Context-driven RDF Data Replication on Mobile Devices [1]

Editor(s): guest editors
Solicited review(s): Jérôme Euzenat, INRIA Grenoble Rhône-Alpes, France. Martin Raubal, University of California, USA.

Stefan Zander [a] and Bernhard Schandl [a,b]
[a] *University of Vienna, Research Group Multimedia Information Systems*
Liebiggasse 4/3-4, 1010 Wien, Austria
E-mail: {firstname.lastname}@univie.ac.at
[b] *Gnowsis.com*
Graumanngasse 7/B/5, 1150 Wien, Austria
E-Mail: bernhard.schandl@gnowsis.com

Abstract. With the continuously growing amount of structured data available on the Semantic Web there is an increasing desire to replicate such data to mobile devices. This enables services and applications to operate independently of the network connection quality. Traditional replication strategies cannot be properly applied to mobile systems because they do not adopt to changing user information needs, and they do not consider the technical, environmental, and infrastructural restrictions of mobile devices. Therefore, it is reasonable to consider contextual information, gathered from physical and logical sensors, in the replication process, and replicate only data that are actually needed by the user. In this paper we present a framework that uses Semantic Web technologies to build comprehensive descriptions of the user's information needs based on contextual information, and employs these descriptions to selectively replicate data from external sources. In consequence, the amount of replicated data is reduced, while a maximum share of relevant data are continuously available to be used by applications, even in situations with limited or no network connectivity.

Keywords: Mobile applications, data replication, context awareness

1. Introduction

Mobile devices have become central parts of our everyday lives for managing our digital assets and lifestyle. Due to the convergence of traditionally separated networks and information channels and the continuing technical progress of mobile devices, network and online services can now be accessed regardless of spatial or temporal constraints: *anytime*, *anywhere*, and whenever network connection allows to do so. In parallel, with the emergence of the Web of Data, the amount of structured data available on the Web and in particular on the Semantic Web has been continuously growing throughout the past years. An increasing advent of applications that utilize and integrate such data from different, distributed sources can be observed, providing additional services on top to users or software agents. This trend is even more accelerated by so-called *Semantic Web 2.0* [23] applications.

A common strategy to maintain service availability and to guarantee a certain service quality is replication of remote data sets. However, traditional replication mechanisms do not apply properly to mobile scenarios for the following reasons:

– *Technical limitations:* mobile devices are restricted in terms of memory capacity, CPU performance, and power supply, which may hinder the local replication of large data sets.

[1]This paper is an extended version of [49].

- *Environmental, infrastructural, and security constraints*: network connectivity may be limited technically (no cellular radio coverage), economically (the network connection may be expensive), or because of security restrictions (e.g., when the network does not permit to establish a VPN connection). Consequently, in case of unstable network connections only the most important and relevant data sets should be replicated.
- *Different application and operation models*: mobile devices employ different application models and operating system infrastructures originating from ad-hoc and situational usage-patterns. For instance, since mobile devices use different modalities in accessing information resources and are operated in different contexts, it is more common that current tasks might be intermitted abruptly or moved to the background.

Due to these significant differences, mobile data replication should consider the importance of replicated data in relation to user tasks and activities as well as their operating environments. We therefore adopt the concepts of *context* and *context awareness* and utilize them for replication of RDF data sets to mobile devices. This allows for a proactive, selective, and transparent replication, focusing on the user's situation and information needs. Our proposed solution addresses these issues from two sides: first, it considers the current (and future) context of the user and, based on this information, selects subsets of remote data sources for replication. Hence, the amount of data to be replicated (i.e., to be transferred to, and stored on the mobile device) is reduced. Second, these subsets are replicated to the mobile device proactively and transparently, whenever network connectivity allows to do so. As a consequence, data are still available when no network connectivity is present, while access times are significantly reduced since data can be reused from the local replica. As a side effect, semantic technology infrastructure is brought to mobile devices, which can be utilized by any application.

This paper presents the *MobiSem Context Framework*[1], which is designed as a situation-sensible infrastructure framework for Semantic Web applications running on mobile devices. It uses a loosely coupled combination of context- and data providers to populate the local triple store with data from remote sources. It considers context information acquired from the device itself or the surrounding environment, thus hiding the tasks of context acquisition and data provisioning from users and applications.

We want to motivate our approach through a typical use case of a knowledge worker and its daily working data items. In this scenario, we assume that the user will be on travel during the next three days, where a number of business meetings will take place. The user cannot rely on a stable network connection during this trip. Therefore, it is desirable to transfer relevant information about the meetings and, in particular, the persons that will participate in the meetings to the mobile phone. This information can originate from a variety of sources, including public knowledge bases like Wikipedia/DBpedia[2] and GeoNames[3], or the user's private data which might be available through their Semantic Desktop environment (e.g., relevant documents, email messages, and to-do lists, which are represented in a machine-processable format).

The next section (Section 2) gives an introduction to the notion of context and context awareness, and it elaborates on how they can be augmented using Semantic Web concepts and technologies. It is followed by an overview of currently available mobile RDF frameworks and related context-aware Semantic Web projects (Section 3). The architecture of the MobiSem framework, which can be deployed to mobile devices, and technical details are presented (Section 4). The feasibility of this context awareness approach is shown through a prototypical implementation of an application scenario on the Android platform (Section 5) followed by a comparative performance evaluation of the underlying RDF processing infrastructure (Section 6).

2. Context and Context Awareness

For the intelligent provision of user-related data, context awareness it is essential to capture the context in which the user currently operates. We aim to utilize the notion of context in order to describe and represent the user's information needs so that relevant data sets can be proactively retrieved from external data sources in a transparent and automated manner. In this section, we provide an introduction to the concepts of context and context awareness from a technical perspec-

[1]The MobiSem Context Framework has been developed in the course of the MobiSem project (see `http://www.mobisem.org`) and is currently being transitioned into a commercial solution.

[2]`http://dbpedia.org/About`

[3]`http://www.geonames.org/`

tive and discuss their main problems when used in information systems. That followed, we present several areas where concepts and technologies from the Semantic Web can substantially enhance context-aware computing on mobile systems.

2.1. Context and Context Awareness in Information Systems

Many definitions have been proposed for the notions of context and context awareness. Context in its widest sense is defined as "*everything that surrounds a user or device and gives meaning to something*" [43] as well as "*anything that can be used to characterize the situation of an entity*" [14]. We define the term *contextual information* to refer to any information that is relevant for describing the situation a user or device operates in. Consequently, context can be acquired *explicitly* where context related information is manually specified by the user, or *implicitly* where context information is captured using specific technologies such as sensors or network communication, or by monitoring user behavior. The main focus of our framework lies on the implicit acquisition of contextual information, especially from physical sensors embedded in the device or ubiquitous sensors located in the immediate vicinity, as well as *logical* or *software sensors* that extract context-relevant information from personal sources such as emails, calendars, or web services. In this respect, the challenge is to identify the set of relevant features used for capturing and describing a situation or parts of the environment sufficiently [4].

In general, two forms of context awareness can be found in information systems [21]: *direct awareness* shifts the process of context acquisition onto the device itself, usually by embodying sensors that autonomously obtain contextual information; e.g., location ascertainment using the device-internal GPS sensor. *Indirect awareness*, in contrast, captures contextual information by communicating with sensors or services via the surrounding environment or infrastructure. For instance, to capture the social context of a user, a mobile device may request data from social communities or portals; to track the user's location, a remote geocoding service (based on the user's IP address) may be employed.

A fundamental problem of context-sensitive systems is that there exists no general model of context and context awareness. Especially in mobile computing, the notion of context is used very ambiguously across communities and is usually defined according to specific application domains (cf. [8,37]). This problem is also reflected in the developments of mobile context-aware applications since no widely accepted and well-defined programming model exists, resulting in a tight coupling and low-level interaction between application code and context acquisition components. Consequently, interpretation and exchange of sensed values is anchored within applications in a proprietary manner. Recent approaches propose a more flexible architectural and conceptual design for representing and processing context-relevant information by using formal models for context and context awareness (e.g., [4,6]) complemented with user and task analysis to enable a dynamic interaction between context-, task-, and user models (e.g., [38]), or employ middleware infrastructures (e.g., [24,28]) that encapsulate sensor-specific APIs in dedicated components in order to facilitate communication and interoperability between context processing components[4] and the underlying framework, while making use of knowledge representation frameworks such as RDF for describing context information [17,35].

The MobiSem framework extends this idea in that it has been designed specifically to operate on mobile systems, and to use Semantic Web technologies to acquire, interpret, aggregate, store, and reason on contextual information, independent of any application or infrastructure. Semantic Web technologies and practices, which are designed as an information processing infrastructure for heterogeneous environments, can help to solve some of the issues described before, and are therefore highly relevant for the design and development of ubiquitous and mobile context-aware systems.

2.2. Dynamically Evolving Context Descriptions

Especially in technical disciplines it is a predominant practice to concentrate on sensorial and static data such as location, time, identity, activity etc. (cf. [34,41]). In such disciplines, context is predominantly considered as a representational issue, where the focus is put on its codification and representation [43]. According to that perception, context can be scoped in advance, is instance-independent, and separable from user activities [15]. The reasons for that predominant practice of utilizing context in information systems

[4]That is *context producers* such as context acquisition components, and *context consumers* such as services, applications, or the device itself.

can be attributed to the adherence to existing software methodologies [43].

However, this static perception entirely neglects the dynamic aspects of context, which arise in the course of interaction and render the determination of relevance of contextual facts a priori at design time impossible. Context should be considered as an *emergent phenomenon* or *feature of interaction* that is centered around user activities [43] and continuously renegotiated between communicating partners [12,15,34]. Therefore, the determination of a relevant set of canonical context properties in advance is very difficult and nearly impossible [17].

To cope with the dynamic and emergent nature of context, a context processing and management framework must facilitate flexible, extensible, and open context descriptions that are not restricted to a single static vocabulary or predefined schema. Static context descriptions are not able to deal with unknown context information at run time, but require links between different context vocabularies to be specified at design time [17]. Therefore, a fundamental requirement of our proposed context framework is the ability to handle new types of context information dynamically using well-accepted standard vocabularies to guarantee their accuracy and evolution. In this respect, one can observe an analogy to the "real" Semantic Web which deals with providing infrastructure to process information in a distributed and heterogeneous manner.

A general problem of context management and processing is that of *context ambiguity* [14]. Most context-aware computing approaches are based on the implicit assumption that the acquired context is a 1-to-1 representation of the surrounding real world context. Obviously, this assumption is wrong due to the inherently existing differences in the way context is sensed and represented electronically, and the way it is perceived by individuals [14,15]. Therefore, a context framework can only work on a more or less accurate representation of the surrounding real-world context, where the degree of accuracy depends on a multitude of different factors (e.g., the user's task at hand, their information needs, personal goals, etc.). The dynamic nature of context makes it difficult to specify all relevant context parameters at a system's design time, since in general context is always defined relative to the situation in which it is used. Modeling context in information systems is therefore never universal in that a context model encompasses all information characterizing a certain situation, but rather represents a relevant subset of the constituting characteristics [14,15]. This leads to cases of having multiple representations of the same situation differing in the accurateness and the contextual aspects they include.

Detecting all artifacts that constitute a specific context is nearly impossible and cannot be fulfilled by any context framework. However, applying reasoning and machine learning techniques only increases the accuracy of context acquisition and context recognition processes but never accounts for identifying all the possible artifacts constituting a specific context or situation respectively. Context-aware computing is therefore always an approximation to a real-world situation rather than a 1-to-1 reflection of it.

To deal with that issue, several techniques and methodologies from different fields such as activity theory [36,43], aspect-oriented context modeling and modularization [11], or situational reasoning [6,33] have been applied to process context on higher, more abstract levels by aligning contextual aspects to abstract concepts (e.g., "business meeting") adhering to upper-level ontologies. The idea is to aggregate and transform quantitatively acquired context artifacts into qualitative statements in order to express *complex conceptual relationships and dependencies* [6], and for applying *classification-based reasoning techniques* [22]. Additionally, high-level representations of contextual information unify access and utilization among applications. Context consumers do not need to be familiar with low-level data processing and interpretation, thus context sharing and exchange are simplified. Such transformations are considered as a means to make contextual information domain-independent.

The MobiSem Context Framework follows this idea in that it provides the technical infrastructure on which additional more sophisticated layers (e.g., for situation-awareness) can be deployed. Such layers allow for aggregating the contextual artifacts acquired by the underlying framework and apply different methodologies (e.g., Bayesian networks, case-based reasoning, stochastic methods, etc.) for their interpretation, consolidation, and augmentation. Section 5 describes a use case in which new contextual information can be derived by intelligently combining independently acquired contextual artifacts to provide a more sophisticated representation of a contextual aspect (in that case, location).

2.3. *Semantic Web-based Context Representation and Processing*

A general approach to systematically manage context information is to use ontologies, which provide a common structure for representing and describing information. The Resource Description Framework (RDF) enables communication and sharing of context descriptions between collaboratively communicating partners; i.e., services or devices. Its open architecture allows for the integration of different vocabularies so that context descriptions can dynamically grow and become more elaborated. Different works in the fields of pervasive and ubiquitous computing (e.g., [8,17]) have shown that both RDF(S) and OWL are appropriate languages for representing dynamic and evolving context descriptions [34]. Since these are grounded on the *open world assumption*, the possibility of adding new and more detailed information to existing descriptions makes them applicable in dynamic and unpredictable environments.

Ontologies further help in matching expressed context information to application or service needs in that only relevant statements are extracted. A context consumer, i.e., a device or application only needs to query for the information it is interested in, instead of processing the entire context description. If parties expose context descriptions that cannot be understood by others, ontology matching algorithms can be applied in order to reconcile differences in the description semantics. *Ontology alignment services* [16] can be used to account for the compatibility between different context models by identifying correspondences between context descriptions and performing query transformations to better reflect domain and information space evolutions [17].

Semantic technologies facilitate both direct and indirect context awareness, since context-related information can be acquired from external services or repositories in a structured and well-defined way based on explicitly represented semantics using open standards. Sensorial context data can be mapped to vocabularies so that sensed values are embedded in a controlled context description based on ontological semantics, where new facts can be discovered via aggregation and reasoning. In this respect, RDF simplifies the aggregation of heterogeneous context information, both on the semantic and syntactical layer.

Since technologies and concepts from the Semantic Web have been designed for heterogenous environments, they offer languages and technologies that serve as standards for expressing contextual information, and can therefore be shared and exchanged among systems and applications. RDF further allows one to represent contextual information in multiple ways by using different vocabularies and transformation rules so that it can be used and understood by different components or context consumers. RDF thus facilitates transformations or mappings between heterogenous context representations as well as the reconciliation of context heterogeneity.

If context-relevant data are represented using Semantic Web languages, they can be integrated and processed even if they were not known at design time of a mobile system (see [17] p.30 for an example). This also applies to divergent sensor or service feature descriptions where the identification of correspondences between heterogeneous descriptions serves as a basis for utilizing services and integrating acquired information that were not been anticipated at design time of a mobile system.

Additionally, ontologies facilitate the interpretation of sensed or derived values to allow for their aggregation and transformation into symbolic values, i.e., transforming collected data into statements adhering to a prescribed vocabulary. Hence, context acquisition components do not need to anticipate possible queries beforehand, but provide the data they have and let the requesting components decide which information is of relevance to them.

The Semantic Web community has already developed a wide range of vocabularies that can be used to describe contextual information (including physical parameters like time[5] and location[6], technical parameters[7], or social aspects[8]). The terms defined in these vocabularies are known across communities and adhere to a well-defined and commonly understood semantics. Such vocabularies facilitate data interchange between heterogeneous systems, and are often maintained by a large number of people to guarantee their accurateness and relevance. Not being bound to a single vocabulary also adheres to the idea of dynamic and flexible context descriptions evolving in the course of user-relevant activities that can not be determined a priori—especially not at design time of a mobile system or a mobile application.

[5]http://www.w3.org/TR/owl-time
[6]http://www.w3.org/2003/01/geo
[7]http://www.w3.org/Mobile/CCPP
[8]http://www.foaf-project.org

In this section, we have outlined some of the areas in context-aware computing where Semantic Web technologies can make substantial contributions in representing, processing, and sharing contextual information as well as in the reconciliation of heterogeneous context semantics. The potential benefits of semantic technologies for related areas such as pervasive computing have been discussed in previous works [17]. In the following, we discuss relevant work in terms of related mobile replication approaches, mobile RDF frameworks, and existing context-aware mobile Semantic Web applications, and provide an overview of the MobiSem Context Framework that implements the ideas and concepts presented in this section. We denote this form of context-aware computing as *Semantic Web-based context-aware computing*.

3. Related Work

For realizing our idea of making Semantic Web technologies available on mobile systems for the intelligent, context-dependent provision of user-related data, we analyzed existing Semantic Web frameworks according to their appropriateness and deployability on mobile platforms and discuss existing projects that aim to synthesize semantic technologies, mobile systems, and context-aware computing.

3.1. Mobile Data Replication

The problem of replicating data to mobile devices is not new. Standard replication strategies—as known e.g., from relational data bases—cannot be directly applied to mobile scenarios because of the special restrictions imposed by changing context parameters, as outlined in Section 2. Therefore, several algorithms were proposed that estimate the costs of data usage based on various context parameters, and adapt the used replication strategies accordingly (e.g., costs of data transmission [27], access frequency [47], location [48], or device and environment characteristics [3]). These approaches are highly optimized towards single specific context parameters but do not consider the entire user context; especially they do not focus on the semantics of replicated data. However, they can be considered complementary to our approach since they can be used to determine the frequency of replication updates.

Several approaches follow a more generic strategy and provide architectures that are extensible w.r.t. the considered context parameters and replicated data (e.g., [25,32]). However, all these approaches are depending on a server infrastructure, on which context processing and inference tasks are performed. To the best of our knowledge, no approach exists that solely relies on processing executed on the mobile device itself, without depending on external components and services.

3.2. Mobile Semantic Web Frameworks

Typical Semantic Web frameworks like *Sesame*[9], *Virtuoso*[10], and *Jena*[11] hide the details of RDF data processing, serialization, and query execution from higher-level applications. However, these heavy-weight systems cannot be deployed on typical mobile devices because of their limited memory and processing capacities, latencies as well as incompatible application models and operating system infrastructures [18,30]. Those frameworks are usually developed for powerful server or desktop computing infrastructures incorporating many-core architectures, whereas mobile devices in general contain dedicated single-core RISC-based processors whose architecture was not designed for processing large data amounts.

Although they have proven to be powerful means to process, store, and reason over RDF data, they cannot be efficiently deployed on mobile systems due to the previously mentioned reasons and are therefore not considered in our related work analysis. Instead, we exclusively concentrate on RDF frameworks that have been specifically designed for deployment on mobile platforms and are available as Java libraries as well as mobile query and storage frameworks that are built on top of existing RDF frameworks and provide additional functions for local RDF data query and persistence.

3.2.1. Mobile XML Parsers

kXML[12] is a lightweight XML pull parser that was specifically designed for constrained environments such as Applets or Java ME-based mobile devices. It is based on the *Common XML Pull API*[13] and combines advantages of XML DOM and SAX parsers, such as aligning XML processing routines to the structure of an XML document and, at the same time, providing instant access to parsed document elements. It was

[9]`http://www.openrdf.org`
[10]`http://virtuoso.openlinksw.com`
[11]`http://jena.sourceforge.net`
[12]`http://kxml.sourceforge.net/`
[13]`http://xmlpull.org/`

specifically designed to be used in CLDC[14] applications. However, development stalled in 2005.

NanoXML for J2ME (+RDF/OWL)[15] is a J2ME port[16] of the original non-validating XML parser NanoXML[17] for Java, and has been extended with RDF and OWL support. It is dedicated to mobile environments and offers convenience methods for navigating and retrieving data from RDF and OWL documents such as resource or property values, but neither supports inferencing nor elaborates on RDFS/OWL semantics.

3.2.2. Mobile RDF Frameworks

Mobile RDF[18] is a Java-based open source implementation for the RDF data model, providing a simple and easy-to-use API for accessing and serializing RDF graphs. It is specifically designed for Java ME Personal Profile[19] and Connected Device Configuration (CDC)[20] compliant devices, which is one of the main drawbacks of this framework since these application environments are only supported by a comparatively small amount of devices, namely those that employ a CDC-specific Java Virtual Machine (JVM). Most current and older J2ME-compliant devices deploy the more widely-used CLDC profile. It provides specific packages for creating, parsing, and serializing RDF/S and OWL ontologies, and supports RDF Schema type and property propagation rules as well as rule-based inferencing. However, RDF graph modifications like deleting or editing RDF triples are not supported.

μJena[21] is a port of the popular Jena Semantic Web framework, targeted for low-capacity mobile and embedded devices. Although its API is currently in a prototypical state and only allows for processing RDF data serialized in N-Triples format, it covers the entire set of RDF modeling primitives, provides ontology and limited inference support, as well as convenience classes for handling OWL ontologies. Like in Jena, RDF data are represented on two levels: on the lower more generic level, μJena stores triple nodes, where a model API is deployed on top that offers convenience methods for accessing and manipulating RDF models.

Androjena[22] is a more recent Jena port specifically created for the Android platform. It adopts Jena version 2.6.2 and offers all the functions and libraries Jena includes such as full RDF and ontology support, inferencing, as well as reading and writing RDF data in different serialization formats. The Androjena core libraries—as the original Jena libraries—do not include specific APIs for querying RDF data, persistence, Named Graphs [10], or support for external reasoners. However, to provide at least a minimum of query functionality, the Androjena project page also hosts the *ARQoid* project[23], which is a reduced port of Jena's SPARQL query engine ARQ. Currently, ARQoid is in prototypical status and lacks some of ARQ's original features such as full-text query support.

In summary, none of the existing mobile RDF frameworks fully supports queries on RDF data via SPARQL or other query languages, although Androjena provides a prototypical implementation of the Jena ARQ libraries. A storage mechanism that translates RDF data into internal storage formats used by mobile devices (e.g., the *SQLite* database provided natively by the Android platform) and vice versa could also not be found.

3.2.3. Query and Persistence Frameworks

RDF On the Go[24] is a full-fledged RDF storage and query framework specifically designed and implemented for mobile devices that feature the Android operating system. It follows an approach similar to Androjena, as the Jena core APIs including ARQ have been adapted to the Android platform to allow developers to directly operate on and manipulate RDF data models. The primary storage infrastructure are *B-Trees* as provided by a lightweight version of the *Berkeley DB*[25] adopted for mobile usage and deployment. The internal query processor provides support for both standard and spatial SPARQL queries, where an *R-Tree* based indexing mechanism is used for storing URIs with spatial properties [31]. The current ver-

[14] `http://java.sun.com/products/cldc/`

[15] `http://nanoxml-j2me.sourceforge.net`

[16] `http://java.sun.com/javame/index.jsp`

[17] `http://devkix.com/nanoxml.php?lang=en`

[18] `http://www.hedenus.de/rdf/index.html`

[19] `http://java.sun.com/products/personalprofile/`

[20] CDC is a framework specification for deploying and sharing mobile Java applications on hardware-constraint devices such as mobile devices or set-top boxes. It defines a basic set of libraries and virtual machine features that the underlying runtime environment must exhibit.

[21] `http://poseidon.elet.polimi.it/ca/?page_id=59`

[22] `http://code.google.com/p/androjena/`

[23] `http://code.google.com/p/androjena/wiki/ARQoid`

[24] `http://code.google.com/p/rdfonthego/`

[25] `http://www.oracle.com/technetwork/database/berkeleydb/overview/index.html`

sion as of March 2011 supports a large set of standard SPARQL query operations where aggregation, sorting, and some spatial operations are subject to future implementations [31].

SWIP: Semantic web in the pocket[26] was developed in order to support RDF data storage and exchange in a uniform, schema-less, and system-wide way based on the Linked Data principles [5]. SWIP represents an Android-specific implementation of an RDF storage infrastructure that is based on the Android-internal concept of *ContentProviders*[27] for application-wide data storage and exchange across applications and processes. It maps URIs to data stored in the local SQLite database deployed on Android systems and returns data in the form of triple sets or tuple tables. It employs a simple subject-predicate-object table layout for RDF data storage and is currently in prototypical status [13]. For demonstration purposes, data stored in device-internal data sources such as calendar entries or contacts have been exposed as RDF-based Linked Data and visualized through a generic browser interface.

However, these RDF storage and query infrastructures are available as experimental prototypes or concept studies and lack specific storage and query optimizations for mobile platforms. Nevertheless they demonstrate that typical RDF processing and storage tasks can be executed on mobile devices although the efficient execution of complex processing operations (e.g., reasoning) or indexing mechanisms is still subject to further research.

3.3. Mobile Semantic Web Applications

DBpedia Mobile[28] [2], a location-aware mobile application, allows users to access information from the DBpedia project[29] about the physical environment surrounding them. Users are able to receive additional information by exploring links to other resources located in the Semantic Web.

mSpace Mobile[30] [46] takes a similar approach, where access to related location-based information with respect to the user's current situation is provided via a spatial browser. Considered contexts are time, space, and subject.

[26]http://swip.inrialpes.fr/
[27]http://developer.android.com/guide/topics/providers/content-providers.html
[28]http://wiki.dbpedia.org/DBpediaMobile
[29]http://dbpedia.org
[30]http://mspace.fm/projects/mobile

IYOUIT[31] [6] collects contextual information about certain aspects of the user's lifestyle—such as visited places, or people met—and displays them on the Web. People are able to share their personal contexts within a community portal.

Although these projects make use of Semantic Web technologies such as RDF, the processing of contextual data is done on external servers or applications rather than on the device itself. This means, however, that in case of missing network connectivity the applications become practically useless. While a system that is deployed on the mobile device also does not allow to proactively update data from remote sources without connectivity, it provides at least a local buffer of the data that has been replicated so far, and hence allows the user to continue using the applications, although in a restricted manner. Another distinct aspect is that context acquisition and context representation is not limited to a predefined set of contextual aspects, i.e., the context descriptions created by the framework are dynamic and include as many aspects as could be acquired. Applications can process the data they are interested in leading to a greater flexibility in elaborating on contextual constellations.

In summary, our analysis revealed that context-driven replication of RDF data to mobile devices has not been addressed by current or related research yet. The RDF frameworks currently available for mobile systems provide the necessary functions for such a replication infrastructure although much space is left for optimization. In Section 6 we therefore analyze the performance of mobile RDF frameworks in creating, parsing, and storing RDF triples directly on a device. There exist a few mobile storage and query frameworks however, but they are mostly in prototypical status to date although recent developments indicate an increasing awareness of deploying Semantic Web technology on mobile devices (cf. exploiting linked data for mobile Augmented Reality [39], SWIP [13], i-MoCo [45]).

4. System Design and Architecture

The MobiSem framework has been specifically designed for direct deployment on mobile platforms. This allows it to acquire, process, store, and manage contextual information independently of any applica-

[31]http://www.iyouit.eu/portal

tion or client-server infrastructure. The main goals of the MobiSem Context Framework can be summarized as follows:

– *To provide a storage repository for semantic data on a mobile device.* With the increasing proliferation of services based on Semantic Web technologies, the need for mechanisms to store, manipulate, and retrieve RDF data on mobile devices becomes apparent. The local storage of RDF data on a mobile device not only reduces the dependency on a permanent network connection, but also enables the implementation of more efficient search and reasoning algorithms, and extends the user's local information space.
– *To make efficient use of available context information.* Modern mobile devices provide a magnitude of options to capture the user's context, which can be used to infer future information needs and adapt application and device behavior. A semantically appropriate interpretation of these context data helps to build more user-oriented applications and services and enhance the overall mobile user experience.
– *To proactively provide context-relevant data on the device.* As stated before, we cannot rely on a permanent network connection in mobile scenarios. On the other hand, we can infer future information needs from the user's current context information and thus proactively retrieve data from remote data sources to the mobile device that might become relevant in the future, and buffer it using the local storage repository.
– *To provide the technical infrastructure for high-level context processing.* The dynamic and flexible characteristic of our context framework enables the deployment of additional high-level context recognition and utilization services on mobile devices to enable situation-awareness (cf. [1,19,33,42,44]). The framework facilitates almost all aspects of a mobile context processing and management architecture and serves as a foundation for the systematic management and exchange of context descriptions using open semantic standards.

To realize these goals it is necessary to combine the processing of context information with the local replication of remote data sources. However, it is also necessary to keep the framework design as flexible as possible: it depends on the capabilities of the mobile device which context information can be tracked. Further, the user's information needs might evolve over time, hence the approach cannot be restricted to a fixed set of remote data sources and should be flexible enough to enable the dynamic integration of new potential context sources on the fly.

We have decided to decouple the tasks of context acquisition and data replication (cf. Figure 1). Context relevant data are retrieved by dedicated components (called *context providers*) and are converted into RDF-based context descriptions. These descriptions are aggregated to an RDF-based *global context model* that is used by *data providers* to replicate RDF data to the device. Replicated data are stored in a local *triple store* and made available through a *data access API*. A loose, data-based coupling between context providers and data providers is realized through a *context dispatcher*, which is notified every time a context provider detects a change in a context source it observes. The context dispatcher aggregates, consolidates, and reasons on context information, and forwards them to the appropriate data provider components.

This architecture exhibits two significant advantages in comparison to server-based approaches, as it does not require context information to be transferred outside the mobile device. First, the system does not depend on the availability of an external system. Second, all contextual data (which may include highly private information, like the current position, contacts, appointments, and so on) are processed only on the mobile device, which reduces security and privacy issues.

In the following, we describe in more detail the individual system components.

Context Providers We employ two types of context providers: *primary* (i.e., active) and *complementary* (i.e., passive) context providers. Primary context providers encapsulate a hardware or software sensor and become active whenever a change in a context source is detected. Complementary, that is passive or re-active context providers react according to changes in primary context providers and become active when a corresponding primary context provider delivers an updated context model. They complement the contextual data retrieved from primary context providers by taking these context descriptions as input for initiating their acquisition tasks (*context augmentation*).

To provide the necessary flexibility in acquiring context-relevant data, context providers implement their own logic and heuristics for transforming any kind of input data (either sensorial or web-based content) into an RDF-based context description by using

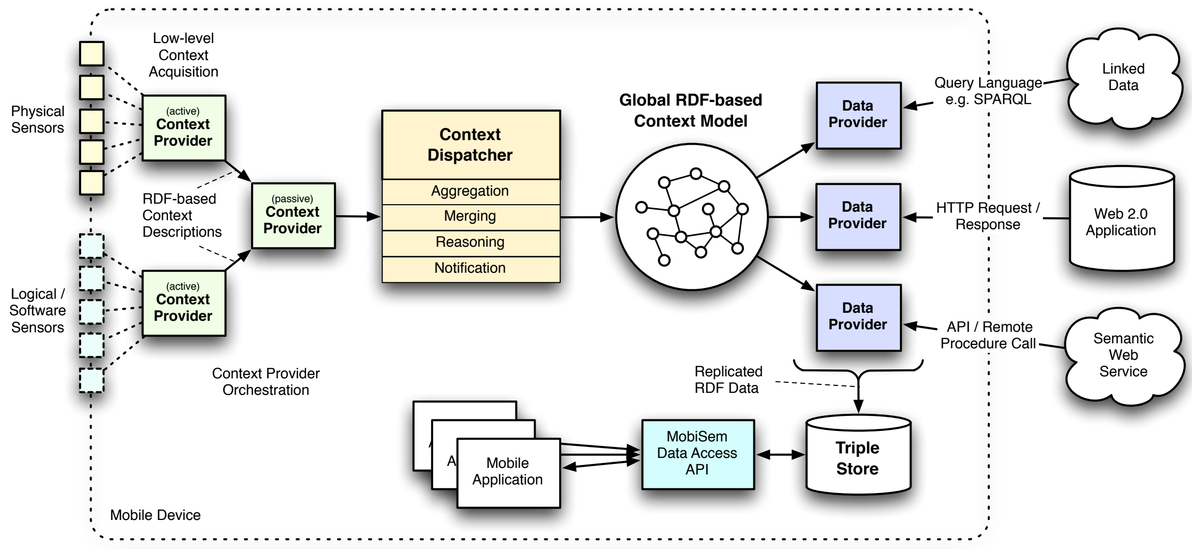
Low-level Context Acquisition
Physical Sensors
(active) Context Provider
RDF-based Context Descriptions
(passive) Context Provider
Logical / Software Sensors
(active) Context Provider
Context Provider Orchestration
Context Dispatcher
Aggregation
Merging
Reasoning
Notification
Global RDF-based Context Model
Data Provider
Data Provider
Data Provider
Query Language e.g. SPARQL
HTTP Request / Response
API / Remote Procedure Call
Linked Data
Web 2.0 Application
Semantic Web Service
Replicated RDF Data
Mobile Application
MobiSem Data Access API
Triple Store
Mobile Device

Fig. 1. Architecture of the MobiSem Context-Processing Framework

well-defined and well-accepted semantic vocabularies. As previously outlined, the acquisition of contextual data should not be restricted to capture sensorial data exclusively since the Internet and Web 2.0 applications in particular provide excellent sources for gathering context-relevant data. Context providers therefore are able to request data from four different types of sources:

(i) *Hardware sensors* that are integrated into the mobile system such as GPS module, luminosity sensor, camera etc. Most modern mobile platforms provide specific APIs for accessing and utilizing locally deployed hardware sensors.
(ii) *Ubiquitous sensors or devices* that are located in the physical environment [20]. Such sensors must provide open accessible interfaces based on open network and access protocols.
(iii) *Web applications* such as Facebook[32], LinkedIn[33] etc. often contain useful information w.r.t the users' social relationships. Online and Linked Data repositories in particular provide magnitudes of freely available context-relevant data that can be exploited for complementing sensorially captured data.
(iv) *Software* or *logical sensors* that allow for monitoring user or application behavior to deduce on the type of data that is relevant to the user in a specific situation.

By employing logical sensors, the acquisition of user-related contexts is emphasized. Such sensors can be adjusted towards a particular system infrastructure to gather context-relevant information by monitoring system processes to deduce information about the currently running applications as well as the data they operate on[34]. Context providers can make use of context descriptions from other context providers as well as external data sources; e.g., a component may use the GPS coordinates provided by another context provider to look up names of the current location using an external service[35].

Orchestration Framework To facilitate this kind of cooperation between decoupled context providers, an *orchestration framework* dynamically routes data between context providers based on the type of context information they provide. It orchestrates context providers in form of a directed acyclic graph. Within this graph, *primary context providers* represent starting nodes, while *complementary context providers* represent adjacent nodes. Edges represent data flow between context providers; i.e., they indicate compatibility in terms of contextual data so that the data deliv-

[32]`http://developers.facebook.com/`
[33]`http://developer.linkedin.com/index.jspa`

[34]We implemented software sensors that track user queries issued to various mobile applications such as browsers or the internal 'quicksearch'-function on an Android device.
[35]See Figure 4 in Section 5 for an example.

ered by one context provider can be further processed by another context provider.

The orchestration framework analyzes the *data description* of each context provider. Such a data description consists of sets of mandatory and optional namespaces as well as terms, which can be processed as input data by the respective context provider, as well as namespaces and terms that the context provider uses in its output data.

Figure 2 depicts an excerpt of an exemplary data description. This complementary context provider extracts contact data from acquired calendar data. A data description consists of an *input description* (indicated by the `ddesc:input` property) and an *output description* (indicated by `ddesc:output` property). The former specifies the data a context provider needs for performing its acquisition tasks. It may contain multiple `ddesc:vocabulary` properties, covering the case that context providers may be capable of processing data described with different vocabularies. Multiple vocabulary properties are interpreted by the orchestration framework as alternatives, that is, they are interpreted as being connected with a logical *or*.

A vocabulary specification consists of three parts: the `ddesc:namespace` property, which holds the vocabulary's namespace that is used for an upper-level orchestration, and the `ddesc:concepts` and `ddesc:properties` statements, which specify mandatory and optional concepts and properties that the context provider processes. Mandatory elements (indicated by the `ddsec:mandatory`-property) are those that are inevitable for a complementary context provider to perform its acquisition tasks. Optional elements (indicated by the `ddsec:optional`-property) refer to those elements that a context provider is capable to process but they are not necessarily needed for a successful execution of the context provider's acquisition tasks. Those specifications allow for a detailed, element-level orchestration of context providers.

Additionally, a data description specifies the namespaces and terms that the context provider emits as output data (indicated by the `ddesc:output` property). This property is mandatory for all context providers. The output description follows the schema of the input description, consisting of parts for vocabulary, concepts, and properties. In contrast to the input specification, the output specification may consist only of mandatory elements[36].

The orchestration framework can be configured to either perform a loose orchestration on the namespace level, or a detailed one by considering concepts and properties given by the context providers' data descriptions. When a new context provider is found in the system, the orchestration manager analyzes its data description and based on its configuration integrates the context provider in the *orchestration graph*. While running completely decoupled from the context framework, rebalancing the orchestration graph does not affect context acquisition tasks as such.

The orchestration graph is represented as an *adjacency matrix* whose values are decimal numbers between 0 and 1, indicating the degree of compatibility between two context providers. The matching value for each pair of context providers is computed by a *matching algorithm* based on configurable scores for correspondences on the namespace, concept, and property levels. The matching algorithms performs an arithmetic matching based on data similarities and is additionally capable of including RDFS semantics such as `rdfs:subClassOf` relationships. For instance, if one context provider emits `foaf:Person` instances and another context provider requires `foaf:Agent` instances as input data, the matching algorithm detects the compatibility between these differing concepts since `foaf:Person` is a subclass of `foaf:Agent` according to the FOAF ontology [9].

Context Dispatcher The context dispatcher is notified by context providers whenever a context description has changed. Before propagating updated context descriptions to data provider components, the dispatcher performs additional processing on the data, like inference and consolidation. Currently, the reasoning component uses (*i*) a generic lightweight rule-based reasoner, which allows to specify conditions under which new triples are added to the knowledge base, and (*ii*) hard-coded rules which are expressed by implementing a Java interface. The combination of these two mechanisms can, for instance, be used to specify that if one resource has multiple values for a functional property, the values denote the same resource (the corresponding rule $(\texttt{A :ifp X}) \wedge (\texttt{A :ifp Y}) \Rightarrow (\texttt{X owl:sameAs Y})$ can be interpreted by the rule-based reasoner), and that multiple resources that are re-

[36]According to the RDF semantics it is possible to specify optional data, although they will not be considered by the orchestration framework in its current version.

```
  <urn:uuid:b772a3a2-46d4-4c43-8f71-7080915ddba7>
    a  ddesc:ContextProvider ;
    ddesc:input [
      ddesc:vocabulary [
        ddesc:namespace <http://www.semanticdesktop.org/ontologies/ncal#> ;
        ddesc:concepts [
          ddesc:mandatory ncal:Attendee, ncal:Calendar, ncal:Event ;
          ddesc:optional ncal:Organizer, ncal:EventStatus ] ;
        ddesc:properties [
          ddesc:mandatory ncal:member, ncal:method ;
          ddesc:optional ncal:eventStatus ] ] ] ;
    ddesc:output [
      ddesc:vocabulary [
        ddesc:namespace <http://xmlns.com/foaf/0.1/> ;
        ddesc:concepts [
          ddesc:mandatory foaf:Organization, foaf:Person ] ;
        ddesc:properties [
          ddesc:mandatory foaf:knows, foaf:status, foaf:name ] ] ] .
```

Fig. 2. Exemplary data description for a complementary context provider for extracting contact data from calendar entries

lated via a `owl:sameAs` property can be merged into a single resource in order to simplify further processing (a corresponding algorithm can be implemented as a Java class and be integrated into the reasoning process).

Context descriptions are forwarded not only to data providers, but also back to context providers, so that they are enabled to mutually reuse and augment their context descriptions[37].

Communication between the context providers and the context dispatcher is realized via a context description queue that not only buffers the most recent context updates, but also stores previous context updates for compensation strategies in case a context source is temporarily not available or malfunctioning. In such cases, the context dispatcher can revert to previously committed context description to continue the context acquisition process. However, the context dispatcher employs some logic to maintain consistency among aggregated context descriptions.

Global Context Model The global context model represents an aggregated version of all context providers' context descriptions received by the context dispatcher. It is created whenever a primary context provider had detected a change in the context source it observes and delivered an updated context description. This context update will first be propagated to all complementary context providers to enrich it with additional data. When all context acquisition tasks are completed, the context dispatcher collects the updated context descriptions, aggregates them, applies reasoning rules as described before, and creates the global context model while maintaining context completeness, consistency, and accuracy.

Data Providers Data providers are responsible for handling RDF data replication tasks. They receive aggregated context description models from the context dispatcher and subsequently replicate data of any kind to the triple store. These data are usually retrieved from external data sources or may be generated by the data provider itself. For instance, a data provider may act upon changes of the current location and retrieve information about nearby points of interest. Each data provider is assigned a named graph under which it stores its data replicas in the triple store.

In addition to the default data providers that merely retrieve data from remote sources and store them in the triple store, we have implemented a *selective check-out data provider* that makes use of a partial versioning mechanism for RDF triples based on triple bitmaps [40] as well as a *write-back data provider* that synchronizes the partially replicated data back to the repository, if the latter supports write operations.

Triple Store Modern mobile platforms provide transparent access to persistent storage devices (e.g., flash memory cards) through a file system API. Therefore, the most straightforward way to store RDF data on a mobile device is to serialize it into a file on such a device using a standard RDF serialization format,

[37]Figure 4 depicts an example of augmenting GPS-coordinates with data from the `http://www.geonames.org` web service.

like RDF/XML or N3. While this storage mechanism is extremely fast compared to DB-backed mobile storage solutions (cf. Section 6), it also has the significant disadvantage that RDF graphs must completely be loaded into the mobile device's working memory (RAM) before they can be further processed (e.g., before a SPARQL query can be issued). Alternatively, triples can be stored in a relational database, which causes an increase of read and write times but provides the possibility for structured queries over the data.

Regardless of which actual storage solution is used, it can be wrapped by a Java class that maps all read and write access methods to corresponding operations on the underlying physical representation (either flat files or a relational model). Currently, our triple store implementation does not perform in-memory buffering or caching. However, it can be wrapped by an additional in-memory `Graph` instance (which provides faster access) that regularly synchronizes itself with the database-backed instance.

Data Access API Applications can use the MobiSem *Data Access API* to access data stored in the device's local triple store. The API assigns to each replicated graph a unique URI, which can be used to access and retrieve the data contained in the graph. It exposes insert, update, delete and query methods and offers multi-grained access to data replicas, i.e., applications can access all replicas cached in the database, a specific replica, or a specific resource including all adhering triples of a specific replica.[38] In the background, this API hides the details of context processing and data replication from applications; from the outside the MobiSem framework looks like a common triple store whose data are regularly updated.

5. Implementation and Case Study

To demonstrate the feasibility of our architecture, we have implemented a prototypical framework plus an initial set of context and data providers. The selection of these components is based on the assumption that the information needs of a mobile user depend on their current context (e.g., their location) as well as their future context. However, we want to emphasize that this framework is to be considered as an infrastructure, upon which end-user applications that provide specific functionality, based on specific context information and replicated data, can be built.

Our implementation is based on the Android platform[39] and uses the μJena Framework (cf. Section 3.2.2) to process RDF graphs[40]. In the following we demonstrate how the MobiSem framework can be used to proactively provide RDF data on the mobile device. Our objective is to permanently equip the user with data about the locations they are going to visit, about people they are likely to meet in the upcoming days, as well as people that are based near the user's current position. To accomplish this, different kinds of contextual information are utilized, including the device's current position and the user's calendar data.

Context Acquisition We have implemented three context providers: first, a location context sensor using the device's built-in GPS unit to track geographical coordinates returns context descriptions that contain a `context:currentLocation` property to describe the coordinates of the current location (cf. Figure 3).

A second context provider uses the GeoNames service[41] to resolve GPS coordinates to geographical entities. This component receives context updates from the context dispatcher, extracts properties that represent geographical coordinates, and returns information retrieved from the web service—in our example, a reference to a geographical entity as well as its name (cf. Figure 4).

In parallel, a third context provider regularly scans the user's calendar and extracts all appointments within the next 72 hours. From these appointments the e-mail addresses of all participants are extracted and returned, as depicted in Figure 5 (in this case, two e-mail addresses are returned). Further, the locations of appointments are extracted and are returned as GeoNames features. This context provider uses terms from the NEPOMUK ontologies[42] and from FOAF to describe the extracted resources.

[38]This functionality is implemented through an *Android Content Provider* that allows for defining explicit URI schemes for data replicas through which operating system-wide data access and data utilization is offered. By exposing distinct URIs (e.g. `content://org.mobisem.rdfprovider/graph#<graphid>`) triples can be retrieved, added, deleted, and updated.

[39]`http://developer.android.com`

[40]As shown in Section 6, μJena exposes a very weak performance compared to other RDF frameworks; however, more efficient implementations have been made available only recently. We plan to port our implementation to a more efficient RDF framework in the near future.

[41]`http://www.geonames.org`

[42]`http://www.semanticdesktop.org/ontologies`

```
[] a context:Context ;
  context:currentLocation [ geo:lat "48.175443" ; geo:long "16.375493" . ] .
```

Fig. 3. Context description retrieved by a GPS sensor

```
[] a context:Context ;
  context:currentLocation [ geonames:nearby <http://sws.geonames.org/2761369/> . ] .
<http://sws.geonames.org/2761369/>
  a geonames:Feature ; rdfs:label "Vienna" .
```

Fig. 4. Context description retrieved by a GeoNames sensor

```
[] a context:Context ;
  context:upcomingEvent [ ncal:attendee
      [ foaf:mbox <mailto:bernhard.schandl@univie.ac.at> ] ,
      [ foaf:mbox <mailto:stefan.zander@univie.ac.at> ] ;
    ncal:location [ a geonames:Feature ; rdfs:label "Munich" ] .
  ] .
```

Fig. 5. Context retrieved from the user's calendar

The context dispatcher—which receives notifications from the context providers every time a context value changes—buffers, combines, and enriches the context description graphs with additional information. It merges all resources typed as `context:Context` into a single one, assigns it a URI (enabling it to be referenced by other context descriptions), and adds a timestamp as well as a link to the preceding context descriptor. Moreover, it applies simple inference rules to the context model: for example, the `context:currentLocation` property has been defined as functional property (since we assume that the user can be at only one location at the same time), from which the reasoner can deduce that the two anonymous location resources returned by the different context providers are actually the same and can likewise be merged, as shown in Figure 6.

The context dispatcher distributes this aggregated context description model to all data providers in the system whenever a contextual change is detected. It is then up to each data provider to decide whether to initiate a new replication tasks, and which information from the context description they use for this purpose.

Data Provisioning We have implemented a number of data providers that address different information needs and replicate data from different sources to the mobile device. One data provider uses the Sindice Semantic Web index[43] to retrieve information from FOAF descriptions (which are distributed across the Web) based on the e-mail addresses found in the context description. This includes names, contact and location information, and personal interests of the user's prospective business partners. Also, it includes the social network of the meeting participants and is therefore valuable information for business negotiations as well as smalltalk.

A second data provider retrieves triples about people that are based near the user's current location by looking up resources that are `foaf:based_near` the current and future locations. This information allows the user to increase the effectiveness of their trip by scheduling additional meetings with these persons without additional travel costs.

A third data provider returns additional data from DBpedia about the user's current and future locations, by reusing the GeoNames URI provided by the location context provider (a code excerpt from this data provider is depicted in Figure 7). By doing so, the user is automatically equipped with information about the locations they will visit, and about points of interests in their vicinity.

From an initial analysis, we can expect a significant effect on the amount of potentially interested data that is to be replicated to a mobile device. For instance,

[43] `http://sindice.com`

```
<urn:uuid:baac630a-5cdb-4c79-92e6-6ce3d07419bc>
  a context:Context ;
  context:timestamp "2009-06-16T15:58:22"^^xsd:dateTime ;
  context:previous <urn:uuid:d3ee316b-5704-4893-acb9-df1495c79011> ;
  context:currentLocation [
    geo:lat "48.175443" ; geo:long "16.375493" ;
    geonames:nearby <http://sws.geonames.org/2761369/> .
  ] ;
  context:upcomingEvent [ ncal:attendee
      [ foaf:mbox <mailto:bernhard.schandl@univie.ac.at> ] ,
      [ foaf:mbox <mailto:stefan.zander@univie.ac.at> ] ;
    ncal:location [ a geonames:Feature ; rdfs:label "Munich" ] .
  ] .
<http://sws.geonames.org/2761369/>
  a geonames:Feature ;
  rdfs:label "Vienna"@en .
```

Fig. 6. Aggregated context description model

```
DESCRIBE ?c WHERE {
  { ?c rdfs:label ?l .
    ?l bif:contains "Vienna" .
    ?c rdf:type dbpedia-owl:Place .
  } UNION
  { ?c rdfs:label ?l .
    ?l bif:contains "Salzburg" .
    ?c rdf:type dbpedia-owl:Place .
  } UNION
  { ?c rdfs:label ?l .
    ?l bif:contains "Munich" .
    ?c rdf:type dbpedia-owl:Place .
  }
}
```

Fig. 8. An example SPARQL query produced by the `DBpediaLocationDataProvider`

the public DBpedia data set contains information about around 462,000 places. While no detailed information is available, from the overall size of the data set we can estimate that these places are described by around 88 million triples[44]. By analyzing the user's calendar and querying DBpedia for corresponding resources, this amount of data can be significantly reduced. For instance, if the MobiSem Context Framework detects three locations in the user's calendar, it can convert them into a SPARQL query (cf. Figure 7) and query DBpedia. In case the user's upcoming events within the next 72 hours take place in Vienna, Salzburg, and Munich, the corresponding query (cf. Figure 8) yields around 8,500 triples, which can be handled by common state-of-the-art smartphones (cf. Section 6).

All replicated data is persisted by a storage component that is compatible to the MobiSem Context Framework (cf. Section 4). In the case of Android, RDF graphs are either serialized into flat files (which is very performant but cannot be directly queried) or are stored into a custom triple store that is backed by a SQLite database. Its table layout applies the *normalized triple store* approach; i.e., it stores triples within a `Triple` table that holds references to separate tables for resources and literals. Moreover, it provides lightweight support for *named graphs*; therefore the relational schema contains a separate `Graph` table.

Any application built on top of this framework is now enabled to directly access these data via the MobiSem Data Access API. It could, for instance, iterate over all resources that are typed as `foaf:Person` and provide a list of names and phone numbers, disburdening the user from the need to manually search for these data in case they will miss an appointment and needs to notify the participants. The MobiSem framework entirely hides all context processing steps: an application is presented with a simple view on the triple store which is always populated with context-relevant information.

6. Performance Evaluation of Mobile Semantic Web Platforms

In the resource-limited context of mobile devices, efficient processing of RDF data is crucial. In order to

[44]`http://blog.dbpedia.org/2011/01/17/dbpedia-36-released/`

```
public class DBpediaLocationDataProvider extends AbstractDataProvider
{
  // called when the context description is updated
  @Override
  protected void updateContextImpl() {
    this.currentResourceLabels = new ArrayList<String>();

    // iterate over all geonames features in the context model
    StmtIterator si1 = this.contextModel.listStatements(
      null, RDF.type, GEONAMES.Feature);
    while(si1.hasNext()) {
      Resource featureResource = si1.nextStatement().getSubject();

      // iterate over all properties of these features
      StmtIterator si2 = this.contextModel.listStatements(
        featureResource, null, (Literal)null);
      while(si2.hasNext()) {
        // check if a label property is attached
        Statement s = si2.nextStatement();
        this.currentResourceLabels.add(s.getString());
      }
    }
  }

  // update data from the remote data source
  @Override
  protected void updateDataImpl(Model targetModel) {
    // construct DESCRIBE query for all location resources
    StringBuffer queryBuffer = new StringBuffer();
    queryBuffer.append("DESCRIBE ?concept WHERE { \n");
    for(String featureLabel: this.currentResourceLabels) {
      queryBuffer.append("{ ?c rdfs:label ?l . " +
        "?l bif:contains \"" + featureLabel + \"" . " +
        "?c rdf:type dbpedia-owl:Place . } UNION \n ");
    }
    queryBuffer.append("{} }");

    // send query to DBpedia
    String url = "http://dbpedia.org/sparql?query=" + URLEncoder.encode(
      queryBuffer.toString());
    try {
      // read model into targetModel (for further processing by abstract superclass)
      this.targetModel.read(url, "N-TRIPLE");
    } catch (Exception e) { // error handling
    }
  }
}
```

Fig. 7. Code snippet of `DBpediaLocationDataProvider`, querying DBpedia for data about location resources. `updateContextImpl()` is called by the context dispatcher every time the global context model is updated, while `updateDataImpl()` is called whenever the data provider is requested to actually replicate data from the remote data source.

obtain insights on the processing capabilities of modern mobile platforms, we have carried out a performance evaluation of the three existing mobile RDF frameworks *Androjena*, *μJena*, and *Mobile RDF* (cf. Section 3.2) on three different mobile devices (cf. Table 1). A very important factor of efficient processing is the time needed to create and store an RDF model in-memory, as this is usually the basis for further computation, analysis, inference, or transmission of data over a network. We did not include *RDF on the Go* and *SWIP* in our evaluation since they either exist as an implementation of a specific platform-dependent technology (SWIP) or have been released after our evaluation has been conducted (RDF on the Go).

6.1. Test Environment

The Android HTC G1[45], released in 2008, was one of the first Android devices available on the market and now represents the entry-level device class. It contains a 32-bit Qualcomm MSM7201A RISC CPU that runs with a clock speed of 350 MHz. Tests on this device were performed with the standard memory capacity of 192 MB under the Android operating system version 1.6 update 4.

The Motorola Milestone[46] was released in December 2009 and represents the middle-class of Android capable devices. It runs on a 32-bit TI OMAP3430 Superscalar ARM Cortex-A8 RISC CPU with a nominal clock speed of 600 MHz. On this device, the tests were performed with the standard memory capacity of 256 MB under the operating system version 2.1 update 1.

Finally, we have tested a Samsung Galaxy S I9000[47] smartphone, which was released in Summer 2010. It uses a Qualcomm S5PC111 ARMv7-compatible CPU named "Hummingbird" with a nominal clock speed of max. 1 GHz paired with a PowerVR SGX540 GPU chip. This device uses 512 MB main memory and runs the Android system version 2.2.

We analyzed the creation, parsing, and storage time for RDF models of various sizes, ranging from 10 to 50,000 triples. These models represent the different model sizes that are involved in the context processing and data replication tasks performed by our framework, as described in Section 4. Typically, a single context provider emits very small models in the range of 10 to 100 triples, while a complete context model that has been aggregated from the single context providers may have several hundred to thousand triples in total. Data that are replicated from external sources may in principle be of arbitrary size, therefore we have scaled our tests up to 50,000 triples in a single RDF model.

The distribution of distinct subject, predicate, and object nodes has been estimated based on an analysis of the 2009 Billion Triple Challenge data set [40]. In these data we can observe that typically RDF data sets have a very high number of distinct object values and a low number of distinct predicates, while the number of distinct subjects ranges in between these boundaries. All benchmarks were performed on the mobile devices during regular usage of a device where the usual system processes were running in parallel to our tests.

For each framework, device, and operation, we measured the total amount of time needed in milliseconds. From these measurements we can calculate the standard deviation between different test runs for each size as well as the number of triples that the particular combination of a device and a framework is able to process within one second.

In order to eliminate technological differences between SD cards in terms of access times as well as read-/write performance, we first copied data replicas from the SD card to the internal non-volatile memory (ROM) of a device from where they are then parsed and transformed into a working in-memory model.

Before each benchmark was initiated, the device had been restarted to ensure identical run-time conditions. At the end of each benchmark, all files and data that had been created during a test run were deleted and the test environment was reseted to avert an influence on consecutive benchmarks.

6.2. Results

Figures 9, 10, and 11 depict the results of our measurements (detailed numbers can be found in the appendix) for each analyzed device[48].

[45]`http://www.htc.com/www/product/g1/specification.html`

[46]`http://www.motorola.com/Consumers/XW-EN/Consumer-Products-and-Services/Mobile-Phones/ci.Motorola-MILESTONE-XW-EN.alt`

[47]`http://pdadb.net/index.php?m=specs&id=2298&c=samsung_gt-i9000_galaxy_s_16gb`

[48]'RDF/XML', 'N3', and 'N-TRIPLE' refer to the different serialization formats supported by the Androjena framework. For readability issues we excluded the framework's name 'Androjena' and just referred to the respective format for all parsing and storage figures.

Table 1
Overview of the Android Devices' Specification

	HTC G1	Motorola Milestone	Samsung Galaxy S I9000
Processor	Qualcomm MSM7201A^{TM}	TI OMAP3430 ARM Cortex A8	Qualcomm S5PC111 (ARMv7-comp.)
Clock speed in MHz	350 MHz	600 MHz	1 GHz
Memory Capacity (RAM)	192 MB	256 MB	512 MB
OS Version	Android 1.6-update4	Android 2.1-update1	Android 2.2
Release	09/2008	12/2009	06/2010

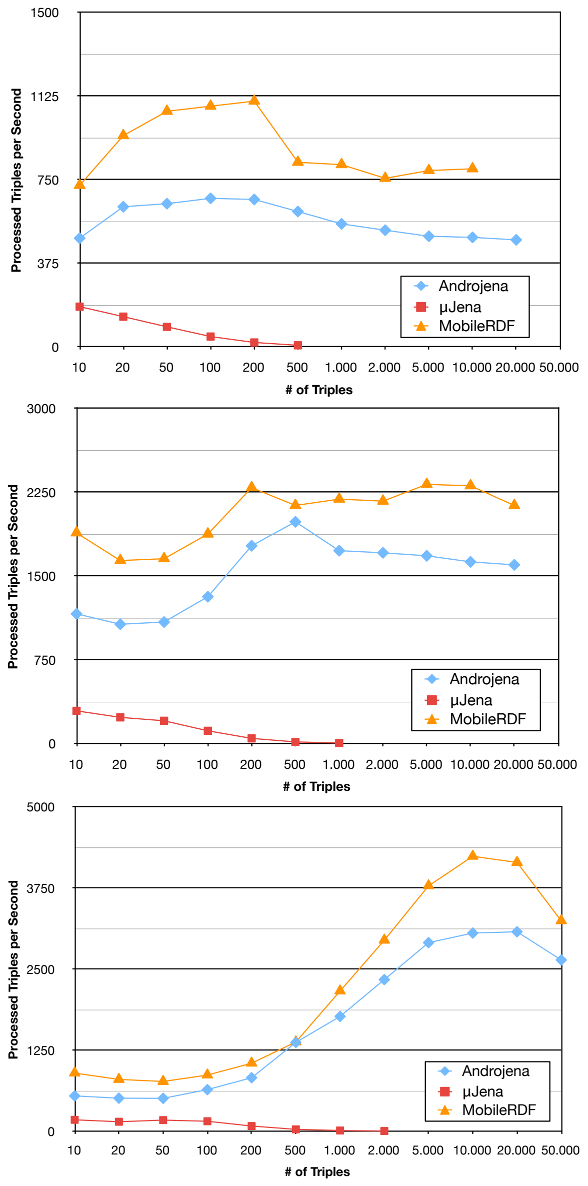

Fig. 9. Construction of RDF graphs (Android HTC G1, Motorola Milestone, Samsung Galaxy S I9000)

Constructing In-memory RDF Graphs. When creating in-memory RDF graphs of certain sizes, we can observe a similar behavior on all three tested platforms. Androjena and Mobile RDF exhibit very similar results, namely, a nearly constant processing time per triple, even with increasing model size. Although processing times of mobile RDF frameworks vary considerably across small context descriptions with sizes smaller than 500 triples (up to factor 10 on the Samsung Galaxy S I9000 using Mobile RDF for processing a model containing 100 triples), processing times normalize for models of size greater or equal than 1000 triples on the two frameworks. In general, we can observe that Androjena and Mobile RDF are able to handle RDF graphs containing 20,000 or more triples, although the limiting factor is the device's memory capacity.

Additionally, the total execution time (in ms) for Androjena and Mobile RDF scales almost linearly with the size of context descriptions. The performance of μJena, on the contrary, decreases significantly with increasing model sizes, leading to very low processing times with models larger than 100 triples. μJena tests with more than 2,000 triples failed on all devices, making it basically unsuitable for the processing of voluminous RDF data.

Processing speed of Androjena ranges between 480 and 680 triples per second on an Android HTC G1, and 1000 and 2000 triples per second on a Motorola Milestone. Interestingly, on the Samsung Galaxy we can observe that the performance increases when models with more than 200 triples are processed. The performance of μJena decreases with increasing model size on all three devices. Mobile RDF exhibits a similar performance behavior compared to Androjena where a significant increase in triples per second values on a Samsung Galaxy can be observed for models with more than 500 triples. In general, Mobile RDF has shown to be the most performant framework w.r.t. the amount of triples processed per second on all tested devices.

When comparing the different devices, we can observe the expected behavior that the Android HTC G1 exposes the weakest results due to its slow CPU and small main memory, leading to memory problems when creating models with 20,000 or more triples. The other devices expose a better performance, making them more suitable for processing larger volumes of RDF data. Only the Samsung Galaxy I S9000 was able to handle a model of 50,000 triples; on the other devices tests with this model size failed with "out of memory" errors.

Parsing RDF Graphs. Androjena scales reasonably well with available processing power and yields best parsing results in terms of triples per second ratios with RDF graphs containing more than 200-500 triples. However, we were not able to notice a remarkable difference between the different serialization formats on newer mobile device with graphs smaller than 100 triples, i.e., significant differences in benchmark results among different serialization formats can first be noticed on newer mobile devices for graphs with more than 100 triples.

μJena yields best results with very small RDF graphs containing less than 20 triples. However, we were able to observe a dramatic decrease in parsing performance with models containing more than 20 to 50 triples, which renders μJena inappropriate for processing larger data replicas.

MobileRDF also scales reasonably well with available processing power and turns out to be the fastest RDF framework in terms of parsing performance, especially for larger RDF graphs with more than 100 to 200 triples. This behavior was more distinctive on less powerful devices such as the HTC G1 or the Motorola Milestone but dissolved on recent, more powerful devices such as the Samsung Galaxy[49]. The best performance results could be measured with RDF graphs containing around 5,000 to 10,000 triples on the Samsung Galaxy S I9000.

In summary, the parsing benchmark exhibits similar behavior on all three devices revealing that MobileRDF yields the fastest parsing performance followed by Androjena and μJena. Additionally, Androjena and MobileRDF scale reasonably well with available processing power. Considering the different serialization formats supported by Androjena, the best parsing results were measured with N-Triple serialized graphs followed by N3 and RDF/XML.

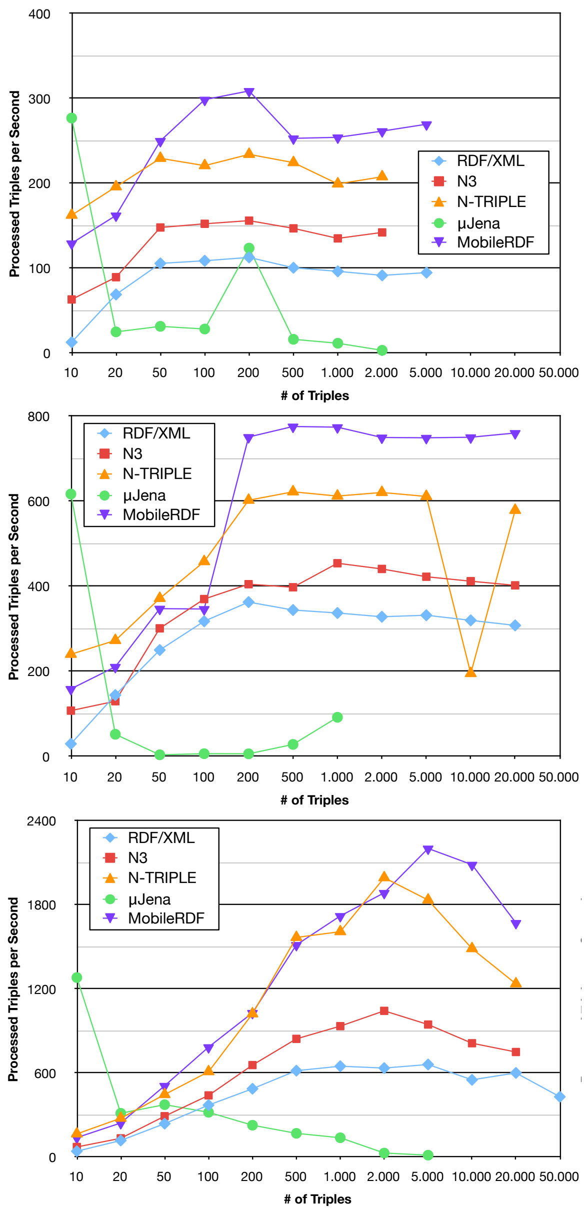

Fig. 10. Parsing of RDF graphs (Android HTC G1, Motorola Milestone, Samsung Galaxy S I9000)

Serializing RDF Graphs. Storage times of all frameworks are relatively linear with the amount of triples to be stored, i.e, we were able to observe a linear scaling between storage run-times and the amount of triples to be saved on all three frameworks and on each device. However, no significant difference w.r.t. the file sizes between the different frameworks and serializa-

[49]We verified this assertion using a Dell Streak smartphone that also runs an ARM Cortex A8 CPU clocked at 1 GHz, where we were able to ascertain a similar behavior.

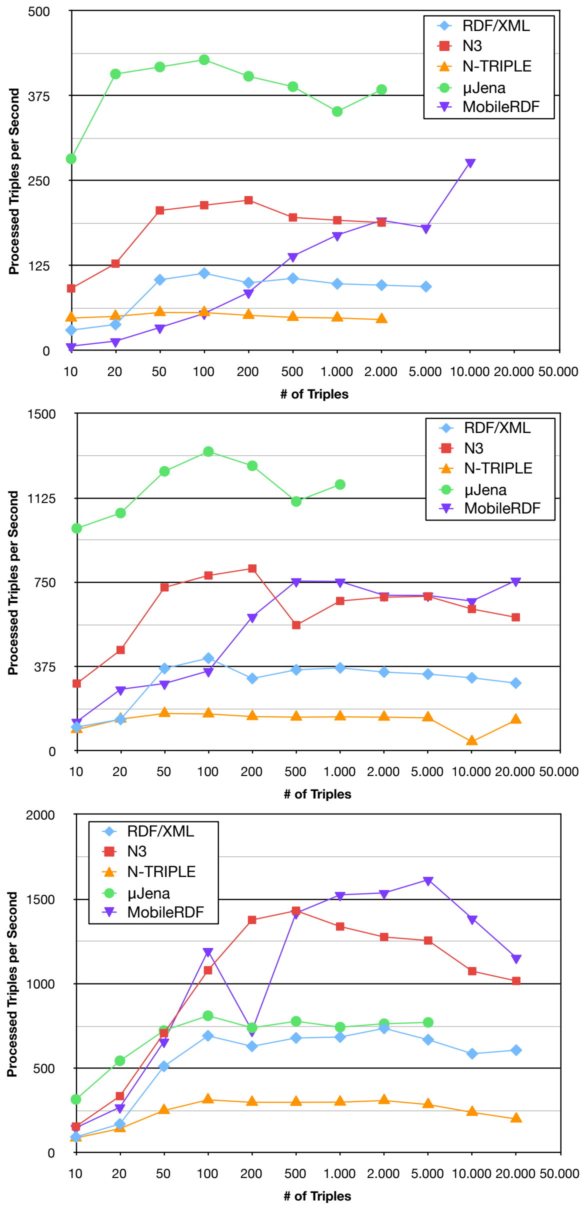

Fig. 11. Serialization of RDF graphs (Android HTC G1, Motorola Milestone, Samsung Galaxy S I9000)

tion formats could be found, which indicates that storage algorithms do not make use of e.g. *QNames*. File sizes of the serialized data replicas are rather similar among all frameworks and devices.

Androjena's saving performance scales reasonably well w.r.t. available processing power where best results could be achieved on the Samsung Galaxy; total storage times were seven times faster compared to those measured on the HTC G1 for all serialization formats. Serializing RDF graphs in the N3 format yields the best triples per second ratio, followed by RDF/XML and N-Triple. The best storage performance results could be measured with graphs of sizes between 100 and 2,000 triples irrespectively of the serialization format and device.

Although by far the least competitive framework in terms of creation and parsing performance, μJena yields the best storage performance on the HTC G1 and the Motorola Milestone. However, this behavior disappeared on the Samsung Galaxy and similar devices such as the Dell Streak[50] where MobileRDF and N3-serialized graphs using the Androjena framework showed the best results[51]. Interestingly, the best storage performance results could be measured on the Motorola Milestone that exceeds the results of the other two devices considerably.

Storage performance of MobileRDF scales with available processing power for RDF graphs with triple sizes greater than 200 to 500. Best results could be measured for graph sizes between 500 and 5,000 triples where the triples per second ratio differs by the factor 8 between the Samsung Galaxy and the HTC G1.

In summary we can see that modern mobile devices, in combination with recent RDF frameworks that are optimized for mobile devices, can without hesitation be used as the basis for Semantic Web applications on mobile devices. Although the behavior of the tested frameworks differ across machines, we can observe certain trends regarding the applicability of RDF frameworks for specific purposes. In further work, we aim to analyze the behavior of these devices w.r.t. modification and deletion operations, as well as querying and inference over RDF data, depending on the availability of such implementations.

7. Conclusions and Future Work

The notion of context and context awareness are key factors in providing a selective RDF-based data replication infrastructure for mobile devices. We have outlined that traditional replication strategies do not hold in mobile scenarios for several reasons. They should be improved by considering current and future users' information needs as well as the different contexts they are operating in, thus replicating only selected subsets

[50]`http://www.dell.com/us/p/mobile-streak/pd`
[51]We tested the storage performance also on a Dell Streak smartphone, which exhibits similar processing power and clock speed

of the base data. We therefore adopted the notion of context and context awareness and synthesized it with semantic technologies since they provide the necessary flexibility and expressivity for context-dependent RDF-based data replication on mobile devices. Our framework employs a loose coupling between context acquisition and data provisioning components, gained by applying semantic technologies (data models, vocabularies, inference) to interpret and process context information. We implemented an example scenario in which personal information from Linked Data sources is replicated based on the user's current location and upcoming appointments. Our performance evaluation has shown that the performance of current RDF processing frameworks, deployed on state-of-the-art mobile devices, is acceptable for the processing of RDF models of several thousand triples.

Although we have demonstrated that semantic technologies can provide substantial contributions in realizing a mobile context-aware infrastructure for RDF(-based) data replication, there are still some open issues that need to be addressed in future research: the integration of dynamically discovered context sources is a challenge most context-management frameworks face, especially in ubiquitous environments. We therefore plan to investigate additional methods for dynamic context source discovery and integration as well as heuristics for transforming sensorial data into qualitative context descriptions. We further plan to consider re-using functionality already built into the framework (namely, the acquisition and combination of contextual information from varying sources) to decide upon the optimal time for initiating replication tasks. Currently, our framework does not include feedback loops that would allow for adjusting context acquisition and aggregation tasks according to data provisioning needs, and it lacks advanced reasoning capabilities, which we plan to implement in the near future.

An approach as proposed by [29] to integrate formal rule languages like SWRL [26] into context processing tasks would allow for the user- and application-driven specification of aggregation, reasoning, and consolidation rules for collected and augmenting contextual data. Additionally, context processing could be complemented with machine learning techniques for detecting usage patters, as proposed by [6,7]. However, a context framework by itself can be made context-aware to adapt its processing rules and policies according to specific circumstances, for instance to reduce replication cycles in case of low battery etc. We plan to address these issues in future work.

Acknowledgements This work has been funded by the FIT-IT grant 815133 from Austrian Federal Ministry of Transport, Innovation, and Technology. We would also like to thank our reviewers Jérôme Euzenat and Martin Raubal for their valuable feedback.

References

[1] C. B. Anagnostopoulos, Y. Ntarladimas, and S. Hadjiefthymiades. Situational computing: An innovative architecture with imprecise reasoning. *J. Syst. Softw.*, 80(12):1993–2014, 2007.

[2] C. Becker and C. Bizer. DBpedia Mobile: A Location-Enabled Linked Data Browser. In *Workshop on Linked Data on the Web (LDOW2008)*, 2008.

[3] A. Beloued, J.-M. Gilliot, M.-T. Segarra, and F. André. Dynamic Data Replication and Consistency in Mobile Environments. In *Proc. of the 2nd international doctoral symposium on Middleware*, pages 1–5, New York, NY, USA, 2005. ACM.

[4] G. Biegel and V. Cahill. A Framework for Developing Mobile, Context-aware Applications. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications*, pages 361–365, March 2004.

[5] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data: Principles and State of the Art. In *17th World Wide Web Conference (WWW2008), W3C track*, April 2008.

[6] S. Boehm, J. Koolwaaij, M. Luther, B. Souville, M. Wagner, and M. Wibbels. Introducing IYOUIT. *The Semantic Web - ISWC 2008*, pages 804–817, 2008.

[7] S. Böhm, J. Koolwaaij, M. Luther, B. Souville, M. Wagner, and M. Wibbels. Iyouit - share, life, blog, play. In C. Bizer and A. Joshi, editors, *International Semantic Web Conference (Posters & Demos)*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[8] C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca. A Data-oriented Survey of Context Models. *SIGMOD Rec.*, 36(4):19–26, 2007.

[9] D. Brickley and L. Miller. The Friend Of A Friend (FOAF) vocabulary specification, November 2007. `http://xmlns.com/foaf/spec/`.

[10] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs. *Journal of Web Semantics*, 3(4):247–267, 2005.

[11] A. Carton, S. Clarke, A. Senart, and V. Cahill. Aspect-oriented model-driven development for mobile context-aware computing. In *Proc. of the 1st Int'l Workshop on SW Engineering for Pervasive Comp. Applications, Systems, and Environments*, page 5, Washington, DC, USA, 2007. IEEE Computer Society.

[12] J. Coutaz, J. L. Crowley, S. Dobson, and D. Garlan. Context is Key. *Communications of the ACM - Special Issue: The disappearing computer*, 48(3):49–53, 2005.

[13] J. David and J. Euzenat. Linked data from your pocket: The android RDFContentProvider. In *9th International Semantic Web Conference (ISWC2010)*, Nov. 2010.

[14] A. K. Dey. Understanding and Using Context. *Personal Ubiquitous Comput.*, 5(1):4–7, 2001.

[15] P. Dourish. What We Talk About When We Talk About Context. *Personal Ubiquitous Comput.*, 8(1):19–30, 2004.

[16] J. Euzenat. Alignment Infrastructure for Ontology Mediation and Other Applications. In *MEDIATE2005*, volume 168, pages 81–95, 2005.

[17] J. Euzenat, J. Pierson, and F. Ramparany. Dynamic Context Management for Pervasive Applications. *The Knowledge Engineering Review*, 23(1):21–49, 2008.

[18] G. H. Forman and J. Zahorjan. The Challenges of Mobile Computing. *Computer*, 27(4):38–47, 1994.

[19] J. D. Gehrke. Evaluating situation awareness of autonomous systems. In *PerMIS '08: Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, pages 206–213, New York, NY, USA, 2008. ACM.

[20] H. Gellersen, G. Kortuem, A. Schmidt, and M. Beigl. Physical prototyping with smart-its. *IEEE Pervasive Computing*, 3(3):74–82, 2004.

[21] H. W. Gellersen, A. Schmidt, and M. Beigl. Multi-sensor Context-awareness in Mobile Devices and Smart Artifacts. *Mobile Networks and App's*, 7(5), 2002.

[22] F. Gomez and C. Segami. Classification-based reasoning. *Systems, Man and Cybernetics, IEEE Transactions on*, 21(3):644 –659, 6 1991.

[23] M. Greaves. Semantic Web 2.0. *IEEE Intelligent Systems*, 22(2):94–96, 2007.

[24] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for Distributed Context-Aware Systems. In *On the Move to Meaningful Internet Systems 2005*. Springer, 2005.

[25] H. Höpfner and K.-U. Sattler. Semantic Replication in Mobile Federated Information Systems. In *Proc.of the Fifth Int'l Workshop on Engineering Federated Information Systems (EFIS), Coventry, UK*, 2003.

[26] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosofand, and M. Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML.* W3C, May 2004. Last access on Dez 2008 at: http://www.w3.org/Submission/SWRL/.

[27] Y. Huang, P. Sistla, and O. Wolfson. Data Replication for Mobile Computers. In *Proc. of the ACM SIGMOD international conference on Management of data*, pages 13–24, New York, NY, USA, 1994. ACM.

[28] C. Huebscher and A. McCann. An Adaptive Middleware Framework for Context-aware Applications. *Personal Ubiquitous Comput.*, 10(1):12–20, 2005.

[29] C. Kessler, M. Raubal, and C. Wosniok. Semantic rules for context-aware geographical information retrieval. In *Proceedings of the 4th European conference on Smart sensing and context*, EuroSSC'09, pages 77–92, Berlin, Heidelberg, 2009. Springer-Verlag.

[30] J. Krogstie, K. Lyytinen, A. Opdahl, B. Pernici, K. Siau, and K. Smolander. Research areas and challenges for mobile information systems. *International Journal of Mobile Communications*, 2(3):220–234, 2004.

[31] D. Le-Phuoc, J. X. Parreira, V. Reynolds, and M. Hauswirth. RDF on the go: An RDF storage and query processor for mobile devices. In *9th International Semantic Web Conference (ISWC2010)*, Nov. 2010.

[32] M. Luther, S. Böhm, M. Wagner, and J. Koolwaaij. Enhanced Presence Tracking for Mobile Applications. In *ISWC'05 Demo Track*, 2005.

[33] M. Luther, Y. Fukazawa, M. Wagner, and S. Kurakake. Situational reasoning for task-oriented mobile service recommendation. *The Knowledge Engineering Review*, 23(1):7–19, 2008.

[34] K. Mihalic and M. Tscheligi. 'Divert: Mother-in-law': Representing and Evaluating Social Context on Mobile Devices. In *MobileHCI '07: 9th int. conf. on Human computer interaction with mobile devices & services*, pages 257–264. ACM, 2007.

[35] P. Pawar, A. T. van Halteren, and K. Sheikh. Enabling context-aware computing for the nomadic mobile user: A service oriented and quality driven approach. In *IEEE Wireless Communications and Networking Conference WCNC 2007*, pages 2531–2536. IEEE Communication Society, March 2007.

[36] P. Prekop and M. Burnett. Activities, context and ubiquitous computing. *Computer Communications*, 26(11):1168 – 1176, 2003. Ubiquitous Computing.

[37] D. Raptis, N. Tselios, and N. Avouris. Context-based Design of Mobile Applications for Museums: A Survey of Existing Practices. In *MobileHCI '05: 7th int. conf. on Human comp. interaction w. mobile devices & services*. ACM, 2005.

[38] M. Raubal and I. Panov. A formal model for mobile map adaptation. In G. Gartner and K. Rehrl, editors, *Location Based Services and TeleCartography II*, Lecture Notes in Geoinformation and Cartography, pages 11–34. Springer Berlin Heidelberg, 2009. 10.1007/978-3-540-87393-8_2.

[39] V. Reynolds, M. Hausenblas, A. Polleres, M. Hauswirth, and V. Hegde. Exploiting linked open data for mobile augmented reality. In *W3C Workshop: Augmented Reality on the Web*, June 2010.

[40] B. Schandl. Replication and Versioning of Partial RDF Graphs. In *Proceedings of the 7th European Semantic Web Conference (ESWC 2010)*, 2010.

[41] A. Schmidt, M. Beigl, and H.-W. Gellersen. There is More to Context than Location. *Computers and Graphics*, 23:893–901, 1998.

[42] T. Springer, P. Wustmann, I. Braun, W. Dargie, and M. Berger. A comprehensive approach for situation-awareness based on sensing and reasoning about context. In *UIC '08: Proceedings of the 5th international conference on Ubiquitous Intelligence and Computing*, pages 143–157, Berlin, Heidelberg, 2008. Springer-Verlag.

[43] H.-S. Teo. An Activity-driven Model for Context-awareness in Mobile Computing. In *MobileHCI '08: 10th int. conf. on Human Computer Interaction w. mobile devices & services*, pages 545–546, New York, NY, USA, 2008. ACM.

[44] K. Thirunarayan, C. A. Henson, and A. P. Sheth. Situation awareness via abductive reasoning from semantic sensor data: A preliminary report. *Collaborative Technologies and Systems, International Symposium on*, 0:111–118, 2009.

[45] C. Weiss, A. Bernstein, and S. Boccuzzo. i-MoCo: Mobile Conference Guide - Storing and querying huge amounts of Semantic Web data on the iPhone/iPod Touch, October 2008.

[46] M. Wilson, A. Russell, D. A. Smith, A. Owens, and m. c. Schraefel. mSpace Mobile: A Mobile Application for the Semantic Web. *End User Semantic Web Workshop, ISWC2005*, page 11, 2005.

[47] O. Wolfson, S. Jajodia, and Y. Huang. An Adaptive Data Replication Algorithm. *ACM Trans. Database Syst.*, 22(2):255–314, 1997.

[48] S. Y. Wu and K.-T. Wu. Dynamic Data Management for Location Based Services in Mobile Environments. In *7th International Database Engineering and Applications Symposium (IDEAS)*, pages 180–191. IEEE Computer Society, 2003.

[49] S. Zander and B. Schandl. A Framework for Context-driven RDF Data Replication on Mobile Devices. In *Proceedings of the 6th International Conference on Semantic Systems (I-Semantics)*, Graz, Austria, 2010.

Table 2

Construction of RDF graphs (Android HTC G1, Motorola Milestone, Samsung Galaxy S I9000)

	Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000
Androjena	Execution Time (ms)	20.5	31.8	77.8	150.0	302.3	823.0	1,811.4	3,819.9	10,069.7	20,343.0	41,627.3	DNF
	Standard Deviation	6.40	1.40	2.04	3.46	3.46	63.86	12.47	104.14	78.39	159.39	303.21	DNF
	Triples per second	487	628	642	666	661	607	552	523	496	491	480	DNF
μJena	Execution Time (ms)	55.3	147.0	553.3	2,138.9	9,914.4	67,168.3	DNF	DNF	DNF	DNF	DNF	DNF
	Standard Deviation	3.27	10.74	23.38	75.48	308.93	5,054.04	DNF	DNF	DNF	DNF	DNF	DNF
	Triples per second	181	136	90	47	20	7	DNF	DNF	DNF	DNF	DNF	DNF
Mobile RDF	Execution Time (ms)	13.8	21.1	47.3	92.6	181.4	603.7	1,222.9	2,644.2	6,318.2	12.515.4	DNF	DNF
	Standard Deviation	7.47	0.74	1.83	2.12	2.88	4.76	7.89	59.62	73.28	174.13	DNF	DNF
	Triples per second	724	947	1,057	1,079	1,102	828	817	756	791	799	DNF	DNF

	Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000
Androjena	Execution Time (ms)	8.6	18.7	45.9	76.0	112.9	251.8	578.8	1,170.6	2,973.1	6,144.3	12,494.2	DNF
	Standard Deviation	0.70	3.02	3.03	7.04	2.85	6.97	7.27	21.36	41.75	23.45	74.18	DNF
	Triples per second	1,163	1,070	1,089	1,316	1,771	1,986	1,728	1,709	1,682	1,628	1,601	DNF
μJena	Execution Time (ms)	33.9	84.5	242.5	858.5	4,107.8	29,055.2	143,648.5	DNF	DNF	DNF	DNF	DNF
	Standard Deviation	8.50	5.25	15.47	22.96	116.82	2,772.42	12,689.76	DNF	DNF	DNF	DNF	DNF
	Triples per second	295	237	206	116	49	17	7	DNF	DNF	DNF	DNF	DNF
Mobile RDF	Execution Time (ms)	5.3	12.2	30.2	53.3	87.3	234.5	456.9	921.4	2,154.7	4,332.6	9,383.3	DNF
	Standard Deviation	1.57	1.14	4.89	2.21	6.02	6.22	10.05	14.79	11.98	81.84	110.98	DNF
	Triples per second	1,887	1,639	1,656	1,876	2,291	2,132	2,189	2,171	2,321	2,308	2,131	DNF

	Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000
Androjena	Execution Time (ms)	18.2	38.9	97.7	154.8	241.2	364.1	563.7	854.4	1,718.1	3,270.3	6,498.8	18,913.7
	Standard Deviation	6.25	6.31	17.41	34.78	15.25	4.07	15.11	28.18	83.38	59.84	38.06	713.87
	Triples per second	549	514	512	646	829	1,373	1,774	2,341	2,910	3,058	3,077	2,644
μJena	Execution Time (ms)	55.3	132.3	283.4	630.0	2,345.5	14,718.9	61,784.5	236,003.4	DNF	DNF	DNF	DNF
	Standard Deviation	26.25	16.59	20.13	36.28	101.60	1,261.50	14,590.10	63,201.29	DNF	DNF	DNF	DNF
	Triples per second	181	151	176	159	85	34	16	8	DNF	DNF	DNF	DNF
Mobile RDF	Execution Time (ms)	11.1	24.9	64.8	114.9	189.9	362.4	461.9	678.5	1,320.9	2,358.1	4,824.4	15,404.6
	Standard Deviation	3.96	4.23	3.29	7.49	24.04	22.70	33.52	18.34	90.27	39.50	51.57	667.82
	Triples per second	901	803	772	870	1,053	1,380	2,165	2,948	3,785	4,241	4,146	3,246

Table 3

Parsing performance of data replicas (Android HTC G1, Motorola Milestone, Samsung Galaxy S I9000)

	Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000
RDF/XML	Execution Time (ms)	768.3	288.9	472.3	917.8	1,771.2	4,964.6	10,360.7	21,795.6	52,643.0	DNF	DNF	DNF
	Standard Deviation	1,906.75	2.33	2.98	9.33	9.62	96.79	376.72	2,430.40	460.09	DNF	DNF	DNF
	Triples per second	13.02	69.23	105.86	108.96	112.92	100.71	96.52	91.76	94.98	DNF	DNF	DNF
N3	Execution Time (ms)	157.7	223.2	337.5	655.8	1,280.3	3,398.7	7,390.5	14,051.3	DNF	DNF	DNF	DNF
	Standard Deviation	60.15	2.39	5.13	10.12	28.39	35.67	189.19	341.87	DNF	DNF	DNF	DNF
	Triples per second	63.41	89.61	148.15	152.49	156.21	147.12	135.31	142.34	DNF	DNF	DNF	DNF
N-Triple	Execution Time (ms)	61.5	102.0	217.8	452.6	853.8	2,226.9	5,013.8	9,623.0	DNF	DNF	DNF	DNF
	Standard Deviation	12.03	2.40	7.74	30.16	26.54	58.37	143.10	665.70	DNF	DNF	DNF	DNF
	Triples per second	162.60	196.08	229.57	220.95	234.25	224.53	199.45	207.84	DNF	DNF	DNF	DNF
μJena	Execution Time (ms)	36.1	792.6	1,576.0	3,496.8	1,614.6	30,357.8	84,543.4	568,003.4	DNF	DNF	DNF	DNF
	Standard Deviation	8.08	371.51	569.33	874.84	234.65	5,556.99	12,624.49	119,008.18	DNF	DNF	DNF	DNF
	Triples per second	277.01	25.23	31.73	28.60	123.87	16.47	11.83	3.52	DNF	DNF	DNF	DNF
MobileRDF	Execution Time (ms)	77.2	123.2	199.9	334.8	647.4	1,975.3	3,932.4	7,644.7	18,536.5	DNF	DNF	DNF
	Standard Deviation	35.09	34.66	73.33	31.61	85.85	108.41	304.45	219.08	318.80	DNF	DNF	DNF
	Triples per second	129.53	162.34	250.13	298.69	308.93	253.13	254.30	261.62	269.74	DNF	DNF	DNF

	Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000
RDF/XML	Execution Time (ms)	335.5	138.4	199.7	314.4	550.8	1,451.6	2,963.1	6,086.4	15,050.4	31,231.9	64,873.3	DNF
	Standard Deviation	747.50	9.00	35.50	5.90	15.40	51.90	41.50	877.70	95.70	168.90	1,021.00	DNF
	Triples per second	29.81	144.51	250.38	318.07	363.11	344.45	337.48	328.60	332.22	320.19	308.29	DNF
N3	Execution Time (ms)	92.7	153.8	165.9	270.1	493.6	1,256.3	2,200.4	4,534.6	11,828.1	24,257.7	49,693.6	DNF
	Standard Deviation	26.40	91.90	24.10	19.90	46.90	77.30	60.10	31.40	56.50	137.50	131.30	DNF
	Triples per second	107.87	130.04	301.39	370.23	405.19	397.99	454.46	441.05	422.72	412.24	402.47	DNF
N-Triple	Execution Time (ms)	41.6	73.3	134.5	218.2	331.9	803.3	1,632.9	3,221.3	8,176.0	51,243.0	34,548.4	DNF
	Standard Deviation	4.40	0.80	12.50	15.80	26.70	35.80	47.90	62.30	79.70	40,861.10	535.20	DNF
	Triples per second	240.38	272.85	371.75	458.30	602.59	622.43	612.41	620.87	611.55	195.15	578.90	DNF
μJena	Execution Time (ms)	16.2	382.6	12,491.7	15,209.5	31,491.5	17,595.4	10,845.6	DNF	DNF	DNF	DNF	DNF
	Standard Deviation	8.02	137.63	24,653.70	23,229.44	41,285.81	1,996.35	1,015.30	DNF	DNF	DNF	DNF	DNF
	Triples per second	617.28	52.27	4.00	6.57	6.35	28.42	92.20	DNF	DNF	DNF	DNF	DNF
Mobile RDF	Execution Time (ms)	62.9	94.8	143.8	288.3	266.2	644.3	1,291.2	2,666.0	6,671.4	13,320.3	26,291.0	DNF
	Standard Deviation	26.41	28.20	33.35	95.12	27.34	22.68	58.29	95.79	93.97	109.76	279.99	DNF
	Triples per second	158.98	210.97	347.71	346.86	751.31	776.04	774.47	750.19	749.47	750.73	760,72	DNF

	Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000
RDF/XML	Execution Time (ms)	235.0	168.0	209.0	269.0	409.6	810.0	1,540.3	3,144.7	7,559.3	18,116.6	33,256.2	115,976.8
	Standard Deviation	303.58	20.94	22.08	23.28	116.81	60.37	102.61	535.18	79.75	112.81	2,579.66	12,876.26
	Triples per second	42.55	119.05	239.23	371.75	488.28	617.28	649.22	635.99	661.44	551.98	601.39	431.12
N3	Execution Time (ms)	137.2	148.5	170.9	226.8	304.3	592.0	1,069.3	1,914.9	5,281.6	12,295.6	26,646.0	DNF
	Standard Deviation	59.55	19.91	22.54	22.02	25.54	48.41	91.69	114.51	90.45	256.86	1,917.43	DNF
	Triples per second	72.89	134.68	292.57	440.92	657.25	844.59	935.19	1,044.44	946.68	813.30	750.58	DNF
N-Triple	Execution Time (ms)	60.2	72.0	111.9	163.9	195.5	318.8	621.3	1,002.7	2,724.1	6,719.1	16,166.0	DNF
	Standard Deviation	27.70	12.10	28.63	20.51	23.61	39.85	200.39	31.53	109.69	556.76	860.41	DNF
	Triples per second	166.11	277.78	446.83	610.13	1,023.02	1,568.38	1,609.53	1,994.61	1,835.47	1,488.29	1,237.16	DNF
μJena	Execution Time (ms)	7.8	63.8	133.4	312.3	877.2	2,940.1	7,216.2	67,689.3	352,660.7	DNF	DNF	DNF
	Standard Deviation	13.46	21.15	39.14	46.80	99.00	402.41	955.89	17,684.88	63,039.47	DNF	DNF	DNF
	Triples per second	1,282.05	313.48	374.81	320.20	228.00	170.06	138.58	29.55	14.18	DNF	DNF	DNF
MobileRDF	Execution Time (ms)	71.6	82.0	98.1	127.3	194.1	330.0	580.8	1,060.0	2,269.2	4,789.7	11,959.3	DNF
	Standard Deviation	44.51	13.77	24.69	20.32	38.81	60.63	75.29	206.26	122.68	156.18	633.84	DNF
	Triples per second	139.66	243.90	509.68	785.55	1,030.40	1,515.15	1,721.76	1,886.79	2,203.42	2,087.81	1,672.34	DNF

Table 4

Storage performance of data replicas (Android HTC G1, Motorola Milestone, Samsung Galaxy S I9000)

	Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000
RDF/XML	Execution Time (ms)	329.2	517.8	478.7	876.8	1999.4	4694.9	10148	20713.5	52946.5	DNF	DNF	DNF
	Standard Deviation	114.95	11.38	5.58	20.85	29.53	100.08	274.74	740.17	448.72	DNF	DNF	DNF
	Triples per second	30.38	38.62	104.45	114.05	100.03	106.50	98.54	96.56	94.43	DNF	DNF	DNF
N3	Execution Time (ms)	109	156	242.1	466.8	902.6	2,548.3	5,205.2	10,598.4	DNF	DNF	DNF	DNF
	Standard Deviation	50.95	1.15	2.77	14.90	10.84	24.84	51.53	264.18	DNF	DNF	DNF	DNF
	Triples per second	91.74	128.21	206.53	214.22	221.58	196.21	192.12	188.71	DNF	DNF	DNF	DNF
N-Triple	Execution Time (ms)	207.7	395.3	888.7	1,779.8	3,837	10,177.3	20,738.9	43,593.9	DNF	DNF	DNF	DNF
	Standard Deviation	10.58	18.00	18.48	25.38	114.07	147.46	776.31	1,427.98	DNF	DNF	DNF	DNF
	Triples per second	48.15	50.59	56.26	56.19	52.12	49.13	48.22	45.88	DNF	DNF	DNF	DNF
μJena	Execution Time (ms)	35.4	49.1	119.7	233.5	495.1	1,286.3	2,839.5	5,202.1	DNF	DNF	DNF	DNF
	Standard Deviation	21.56	0.88	14.53	28.52	93.90	21.10	171.08	186.73	DNF	DNF	DNF	DNF
	Triples per second	282,49	407,33	417,71	428,27	403,96	388,71	352,17	384,46	DNF	DNF	DNF	DNF
MobileRDF	Execution Time (ms)	1,462.2	1,411.8	1,458.2	1,821.2	2,329.4	3,583.9	5,868	10,420.4	27,577.9	35,948.7	DNF	DNF
	Standard Deviation	332.58	119.48	122.19	132.63	148.70	137.66	148.88	338.46	3,699.14	7,672.72	DNF	DNF
	Triples per second	6.84	14.17	34.29	54.91	85.86	139.51	170.42	191.93	181.30	278.17	DNF	DNF

	Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000
RDF/XML	Execution Time (ms)	93.6	141.8	136.1	242.1	619.4	1,383.6	2,703.3	5,694.2	14,616.3	30,694.6	66,160.3	DNF
	Standard Deviation	46.19	6.76	11.93	1.85	36.82	75.19	62.55	78.90	133.99	236.04	988.00	DNF
	Triples per second	106.84	141.04	367.38	413.05	322.89	361.38	369.92	351.23	342.08	325.79	302.30	DNF
N3	Execution Time (ms)	33.3	44.5	68.7	128.1	246.4	892.8	1,498.5	2,923.6	7,268.5	15,833.2	33,629.1	DNF
	Standard Deviation	7.42	0.53	3.02	5.26	7.75	207.98	43.81	60.84	78.71	1,107.54	425.26	DNF
	Triples per second	300.30	449.44	727.80	780.64	811.69	560.04	667.33	684.09	687.90	631.58	594.72	DNF
N-Triple	Execution Time (ms)	104.5	140.6	298.9	605.8	1,303.5	3,313.0	6,570.7	13,266.6	33,809.2	238,095.7	145,497.5	DNF
	Standard Deviation	20.51	11.90	11.52	35.02	52.13	59.96	50.29	84.66	465.27	160,267.56	1,818.58	DNF
	Triples per second	95.69	142.25	167.28	165.07	153.43	150.92	152.19	150.75	147.89	42.00	137.46	DNF
μJena	Execution Time (ms)	10.1	18.9	40.2	75.1	157.7	450.6	843.8	DNF	DNF	DNF	DNF	DNF
	Standard Deviation	0.32	3.87	4.87	0.74	10.88	22.38	24.30	DNF	DNF	DNF	DNF	DNF
	Triples per second	990.10	1,058.20	1,243.78	1,331.56	1,268.23	1,109.63	1,185.11	DNF	DNF	DNF	DNF	DNF
Mobile RDF	Execution Time (ms)	76.4	72.8	166.4	280.7	334.8	661.5	1,326.4	2,887.8	7,227.2	14,993.8	26,380.2	DNF
	Standard Deviation	29.00	13.36	33.06	145.99	104.13	26.90	55.08	61.26	172.31	337.11	5,592.70	DNF
	Triples per second	130.89	274.73	300.48	356.25	597.37	755.86	753.92	692.57	691.83	666.94	758.14	DNF

	Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000
RDF/XML	Execution Time (ms)	105.6	116.6	97.6	144.4	317.2	735.4	1,459.4	2,715.4	7,467.9	17,042.0	32,910.5	DNF
	Standard Deviation	18.46	13.16	9.44	20.69	22.56	63.44	51.92	122.08	87.76	373.78	1,105.70	DNF
	Triples per second	94.70	171.53	512.30	692.52	630.52	679.90	685.21	736.54	669.53	586.79	607.71	DNF
N3	Execution Time (ms)	63.9	59.5	70.6	92.6	145.2	349.2	747.1	1,566.9	3,981.7	9,306.2	19,668.5	DNF
	Standard Deviation	6.82	4.95	13.13	8.30	7.13	32.40	47.28	100.89	199.56	155.59	694.85	DNF
	Triples per second	156.49	336.13	708.22	1,079.91	1,377.41	1,431.84	1,338.51	1,276.41	1,255.75	1,074.55	1,016.85	DNF
N-Triple	Execution Time (ms)	112.7	138.9	199.0	318.4	667.6	1,669.8	3,327.1	6,454.1	17,463.6	41,645.0	99,575.2	DNF
	Standard Deviation	19.90	22.23	9.68	31.21	91.70	64.35	257.44	380.30	197.23	708.54	5,115.08	DNF
	Triples per second	88.73	143.99	251.26	314.07	299.58	299.44	300.56	309.88	286.31	240.12	200.85	DNF
μJena	Execution Time (ms)	31.6	36.7	69.1	123.3	270.2	642.5	1,344.3	2,618.5	6,478.4	DNF	DNF	DNF
	Standard Deviation	7.96	2.75	5.15	15.24	14.45	17.68	22.56	42.07	205.29	DNF	DNF	DNF
	Triples per second	316.46	544.96	723.59	811.03	740.19	778.21	743.88	763.80	771.80	DNF	DNF	DNF
MobileRDF	Execution Time (ms)	67.1	73.9	75.9	83.7	276.6	352.6	655.4	1,301.3	3,096.3	7,215.4	17,323.0	DNF
	Standard Deviation	15.80	12.40	9.53	6.77	352.16	100.45	33.25	80.30	116.22	298.55	1,200.68	DNF
	Triples per second	149.03	270.64	658.76	1,194.74	723.07	1,418.04	1,525.79	1,536.92	1,614.83	1,385.92	1,154.53	DNF