Connecting Customer Relationship Management Systems to Social Networks

Hanno Zwikstra, Frederik Hogenboom, Damir Vandic, and Flavius Frasincar

Abstract As the popularity and the commercial potential of social networks such as LinkedIn and Facebook increase, we present a framework that aims to reuse social networks data within a customer relationship management (CRM) application. The framework has been implemented in *LinkedInFinder* that pulls data from LinkedIn into the Microsoft Dynamics CRM system. Our proof-of-concept implementation demonstrates the use of the proposed framework, based on a use case to find second-degree connections within one's network that work at a specific company of interest. A survey amongst target users suggests that the application is useful and adequately designed for the intended use.

1 Introduction

With the advent of Web 2.0 [6], there has been a growing importance regarding the social aspects of the Web. Even though social networks have been existing as long as there have been societies, digital networks – such as Twitter, Facebook, and LinkedIn – experienced a substantial growth over the last decade. Due to their promising commercial potential, there has been put an increasing amount of effort and research into social networks on the Web. Web 2.0 social networks are defined as sets of social entities (e.g., people, organizations, etc.) connected by a set of social relationships (e.g., friendship, co-working, information exchange, etc.) [4].

The rich potential of social networks comes from two aspects. First, social networks can be utilized to push information to a target group, e.g., company advertisements, blogging, tweeting at Twitter, etc. Second, pulling information from social networks into a customer relationship management (CRM) system is also possible.

Hanno Zwikstra (e-mail: 265948jz@student.eur.nl) · Frederik Hogenboom (e-mail: fhogenboom@ese.eur.nl) · Damir Vandic (e-mail: vandic@ese.eur.nl) · Flavius Frasincar (e-mail: frasincar@ese.eur.nl)
Erasmus University Rotterdam, P.O. Box 1738, NL-3000 DR Rotterdam, The Netherlands

An example is monitoring social network sites for content (conversations, blogs, tweets on Twitter, etc.) in which a company or brand is mentioned, providing potentially valuable information and means to interact with customers. Microsoft already provides an add-on for its CRM system that imports tweets from Twitter in which a company is mentioned, into the CRM application. Oracle, the world's leader in CRM, advocates Social CRM actively by offering Oracle Social CRM Applications.

Due to the increasing popularity of social networks as well as their commercial potential, in this paper we present a framework that aims to reuse LinkedIn data within a CRM application. We evaluate the framework by means of a proof of concept implementation. In our development, we aim for simplicity, yet we also allow for future extendability. The remainder of this paper is organized as follows. Related work is described in Sect. 2. We then discuss our framework and its implementation in Sects. 3 and 4. The implementation is evaluated in Sect. 5. Finally, we conclude and provide directions for future work in Sect. 6.

2 Related Work

With the rise of Web 2.0, many new features emerged that enriched the Web environment, e.g., blogs, wikis, (social) tagging in folksonomies, and mashups that combine data from one or more other sources to create a new application, usually using an Application Programming Interface (API). One of the most noticeable features of Web 2.0 are social networks, like LinkedIn and Twitter. Social networks – also known under the common denominator as Social Networking Sites (SNS) – are a specific type of Social Media. Their content is generated and maintained by its visitors, without central coordination. Typically, in Social Media Web sites people form relationships with other users and interact with them [2]. Figure 1 depicts the number of unique visitors for LinkedIn and Twitter in the Netherlands, illustrating the immense growth in terms of visitors in the past 18 months. Interestingly, Twitter is still less popular than LinkedIn in the Netherlands, while in the rest of the world it is the other way around.

In the early 1990's the term CRM began to emerge [1]. Nowadays, many CRM providers exist, which offer Software as a Service solutions, e.g., Salesforce, Microsoft Dynamics CRM online, Oracle CRM On Demand, and SageCRM. The dif-

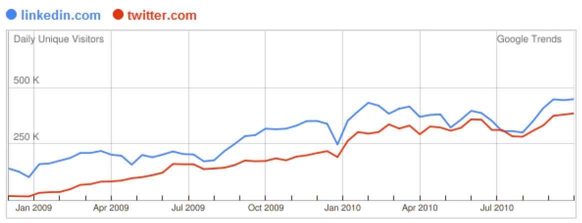


Fig. 1 Trend lines of unique visitors of LinkedIn and Twitter.

ference between CRM and traditional marketing is in the focus on customer relationships. Traditional marketing's main focus is on acquiring new customers, while CRM focuses on developing long term relationships with existing customers. According to Payne and Frow [7], three views CRM exist, ranging from a narrow and tactical view where CRM is a specific technical solution or tool, to a broad and strategic, or even philosophical view where CRM is a holistic approach for managing customer relationships.

Driven by changes in customers, starting in 2007 and more rapidly in 2008, CRM began to transform into what is now known as CRM 2.0 or Social CRM (sCRM) [3]. Both terms hint to Web 2.0 which is also called the Social Web. Today's customers are no longer passive customers, but have become social customers, who are active on the Internet, writing blogs, using Twitter, having discussions and informing themselves on social networking sites etc. and thus are well-informed about companies and products and services that they offer. CRM 2.0 is about joining the conversation [5] and extending CRM out of the business offices into the Social Media on the Internet, implying a change in strategy from a focus on customer transactions to a focus on both customer interactions.

3 Framework

This section discusses a general approach to social network data pulling called *SocialCRMConnector*. The *SocialCRMConnector*, which is a mashup between an arbitrary social network and a CRM system, can be used for pulling data from social networks using available APIs to build sCRM applications that use this data. Data can be pulled from a social network by sending a request to the API, which returns a response with the requested data. Our framework is targeted at retrieving profile data of social network members, i.e., personal information about social network members that is available by using the API of the social network. Although such profile data is also available on the social network sites themselves, there are three reasons why a company might want to retrieve the data by using an API. The first is simplicity. An application can contain specific business rules that are executed without input that is required from the user. This makes it easier to retrieve the desired data. The second is control. Data that is retrieved from an API can be processed in other applications or (temporarily) stored in internal information systems for further processing. Third, APIs enable applications to keep information up-to-date by allowing real-time access to data.

3.1 Entities and Relationships

At its core, a social network consists of *Users*, i.e., the people who have an account at the social network, and inter-user *Connections* define the relationships between users. Users have several attributes, i.e., a unique ID, nickname and real name, birthday, and likely a number of other attributes. With these entities it is already possible to create a network, though the possibilities of such a network are of course very limited at this point. Two *Users* with a *Connection* between them form a very small network. This simple model can be extended to better reflect real world online social networks. We have examined several online social networks, with a focus on Facebook and LinkedIn. To visualize the model, we have created an entity-relationship diagram for the model as an example, which is shown in Fig. 2. We focus only on the entities that may be useful sCRM applications; in practice social networks have a more complex structure, with additional entities such as *Event* or *School* and support for multiple media types such as videos, photos, and music.

Compared to the simple model that consists only of *Users* and *Connections*, we have introduced some new entities that are common for social networks. The figure illustrates that the *User* is the central entity in a social network. The first entity that we added is *Message*. Being able to send messages to other members of a social network is a very important feature of social networks. Without the ability to send messages, it would be impossible to interact with other users. Messages can be sent from a user to another user, but also to other entities, for example a user can comment on a photo that is posted by someone else, or post a comment to a discussion in a *Group*. For companies who want to monitor social networks for comments on their brand, the *Message* entity is an important one.

Instead of an entity that may have been called "*MediaItem*" to reflect all types of media items, we have added an entity called *Photo*, because photos are supported by all sites, even if it is only to add a picture to the user profile. We assume that the only useful media item to pull from a social network in relation to sCRM would be a profile picture. Comments that may have been added to profile pictures by other users are not really relevant in a business application, nevertheless for completeness reasons we decided to keep the link between *Message* and *Photo*.

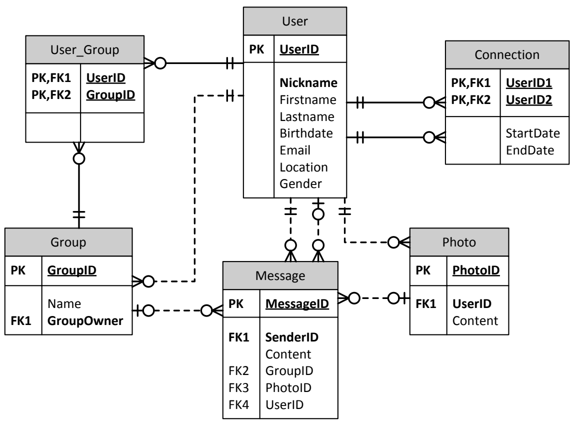


Fig. 2 Entity-relationship diagram of the social network model.

Another feature of many social network sites is the concept of *Groups*. Groups are a convenient way for users to connect with other users who share a similar interest. Groups can have many different forms, for example companies, schools, brands, persons, and virtually anything else. Groups are basically a collection of users, and are in many ways similar to individual users, but the concept and purpose is too distinct to treat them as one entity. The connections in groups are not direct connections (users do not become "friends" when joining the same group), but it makes it easier to meet new people. There is a many-to-many relationship between *Users* and *Groups*. There is also a direct relationship between *User* and *Group*, because a group is created by a user who then becomes the group owner. This relationship might also be displayed as a many-to-many relationship if the group can be owned by multiple users. Groups make it much easier for companies to find the consumers that they are interested in and the companies can participate in relevant groups to join the conversation with the consumers.

CRM databases are relational databases and usually consist of many tables, such as companies, persons, opportunities, products, orders, quotes, support calls, appointments, etc. Like the social network structure, our focus is on those entities that might be useful in the *SocialCRMConnector* framework. The CRM structure is shown in Fig. 3. Since we target social applications, the *Person* entity is an important entity in our CRM model, just like *User* is in a social network. However, *Companies* are the central entity in a CRM system. A company has a specific type that indicates the relationship between that company and the super company in which the current company is listed. Specific company types can be customers, competitors, suppliers, partners, government, etc. A third entity called *Lead* is also added. Leads, or prospects, are defined as potential customers that do not yet have a relation with the company. Finally, an *Opportunity* entity is used to record a potential sale. An opportunity is given an estimated value and expected probability that the sale will be made. Linked to an opportunity must be one or more *Products* that specify what is going to be sold.

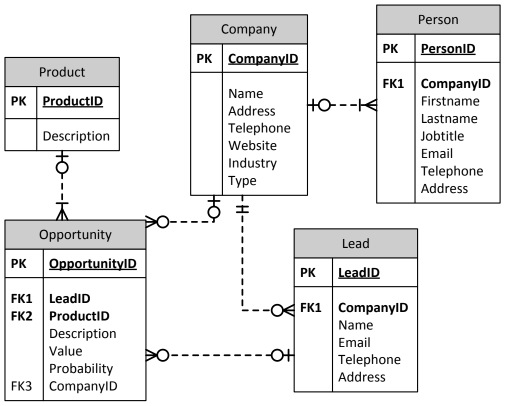


Fig. 3 CRM entity-relationship diagram (simplified).

3.2 Framework Architecture

After defining a model that covers the part of social networks that is relevant for our research, as well as a model that covers the structure of a CRM system, we can define the framework architecture. The framework consists of the following steps, which are also depicted in Fig. 4:

1. Generate query for retrieving profile information from a social network;
2. Retrieve data by sending the query to the API;
3. Process the data so it can be used in an application;
4. Present information from the application to the user;
5. Store data in the CRM system (optional, depending on user's choice).

The content of a query needed for retrieving the necessary data from a social network depends on a number of input parameters, for example the current user that uses the application, entity attributes that must be retrieved (dictated by business rules), and optional other parameters to refine the query such as a specific company or person to search for. Because the APIs flatten the data before returning it, queries to retrieve data are relatively simple. Flattening data is the opposite of normalizing data in the database. For example, a normalized *User* in a database may store a user's country with a foreign key to a record in a *Country*. The flattened data will return the data for a specific user with a country attribute that contains the country name. The pseudo SQL query for retrieving user data is:

```
SELECT attribute1, attribute2, ..., attributeN
FROM User, ...
WHERE condition1, condition2, ..., conditionN;
```

Depending on the type of application, information can be inserted into the CRM system. Some applications might only display additional information without the

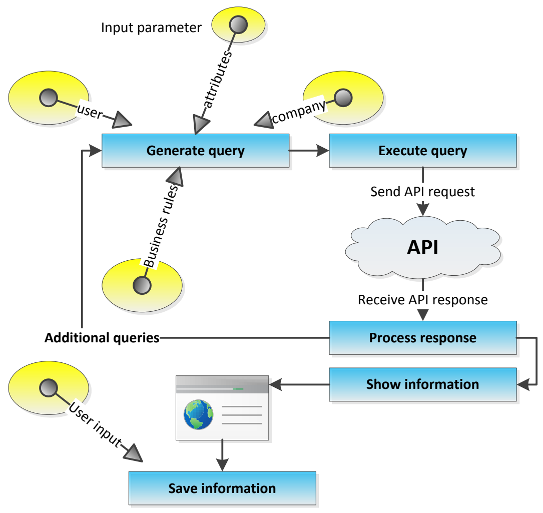


Fig. 4 Steps within the *SocialCRMConnector* framework steps.

need to store anything in the CRM system. Caution is required when data is going to be saved, because while it is technically possible to store data in the CRM system once it has been retrieved from a social network, not just any data can be stored without considering issues concerning privacy or obsoleteness. In contrast to contents of social networks, the IDs of records never change, and hence only insert queries (i.e., no update queries) are required. An insert query in pseudo SQL is:

```
UPDATE Person
SET Person.socialnetworkID = 'ID'
WHERE condition1, condition2, ..., conditionN;
```

4 Implementation

Now that we have defined the general framework, this section introduces the implementation of the *SocialCRMConnector*. *LinkedInFinder*, which integrates Microsoft Dynamics CRM with the LinkedIn social network, supports a use case where CRM users aim to find people that have a job at specific companies that are not yet listed within the CRM and with whom the company that uses the CRM would like to get into contact. These employees, which are supposedly the connection to a company of interest, are connected through first-degree connections with the CRM users. Connections of a second-degree or higher do not really make sense as it makes communication difficult. For demo and evaluation purposes, we have developed a stand-alone version of the *LinkedInFinder* application, which is available at `http://linkedin.hantheman.tk`. The main flow of the application is as follows.

After presenting the user a login page hosted by LinkedIn in order to authorize *LinkedInFinder*, first, a list of LinkedIn members that work at a certain company is retrieved with a call to the LinkedIn Search API. The list is then filtered so that only second degree connections are displayed to the user. Second degree connections have a distance of 2 to the user, which is measured as the number of steps from the user to the LinkedIn member. This means the distance to the own profile is 0, the user's connections have a distance of 1 and their connections have a distance of 2. Second, after selecting a name from the list by the user, more detailed information is displayed, including how the user is connected to this person. This connection is presented as a list of one or more of the user's first-degree connections. This list of connections actually is a list of mutual connections between the user and the person that is viewed.

4.1 Application Back-End

The *LinkedInFinder* application is integrated with Microsoft Dynamics CRM, which is an ASP.NET Web application that uses .NET Web services to communicate with the CRM database. For easy integration, our application is therefore also an ASP.NET Web application. It is built in Visual Studio 2010, using the (ASP).NET 3.5 Framework and C# as programming language. For connecting and interacting with the LinkedIn API, we employ the LinkedIn Developer Toolkit, which is an open source library for using the LinkedIn API in .NET applications. This library provides .NET support for the LinkedIn API, by providing .NET wrapper methods for most LinkedIn API methods and an implementation of the OAuth (Open Authentication) protocol that is used by many social network sites to authenticate requests to their APIs. It should be noted that the LinkedIn API has a number of technical restrictions that limit the possibilities of building applications that use the API. More complex applications that require more API calls are expected to suffer more from these limitations than simple applications that only use a few simple API calls. First, LinkedIn restricts the use of its API by limiting the number of calls to the API methods, i.e., throttling. Second, the number of results returned for a people-search query is limited by the account-level of the LinkedIn member.

The entity-relationship diagram underlying our implementation as depicted in Fig. 5 is a modified version of the entity-relationship diagram discussed in Sect. 3 (Fig. 2), reflecting the LinkedIn structure. In LinkedIn, media items are not as important as they are in other social networks like Facebook. Only photo items are available for storing a profile picture of the members. A user profile in LinkedIn has a large number of attributes and links to other entities. We have added the attributes and entities that are relevant for our application. For privacy reasons, LinkedIn does not return exact addresses, but areas, and hence we have added a *location* attribute to the model. The email address of LinkedIn members cannot be retrieved by any API calls, so we have removed it from the model. Phone numbers are optional and stored in a separate entity *PhoneNumber* that stores the actual phone number and type of phone number (e.g., "mobile"). Subsequently, we include a *URL* attribute, that stores the public profile URL. Furthermore, the *industry* attribute indicates in

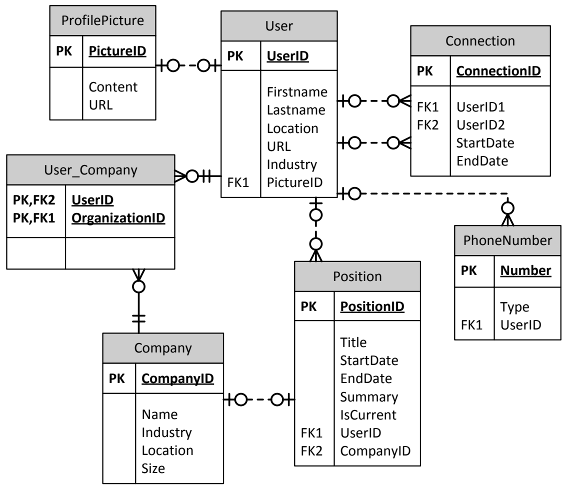


Fig. 5 Entity-relationship diagram of LinkedIn.

which industry the person is active; this may give some additional background information about the person's job. Then, another important entity in our application is *Company*. The relationship between *Users* and *Companies* is a many-to-many relationship, i.e., users can work at multiple companies. In our application we search for LinkedIn members that work at a specific company. The last entity that we need in our model is *Position*. A position describes the actual job that the LinkedIn member has at a company. The *iscurrent* attribute indicates whether the position is a current - or past position. In the LinkedInFinder application we only use current positions.

In our implementation, the application is started from a company record in the Microsoft Dynamics CRM system, which is opened in a Web form. The company name of the record is sent to the application as input parameter. The most important items of the implementation are the API requests that are sent to LinkedIn. The application uses a Search API request and a Profile API request. The API request has three variable parameters, which are the `company-name`, `start` and `count`. The `company-name` specifies the company name to search for. The `start` and `count` parameters specify the indexes for a subset of data. For example `start=0`, `count=10` retrieves the first 10 results. The other parameters will remain the same for each request. A search for the first ten people that currently work at "Erasmus" would yield the following request:

```
http://api.linkedin.com/v1/people-search:(people:(id,distance,
first-name,last-name))?company-name=Erasmus&current-company=1&
sort=relevance&start=0&count=10
```

This request is divided into several parts. The first part, i.e., `http://api.linkedin.com/v1/`, is the base URL of the LinkedIn API. The second part is the API function that is called, i.e., `people-search:`. The third part contains field selectors and utilizes a JSON-like syntax, i.e., `(people:(id,distance,first-name,last-name))`. The rest of the query, i.e., `?company-name=Erasmus&current-company=1&sort=relevance&start=0&count=10`, comprises an optional query string with the three variable parameters. In this case we specify that the company name is Erasmus, it must be the current company, search results are ordered by relevance (other options are number of connections and distance), and the number of results that is returned per request is limited to ten.

The aforementioned request returns ten profiles (assuming at least ten people work at Erasmus) with the `id`, `distance`, `first-name`, and `last-name` fields. The distance field tells the distance between the user that executes the query and the person that is returned by the API. People with a distance of 1 are first-degree connections, and people with a distance of 2 are second-degree connections. We use the distance field to filter the search results to contain only second-degree connections (we cannot specify this in the search query).

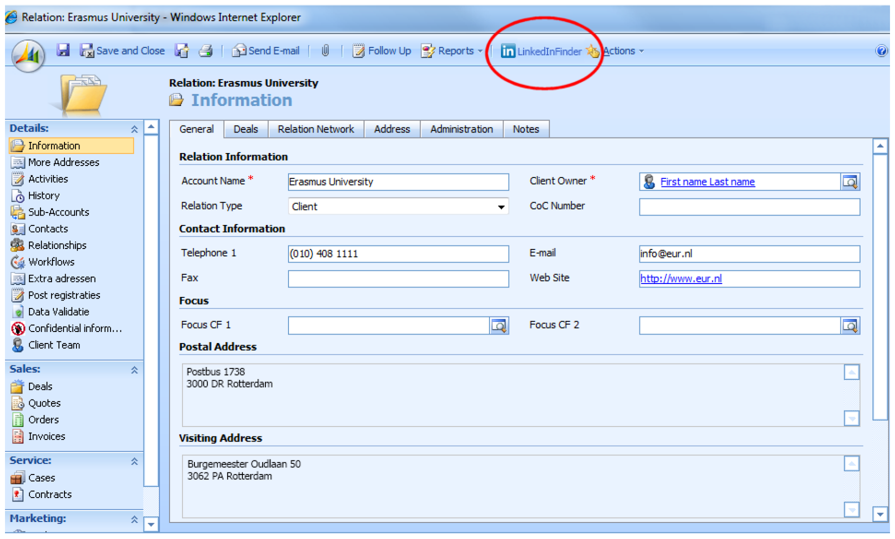
The Profile API request has only one variable parameter, which is the profile id. The request is as follows:

```
http://api.linkedin.com/v1/people/id=nnl7Qkt7Kb:(last-name,
first-name,num-connections,num-connections-capped,phone-num
bers,three-current-positions,picture-url,location:(name),re
lation-to-viewer:(distance,num-related-connections,related-
connections),positions:(title,company:(name)))
```

The response for this request contains the relevant profile details and is parsed and displayed to the user who can then decide to use this information or to discard it and view another profile.

4.2 Application Front-End

The front-end of the application has three main windows. First, when opening an account record within the CRM tool, there is a button that will open the *LinkedIn-Finder* application shown in Fig. 6. Upon first use, the user will be redirected to a LinkedIn page to authorize the application using OAuth. Subsequently, the application redirects to a secure page on the LinkedIn Web site, and after entering the LinkedIn account credentials, the user is redirected back to the application. Second, there is a Web page (`Search.aspx`) displaying the results of the search query as a list of hyperlinks. The names of the people that are returned by the search query are displayed as the hyperlink text. Third, another Web page (`Details.aspx`) is opened when a hyperlink is clicked. This page, as depicted in Fig. 7, displays the detailed information for that person, which requires a Profile API call.



Relation Information

Account Name *	Erasmus University	Client Owner *	First name Last name
Relation Type	Client	CoC Number	

Contact Information

Telephone 1	(010) 408 1111	E-mail	info@eur.nl
Fax		Web Site	http://www.eur.nl

Focus

Focus CF 1		Focus CF 2	

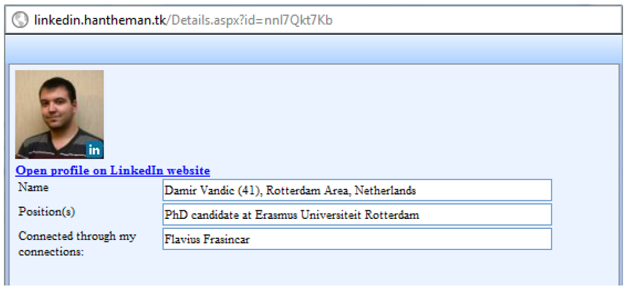
Postal Address

Postbus 1738
3000 DR Rotterdam

Visiting Address

Burgemeester Oudlaan 50
3062 PA Rotterdam

Fig. 6 Account form in Microsoft Dynamics CRM.


linkedin.hantheman.tk/Details.aspx?id=nnl7Qkt7Kb

Open profile on LinkedIn website

Name	Damir Vandic (41), Rotterdam Area, Netherlands
Position(s)	PhD candidate at Erasmus Universiteit Rotterdam
Connected through my connections:	Flavius Frasincar

Fig. 7 Person details in the *LinkedInFinder* application (`Details.aspx`).

5 Evaluation

In terms of evaluation, we first validate our implementation by comparing search results of *LinkedInFinder* with the LinkedIn Web site search engine (using advanced search). For several companies, the search results are identical in all cases. Interestingly, when using the standard search function on the LinkedIn Web site, search results are different in terms of display order, pointing to a different ranking algorithm.

In order to evaluate the *LinkedInFinder* tool qualitatively, we conduct a survey in which participants are asked for their opinion about the following statements in a questionnaire:

Statement 1: I find the application useful.
Statement 2: The application shows enough information to be useful.
Statement 3: The application offers enough functionality to be useful.
Statement 4: I would use the application for my job.

A five-point Likert scale is used for the available answers (ranging from 1 [totally disagree] to 5 [totally agree]), which provides an ordinal scale. The results of the survey (amongst 17 participants that are CRM users and members of the authors' LinkedIn network) are depicted in Fig. 8. Furthermore, we define two hypotheses to evaluate the *LinkedInFinder* application, i.e.:

Hypothesis 1: The *LinkedInFinder* application is useful.
Hypothesis 2: The *LinkedInFinder* application design is adequate.

To assess hypothesis 1, we use statements 1 and 4. To assess hypothesis 2 we use statements 2 and 3. When employing the χ^2-test to determine whether the attitude towards the *LinkedInFinder* is neutral, we obtain significant outcomes to reject this null hypothesis (with p-values lower than 0.01) for all statements, which means we can accept both hypotheses.

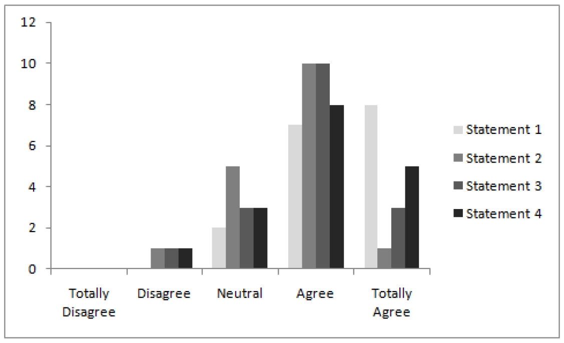


Fig. 8 *LinkedInFinder* evaluation results.

6 Conclusions

In this paper, we presented the *SocialCRMConnector* framework that aims to feed CRM applications with Web 2.0 social networks data. The framework has been implemented as a tool called *LinkedInFinder* that pulls data from LinkedIn into the Microsoft Dynamics CRM system. Our implementation validates the proposed framework by means of a use case to find second-degree connections within one's network that work at a specific company of interest. Results from our evaluation based on a user survey indicate that the application is useful and adequately designed for the intended use. As future work, we envision implementations of our framework using other social networks (possibly for other purposes as well) and CRM systems. Also, one could employ data pulled from social networks to other applications, such as personalization tools.

References

1. Buttle, F.: Customer Relationship Management: Concepts and Technologies, 2nd edn. Butterworth Heinemann (2009)
2. Gilbert, E., Karahalios, K.: Predicting Tie Strength With Social Media. In: 27th International Conference on Human Factors in Computing Systems (CHI 2009), pp. 210–220. ACM (2009)
3. Greenberg, P.: The impact of CRM 2.0 on Customer Insight. Journal of Business & Industrial Marketing **25**(6), 410–419 (10)
4. Laura Garton, C.H., Wellman, B.: Studying Online Social Networks. Journal of Computer-Mediated Communications **3**(1) (1997)
5. Leary, B.: The Tweet Is Mightier than the Sword. CRM Magazine **13**(1), 48 (2009)
6. O'Reilly, T.: What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. International Journal of Digital Economics **65**(1), 17–37 (2007)
7. Payne, A., Frow, P.: A Strategic Framework for Customer Relationship Management. Journal of Marketing **69**(4), 167–176 (2005)