Yarta: A Middleware for Managing Mobile Social Ecosystems*

Alessandra Toninelli, Animesh Pathak, and Valérie Issarny

INRIA Paris Rocquencourt, France
{alessandra.toninelli,animesh.pathak,valerie.issarny}@inria.fr

Abstract. With the increased prevalence of advanced mobile devices (the so-called "smart" phones), interest has grown in mobile social ecosystems, where users not only access traditional Web-based social networks using their mobile devices, but are also able to use the context information provided by these devices to further enrich their interactions. In complex mobile social ecosystems of the future the heterogeneity of software platforms on constituent nodes, combined with their intrinsic distributed nature and heterogeneity of representation of data and context raises the need for middleware support for the development of mobile social applications.

In this paper, we propose Yarta, a novel middleware designed for mobile social ecosystems (MSE), which takes into account the heterogeneity of both deployment nodes and available data, the intrinsic decentralized nature of mobile social applications, as well as users' privacy concerns. To validate our approach, we show how we developed two mobile social applications over Yarta, and report on both its efficiency and ease-of use by way of extensive evaluation on smart phones and laptops.

1 Introduction

Social ties such as friendship, common interests, and shared professional activities are central to humans as these ties bind individuals together. This web of social bindings is referred to as a *social network*, and is the focus of so-called *social applications*, i.e., applications that support human social interactions and are characterized by their swarming, transitory, and informal qualities [4, 8]. Recent advances in wireless network technologies and the increasing diffusion of smart phones equipped with sensing capabilities represent a unique chance to enhance social applications and make them truly pervasive in everyday life [4, 14].

A salient feature of these situations is that physical places can also act as social filters. For example, a conference venue groups together attendees belonging to different organizations, coming together and socially interacting in a number of ways, such as listening to and making comments on talks and presentations, setting up scheduled and spontaneous meetings, exchanging technical knowledge and extending their professional network. Similarly, people attending a football

* Funded in part by an ERCIM *Alain Bensoussan* Fellowship Programme project.

J. Riekki, M. Ylianttila, and M. Guo (Eds.): GPC 2011, LNCS 6646, pp. 209–220, 2011.

match generally share an interest in football and sport, and support the same team, especially if they find themselves in physical proximity. We propose the term *mobile social ecosystem* (MSE) to describe this richer set of interactions occuring between the participants in these situations. In MSEs, the heterogeneity of software platforms on constituent nodes, combined with their intrinsic distributed nature and heterogeneity of representation of data and context raises the need to support the development of *mobile social applications* (MSAs).

As discussed in the next section, several MSAs and supporting middlewares have emerged recently. However, developing mobile social applications is still a challenging task for several reasons:

Lack of consistent API for social sensors. Today, application developers need to directly interface with sources of social information (also known as *social sensors*), such as the user's contacts list or call log on the phone, or his personal or professional profile in online social networking sites. Although popular social applications, such as LinkedIn[1] or Facebook[2], have recently allowed to export the user's profile in a standard format, a common platform to mediate access to all social data by different applications is still lacking.

App-specific data silos. Related to the above, the social data collected by today's MSAs are not designed to be accesible to other MSAs. E.g, Facebook owns users' data that might be reused by other applications only if those applications are integrated into it. Similar considerations hold for most mobile social applications, such as [12] and [1]. As a result, current mobile social applications produce and manage their own data, which can be imported to other applications only with considerable effort, raising issues such as semantic consistency.

Lack of advanced social access control mechanisms. Since MSAs manage contextual data such as mobility traces, user preferences and activities, which are sensitive per se and can be further used to infer sensitive information, it raises critical security issues, particularly in terms of privacy and access control of users' data. The task becomes even more difficult given the networked nature of mobile social applications, where information comes from multiple sources, moves to multiple destinations (possibly unforeseen at information production time) and is linked to other information following unpredictable patterns. Recent solutions have shed light on these issues, and made a relevant, albeit only initial, effort to tackle the problem [5].

Dependence on centralized solutions. Ubiquitous environments are naturally decentralized, and users must be able to access their social data anywhere and anytime, regardless of access to the Internet. In addition, a global view of users' MSE may not (always) be available, while privacy and portability issues might discourage approaches based on MSE data replication on each user's device [5]. Centralized architectures used by current Web social applications and platforms, thus, are not appropriate for the mobile setting, nor their extensions

[1] `http://linkedin.com`
[2] `http://facebook.com`

to support access from mobile devices, which always rely on some server to store data and manage users' interactions.

To address the above challenges, in this paper we propose Yarta, a novel middleware support platform for mobile social applications (or an *MSE management middleware*). In particular, (i) Yarta is based on an expressive and extensible model to represent MSE and the interactions possible in them, which acts a semantic interoperability platform by enabling different applications to share and reuse their respective knowledge. This supports interoperability between separately developed applications. (ii) It implements a set of components to help social application developers manage their MSE by allowing social information storage and retrieval, and provides a set of support tools to generate code based on the application data model. (iii) It supports access control to the user's social data based on socially aware policies, i.e., it allows each user to customize access to his/her social data based on social information itself, as well as context. (iv) It provides a versatile support for decentralized mobile social applications, which enables execution on both computers and smart phones with minimal configuration effort, and seamless communication over heterogenous wireless networks, allowing users to maintain social data local to their device(s).

The rest of the paper is organized as follows. In Section 2, we highlight the differences of our work with respect to existing work in the domain. Our contributions are discussed in detail in Section 3, where we discuss the architecture of Yarta. Section 4 provides the implementation details of our middleware, as well as an extensive evaluation of the prototypes in terms of performance, scalability, as well as ease of use in using Yarta to develop applications. Section 5 concludes with a sketch of our planned research in the near future.

2 Related Work

Current MSAs, such as applications supporting the dissemination of content updates (e.g., news or traffic information) over a mobile social network [12], or exploiting mobile social networking to enhance group communication [9], are often designed from scratch by embedding into the application logic all MSE management functionalities and providing application-specific data representation models. In this section we focus on existing platforms and middleware architectures for supporting mobile social applications. An overview of related research regarding MSAs and MSE modeling can be found in [20].

Some authors have recognized the need to externalize social management functionalities [17],[21], but to the best of our knowledge, only a few middleware frameworks to support MSAs have been proposed. Table 1 summarizes the main contributions and drawbacks of existing solutions with respect to the challenges discussed in Section 1. The MobiSoc [10] and MobiClique [16] middleware provide simple data models compared to Yarta MSE model, while the IYOUIT application offers a wide set of concepts to model users' activities and interactions [3]. IYOUIT cannot be considered a middleware since it is provided as a stand-alone application, while MobiSoc and MobiClique both provide open APIs.

Table 1. Comparison of middleware for supporting mobile social applications

Middleware	MSE Model	Privacy & Access Control	MSE Mgmt Features	Decentralized Architecture	Reusability
MobiSoc	People, Place People-to-People, People-to-Place	Authentication, Confidentiality (Encryption)	Social Data Inference, Event Mgmt.	No	Open APIs
MobiClique	User Profile (specific format)	No	Proximity-based Social Interaction Data Extraction	Yes (Opportunistic Networks)	Open APIs
Middleware PSC	Augmented FOAF (Tasks, Preferences)	No	User's Task Matching	Optional	N/A
IYOUIT	Location, Experience, Pictures, Interests, Buddies, ...	Permissions (identity/group)	Social Data Inference, Friends Network	Distributed	Proprietary
Prometheus	People, type of relation, strength	Privacy (Encryption) Access Control, Trust	Social Inference over Multiple Sources	Yes	Open APIs
PrPl	Multi-application	Authentication (OpenID)	Access to social data, query	Decentralized	Open API, SociaLite Language
Yarta	Agent, Event, Place Content, Topic (extensible)	AC Policies, Authentication, Confidentiality	MSE Creation, Update, Exchange, & Extraction	Multi-radio Multi-platform (iBICOOP)	Open APIs (LGPL)

The middleware for Pervasive Social Computing proposed in [1] adopts a similar model to Yarta (RDF-based), which however lacks in generality. In addition, the middleware is not available for reuse. Also worth mentioning is Motorola's soon-to-be-discontinued MOTOBLUR[3] UI, which aggregates updates from a user's social networks and allows posting in multiple places. The PrPl architecture, targeted to decentralized environments, allows users to access (and share) data stored by different social applications by a special purpose query language [18]. Prometheus is a P2P architecture that collects and manages social information from multiple sources and implements a set of social inference functions [15].

Yarta differs from all previous approaches since it allows the exchange of social graphs between users, and between applications, in an interoperable format, which also allows reasoning. Additionally, to the best of our knowledge, Yarta is the only MSE middleware that can fully execute on mobile phones. Finally, its access control model is socially aware and takes advantage of semantic reasoning.

3 The Yarta Middleware

To address the challenges discussed in Section 1, we have developed the Yarta middleware architecture, consisting of two layers (see Fig. 1): the *MSE Management Middleware* layer, managing and allowing access to MSE data, and the *Mobile Middleware* layer, handling low level communication/coordination issues.

3.1 Managing Knowledge in Mobile Social Ecosystems

Yarta is based on an expressive and extensible model to represent MSE, which acts a semantic interoperability platform by enabling different applications to share and reuse their respective knowledge. We first describe the data model, and then present features offered by the middleware component in charge of managing the user's Knowledge Base.

[3] http://www.motorola.com/Consumers/US-EN/Consumer-Product-and-Services/MOTOBLUR/Meet-MOTOBLUR

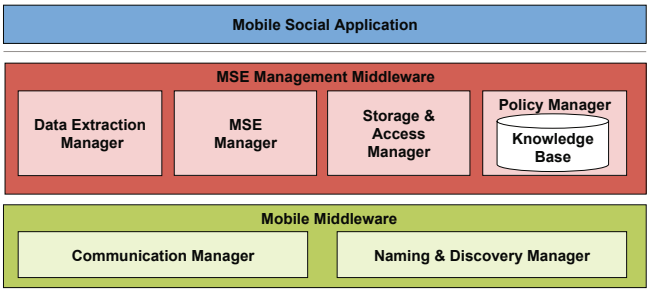


Fig. 1. Yarta Middleware Architecture

MSE Representational Model. Our model is based on the representational model in [20], to which we have added more details, such as a name, a latitude and a longitude value for each *Place*, as suggested by most existing geo-positioning standards[4], and unique IDs for resources[5]. Note that this model re-uses concepts defined in existing social information and content ontologies such as Friend-of-a-Friend (FOAF) [7] and the Dublin Core [6].

Both the base model defined in [20] and our augmented model discussed above are represented using the Resource Description Framework (RDF)[6], a base Semantic Web standard. Because all mobile social applications share a common data model, they rely on a shared platform of common meaning, which they can further extend based on specific requirements: by means of automated reasoning, classes and properties defined in application-specific extensions are put in clear semantic relation with base classes, thus enabling data interoperability. We provide a detailed example of this in Section 4.2.

Accessing and Processing Social Data. Data represented according to the above model are connected to form a uniform graph of social information. This is well suited to a scenario where each user owns his data locally, and autonomously manages his graph by possibly adding to it portions of graphs provided by other users, as well as by other sources of social information (see Section 3.2). The social graph is managed by the *Knowledge Base* (KB) middleware component.

The KB offers both low-level (RDF-oriented) as well as high-level APIs to access, update and remove social data. These functionalities are provided independently of the specific implementation chosen for semantic data management. The KB is also able to handle the merging of MSE graphs coming from different users. Finally, the KB is wrapped by a *Policy Manager* to ensure access control enforcement according to the policies defined in the system, as explained later.

[4] `http://www.w3.org/2003/01/geo/wgs84_pos#`

[5] Due to lack of space we refer the refer the reader to [20] for the graph of first-class entities and relationships.

[6] `http://www.w3.org/TR/rdf-primer/`

3.2 Providing MSE Management Functionalities

Yarta supports mobile application developers along three directions:

Knowledge Abstractions for Application Developers. Yarta provides an access interface to the KB from the user's perspective. Specifically, it allows the user application to add and retrieve information to/from the KB by means of high level queries hiding the details of the internal RDF representation (e.g., nodes or triples). It also allows the execution of remote queries, performed over the KB of another user. This is done via a dedicated component, namely the *Storage & Access Manager*. In particular, the task of providing a high-level abstraction of the knowledgebase to the application developers is divided into two parts. Firstly, the developer is provided a `ResourceFactory` class which he can use to create new instances of resources. These objects in turn provide the basic getter and setter methods of a Java Bean for each of its properties, as well as properties inherited from its superclasses.

Secondly, the Storage and Access Manager also provides the users with methods to add/delete/query the relationships between resources as defined by the model (e.g. `addMember(Agent,Group)`, `getMembers(Group)`, `getMembers_ inverse(Agent), and isMember(Agent, Group)`). Additionally, the `StorageAccessManager` class also provides an `executeRemoteQuery(remotePeer, query)` method which can be used to execute a subset of these queries on the KBs of a user's peers, and add the information returned to the user's KB in accordance with the access control policies in place.

Automatic Generation of API to Access KB. We have designed a tool to alleviate the burden of developers who want to extend the core model of Yarta to develop their own applications. Specifically, this tool takes an RDF model representing the types of resources in the application and the relationships between them, and generates source files which provide an object-oriented API over the knowledgebase. The facility is akin to those provided by toolkits used by developers interacting with database systems[11]. Note that an essential feature of an API generator for Yarta is the support for *multiple inheritance*, a feature provided by RDF, but not natively supported by languages such as Java.

Extracting Data from Social Sensors. We have designed the *Data Extraction Manager* to populate the user's KB by extracting data from two distinct types of sources. The first already contain social links such as "friendship" in addition to general information, while the second do not contain social links, but may contain information which can be correlated to infer social links. For the former, adapters can be written in their API to import their data into Yarta; while for the latter, we need to employ inference algorithms to correlate data and guess/recommend social links. The structure of the Data Extraction Manager is modular to allow for plug-and-play behavior of adapters.

3.3 Controlling Access to MSE Knowledge

Yarta provides a flexible and powerful support for access control, which can be fine-tuned based on the social preferences of the user. In particular, it adopts semantic policies to define access directives. Policies are completely decoupled from both the application logic and the KB management, to allow fine-grained and customizable security behavior. Policies allow read/add/remove actions on triples or graphs (i.e., sets of triples) and are modeled based on the socially aware policy model presented in [19]. As a key feature, they define access rules to data based on social information. Policies are currently represented as a combination of SPARQL queries and RDF statements. For the sake of conciseness we do not describe the policy model in detail, but refer the reader to [19].

The Yarta middleware includes a dedicated component for the definition, management, evaluation and enforcement of access control policies over the KB, called *Policy Manager*. The Policy Manager intercepts any tentative access action on the KB, and performs reasoning on defined policies and the access request's context (e.g., relation with the requester, properties of resource, etc.) to determine whether the action is permitted.

3.4 Supporting Decentralized and Heterogeneous Environments

Yarta is designed to execute in ubiquitous environments, characterized by a high degree of heterogeneity in both connectivity options and harware platform. In addition, it does not assume any centralized server to collect and manage the user's data, nor to perform any other social functionality, as detailed below.

Communicating over Heterogeneous Networks. Yarta is supported by a multi-platform communication layer that offers both synchronous and asynchronous messaging support over multi-radio links, and supports data transfer over heterogeneous network interfaces and connecting technologies, even in case of temporary disconnections due to user mobility. In addition, it provides support for network-agnostic service/device naming and discovery. Yarta also implements base security features via authentication and confidentiality mechanisms.

Execution on Multiple Mobile Platforms. Yarta is developed in Java to take maximum advantage of portability across mobile platforms. Yarta can execute on different nodes: smart phones running Android, and laptops/workstations with different operating systems thanks to the language portability. This applies to all middleware components and actually allows the deployment of the whole platform on resource-constrained devices, without the need for an external proxy handling semantic processing.

4 Implementation and Evaluation

4.1 Implementation Details

The Yarta middleware prototype [22] is written in Java2 SE and has been deployed both on laptops running Windows/Mac OS, and on smart phones running

Android. In the laptop prototype, both the Knowledge Base and the Policy Manager rely on capabilities offered by the Jena Semantic Web Framework [13]. The KB currently uses the filesystem as a backing store. For the Android prototype we exploit Androjena[7], an Android-compatible port of the Jena framework.

As a first implementation of the Data Extraction Manager's social sensors, we wrote adapters for Facebook and LinkedIn using their native APIs. For the data sources which are not intrinsically social, we implemented adapters which used the data stored in the user's phone contacts, and correlated it with his call logs and SMS logs to draw some basic inference about the contacts whom a user "knows". This work is at an early stage, but the modular architecture of the data extraction manager is found to be suitable for easily writing more adapters, or testing better inference algorithms. To help developers in extending the core Yarta model to for their application, we have implemented the API generator discussed in Section 3 by extending the Jastor[8] toolkit. For Android phones, we also provided a wrapper of the storage and access manager in the form of a `ContentProvider`, the standard way for Android applications to access data.

Finally, we exploit the iBICOOP middleware [2] as part of our Communication Manager and Naming/Discovery Manager to provide discovery and messaging abilities. Yarta services residing on mobile devices are accessed through their iBICOOP URL (iBIURL), created by combining the userID and deviceURI from our model with the Yarta application name and the specific service needed.

4.2 System Evaluation

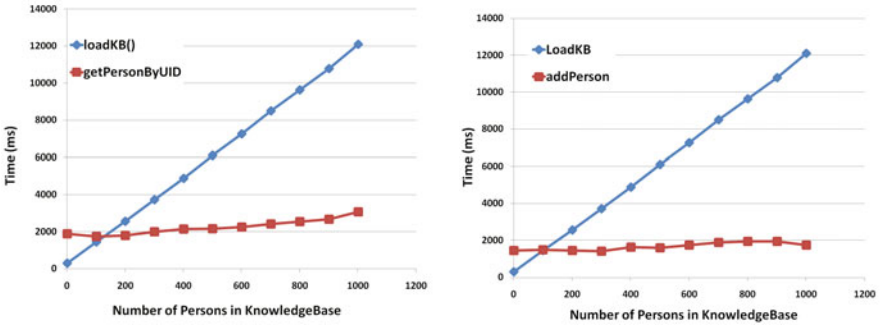
To evaluate Yarta, we collected data on both the scalability of the core features provided by Yarta, as well as the development effort involved in building new applications on top of it.

Performance and Scalability Evaluation. To evaluate the scalability of Yarta, we profiled its behavior during several operations, including those for adding a new person to the KB, adding a KB from a file, retrieving a person and his aquaintences from the KB, and performing operations on a remote node.

For each of the above methods, we measured the time taken for their execution on the Google Nexus One mobile phones running Android 2.2 OS, with a 1 GHz processor and 512 MB of RAM. We used KB sizes ranging from 1 to 1001 entries in increments of 100, with 5 applicable access control policies in place, and averaged the times taken in performing the operations for 10 runs.

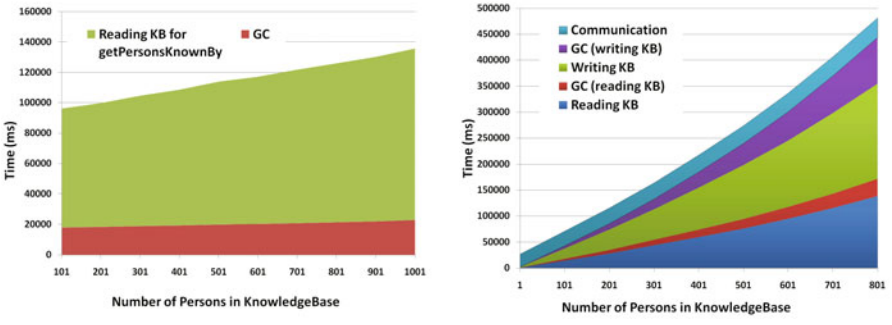
We present the observations from our experiments in Figures 2 and 3. Figure 2(a) illustrates the time taken for adding information in the KB from a file, and the time taken for reading one person from that KB after that. Note that although the times increase with the size of the KB, the time taken for the interactive `getPersonByUID` method is well within the 5 second interactive UI response time limit used by the Android OS. The same can be seen for the interactive `addPerson` method profiled in Figure 2(b). Further, Figure 3(a) presents

[7] `http://code.google.com/p/androjena/`
[8] `http://jastor.sourceforge.net`



(a) (b)

Fig. 2. Time taken for `loadKB` and then a)`getPerson` b) `addPerson`



(a) (b)

Fig. 3. Time taken for a) `getPersonsKnownBy` and b) `remoteGetAllPersons`

the amount of time taken for getting the list of persons known by someone (set to return 50 persons in the KB), which was seen to take inordinately long times for the user to wait. Since Android allows lists to be populated in an incremental manner as data flows in, this may be used to mitigate the slow response time of this operation. Finally, Figure 3(b) provides the details of the time taken in the various steps of getting the list of all the persons in the KB of another peer. We have noted the time taken for reading the remote KB and writing the information in the local KB, with the latter being higher since Jena performs duplication checks at write time. Note also the time taken by Android in garbage collection during the read and write operations. After inspecting and optimizing our code, we believe that the next step would be to improve the Androjena library so that it uses memory more efficiently. The other operations were also seen to perform within acceptable time bounds with similar profiles. On the laptop, we observed similar trends in the times taken, but the actual times taken were much less, owing to better hardware.

Developing mobile social applications on top of Yarta. To showcase the real-world applicability of our approach and the extensible nature of our middleware discussed in Section 3, we developed two proof-of-concept applications that allow mobile users to perform various social activities in different scenarios discussed in Section 1 – a *Conference Mate* app to assist the attendees of a conference, and a *FIFA* app to allow the fans attending a football match.

Extending the Data Model. For both applications we extended the Yarta data model to include concepts specific of each scenario. Extension proceeds by subclassing RDF classes/properties and possibly importing other (portions of) ontologies, provided that they are compliant with the base model. Extensions included subclasses of Event (*Talk*, *Coffee Break* and *Meeting*) for the Conference Mate app, and *Match* as a subclass of Event for the FIFA app. For the sake of brevity, we do not mention all the defined classes and properties here. After defining the data model, we created access control policies for the MSE data, including complex application-specific policies such as "any friend (person I know) that supports my team is allowed to read the list of my friends who also support the same team, as well as their affiliation to my fan club" for the FIFA app.

Writing the Application Code. The first step toward the development of an application is the extension of the Storage and Access Manager to incorporate the extensions made to the model. For this purpose, we used the automatic code generation tool described in Section 3.2. This greatly helped reduce the programmer's burden, since the $\sim$ 5000 lines of code for the new API for the FIFA app were automatically generated, for example. Overall, this auto-generated code constituted $\sim$ 70% of the total code of the apps, with the developer-written code mostly implementing a user interface over the API provided by Yarta.

In summary, our evaluations show that Yarta performs well in terms of efficiency and expressiveness, the two main desirable properties of middleware.

5 Conclusions and Future Work

In this paper, we presented Yarta, a novel middleware for supporting complex mobile social ecosystems of the not-so-distant future. Our middleware allows knowledge exchange between users and between applications, and provides flexible policies controlling access to MSE data based on users' social preferences. Yarta implements a set of components that allow social information storage and retrieval, automatic generation of code for mobile social applications, and extraction of social data from available social sensors. It can execute on smart phones and laptops and is able to communicate over heterogenous wireless networks. We demonstrated the efficacy and usability of our middleware by providing an extensive evaluation of the middleware and two prototype applications, running on Android phones, that we developed on top of it. We are currently working on the Yarta middleware along several directions, including developing more advanced algorithms to extract MSE from location/context data sets, providing the KB a database backing store, and improving performance on mobile phones.

References

1. Ben Mokhtar, S., Capra, L.: From pervasive to social computing: algorithms and deployments. In: ICPS 2009: Proceedings of the 2009 International Conference on Pervasive Services. ACM, New York (2009)
2. Bennaceur, A., Singh, P., Raverdy, P.G., Issarny, V.: The ibicoop middleware: Enablers and services for emerging pervasive computing environments. In: PerCom Workshops, pp. 1–6. IEEE Computer Society, Los Alamitos (2009)
3. Boehm, S., Koolwaaij, J., Luther, M., Souville, B., Wagner, M., Wibbels, M.: Introducing IYOUIT. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T.W., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 804–817. Springer, Heidelberg (2008)
4. Churchill, E.F., Halverson, C.A.: Guest editors' introduction: Social networks and social networking. IEEE Internet Computing (2005)
5. The Diaspora Project, `http://www.joindiaspora.com/` (last visited: May 2010)
6. Dublin Core metadata element set, version 1.1, `http://www.dublincore.org/documents/dces/` (last visited: May 2010)
7. Friend of a Friend, `http://www.foaf-project.org/` (last visited: March 2010)
8. Foth, M.: Facilitating social networking in inner-city neighborhoods. IEEE Computer 39(9), 44–50 (2006)
9. Grob, R., Kuhn, M., Wattenhofer, R., Wirz, M.: Cluestr: mobile social networking for enhanced group communication. In: GROUP 2009: Proceedings of the ACM 2009 International Conference on Supporting Group Work, pp. 81–90 (2009)
10. Gupta, A., Kalra, A., Boston, D., Borcea, C.: MobiSoC: a middleware for mobile social computing applications. Mob. Netw. Appl (2009)
11. Hibernate, relational persistence for java and .net, `http://www.hibernate.org/`
12. Ioannidis, S., Chaintreau, A., Massoulié, L.: Distributing content updates over a mobile social network. ACM SIGMOBILE Mobile Computing and Communications Review 13(1), 44–47 (2009), `http://dx.doi.org/10.1145/1558590.1558599`
13. Jena, `http://www.jena.sourceforge.net/` (last visited: May 2010)
14. Jones, Q., Grandhi, S.A.: P3 systems: Putting the place back into social networks. IEEE Internet Computing 9(5), 38–46 (2005)
15. Kourtellis, N., Finnis, J., Anderson, P., Blackburn, J., Borcea, C., Iamnitchi, A.: Prometheus: User-controlled P2P social data management for socially-aware applications. In: Gupta, I., Mascolo, C. (eds.) Middleware 2010. LNCS, vol. 6452, pp. 212–231. Springer, Heidelberg (2010)
16. Pietiläinen, A., Oliver, E., LeBrun, J., Varghese, G., Diot, C.: MobiClique: middleware for mobile social networking. In: Proceedings of the 2nd ACM Workshop on Online Social Networks, pp. 49–54. ACM, New York (2009)
17. Rana, J., Kristiansson, J., Hallberg, J., Synnes, K.: An architecture for mobile social networking applications. In: First International Conference on Computational Intelligence, Communication Systems and Networks, CICSYN 2009, pp. 241–246 (July 2009)
18. Seong, S.W., Seo, J., Nasielski, M., Sengupta, D., Hangal, S., Teh, S.K., Chu, R., Dodson, B., Lam, M.S.: Prpl: a decentralized social networking infrastructure. In: MCS 2010: Proceedings of the 1st ACM Workshop on Mobile Cloud Computing and Services, pp. 1–8. ACM, New York (2010)

19. Toninelli, A., Montanari, R., Lassila, O., Khushraj, D.: What's on users' minds? toward a usable smart phone security model. IEEE Pervasive Computing 8(2), 32–39 (2009)
20. Toninelli, A., Pathak, A., Seyedi, A., Cardoso, R.S., Issarny, V.: Middleware support for mse management. In: Proceedings of the 2nd IEEE International Workshop on Middleware Engineering, to be held with COMPSAC 2010 (2010)
21. Tran, M., Han, J., Colman, A.: Social context: Supporting interaction awareness in ubiquitous environments. In: 6th Annual International on Mobile and Ubiquitous Systems: Networking & Services, MobiQuitous 2009, pp. 1–10 (July 2009)
22. Yarta, `https://www.gforge.inria.fr/projects/yarta/` (last visited: May 2010)