Semantic Annotation of Mobile Data for Language Access

Raimondas Lencevicius
Nokia Research Center Cambridge
3 Cambridge Center
Cambridge, MA 02142
Raimondas.Lencevicius@nokia.com

Alexander Ran
Nokia Research Center Cambridge
3 Cambridge Center
Cambridge, MA 02142
Alexander.Ran@nokia.com

ABSTRACT
Mobile devices both host and collect significant amount of data that could be interesting to users. To make this data easily accessible, it has to be stored in semantic repositories using a well-defined ontology. Relationships between data from various sources should be explicit. Natural language interface to such data is an attractive option for information access. However, there are semantic gaps between the data repositories and the formal representation of meaning produced by language understanding systems. This paper describes a solution to the issues above. We have implemented a system that converts the mobile data into RDF format and annotates it with information necessary for efficient access via natural language. We have designed and implemented Natural Query system that automates the interface of natural language system and the semantic data repository. Language tags are used to map between the natural language meaning representation and the repository elements. Repository graph search is used to discover the knowledge about the repository structure.

Categories and Subject Descriptors
H.3.3 [**Information Search and Retrieval**], H.5.2 [**User Interfaces**]: *Natural language,* I.2.4 [**Knowledge Representation Formalisms and Methods**]: *Semantic networks*

Keywords
Semantic annotation, query language, natural language.

1. INTRODUCTION
Natural language based interaction with software is increasingly viewed as a promising addition and sometimes even alternative to graphical user interfaces (GUIs), especially in the domain of mobile devices. Mobile devices host structured and semi-structured information bases, software services, and integrated devices such as cameras, music players, etc. Mobile devices also make a perfect user interface to the real-world environment. They are constantly carried with the user [2] enabling gathering of user location information. Mobile devices are equipped with more and more sensors including GPS receivers, Bluetooth transmitters and receivers, RFID receivers and others. They also receive and store information about such events as messages, phone calls, meetings, application usage and access to digital services. It is therefore natural to expect that this data should be collected and made accessible on mobile device.

However, there are some open questions that need to be resolved in order to make this data useful and accessible both to the programs and to the mobile device users. Collected real-world data must be structured and integrated with other information available on mobile devices such as the information found in the user's phone book or calendar. There also needs to be an intuitive interface that allows flexible access to collected information.

Mobile devices store a rich set of structured information. The address book or phone book application contains names, phone numbers, addresses and affiliations of personal contacts. The calendar application contains entries for meetings with participants, meeting location and time. We exchange messages and calls with people and organizations listed as our contacts. All these data are related. Retrieving these data based on their relation could be very useful for device owners. With such retrieval capabilities they could learn who called them when they were in California, or when is their next meeting with Ann from Accenture. Unfortunately, the relations between different data items are not always recorded explicitly when the events occur or information is entered in some application. Therefore it is important to integrate the collected data by explicating its relation to the data available on the device. To achieve this goal, we have developed an extended PIM ontology that covers all relevant types of information available on the mobile device: from observed events, information from external data stores, to on-device data from several mobile applications. Once the data was structured and augmented with relations, it is stored in RDF [15] repository.

So far mobile applications have been designed with their own user interfaces, mostly GUIs, and occasional dedicated hardware controls. Most of the application software on the mobile device could benefit from a natural language interface to its functionality that would simplify and streamline performing various tasks.

As a rule, language systems and mobile applications software are developed independently of each other. To recoup the investment in the development of a language system it must be capable to integrate with a broad range of information sources. Unfortunately information bases are not designed for interaction using natural language. As a result this integration process is mostly ad hoc, manual process. This severely limits the impact

that maturing language processing technology can have on transforming the way we interact with the mobile devices.

In our research, we have investigated ways of created robust and portable natural language interfaces to semantic repositories. We created a novel Natural Query (NQ) language and data access engine that greatly reduces the costs of providing natural language interfaces to semantic repositories. NQ can heuristically attach operational semantic interpretation to a database independent meaning representation of a natural language question over a given semantic repository. NQ enables us to provide a natural language interface to the integrated real-world and on-device data. NQ requires attaching basic linguistic information to structural elements of semantic repository. In this paper we give a brief overview of such annotations for ontology in the extended PIM domain.

The paper describes the mobile data conversion into RDF and semantic annotation (Section 2). Additional annotation and knowledge extraction is needed for automated natural language interface to the data repository (Section 3). Our experience with the system is presented in Section 4. We finish with the description of related work and conclusions.

2. MOBILE DATA INTEGRATION INTO SEMANTIC REPOSITORY

We had to deal with two major data sources: events gathered by data collection framework and PIM data available from PIM applications. This section describes data from both sources, necessary data conversion and integration into semantic repository.

2.1 Mobile Device Data

Data on mobile devices is owned by different applications. This makes it hard to establish and explicitly indicate semantic relationship between different data items. This situation is acceptable as long as the users can only interact with their data using the limited set of functionality provided by the applications. However, if we open these data for language based access, it becomes necessary to support access to different data items using their semantic relationships. Some examples are referring to people by their affiliations, titles, city of residence or office location; referring to meeting by their participants, subject, or location; referring to received calls by the name of caller's organization.

In our project we dealt with data that originated from the phone book application (sometimes also called address book) and the calendar application. Data in these applications are stored in separate Symbian data bases [6]. Since these databases cannot be changed without interfering with the functionality of standard applications we chose to integrate all data in a separate semantic repository. We designed an extended Personal Information Management (PIM) ontology that adequately represented all data items that we were interested in and their relationships. We implemented a set of Python scripts that extract the data from native databases and import them into the PIM ontology. We used RDF repository for data storage.

We created the PIM ontology to cover all data available in the device. We considered using such standard ontologies as W3C foaf [8] and vcard [20]. However, the information available on the mobile device was richer than the types supported by standard ontologies; therefore we decided to create our own ontology. Vcard also uses string values for certain objects that we wanted to represent as full fledged RDF objects with URIs and attributes so they would have identities and we could add information about them. For example, city and country fields are represented as strings in vcard. However in order to represent even basic geographic relationships cities and countries must be represented as objects.

We also considered mixing and matching types from several ontologies for our data. This approach has the advantage of using types possibly known by other systems. However this approach, leads to a rather incoherent architecture of the ontology. We decided that creating a single internally consistent ontology was preferable in our case. If needed, classes and properties in our ontology can be related to types in vcard and foaf via equivalence declarations using RDFS and OWL [21].

Main class for contacts in our ontology is the *Contact* class. It contains *address*, *email*, *group*, *phoneNumber* and *URL* attributes. Organizations and persons can be *Contacts*, so we have *Organization* and *Person* classes inheriting from the *Contact* class. In addition to inherited attributes, *Organization* class also has *name* and *representative* attributes. *Person* class adds *affiliation*, *birthday*, *btDevice*, *familyName*, *givenName*, and *nickname* attributes. *Affiliation* class showing the affiliation of a person with some organization has *organization* and *title* attributes. Part of the ontology relating these classes is shown in Figure 1.

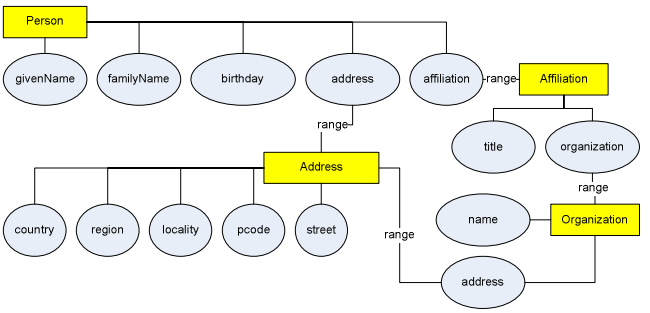

Figure 1. Part of Mobile PIM ontology

Group class describes groups of contacts, such as office colleagues or baseball friends. It has *contacts* attribute that contains contacts belonging to the group and *name* attribute.

Location is a generic class describing locations that has a number of subclasses: *Address*, *Country*, *GPSLocation*, *GSMLocation*, *Locality*, *Pcode* and *Region*. *Address* represents detailed addresses and contains *country*, *locality*, *pcode*, *pobox*, *region*, and *street* attributes. *Country*, *Locality*, *Pcode* and *Region* classes are simple with just a *name* attribute for respective objects. *GSMLocation* class describes locations as obtained from GSM network. It has *carrier*, *cellTower* and *lac* attributes. *Carrier* is the cellular network operator, *cellTower* has a single cell tower ID, and *lac* is a Location Area Code describing a certain region within the network. *GPSLocation* specifies locations using *latitude* and *longitude* attributes.

Mobile device Calendar application contains information about meetings. *Meeting* class has *subject*, *location*, *participants*, *start* and *end* attributes.

Message class objects represent messages. They indicate *messageSubject*, *messageBody*, *receiver* and *sender*.

One of the goals of semantic web is developing standard universal ontologies. Unfortunately, neither the existing ontologies, nor the one we used in our project can be claimed to be standard. Attributes and data in different applications and domains vary significantly. For example, some calendar applications may specify participants, while others don't. Some address book applications may allow specifying birthdays for contacts, but others do not. Ontologies seem to follow in their structure the applications or uses that their creators considered at the ontology creation time. Classes are created based on particular use cases. Attributes are chosen based on data availability and planned use of that data. Rather than focusing on the standardization, we discovered that an important value of RDF ontology is its extensibility – ability to accommodate new types and attributes at any time.

2.2 Event Data

For data collection on mobile devices, we have used one of the frameworks available within Nokia to collect events that occur on a mobile device: phone calls, SMS messages, nearby Bluetooth devices, and GSM locations. All of these events are tagged with a timestamp when they occur. For phone calls the device records the phone number called (or the phone number that called the user) and call duration. For messages, the phone number and the message text is recorded. A GSM location change event is recorded when the cell tower associated with the phone changes. Finally, the phone periodically scans for Bluetooth devices in its vicinity and records their names and IDs. All observations are stored in the objects of *Observation* subclasses: *BTDeviceObserved, CallObserved, MessageObserved,* and *LocationObserved*.

Although the gathered data is interesting by itself, it becomes even more useful when properly linked to the data already available in the device. For example, user may want to know where the person who called them lives. This information could be found by relating the call log to the phone book on the device that maintains the association of phone numbers to people and their addresses. To enable this connection, it is important to collect and preserve semantically relevant information. The connection of gathered information to other data can be achieved through time and location relationships, phone numbers, email addresses, Bluetooth IDs and other inverse functional properties. Time and location can be used to relate data items that are either associated with same time period or the same location. All event data is time stamped, which makes such associations relatively simple. Location can be related to time stamped data items through location observed during the same period of time. Unfortunately for establishing some other relationships however there might be no generic approach. For example in order to connect phone call and message data to other data associated with the phone number, the phone number has to be known in a standard form URI. We used the standard international form of the phone number with country code and long distance code, for example *+1 555 555-5555*. However, data processing may be needed to infer and attach these codes to some phone numbers that enter the system without such codes. For example, the phone number supplied via caller ID does not always include the country code. Custom code has to be written for many data items to convert them on entry into the form required by the semantic repository.

The attributes of *Observation* objects connect with other objects of the repository. For example, the *phoneNumber* attribute of a *CallObserved* is of type *PhoneNumber*, which is also used in the attribute *phoneNumber* of a *Person* or *Organization* class. Therefore the gathered data semantically integrates with the on-device data. Common classes are basis for building relations between data classes belonging to different applications.

Another area where observed data integrates with on-device data is the location information. GSM locations gathered on the phone can be related to geographical locations, such as cities, states or countries. Some data processing and additional relations in the RDF repository are needed for this. We use the *partOf* relation between different objects to represent geographic or organizational inclusions. For example, a relation can indicate that Boston is a part of Massachusetts, which in turn is a part of the USA. This attribute is also used to describe the GSM location containment within a certain geographical object. Since GSM locations are somewhat imprecise, we have chosen to associate them with town or city level geographical entities. This provides sufficient information in most cases. If a more precise location can be determined, it could be associated with a city neighborhood, street, house or even part of the office building.

For some other data, programs or users have to add information to facilitate integration. Bluetooth device IDs need to be associated with specific persons, since such association is not usually available in the mobile device phone book. For this reason we added *btDevice* attribute to the *Person* class. It has to be filled in with concrete values in order to associate the *BTDeviceObserved* observation to a specific person carrying a Bluetooth device.

2.3 Discussion

In a number of cases we had to decide whether to represent particular entities as strings or as objects using URIs. It seems that constructing an object is almost always worthwhile, since such objects can be later used for inter-object relations. For example, by having Country, Region and City objects, we are able to indicate *partOf* relations between them. Also a single URI for a particular object, for example, city, allows to detect such connections as people living or working in a single city.

Overall, we found that our RDF repository is significantly more flexible than a relational database. It naturally supports multiple classes of contacts, multiple affiliations per person, and supports a sophisticated typing system.

3. NATURAL LANGUAGE INTERFACE

Although the repository of integrated real-world and in-device data can be used in a variety of ways, for example, via querying it using SPARQL [19], we were interested to provide an intuitive and flexible user interface to it. A general natural language interface to a rich data set could be more effective than a GUI based application.

As a rule, information bases and language systems are developed independently of each other. Therefore information bases are not designed for interaction using natural language and their integration process is mostly ad hoc, manual process. Figure 2 is a sketch of a typical architecture that is used to provide a natural language interface to databases and other back-end or native services.

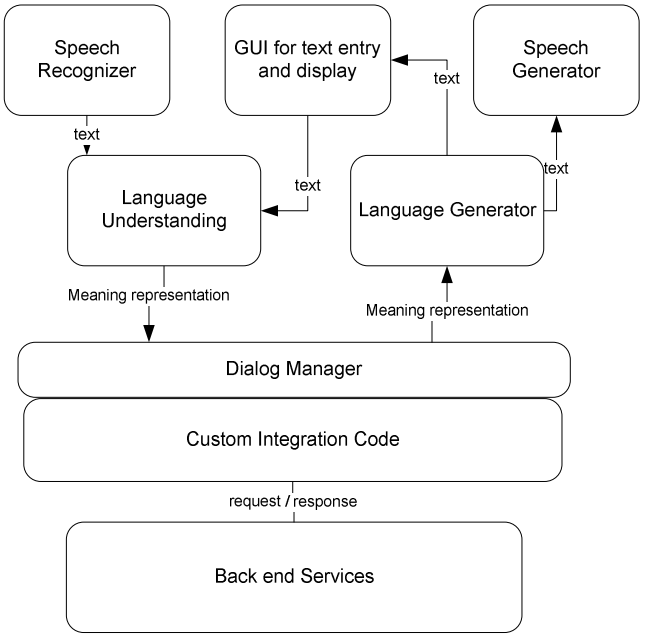

Figure 2. Architecture sketch of Natural Language Interface to Services

The speech recognition and generation components translate between text and speech modalities. The language understanding component converts the text into a formal representation of meaning sometimes called *semantic frame* [17]. The language generation component converts the formal meaning representation to a natural language text [1]. The dialog manager uses the context of conversation to complete frames received from the language understanding module or created by the custom integration code from responses of backend services. The custom integration code also translates meaning representation frames it receives from the dialog manager into a standard database query or backend specific API requests.

Let us assume the user asks the system about contacts in some organization and geographical location:

Who do I know at IBM Ulm?
Who are my contacts at IBM in Ulm?
What are the names of my contacts at IBM in Ulm?[1]

The operational semantics of these questions can be adequately represented with a database query. Let us consider how this request would need to be posed to an RDF repository. SPARQL [19] query corresponding to our example question over the ontology shown on Figure 1 looks as follows:

SELECT DISTINCT $person ?givenName ?familyName
FROM <http://localhost/pim.rdf>
WHERE { $person a pim:Person; pim:givenName ?givenName; pim:familyName ?familyName; pim:affiliation ?affiliation; pim:address ?person_address.
$affiliation pim:organization $organization.
$organization pim:address ?organization_address; pim:name "IBM".
{?person_address pim:locality "Ulm"} UNION
{?organization_address pim:locality "Ulm"}}

Unfortunately in order for a language system to generate such semantic representation from the original questions, the language system must contain a large amount of information about the structure of the database and its content. Such information includes the facts that IBM is a name of an organization and Ulm is a name of a city, cities can be related to organization through their addresses, organizations are related to people through their affiliations, people are related to cities through their home and office addresses, and all these relationships and objects are represented by the specific structures and entities of the database.

Entering such information into a language system is a tedious and costly process that is not only domain dependent but also is sensitive to specific choices of database organization. There is an obvious advantage in maintaining some independence between the database and the language system. One way to achieve this independence is to have the language system generate semantic representations of the questions that are as independent of the database organization as possible.

In the example above semantic information contained in the question and independent of database organization amounts to the following meaning representation:

contact.name: ?
organization: IBM
city: Ulm

It is possible to have the language system produce such database independent meaning representation of questions. But is the information in such meaning representation of the question sufficient to perform the requested operation? Obviously there are several information gaps between this database independent meaning representation and the database specific semantic representation of the question in the form of a formal query.

The first gap is due to different names used to refer to the same elements in the language system and the repository. For example, the category called "*city*" in the language system corresponds to the attribute *locality* of the *Address* class. Therefore there is a need to maintain the mapping between the two naming systems.

The second kind of gap between the two systems is that one element in the language system may correspond to multiple elements in the repository and vice versa. In our example the reference to the address can map to home address, work address, or the organization address of the contact. This is partly due to the ambiguity of the natural language, which is not the main focus of our discussion in this paper. There are also situations where the granularity of categorization is different between natural language and repository representations. This happens when several

[1] The name of the organization and the city were selected for shortness and carry no other information

different concepts exist in the repository for objects which are viewed as instances of the same concept in natural language. In our example this gap required the UNION in the query to represent the original natural language request.

Third and the most important source of the information gap between the meaning representation of the natural language request and the SPARQL query is due to the fact that the query must specify the navigation to the information in the repository using the repository structure. This information about the repository organization is entirely absent from the natural language question and cannot appear in a database independent semantic representation.

We have designed and implemented the Natural Query (NQ) language and engine [14] that bridges the gaps identified above thus opening a way for portable (database independent) natural language interfaces to semantic repositories. NQ can automatically map meaning representation produced by language systems into precise queries. NQ employs two mechanisms: language tags and data graph search to return requested data using only the information in the database-independent meaning representation of the user request.

3.1 Language Tags

Language tags are words, expressions, and linguistic tokens attached to database elements such as classes and properties. Multiple tags can be attached to a single element and a single tag can be attached to multiple elements. Language tags are the names of the corresponding categories used by the language system(s). When a language system produces a form like the one in our example,

contact.name: ?
organization: IBM
city: Ulm

under the NQ system its interpretation is:

find the attributes tagged as "name" of an instance of the class tagged as "contact" related through properties tagged as "organization" and "city" to values "IBM" and "Ulm" respectively

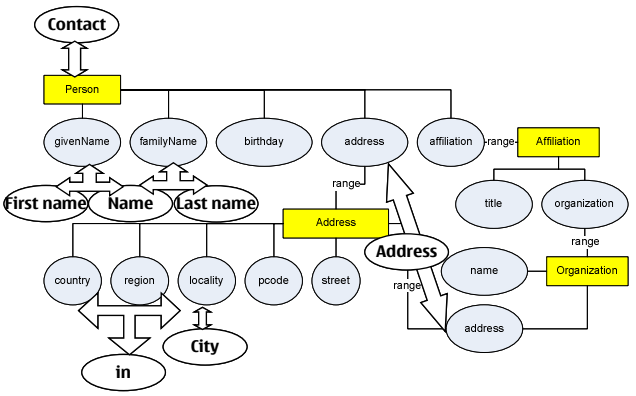

Figure 3. Language tags for database elements

Language tags provide an opportunity for a semantic annotation additional to the class names and their properties. In a natural language system accessing an RDF repository data, we have three layers of semantic information: database-independent meaning representation, the data and ontology, and the language tags. It could be argued that if there were correspondence between the categories of database-independent meaning representation and the data and ontology, the language tags would not be needed. Unfortunately, if the ontology and language system are to be developed independently, there is no way to maintain or ensure such match. Thus language tags provide the many-to-many mapping between the two independent systems of categorizations and eliminate the first and second kind of information gaps between the meaning representation and semantic repositories.

Figure 3 illustrates language tags associated with a part of our PIM ontology. A generalization like "*Contact*" can be attached to specific classes like "*Person*" and "*Organization*". A general reference like "*Name*" can be attached to multiple elements like "*givenName*", "*familyName*", and so on. In our RDF repository of real-world and in-device data, we added language tags to the RDF objects using a subproperty of RDFS *label* field.

3.2 Graph search

The third gap that exists between the database independent meaning representation of the natural language request and the formal query that actuates it over a given database is the information about the organization of the data repository. In order to navigate from the given attributes of an object to the target of the query, SPARQL queries need to know the specific path that connects them on the database graph. In current language systems, this path is encoded by the query and stored in the custom integration code for every different type of query. Thus a query defines a subgraph with given properties some of which are specified in the database-independent meaning representation of the natural language request and some are encoded in the custom integration code component.

While a formal query defines a connected subgraph as illustrated on Figure 4, the database-independent meaning representation only identifies some nodes and edges of this subgraph. Identified fragment might be disconnected. In the example above it identifies "*Person*" and "*Organization*" classes as well as "*Ulm*" value of "*locality*" property (by reference to its language tag "*city*") and "*IBM*" as a value of "*name*" property of an instance of "*Organization*" class. This leads to an important idea: that the knowledge embedded in the formal queries that *know* the database organization, can be also extracted from the natural language meaning representation and the data repository itself.

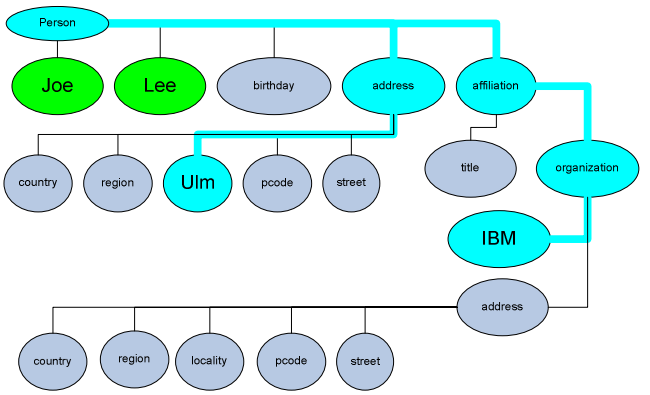

Figure 4. Answering query via graph search

In Figure 4 it is possible to notice that for a given set of elements identified by a meaning representation of natural language request it is possible to identify the query subgraph by searching the database. In other words, a program could find paths connecting the nodes known from the meaning representation, such as *"Person", "name", "Organization", "City", "Ulm",* and *"IBM"*. One of such paths is highlighted in the picture.

Therefore while traditional approaches to semantic analysis of natural language questions over databases rely on hand crafted code or data for representing the information about the organization of the database, NQ extracts such knowledge from the data repository by using graph search. Given a question "*Who are my contacts at IBM in Ulm?*", NQ finds paths connecting the nodes known from the database independent meaning representation, such as *"Person", "name", "Organization", "City", "Ulm",* and *"IBM"*.

3.3 NQE Discussion

NQE may find multiple subgraphs that connect all given elements. In such cases we apply heuristic ranking of these subgraphs in order to determine the most relevant ones. So far we experimented with several ranking mechanisms all of which are variations on path length (weight) between the elements specified by the meaning representation. In all our experiments the results retrieved by the system in response to natural language questions correspond well with intuition of human subjects.

The results returned by NQE are designed to support the needs of conversational interfaces. If no results are found that match the elements specified in the meaning representation, NQE returns best matches that include only a subset of elements in the query. For example, if no contacts at IBM in Ulm can be found, contacts at IBM in other cities would be returned as well as contact from Ulm that are not affiliated with IBM

NQE can perform basic reasoning over type hierarchy. A "*Person*" is substitutable for a "*Contact*", a "*MobilePhoneNumber*" for a "*PhoneNumber*", but the opposite is not true. NQE supports organizational and geographic inclusion and can perform corresponding reasoning. When a calendar application lists meetings in Helsinki and Oulu, NQE can answer questions regarding meetings in Finland, where these cities are located. Similarly information about organizational structure can be used to answer questions about Nokia while the database only records Nokia's internal organizations like Multimedia or Enterprise Solutions. Finally NQE creates structures that can be used to produce explanations regarding how the answers relate to the questions.

We have created a proof of concept implementation of NQ in Python [12] that runs on S60 [16] mobile phones. Full description of the Natural Query system implementation is outside the scope of this paper.

4. EXPERIENCE WITH THE SYSTEM

We tested our system on a PIM test data set containing 550 contacts with about 150 meetings and 250 phone calls, which is normal for executives with many active contacts and frequent meetings. The repository contained over 11000 RDF triples. We asked over 50 natural queries corresponding to over 600 parameterized questions.

The system can answer questions ranging from "*What is the email of John?*" to "*Where does Ann work?*" to "*My meetings next week in Cambridge with John from MIT*" and "*Who called me yesterday during the meeting with Ann?*". Some of these questions would convert to quite complex relational or SPARQL queries. For example for the query "*Who called me yesterday*", we need to find all telephone numbers of calls that occurred yesterday and then find all people who have these telephone numbers. NQ query for this is very simple: "*:select 'Person' :where ("Received Call", Time ('yesterday'))*".

If we classified questions according to domains, one domain would contain questions about the personal information data from an address book application, for example *"Who works as a real estate broker?*". Another set of questions is about meetings, for example, "*When are my meetings next month at MIT?*". Yet another set is about calls and messages, for example, "*Who called me last Friday?*". Finally there are questions spanning multiple domains, for example, "*What are emails of people who participated in a meeting on Monday?", "Who called me when I was in Finland*?", and so on. All these types of queries were successfully created and executed on the extended PIM data store.

We found out that we could easily ask questions both about the in-device data and the collected real-world data. Semantic integration of multiple data sources enhanced our question answering capability significantly, allowing such questions as "*Who called me when I was in Helsinki?*", "*Which messages did I receive during the meeting with Juha?*", etc. Although an out-of-pattern detection of someone's Bluetooth device is a weak indication the phone user met the owner of the Bluetooth device, in our experiments we assumed such implication. This allowed us to ask questions such as "*Who did I meet last week?*" or *"At what time did I meet Ann last Saturday?"*

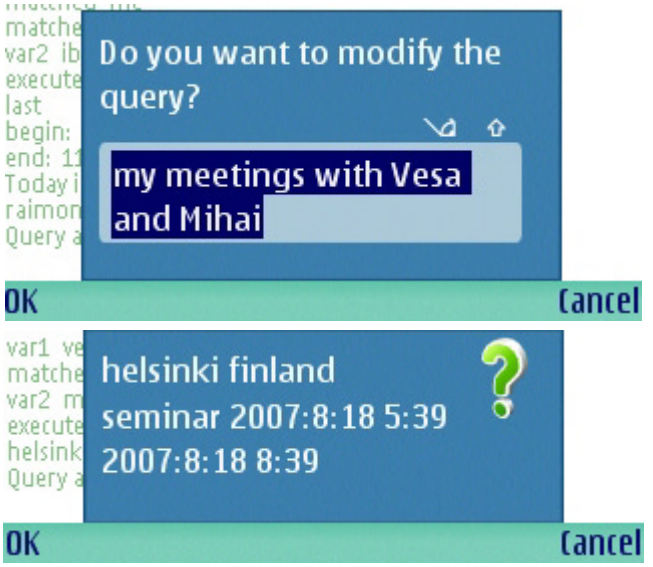

Figure 5. Example question and answer

Test NQ queries mostly returned expected answers (96% recall, 92% precision) (Figure 5) including the approximate answers where the exact answers were not available. For example, the question "*When was my meetings with Sam last month?*" had no exact answers, so the system returned approximate answers of

meetings with Sam that did not occur last month as well as the meetings that occurred last month, but did not include Sam.

The performance of the system was acceptable with answers taking from less than a second to several seconds. The system implementation is a prototype written in Python that was not optimized for memory or speed. The detailed evaluation of system performance is outside the scope of this paper. We are planning to optimize the system performance in the near future.

5. RELATED AND FUTURE WORK

Mobile data storage in RDF repositories is investigated by ConnectingMe [9] project at Nokia Research Center. We have collaborated with ConnectingMe in the ontology and repository development. Some tools for data extraction and conversion are shared between our two projects.

Semantic markup and annotation of web [7][4] and media [3] data is a topic of active research. Our research is related to the mobile media data annotation. There has been a lot of research on ontology creation tools. We used one of such tool—Protégé [11] to design our extended PIM ontology.

Event data has been gathered on mobile devices by a number of projects including Context [13] and Reality Mining [5]. In our work, we have extended one of the data gathering frameworks available at Nokia.

We have not discovered any research directly corresponding to the Natural Query approach. The Precise system by Popescu et al. [10] attaches language tokens to database elements in a way very similar to language tags of NQ. Also the query derivation approach of Precise is based on database graph search. NQ uses a more flexible data model, supports incomplete answers, and collects data for explanations.

In the future, we plan to connect our system to such natural language and speech systems as TINA [17] and Galaxy [18]. We plan to perform user trials to evaluate our system and its user interface to real world data. We will collect additional data such as email messages, songs listened, and pictures viewed and taken. We will also optimize the current prototype implementation.

6. CONCLUSIONS

Mobile devices are now able to continuously collect various events interesting to the user. Mobile devices also host structured and semi-structured information bases. We have demonstrated the integration of all this data using a flexible and powerful RDF repository and a common ontology. We have designed and implemented a query language and engine NQ that can automatically map meaning representation produced by language systems into formal queries on RDF repositories. We have used language tags for mapping of the meaning representation to the data classes. NQ uses graph search to extract the information about the repository's structure. Our experience shows that semantic data annotation and knowledge extraction significantly improves the capability of natural languages interfaces to mobile data.

7. REFERENCES

[1] Baptist L. and S. Seneff, "Genesis-II: A Versatile System for Language Generation in Conversational System Applications," *Proc. ICSLP '00*, Vol. III, pp. 271-274, Beijing, China, Oct. 2000.

[2] Chipchase, J., "Why do People Carry Mobile Phones?", http://www.janchipchase.com/blog/archives/2005/11/mobile_essentia.html, 2005.

[3] Davis, M., King, S., Good, N., Sarvas, R., "From Context to Content: Leveraging Context to Infer Media Metadata" *Proceedings of the 12th annual ACM international Conference on Multimedia*, New York, NY, USA, pp: 188 – 195, 2004.

[4] Dill, S, et al., "SemTag and Seeker: Bootstrapping the Semantic Web via Automated Semantic Annotation", *Proceedings of the 12th international conference on World Wide Web*, Budapest, Hungary, pp: 178 – 186, 2003.

[5] N. Eagle, "Machine Perception and Learning of Complex Social Systems", *Ph.D. Thesis*, Program in Media Arts and Sciences, Massachusetts Institute of Technology, June 2005.

[6] Edwards, L., Barker, R., et al. *"Developing Series 60 Applications"*, Addison Wesley 2004.

[7] M. Erdmann, A. Maedche, H.P. Schnurr, S. Staab, "From manual to semi-automatic semantic annotation: About ontology-based text annotation tools", *Proceedings of the Workshop on Semantic Annotation and Intelligent Content*, 2000.

[8] FOAF Vocabulary Specification 0.9, http://xmlns.com/foaf/0.1/, 2007.

[9] Lassila, O. et al, "ConnectingMe", http://research.nokia.com/research/projects/connectingme/index.html, 2007.

[10] Popescu, A., Etzioni, O., and Kautz, H. 2003. Towards a theory of natural language interfaces to databases. *Proceedings of the 8th international Conference on intelligent User interfaces* (Miami, Florida, USA, January 12 - 15, 2003). IUI '03. ACM Press, New York, NY, 149-157.

[11] Protégé Ontology Editor and Knowledge Acquisition System, http://protege.stanford.edu/, 2007

[12] Python for S60, http://sourceforge.net/projects/pys60, 2007

[13] Mika Raento, "Context software - A prototype platform for contextual mobile applications". *Proceedings of the International Proactive Computing Workshop*. University of Helsinki, 2004.

[14] Ran, A., and Lencevicius, R., "Natural Language Query System for RDF Repositories", To appear in *Proceedings of the Seventh International Symposium on Natural Language Processing*, SNLP 2007, 2007.

[15] Resource Description Framework, http://www.w3.org/RDF/, 2007.

[16] S60 platform, http://www.s60.com, 2007

[17] S. Seneff, "TINA: A natural language system for spoken language applications," *Computational Linguistics*, vol. 18, no. 1, pp. 61-86, March 1992.

[18] S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue, "GALAXY-II: A Reference Architecture for Conversational

System Development," *Proc. ICSLP 98*, Sydney, Australia, November 1998.

[19] SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/, 2007.

[20] Vcard, http://www.w3.org/TR/vcard-rdf, 2007.

[21] Web Ontology Language, http://www.w3.org/TR/owl-features/, 2007.