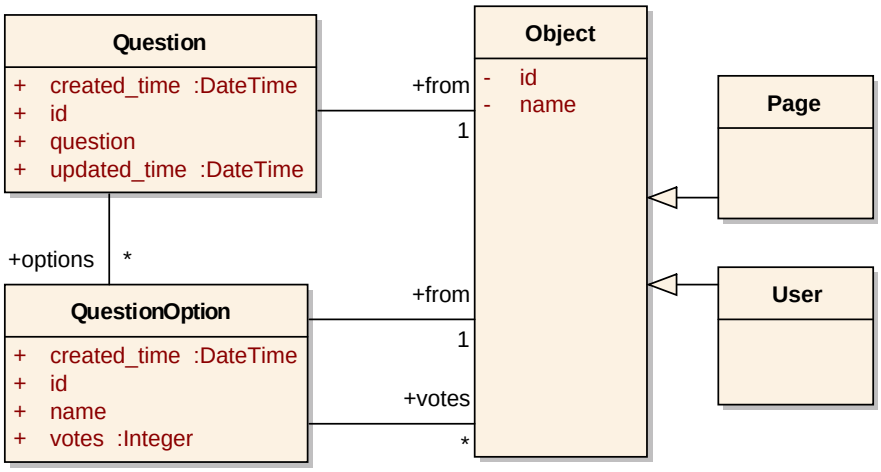


Figure 3.17: Facebook Question

Data Access (GET Requests)

Additional permissions required: user_questions for questions asked by the current user, friends_questions for questions asked by friends of the current user

- Question: https://graph.facebook.com/ID where 'ID' is the question id
- Question Connections, https://graph.facebook.com/ID/CONNECTION_NAME:
  - options
- QuestionOption Connection, https://graph.facebook.com/OPTION_ID/votes

Inline Object (Relation) of 'Question'

- from

Inline Object (Relation) of 'QuestionOption'

- from

Facebook Status

A Facebook Status (cf. Figure 3.18) is quite similar like a Facebook Post, but has no multimedia attached.

Data Access (GET Requests)

- Status: https://graph.facebook.com/ID where 'ID' is the status id
- Status Connections, https://graph.facebook.com/ID/CONNECTION_NAME:
  - comments, likes

Inline Object (Relation) of 'Status'

- from

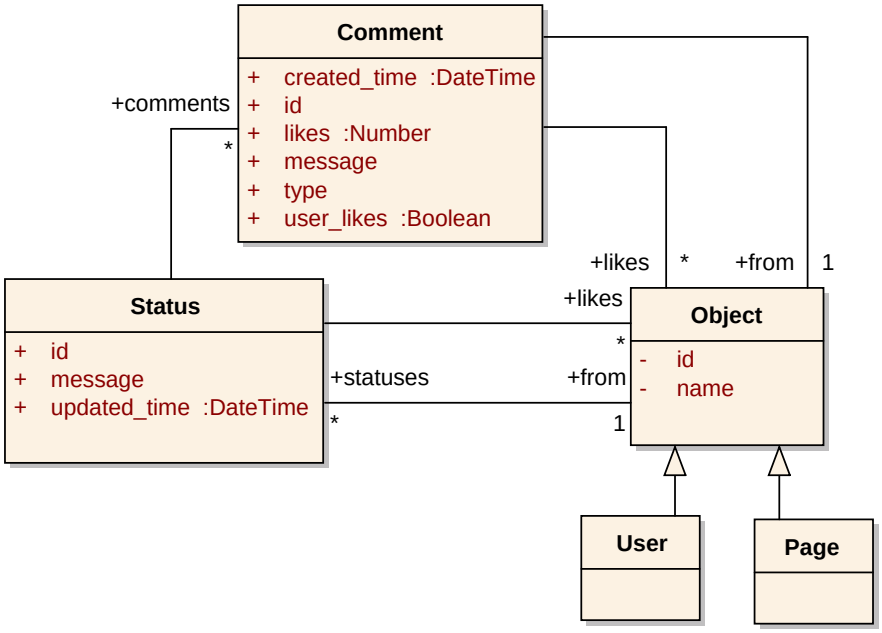


Figure 3.18: Facebook Status

3.2 LinkedIn

LinkedIn, founded in 2002 in California and launched in 2003, is with more than 135 million users the biggest business social network in the world and available in 14 languages (November 2011). LinkedIn filed for an initial public offering in January 2011 and traded its first shares on May 19, 2011, NYSE. The initial priced rose from $45 to $122.70 in the first day of trading. The main competitors of LinkedIn are Viadeo[17] with about 35 million users and the German business social network XING[18] with about 10 million users.

Besides private profiles LinkedIn also provides business pages for employers but no possibility of internal applications to enhance the business page functionalities like in Facebook.

For the manual evaluation of the data structure to create class diagrams I used the developers portal[19]. LinkedIn use oAuth 1.0a authorization, which will be explained in Section 4.3. LinkedIn is not qualified for an evaluation with the Json2Ontology tool because there is a basic barrier: the LinkedIn API force developers to use a so called field selector. With this field selector the developer must specify exactly which elements the API should respond. This fact make a tool like this useless, because I must know and list each element and will never find any hidden element.

[17] `http://www.viadeo.com`
[18] `http://www.xing.com`
[19] `https://developer.linkedin.com`

Basic Features and Characteristics

The followingitems are described in Chapter 2.3.

- **Connections**:
  - **Bilaterally 1:1 Connection:** Connections (cf. Figure 3.19)
  - **Bilaterally 1:n Connection:** Closed groups (request to join) (cf. Figure 3.22)
  - **Unilaterally 1:1 Connection** -
  - **Unilaterally 1:n Connection** Open groups (auto join) (cf. Figure 3.22)
- **Private User Profile**: Standard user profile after registration with lot of possibilities to share information about the business career, Figure 3.19
- **Non-Private Profiles**: Pages about companies, employer, for example about the University of Technology Vienna[20]
- **Communication**: Write update comments, private mails, recommendation, job suggestions, job bookmarks, Figure (cf. Figure 3.21)
- **Application Programming Interface (API)**:
  - **Authorization Type:** oAuth 1.0
  - **Internal Applications:** -
  - **External Applications:** 'Apply with LinkedIn', company or member profile widgets, share or recommend buttons and a lot of more plugins[21]
  - **Single Sign On:** Sign In with LinkedIn[22]
- **Customer Relationship**
  - **Import contacts** support mail providers like Gmail, Hotmail, Gmx, ...
  - **Newsletter** weekly LinkedIn network update mails
  - **Comeback mails** unknown

LinkedIn User Private (cf. Figure 3.19)

Data Access (GET Requests) Field selector must define all elements which are needed!

- Current User: http://api.linkedin.com/v1/people/
- User by ID: http://api.linkedin.com/v1/people/id=12345

[20] `http://www.linkedin.com/company/166803`
[21] `https://developer.linkedin.com/plugins`
[22] `https://developer.linkedin.com/plugins`

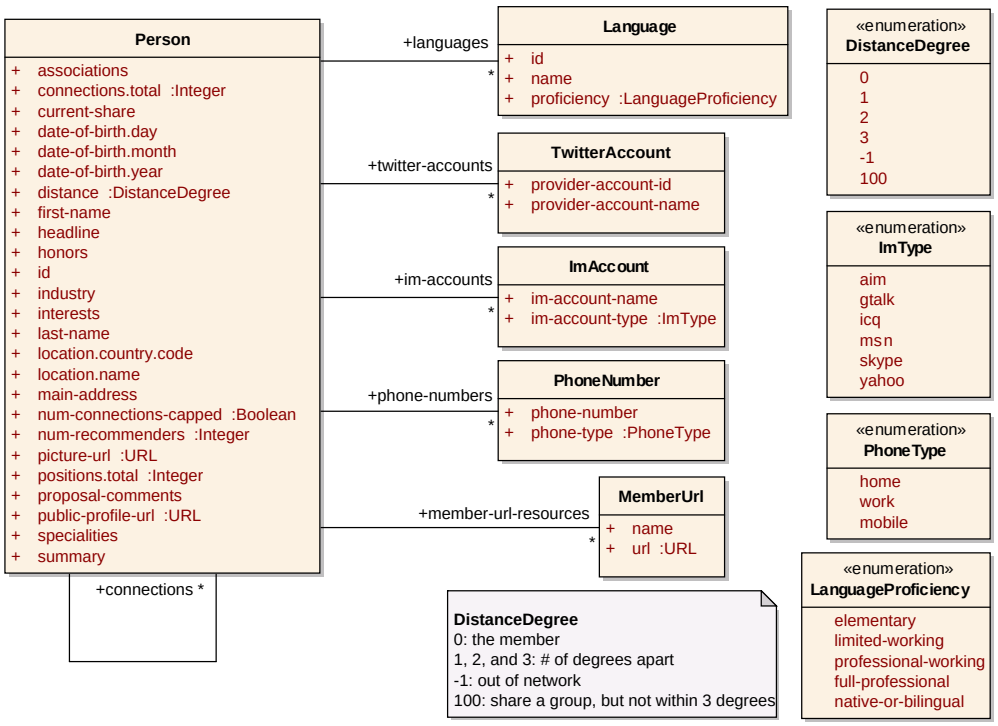


Figure 3.19: LinkedIn User Private

- User by profile URL: http://api.linkedin.com/v1/people/url=<public_profile_url>
- Example FieldSelector: http://api.linkedin.com/v1/people/ :(id,last-name,languages)

Inline Classes of 'Person'

- Language, TwitterAccount, ImAccount, PhoneNumber, MemberUrl

LinkedIn User Business (cf. Figure 3.20)

Data Access (GET Requests) Field selector must define all elements which are needed!

- Company: http://api.linkedin.com/v1/companies/ID
- Connections:
  - Groups: http://api.linkedin.com/v1/people/ /group-memberships

Inline Classes of 'Person'

- Position (positions, three-current-positions, three-past-positions), Skill, Certification, Education, Patent,

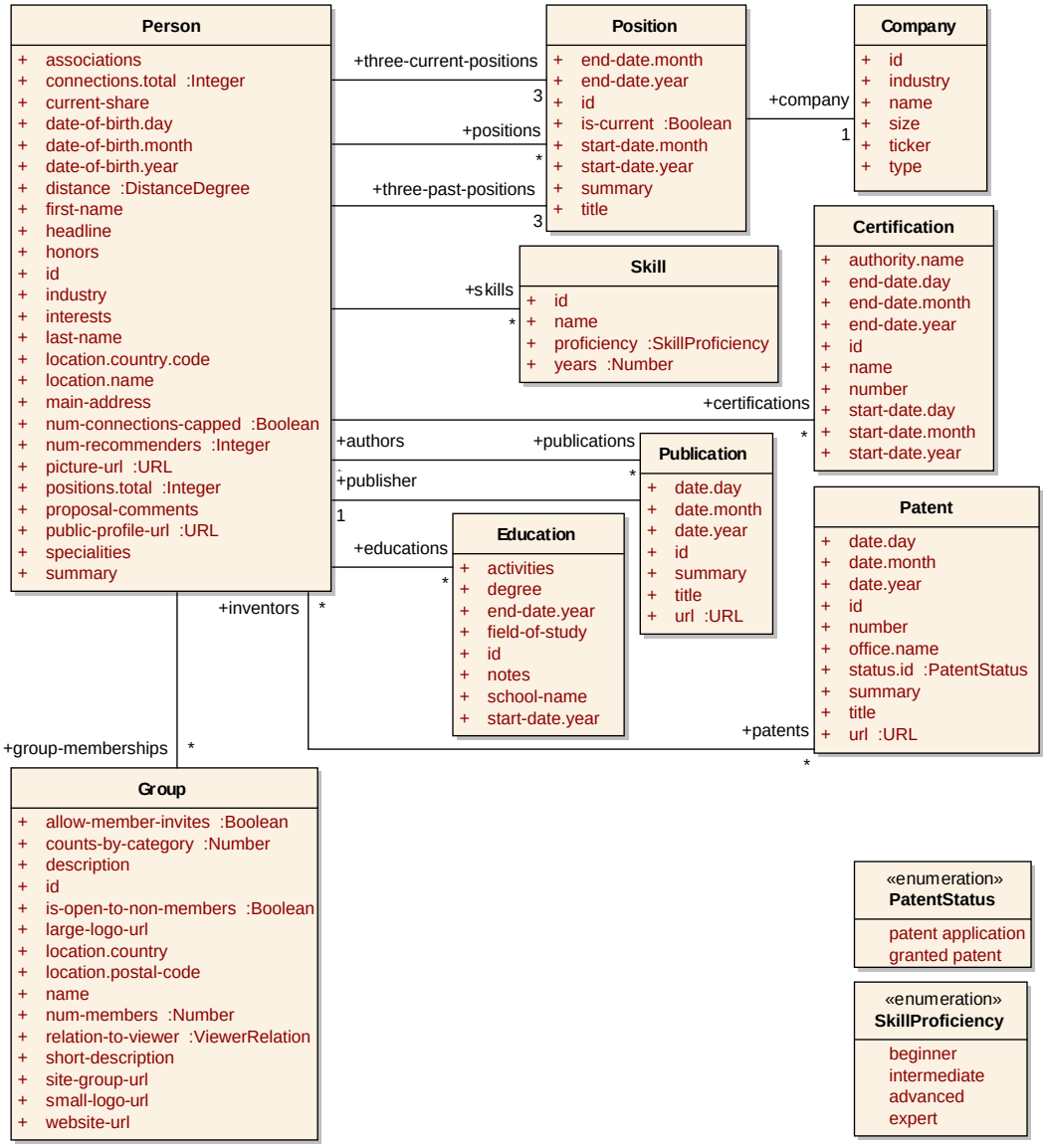


Figure 3.20: LinkedIn User Business

LinkedIn User Media (cf. Figure 3.21)

Data Access (GET Requests) Field selector must define all elements which are needed!

- Connections:
  - Jobs: http://api.linkedin.com/v1/people/ /job-bookmarks
  - Jobs: http://api.linkedin.com/v1/people/ /job-suggestions
  - Job by ID: http://api.linkedin.com/v1/jobs/ID

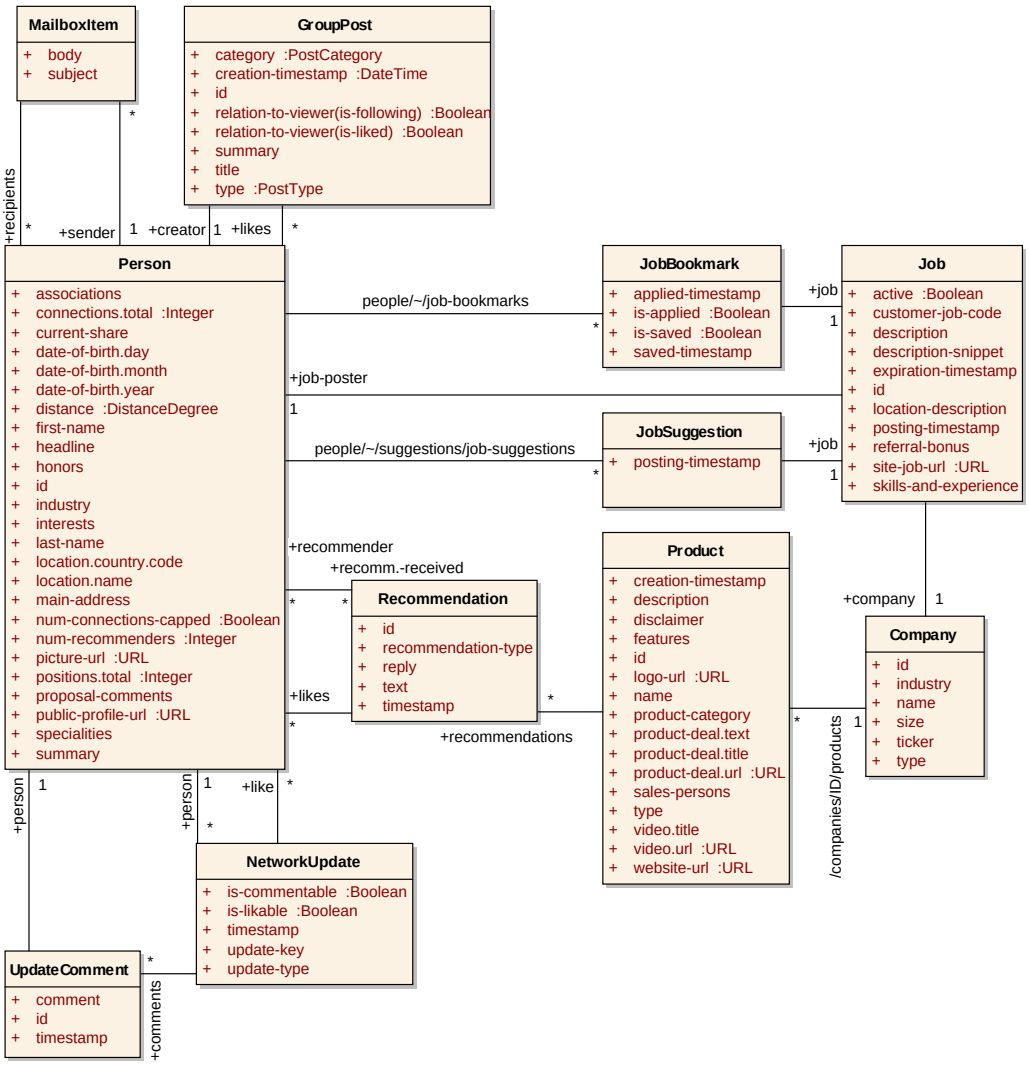


Figure 3.21: LinkedIn User Media

- Company by ID: http://api.linkedin.com/v1/companies/ID
- MailboxItems: http://api.linkedin.com/v1/people/ /mailbox
- Network Updates: http://api.linkedin.com/v1/people/ /network/updates
- Posting Updates: http://api.linkedin.com/v1/people/ /person-activities

Inline Classes of 'Person'

- Recommendation (recommendations-received)

Inline Classes of 'Recommendation'

- Person (likes, recommender)

Inline Class of 'NetworkUpdate'

- Person (like, person), UpdateComment (comments)

Inline Class of 'UpdateComment'

- Person (persons)

Inline Class of 'JobBookmark/JobSuggestion'

- job

Inline Classes of 'Job'

- Person (job-poster), Company

Connection of 'Company'

- Products: http://api.linkedin.com/v1/companies/ID/products

LinkedIn Group (cf. Figure 3.22)

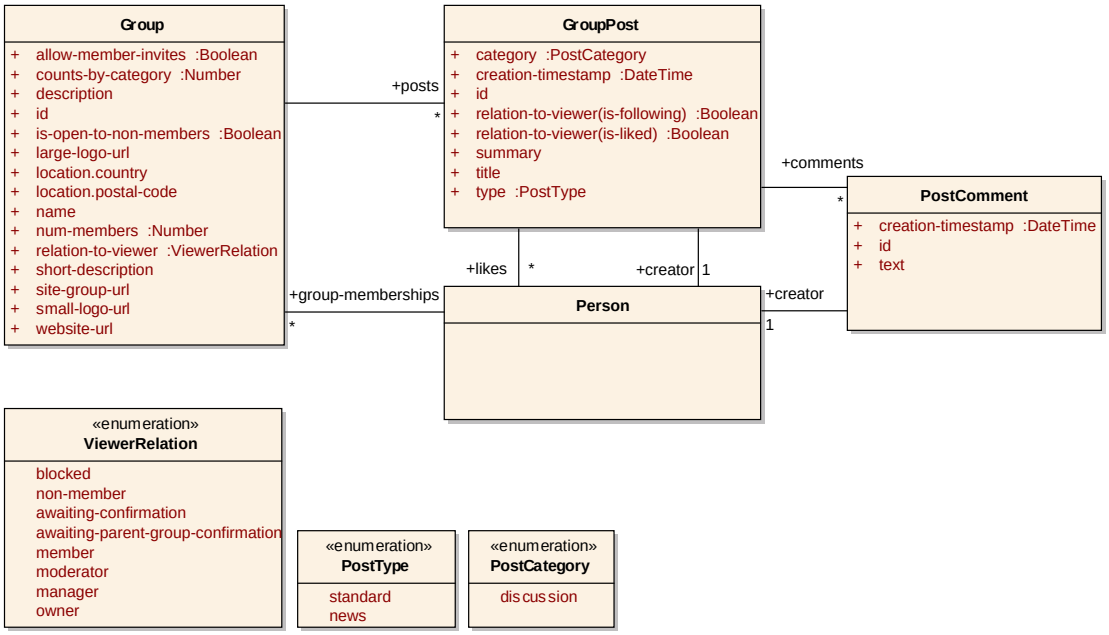


Figure 3.22: LinkedIn Group

Data Access (GET Requests) Field selector must define all elements which are needed!

- Group: http://api.linkedin.com/v1/groups/ID

Inline Classes of 'Group'

- GroupPost (posts)

Inline Classes of 'GroupPost'

- Person (likes, creator), PostComment (comments)

Inline Classes of 'PostComment'

- Person (creator)

3.3 Twitter

Twitter is a worldwide social network with focus on microblogging, launched in 2006 and available in 17 languages. Users can write 140 character messages called 'Tweets' on their public wall. When an user wants to share a tweet from another user, this is called 'Retweet'. Followers are users, which subscribe Tweets from an user. When user A follows user B and user B follows user A, these users are friends on Twitter. Friends also can write direct messages to each other which are not public.

Twitter has about 100 million active users and nearly 250 million tweets per day[23].

Twitter use oAuth 1.0a authorization, which will be explained in Section 4.3. The response format can be defined in the request URL, for example: https://api.twitter.com/1/users/lookup.json.

For manual re-engineering I used the documentation[24] of Twitter.

Basic Features and Characteristics

The following items are described in Chapter 2.3.

- **Connections**:
  - **Bilaterally 1:1 Connection:** Friend 3.23
  - **Bilaterally 1:n Connection:** -
  - **Unilaterally 1:1 Connection** Follower 3.23
  - **Unilaterally 1:n Connection** -

[23] `http://mashable.com/2011/10/17/twitter-costolo-stats/`
[24] `https://dev.twitter.com/docs/api`

- **Private User Profile**: Standard user profile after registration with limited information like current location, website, short biography , Figure 3.19

- **Non-Private Profiles**: Enhanced profile pages, `http://business.twitter.com/advertise/enhanced-profile`

- **Communication**: Private: Direct messages; Public: Tweets (status message) with pictures, videos, user mentions, hashtags and Retweets (share status messages from others)

- **Application Programming Interface (API)**:

  - **Authorization Type:** oAuth 1.0
  - **Internal Applications:** -
  - **External Applications:** Tweet, follow an user, tweet with hashtag and tweet to an user buttons are plugins[25] for websites as well as widgets[26] to present your profile or display search results and favorite tweets.
  - **Single Sign On:** possible with oAuth but not official supported

- **Customer Relationship**

  - **Import contacts** Find-Friends function with Gmail, Yahoo, Hotmail, MSN Messenger and AOL contact import function
  - **Newsletter** optional newsletters with information about product updates
  - **Comeback mails** 'Discover more on Twitter'-Mail with suggestions and instructions to reset your password if you cannot log in.

Twitter User Private (cf. Figure 3.23)

The limited information about the user is available in one class 'User'.

[25] `https://dev.twitter.com/docs/twitter-for-websites`
[26] `https://twitter.com/about/resources/widgets`

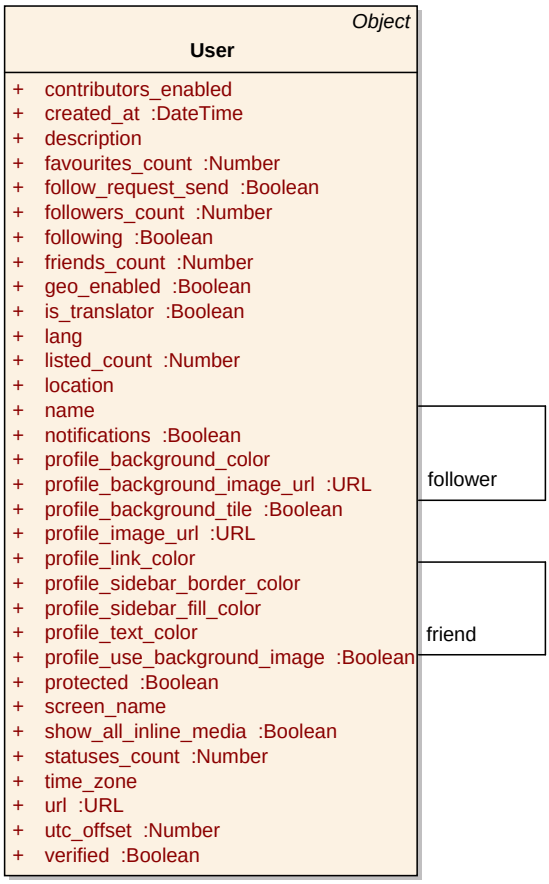


Figure 3.23: Twitter User Private

Data Access (GET Requests)

- Current User: https://api.twitter.com/1/account/verify_credentials
- User Search by screen name: https://api.twitter.com/1/users/lookup.json?screen_name=twitterapi or user_id=12345
- Show specified User by screen name or user id:

  https://api.twitter.com/1/users/show.json?screen_name=twitterapi or user_id=12345
- Follower:
  - List followers: https://api.twitter.com/1/followers/ids
  - List friends: https://api.twitter.com/1/friends/ids

Twitter User Media

The focus of Twitter is on microblogging, so the media objects comprise Tweets (Class Status in Figure 3.24). Tweets can consist of HashTags, where some topics can be tagged, UserMentions, where users can be tagged or include URLs or images (MediaEntity). When an user publishes a Tweet from another user, this action is called retweet.

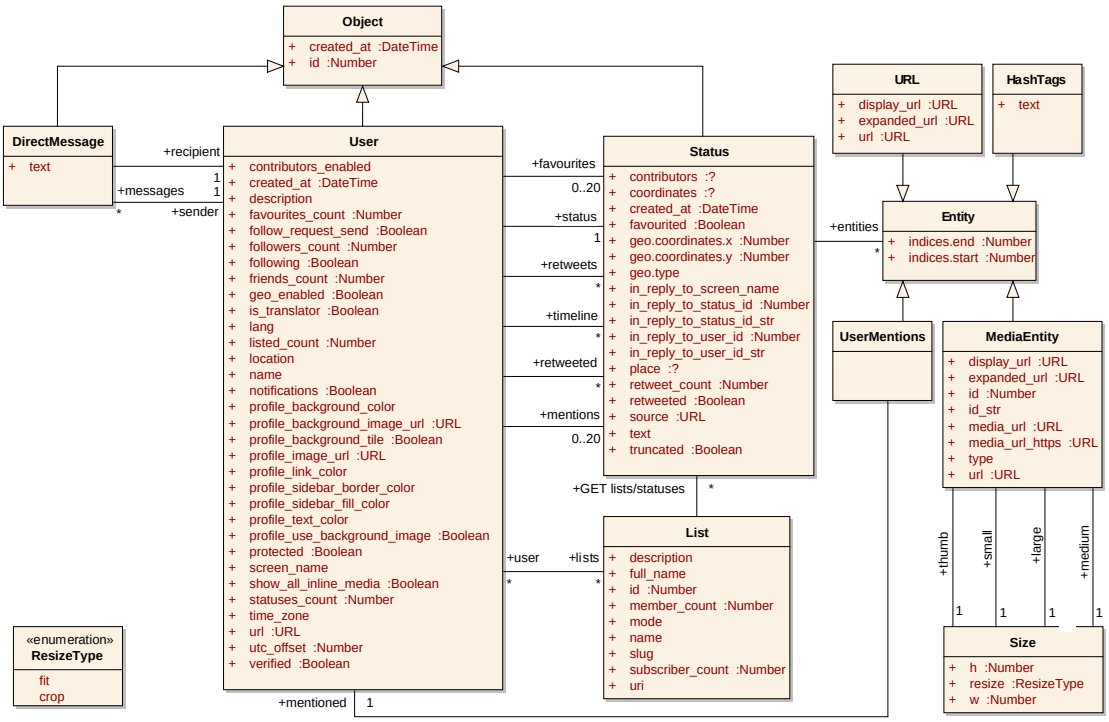


Figure 3.24: Twitter User Media

- Direct Messages: https://api.twitter.com/1/direct_messages
- Outgoing Messages: https://api.twitter.com/1/direct_messages/sent
- Show specified Message: https://api.twitter.com/1/direct_messages/show/:id
- Tweets (Status), https://api.twitter.com/1/...
  - Posted by the current user and user's they follow: statuses/home_timeline
  - containing @username: statuses/mentions
  - Posted by the current user: statuses/user_timeline
  - public: statuses/public_timeline
  - statuses/retweeted_by_me

- statuses/retweeted_to_me
- statuses/retweets_of_me
- specified by tweet id: statuses/show/:id
- retweets of specified tweet: statuses/retweets/:id
- users who retweeted a specified tweet: statuses/:id/retweeted_by

Twitter User System (cf. Figure 3.25)

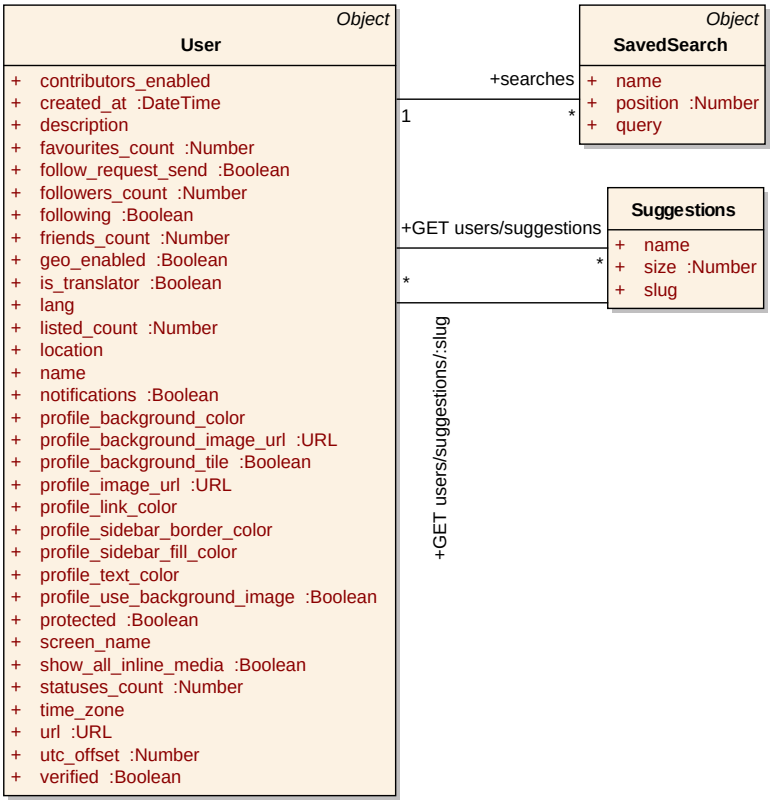


Figure 3.25: Twitter User System

Data Access (GET Requests)

- Saved Searches of the current user: https://api.twitter.com/1/saved_searches
- Retrieve information about a specified search: https://api.twitter.com/1/saved_searches/show/:id
- Get suggested users of the current user: https://api.twitter.com/1/users/suggestions
- Follower:

- List followers: https://api.twitter.com/1/followers/ids
- List friends: https://api.twitter.com/1/friends/ids

Twitter Trend

With Twitter Trend (cf. Figure 3.26) you can request trends of a specific geographic location, defined by WOEID, which is a Yahoo! Where On Earth ID. You will get an array of local trends with a name, URL and query parameter

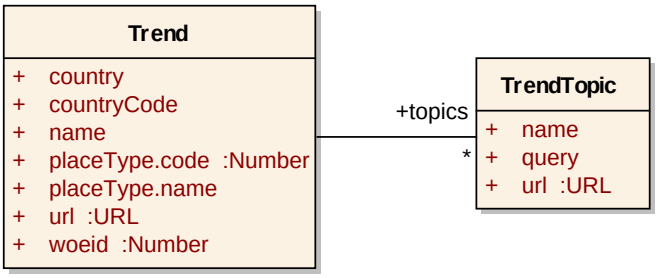


Figure 3.26: Twitter Trend

Data Access (GET Requests)

- Trend at WOEID: https://api.twitter.com/1/trends/:woeid
- Get locations where trends are available: https://api.twitter.com/1/trends/available

3.4 Google+

Google+ (GooglePlus) is a social network platform operated by Google Inc. Since the first testing phase and launch in June 2011, Google+ reached about 40 million users. After the early beta stadium, where some users have the possibility to invite up to 150 users, Google+ was opened to everyone in September 2011. Experts expect that Google+ will be a potential rival and competitor of Facebook. With the power and background of Google from the other popular products like Google Mail, Google Maps or Google Docs they have already a big community in the first weeks after release and lot of data about their users to use them in Google+. Suggestions to add users to circles come from the Google Mail contacts. The photo service Picasa is integrated in the profiles so that a new Google+ user can see the own photos in the new profile. The main feature is the news stream, where user can publish messages, photos or videos, tag them with the current locations and read the messages from other users, which are in so called circles, a type of friend list. Because of the big success of social games in Facebook, Google+ also provide a platform for social gaming companies like Zynga, well known from games like FarmVille or MafiaWars.

GooglePlus use oAuth 2.0 authorization, which will be explained in Section 4.3.

For re-engineering and discovering the GooglePlus data structure, I used a tool called „Google APIs Explorer “[27]. With this tool, provided by Google for all available APIs, developers are able to request an access token with previous selected scope or with private access. The API offer the possibility specify fields for a partial response. Another source of information is the official API documentation[28]. In my evaluation I noticed, that the documentation is not 100% persistent and a lot of documented fields were not available in the user interface or not accessible in the API.

Basic Features and Characteristics

The followingitems are described in Chapter 2.3.

- **Connections**:
  - **Bilaterally 1:1 Connection:** -
  - **Bilaterally 1:n Connection:** -
  - **Unilaterally 1:1 Connection** Circles
  - **Unilaterally 1:n Connection** -
- **Private User Profile**: Standard user profile after registration with basic private information and information about occupation, employment, education, hometown, current location, photos, videos and postings, (cf. Figure 3.27)
- **Non-Private Profiles**: Pages with more features than profiles like a +1 button and some differences in privacy and circle possibilities[29]
- **Communication**: Send an email with Google Mail, publish news with photo, link, video or location on the GooglePlus Stream for people who have the publisher in their circles, privacy settings for news postings to set which circles get access
- **Application Programming Interface (API)**:
  - **Authorization Type:** oAuth 2.0
  - **Internal Applications:** -
  - **External Applications:** Plugins like Google+ Badge[30], +1 Button[31] and a API for developing hangout applications[32]
  - **Single Sign On:** Google Friend Connect, oAuth

[27] http://code.google.com/apis/explorer
[28] https://developers.google.com/+/overview
[29] http://www.socialbrite.org
[30] https://developers.google.com/+/plugins/badge/
[31] https://developers.google.com/+/plugins/+1button/
[32] https://developers.google.com/+/hangouts/

- **Customer Relationship**

  - **Import contacts** automatic import of Google Mail contacts
  - **Newsletter** -
  - **Comeback mails** -

Google+ User Private

This are the documented elements of the private part of a users profile (cf. Figure 3.27), there will be a lot of other data, stored by Google about an user, but not accessible by developers.

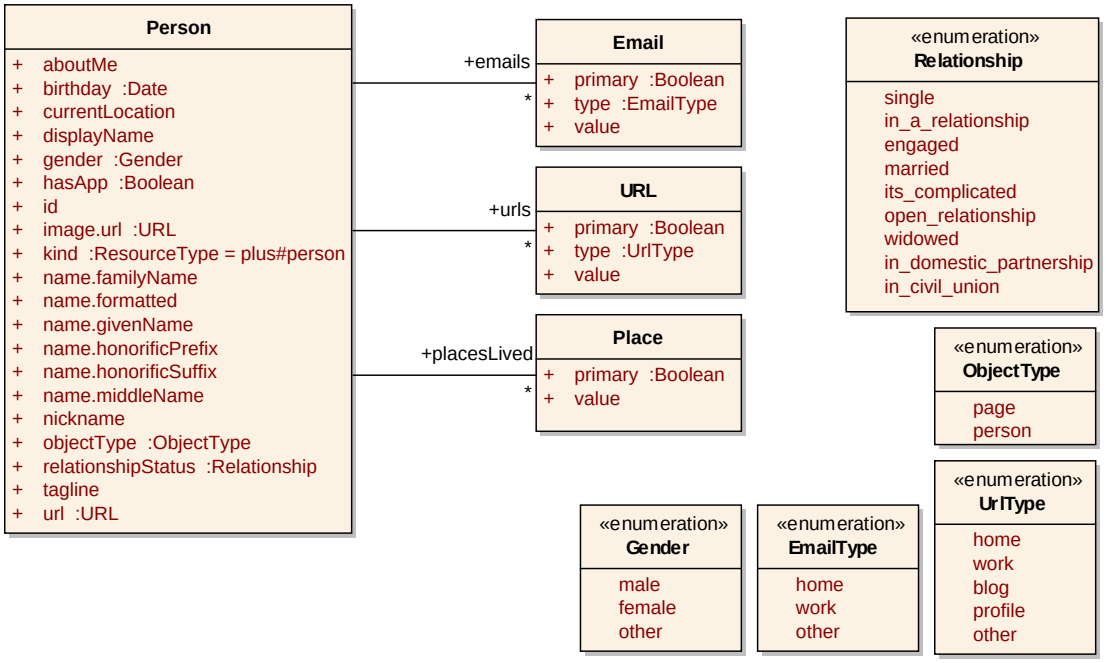


Figure 3.27: Google+ User Private

Data Access (GET Requests)

- current Person: https://www.googleapis.com/plus/v1/people/me
- other Person: https://www.googleapis.com/plus/v1/people/ID where ID is the ID of the desired person
- search Person: https://www.googleapis.com/plus/v1/people?query=...

Inline Classes of 'Person'

- Email (emails), URL (urls), Place (placesLived)

Google+ User Business (cf. Figure 3.28)

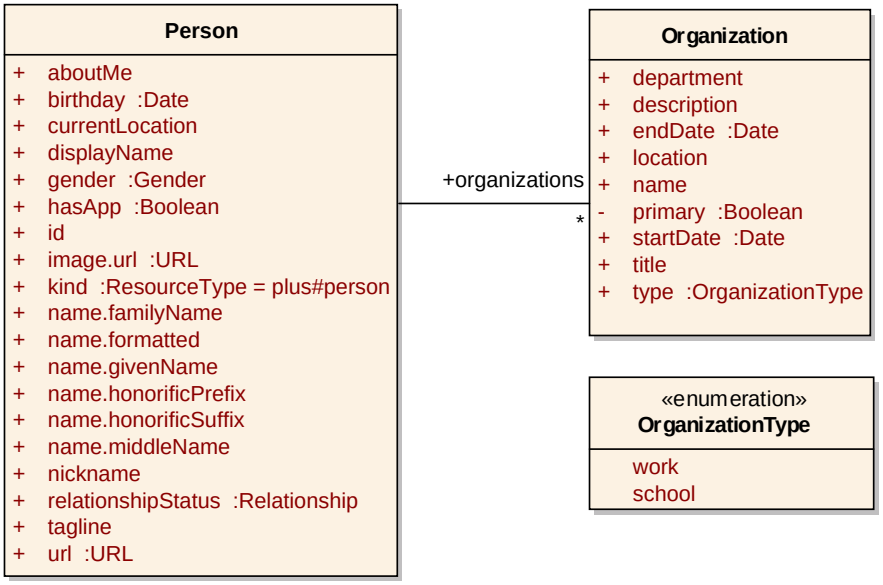


Figure 3.28: Google+ User Business

Inline Classes of 'Person'

- Organization (organizations)

Google+ User Media (cf. Figure 3.29)

Like in Facebook, users of Google+ have a main screen where postings are displayed, which is called 'Stream'. Users can publish messages with photos, videos, links and information about their current geographic location. For all messages privacy settings can be defined. There exist so called 'Circles', where a user can add other users. These circles are not documented or available by the API. Users are able to filter their own stream by selecting a created Circle, or publish a message just for a specified Circle.

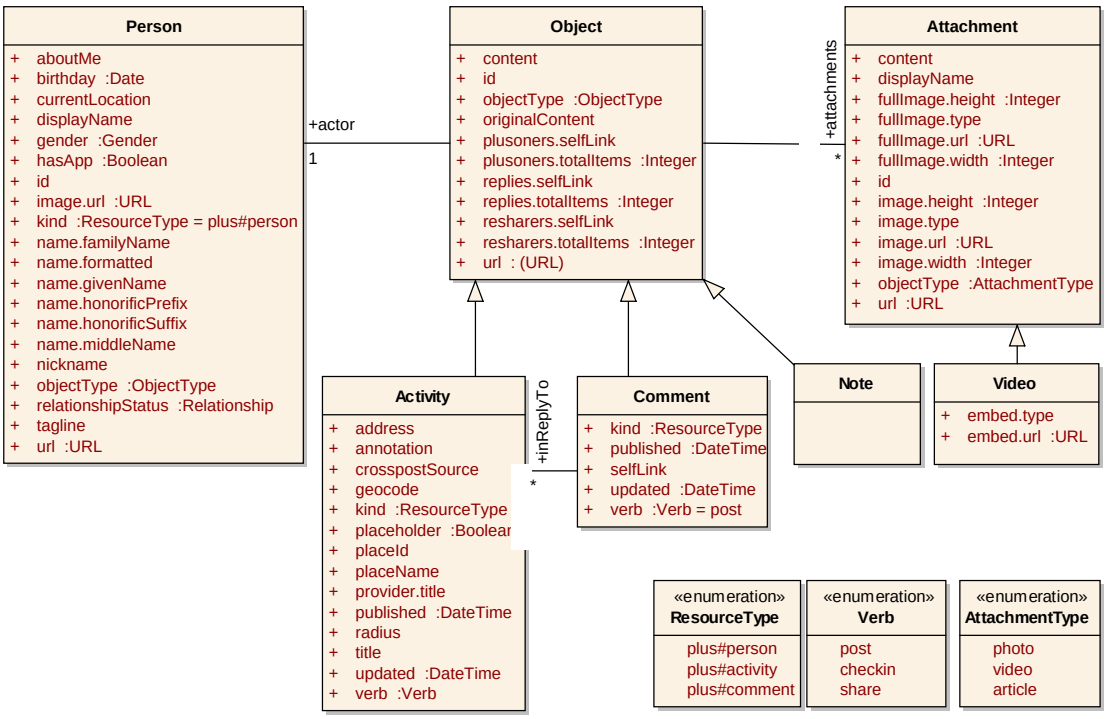


Figure 3.29: Google+ User Media

Data Access (GET Requests)

- Plusoner of an Activity: https://www.googleapis.com/plus/v1/activities/ID/people/plusoners
- Resharers of an Activity: https://www.googleapis.com/plus/v1/activities/ID/people/plusoners
- Comments of an Activity: https://www.googleapis.com/plus/v1/activities/ID/comments
- Search for an Activity: https://www.googleapis.com/plus/v1/activities?query=...
- Activities of an User: https://www.googleapis.com/plus/v1/people/ID/activities/public
- Activity by ID: https://www.googleapis.com/plus/v1/activities/ID
- Comment by ID: https://www.googleapis.com/plus/v1/comments/ID

Inline Classes of 'Object'

- Person (actor), Attachment (attachments)

Google+ Activity (cf. Figure 3.30)

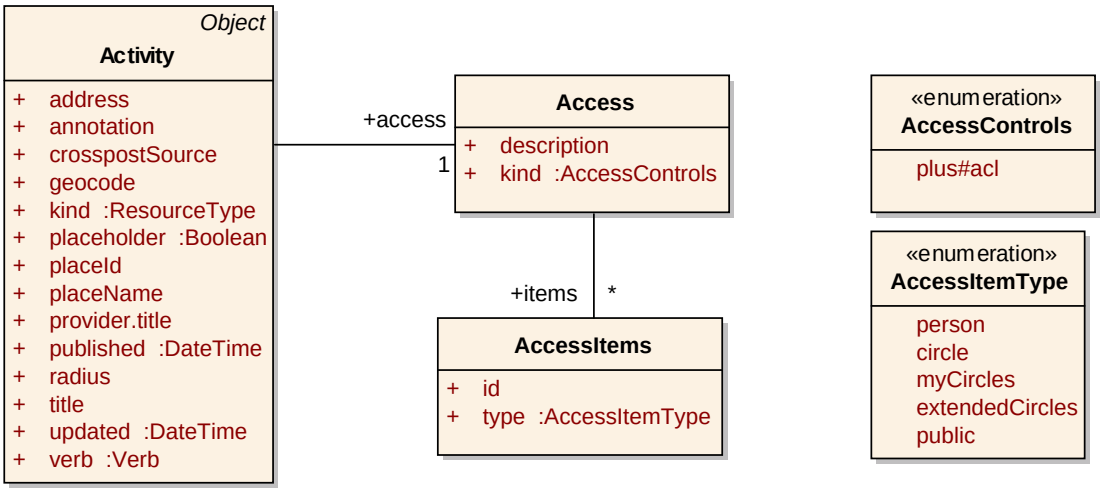


Figure 3.30: Google+ User Activity

Inline Classes of 'Activity'

- Access (access)

Inline Classes of 'Access'

- AccessItems (items)

CHAPTER 4

Accessing Social Data

In this chapter, the methods and standards to access the social data are explained. This is necessary for implementations like the Json2Ontology tool (cf. Chapter 5) or an social data adaptor.

4.1 REST

REST, which stands for Representational State Transfer was described in the dissertation „Architectural Styles and the Design of Network-based Software Architectures“ [9] of Roy Thomas Fielding in 2000. REST is an architecture, based on the HTTP protocol, which is used by many web services for a data interface because of the simple usage and style. A REST web service follows four basic design principles [1]:

- Use HTTP methods explicitly.
- Be stateless.
- Expose directory structure-like URIs.
- Transfer XML, JavaScript Object Notation (JSON), or both.

HTTP methods. There exist four HTTP methods, which are explicitly used by a REST web service. HTTP POST is used to create resources on a server. In the topic of my master thesis a resource may can be a new Facebook status message, which is created via the REST web service of Facebook. To get this message or any other resource back from the server, the HTTP GET method is used. The HTTP DELETE method will delete the status message or any other resource. Other HTTP methods like OPTIONS, HEAD, PUT, TRACE or CONNECT are not used in the discovered social network APIs but exist in the HTTP/1.0 RFC [18]

Be stateless. Each request from a client to a server must contain all information, so that the server is able to execute the request and optionally send back the response. There is no stored

context on the server. The sessions are kept entirely on the client, which make stateless web services more scalable. Requests like 'getNextPage' are not possible, we have to use a request with all information like 'getPage&id=2'.

Expose directory structure-like URIs. REST Web service URIs should be intuitive to the point where they are easy to guess. Think of a URI as a kind of self-documenting interface that requires little, if any, explanation or reference for a developer to understand what it points to and to derive related resources. To this end, the structure of a URI should be straightforward, predictable, and easily understood [1].

Transfer XML, JavaScript Object Notation (JSON), or both. The last principle recommend to use structured and machine-readable languages, which are developed for the usage over the internet to represent data structures like XML [3] or JSON. All of my selected social networks offer JSON as interchange format and so I decided to use it for the implementation of my transformation strategy. This is described in the following Section 4.2.

4.2 JavaScript Object Notation (JSON)

JSON[1], invented in 2006, is an open, text-based standard for a lightweight data-interchange format for the serialization of structured data. It is easy to read an write for humans and also easy to parse and generate for machines. REST web services[2] and by most of the well-known social network APIs use JSON as response format. With JSON we are able to represent 4 primitive types (strings, numbers, booleans and null) and two structured types (objects and arrays) as described in RFC 4627 [5].

Structural Characters (Tokens) in JSON

1. **[** left square bracket to represent the beginning of an array
2. **{** left curly bracket to represent the beginning of an object
3. **]** right square bracket to represent the end of an array
4. **{** right curly bracket to represent the end of an object
5. **:** a colon is used to separate between name and value
6. **,** a comma is used to separate values

[1] `http://www.json.org`
[2] `https://www.ibm.com/developerworks/webservices/library/ws-restful`

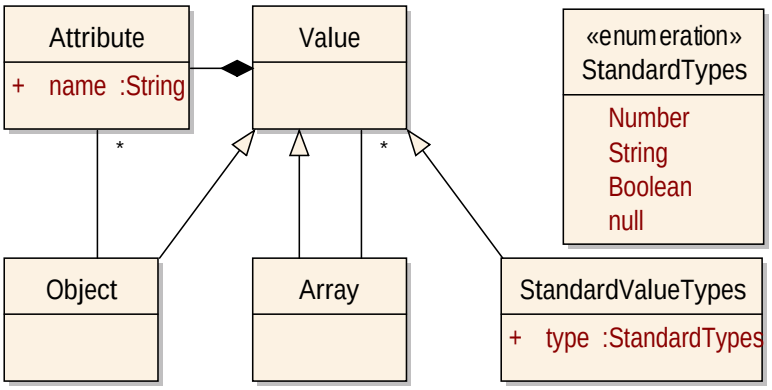


Figure 4.1: JSON Class Diagram

JSON Elements

- **Objects** contain attributes with name/value pairs to represent data
  - A *name* is a string (e.g. Listing 4.1 Line 2, *stringAttributeName1*.
  - A *value* must be an element like an object (e.g. Listing 4.1 Line 4-8), array (e.g. Line 10-17), number (Line 7), string (e.g. Line 2), or one of the three literal names 'false', 'null' or 'true'.
- **Arrays** contain zero or more *values* (e.g. Listing 4.1 Line 20) or *Objects* (e.g. Line 11-16)

Listing 4.1: JSON Example

```
1  {
2      "stringAttributeName1": "value1",
3      "newObjectName"   :
4      {
5          "stringAttributeName2": "value2",
6          "booleanAttributeName": true,
7          "numberAttributeName": 12345,
8      },
9      "newArrayName1":
10     [
11         {
12           "stringAttributeName3"   : "value3.1"
13         },
14         {
15           "stringAttributeName3"   : "value3.2"
16         }
17     ],
18     "newArrayName2":
19     [
20         "value_A", "value_B"
21     ]
22 }
```

4.3 Authentication and Authorization

To get access to social data, represented by JSON objects or XML an user must authenticate and authorize. With authentication an user can confirm the own identity. Authorization means, that an user verify that he is allowed to get access to a ressource.

OpenID

An open standard for decentralized authentication in the area of web platforms is OpenID[3]. An average user in the web use a lot of platforms and services. On each platform, a registration is needed which ends in multiple usernames and passwords. The solution is called Single Sign On and this may can be realized with OpenID. I try to briefly explain the OpenID systems and some terms. The Identifier is a secured (HTTPS) or non-secured (HTTP) URI, which identifies an user, (e.g. https://myopenid.com/andreas.munk). A web platform which wants to proof the end user need the users Identifier and is called Relying Party. The third important part is the OpenID Provider, a server where the OpenID authentication system is installed and the private information of the user is stored. The OpenID Provider specify the scope of the data which an user can define and provide an individual OpenID Identifier (URI). The user has to know the credentials for his OpenID profile for identification. The advantage is, that the user can use the credentials and stored information to register at web platforms, which support OpenID. An end user can freely choose which OpenID provider to use. Everyone can host an own OpenID service provider.

Besides the authentication, web platforms provide the possibility to get access to some private data. An widespread authorization system is oAuth[4]:

'OAuth provides a method for clients to access server resources on behalf of a resource owner (such as a different client or an end-user). It also provides a process for end-users to authorize third-party access to their server resources without sharing their credentials (typically, a username and password pair), using user-agent redirections.' [12]

Social networks use oAuth to grant access to their resources, delivered via an API. Developers of third-party applications have to redirect the user to an authorization flow, where the credentials are checked and the approval to get access to the social data of the user is granted or denied. The authorization flow depends on the used version of oAuth:

oAuth 1.0

oAuth 1.0 [12] was first released in 2007 and is used by social platforms like Twitter or LinkedIn. Developers have problems with this system because of the complexity. There are 3 actors: the user, the consumer (e.g. a social application) and the service provider (e.g. Twitter). The authorization flow is described in Figure 4.2: In the first step, the consumer ask for an unauthorized request token (1). This request include the following parameters: a consumer key provided by

[3] http://openid.net/
[4] http://oauth.net

the platform for the developer of an application, which want to get access to the data of the user. A signature method (HMAC-SHA1, RSA-SHA1, and PLAINTEXT) and the signature which is an encoded string containing all parameters. The last parameters are a timestamp, nonce (a random string uniquely generated by the client to allow the server to verify that a request has never been made before), optional the oauth version and a callback URL. The service provider verifies the signature and the consumer key (2) and generate a request key and request token (3). With these parameters, the consumer redirect the user (e.g. in a browser window) to the provider's authorization page (4). When the user grant the application, he got redirected to the callback URL of the application with an verifier parameter (5). Together with the request token the verifier is used by the application (6) to get an access token for further data requests, provided by the service provider (6).

For data requests (8), the consumer application also must do complex requests like mentioned before, consisting of the consumer key, access token, signature method end the encoded signature, timestamp, nonce and the optional oAuth version.

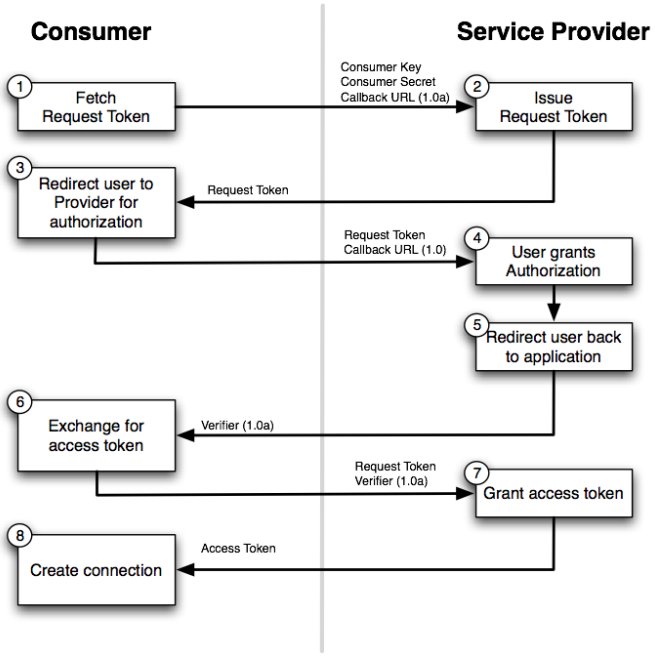


Figure 4.2: oAuth 1.0 Flow, springsource.org

oAuth2.0

In contrast to oAuth 1.0, the second version of oAuth is quite simple to use [13]. There are a lot of improvements like the support for desktop applications. With oAuth 1.0, desktop applications redirect the user to the service in a web browser where the user must authenticate and copy a generated token back to the application, where the authorization procedure continue. Beside

security issues, this flow is not very user-friendly. oAuth 2.0 support native applications like desktop or mobile applications with special consideration related to security, platform capabilities and end-user experience. The most important improvement is that developers no longer have to implement cryptography, the signatures are not so complicated and the authorization flow got reduced. There exist only one access token for data requests, which is granted for a user defined scope. So it is possible to limit the scope to some specific data elements.

In the authorization flow of oAuth2.0 the consumer start (1) with the redirect of the user to the provider for authorization, with the parameter client id of the consumer application, a redirect URI and an optional scope. The user can see the authorization screen of the provider and has to grant the permission for the data access (2). The user got redirected to the application (3) and the consumer get an application grant (e.g a code), which allows the application to request an access token(5).

For data requests (6), it is sufficient to concatenate the access token with the data request URI, where it is recommended to use a secured connection (SSL).

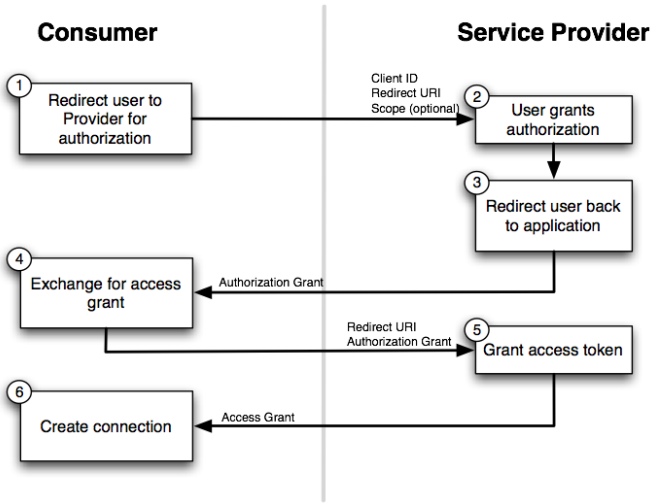


Figure 4.3: oAuth 2.0 Flow, springsource.org

Platform specific systems

Platform specific systems like Facebook Connect or the equivalent products of Google+, LinkedIn or Twitter combine both, an authentication and authorization system. Facebook Connect can be used by (web) developers to enhance the social experience of the visitors, log in with their Facebook account and have the same opportunities like with OpenID. But it is not only a single sign on system, Facebook Connect also use oAuth for the authorization of further data exchange. The extension of the data access can be set by developers, using the scope. With a full scope it is possible to get access to a very huge amount of data. This data access must be granted by the user

and is used in my implementation of the model-based reverse engineering tool 'Json2Ontology' (cf. Chapter 5.1).

CHAPTER 5

Json2Ontology Tool

The maintenance of social network ontologies and access adaptors is tedious. We are in an early stage of the development of social networks and so the expanding and frequent updates of the platforms infer a continuous change of data structure. New classes, attributes and relations were added, names changed or elements removed. The data structure and the request URL must be known to access the data.

5.1 TheHiddenU Ontology Language

Thus, one approach is a maintenance-free generator of the ontology of a social network, derived from the API response. For TheHiddenU, an individual ontology language (cf. Figure 5.1 has been developed to describe the ontology of social networks and further on the THU core ontology. The resulting ontology can be used to generate RDF code or java classes.

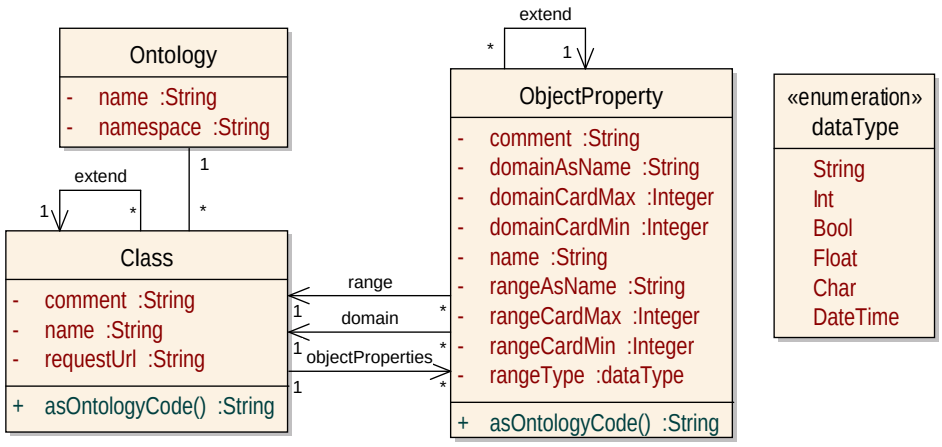


Figure 5.1: Class diagram: THU Ontology Language

Default Declarations

An ontology starts with the expression *ontology*, followed by the name of the ontology (Line 1 of Listing 5.1). In line 3 there is an *namespace* definition, where *http* as prefix and # as suffix will be automatically added when creating the corresponding rdf or in the future the java classes for the TheHiddenU project. From line 5 to 31 there are default declarations which are identical for every ontology, like basic classes, relations and the implemented data type literals String, Int, Bool, Float, Char and Datetime.

Listing 5.1: THU Ontology Language: Default Declarations

```
1  ontology theHiddenU {
2
3      namespace : "social-nexus.net/thu";
4
5      class Resource { }
6
7      class Literal { }
8      class StringLiteral extends Literal {
9          datatypeProperty value : STRING ;
10     }
11     class IntLiteral extends Literal {
12         datatypeProperty value : INT ;
13     }
14     class BoolLiteral extends Literal {
15         datatypeProperty value : BOOL ;
16     }
17     class FloatLiteral extends Literal {
18         datatypeProperty value : FLOAT ;
19     }
20     class CharLiteral extends Literal {
21         datatypeProperty value : CHAR ;
22     }
23     class DateTimeLiteral extends Literal {
24         datatypeProperty value : DATETIME ;
25     }
26
27     objectProperty Relation {
28         domain : Resource as inverseRelation;
29         range : Resource as relation;
30     }
31 }
```

Ontology-specific Declarations

The ontology code for a social network contains two important elements. The first element is the *class* element (line 1 of Listing 5.2) which can optionally *extends* other classes (line 8) and optionally can have a *requestURL* (line 11). The requestURL is important for the java class generation, because an adaptor must be able to know the URL where the social data can be requested. The second important element is the *objectProperty* followed by the name of this property starting with a capital letter (line 3), wherein the *domain* (line 4) and *range* (line 5) of the property can be defined. Both of these elements can optionally be extended with minimal and/or maximal cardinalities, expressed by *min=n* and *max=x* (line 18). The metalanguage

allows to define optional names for the range and domain attributes (lines 23, 24) after an *as* expression. All attributes of classes will be represented as *objectProperty*s.

Listing 5.2: THU Ontology Language: Ontology specific Declarations

```
1   class Agent { }
2
3   objectProperty AgendID {
4     domain : Agent;
5     range: IntLiteral[min=1,max=1];
6   }
7
8   class User extends Agent { }
9
10  class FBUser requestURL "https://request.url.com" { }
11
12  objectProperty UserName {
13    domain : User;
14    range : StringLiteral[min=1];
15  }
16
17  objectProperty AgentRelation {
18    domain : Agent[min=1,max=2];
19    range : Resource as agentRelation;
20  }
21
22  objectProperty KnowsUser2UserRelation extends AgentRelation {
23    domain : User[max=1] as isKnownBy;
24    range : User[min=1,max=100] as knows;
25  }
```

5.2 Implementation

The following information about the implementation focus on the extraction of the data structure of Facebook. The extensive amount of data collected by Facebook make this challenge interesting. Before I come to the transformation of JSON Objects (cf. Section 4.2) to THU Ontology Code, I will explain the basic method. One important part in social networks are the connections between objects. You can imagine this like a graph with edges (connections) and nodes (objects). Objects can be humans, things or some else artificial individuals. In this transformation I will use this edges to go from one node to another node or from one object to another object. One challenge of analyzing the data structure is to get all possible attributes. Not all social networks respond a full data structure, even if some attributes were not set by the user. The extraction of the class data structure of a Facebook User, who has not set all attributes, connections or relations will result an incomplete ontology. It is not sufficient to analyze only one User object to get all possible attributes, but use the connections and relations to analyze other (user)objects and add missing attributes to the database.

The first information we need is one or more start points (URL) where the tool begin the extraction. In Facebook, such a start point is `https://graph.facebook.com/me` where you can retrieve your personal user object. Besides some basic elements, this object may includes elements or objects which suggest that there is a connection to another object. Different to the

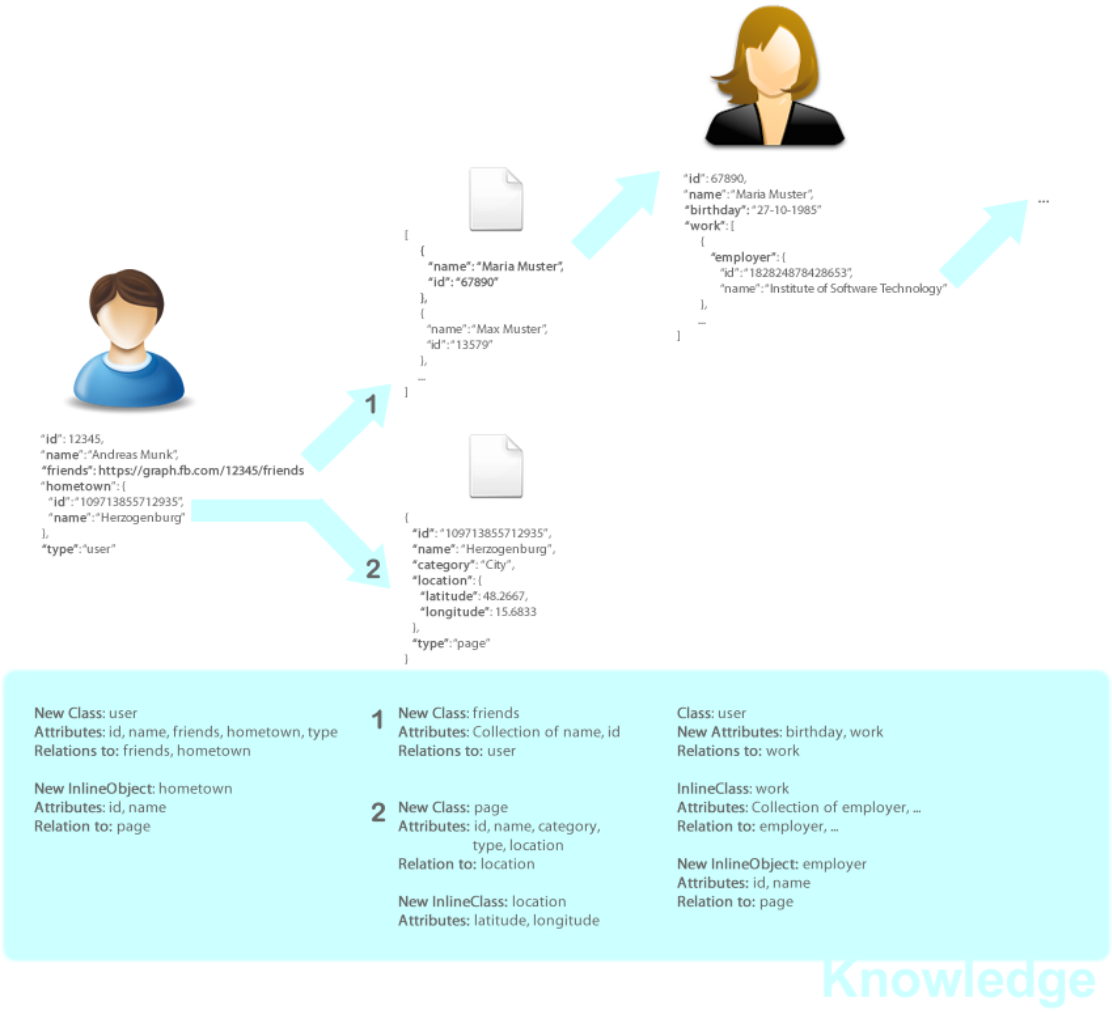


Figure 5.2: Knowledge Extraction

main purpose of such RESTful web services or APIs, not the data itself cause my interest. The important information is the data structure, the classes, request URLs, element names and the connections or relations.

To access information an access token is needed. In future work this access token may be retrieved in the easy way by a web application where the user can grant the permissions to access the data. The workaround in this prototype is to set this access token manually or, in case of oAuth 1.0 APIs (Twitter, LinkedIn) with instructions for the user. The user has to got to an generated URL, grant the permissions for the created test application and type in the PIN code, provided by the authorization flow, in the console. In the documentation of Facebooks Graph API[1] there are links to your own connections like friends or the home feed where you can cut a valid access token from the URL. This access token is not domain-based and so you can use it as your access token for tools like this, executed on localhost. In case of Google Plus, an independent access token will be provided by an API explorer[2]. Depending on the authorization method the start point URL and each other request URL must be signed, encrypted or concatenated with the access token or other secret keys.

Before analyzing the response of the start points, we have to define the name of the social network because of the implemented individualities and for the default declarations, explained in Section 5.1. This *OntologyName* also used for the namespace definition (i.e., for TheHiddenU *http://social-nexus.net/thu/adaptor/OntologyName*.

For the further usage I mapped the objects from the ontology language (cf. Figure 5.1) to the Java classes *Class* and *ObjectProperty*. To reduce execution time each analyzed URL will be saved in a HashMap with the reference to the derived class. For testing, the JSON response is cached to reduce the requests to the social network. The following subsection explain the different types of elements. The function for analysis and transformations get the response text and optional a class name and URL as parameters.

Transformation

In this section I will show some examples of a JSON response and explain how the response will be transformed to ontology code, implemented in the Json2Ontology tool.

Class

Analyzing a JSON object from a response will generate a class. To get the name of the class there are different approaches. Facebook supply a *type* attribute, which will be a good class name (e.g. type, page). Other approaches will be explained in the sections of the different class types. If there is no way to get a class name out of the response, the class is named „unknownClass“, concatenated with an increasing index.

[1] `http://developers.facebook.com/docs/reference/api/`
[2] `http://code.google.com/apis/explorer/#_s=plus&_v=v1&_m=people.get`

The following example show parts of the Facebook response of my user object with the URL „https://graph.facebook.com/1356535279“.

Listing 5.3: JSON Response: Attributes of a Facebook User object

```
{
    "id": "1356535279",
    "type": "user"
}
```

As mentioned above, the „type“attribute is catched for the class name. Other social networks would have different attributes, which indicate the class name. These attributes can be optionally added to prevent the tool naming a class „unknownClass“.

The second important thing is to discover the request URL. To use the ontology code for java code generation an independent request URL is needed. The analyze function detect that a value of an element in the JSON response is equal a part of the request URL, namely the „1356535279“from the „id“attribute.

To enable the generated java code to use a valid request URL to extract the data from a specific object (or user) the value of the id got replaced by the name of the element:
„https://graph.facebook.com/1356535279“-> „https://graph.facebook.com/__Attr__id__Attr__“

Listing 5.4: THU OntologyCode: Class definition of a Facebook User object

```
class user requestURL "https://graph.facebook.com/__Attr__id__Attr__" { }
```

Attribute

The standard name of an ObjectProperty will be composed of the class name (=domain), the key word „__Attr__“and the name of the property (e.g. „user__Attr__location“for the attribute location of the class user). This is necessary because of the individual defined ontology code metalanguage and the restriction that every attribute is represented by an ObjectProperty with an unique name. Further naming convention will be explained in the individual description of the possible objects in the transformation of JSON to OntologyCode.

Listing 5.5: JSON Response: Attributes of a Facebook User object

```
{
    "id": "1356535279",
}
```

Listing 5.6: THU OntologyCode: Attributes of a Facebook User object

```
objectProperty user__Attr__id   {
    domain : user;
    range : IntLiteral[max=1];
}
```

The range (type) of the objectProperty depends on the value of the attribute from the response. In this prototype there are the following literals implemented: Int, Bool, String. Possible

extensions in future work can be the analysis of values occurred to infer an enumeration. The cardinality will be max=1 if there is a standard value, no cardinality means a 1:n relation, for example if the value is an array of strings.

InlineClass

Not every class has its own request URL. Beside the main class type I mentioned before, I will define the new class type „InlineClass“. An InlineClass is responded inside of another (parent)class. The result is a new class with attributes (ObjectProperties) and a relation from the parent class to the new InlineClass. I will continue with an example of Facebook **Page** class, where the location attribute of the Facebook page of the Vienna University of Technology is listed.

Listing 5.7: JSON Response: Location Attribute (InlineClass) of a Facebook Page object

```
{
    "location": {
      "street": "Karlsplatz 13",
      "city": "Vienna",
      "country": "Austria",
      "zip": "1040"
    }
}
```

The value of the element *location* in the Facebook class *Page* is an object with the elements street, city, country and zip. This InlineClass itself cannot be accessed directly via an URL and therefore has no requestURL. The name of the new InlineClass is the name of the parent class, concatenated with the name of the attribute in the JSON response.

Listing 5.8: THU OntologyCode: Location Attribute (InlineClass) of a Facebook Page object

```
class page requestURL "https://graph.facebook.com/__Attr__id__Attr__" { }

    objectProperty page__Attr__location   {
        domain : page;
        range : page__location[max=1];
    }

//inlineClass
class page__location { }

    objectProperty page__location__Attr__street   {
        domain : page__location;
        range : StringLiteral[max=1];
    }

    objectProperty page__location__Attr__city   {
        domain : page__location;
        range : StringLiteral[max=1];
    }

    objectProperty page__location__Attr__country   {
        domain : page__location;
        range : StringLiteral[max=1];
```

```
24      }
25
26      objectProperty page__location__Attr__zip    {
27          domain : page__location;
28          range : IntLiteral[max=1];
29      }
```

InlineObject (Relation)

InlineClasses often have much more important information than their standard attributes. Facebook, for example, deliver with the user object hints for possible relations. Is the analyzed snippet of the JSON response an InlineClass and do there exist attributes like an „ID“(and on Facebook „name“), we can assume that there is a hidden relation. An important thing is that we have registered an InlineClass, because every standard class has of course an „ID“and this would not be a relation (to itself).

Here is an example of the JSON response of an user object with the element „hometown“.

Listing 5.9: JSON Response: Hometown Attribute (InlineObject) of a Facebook User object

```
1 "hometown": {
2     "id": "109713855712935",
3     "name": "Herzogenburg"
4 }
```

In addition to the procedure explained before with InlineClasses, which will generate the two ObjectProperties „id“and „name“, the id (and name) attribute initiate the tool to generate an additional ObjectProperty with a relation to the, up to now, unknown class. An advantage of the Facebook API is that there exist a standard request URL, so that every object with an ID can be requested via „https://graph.facebook.com/ID“. To extend the scope of the extraction and discover the unknown range class of the additional ObjectProperty, the ID is used to call the analyzing function for the hidden object (in this example the object with the id 109713855712935, which is the page of the city Herzogenburg). The analyzing function return the type of the response (in this example the class page).

The name of the new ObjectProperty which suggest the relation is a concatenation of „CurrentClassName__Attr__object“with the current class as the domain and the name of the analyzed related class as the range. The name of the new InlineClass is the name of the parent class, concatenated with the name of the attribute in the JSON response.

Listing 5.10: THU OntologyCode: Hometown Attribute (InlineObject) of a Facebook User object

```
1 class user requestURL "https://graph.facebook.com/__Attr__id__Attr__" { }
2
3     objectProperty user__Attr__hometown    {
4         domain : user;
5         range : user__hometown[max=1];
6     }
7
8 //inlineClass
9 class user__hometown { }
```

```
10
11      objectProperty user__hometown__Attr__id    {
12          domain : user__hometown;
13          range : IntLiteral[max=1];
14      }
15
16      objectProperty user__hometown__Attr__name    {
17          domain : user__hometown;
18          range : StringLiteral[max=1];
19      }
20
21      //InlineObject: this element refers to another object with more information
22      objectProperty user__hometown__Attr__object    {
23          domain : user__hometown;
24          range : page[max=1];
25      }
```

InlineClass with Relation

Similar to the explained strategy of the InlineClass and the InlineObject both can be combined if in the InlineClass exist „id“, „name“and other elements which are treated like standard attributes. In the following example an additional attribute „description“was found.

Listing 5.11: THU OntologyCode: Education Classes InlineClass with InlineObject

```
1  class user__education__classes { }
2
3      objectProperty user__education__classes__Attr__description    {
4          domain : user__education__classes;
5          range : StringLiteral[max=1];
6      }
7
8      objectProperty user__education__classes__Attr__id    {
9          domain : user__education__classes;
10         range : IntLiteral[max=1];
11     }
12
13     objectProperty user__education__classes__Attr__name    {
14         domain : user__education__classes;
15         range : StringLiteral[max=1];
16     }
17
18     //this element refers to another object with more information
19     objectProperty user__education__classes__Attr__object    {
20         domain : user__education__classes;
21         range : page[max=1];
22     }
23 }
```

Metadata

Some social networks like Facebook (or Viadeo) provide additional information about the requested object. By concatenating the Facebook request URL with the parameter „metadata=1“, the response consist a metadata object with two elements, represented by objects. The „connection“object with name/value pairs for every connection from the current JSON response (object)

to another JSON request url. The second element is the „fields“array with objects consisting „name“and „description“attributes. This data enable the Json2Ontology tool to comment the resulting ontology code with the description of each extracted element to enhance the understanding of the ontology code when it is used for future work.

Listing 5.12: JSON Response: Metadata of a Facebook User object

```
"metadata": {
      "connections": {
         "family": "https://graph.facebook.com/ID/family?access_token=...",
         ...
      },
      "fields": [
         {
            "name": "timezone",
            "description": "The user's timezone offset from UTC. Available only for the
                 current user."
         },
         ...
      ]
}
```

Connections

This information is used to navigate from one object to the corresponding relations by requesting the extracted URL and analyze the structure of the data of the related object. The name of the class is a concatenation of the name of the parent class(es) and the connection name (*family*). A connection includes the subclass paging and a data InlineClass, which you can see below. This InlineClass consist of several attributes, for example the „relationship“and an InlineObject to the corresponding user.

Listing 5.13: THU Ontology Code: Connection of a Facebook User object

```
class user__metadata__connections__family__data { }

    objectProperty user__metadata__connections__family__data__Attr__id    {
        domain : user__metadata__connections__family__data;
        range : IntLiteral[max=1];
    }

    objectProperty user__metadata__connections__family__data__Attr__name    {
        domain : user__metadata__connections__family__data;
        range : StringLiteral[max=1];
    }

    objectProperty user__metadata__connections__family__data__Attr__relationship    {
        domain : user__metadata__connections__family__data;
        range : StringLiteral;
    }

    //this element refers to another object with more information
    objectProperty user__metadata__connections__family__data__Attr__object    {
        domain : user__metadata__connections__family__data;
        range : user;
    }
```

Fields

The additional information about the attributes of an object can be used as comments for the THU Ontology Code, or in future work for description or JavaDoc of the generated java classes.

Listing 5.14: THU Ontology Code: Description of Elements of a Facebook User object

```
1    //The user's timezone offset from UTC. Available only for the current user.
2    objectProperty user__Attr__timezone   {
3        domain : user;
4        range : IntLiteral[max=1];
5    }
```

Configuration

For an optimal extraction within time or resource limits there exist some configuration possibilities.

Maximum Level

The function to analyze a JSON object is called recursive. Connections or relations between objects cause that the generator follow the path from an object A to another object B. If there are also connections from object B to an object C and so on it will be possible that the generator get in an endless loop.

The user object of Facebook is a good example for a loop. The connection of the attribute *significantOther* of user A, which represent a users girl- or boyfriend, refer to an user object B, where hopefully there will be a connection of this attribute back to user A. In the Education inline class there exist an array with User inline classes to represent users which were added by the user because they attend the same education (e.g. a high school class in 2006). This example show the possibility, that the generator follow the path from an user A, to another user B who attend the same education. This user B may have some connection like the *significantOther* attribute discussed before which cause the generator to go to an user C. If the maximum level configuration is set to 2, the generator will not access this user C and continue with the analysis of user B to reduce time and resources.

The configuration must be balanced to prevent getting into loops, but extract a complete data structure as good as possible.

Maximum Array Depth

Some elements like the Connections or the work InlineClass in Facebook consists of JSON arrays.

Let me explain the advantage of this configuration element by the friends connection of a Facebook user. When this connection got requested, the response are user InlineObjects for all friends of the current user. The challenge discussed in the Implementation Section that not every user

has filled in all elements and so this will result an incomplete data structures will be solved by extracting more than one user object. Another example is the work InlineClass in the user object. This attribute consist of zero or more array elements (class instances), which I will call the work InlineClass. This InlineClass has some fixed attributes like the employer, location or position but can be extended by additional elements like the start or end date of this work entry.

With the maximum array depth configuration it is possible to limit the analysis for the first, for example 5 objects in the returned JSON array to do a well-balanced analysis between completeness of the data and rational execution time.

Social Network Individualities

Every social network has their own characteristics. In the best case, the Json2OntologyCode tool is able to extract a complete data structure without any adaption. This work well if the response consist of, for example, a full user dumb with all information about the user together with InlineClasses with all information about the relations of the user.

The extraction of a Facebook ontology could be improved with the knowledge, that InlineObjects must have the elements name and id and could be extracted by using the standard URL together with the id value. To know that there exist some metadata with possible connections (relations) or description fields also enhance the tool.

Statistics

After a successful extraction the tool shows statistics about the run. These statistics include the number of analyzed request URLs, the analysis time, the number of extracted classes and attributes, the minimal, maximal and average response time.

Listing 5.15: Example Statistics of the Json2Ontology Tool

```
1  ------------------------------
2  Extraction complete...
3  *** 90 URLs in 40937ms analyzed
4  *** Classes: 36
5  *** Attributes: 155
6  ------------------------------
7  Statistics:
8  *** Requested URLs:    34
9  *** minResponse Time: 182ms
10 *** maxResponse Time: 1717ms
11 *** avgResponse Time: 330ms
12 *** Analysis Time:    29717ms
```

CHAPTER 6

Evaluation

In this section I will explain the evaluation of the Json2Ontology tool by comparing the result of the tool and the documented data structure of the social network. For this evaluation I used my own user accounts to analyze the data structure of the JSON response, returned by the API of Facebook, GooglePlus and Twitter. Information about LinkedIn at the end of this Section. Let me explain some basics of the evaluation, an detailed report will be in corresponding subsections of the social network.

Data Elements. One part is to get as much data elements as possible. The amount of data elements depend on two things. First, do the API return all data elements, even if they are not set by the user or do the API just return the elements with values. Second, is it possible to extract more than one objects of a class. For example, Facebook allow to navigate from an user to other objects, which my refers to other users. In this case, the user class may be extended by data elements, which are not set by the starting user but are set by other extracted users. To do this, it must be possible to recognize that a JSON object represent another user.

Relations. There exist different relations. When a JSON response of an user has an inline class about the hometown, this relation from the user to the hometown got extracted very well. Not so easy are relations, which are derived by additional knowledge. The tool must know the request URL of the remote object and extract an appropriate name of the related object. The data quality and quantity of the user account is very important, when relations can be extracted. If there are no or less relations, for example if the user has no friends, this relation cannot be extracted because the friends-object will be empty.

Class Diagram vs. OntologyCode. When you read the statistics in the following subsections, you will notice that there are a very high number of classes and attributes. The reason is the difference between a class diagram or documented data structure and the data structure of the JSON response or OntologyCode, respectively. Let me explain this with an example: in the Facebook class diagram, there exist a user, a page class and a relation between them called hometown.

The page class represent the city, which is the hometown of an user. So we have 2 classes and 1 relation apart from the number of attributes of the two classes. The extracted ontology code of the JSON response also extract the user and page class. The user class has an inline class called hometown, which will extracted as an relation to a class called user_hometown. This user_hometown class has two attributes, called id and name, which are in this case the indicator for a further extraction of the remote class page. This remote object will effect an relation from user_hometown to page and the extraction of the page class. Compared to the number of elements in a class diagram (2 classes, 1 relation), the extraction produce (3 classes, 2 relations and 2 additional attributes). Another reason for the difference is that some inline classes are equal to another basic class, but it is hard work to detect the similarity. In this prototype, this case will generate a new class for each inline class.

Configuration. To run the Json2Ontology tool, an individual configuration is needed. Besides the standard request URL and the start point(s), the oAuth version must be set to oAuth 1.0 or 2.0. As described in Section 5.2, a maximum level for the extraction has to be defined. In the evaluation I tried to set this level as high as possible and useful. A high level will increase the number of requests but will not compulsory improve the result. Another setting is called maximum array depth, described in Section 5.2.

6.1 Facebook

My test object in the evaluation of the tool with Facebook is my own user account. This account exist since June of 2009. I am very active on Facebook and so this are good prerequisites for the evaluation. Facebook only return data elements with values, so it will be possible, that the attributes in classes may not be complete. Relations can be extracted in Facebook. We know that an inline class with name and id attributes refer to another object, with perhaps more information than achieved by the inline class itself. A hometown inline class refers to a page class about the city, which represents the hometown of the user. To extract the extended class, we have to know the request URL. In Facebook, there exist a standard request URL, namely `https://graph.facebook.com`, where we know that the standard request URL concatenated with the id of the object will be the right request URL for the remote object. The

GlobalSettings

- oAuth Version: 2.0
- Preconditions for relation detection: id and name attribute in an inline class
- Standard URL: https://graph.facebook.com/
- Start Point: https://graph.facebook.com/me

Configuration tests. With the following statistics I will show the impact of the maximum level and maximum array depth setting in this case.

maximum Level	1
maximum Array Depth	1
Requests	1
Classes	24
Attributes	144

Table 6.1: Evaluation Facebook: Configuration Test #1

The extracted data structure of the test run 1 (Table 6.1) represent the user class and the inline classes of the user object like user__hometown, user__education. The quality is not sufficient, lot of detected relations have no correct range, for example the relation of the user__hometown class to the corresponding page class. The related object was not extracted because of the maximum level restriction. Because of the maximum array depth is set to 1, just the first element of arrays like the users education is analyzed. In the worst case, the first element of the user may contain less attributes than the second education element, which will not be reached. Therefore in the next step the maximum array depth is increased to 5.

maximum Level	1
maximum Array Depth	4
Requests	1
Classes	26
Attributes	153

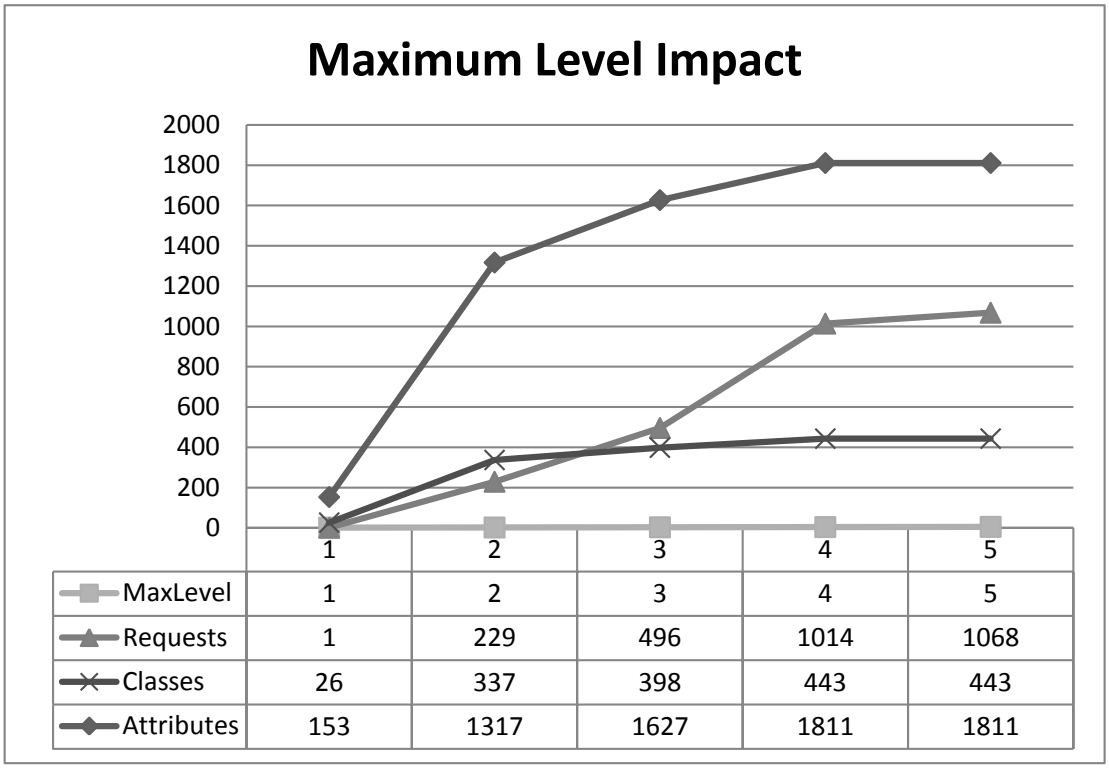
Table 6.2: Evaluation Facebook: Configuration Test #2

In this configuration test (Table 6.2), the maximum Array Depth was increased to 4 which cause that the first 4 elements in an array got analyzed. In this test run, a higher maximum array depth will have no effect. Comparing the OntologyCode of this run with the run before, there are 9 new attributes and 2 classes. I found the one missing attribute in the user__work class, where the end-date attribute (1 attribute) of a work array element did not appear in the first run. Another missing elements of the first run are the concentration and degree attributes (2 attributes) of the user__education, with the corresponding classes (2 classes) and the attributes in this new classes (name, id and object relation for each class(in sum 6 attributes)). Obviously, the first element in the users education array not contain these attributes. The relation of a new class, for example the education__degree did not have the right range which depends on the maximum level setting. The related object was not extracted and analyzed. In the next run the maximum level setting is increased.

maximum Level	2
maximum Array Depth	4
Requests	229
Classes	334
Attributes	1317

Table 6.3: Evaluation Facebook: Configuration test #3

Increasing the maximum level to 2 leads to better results, as is shown Table 6.3. All the remote objects of detected relations in the first level will be reached. This are, for example the page class (range of relations of classes like user__hometown, education__degree and much more) and the 37 connections, listed in the metadata of the user object, which result in at least 37 new classes but in fact there are a lot of inline classes in this classes and so the high amount of classes was generated. After hours of testing configurations with the goal to extract as much classes and elements as possible and the fear of being caught by the abuse systems of Facebook because of too much requests, I found a balanced configuration.



	1	2	3	4	5
MaxLevel	1	2	3	4	5
Requests	1	229	496	1014	1068
Classes	26	337	398	443	443
Attributes	153	1317	1627	1811	1811

Figure 6.1: Impact of the maximum level setting, maximum Array Depth = 4

maximum Level	4
maximum Array Depth	4
Requests	1014
Classes	443
Attributes	1811
minimal Response Time	202ms
maximal Response Time	17276ms
average Response Time	501ms
Analysis Time	5962ms

Table 6.4: Evaluation Facebook: Configuration test #4, final Statistics

With this run (Table 6.4), I compared the OntologyCode with the documentation[1] of Facebook to assess the result. In the documentation there are 23 objects (=classes). By analyzing the documentation a found additional 6 inline classes, for example user-work or user-education. These 29 classes should be at least extracted. The reason for the difference of the amount of the documented classes and the extracted classes is explained at the top of the evaluation section (Class Diagram vs. OntologyCode).

The result strongly depends on the user, who run the tool!

Sum of documented classes	29
Sum of extracted classes	23
documented - extracted ratio	79,31%
Sum of newly found classes	7
Sum of documented attributes	202
Sum of extracted attributes	161
documented - extracted attributes ratio	79,70%
Sum of newly found attributes	162
documented+new - extracted+new attributes ratio	88,74%
Sum of documented relations	161
Sum of extracted relations	97
documented - extracted attributes ratio	60,25%
Sum of newly found relations	23
documented+new - extracted+new attributes ratio	65,22%

Table 6.5: Evaluation Facebook: Result

The question is, what classes, attributes and relations are missing? 5 of the 6 missing classes are Achievement, Achievement(Instance), Order, Review and Subscription, which are not used by the authenticated user (my user account) and the Thread class is deprecated.

[1] `http://developers.facebook.com/docs`

There are 41 attributes missing: 27 attributes are not set by the user and so not responded by the Facebook API. Example: in my user account the attribute middle-name is empty because I have no middle name. This lack can be reduced by tool enhancements, like running the tool with more than one accounts, described in the Future Work Section. Some attributes are only accessible by the own user, so the navigation to other user will not improve the lack. 10 attributes were deprecated (documented but not used anymore), 4 attributes are administrative elements, which only are accessible by an administrator (for example of a page).

Where are the 64 relations? 4 are deprecated, 19 are not set by the authenticated user, 5 only for administrators of a page and 32 can not be reached because a limitation, that connections of objects, which are found itself in connections will not be analyzed. A try to disable this limitation was aborted after the run longs more than 10 minutes and 3.000 requests were produced. There will be possible enhancements in Future Work, Chapter 8.

7 newly found classes (or inline classes), 162 new attributes and 23 new relations demonstrate the potential of the tool. All this elements are not documented but available and might be interesting for developers.

6.2 LinkedIn

LinkedIn is not qualified for an evaluation because there is a basic barrier: the LinkedIn API force developers to use a so called field selector. With this field selector the developer must specify exactly which elements the API should respond. This fact make a tool like this useless, because the developer must know and list each element and the tool will never find any hidden element.

The tool could maybe used as Ontology Code generator, which will be the input for a Java Class generator for basic data.

6.3 Twitter

Twitter has a few basic differences to Facebook or GooglePlus. The authorization method is oAuth 1.0, all attributes were responded by the API, even if they were not set by the user and the bad news, there is no generic way to navigate to remote objects. Twitter offer the web service strictly in accordance with the rules of REST web services (Expose directory structure-like URIs, Section 4.1). So, the more start-points, the better quantity of the result.

My start-points are:

My User	https://api.twitter.com/1/account/verify_credentials.json?include_entities=true
My Timeline	https://api.twitter.com/1/statuses/user_timeline.json
My Retweets	https://api.twitter.com/1/statuses/retweeted_by_me.json?include_entities=true
My Friends	https://api.twitter.com/1/friends/ids.json
My Searches	https://api.twitter.com/1/saved_searches.json
My Suggestions	https://api.twitter.com/1/users/suggestions.json
A City	https://api.twitter.com/1/geo/id/674a484576216a45.json

Table 6.6: Start Points of Twitter Evaluation Run

Because of the fact that we cannot navigate, the maximum level setting is useless. In addition to this, the fact that there will be all attributes in the response, even if they are empty, an comparison of the documented elements and the returned elements to find elements which were not returned will not be very interesting. Twitter respond complete inline classes. When requesting an user, the last status message for instance is returned completely.

By comparing the documented attributes with the extracted attributes of the user class, 6 new elements were found. Like the other evaluated social networks, Twitter also has no correct documentation, probably based on the fast development iterations.

maximum Level	1
maximum Array Depth	2
Requests	7
Classes	36
Attributes	329
minimal Response Time	670ms
maximal Response Time	4377ms
average Response Time	1804ms
Analysis Time	347ms

Table 6.7: Evaluation Twitter: final Configuration, final Statistics

6.4 Google Plus

This social network is very new and of course, my user account is very new. In GooglePlus I run the tool with two start-points: `https://www.googleapis.com/plus/v1/people/me` which represent the own user object and `https://www.googleapis.com/plus/v1/people/me/activities/public`, which represent the own activities. To navigate to remote objects the tool has to detect an attribute named selfLink, which indicate a relation. The value of this attribute is the new request URL.

To get the comments (replies) of an activity, there exist a selfLink attribute with the value like this: `https://www.googleapis.com/plus/v1/activities/ACTIVITY_ID/comments`. After tests like described in the Facebook evaluation part, I used the result of the following configuration for the comparison:

maximum Level	4
maximum Array Depth	8
Requests	89
Classes	36
Attributes	146
minimal Response Time	215ms
maximal Response Time	1019ms
average Response Time	370ms
Analysis Time	546ms

Table 6.8: Evaluation GooglePlus: final Configuration, final Statistics

With the documentation[2] of GooglePlus I defined 11 classes and inline classes, which should be extracted through the defined start-points.

Sum of documented classes	11
Sum of extracted classes	10
documented - extracted ratio	90,90%
Sum of newly found classes	3
Sum of documented attributes	73
Sum of extracted attributes	48
documented - extracted attributes ratio	65,75%
Sum of newly found attributes	14
documented+new - extracted+new attributes ratio	70,45%
Sum of documented relations	15
Sum of extracted relations	14
documented - extracted attributes ratio	93,33%
Sum of newly found relations	3
documented+new - extracted+new attributes ratio	94,44%

Table 6.9: Evaluation GooglePlus: Result

The 1 missing class is the email class, which is documented but at the moment there is no possibility in the user interface to set e-mail addresses and die API do not respond any e-mail address, even the mandatory registration mail address is not in the JSON response.

14 attributes are documented, but not available in the user interface. One example are the rep-

[2] `https://developers.google.com/+/api/`

resentation attributes of a users name. I cannot find a corresponding field in the user interface to set, for example a honorific prefix of my name. 8 attributes are documented, I can set them in the user interface but the API do not offer them in the JSON response, for example the start- and end-date of an employment (class organization). 3 attributes are administrative ones which I could not access, for example a placeholder attribute in the ActivityItem class.

The 1 missing relation is the relation from the user class to the email class, as described above.

This evaluation show the the potential of a reverse engineering approach. There are a lot of elements, which are not documented but relevant for social applications. To increase the ratio of the documented elements which can be found, access to the data of users with a high social activity on the platforms are needed.

CHAPTER 7

Related Work

Social media is a very hot topic in science and so there are many publications about social networks. In the area of model transformations, there are some approaches with social media aspects like API2MoL [15].

Api2mol (API to Metamodel Language) [14] is a Domain Specific Language (DSL) which allow to define a mapping between an API and a metamodel (i.e., API classes and metamodel elements). The mapping definition is used to project models form/into APIs, that is, to inject/extract models by calling to API methods. With API2MoL, developers can define the mappings between an API and a metamodel (i.e. mappings between API classes and metamodel elements). It is also possible to automatically create a metamodel definition directly from the API information if not existing metamodel is already available. These mappings can then be used at anytime one need to bridge the gap between software and models.[1]

This scientific project has some similarities with the Json2Ontology tool, but does not deal with authorization or navigation in REST web services and, of course, the output of the generator is not the expected output of TheHiddenU project.

In [2], social data is continuously analyzed to observe trends in social networks. The difference to this thesis is the type of data and the goal of the analysis. The approach in the cited paper use real time values and compare changes to observe trends. Instead of this, the Json2Ontology tool perform the analysis out of static data, analyze the data structure and produce a data schema.

Other projects focus on transforming given JSON objects to Java classes like 'jsonschema2pojo'[2], which also use the structural rules of JSON like I do. 'json gen beta'[3] is a web based code generator that parses JSON files. Both projects make no requests or navigation to other JSON objects

[1] http://modelum.es/trac/api2mol
[2] http://code.google.com/p/jsonschema2pojo
[3] http://jsongen.byingtondesign.com/

of a REST web service.

The main aspect to make research and develop an own generator was the precise requirements of TheHiddenU project and the need of generating an input for java class generation in the own THU ontology language. Another requirement will be the integration of the collected information.

Social data exchange is of course, only a subproblem of data exchange in general which leads to a much broader research field, namely information integration, with database integration as its most prominent protagonist. Database integration has a long history, e.g., one of the earliest systems for realizing information integration was the EXPRESS system, developed in the 1970s [19]. Integration scenarios in the data engineering field mostly concern integrating different local schemas (e.g. of social networks) into one global schema (e.g. a core ontology for social user profiles), however also data exchange between a source schema and a target schema have been investigated, e.g., between relational, object-oriented, and XML databases and data warehouses [6, 11]. In this thesis, I have focused on bridging technological heterogeneities between REST Web services and social data ontologies, needed as a major building block for analyzing social data.

CHAPTER 8

Conclusion and Future Work

8.1 Conclusion

The first goal of this thesis was to identify the features and to come up with a definition of available social networks. The most important things in such networks are actors and the connections between them. Successful social networks try map the real life in a specific topic to web based platforms, which help people to connect and share information. I defined some basic categories and features of social networks like the types of connections, profiles, communication and possibilities for developers and evaluated four social networks, which were selected in accordance to the TheHiddenU project team.

With an short overview about the social networks Facebook, LinkedIn, Twitter and GooglePlus together with the classification I started the evaluation of the selected social networks. The main task of Chapter 6 was the creation of class diagrams out of the API documentation to represent the data structure and help the THU team to get an overview of the possibilities for profiling. To compare the data the diagrams have been split up into topic related an comparable diagrams. This resulted in more than 30 diagrams as can be seen in Chapter 6.

To get access to the values of the described data structure, I studied the APIs of the social networks and investigated the fundamentals of REST web services as well as the different authentication and authorization methods like OpenID or oAuth. This helped to develop a first adaptor prototype for TheHiddenU, where the authorization and data extraction is an elementary task. LinkedIn and Twitter use oAuth 1.0, which is much more complicated than oAuth 2.0 because of the necessary encryption and the difficult signature generation. To understand the responses of the web services I examined the JavaScript Object Notation (JSON). Most of the social networks use JSON for data representation of responses. With the knowledge about JSON, I was able to pursue an idea about transformation of JSON response into a domain specific language.

The idea for the Json2Ontology tool, described in Chapter 5, was born in Linz during a project

meeting. I thought that it would be interesting to create a system which analyzes the response of the API and generate a code for further usage. This THU ontology code was used to describe the core ontology of TheHiddenU and for generating the corresponding Java classes. With a few extensions, I could use this code as output of my tool. Another good reason to develop a tool like this was the fast moving development of social networks and the bad documentation. Not every element which was responded was also documented. My interests were not on the values of a social networker, but rather on the data structure of the response. The values only were used to detect the data type of the data element. The first transformation rules were to create a class out of a JSON object with the attributes out of the object elements. Because of the big scope and the good preconditions of Facebook, I decide to use Facebook as my test platform. With oAuth 2.0 it was easy to deal with access tokens for the data access. Another advantage was that Facebook supply metadata information about relations of the requested object which can be used for navigation. I started an my own user object and navigated through related objects, accessed via a standard request URL concatenated with the ID of the object and the information about the connections. This ended in an aborted test run with more than 3000 requests to Facebook. With the possibility to configure the maximum extraction level, this could be shortened to a run with more than 1000 requests, 443 classes and 1811 attributes, which were extracted out of my own user as a starting point. This amount of elements is not direct comparable with the classes and elements of a class diagram, because the JSON response not only supply a relation but give information about the related class, which I called InlineClass. Simple relations were detected as InlineObjects. The amount of extracted elements strongly depend on the authorized user. If there are just a few attributes and relations responded by the API, the tool wont produce a good ontology. The navigation from one object to another object is one of the most important tasks, where a lot of improvements could be done in future work. I tested the tool with the other social networks Twitter and GooglePlus with good basic results but with limitations with the navigation because of the missing standard request URL and the leak of information about connections. LinkedIn was not applicable because developers must specify a so called field selector, where all desired attributes must be listed. With this barrier, unknown elements can not be extracted and the navigation would be difficult. In Chapter 6 I evaluated the tool against the information from the documentation of the social networks of Chapter 6. With the comparison I noticed a lot of possibilities to enhance the basic features of the tool in future work.

8.2 Future Work

A tool like Json2Ontology is much more complex than I thought at the beginning of the work. To read a JSON object, create the corresponding classes and attributes is well done by the tool. The basic data types of the attributes like String or Integer can be detected by basic java features.

Enumerations. To detect enumerations an heuristic analysis of the responded values of attributes must be done. The question is, what is a good indicator for an enumeration? Besides the saved information about the data structure, the values of the attributes must be saved and analyzed. Therefore, a lot of data samples are required to make a detection of an enumeration. A problem could be that free text values in social networks may contain same values but are

free text. Another aspect is, that a detected enumeration may contain more valid values than extracted and these missing values may be needed in use cases like in TheHiddenU [16]. To get more data samples and enhance the quality of the result, I will come to another nice feature in the future.

Navigation. To enhance the outreach of the tool, the navigation is very important. Some social networks like Facebook provide a good basis to navigate from one start point to other objects because of the connection values in the metadata of a JSON object, which hint that there is a possibility to navigate, or the standard request URL. In other social networks it is more complicated to navigate and so there always are possible enhancements or changes to make the tool more generic.

Ontology Updates. At the moment a new ontology code is generated at each run of the tool. As described in the evaluation of the tool (Chapter 6), the result depends on the authenticated user who start the tool. The perfect case will be an user, who had set all fields in the profile because we cannot assume that the REST web service respond all possible attributes, even if they are not set by the user. A possible feature will be that a saved ontology will be used in a new run with another user, extended by new attributes, relations or classes which were responded by the web service. Different users have different data in their profiles. Every run with a new user will make the resulting ontology more complete. A important task is how to handle missing elements and detect if they are just missing in the current response or missing because of data structure updates of the social network or web service provider.

Ontology Updates on the Fly. For projects like TheHiddenU [16], where hopefully a lot of users will take the advantage of the platform, a lot of requests will be sent to web services because the data is needed for profiling. Besides the profiling, the responded data could be analyzed like in Json2Ontology to make ontology updates as described before on the fly. So, every user will enhance the ontology.

Java Generation on the Fly. Together with the updates of the ontology while profiling, the java class generation is a possible feature. Like generating ontology code, a java code generation function will reduce time and increase the amount of accessible data.

Class Diagram Generator. With class diagrams we can enhance the understanding for new collaborators with minimal knowledge about a special data structure or social network. In my thesis it was hard work to create class diagrams, especially in such huge social networks like Facebook. An automatic generation of class diagrams will be a nice feature to keep the class diagrams up to date, because of frequently data structure changes. The most important thing is to keep in mind that the result of the JSON response analysis is not the same es the information which is used in class diagrams. JSON responses have much more elements, because of inline classes which are separate classes but responded inside another class or relations, which are responded as inline objects with a few basic information about the related class. This elements must be cleaned before generating class diagrams to keep the diagrams small and simple.

Web Platform. To make it possible to incent users which might help creating a better ontology, a web platform or application will be a good additional feature, where the user can grant the permission to access the data of a specific social network and provide the data for data structure analysis. The developed Json2Ontology prototype is command line based and so just for skilled users.

List of Abbreviations

API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
SMS	Short Message Service
SSL	Secure Sockets Layer
THU	TheHiddenU Project
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	Extensible Markup Language

List of Figures

List of Tables

Listings

Bibliography

[1] IBM Alex Rodriguez. Restful web services: The basics. http://www.ibm.com/developerworks/webservices/library/ws-restful/. [Online; accessed 10-November-2011].

[2] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. Continuous queries and real-time analysis of social semantic data with c-sparql. In *SDoW2009*, volume 520 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009. online http://ceur-ws.org/Vol-520/paper02.pdf.

[3] Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, Eve Maler, and Francois Yergeau. W3c: Extensible markup language (xml) 1.0 (fifth edition). http://www.w3.org/TR/2008/REC-xml-20081126/, 2008. [Online; accessed 20-November-2011].

[4] The Nielsen Company. Global faces and networked places - a nielsen report on social networkings new global fingerprint. http://blog.nielsen.com/nielsenwire/wp-content/uploads/2009/03/nielsen/_globalfaces/_mar09.pdf, 2009. [Online; accessed 11-July-2011].

[5] D. Crockford. The application/json media type for javascript object notation (json), request for comments: 4627. http://tools.ietf.org/html/rfc4627, 2006. [Online; accessed 20-November-2011].

[6] AnHai Doan and Alon Y. Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine*, 26(1):83–94, 2005.

[7] Jim Dolwick. 'The Social' and Beyond: Introducing Actor-Network Theory. *Journal of Maritime Archaeology*, 4(1):21–49, June 2009.

[8] Anja Ebersbach, Markus Glaser, and Richard Heigl. *Wiki : Web Collaboration*. Springer, November 2005.

[9] Roy T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, Irvine - Irvine, CA 92697, USA, 2000.

[10] Object Management Group. Uml specification: Omg unified modeling language (omg uml). http://www.omg.org/spec/UML/2.3/Superstructure/PDF/. [Online; accessed 21-October-2011].

[11] Laura M. Haas. Beauty and the beast: The theory and practice of information integration. In *11th Int. Conf. on Database Theory (ICDT'07)*, 2007.

[12] E. Hammer-Lahav. The oauth 1.0 protocol, request for comments: 5849. `http://tools.ietf.org/html/rfc5849`, 2010. [Online; accessed 20-November-2011].

[13] E. Hammer-Lahav, D. Recordon, and Hardt D. The oauth 2.0 authorization protocol, draft-ietf-oauth-v2-22. `http://tools.ietf.org/html/draft-ietf-oauth-v2-22`, 2011. [Online; accessed 20-November-2011].

[14] Javier L. Izquierdo, Frédéric Jouault, Jordi Cabot, and Jesús G. Molina. API2MoL: Automating the building of bridges between APIs and Model-Driven Engineering. *Information and Software Technology*, October 2011.

[15] Javier Luis Canovas Izquierdo, Frederic Jouault, Jordi Cabot, and Jesus Garcia Molina. Api2mol: Automating the building of bridges between apis and model-driven engineering. *Information and Software Technology*, (0):–, 2011.

[16] Gerti Kappel, Johannes Schönböck, Manuel Wimmer, Gabriele Kotsis, Angelika Kusel, Birgit Pröll, Werner Retschitzegger, Wieland Schwinger, and Stephan Lechner. Thehiddenu - a social nexus for privacy-assured personalisation brokerage. In *Proceedings of the 12th International Conference of Enterprise Information Systems (ICEIS'2010)*. INSTICC Press, 2010.

[17] E. Kapsammer, S. Mitsch, B. Pröll, W. Retschnitzegger, W. Schwinger, M. Wimmer, and M. Wischenbart. A first step towards a conceptual reference model for comparing social user profiles. In *UWeb 2011 International Workshop at ESWC 2011 on User Profile Data on the Social Semantic Web*, Heraklion, Crete, Greece, 2011.

[18] J.Mogul H.Frystyk L.Masinter P.Leach T.Berners-Lee R.Fielding, J.Gettys. W3c: Hypertext transfer protocol – http/1.1, request for comments: 2616. `http://www.w3.org/Protocols/rfc2616/rfc2616.html`, 1999. [Online; accessed 20-November-2011].

[19] Nan C. Shu, Barron C. Housel, Robert W. Taylor, Sakti P. Ghosh, and Vincent Y. Lum. EXPRESS: A Data EXtraction, Processing, amd REStructuring System. *ACM Trans. Database Syst.*, 2(2):134–174, 1977.

[20] Jenna Wortham. After 10 years of blogs, the future's brighter than ever. `http://www.wired.com/entertainment/theweb/news/2007/12/blog_anniversary`, 2007. [Online; accessed 19-October-2011].