A mobile data management architecture for interoperability of resource and context data

Andreas Brodt*, Oliver Schiller*, Sailesh Sathish†, Bernhard Mitschang*
* *Universität Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany {brodt|schiller|mitschang} @ ipvs.uni-stuttgart.de*
† *Nokia Research Center, Visiokatu 1, 33720 Tampere, Finland, sailesh.sathish @ nokia.com*

***Abstract*—Mobile devices have become general-purpose computers that are equipped with sensors, constantly access the internet, and almost always accompany the user. Consequently, devices manage many different kinds of data about the user's life and context. There is considerable overlap in this data, as different applications handle similar data domains. Applications often keep this data in separated data silos. Web applications, which manage large amounts of personal data, hardly share this data with other applications at all. This lack of interoperability creates redundancy and impacts usability of mobile devices. We present a data management architecture for mobile devices to support interoperability between applications, devices and web applications at the data management level. We propose a central on-device repository for applications to share resource and context data in an integrated, extensible data model which uses semantic web technologies and supports location data. A web browser interface shares data with web applications, as controlled by a general security model.**

I. Introduction

Mobile devices carry lots of different kinds of data, including multimedia files, PIM (personal information management) data, device profile data, location and map data, to name a few. Also, they possess several sensors that allow them to collect data about their environment, i.e. GPS receivers, cameras or accelerometers. Moreover, devices can easily access additional data from the web, or even from surrounding devices via a wireless ad-hoc network [1]. Different applications access and utilize all this data in different ways and for different purposes. While one application simply manages certain resources, another application may use the same resources as context information to adapt its behavior to the current situation. This complies with Dey's definition for context [2] as *"any information that can be used to characterize the situation of an entity"*. An example is a calendar that simply manages meetings, while the telephony application reads the same information to adapt the ringtone.

There is considerable overlap in the data domains of different applications on a mobile device. As a large amount of the data relates to several aspects of the user's life, the same data objects reoccur in different use cases. As an example, Harry, a friend of the user, has an address book entry in the user's mobile device. He tried to call three times today. Harry also sent 12 text messages and 34 e-mails, and he participates in tomorrow's bowling evening, which takes place near Harry's house. According to image annotations, Harry is depicted on 23 photos, and his phone is currently visible in the device neighborhood. Many different applications operate on the same data domains, as every application deals with a certain aspect of the user's context. However, applications on a mobile device typically keep their data in isolated data silos. This causes redundant data management, software gaps, and ultimately a bad user experience, as the potential of the data is not exploited. What is needed is *interoperability* between applications at the data management level.

The fact that mobile devices accompany the user nearly all the time adds a special aspect to the general interoperability requirement: interoperability of spatial data. In contrast to most other data management scenarios, on a mobile device a large share of the managed data possesses spatial relevance. Even entities without direct spatial references, e.g. a phone call or a text document, may be correlated with the user's position and occur in a spatial query, such as *"where was I when I missed this phone call?"* More powerful reasoning about the user's location context is possible through spatial interoperability at the data management level, as this creates a general join criterion for most resources. And as everything that carries a spatial reference can be drawn on a map using the POI (Point of Interest) metaphor, spatial interoperability provides the user an additional entry point to access the data.

Interoperability at the data management level is required not only internally on one mobile device, but also across devices. Mobile devices possess the ability to exchange data spontaneously via wireless ad-hoc networking [1]. This way, data may not only overlap between different applications, but also between different devices. As an example, a meeting may be scheduled in each participant's device, and yet another device might know the geographic location of meeting room 3.436. Connecting co-located devices in a spontaneous fashion results in an even better view on the user's current context, as more data is available. Applications which exploit data from co-located devices typically must implement the communication logic themselves in application code and deal with low-level protocol details. A platform offering data interoperability between devices would hide those details and offer higher-level abstractions for applications to exploit resource and context data from co-located devices easily.

To an increasing degree, web applications are used on mobile devices, as devices are becoming more powerful and mobile web browsers are reaching desktop class. Web applications require special attention in mobile data management scenarios, as they handle large amounts of user data. Many users describe their activities, keep track of friends, and publish digital media via web applications. This content is a rich source of context data. Very limited interoperability is offered through proprietary service APIs and some mobile software platforms provide wrappers to integrate data from selected services. On the other hand, context data that is gathered locally, most notably the user's current GPS position, cannot be provided by servers in the cloud, but must be retrieved on the client side. The strict separation between web applications and local applications makes interoperability very difficult and prevents many potential benefits. E.g., if the user books a flight online, the booking web site could guess the departure location from the user's current position and her past flights, and the destination from the calendar. Also, the web site could make the flight details appear in the user's calendar automatically—without the user needing copy the flight data into the calendar manually.

In this paper we address interoperability at the data management level of mobile devices. After introducing the state of the art (Section II), we present a platform architecture for interoperability between installed applications, co-located devices, and web applications (Section III). Our approach is based on a central data repository on mobile devices which all applications use cooperatively. We discuss the data model and access control and provide further details on the implementation of the data management layer (Section IV).

II. State of the art

A. Domain-specific APIs

Today, most software platforms for mobile devices provide an API for central data domains, e.g. contacts or calendar. These domain-specific APIs support managing data of the same domain in the same data silo across applications. Using the contacts API, e.g., a photo annotation app may offer a list of known persons, so that the user does not need to re-enter them. Or a public transport app may store a train connection in the calendar and create a reminder via the calendar API.

Domain-specific APIs, however, do not solve the general data interoperability problem. They are restricted to their particular domain and designed for common use cases there. Application requirements beyond that, e.g. additional attributes, are not supported. If an application needs to store additional data, it must do so in its own data silo. This prevents other applications from accessing the data and reintroduces redundancy. Furthermore, not all data domains are covered by an API. The public transport application may create a calendar entry about the train connection, but without a map API, the map viewer will not be able to show the geographic location of track 6, from where the train leaves. Finally, as domain-specific APIs are nothing but front-ends for domain-specific data silos, they can only provide interoperability at application level. I.e. if an application needs to interrelate data from different domains, it has to query several APIs and combine the results in application code. Compared to a join in the data management layer, this is inefficient and requires more complicated application code.

B. Semantic web and semantic desktop

The *Semantic Web* [3] community has developed concepts and languages, including RDF, RDF Schema and SPARQL, to model structured data in a way that allows creating relations between resources easily. These techniques are well-suited to augment resources with context data and make them searchable via these annotations. Semantic web technologies are generic and require a domain-specific ontology to specify resources and their relations.

The *Semantic Desktop* aimed at bringing Semantic Web technologies to the desktop and integrating applications through ontologies [4]. The Nepomuk project [5] developed a large software stack and ontologies [6], [7] for the semantic desktop. Nepomuk uses "crawlers" to search a computer and annotate files, e.g. to provide a more useful desktop search. A peer-to-peer architecture facilitates sharing files and their semantic data between users. As the semantic desktop focuses on desktop computers, there is no emphasis on context data, such as location or mobility. Also, as we outlined in [1], [8], a mobile scenario requires wireless ad-hoc networking that is robust towards frequent connects and disconnects rather than structured peer-to-peer networks.

Lehikoinen et. al. [9] prototyped a framework for mobile content management. It managed RDF-based ontologies on top of a relational database on mobile devices. However, it was too inefficient to be used on resource-constrained mobile devices and never made it into production state.

C. Interoperability with web applications

There is currently a trend towards domain-specific APIs for web applications. Basically, they simply replicate the existing domain-specific device APIs inside the web browser as a JavaScript interface. The W3C Geolocation API [10], for instance, specifies an interface for web applications to obtain the position of the user. The W3C Device APIs and Policy Working Group [11] is in the process of defining client-side APIs to enable interaction of web applications with device *services* such as Calendar, Contacts, Camera. As domain-specific on-device APIs, these browser APIs are restricted to the common use cases of their particular domains. As discussed above, interoperability with other domains or further application requirements are impossible.

The discontinued W3C Delivery Context Client Interfaces (DCCI) specification [12] attempts to standardize a generic and domain-independent context provisioning interface for

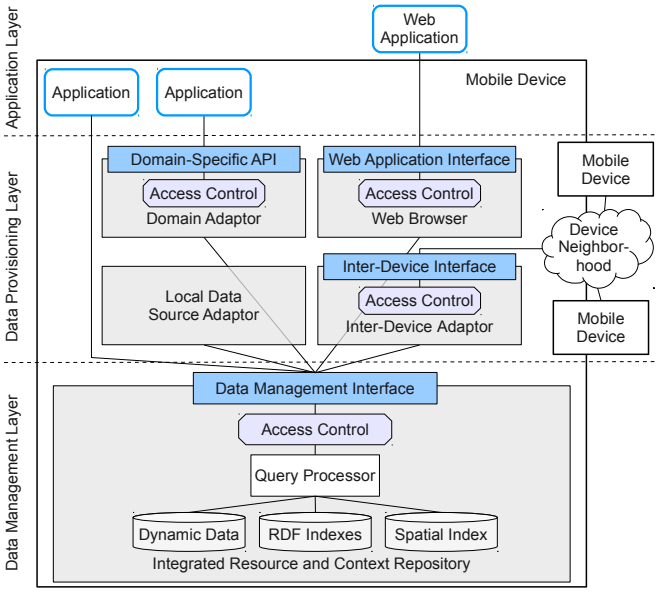

Figure 1. System architecture of our mobile data management platform for interoperability of resource and context data

web applications. DCCI was used in scientific systems [1] but never reached production state.

In addition, several interfaces provide client-side storage inside the web browser. Web applications often store small data items as Cookies in the web browser, e.g. session IDs. Gears [13] extended this principle to a complete relational database inside the web browser. This allows, e.g., a web mail application to store e-mails locally, so they can be read offline and only need to be downloaded once. Popular web applications, including Google Mail and Google Docs, make use of this. With the upcoming HTML5 standard several specification were proposed to let web applications store data on the client: [14] defines an indexed record store, [15] drafts an SQL-based interface, [16] specifies a key-value store. However, these interfaces do not provide interoperability, as they strictly follow the same-origin policy, i.e., only the web application which created the data may access it.

III. Architecture

We developed a platform architecture to enable interoperability between local applications, web applications, and co-located devices at the data management level of a mobile device. It is based on a central repository which manages all resource and context data in a single semantic data model. As this raises security concerns, our platform provides a powerful access control mechanism that is deeply integrated into the system architecture and coupled with the data model.

A. System architecture

As depicted in Figure 1, the system architecture consists of the Data Management Layer, the Data Provisioning Layer, and the Application Layer. The Data Management Layer hosts the Integrated Resource and Context Repository as the storage back-end. The Data Provisioning Layer consists of Domain Adaptors, which provide domain-specific APIs to applications, local data sources (e.g. a GPS receiver or an accelerometer), the Inter-Device Adaptor managing wireless ad-hoc data-exchange with other co-located mobile devices, and, most notably, the web browser. The web browser provides a dedicated data interface for web applications. The Application Layer incorporates the applications, which ultimately consume the resource and context data (and may also contribute). The web browser interface abolishes the data separation between local applications and web-applications. Furthermore, via the Inter-Device Adaptor, applications may access data from other mobile devices transparently.

The Integrated Resource and Context Repository is the crucial component of the platform. The repository achieves interoperability between applications, co-located mobile devices, and web applications by managing their data in one central place. The data is modeled in RDF, which facilitates establishing relations between resources, as well as annotating and augmenting them with additional information. The repository supports spatial data and is able to execute spatial queries efficiently using a spatial index. In addition, it possesses a small main-memory database for dynamic data, e.g. the current GPS position. Such data is frequently updated and thus inefficient to index persistently. The repository still presents the dynamic data as part of a single consistent data model. The repository is accessed through the Data Management Interface. It accepts SPARQL queries, provided the requestor possesses the respective access rights.

Context data from local data sources (sensors, etc.) is brought to the repository by dedicated adaptors. These Local Data Source Adaptors access the specific low-level API of the data source (e.g. the driver of the GPS chipset) and forward the data to the Data Management Interface. Similarly, the Inter-Device Adaptor receives data shared by co-located devices in an ad-hoc fashion and forwards it to the Data Management Interface. It also monitors the device neighborhood, performs access control for incoming requests, and forwards them to the Data Management Interface.

Installed applications can access the Data Management Interface directly to evaluate a SPARQL query. This gives them full flexibility to read and write the entire repository across all application domains, which fulfills our interoperability requirement completely. The repository performs access control checks to enforce the access rights which the application was granted. If an application only needs data of a particular application domain, it can also use a Domain Adaptor. The Domain Adaptors offer higher-level programming abstractions through Domains-Specific APIs which are limited to a particular application domain, e.g. e-mail or calendar. The APIs are less flexible and do not provide interoperability, yet they are easier to use and can, to a certain degree, replace existing API implementations.

Internally, the Domain Adaptors translate their programming abstractions to calls to the Data Management Interface. Yet, as these calls are fairly restricted, domain-specific access control can be performed in the Domain Adaptors, which is much simpler and thus more efficient than access control of the Integrated Resource and Context Repository.

B. Data model

To achieve the desired interoperability, the central requirements for the data model are flexibility, the ability to interlink resources easily, and support for spatial data. Flexibility is required so that all applications can map their specific data model to the general one. Interoperability at the data management level comes through relations between resources of different application domains, which are impossible in isolated data silos. Thus, it must be possible to relate resources created by *application A* with other resources of which application A is unaware. Also, other applications must be able to annotate and augment application A's resources with additional data. The data model must be flexible enough to support this. Finally, on a mobile device it must be possible to annotate everything with location data. The data model must support spatial data types and cater for efficient retrieval of resources by their spatial references.

We use RDF for our data model, as it is very flexible and directly supports relations as first-class objects. RDF models everything as a number of *(subject, predicate, object)*-triples which make a statement about the resource denoted by the subject. In case of an attribute of a resource, e.g. the sender of an e-mail, the predicate denotes the attribute name and the object contains the attribute value, e.g. "john@doe.com". In case of a relation between two resources, the subject and the object denote the resources and the predicate specifies the type of relation that connects them, e.g. to model that "Person 117" is an attendee of "Meeting 1432". All triples together make up one directed labeled graph. It is easy for an application to create additional triples adding further attributes or relations to a resource, which enables interoperability at the data management level.

To express spatial features, we use typed literals of a self-contained complex spatial data type [17]. Thus, we encode the spatial reference of a resource in a single RDF statement, with the object of the triple containing the spatial feature.

A further advantage of RDF is that it also covers the metadata level, i.e. the class-membership of the resources and the properties of these classes, as the defined in an ontology. As the entire data model is a single graph of resources from completely different application domains, there is no structural grouping of resources as opposed to, e.g., a table in the relational data model. Thus, it is important that all resources explicitly model their class membership, so that applications can even distinguish between them. Also, pointing to the metadata level enables a resource to connect to all resources of a class, which is useful, e.g., to model access control at the data management level.

The concrete ontologies which finally express the resources are to a large degree up to the applications to define. They can easily do that by adding the triples to the repository which define the required classes on the metadata level. Yet, especially classes that are frequently used, such as PIM data, should be standardized. The mobile device software platform is likely to include vendor-specific tools, applications, and APIs, such as the domain adaptors shown in Figure 1, which center around a number of core classes. To achieve interoperability between devices of different vendors and device software platforms, standardization efforts are required.

C. Access control

As interoperability is enabled at the data management layer, access control must begin there as well. We use the principle of role-based access control; applications requesting resources are assigned one or more access roles that allow certain operations on the resources. The access control model is tightly coupled to the data model and to a large degree evaluated in the Integrated Resource and Context Repository. The RDF-based data model of our platform explicitly defines all classes and their properties. This can be utilized to grant access rights to an access role on the metadata level. A role may be granted access to a class, specific properties of a class or particular resources. Finally, the owner of a resource, i.e. the application which created it, always has full access rights on the resource. Access roles may be grouped hierarchically, i.e. access roles may be members of higher level access roles. This creates the flexibility to allow very selective and fine-grained access roles and at the same time keep them manageable by a reasonably simple graphical user interface.

To gain access rights, an application asks the data management layer for access roles. This triggers a graphical dialog in which the user may assign the desired access roles to the application. An installed application typically does this once at installation time. A web application may ask for access roles via calls to the Web Application Interface. The granted access roles for a web application may be remembered for a certain time period, but will eventually expire—in the same way that HTTP cookies expire. Also, if the last access role of web application expires, the entire knowledge about the web application, including its owned resources, may be removed from the Integrated Resource and Context Repository. Co-located devices may ask for access roles via the Inter-Device Interface and the decision may be remembered for a certain time.

Naturally, before an app, web application or device may obtain any access roles, it must be authenticated. To authenticate installed applications, most mobile device software platforms require the application to carry a unique application ID and a digital signature [18], [19]. The application ID is used to provide platform security, e.g. to control access to

communication capabilities of the device. The repository can directly use the application ID and assign access roles to it. Web applications are uniquely identified by their URI. However, to prevent man-in-the-middle attacks, web applications must use HTTPS and authenticate themselves using a trusted SSL certificate. If a web application fails to authenticate, the web browser will not accept its request for access roles. Devices connecting via the Inter-Device Interface can be identified by their network device ID.

Generally, the Integrated Resource and Context Repository evaluates the access rights of an application when a query is sent to the Data Management Interface. As the access rights are modeled together with the resource data, the query can be rewritten to restrict the query result to the resource data for which the respective access rights are present. The rewritten query checks existence of an access role which connects the application to the resource. For the most common case of access control based on class-membership, this means fetching the class of every resource and joining the class with the access roles. As most queries likely check the class of resources anyway, we consider this extra effort moderate. By contrast, access control on specific properties of a class requires more complicated rewrites which may require several extra joins with the metadata level. Thus, access control on specific properties should be used with care.

The Domain Adaptors for domain-specific APIs send ordinary queries to the Data Management Interface. Yet, they generate the queries from calls to defined programming abstractions rather than letting the calling application define the query freely. If the implementation of the Domain Adaptor is trusted—we assume it is provided by the software platform vendor—the query can be run as-is, if the application possesses the rights to access the domain-specific API. Thus, the Domain Adaptor checks whether the application is allowed to access resources of the particular domain at all. If this check succeeds, the repository does not need to perform additional access control. Similarly, the Local Data Source Adaptors are likely to be tightly coupled with the platform software and thus may circumvent access control.

IV. The Data Management Layer

The Data Management Layer is the crucial part of our mobile data management platform. It abstracts from storage and query processing details and provides an integrated view on resource and context data originating from applications, co-located devices, local data sources, and web applications. It also performs access control, effectively creating a view on the data which a particular application may access. The Data Management Layer is accessed via the Data Management Interface which passes the queries to the repository.

A. The Data Management Interface

The Data Management Interface provides several interaction methods for applications and the Data Provisioning Layer.

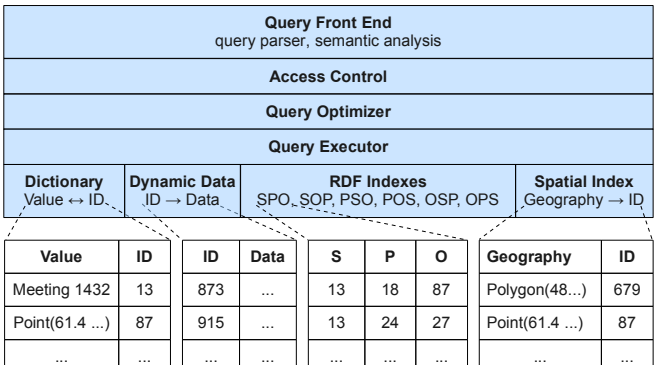

Figure 2. The architecture of the Integr. Resource and Context Repository

The most common one is the query/response interaction method. It accepts a SPARQL query string and executes it on the repository. It returns the query result or, in case of an update, a success or error code. This interaction method satisfies the requirements of most applications. Yet for applications consuming dynamic data, most notably sensor-based context data, query/response interaction is not favorable. Therefore, the Data Management Interface also provides an event-based interaction method for dynamic data. It allows an application to subscribe to a dynamic data resource and notifies the application whenever the resource changes. This way, an application may, e.g., track the current GPS position and is notified on every position update. Finally, it is not optimal if the Local Data Source Adaptors must produce a SPARQL Update string to indicate a change in a sensor reading, for the repository to parse. For this purpose, the Data Management Interface provides a binary interaction mode which allows supplying updates to the repository as binary data blocks. The binary interaction method may also be used for backup, bulk-loading and synchronization purposes. It bypasses the high-level query interfaces by supplying the data in its internal representation. This is more efficient, yet vulnerable to changes in the internals of the repository. Thus, the binary interaction method should only be used by software that is included in the mobile software platform.

B. The Integrated Resource and Context Repository

The Integrated Resource and Context Repository stores and manages the data in the RDF-based data model of our platform. Our preliminary implementation of the Integrated Resource and Context Repository largely utilizes our native RDF triple store with deeply integrated spatial query processing [17], which is based on the open source version of the RDF-3X triple store [20]. The central trick of RDF-3X (and other triple stores) is that it maps all URIs and literals of the RDF data to integer IDs for internal processing instead of dealing with their string representations. This saves memory and enables fast query processing. We adopt this principle and use it throughout the entire repository, i.e. in the RDF indexes, in spatial index and in the dynamic data.

As depicted in Figure 2, the repository consists of a query front end, access control, a query optimizer, the query operators, the dictionary, and data containers. The data containers include the dynamic data, the RDF indexes and the spatial index. The dictionary maps the internal integer IDs to their external representation. Note that it is possible to extend the repository by further data containers, e.g. a full-text index, provided they support ID-based processing.

C. Evaluation

To show the validity of our approach, we measured the query execution times of our repository prototype on a Nokia N900 smart phone (600 MHz processor, 256 MB physical RAM). We executed a typical application query in presence of class-based access control, as discussed in Section III-C. We generated artificial RDF data of 100 RDF classes, 200 access roles and 300 application IDs, resulting in 5.1 million RDF statements and a database file of 120 MB. We queried all instances of different classes (resulting in different amounts of resources) and retrieved different numbers of attributes of each resource. This corresponds, e.g., to the e-mail application loading all e-mails with a number of attributes. All queries were executed in 0.01 to 0.18 seconds, which is suitable for an interactive system. We believe this shows that such a repository works in practice and is indeed able to provide interoperability at the data management layer.

V. Conclusions

We addressed interoperability at the data management level of mobile devices. There is considerable overlap in the data which applications on a mobile device typically manage, as different applications handle similar data domains. However, a lot of this data is kept in separate data silos and is at best accessible via domain-specific APIs. The same holds for web applications, in which users keep large amounts of personal data. This lack of interoperability creates data redundancy and ultimately impacts the user experience of mobile devices. We presented a mobile data management architecture to enable interoperability between installed applications, co-located devices, as well as web applications at the data management level. Our approach is based on a central data repository on mobile devices which all applications use cooperatively. The data is stored and managed in an RDF-based data model and powerful access control that is tightly coupled with the data model regulates access to it. We achieved very good performance of the repository on a mobile device, which shows that our approach is feasible.

In future research we will work on the performance of the repository on larger data sets than the ones used in our evaluation. We expect large performance improvements through specific query operators optimized for the particular access patterns of our platform on flash memory. In addition to that, we will address bulk synchronization with server-based data to achieve better integration with "the cloud".

References

[1] A. Brodt and S. Sathish, "Together we are strong— towards ad-hoc smart spaces," in *PERCOM*, 2009.

[2] A. K. Dey, "Understanding and using context." *Personal and Ubiquitous Computing*, vol. 5, no. 1, 2001.

[3] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, vol. 285, no. 5, 2001.

[4] L. Sauermann, A. Bernardi, and A. Dengel, "Overview and outlook on the semantic desktop," in *Workshop on The Semantic Desktop at the ISWC*, 2005.

[5] A. Bernardi, "FP6027705 NEPOMUK Project Synopsis," DFKI, Tech. Rep., 2008.

[6] L. Sauermann et. al., "Personal information model (pimo)," OSCA, Recommendation, 2009. [Online]. Available: http://www.semanticdesktop.org/ontologies/2007/11/01/pimo/

[7] A. Mylka et. al., "Nepomuk information element ontology," OSCA Foundation, Tech. Rep., 2007. [Online]. Available: http://www.semanticdesktop.org/ontologies/nie/

[8] A. Brodt, A. Wobser, and B. Mitschang, "Resource discovery protocols for bluetooth-based ad-hoc smart spaces: Architectural considerations and protocol evaluation," in *MDM*, 2010.

[9] J. Lehikoinen et. al, *Personal Content Experience: Managing Digital Life in the Mobile Age*. Wiley-Interscience, 2007.

[10] A. Popescu, "Geolocation API Specification," W3C, Editor's Draft, September 2009. [Online]. Available: http://dev.w3.org/geo/api/spec-source.html

[11] R. Berjon et. al., "W3C device apis and policy working group," 2010. [Online]. Available: http://www.w3.org/2009/dap/

[12] K. Waters et. al., "Delivery Context: Client Interfaces (DCCI) 1.0," W3C, Candidate Recommendation, 2007. [Online]. Available: http://www.w3.org/TR/2007/CR-DPF-20071221/

[13] Google Inc., "Gears API," 2007. [Online]. Available: http://code.google.com/intl/de-DE/apis/gears/

[14] N. Mehta et. al., "Indexed database api," W3C, Working Draft, 2010. [Online]. Available: http://www.w3.org/TR/IndexedDB/

[15] I. Hickson, "Web sql database," W3C, Editor's Draft, 2010. [Online]. Available: http://dev.w3.org/html5/webdatabase/

[16] ——, "Web storage," W3C, Editor's Draft, 2010. [Online]. Available: http://dev.w3.org/html5/webstorage/

[17] A. Brodt et. al., "Deep integration of spatial query processing into native rdf triple stores," in *SIGSPATIAL*. ACM, 2010.

[18] Google Inc., "Android developer's guide, security and permissions," 2010. [Online]. Available: http://developer.android.com/guide/topics/security/security.html

[19] Forum Nokia, "Application signing," 2008. [Online]. Available: http://wiki.forum.nokia.com/index.php/Application_Signing

[20] T. Neumann and G. Weikum, "The RDF-3X engine for scalable management of RDF data," *VLDB Journal*, 2010.