Designing and Implementing a Distributed Social Network Service for Mobile Devices

Alexandru RADOVICI, Valentin CRISTEA
"Politehnica" University of Bucharest
aradovici@ilab.fils.pub.ro, valentin.cristea@cs.pub.ro

The paper presents a new paradigm for building social network services. The proposed platform is called eXtensible Social Network. As it uses the XMPP protocol for authentication and communication, it allows users from different service providers interact with each other, without having to change their accounts. Moreover, the platform provides means for users to authenticate and interact with each other while temporary disconnected from the Internet. Moreover, the platform is specially designed for mobile devices, running on their restrictive operating systems and taking advantage of the systems optimizations.
Keywords: *Social Network, Mobile Devices, Service, XMPP, iOS, Android, Distributed, Platform*

1 Introduction

Social networks have lately become the most important mean of communication over the Internet. They currently integrate almost every one of the most used services offered online: email, instant messaging, file sharing and many other functions. When thinking of social networks, one might think they are mainly used for free time activities. Lately, this is not true. Social networks are now being used as well in academia [1] and business communication. Actually, nowadays' social networks can be thought as the evolution and integration of the services offered by the Internet.

Even if they are largely used, social network have several problems and security issues that might prevent users from using them. When it comes to social networks for mobile devices, current days implementations are just user-interfaces for web services offered by several social network services providers.

This paper tries to identify several problems that occur in social network services thus pointing out the importance of building a distributed social network service. The paper also proposes a new paradigm in building a distributed social network service, service that is independent of the current providers and that tries to solve most of the problems pointed out.

The second chapter of the paper describes the problems encountered in the social network services and proves why these problems are important of users.

The third chapter proposes a new paradigm for a distributed social network service, services which are based upon a standard protocol called XMPP [2]. In this way, users of different service providers are able to interact with each other. Moreover, the system describes is independent of any service provider, thus users not being bound to a certain terms of services.

The forth chapter shows the way that the new social network service is implemented for the most used mobile devices systems. iOS [3], [4] and Android [5] will be discussed. Limitations of the mobile systems are also taken into account and the implementation is optimized so that they take advantage of each system that they run on.

The last part of the paper describes briefly the testing using a real life application.

2 Social networks problems

We have identified several problems and security issues in nowadays' social networks. These problems are as follows:

- Users are able to connect and communicate only with other users of the same social network; in order to interact with users of another social network service, one of the users has to sign up for an account on the other social network service. In this way, users have face having

to choose which social network to sign up to so that is has the most advantages [6]. Most of the time, users will have to build accounts on several networks and share data on all of them.

- The social network service provider, based on the license and terms of services imposed to the users that want to use the service [7], has the right to use all the data shared by the users for no matter what purpose for an unlimited time, even after the user has asked the deletion of the data. The use of the data is done without having to notify the user. The service provider might sell the data to advertising companies that might use them for advertising campaigns or for sending spam. There have been accusations, no all being proved, that social networks have done this [8, 9].
- By linking the shared data, the social network provider can find out supplementary information about it users and not only [10]. For instance, using the shared photos' person tagging information, Facebook automatically recognizes persons in the newly shared pictures. This is done without the approval of the persons in that picture. In this way, persons that don't even have a Facebook account might get tagged in photos. It is sufficient for some users to tag a certain person by just writing his name as a string. Facebook will then automatically recognize that person in most of the newly shared photos.
- Social networks don't offer very many services on their own. Most of them offer instant messaging, files, pictures and video sharing, messages and event planning. Third party applications built for the social networks offer most of the valuable services. The problem is that every social network service provides its' own API for writing applications, thus making porting applications from one network to another very difficult. Moreover, in this way, applications, which are actually the main part, become just plugins for the social network.
- Existing applications need to be rewritten in order to benefit of social network functions. For large applications, this is not feasible. An example is a NutriEduc system [11] that needs social network extensions but cannot be rewritten for each social network.
- When it comes to mobile devices, social network systems provide only user interfaces that use the network's services. Moreover, most of them, due to the desire to look alike on all mobile platforms, don't take into account the devices that they are running on, thus being very inefficient on resource management and running slow. If the user chooses to use a third party application to access the social network, he has to give put his credentials. This might be a problem, as some of social network credentials are good for payments as well. Google+ uses the user's Google account, the same account used for Google Checkout service.

To overcome most of these problems, this paper proposes a new and original social network platform, called *eXtensible Social Network* (XSN). Its main strong points are:

- XSN is based upon a standard communication protocol called XMPP. In this way users that have account at different providers can communicate with each other. Moreover, XMPP provides string authentication between servers, thus assuring the identity of users.
- The platform reverses the applications paradigm. Instead of thinking on applications as plug-ins for the social network, XSN becomes a plug-in. By providing a library that can be integrated into existing applications, XSN allows existing applications to be extended with social network services.
- As mobile devices become more and more used, the platform's implementation is optimized for usage on such devices. The implementation takes into account each mobile system and its recommendations for writing applications. iOS and Android are the platforms that

are discussed, as they have together almost 70% market share.

3 The XSN Platform

We have showed before several problems and security issues of the social networks. In order to solve these problems, we have proposed a new social network service based on a new paradigm. It is called *eXtensible Social Network* (XSN). It has been designed having in mind the following ideas that would allow the users to communicate easy, to use existing applications and protect the uses' data from being used by the service provider:

- The system must be distributed in the sense not to require users to have an account at a single service provider. Regardless of service provider, user must have access to all system functions; no matter what service providers their friends have. The service provider must ensure the authenticity of the users and to provide data routing.
- Information stored in the system must not be covered by a license to use the service, which gives the service provider unlimited rights to the use of data loaded.
- The social network should be an extension of existing applications. Programmers should not have to rewrite applications, but must be able to improve them by using the social network. Basically, the social network must be extension to programs and not vice versa.
- The system needs to be used by groups of people who are already in a social network, without having to create new accounts. Users, especially the less technical, are often reluctant to try a new social networking service.
- Being primarily a system for mobile devices, it must run on restricted operating systems. To protect the users' data, many operating systems for mobile devices impose powerful security rules on applications. Such systems are iOS and Windows Phone 7 [12]. Android is a more permissive. To run on such a system, XSN needs to be designed differently from an application running on a standard operating system.
- The system must have minimum interference with the operation of mobile device. It is unacceptable that a call cannot be made because the social network system uses too many resources.
- The system must also take into account that the device runs on battery; although processing resources are quite large, their intensive use will quickly discharge the battery, which is unacceptable for mobile devices.
- It is required that users that are on the same network (LAN) to be able to communicate with each other even if the Internet access is temporarily unavailable. This means that users need to be able to authenticate event when access to the authentication server is not available.

During the description of the proposed system, we will make comparisons with existing systems, especially with the Diaspora *, which is based on some of the principles listed above. In Figure 1 we show the positioning XSN compared to other existing services, in terms of user authentication and data storage media.

It can be seen that it is positioned in the middle, between Facebook, Google + Twitter or LinkedIn using centralized storage of data (in terms of the user), and Diaspora *, which is a distributed system that uses users' personal computers to store the shared data.

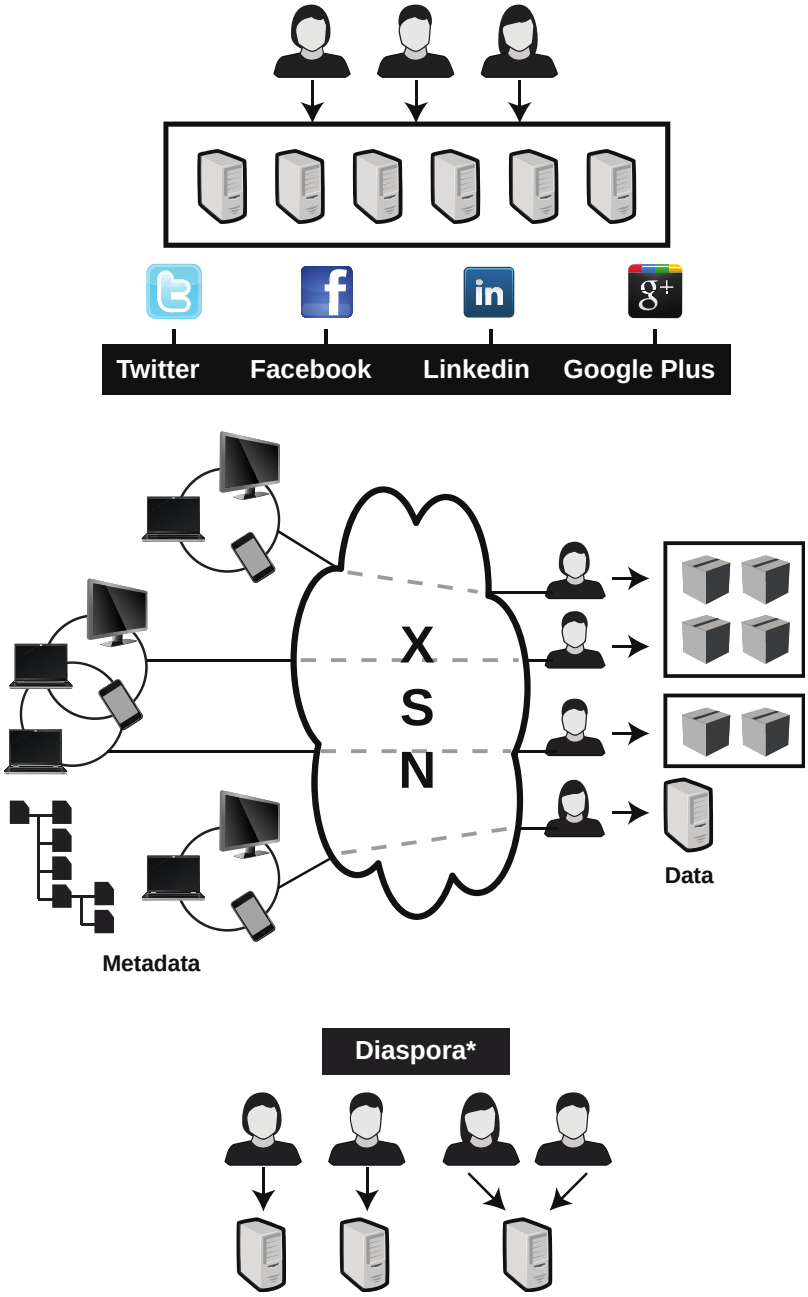

Fig. 1. XSN positioning compared to other existing social networking services.

Usage of the XMPP protocol

To implement the proposed system, we have used the XMPP protocol. This is a standard instant messaging protocol, originally designed under the name Jabber.

XMPP assigns each user an identifier, called JID. It is composed of three parts: user name, the server where the user has registered the account and resource. The latter is a text that identifies the program used for connection. A user can connect simultaneously using more programs. An example of JID is *alice@wonderland.network/fastmessage*, where *Alice* is the username, *wonderland.network* is the server and *fastmessage* is the program.

The resource name is important because the same user using XMPP can connect using multiple clients. Thus, Alice is able to connect both from the mobile device and the PC. In XSN, each user is identified by his JID, also called XSNID.

For XSN, we used XMPP authentication, routing and storing friends list. XSN uses the same identifiers as XMPP.
The XMPP protocol allows users connected on different servers to communicate with each other. Thus, using XMPP, enables XSN interconnection users, no matter what server use. XMPP servers will authenticate each other using certified public key algorithms [13]. This eliminates the possibility of a counterfeit server. If the servers would not authenticate, anyone could create a server with a false address and pose as authentic server. In Figure 2 shows Alice and Bob that have accounts on different servers (*wonderland.xmpp* respectively *movies.xmpp*) and can talk to each other. They access the network using multiple devices simultaneously. Moreover, one can notice that XMPP servers communicate directly with each other.

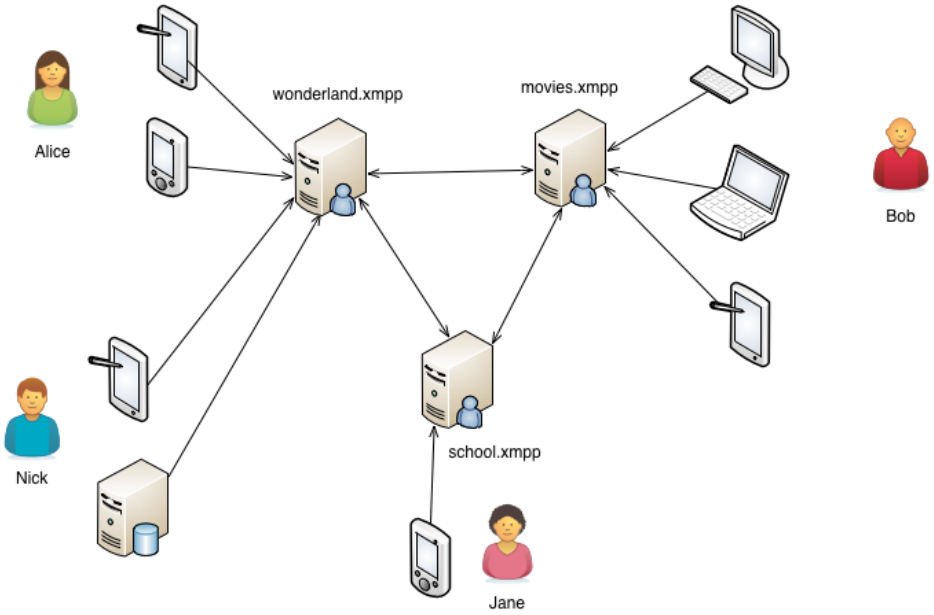

Fig. 2. The architecture of the XMPP protocol

XMPP allows each user to store a list of friends, called a *Roster*. This is actually a list of JID's. Each JID may have a name associated. An important property of the list of friends is the ability to group friends. Moreover, the groups are actually labels; a user can be placed into several groups. XSN uses this property to store the friends list. In order not to interfere with XMPP service, XSN friends are placed in groups with the name beginning with xsn:. Examples of group names are xsn: friends, xsn: work, etc..
In most current systems of social networks, friendship relationships are mutually agreed by the two users. XSN proposed the system used by Google +, namely friendship relations are established unilaterally (one person can be placed on the list of friends without requiring his consent). Access to information is one-sided. For example, if Alice has Bob in her list of friends, that means that Alice gives Bob's right to have access to certain information, but Alice can access Bob's information only of Bob has her in his friends list too.
An additional security feature is the use of public key infrastructure (PKI) system. Besides the JID, XSN assign each user a pair of keys that will be used to authenticate users without access to the XMPP server. The key pair is generated automatically by the XSN client the first time a user adds a friend to his friends list. Thus, the client will check whether there is a group in the Roster whose name begins with xsn:. If so and the client does not have key pair, it means that there are at least another XSN client used by the user that has it. The new XSN client will try

to download the key from the older client. If there is no group beginning with xsn:, the client will generate a new key pair.
When adding a new user in the list of friends, the client will send the public key to the new user's client. Thus, each user will have to hold the public keys of all his friends. The public key of a user is not secret anyway; basically anyone must be able to obtain it.
There are many extensions to XMPP protocol, some of which can be very useful for a social network service (for instance data storage on the server side, vCard stored on the server card, extended friends list, etc..). To be able to be used by any user, regardless of the server where the user has an account, XSN does not use any of these extensions. XMPP standard does not require implementing any extensions, thus the use of extensions by XSN would limit the users that can access the network.
Another important property is that the XSN system does not require any modification of the XMPP servers.
Based on the XMPP protocol, we proposed a social network service where users no longer need to have an account with at a single service provider. It also starts from an existing community of users without interfering with its services. If they wish, users may also run their own serve. Also, XSN does not assume any modification of XMPP servers. Moreover, public servers, like Google or Jabber, ensure the authenticity of their users.

XSN services platform
The usefulness of social network services actually comes from the number of applications and services that they offer. In general, however, existing services tend to provide as many services as possible by giving programmers the possibility to write extensions. These extensions are dependent on the social network service provider, and make applications extensions of the social network service. The Paradigm that we propose involves the exact opposite: social network service should be thought s being an extension to existing applications. Thus, existing applications must not be rewritten. It allows users to use applications that exist already and benefit from social networking services. The best example is NutriEduc, an application developed by IRIT for monitoring and assisting people with diabetes. The application was extended so that users can share recipes and tips with friends. The use of a standard social network service the application would have needed to be rewritten. Using XSN, rewriting was not necessary, only adding one additional library was enough.
Figure 3 shows the components of the XSN platform. We have designed the XSN client so as to provide only basic functions related to authentication of users, management of the list of friends, authorizing access for applications. For any other services we designed a library that provides the interconnection between existing applications and XSN.

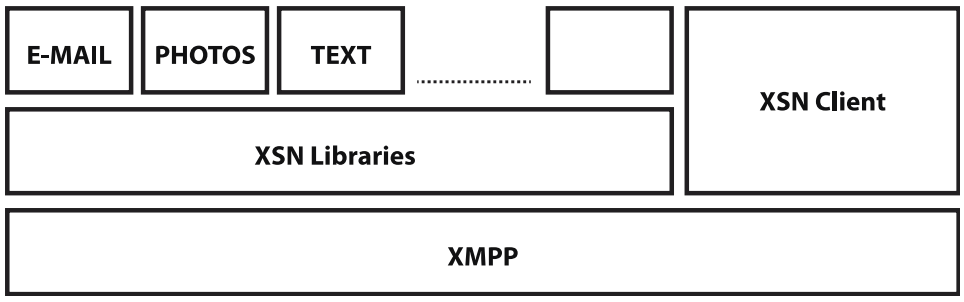

Fig. 3. The XSN platform components. XMPP protocol is used for authentication and transport. The XSN library is linking applications to the XSN network. The XSN client authorizes applications to perform actions on behalf of the user.

Mobile operating system restrictions
Because XSN is designed to operate on mobile devices we had to take into account the particularities of operating systems running

on them. We will present below their limits. We discuss here the most common three, namely iOS developed by Apple, Android made by Google and Windows Phone 7 developed by Microsoft. IOS is based on BSD [14], a UNIX version created by the University of Berkeley, Android uses the Linux kernel over which it implements a specific set of libraries and a Java virtual machine, called Dalvik. Windows Phone 7 is new and information about its foundations is not yet publicly available.

Except Android [15], all these systems are very restrictive. The most important aspects of this are:

- The system can run one application at a time. Although all three systems are do *multitasking*, applications made by programmers are not allowed to run in parallel. Windows Phone 7 does not accept any exception to this rule. iOS, since version 4 (and 4.2 for iPad), accepts restrictive *multitasking*. Thus, applications can run in parallel, but only the application that is present on the screen gets the processor. Background applications retain memory and other data, but do not receive processor. The only way to process the background data is to *ask* system to do that. The system can play audio files, listen to a VoIP socket or notification geographical location changes. Android allows running of GUI programs and services and allows background processing without restrictions. However, Android does not recommended massive background processing or maintaining active sockets. These restrictions are imposed because, although devices have relatively high processing power, the batteries that they are equipped with are not powerful enough.
- Memory limitations of the devices. Because devices need to function quickly and efficiently operating systems on mobile devices do not use swap space. This implies that the memory used by all applications is limited to RAM size. The three systems apply different rules for memory overruns. iOS considers that the currently running program must manage very well its memory. Thus, if the memory used approaches the limit, the system will notify twice running program, after which, if it does not erase some data, it will forcibly stopped. Android will do vice versa, rather than close the program running on screen, it will close background programs. GUI programs will be closed first, and if there is still not enough memory, the system will close programs running as services. Windows Phone 7 does not specify what it does to free memory.
- Programs run sandboxed, each with its own directory. Other programs cannot access its files. iOS and Windows Phone 7 make no exception to this rule. Android has a more complicated system. Devices running Android are able to access data card (usually SD card). Storage space on the card is available to all applications. They can read and write anywhere, regardless of which application the files belong to. The system will inform the user at the application's installation time about the ability to access the SD card and will require the user to approve it.
- Exchange of data between programs is strictly monitored and regulated by the system. The policy applied by the systems is that each program to should run alone, without changing data or information with other programs. Sometimes, however, there is a need to transfer data between programs. iOS allows this via clipboard (copy - paste) or by functions associated with different URLs. The first variant not very effective and safe as the information passes through a public text buffer and any application can access it. How two applications cannot run simultaneously, the user will have to make the transition from one to another. The second option is associating a program that can open certain types of URLs. When an application asks to open a URL, the system will start the program that was associated with the URL and will send

the URL to that program. After finishing, the can send back data to the first program, who originally requested URL opening. Windows Phone 7 does not yet allow the exchange of information, but presumably it will be implement in the future. Android is more permissive. Information exchange can be done either through the Binder system that transmits Java objects that implement the *Parcelable* interface from one program to another, either by *Android Interface Definition Language* (AILD). Binder sends Java objects between programs, wrapped in *Intents*. AIDL is used to control services by function calls. In essence AILD is a RPC system. It is based on Binder, but allows the definition of control functions, not just objects passing [16].

- Once compiled programs cannot be modified or extended by plug-ins. Two aspects determine this: one commercial and one about security. At compilation, each application is digitally signed. When installing and running the system will verify the application's digital signature. This system will ensure that the application was not altered (e.g. virus) since compilation. Moreover, the digital signature allows manufacturers to prevent installing applications that were not approved and that don't go through their distribution system. Android is the only system that is more permissive in that it does not require manufacturers to distribute their applications through their distribution solution, but also does not allow modification of applications or adding components later. iOS even forbids programs from running scripts downloaded over the Internet.

Although XSN is designed to run on mobile devices, it is assumed that the user will have other devices that are connected to the Internet. This includes computers, laptops, smart TVs, etc.

With a clear view of the restrictions on mobile systems, we started from the following premises:

- XSN will communicate with other applications by using short messages at long intervals, so that the iOS URLs passing system to be enough.
- XSN will not store many data on the mobile device and, more importantly, will not be able to make massive background data transfer.

The XSN Client and the applications platform

XSN is running on mobile devices, which means that it is subject to restrictions imposed by the mobile operating systems. Although Android would allow an easier implementation, XSN must be run on other devices as well, because their market share is considerable. It is less likely that Apple and Microsoft will change their mobile operating systems restrictions, and even Android begins to impose some restrictions. As described above, XSN should be a simple client that allows existing applications to use social network services.

A solution is to integrate XSN into applications as a plugin, but this is not possible due to restrictions of the operating systems running on mobile devices. Another possible option would be the integration of XSN in the applications' source code. The idea is not bad, but raises a security issue. If XSN is integrated directly into the applications' source code, in order to access the user's account, applications must receive the user's JID and password associated with that account. It is a major security risk, because no one can guarantee that the application will use this data only for the purpose stated. For example, Google accounts provide support for XMPP (Google Talk client is actually an XMPP client). If a Google user would use an application and should give his username and password, the application could not only use the XSN system, but also would be able to read the user's e-mail or to pay using the user's credit card (if the user uses Google Checkout and) etc.. This is not acceptable, so an innovative solution was needed.

Assuming that the programs may not be extended, and two programs may not run in

parallel, we designed a platform for applications that relies only on delegation of responsibility. In the classical approach, applications for social networking services use the user's account to communicate on his behalf. In our proposed platform, applications are also XSN network users, meaning that are associated with an XSN account (JID). Installing an application by the user requires, in fact, adding it the friends list. During runtime, the application itself will use its XSN account but will have to be authorized to perform certain actions on behalf of the user.

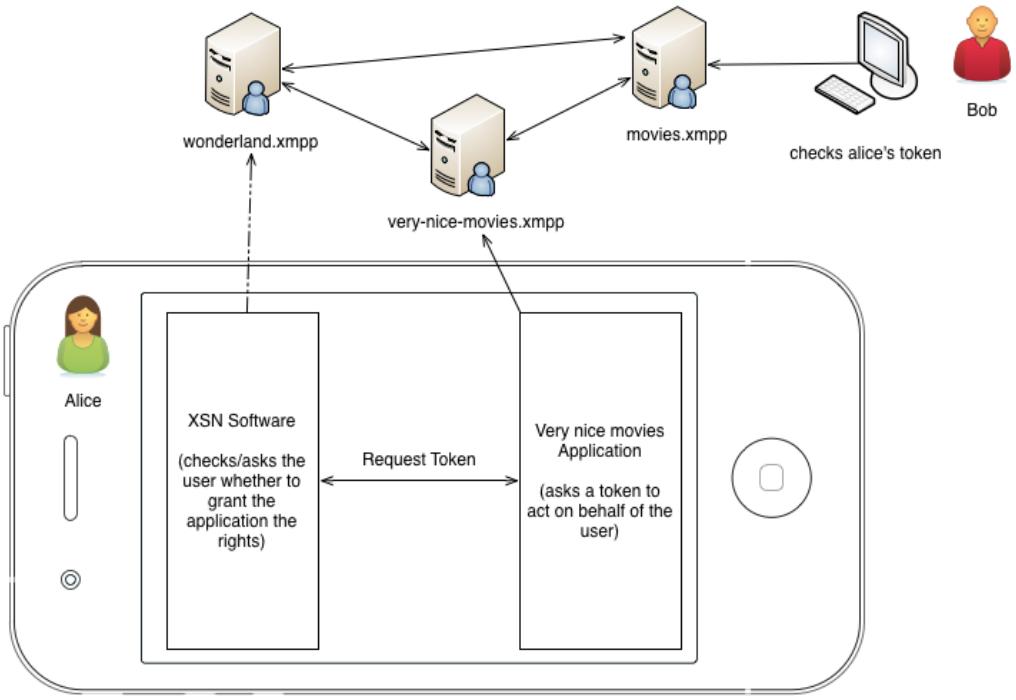

Fig. 4. The responsibility delegation to other applications

Whenever the application wishes to perform an action on behalf of the user, for example to store a file, to read a file, to communicate with another application, it must obtain an authorization from the XSN client located on the same device or from a remote XSN client. Authorization means granting a certificate signed with the user's private key stating that the application is allowed to take the desired action. Figure 4 shows how the application

Very nice movies asks Alice's XSN client to communicate with one of Bob's applications. Once received the certificate, called *token*, the application makes the request to Bob's application. Bob's application checks the authenticity and validity of the certificate using Alice's public key and verifies that the certificate granted the application to take the desired action. One can also clearly notice that the application has a separate XSN account on the *very-nice-movies.xmpp* server.

This method has some disadvantages as well. First, communication is slow as, for various actions, the application must obtain the XSN client's approval. Moreover, if another application wants to obtain some data from a user, it must first ask user's XSN client or his friends' XSN Clients about the JID of the desired application that holds the data.

XSN system services

The XSN system offers developers a platform over which they can build other services. An XSN service is actually a standard. A name and an identifier that is a string identify it. The identifier is used for the certificates. The XSN platform architecture is

shown in Figure 5.

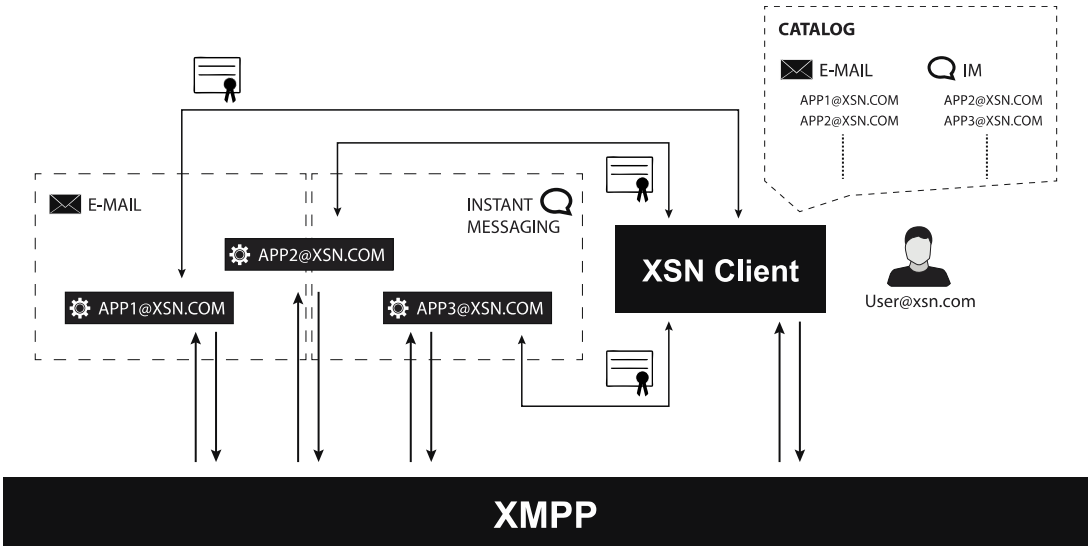

Fig. 5. The XSN system architecture. Authentication and transport is provided by XMPP. Each application has its own JID and communicates directly with XMPP. To perform actions on behalf of users, applications must obtain a certificate from the XSN client. An application may provide one or more services. A service can be offered by several applications.

A service can be offered by several applications. We take the example of the messaging service in Figure 5. Suppose that its ID is *XSN: messages*. For this service, Alice can use *WonderlandPost* application with the JID app1@xsn.com or *PostalApp* with the JID *app2@xsn.com*. Bob can use *MoviesPost* with the JID *post@movies.xmpp*. Although they are different applications, and most likely programs written by different programmers, they provide the same service, which means that Bob can read messages sent by Alice.

An application can, in turn, provide more services. For example, *PostalApp* (*app2@xsn.com*) can provide the e-mail service {*XSN: messages*} and instant messaging service {*XSN: im*}. For each one of the services, the *PostalApp* program must ask the approval of Alice's XSN client application and receive a certificate that allows it to offer this service on behalf of Alice. When installing an application, the user can decide what services he wants to use.

A user can install multiple applications that offer the same service and decide which to use. Of course, he can use both.

The services communication system

Due to the fact that each application has its own XSN ID, we face the following problem: Alice's *WonderPost* application, which provides *XSN:messages* service to send messages, must communicate with Bob's mail service, namely *MoviePost*. How does Alice know what is Bob's mail service JID? We hape proposed the following solutions:

- If at least one of Bob's resources (XSN Clients) is connected, *WonderPost* will select the one with the highest priority and will require additional information on Bob's postal service. Bob's XSN client will check whether *WonderPost* has a valid certificate from Alice, and if so, will provide Bob's service's JID. If Bob has two applications installed that provide the same service, one of them will be chosen randomly.
- It is important to note that the JID that Alice receives must be complete, which is of the form *name@server/resource*. The resource identifies, in fact, very precise the application that should receive the message. We refer here to the actual program running on one of Bob's devices.

- In the event that none of Bob's resource is connected, *WonderPost* will try to ask all of Alice's friends if they know who is Bob's service. Every friend of Alice will check the application's certificate to ensure that, in fact, Alice has authorized *WonderPost* to know which application is Bob's mail service. If one of Alice's friends has the information (which is a mutual friend of the two), it will respond with Bob's application's JID, but without the resource, because it has no way of knowing what Bob's running resource is (or even if it runs). Once *WonderPost* has learned Bob's mail application JID, it has two solutions: either sends a query message to the received JID querying which is Bob's resource, which means that application's makers should remember every time a resource is *online*. Another solution for *WonderPost* is to send the message using the *mailbox* system provided by the storage service [17].

Offline authentication in the temporary absence of Internet access

A problem that occurs when using XSN is the continuous need of Internet access. To communicate with users and their applications XSN must have access to the XMPP servers. The problem is not just for XSN, all social networking services have this problem, even any program that needs a connection to a server. In general, the problem has no solution because while no Internet connection, a user cannot physical communicate with users that are in another place. But the question arises what happens to users that are in the same location, for example in a restaurant. Following studies have proposed solutions to interconnect them, but these solutions cover only the operating system related problems [18]. XSN propose a solution, which assumes that there is a local network between devices, even if Internet is temporarily inaccessible.

Discovering the local network users is done using Bonjour [19] or Zeroconf [20]. These systems allow publication of information about services offered in the local network without needing a server. Once discovered services, users can connect to each other. For authentication, users will use public key system. I have described above that adding a friend on the list, also means getting his public key. Thus, if two users have a bilateral friendship, both have locally stored each other's public key. To connect with each other and to authenticate, they will use these keys for authentication.

If two users that are not mutual friends they will apply the following algorithm for authentication: one of the XSN clients will generate a password that the second user will have to introduce. This implies that the two are side by side or have another way to communicate with each other. Devices that use the Bluetooth system use this system. Once authenticated, users will change public keys with each other and will check these keys when access to the Internet is restored.

Thus, if support for local network interconnection is implemented in the devices' operating system, XSN enables user authentication. Of course, access to services that require Internet connection, such as, most likely, the service for data storage) is not possible. Support for interconnection is implemented iOS as Bonjour and in Android as Zeroconf and, in part, by a library called jmDNS [21] which implements the functions of Bonjour's multicast DNS server. Windows Phone 7 does not specify whether it supports or not local network interconnection.

4 Implementing XSN

We have implemented the XSN client and library for most of the mobile devices that exist on the market. We have chosen Android and iOS devices as they have together almost 70% of market share.

A technical problem arises when implementing XSN for iOS. The most restrictive system, iOS does not allow running programs in background, thus the XSN client cannot process background data. It has yet to run in order to respond to messages about services or applications queries. The technical solution is to register *socket* as VoIP. Thus, XSN will be notified by the operating system whenever it receives data. XSN can thus respond

quickly to various requests coming from the network. Moreover, if a service needs to communicate directly with its correspondent from the user's device, XSN can notify the users who can start the program.

We have done tests using 4th generation iPods, iPads and iPad 2. All are running on Apple's 1 Ghz A4 and A5 processors having from 256 MB up to 1 GB of RAM. Tests have shown that the XSN client has minimal overhead, as is does not run continuously.

Android on the other hand allows services, thus implementing the XSN client was no problem. In order to communicate with the XSN library from applications, we use Android Interface Definition Language (AIDL).

The testing has been done using HTC Hero devices that are more than two years old. It can be said that XSN runs acceptably, but there is still a need for improvement.

In order to test our implementation, we are now extending the NutriEduc application built by Institute de Recherché on Informatique de Toulouse (IRIT). Its purpose is to monitor and assist persons with diabetes. XSN will allow the sharing of recipes and the storage of sensitive information.

5 Conclusions

Upon designing and implementing the XSN system we have come to the following conclusions:

- Using the XMPP protocol platform has two advantages. Unlike traditional social network services like Facebook, LinkedIn or Twitter, XSN does not require users have an account with the same provider in order to communicate with each other. This solves the problem of interconnection of users. Moreover, because many of the existing services are already using XMPP, including Google Talk, there is already a community of users that can use XSN without the need to make other accounts. It can also be used in industrial applications such as those described in PREMINV platform \ cite {article: ve_sibiu_2009}.
- XSN is not using any XMPP protocol extension so that service can run on any server, regardless of what extensions it implements. Moreover, we have not made any changes to XMPP servers so that users can use either your own server, or a public one. This feature brings an advantage on the distributed social network service Diaspora*, which forces the user to use his personal server in order to connect to the network.
- Authentication in XSN is secure, as servers authenticate each other. Moreover, we strongly believe that users will be more willing to user public available XMPP servers, such as Google or Jabber, to create accounts for XSN.
- XSN proposes a method to authenticate users and provide services to some extent in the temporary absence of connection to the Internet.
- The system works well on mobile devices that run restrictive operating systems, such as iOS. As two applications cannot run in parallel and applications must not get the user's authentication data, cryptography is used in order to grant applications to act on behalf of the user, using their own XSN authentication data.

References

[1] T. A. Pempek, Y. A. Yermolayeva, S L. Calvert, "College students' social networking experiences on Facebook", *Journal of Applied Developmental Psychology*, 30(3), 2009, pp. 227 – 238.

[2] P. Saint-Aandre, K. Smith, R. Troncon, *XMPP: The Definitve Guide*, O'REILLY, 2009.

[3] D. Mark, J. LaMarche, *Beginning iPhone 3 Development: Exploring the iPhone SDK*, Apress, 2009.

[4] D. Mark, J. LaMarche, J. Nutting, *More iPhone 4 Development. Further Explorations of the iOS SDK*, Apress, 2011.

[5] M. Murphy, *Beginning Android 2*, Apress, 2010.

[6] S. Spanbauer, "The right social network for you", *PC World*, 26(4), 2008, pp. 105-110.

[7] Facebook. *Statement of rights and responsabilitie,*

http://www.facebook.com/terms.php, accesed in April 2010.
[8] L. Davis. *Facebook plans to make money by selling your data*, February 2009, http://www.readwriteweb.com/archives/facebook_sells_your_data.php, accessed in September 2011.
[9] J Fogel, E. Nehmad, "Internet social network communities: Risk taking, trust, and privacy concerns", *Computers in Human Behavior*, 25(1), 2009, pp. 153-160.
[10] A. Narayanan, V. Shamtikov, "De-anonymizing social networks", *IEEE Symposium on Security and Privacy*, 2009, pp. 173-187.
[11] J. C. Buisson, "Nutri-Educ, a nutrition software application for balancing meals, using fuzzy arithmetic and heuristic search algorithms", *Artificial Intelligence in Medicine*, 42(3), 2008, pp. 213-227.
[12] H. Lee, E. Chuvyrov, *Beginning Windows Phone 7 Development, second edition*, Apress, 2011.
[13] C. P. Pfleeger, S. L. Pfleeger, *Security in computing, third edition*. Prentice Hall, 2003.
[14] S. J. Leffler, *The Design and implementation of the 4.3 BSD UNIX operating system*, Addison-Wesley, 1989.
[15] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, *Google android: A state-of-the-art review of security mechanisms*, CoRR, abs/0912.5101, 2009.
[16] A. Radovici, *"Implementation of a Web Server Optimzied for Services for the Android Platoform" ("Implementarea unui server web pentru platforma Android optimizat pentru oferirea de servicii"), Master Thesis*, "Politehnica" University of Bucharest, 2010.
[17] A. Radovici, *"Contributions to Social Networking Services for Mobile Devices" ("Contributii la service de retele sociale pentru dispozitive mobile"), PhD Thesis*, "Politehnica" University of Bucharest, 2011.
[18] B. A. Ford. "UIA: A Global Connectivity Architecture for Mobile Personal Devices", *PhD Thesis*, Massachusetts Institute of Technology, 2008.
[19] Apple. *Bonjour*, http://developer.apple.com/opensource/, accesed in September 2011.
[20] D. H. Steinberg, S. Cheshire, *Zero Configuration Networking: The Definitive Guide*, O'Reilly Media, 2005.
[21] SourceForge. *Jmdns*, http://jmdns.sourceforge.net/, accessed in September 2011.

Alexandru RADOVICI has graduated the Faculty of Engineering in Foreign Languages of the "Politehnica" University of Bucharest in 2008. He holds a master in Computer Science since 2010 and has finished his PhD in Computer Science in 2011 at the same university. Since 2008 he is a member of the Department of Engineering in Foreign Languages and since 2010 he has been teaching the first Mobile Devices Application Development course in his university. The course covers topics from almost all mobile device operating systems, such as Android, Windows Mobile, Windows Phone, iOS etc. His topics of interests are mobile devices, operating systems and devices programming.

Valentin CRISTEA is Professor of the Computer Science and Engineering Department of the University Politehnica of Bucharest. His main fields of expertise are Large Scale Distributed Systems, Grid and Cloud Computing, and e-Services. He teaches courses and supervises PhD students on these topics. Valentin Cristea is Director of the National Center for Information Technology, leader of the CoLaborator, Distributed Systems and Grid, and e-Business/e-Government laboratories. He has a long experience in the development, management and/or coordination of international and national research projects. Valentin Cristea is member of the IEEE and ACM professional organizations.