A Microservice System for Unified Communication over Multiple Social Networks

Angelina Vujanović, Nemanja Igić, Ivan Luković, Dušan Gajić and Vladimir Ivančević
University of Novi Sad/Faculty of Technical Sciences, 21000 Novi Sad, Serbia
{avujanovic, nemanjaigic, ivan, dusan.gajic, dragoman}@uns.ac.rs

***Abstract*— In this paper, we present a prototype version of a new software solution that allows its users to simultaneously send the same message and set the same status on multiple social networks. We describe the microservice architecture of our solution, where communication with social networks through their Application Programming Interfaces (APIs) is handled by a set of dedicated services, one for each supported social network. Services in our microservice architecture may be used by one or more users, who send requests to the services in order to use their capabilities. In addition to the services for different social networks, there is also a service orchestrator whose task is to direct user requests to the appropriate service. At present, our software solution supports the following social network sites: VKontakte, Twitter, and LinkedIn. We also give an overview of a new textual domain specific language (DSL). This DSL is used in our software solution to specify commands for sending messages and posting statuses on social networks.**

I. INTRODUCTION

In recent years, we have witnessed the growing popularity of social networks. A social network is a structure consisting of nodes (individuals or organizations), as well as links between these nodes [1]. In this paper, we use the term social network also to denote a social web site that allows its users to communicate with each other, express their opinion, post media content. The primary purpose of social networks is communication between people, but social networks may be used also in marketing to promote web sites and various kinds of service (e.g., selling) [2], [3]. Because of their increasing influence on daily lives and their large number of users, social networks are becoming more interesting not just to business users, but to researchers as well.

These days most individuals and companies, small and large alike, use social networks for presenting themselves or their brand, because social networks provide them with a wide number of potential customers, much higher than they would have without social networks [4], [5]. Individuals may use social networks to promote services that they offer (e.g., beauty care services) and express their opinion in order to get more customers or followers, all that being repeated for each social network where these individuals have accounts. Companies use social networks to promote themselves and their services and get new employees and customers. Some of the companies have dedicated employees or even whole departments that are responsible for the company's image on social networks. These employees and departments (e.g., human resource managers/departments) are tasked with sending job offers on behalf of the company, posting advertisements and promoting company on all social networks where the company has an account. Posting the same status or sending the same message on each social network separately is a redundant, often tedious work.

Our goal is to provide users of social networks with a single interface that would offer common social network features for all social networks where the users have accounts, such as sending messages, posting statuses, and changing personal information. To achieve this goal, we developed a system that supports working with multiple social networks. We designed the architecture of our system with the idea of having small services instead of a single monolithic system and, consequently, opted for microservice architecture. This kind of architecture is more conducive to application development and upgrading, as individual services may be independently developed, configured, and tested. Set of dedicated services are in charge of communication with social networks, one for each supported social network. Besides services for social networks, we also created a service that redirects client requests to the appropriate service for social network. We also created a textual domain specific language (DSL) that supports commands for sending messages and posting statuses.

In this paper, we present an application, a prototype version of a new software solution that provides the aforementioned capabilities for the following three social network sites: VKontakte[1], Twitter[2], and LinkedIn[3]. We also give a brief overview of the microservice architecture of our solution, where communication with social networks through their Application Programming Interfaces (APIs) is handled by a set of dedicated services, one for each supported social network. In addition to services for different social networks, there is also a service orchestrator whose task is to direct user requests to the appropriate service for a particular social network. The created textual DSL is also presented in this paper.

The rest of the paper has the following structure. Related work is presented in Section II. In Section III, we describe the selected social networks, their APIs and data taken from social networks. We present the architecture of our solution in Section IV. Section V contains a description of the textual DSL. An overview of the

[1] More about VKontakte at: https://vk.com/
[2] More about Twitter at: https://twitter.com
[3] More about LinkedIn at: https://www.linkedin.com/

implemented application is presented in Section VI. At the end of this paper, we report the most important conclusions.

II. Related Work

This section gives a brief overview of similar software solutions and describes how they differ from our solution. The principal research problem that we identified is the need for an application that would allow people to simultaneously send the same message and post the same status on multiple social network. Because this topic is not only very popular, but also appears to be profitable, besides free and open source solutions, there are many commercial solutions available online.

HootSuite [6] is probably the most popular social media management application. It allows its users to post messages to several popular social networks. HootSuite has also a built-in custom analytics system and option to schedule posts whenever users want. It has recently moved from monolithic to microservice architecture. Its microservices are classified into three categories: data services, functional services, and facade services [7]. Most of its services are written in Scala. In contrast to our solution, Hootsuite does not support sending messages to friends over multiple social networks. Buffer [8] and TweetDeck [9] are two other examples of a similar solution. Buffer is a social posting scheduler for Facebook, Pinterest, LinkedIn, Twitter, etc. Like HootSuite, Buffer has also recently switched to the microservice architecture. Its microservice architecture is deployed by using an open-source system, Kubernetes[4]. The main difference between Buffer and our solution is that Buffer does not allow its users to send messages to friends from different social networks. TweetDeck is an application for managing multiple Twitter accounts. It allows users to follow specific hashtags, reply to other users, etc. TweetDeck uses Flight[5], an event-driven web framework.

Some applications use social network APIs, such as Messenger for LinkedIn [10], which provides the ability to send messages on LinkedIn, and Friend Check-Ultimate social network manager [11], which provides users with an insight into the activities of their friends and followers on Twitter. Tweetogram [12] is an open-source application that uses Twitter API and allows users to view their account, post and explore statuses on their timeline, etc.[6]

To the best of our knowledge, none of the aforementioned solutions features a microservice architecture where communication with social networks is handled by a set of dedicated services, one for each social network. One more difference between our solution and similar publicly available solutions is our reliance on a DSL for sending uniform messages and posting statuses.

[4] More about Kubernetes at: https://kubernetes.io/
[5] More about Flight at: https://flightjs.github.io/
[6] Table of comparison aforementioned solutions is available at: https://drive.google.com/open?id=0B_ANXd6dka68Z0lWbnhxTEt2dk0

III. Description of Data Sources

This section presents a short description of the three selected social networks: VKontakte, Twitter, and LinkedIn. There is also an overview of external APIs and data from these social networks.

A. VKontakte

VKontakte (Russian: "ВКонтакте", translated as "In touch") is the largest social network in Russia, similar to Facebook. Like almost every other social network, VKontakte gives its users the ability to send messages, post statuses, create groups, events, and public sites, share images, audio, and video, etc.

VKontakte provides access to its data through a public API. VKontakte API [13] represents an interface through which data may be received from the VKontakte database without knowing structural details of the database, such as how the database is constructed, from which table and field types it consists of, etc. By using this API, it is possible to create applications that communicate with VKontakte, i.e., read and write data from VKontakte. In order to use all the capabilities of VKontakte API, an application has to be registered at the VKontakte site for developers. Also, almost all the methods require authentication, i.e., the server must verify who want to access to the information. There are multiple ways of authentication that VKontakte supports, like OAuth (Open Authentication), SDK (Software Development Kit) and Open API authentication. However, we did not directly use VKontakte API, but relied on VK.NET [14], which is a .NET library for working with VKontakte API. We utilized the OAuth 2.0 public protocol [15] for authentication, as this protocol is used in the VK.NET library. By using OAuth, access token is obtained. Access token is an array of numbers and letters that is sent to the server along with the request. There are three types of tokens: user, community, and service tokens. A user token may be regarded as a user signature, i.e., in this way the server receives all the necessary information, such as who sent the request and which permissions the user has. A community token is used for working with public pages, groups, and events. In order to work with administrative parts of the application, a service token is used. Because our solution works with user profiles, it utilizes user tokens. In order to call an API method, it is necessary to make a POST or GET request to the specified URL (Unified Resource Location) using HTTPS (Hyper Text Transfer Protocol Secure). A response to the request may be in the JSON (JavaScript Object Notation) or XML (EXtensible Markup Language) format. In case the XML format is preferred, it is necessary to explicitly request it.

B. Twitter

Twitter is a popular microblogging service that allows users to post and read short messages of up to 140 characters, known as "tweets" [16]. What Twitter differs from most of social networks is that the relationship of following and being followed does not require reciprocation. A user can follow any other user, and the

user being followed does not need to follow back. Because of this relationship, some authors do not consider Twitter as social network, than a social media [17]. In this paper, we consider Twitter as a social network.

Like VKontakte, Twitter also provides an API that allows users to programmatically read and write data. In order to use all the capabilities of Twitter API [18], an application has to be registered at the Twitter site for developers. As is the case with VKontakte API methods, almost all Twitter API methods require authentication. Twitter uses the OAuth 2.0 public protocol for authentication. By using OAuth, an access token is obtained. With respect to the way data are accessed, there are two types of APIs on Twitter: REST (Representational State Transfer) API and Streaming API. The result of both REST and Streaming API methods is an object in JSON format. As with VKontakte, we did not directly use the official API, i.e., Twitter API, but relied on a .NET library named TweetSharp [19] instead.

C. LinkedIn

LinkedIn is a business social network for the people looking for work and for the employers looking for new workers. LinkedIn allows its users to set CVs, share content, and join groups and sites.

As is the case with VKontakte and Twitter, LinkedIn provides an API that allows users to read and write data. In order to use these capabilities of LinkedIn API [20], the application has to be registered at the LinkedIn site for developers. Also, almost all LinkedIn API methods require authentication. LinkedIn uses the OAuth 2.0 public protocol for authentication. There are two types of APIs: REST and JavaScript API. Methods of REST API return a response in the JSON format. In addition to the JSON format, the response may also be in the XML format, but it is necessary to explicitly request it.

LinkedIn has significantly limited the possibilities of its API since 2015 [21]. An extremely small number of functionalities have remained available, while additional functionalities may not be used unless special permissions are obtained first. In order to work with LinkedIn API, we used a .NET library named LinkedInNet [22].

IV. System Architecture

In this section, we give a brief overview of the architecture of our solution. The main idea behind the architecture of our system was to divide the system into smaller parts-services. Communication with the social networks should be handled by set of dedicated services, one for each social network. Each service should have its own database and should be independently developed, configured and tested. Services may be used by one or more client applications, which in turn could be web or desktop applications. This method of building a system is known as the microservice[7] architecture [23]. Services communicate over the network by using standard transport protocols, such as REST, AMQP (Advanced Message Queuing), and JMS (Java Message Service).

The approach of building applications as sets of smaller parts is not innovative. Similar philosophy may be found in Unix: "Write programs that do one thing and do it well. Write programs to work together."[8] The microservice architecture approach is also very similar to the one named SOA (Service-Oriented Architecture) [24].

SOA is an architectural style that uses services in order to model information contained in the system. SOA also promotes the use of poorly integrated services to ensure business flexibility. Both approaches are service-oriented, but SOA is a much broader concept and for this reason microservices may be seen as a specialization of SOA.

We opted for the SOA because of its reusability, interoperability, flexibility, and cost effectiveness [24]. One of the contemporary challenges is to provide support for application upgrading in order to best respond to the demands and needs of application users-individuals or companies. By upgrading we consider adding support for other social networks besides those already supported or providing some new functionality within the application. Upgrading an existing service or adding a new service should not affect the whole application, but only the service where the change is made. This is in sharp contrast to the monolithic architecture, in which any change to the system requires rebuilding and redeploying the whole application and not only the affected part. In the case of communication over multiple social networks, if there is a need to make changes at one social network API (e.g., to change library which works with it), we do not need to make changes at the whole application, but only at the service which is using that specific API. However, this approach also has some disadvantages, such as a large number of components and an increase in network traffic [26].

A. Microservice Architecture

For the implementation of our microservice architecture we used Windows Communication Framework (WCF) [27], [28] within Visual Studio Enterprise Edition 2015. The basic components of a WCF application are the following three: WCF service, WCF service host, and WCF service client.

Our system architecture (shown in Fig. 1) includes five services:

- three social network services (VKontakte, Twitter, and LinkedIn),
- one logging service, and
- one service orchestrator.

Communication with the services is done through the Hyper Text Transfer Protocol (HTTP), while Simple Object Access Protocol (SOAP) is used for exchanging messages between the services. The client application is implemented as an ASP.NET web application. All the services, except the service orchestrator, have their own database. Communication with the database is done

[7] The term "microservice" was discussed at a workshop in 2011 as an architectural style and adopted in 2012 [25]

[8] Doug McIlroy about Unix philosophy [33]

through HTTP/REST requests. As database for each service we used a NoSQL database RavenDB [29]. In order to store data into the database we created .NET class Person, which encapsulates user information. Each service has a database where data about friends from some social network are stored. One of our future goals is to use these stored data in order to analyze user profiles.

A WCF service client communicates with WCF services via channels. WCF services are deployed, discovered and consumed as a collection of endpoints. Each service must have at least one endpoint [28], which represents an address where a message should be sent or received. The basic components of an endpoint are address, binding, and contract (mnemonic ABC). Address represents the exact location where message should be received and it is specified as a URI (Uniform Resource Identifier), e.g., *http://localhost:8080/twitter*. Binding defines the way the endpoint communicates. Contract represents a set of operations by which functionalities the endpoint exposes to the clients are specified.

The process of creating a WCF service involves the following steps [27]:

- defining data contract,
- defining service contract,
- implementation of service,
- configuration of the endpoints and service behavior, and
- service hosting.

We implemented microservices by using standard .NET classes and interfaces. Classes represent messages which services send, while interfaces define sets of operations that endpoints expose to the clients [30]. Both of them are annotated with the .NET annotations that represent special WCF attributes.

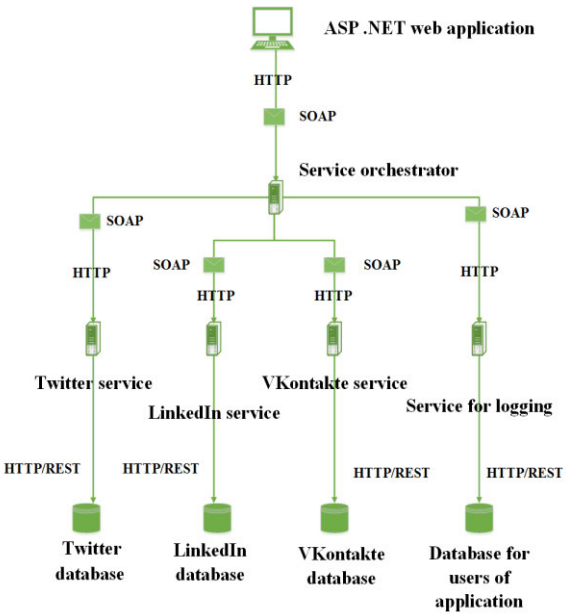

Figure 1. The system architecture

B. *Service Orchestrator*

The main idea behind the service orchestrator is to have only one service where all the requests from clients come to and are then directed to the specific service for a social network.

The service orchestrator works on the principle of the Routing Service [31], a service that mediates communication between the client and the service (shown in Fig. 2). It redirects a user request to the specific service, based on the parameters within the SOAP message.

There are several built-in WCF mechanisms for filtering SOAP messages, such as MatchAllMessageFilters, ActionMessageFilters, and XPathMessageFilter. In our solution, we used the XPathMessageFilter mechanism, which performs the SOAP message filtering based on the access points, names or special XPATH expressions. We chose to work with XPathMessageFilter mechanism, because it proved to be one of the best option for solving the problem at that moment.

The service orchestrator, which is in Fig. 2 marked as WCF router, receives all SOAP messages from clients and filters them based on the value of ServiceType, which is sent as the parameter of the method in SOAP message (e.g., method for posting status). Clients only need to know the address of the service orchestrator, while the orchestrator further redirects requests. Filtering of SOAP messages is done by using an XPATH filter and a filter table, which are defined in the configuration file of the service orchestrator. Filter tables combine filters with service access points.

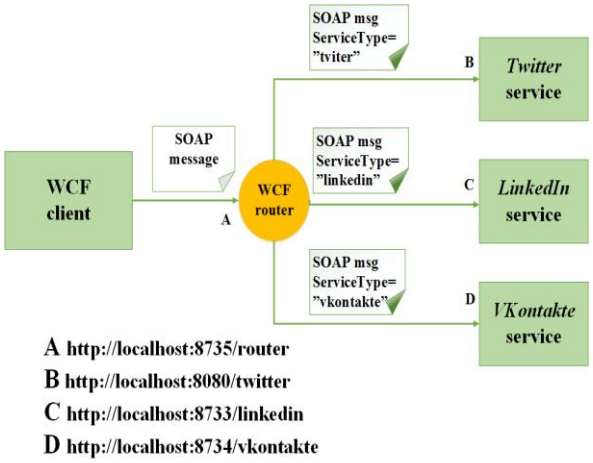

Figure 2. The Routing Service

V. DOMAIN SPECIFIC LANGUAGE

In this section, we present a new textual domain specific language (DSL), which is being used in our solution to specify commands of for sending uniform messages and posting statuses. In order to create a DSL, we first had to identify the concepts specific to the domain. Concepts specific to the domain of sending uniform messages and posting status need to exist on

each of the selected social networks or, to be more precise, should be available using the official API. In order to send uniform messages, we have recognized the following concepts: first name, last name, birthday and city. By using the new textual DSL, the user may specify the message "Hello {FirstName} {LastName}", where FirstName and LastName are the concepts that we recognized. The names of concepts correspond to the fields of the class Person, which encapsulates information about friends from social networks. It is necessary to put each concept between the curly brackets

Moreover, our textual DSL supports detailed text commands for sending messages and posting statuses. In order to post status user should enter a command like:

"**command(***typeOfCommand***; select** *socialNetworks***;** *text***)**"

where

- *typeOfCommand* represents the command type ("status" for status posting and "msg" for message sending),
- *socialNetworks* represents a comma-separated list of social network names, and
- *text* represents the text of the desired status.

The command for sending messages is very similar, but it includes the names of friends to whom the message should be sent. If the user wants to notify all friends, then instead of their names, the character "*" should be entered. For users who are not very familiar with textual commands, we have supported these commands through a graphical user interface. Through this graphical interface, users should only enter the text of their message in the aforementioned format (more information about the usage of our solution is presented in Section VI). After the user chooses the friends from the list of all friends and enters the desired message, the text of the message is generated in the background and aforementioned concepts (e.g., FirstName, LastName etc.) are replaced by the names of the recipients or by other attributes, such as city or birthday. Text generation is also done for the non-graphical command for message sending. To generate the text, we used a .NET library named SmartFormat.NET [32].

VI. An Overview of the Software Solution

This section contains a short overview of the implemented software solution. The application is designed to support display of friends from all social networks (VKontakte, Twitter, and LinkedIn), sending a message to selected friends, posting a status to selected social networks, getting information about friends, and changing personal information. In order to best respond to the demands of the modern business environment, the client application is implemented as a web application, using ASP .NET web technology.

To be able to use the main functionalities, it is necessary to log in to the system. Application credentials, like username and password, do not need to have any relation with user credentials on a particular social network.

Fig. 3 shows the graphical interface for sending messages. Text is entered in the format described in Section V. Alongside the text area, there is a list of concepts that may be used, such as FirstName and LastName.

Fig. 4 shows the graphical interface for posting status. The user enters the text of the status and selects the social networks on which the status should be published.

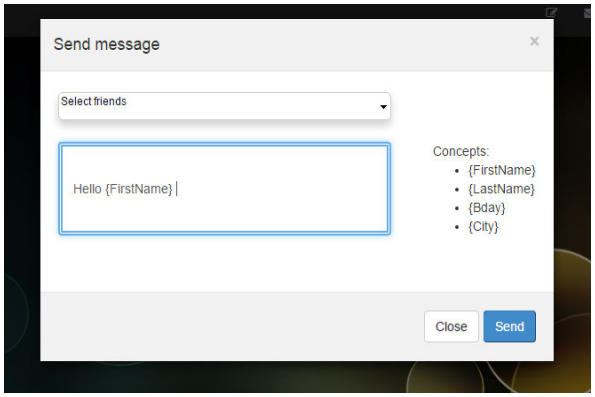

Figure 3. The graphical interface for sending messages

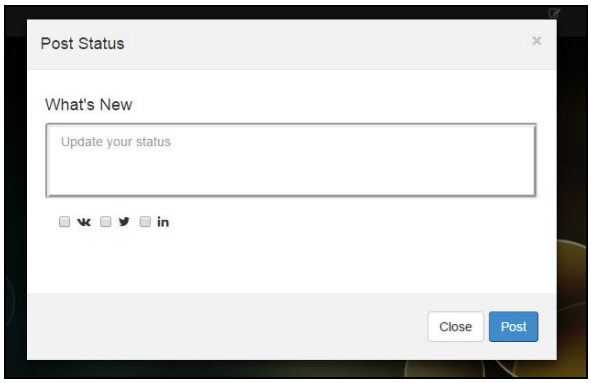

Figure 4. The graphical interface for posting status

VII. Conclusion

The application that is subject of this paper was developed for the needs of the companies and individuals interested in better use of the opportunities that social networks provide. The presented application is a prototype version of software solution that allows its users to select friends from all social networks where users have accounts and send the same message. In order to post a status, users have to enter the text of the status and select social networks on which they want to post it. The application implemented supports the following social network sites: VKontakte, Twitter, and LinkedIn. In this paper, we also give a short description of the used microservice architecture, where communication with social networks through their APIs is handled by a set of dedicated services, one for each supported social network. The system architecture, besides services that work with social networks, includes two more services:

- a logging service and
- a service orchestrator whose task is to direct user requests to the appropriate social network service.

The paper also includes an overview of the created textual DSL, a language that is being used for specifying commands for sending messages and posting statuses.

Changes in the official API or privacy policy of a social network may pose important challenges for our solution. One of the problems that was encountered during implementation was related to the LinkedIn API or, to be more precise, the restrictions that this API has recently introduced. Because of the LinkedIn privacy policy we could not send messages on that social network using its API. Besides LinkedIn, Facebook has also introduced restrictions.

Further development of our solution includes obtaining necessary permissions from LinkedIn in order to access information about connections of users, and then implementing message sending. One of the improvements could also be the support for analysis of data from social networks. For example, by analyzing user profiles on LinkedIn, it could be possible to send specific job offers according to the user skills and endorsements. We could also add the option to schedule posts, which is already available in HootSuite, Buffer, and some other similar solutions. One of the future research goals should be directed at extending the textual DSL, e.g., by adding filters, so users could send messages to their friends from a specific city.

ACKNOWLEDGMENT

The research presented in this paper was partially supported by Ministry of Education, Science and Technological Development of Republic of Serbia, Grant III-44010.

REFERENCES

[1] S. Wasserman, K. Faust, "Social network analysis," in *Cambridge, MA: Cambridge University Press*, 1994.

[2] W. Assad, J.M. Gómez, "Social Network in marketing (Social Media Marketing) Opportunities and Risks," in *International Journal of Managing Public Sector Information and Communication Technologies*, 2011.

[3] M. A. Stelzner, "2016 Social Media Marketing Industry Report," *Social Media Examiner*, 2016.

[4] M. Bîja, R. Balaş, "Social Media Marketing to Increase Brand Awareness," in *Journal of Economics and Business Research,* pg. 155-164, 2014.

[5] V. Pein, "Gute Berater, schlechte Berater. Die richtigen Dienstleister für Social Media finden," in *t3n*, vol. 19, pg.40-41, 2010.

[6] "Hootsuite," [Online]. Available: https://hootsuite.com/ [Accessed: 2.11.2016.]

[7] I. Nadareishvili, R. Mitra, M. McLarty, M. Amundsen, "Microservice Arhitecture: Aligning Principles, Practices, and Culture," *O'Reilly Media, Inc, 1005 Gravenstein Highway North, Sebastopo*l, 2016.

[8] "Buffer," [Online]. Available: https://buffer.com/ [Accessed: 2.11.2016.]

[9] "TweetDeck," [Online]. Available: https://tweetdeck.twitter.com/ [Accessed: 3.11.2016.]

[10] "Messenger for LinkedIn™," [Online]. Available: https://chrome.google.com/webstore/detail/messenger-for-linkedin/hcpfdnmjbcnnaonfbdpklgpmeldjpgnl, [Accessed: 5.11.2016.]

[11] "Friend Check-Ultimate social network manager," [Online]. Available: https://itunes.apple.com/az/app/friend-check-ultimate-social/id578099078?mt=8 [Accessed: 5.11.2016.]

[12] "Tweetogram," [Online]. Available: https://github.com/sahildua2305/tweetogram/ [Accessed: 5.11.2016.]

[13] "VKontakte OpenAPI," [Online]. Available: https://vk.com/dev/openapi, [Accessed: 10.11.2016.]

[14] "VK.NET," [Online]. Available: https://vknet.github.io/vk/, [Accessed: 11.11.2016.]

[15] "OAuth," [Online]. Available: https://oauth.net/, [Accessed: 10.11.2016.]

[16] "Tweet definition," [Online]. Available: http://whatis.techtarget.com/definition/Twitter, [Accessed: 13.11.2016].

[17] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a Social Network or a News Media?," in *19th International Conference on World wide web*, pg. 591-600, 2010.

[18] "Twitter for Developers," [Online]. Available: https://dev.twitter.com/overview/documentation, [Accessed: 13.11.2016.]

[19] "tweetmoasharp," [Online]. Available: https://github.com/Yortw/tweetmoasharp., [Accessed: 14.11.2016.]

[20] "LinkedIn for Developers," [Online]. Available: https://developer.linkedin.com/, [Accessed: 15.11.2016.]

[21] "ProgrammableWeb," [Online]. Available: https://www.programmableweb.com/news/linkedin-to-limit-public-developer-access-to-its-apis/2015/02/12, [Available: 16.11.2016.]

[22] *LinkedInNET* [Online]. Available: https://github.com/SparkleNetworks/LinkedInNET, [Accessed: 15.11.2016.]

[23] S. Newman, "Building Microservices", *O'Reilly Media, Inc, 1005 Gravenstein Highway North, Sebastopol*, 2015.

[24] M. Richards, "Microservice vs. Service-Orientied Architecture", *O'Reilly Media, Inc, 1005 Gravenstein Highway North, Sebastopol*, 2015.

[25] M. Fowler, "Microservices" [Online]. Available: https://martinfowler.com/articles/microservices.html#footnote-etymology, [Accessed: 1.11.2016.]

[26] "Why a microservices approach to building applications?," [Online]. Available at: https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview-microservices, [Accessed: 2.11.2016.]

[27] "Windows Communication Foundation," [Online]. Available: https://msdn.microsoft.com/en-us/library/dd456779(v=vs.110).aspx, [Accessed: 20.11.2016.]

[28] W. Zhang, G. Cheng, "*A Service-Oriented Distributed Framework-WCF*", in *International Conference on Web Information Systems and Mining*, pg. 302-305, 2009.

[29] "RavenDB," [Online]. Available: https://ravendb.net/, [Accessed: 24.11.2016.]

[30] "DataContract and ServiceContract WCF," [Online]. Available: https://www.leadtools.com/help/leadtools/v19/dh/to/leadtools.topics.services.datacontracts~dc.topics.datacontractsandservicecontracts.html, [Accessed: 21.11.2016.]

[31] "WCF Routing Service," [Online]. Available: http://blog.micic.ch/net/creating-a-wcf-4-0-routing-service, [Accessed: 25.11.2016.]

[32] "SmartFormat.NET," [Online]. Available: https://github.com/scottrippey/SmartFormat.NET/wiki, [Accessed: 27.11.2016.]

[33] E.S. Raymond, "Basic of the Unix Philosophy," [Online]. Available: http://www.catb.org/~esr/writings/taoup/html/ch01s06.html, [Accessed: 3.11.2016.]