Semantic Web-enhanced Context-aware Computing in Mobile Systems: Principles and Application

Stefan Zander and Bernhard Schandl

University of Vienna, Department of Distributed and Multimedia Systems
{stefan.zander|bernhard.schandl}@univie.ac.at

Abstract. The goal of this chapter is to provide detailed insights into the field of Semantic Web-based context-aware computing for mobile systems. Readers will learn why context-awareness will be a central aspect of future mobile information systems, and about the role semantic technologies can play in creating a context-aware infrastructure and the benefits they offer. The chapter introduces requirements, enabling technologies, and future directions of such systems. It presents a Semantic Web-based context-sensitive infrastructure that resembles concepts from graph theory and distributed transaction management. This infrastructure allows for an efficient acquisition, representation, management, and processing of contextual information while taking into account the peculiarities and operating environments of mobile information systems. We demonstrate how context-relevant data acquired from local and remote sensors can be represented using Semantic Web technologies, with the goal to replicate data related to the user's current and future information needs to a mobile device in a proactive and transparent manner. In consequence, the user is equipped with contextually relevant information anytime and anywhere.

1 Introduction

The increasing availability and power of mobile devices has significantly increased data-centric mobile applications. Mobility not only influences the types of information we need, but also how we access it, and which tools and mechanisms to process them are at our disposal. To improve mobile application development and the usability of mobile devices in general it is crucial to understand the information needs of mobile users, as well as the interaction metaphors they apply [1].

It is known that mobile search differs substantially from desktop search in terms of intra-query diversity [2]. The diversity of queries initiated in mobile setting is significantly lower compared to queries issued from the desktop. Additionally, query categorization reveals that context searching is similar to desktop, although query exploration is significantly lower. Further differences can be observed w.r.t. the effort needed to set up a query, and the total amount of queries initiated in one browsing session.

Therefore, plain internet access is often not sufficient for adequately addressing information needs of mobile users. Their situational contexts and current activities cannot be sufficiently addressed. Despite the benefits offered by mobile internet access, issues remain that hinder users from satisfying their information needs. These include impedimental interaction with the device while browsing for information, as well as the extensive attention needed for interaction and information seeking tasks [1]. Further, it was observed that mobile users sometimes do not know how to address a specific information need although they had the required resources and tools at their disposal [1].

It has been shown that 72% of mobile information needs can be attributed or related to context [1]. This finding indicates that introducing *context-awareness* into mobile information processing infrastructure can be of significant benefit to end users. Context allows information processing tasks to focus on the user's information needs, depending on their current situation. In a setting where data from various (internal and external) sources are processed on a mobile device, contextual information can help to determine the importance of data in relation to user tasks and activities.

1.1 Context-Awareness in Mobile Information Systems

Context-awareness in its simplest form describes a system's capability to conceive aspects of the physical and virtual environment it is operating in and tailor its behavior and responses according to the computational analysis of such aspects. Context-awareness can therefore be defined as a system's capability of using contextual data for providing relevant information and services with respect to the current situation of the user [3]. A system can be denoted as "context-aware" when it is able to adapt its behavior to ongoing activities, as well as the operational environment where it is used in [4, 5]. Context-aware applications and systems are able to react according those changes in an intelligent and user-related manner [6].

This can be accomplished by sensing and interpreting changing conditions, resources, and processes [7, 8]. Context-awareness can also be thought of as the machine-equivalent to the human capability of judging a situation and taking appropriate actions [9]. From a technical viewpoint, context-awareness refers to the accurate extraction, combination, and interpretation of contextual information, gathered from various multimodal sensors [10], and the objective of context-aware computing lies in the identification of the set of relevant features that describe and represent a given situation with the greatest possible accuracy [11].

Research in the domain of mobile computing [12] has attempted to use context-awareness for overcoming the technical limitation imposed by current mobile devices in terms of small screen sizes and limited interaction possibilities. Users often are confronted with multiple simultaneous activities and information channels, where context-aware computing promises to improve user interaction by reducing explicit user inputs and attention [12]. Context-awareness can be therefore considered as a methodology for facilitating human computer interaction by lowering explicit cognitive load and user attention.

One possible field where context-awareness can increase the quality of information management is *proactive information provisioning*. We can rightfully expect that an information system should be capable of providing information relevant to the user's current task. However, a system that is capable of doing this without the need for the user to explicitly issue search and retrieval operations can bring significant benefit, because often users are not capable of explicitly expressing their information needs. This is especially true for mobile environments and their limited interaction possibilities. In the following we present an example scenario where a user of mobile devices is supported by such a proactive information provisioning system.

1.2 Motivating Example

John is a representative of a medium-size software company. His tasks include to regularly contact potential customers in order to create awareness for his company's most recent products, to maintain relations with already existing customers in order to ensure their support and maintenance plans still work for them, and to represent the company at exhibitions, industry conferences, and relevant meetings.

His company maintains a customer relationship management software, a product database, a shared calendar system, an company-internal Wiki system as collaboration platform, and a shared file server to store all kinds of documents. Because of his job, John is often required to travel to abroad places. In consequence, he heavily relies on mobile infrastructure to get his work done. He has a powerful laptop, which he uses as his primary working device, as well as a mobile phone, which is used as his personal information management device.

When he is on travel, it is crucial for him to be equipped with all relevant information for his business meetings and other activities. However, he can never be sure to have online access to his company's network from wherever he goes, since certain limitations are in place: missing network coverage or security restrictions may prevent him from establishing a connection via the cellular network, and even if he manages to setup a connection, it may be slow and unreliable. For this reason, John often relies on local replicas of relevant data, which he stores on his laptop and (to a far lesser extent) on his mobile phone. However, because of the limited storage capacity of these devices and the necessary infrastructure, he cannot synchronize all data from all systems mentioned before, so he has to carefully select subsets of these data, which is a tedious and error-prone task.

This selection needs to be done before each trip, since he needs different information every time: this includes data about the (potential) customers he is going to meet (this includes organizations as well as persons), the locations and venues he is going to (including points of interest to visit in his spare time), latest information about the products he is trying to sell (which requires close cooperation with his company's product managers and development department), and data needed for his trip planning and administration (including timetables and travel accounting information).

A system that would be able to automatically select data for replication from a variety of systems would be highly desirable for John, since it would save him several hours of preparation time before each longer trip. Such a system could make use of a number of data sources, which provide valuable hints about which data could be of importance during his trip. First, John organizes all his upcoming appointments and travel plans in his digital calendar, which contains dates, locations, and participants of meetings. Additional information about people and organizations can be found in John's personal address book, as well as in the company's customer relationship management system. There, references to products that customers will use are mentioned; these refer to entries in the product database.

Additionally, the system could infer potential selling options from the interest topics that are stored for leads and potentials. Further, it can lookup information about locations and points of interests that John will visit from external public data sources, e.g., the Linked Open Data cloud. Further, it can find (via keyword lookups) articles from the company-internal Wiki system and the shared file server and replicate all these data to John's both mobile devices. For this purpose the system could rank each information item according to its assumed relevance, and replicate data according to the mobile devices' capacities.

During his trip, John will update and extend the replicated data with upcoming information (e.g., contact data and interests of new potential customers). Whenever his devices have sufficient network connection to his company network, the system should automatically synchronize his devices, upload changes, and update his local replicas according to possible changes in his context. After he has returned from the trip, all information is synchronized back to their origin systems, ensuring that no data are lost. If the system is able to track John's actual usage of replicated information during the trip, it can utilize this implicit feedback to adjust its relevance ranking algorithms, and therefore improve the selection for his next trip.

2 Background

This section is intended to provide fundamental background knowledge about technologies and concepts that are necessary for building a Semantic Web-based context-management and processing infrastructure for mobile devices. We provide an introduction to the concepts of context and context-awareness and ascertain the two main streams wherein context is either considered a representational issue reflecting environmental aspects, and as an emergent phenomenon that is continuously re-negotiated between communicating partners and thus cannot be determined beforehand, especially not at the design time of a mobile system. We show, how such dynamically evolving contexts can be represented and processed using technologies and concepts from the Semantic Web, while preserving/maintaining its unpredictable and dynamic characteristics—a requirement neglected by most context-aware computing approaches [13]. The section also gives a brief introduction to the general idea of the Semantic Web, its main constituents and involving technologies, as well as its most prominent knowledge representation languages. The section concludes with a discussion of possible areas where context processing and management can be substantially enhanced by the deployment of Semantic Web technologies, leading to an approach that we denote as *Semantic Web-enhanced Context-aware Computing.*

2.1 Context and Context-Awareness

The notions of context and context-awareness have been subject to controversial discussion and differing perception across communities. Several definitions have been proposed to context, depending on its actual usage as well as on the domain in which it was utilized. The word context is derived from the Latin word *con*, which means 'together' or 'with', and *texere*, which is the present infinitive of *'texö'*, meaning 'to weave' or 'intertwine'. A well-known and frequently stated definition of context has been proposed by Abowd and Dey [5] which define context as follows:

> *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

This definition describes context as a set of situations and actions (cf. [6]) that are subject of dynamic and frequent changes including the states of the involved entities (contextual information fragments). Context in general can be defined as "*everything that surrounds a user or device and gives meaning to something*" [13].

In [4] context is referred to as an entity that comprises location, identity, objects, as well as changes that apply to those objects. Ryan et al. [14] add the dimension of environment and time. A more user-related definition has been provided by [15] and [16] in which context is defined relatively to the user's emotional state, their believes, intentions, and concerns. These definitions take into account people in the closest proximity of the user. Other definitions such as in [17] and [18] define context on more abstract levels and wider scale: context is considered as "*the aspects of current situations*" as well as "*elements of the user's environment that are known by a computer*" [19].

Bolchini et al. [20] define context as an abstract process rather than a profile that determines how humans interweave their experiences with the environment surrounding them to give meaning to something. Other authors (e.g., [21, 22]) highlight that context is never universal in that it encompasses all information constituting a specific situation but always defined relative to a concrete situation. Coutaz et al. [23] characterize context as "*not simply a state of a predefined environment with a fixed set of interaction resources*" rather than part of an interaction process with the polymorphic and varied environment that is composed of "*reconfigurable, migratory, distributed, and multi-scale resources*".

However, due to the different proliferations on the notion of context, there has not been a clear consensus established on what context exactly is [22], but there is a common agreement on what it is about: context is concerned with an evolving, structured, and shared information space that is designed and utilized to serve a particular purpose [23].

Contextual information, which are constituent parts of surround contexts, can be considered as any information that may be used for describing the situation the user or a device is currently operating in. Contextual information is gathered through a variety of different technologies and considered as computational abstractions over distinguishable virtual or real-word aspects that have a specific relationship to the current task at hand. Contextual information can be *static* (e.g., the date of a person's birthday) or *dynamic* (e.g., a person's location). Dynamic context information is usually captured *indirectly* using sensors. Additional characteristics for classifying contextual information have been proposed by [25].

Basically, contextual information can be distinguished according to the way they are acquired: (1) Explicitly acquired contextual information is manually specified by the user and refers to information such as established social relationships or fields of interest. (2) Implicit contextual information is acquired via communication with hardware or software sensors, which capture specific aspects of the surrounding context by using sensing technologies or by monitoring user and system behavior. Most context frameworks acquire contextual information implicitly, especially from locally deployed physical sensors or embedded ubiquitous sensors. The challenge thereby is to identify the set of features that describe a given situation with the greatest possible accuracy and relevance [10]. Deriving reliable information from multiple heterogeneous sources in uncertain and rapidly changing environments is mandatory for context-awareness in the domain of mobile computing [26].

Consequently, two forms of context-awareness can be found in information systems [27]: *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.

2.2 Positivist and Epistemological View on Context

The technical or positivist school treats context as a conceptualization of human action and their interaction with the system. As a consequence, context is considered as a set of features of the environment surrounding the user's tasks at hand and generic actions, which can be computationally captured, represented, and processed. Especially technical disciplines consider context a representational issue and concentrate on sensorial or static data such as location, time, identity etc. (cf. [11]) putting emphasize on its codification and representation [24, 13]. Following this argumentation, context is instance-independent, separable from user activities, and can be scoped in advance [22]. This form of perceiving context has its root in information system since it adheres very well to existing software methodologies [13] but is contradictory to the phenomenological view of context that is grounded on subjective and qualitative analysis. This view considers context and activity inextricably connected and fundamental in giving meaning to something [22]:

> *Rather than considering context to be information, [the phenomenological view] instead argues that contextuality is a relational property that holds between objects or activities. It is not simply the case that something is or is not context; rather, it may or may not be contextually relevant to some particular activity.*
> *[...] the scope of contextual features are defined dynamically.*

This consideration emphasizes the aspect of relevance as context as such can not be defined or determined in advance since its scope is dynamically defined and changes frequently and unpredictably. It should be considered an *occasioned property* to emphasize its dynamic, fluent, and relative character as context arises in the course of action [22]:

> *Context isn't just "there", but is actively produced, maintained and enacted in the course of the activity at hand. [...] Context isn't something that describes a setting; it's something that people do. It is an achievement, rather than an observation; an outcome, rather than a premise.*

The epistemological view considers context as an interactional feature inspired by "sociological investigations of real-world practices" [22]. Context is inextricably linked to the process in which it is conceived, which renders the positivist and phenomenological view on context incompatible. As [34] points out:

Action and context are inseparable and should be analyzed as a whole. This implies that context-awareness should be examined in its relationship and consequences to the supported activity or actions.

As a consequence, the dynamic aspects of context are entirely neglected by technically oriented disciplines as context is a constitutional part of interaction processes that emerge opportunistically. Context should be rather considered as an emergent phenomenon or feature of interaction that is inextricably linked with user activities [34, 13] and continuously renegotiated between communicating partners [22–24] wherefore a pre-determination of the relevance of contextual elements is impossible – especially at design time of a system [21].

Therefore, a context management and processing framework should expose flexible, extensible, and open context descriptions that are not bound to a single static vocabulary to facilitate the dynamic and emergent nature of context. Context descriptions exposed by a context processing and management framework must be flexible, extensible, and use open vocabularies in order to facilitate the dynamic and emergent nature of context and should not be restricted to static schemas or single vocabularies. 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 [21]. The ability to dynamically handle and integrate new types of context information into existing structures is therefore a fundamental requirement of a context framework where open, and well-accepted vocabularies help in describing contextual information to guarantee their evolution and accurateness.

2.3 Context and Context-Awareness in Information Systems

The notion of context is mainly used in the information systems discipline for two reasons (cf. [22]): (1) contextual information is encoded to increase the accuracy of information retrieval processes and (2) contextual information is utilized for adopting systems to the environments in which they are used. In mobile computing, in contrast, context is used for (1) intelligent service provision, (2) realizing adaptive user interfaces, and (3) increasing the accuracy of information retrieval processes since context-aware information systems in general provide a "more natural and less obtrusive way of interaction" [24]. This is achieved by filtering the flow of information (i) from the device to the user to decrease information overload as well as (ii) from the user to the device, where user-generated data is augmented with contextual information predominantly in an automated and transparent manner. A variety of research endeavors elaborated on the nature of context and proposed a multitude of different definitions as well as—mostly taxonomical—classification schemes. A large share of classification schemes distinguish between *physical context*, that is, contextual information retrieved from physical or hardware sensors referring to the surrounding physical or virtual environment, and *logical context* that encompasses user-related information such as a user's goals, their tasks, emotional states etc. Such information is acquired by monitoring user interactions, specified by the users themselves, or inferred from physical contexts. Other works (e.g. [35]) classify context as *meaningful* if it has a particular relationship to the user's current high-level goals, or *incidental* in case the user enters an unpredicted or unexpected situation that has no special relationship to their high-level goals.

Due to the diversity of contextual information, several categorization frameworks have been introduced to manage and comprehend such information systematically. A context model in general is used for the definition and storage of contextual data in a machine-representational form. The most prominent approaches have been summarized in [36], ranging from simple key-value models, to complex logic or ontology-based models.

The diversity of contextual information is also reflected in the architectures proposed for context management and processing, which differ in their technical capabilities, acquisition techniques, context representations and reasoning capabilities etc. Despite the wide variety of different context management and processing architectures, a common architecture is identifiable [19][1]. Figure 1 displays this conceptual architecture which consists of five separate functional layers:

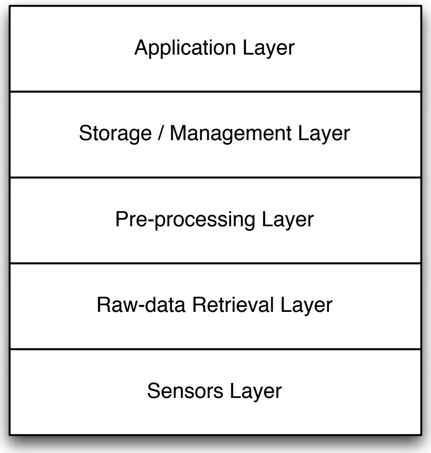

Fig. 1. Layered conceptual architecture of context frameworks (adapted from [19])

Sensors layer: The sensors layer is the bottommost layer and comprises the set of sensors that a context framework exploits. Such sensors can be classified into three groups [37]: (i) *physical* or *hardware sensors* for gathering physical context information, (ii) *virtual sensors* that capture data form software applications and service by observing user interaction and user behavior, and (iii) *logical sensors* that are responsible for deriving higher-level context information usually by augmenting physically or virtually acquired contexts with additional information from repositories or other data sources.

Raw data retrieval layer: This layer is concerned with the retrieval of raw sensorial data usually by encapsulating driver logic or sensor APIs in dedicated wrapper components (cf. Widgets [28]). It offers common interfaces that encapsulate specific drivers or sensor APIs thus making them exploitable for upper layer components. This layer exposes query-functionality that abstracts from low-level raw sensorial data and provides higher-level representations and functions.

Pre-processing layer: In case sensorial data is too coarse [19] or need to be clustered for further processing, this is handled by the pre-processing layer. It also provides appropriate abstractions of contextual data whose representation is too technical (e.g. a bit-wise representation of contextual data). Interpretation, aggregation, and reasoning tasks are also handled by this layer together with quantization algorithms for aligning raw-sensorial data with the elements of a framework's context model.

Storage and Management layer: This layer serves as a context repository as it organizes contextual information and offers interfaces to client applications. Components in this layer expose different operation models where contextual information can be either requested in a polling-based style where the client applications issues requests to a context server in regular time (*synchronous*) or via a subscription mechanism where clients are notified by the context server whenever new contexts are available (*asynchronous*). Due to the

[1] See [19] for a more detailed discussion about the different layers.

dynamic and unpredictable nature of context (cf. [22, 23, 13]) the use of an asynchronous communication style (cf. [38]) is suggested [19].

Application layer: The application layer is the uppermost layer and hosts the client applications that request and process contextual information from a context repository or a framework respectively. Contextual information is usually consumed by applications in this layer in order to adopt their actions with respect to user-related tasks. One popular example is increasing the background luminosity level in case the user enters an outdoor area.

Context can be acquired from a multitude of different sources, e.g., by applying sensors or sensor networks, by deriving information from the underlying communication infrastructure or network, or by acquiring status information and user profiles directly from the device the user currently uses [19]. In general, every information or any content can be considered context-relevant if it provides information related to the user's current tasks and activities [39]. Therefore, context-acquisition should not be restricted to capture context merely from hardware sensors. Instead, additional information sources should be integrated into the acquisition process. A social bookmarking service for example could be requested for obtaining information related to the user's interests or information needs. Especially when considering Web 2.0 applications in which users actively contribute, such services likely contain valuable information that can be used for complementing sensorial data [39].

The way contextual data are acquired significantly influences the architectural style of a context-aware system [19]. Context acquisition architectures can be broadly classified into three different groups (cf. [40]): (1) *proprietary architectures* that directly access locally deployed sensors where sensor and driver logic is directly implemented in application code limiting context reuse and exchange, (2) *middleware infrastructures* which employ a layered system architecture (e.g. [30, 41, 42]) encapsulating sensor specifics in dedicated components and expose uniform interfaces for context utilization, and (3) *context server architectures* (e.g. [43]) that operate similar to database management systems and offer remote access to contextual information hosted within a context repository. A detailed discussion about the advantages and limitations of each architectural style can be found in [40] and [19].

Different classification schemes for context management architectures have been proposed by, e.g., [28, 19, 21]. They distinguish between a (1) direct integration of context management into context-aware applications, (2) context management services, and (3) context-aware devices and services augmented with context management functionalities. By directly integrating context management functionality into mobile applications, such applications need to know in advance which sensors are available and how to communicate with them. As a consequence, context data semantics are limited to the applications in which they were interpreted and can, in most cases, not be shared between applications which renders a dynamic integration of new sensors in running frameworks difficult. Sensors have to be queried by each application separately, which impede the process of context aggregation, exchange, and dissemination. In contrast, if context management functionality is implemented in devices or sensors, their functionality can be shared among components and integrated during run-time (e.g. [10, 21]).

The usage and utilization of the notion of context in information systems raises some technological as well as human-related challenges. Dey et al. [28] identified the poor understanding on what context constitutes, how it is to be represented in information systems, and the lack of conceptual models, methods, and tools that would promote the design of context-aware mobile applications as one of the main reasons why context-awareness has only insufficiently found its way into the essence of mobile and ubiquitous computing yet. This additionally hampers empirical investigations in human-computer interaction and interaction design.

Many works indicated the non-existent availability of a general model of context and context-awareness as one of the main problems of context-sensitive systems. This fact in particular concerns mobile computing where the notions of context and context-awareness are used ambiguously across communities and reflect specific application domain peculiarities (cf. [20, 29]). This problem also hampers context-aware application development for mobile systems since a widely-accepted and well-defined programming model does not exist wherefore sensor-logics are often hard-wired into application code and application developers have to deal with low-level interactions between sensors and context-acquisition components [28]. Therefore, the semantics of contextual information are limited to the applications in which they were acquired and are represented using proprietary formats. Newer approaches (e.g., [10, 30]) employ more flexible designs for context processing and representation where sensor-logics or sensor-specific APIs are encapsulated in specific components that can be mutually shared or employ middleware infrastructures (e.g., [31, 32]) for facilitating communication and interoperability between context processing components while using knowledge representation languages from the Semantic Web such as RDFS or OWL for representing contextual information [33, 24].

2.4 Problems of Context-aware Computing

A fundamental problem of context-aware computing is that of *context ambiguity* [3] and *context imperfection* [25] which refers to the implicit assumption shared by many context-aware computing approaches that the computational context is a 1-to-1 reflection of the real-world context. Evidently, this assumption is wrong since the way context is conceived by individuals differs substantially from the way it is acquired and represented electronically [3, 22]. A logical consequence of that misperception is that a context framework can only work on a more or less accurate context representation where the degree of accuracy is determined by numerous technical and soft factors. The unpredictable and relative nature of context renders a determination of all contextual aspects that constitute a specific context at a system's design time difficult, if not impossible, since context is always defined relative to the situation in which it is used. An electronic representation of context can therefore never be universal in that a context model contains all information that characterize a given situation; instead it only represents a relevant subset of the constituting real-world context [3, 25, 22]. The problem is that a 1-to-1 relation between a situation and the describing context information does, in most cases, not exist wherefore a situation can be represented by multiple context models with a specific degree of accuracy w.r.t. the several viewpoints. Several methodologies such bayesian networks, case-based reasoning, stochastic models, or machine learning techniques have been proposed for defining precise transitions between different context descriptions with as little ambiguity and overlapping as possible. However, such approaches contribute towards increasing the accuracy of context acquisition and context representation but do not help in identifying all constituting aspects of a specific situation. Context-aware computing in general is only an approximation to a real-world situation rather than a 1-to-1 reflection of it.

Another problem of context-aware computing is that most architectures are targeted towards a specific application or domain [21]. The difficulties hereby are that certain high-level context interpretations are not absolute characterizations per se, since the concept 'high temperature' for instance depends on the context or situation in which it was acquired. This dependency makes it difficult to share context information in an application and domain-independent manner since implementing new application behaviors based on context characterizations made for one application might not be appropriate for another one [21]. Some works [12, 44, 13] therefore focus on describing and representing context in an application independent manner by means of concepts from activity theory [45, 46] using collaborative plans [47], task analysis [48, 49], aspect-oriented context modeling and modularization [50], or situational reasoning [30, 42] to decouple context from specific application domains and provide abstract contextual concepts (e.g., "business meeting") that adhere to upper-level ontologies. Context and contextual information needs a uniform representation to be effectively managed, integrated, and processed by reducing the ambiguities inherently attached [8].

The proposed context management and processing framework provides the necessary 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 context interpretation, consolidation, and augmentation.

2.5 Semantic Web

The *Semantic Web* [51] is the idea of expressing rich, machine-processable knowledge using the Web infrastructure. Descriptions about *resources* (which are entities of any kind, including digital objects like documents and media, physical objects like humans, cars, or buildings, and abstract concepts like locations, topics, and time periods) are published in a structured format and using vocabularies that follow a well-defined semantics. Applications can consume these descriptions, interpret them, merge them with descriptions from other sources, and infer new knowledge or determine the truth value of statements. In analogy to the World Wide Web, which nowadays serves as one underlying knowledge-provisioning infrastructure for a wide variety of human knowledge workers, the Semantic Web is envisioned to serve as an underlying information-provisioning infrastructure for information-centric applications, whereas the information processing is performed semiautomatically by computer programs.

Broadly, the Semantic Web consists of a stack of technologies that build on each other. The core technologies are shared with the World Wide Web: *Uniform Resource Identifiers* (URIs) [52] to identify resources, and *Hypertext Transfer Protocol* (HTTP) [53] for the transportation of information. On top of these core technologies, the *Resource Description Framework* (RDF) [54] is used as the abstract data model in which all information is represented. RDF is a triple-based graph model, with the *statement* being the atomic unit of information. Each statement consists of three elements (*subject*, *predicate*, and *object*), where the *predicate* identifies the relationship that is asserted to exist between the *subject* and the *object*. URIs can be used for all three elements; whenever the same URI is used in multiple statements these statements are referring to the same resource (or real-world entity). Therefore, a set of RDF statements that shares common URIs can be interpreted as connected graph (cf. Figure 2). RDF itself is an abstract data model; in turn, there exist several serialization formats that can be used to exchange RDF statements between parties, including RDF/XML [55], Turtle [56], and RDF/JSON[2].

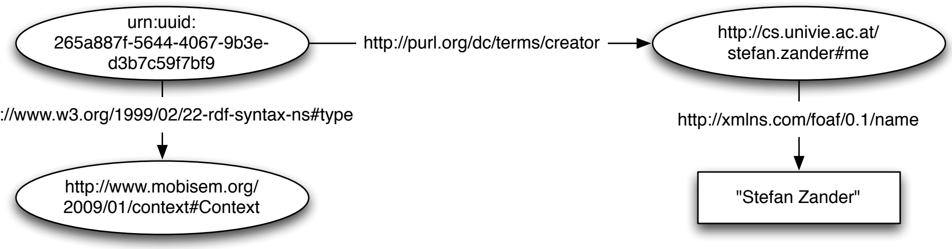

Fig. 2. Example RDF Graph

Based on RDF a set of technologies has been developed that aims to make more "knowledge" out of the data represented in RDF graphs. Higher-level languages like *RDF Schema* (RDFS) [57], *Web Ontology Language* (OWL) [58], and *Rule Interchange Format* (RIF) [59] can be used to define the constraints of a domain of discourse based on formal logics; using these languages, valid combinations of statements can be defined axiomatically. This allows implicit knowledge to be discovered based on asserted information, to detect inconsistencies in knowledge bases, or to determine the truth value of statements, given a set of background knowledge. A query language (*SPARQL*) [60], which resembles similarity to the SQL language for relational data, can be used to formulate structured information needs, which are evaluated against a set of RDF graphs.

One line of development within the ongoing Semantic Web research field is *Linked Data* [61], which denotes the practice of publishing data on the Web according to simple core principles [61]. Its core idea is to denote

[2] `http://n2.talis.com/wiki/RDF_JSON_Specification`

resources (which includes real-world entities as described above) exclusively using HTTP URIs, and to allow data consumers to directly de-reference these URIs (i.e., to fetch their representations using HTTP `GET` methods). Upon this de-referencing, structured information about the resources is returned, which contains links to other relevant resources. These other resources can then, in turn, be retrieved by the client, allowing it to navigate through a global information network based on the "follow-your-nose" principle.

Since 2007, when the Linked Data W3C community project[3] was established, a significant amount of data has been published according to the Linked Data principles. This includes popular data sets of general interest like *DBpedia* (consisting of structured information extracted from Wikipedia pages), geographic information (like *Geonames*), media-related content like *BBC Programmes*, and bibliographic information (e.g., *DBLP*). An example of a highly distributed data set is the entirety of all *FOAF Profiles*, which are usually served on private infrastructure, and are interconnected based on social relationships between their owners. Through the (partly indirect) interconnection of these data sets, light-weight data integration can be performed, and information about the same entities can be gathered and combined from heterogeneous, distributed sources.

2.6 Semantic Web-enhanced Context-aware Computing

For managing context information systematically, a common structure for representing contextual information need to be established [26]. The Resource Description Framework, discussed in the previous section, has proven to be an appropriate representation framework for representing complex contextual constellations and facilitating the sharing and exchange of context descriptions based on ontological semantics [62]. It can be used for codifying the semantics of contextual information as well as the relationships among them in a well-defined, uniform, and systematic way. On top of RDF, RDF Schema (RDFS) offers a simple set of common language properties that can be used for building context descriptions which can be shared among different context providers and consumers collaboratively [26]. However, the set of RDFS language elements is not sufficient for expressing rich contextual constellations, wherefore the use of more expressive languages such as OWL is suggested [24, 21]. Generally, the use of ontologies as a key component for building a context-aware computing framework is broadly acknowledged [42, 63, 21, 26], and it has been shown that Semantic Web technologies are sufficiently mature and performant to be deployed on mobile devices [64].

A context ontology serves as a uniform representation of contextual information and enables a systematic management of context-relevant aspects; it should be separated from application logic [65]. A number of approaches use single ontologies for context information representation and for the transformation of raw, low-level context data into high-level context descriptions [66, 65]. Some of these ontologies refer to the analogy of physical objects, i.e., their concepts refer to objects in the real world (the studied context).

Several works have already demonstrated that ontologies are appropriate means for expressing and representing contextual information since they incorporate some characteristics that are essential for mobile and ubiquitous environments [21, 41, 67, 68]. Ontologies are highly expressive and widely adopted knowledge representation techniques, and a multitude of open software tools for their design, creation, management, and storage are available [33]. Ontologies offer a well-defined set of concepts and relationships to model the domain of interest, which can be adopted by context-management frameworks to integrate and share contextual knowledge from other domains to facilitate context exchange and reuse. Ontologies are based on knowledge representation languages that are open with respect to evolutions of the domain they describe. This allows ontologies to be adapted and extended according to domain-internal changes.

Building a context-awareness computing infrastructure on the principles and technologies of the Semantic Web has several implications and advantages: due to the fact that ontologies are based on the *open world assumption*, context ontology evolution is a central aspect in context management and allows for adapting

[3] `http://esw.w3.org/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`

and modifying a context ontology according to changed conditions. Due to the fact that RDF is a system- and application-independent framework for modeling data and its close relatedness to Web technologies, it is well suited for the data exchange between components (e.g., using HTTP). This facilitates interoperability among context frameworks and services since established and well-known vocabularies together with their inherent semantics can be understood and used across systems.

Since ontologies provide a common structure for representing and describing the relationships and semantics of context-relevant information in a machine-processable way, they can be conceived as a general approach for systematic management of context information. In such a setting, RDF can be used as a description syntax to enable the communication and sharing of context information between collaboratively communicating partners, i.e., applications, services, and devices. These descriptions are represented as *labeled multi graphs*, where the contained entities are referred to through HTTP URIs (see Section 2.5). Its open architecture allows for the integration of different context-relevant vocabularies so that context descriptions can dynamically grow and become more elaborated to better reflect intra-domain evolutions.

Ontologies help in expressing application or service needs, and in aligning them to acquired context information wherefore only relevant information is extracted. This simplifies query processing since a context consumer can limit queriers to relevant information, instead of processing the entire context description. In cases of incompatibility of context descriptions, ontology matching algorithms help in reconciling differences in description semantics. Euzenat [69] therefore suggests the use of *ontology alignment services* to identify correspondences between incompatible context descriptions. Such services perform query transformations and reflect domain and information space evolutions [21].

Semantic Web technologies allow for mapping low-level sensor data to high-level ontological concepts so that collected context-relevant information is transformed and embedded in a controlled context description. Based on ontological semantics, new facts can be deducted by applying aggregation and reasoning heuristics. In this way, Semantic Web languages such as RDFS and OWL allow for aggregating heterogeneous and autonomously acquired context information both on a syntactic and semantic layer. By transforming sensorial data into RDF statements, context acquisition components are not required to anticipate possible queries beforehand. Instead, the requesting context consumers determine the data that are relevant for them.

In the fields of Semantic Web and Linked Data, a number of vocabularies and ontologies have emerged that are of interest for the representation of contextual and situational information (e.g., time[4] and location[5], technical parameters[6], or social aspects[7]). The elements (terms and concepts) of those vocabularies are well-known across communities and expose a well-defined and commonly understood semantics that allows for information integration and exchanges especially in heterogeneous system and network infrastructures. Additionally, such vocabularies are continuously maintained by communities to guarantee their accurateness and evolution. By re-using such vocabularies and (implicitly) connecting context descriptions to external Linked Data sources, we gain two benefits: first, if context descriptions are distributed (either to the public or within a closed environment, e.g., a corporate network) they can be directly combined with already existing data, and existing tools can be directly applied to contextual information without the need to adapt existing software. Second, data from external sources can be imported and used to enrich the context descriptions, leading to a richer semantics, which facilitates more powerful processing and reasoning.

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

3 Architecture of a Semantic Web-based Context Framework for Mobile Systems

3.1 Overview

In general, a framework-based approach for context management is broadly suggested to facilitate and speed-up context-aware application development and allow for gathering, aggregating, and interpreting contextual data in a structured, well-defined, and controlled way [8]. 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. In the following, we give an overview of the main concepts and functionalities of the proposed framework:

1. *Acquisition.* The framework allows for acquiring context-relevant data from a wide variety of sources, ranging from locally deployed hardware and software sensors, over sensors located in ubiquitous environments, towards Web 2.0 APIs for retrieving data related to a user's personal or social networks. Raw sensor data are represented in form of an RDF-based context description and transformed into a high-level context description using concepts and languages from the Semantic Web.
2. *Representation.* The architecture of the context framework has been designed to impose minimal to none restrictions on the representation of contextual data and allows for using individual vocabularies and heuristics for explicitly representing data semantics and reason on contextual data.
3. *Aggregation.* Reuse, exchange, and augment contextual information to built richer and more elaborated context models is a fundamental principle of the proposed framework. Context descriptions can be mutually refined and complemented with additional data. Since context descriptions are represented as RDF graphs using open and well-known Semantic Web vocabularies, their data semantics can be understood across components.
4. *Orchestration.* An orchestration framework analyzes the data descriptions of context providing components called *context providers* and use them as a basis for cascading context providers in a *directed acyclic network graphs* to dynamically route context descriptions between them. The orchestration framework has been designed to allow for integrating individual orchestration rules and metrics, where the results are stored in an *adjacency matrix.* This matrix is dynamically and transparently re-created whenever a new context provider is integrated into the framework.
5. *Consolidation.* To maintain context consistency, accurateness, and completeness, context descriptions are collected in a central place to guarantee consistency among context descriptions and acquisition processes while taking into account technical and operating system peculiarities of mobile platforms.It also incorporate compensation strategies to minimize the impact of malfunctioning or unavailable context providers and sustain a proper functionality of the context framework's processes.
6. *Reasoning.* A lightweight forward-chaining rule reasoner has been developed in order to consolidate context descriptions, detect inconsistencies, and transforms them into a global, consistent, and coherent context model that represents the current user context.
7. *Dissemination.* User context is forwarded to other components deployed in the framework in an automated and transparent manner using a push-based notification mechanism. Additionally, external context services can request a model of the user's current context via an elementary HTTP server.
8. *Replication.* Data is replicated in a transparent and automated fashion to the device where no restrictions are imposed to the data sources, which might range from file systems, data bases, web repositories, towards web applications etc. Data replication is completely decoupled from context acquisition where mobile applications can access data replicas from a local triple store in a controlled and uniform way.

9. *Storage.* The framework incorporates a persistence layer that allows for storing replicated data sets in a local database from where they can be accessed and utilized. The persistence layer also includes support for named graphs and contains projections for transforming RDF graphs into a relational database scheme and vice versa.
10. *Data Access and Provision.* Access to replicated data stored in the local SQLite database is provided via an Android content provider that has been adapted for RDF data provision and storage. The RDF content provider assigns a unique URI to each replicated data sets and allows other mobile applications to access and utilize data replicas.

To realize these concepts, we synthesized concepts form graph theory, distributed transaction management, and the Semantic Web to build an architectural infrastructure for mobile systems that combines context acquisition and data replication to replicate data related to the user's current and future information needs in a transparent and proactive manner. The framework cascades context providers within so-called *orchestration trees* that resemble concepts of a workflow scheme. They allow for a controlled execution of combined context acquisition tasks in a decoupled manner while maintaining data and process consistency. This form of context provider orchestration allows for an efficient acquisition and aggregation of contextual information while taking into account the mobile operating system peculiarities [70, 71].

Our approach exposes two significant advantages compared to existing server-based approaches: contextual information is acquired, processed, and disseminated directly on a mobile device and does not depend on the availability of external systems. This reduces security and privacy issues since highly private data such as contact information or appointments do not need to be transferred outside a mobile system. It also serves as a technical infrastructure for the deployment of high-level context processing and recognition services to enable situation awareness (cf. [72, 73]).

3.2 Components

Context Providers As illustrated in Table 1, the context framework is able to acquire contextual information from a variety of context sources and sensors. Those sources are wrapped by *context providers*, which employ two operation modes. *Primary context providers* are self-contained components that operate independently and autonomously, and provide contextual information in a proactive manner. They become active whenever a change in the corresponding context source is detected. In contrast, *complementary context providers* react according to updates received from primary context providers and complement contextual information acquired by primary context providers by taking those context descriptions as input data for initiating their acquisition tasks.

Table 1. Types of context sensors

	Local	**Remote**
Hardware	Device-internal hardware sensors for capturing physical context (e.g., location, inclination, orientation, etc.)	External ubiquitous sensors deployed in ubiquitous environments that allow for acquiring mostly physical context
Software	Locally deployed software or logical sensors for monitoring application and user behavior to deduce current and future mobile user information needs	Software sensors that wrap interfaces or APIs of remote data sources mostly for complementing primarily acquired context information (e.g., Web services, APIs, online repositories, etc.)

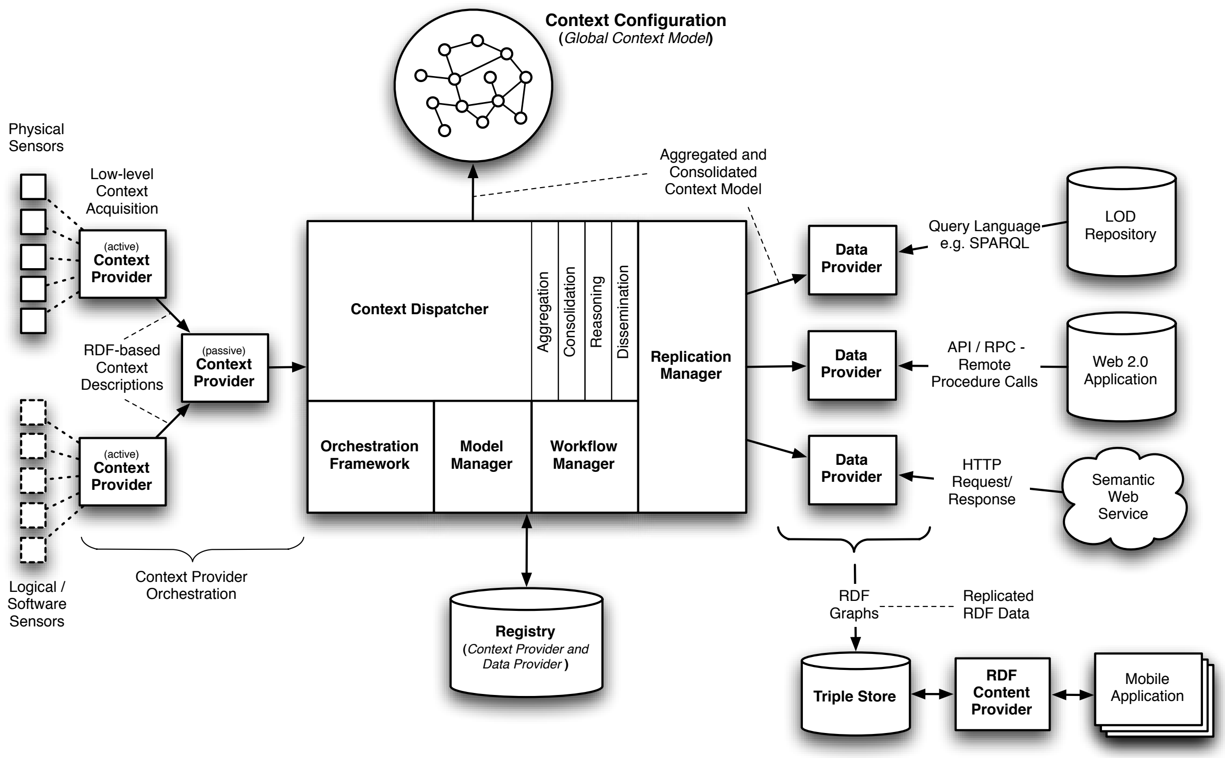

Fig. 3. Architecture of the MobiSem Context Processing Framework

A context provider captures a specific and relevant contextual aspect of the user's current context. A contextual aspect is represented in structured and well-defined ways using semantic technologies (RDF, RDFS, OWL) to facilitate context information exchange and integration. This allows them to be enriched and complemented with additional and related information in order to enhance the richness and accuracy of the contained information.

Context Dispatcher The context dispatcher is the central component within the context framework. It handles all communication between context providers and propagates context models between them. It is notified whenever a new context model has been acquired by a context provider. The context dispatcher collects such updated context models, aggregates them, performs additional processing like inference and consolidation, and creates the so-called *context configuration* (see below), which is a complete, accurate, and consistent representation of the current context, and is subsequently propagated to data providers. Currently, reasoning and consolidation is performed on the basis of hard-coded rules. In order to dynamically deploy new reasoning rules to the context dispatcher, a lightweight rule-based reasoner has been developed and integrated into the framework.

Context Description Queue The context description queue is part of the context dispatcher[8] and handles the communication between the context providers and the context dispatcher. It implements specific logic for the management and exchange of context models to enable the recovery of lost descriptions and also serves as a *compensation mechanism* in case of temporarily unavailable context sources or malfunctioning context providers. Therefore, it not only buffers the most recent context updates, but also stores previously committed context updates. This allows the context framework to revert to previous context models to sustain the proper execution of context acquisition processes.

[8] Please note that the context description queue is not included in the Figure 3.

Model Manager The model manager is used for storing and tracking the context providers' context models that have been acquired in the course of context acquisition processes. For every context provider deployed in the framework, the model manager stores the most recently updated context model together with status information about the corresponding provider.

Orchestration Framework The orchestration framework dynamically orchestrates compatible context providers in form of a direct acyclic graph based on the type of context information they provide. This graph is called *orchestration tree* and resembles the concepts of a workflow scheme that allows for a controlled execution of combined context acquisition tasks in a decoupled manner while maintaining data and process consistency. By analyzing such data descriptions, the orchestration framework computes compatibility measures so that relationships between context providers can be inferred. Every context provider must therefore provide an RDF description about the type of data it acquires and the vocabularies it uses for representing that data. The orchestration framework operates completely independent from other framework components and is initiated automatically whenever a new context provider is deployed in the framework.

Workflow Manager The workflow manager is responsible for the management and coordination of the context providers' acquisition processes where compatible context providers are orchestrated in an *orchestration tree*. Those orchestration trees are executed in *context acquisition workflows*, that control and manage the acquisition tasks of the involved context providers to sustain a deterministic and consistent behavior. Context updates are propagated through context acquisition workflows and routed between compatible context providers. When all context acquisition tasks have finished, the workflow manager sets the corresponding entries in the model manager and notifies the context dispatcher that a new global context model can be created.

Registry The registry is the central storage component where all context and data provider deployed in the framework must register in order to be integrated in acquisition and replication workflows. It automatically notifies the orchestration framework whenever a new context provider has been registered so that it can be automatically orchestrated with compatible providers.

Context Configuration The context configuration represents an aggregated version of all context providers' context models. It is created by the context dispatcher when all context acquisition workflows completed and the updated context models of their containing context providers are available.

Replication Manager The replication manager controls and orchestrates all data replication tasks. It operates completely decoupled from the other framework components and gets notified by the context dispatcher whenever a new context configuration has been created. The replication manager is responsible for the instantiation of the *data provider control threads* which control and monitor data replication tasks and propagates the context configuration to each data provider. The replication manager also receives notifications about changed data replicas in order to initiate write-back and synchronization operations.

Data Providers Data providers replicate any kind of RDF data to the mobile device; they can request data from virtually any external data sources or generate data replicas themselves and operate completely decoupled and independent from each other. Each data provider is assigned a unique identifier that is used as part of the addressing scheme to store data in the local triple store. Data provider adjust and initiate their data replication tasks based on the analysis of the context configuration that they receive by the replication manager. For instance, a data provider can analyze data regarding the current location of a device and retrieve information about nearby points of interest.

Triple Store Our triple store implementation is designed to be a lightweight, efficient storage and retrieval mechanism for RDF triples. It abstracts over the concrete storage mechanism that is used by the mobile platform[9] and provides support for *named graphs* [74], persistence, and RDF serialization and de-serialization. It employs a *normalized table layout* (cf. [75]) where resources, literals, and blank nodes are stored in separate tables[10] and provides support for named graphs.

RDF Content Provider Replicated data sets are provided to external applications via the RDF content provider. This provider is an individual implementation of an Android content provider (see [80]) for the system-wide provision of RDF data and contains the projections for transforming RDF graphs into the relational database schema of the local SQLite database and vice versa. It exposes a common interface applications can use for performing query, update, insert, and delete operations on replicated data. It has been extended with named graphs support [74] where each data replica is stored using a unique URI. Content providers expose configurable content URIs that can be used for addressing specific parts of replicated data.

3.3 Orchestration of Context Providers

To facilitate this kind of cooperation between decoupled context providers, the context framework dynamically routes data between context providers based on the type of context information they provide. Therefore, 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 4 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. The latter 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[11].

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 framework 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

[9] Most mobile systems use specific storage systems such as the Record Management System (J2ME compatible devices) or a SQLite database (Google Android).

[10] A discussion regarding other database layouts for storing RDF triples including their advantages and limitations can be found in [76].

[11] 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. 4. Exemplary data description for a complementary context provider for extracting contact data from calendar entries

acquisition tasks as such. 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 [77].

3.4 Formal Model for Orchestrating Context Providers

The idea of orchestrating context providers is to augment and complement contextual information to build richer, more expressive and elaborated context descriptions while maintaining data and processing consistency, accurateness, and completeness. Therefore, context providers are orchestrated in a *directed acyclic tree* called *orchestration tree* based on the type of context information they acquire and provide.

Let $\mathbb{P}$ be the set of all primary context providers P_i with $\mathbb{P} := \{P_i | i = 1, \ldots, x\}$ and $\mathbb{C}$ the set of all complementary context providers C_j with $\mathbb{C} := \{C_j | j = 1, \ldots, y\}$. A context source that is wrapped by a context provider is denoted as S_k; let $\mathbb{S}$ be the set of all potential context sources $S_k \in \mathbb{S}$ and let $\mathbb{A}$ be the set of all context providers A_k where $\mathbb{A} := \{A_k | k = 1, \ldots, x + y\}$ that are deployed on the context framework and wrap a specific context source S_k irrespectively of their concrete type. Therefore, we can state that the set of context providers $\mathbb{A}$ is the union of the sets of all primary and complementary context providers ($\mathbb{P} \cup \mathbb{C}$) where $\mathbb{P}$ and $\mathbb{A}$ can be equal in case only primary context providers are deployed on the framework, whereas $\mathbb{C}$ is always a real subset of $\mathbb{A}$ since complementary context providers always require a primary context provider to be operational:

$$\mathbb{A} = \mathbb{P} \cup \mathbb{C} \text{ with } \mathbb{P} \subseteq \mathbb{A} \wedge \mathbb{C} \subset \mathbb{A} \text{ and } \mathbb{P} \cap \mathbb{C} = \emptyset \tag{1}$$

Context providers specify the data they provide in a data description (see Section 3.3) that serves as a basis for defining relations between compatible context providers in terms of the contextual data they acquire and require for performing their acquisition tasks. These relations are analyzed by the orchestration framework and recorded in an *adjacency matrix* (cf. Figure 6). For each primary context provider $P_i \in \mathbb{P}$ we can derive an orchestration tree O_i^P from the adjacency matrix (cf. Figure 7). An orchestration tree O is a *directed acyclic tree* whose root element is always a primary context provider P_i and the adjacent nodes represent complementary context providers C_j that complement the data acquired by the corresponding primary context provider P_i. Therefore, there is always a 1:1 relationship between a primary context provider P_i and it corresponding orchestration tree O_i^P where a projection $P_i \longmapsto O_i^P$ exists for all $P_i \in \mathbb{P}$ and $O_i^P \in \mathbb{O}$.

An orchestration tree O_i^P can be represented as a directed acyclic graph (V, E, α, ω), where the vertices V represent context providers, and the edges E represent relations between them in the form of exchanged context models, where the context model of a predecessor node is consumed by its successors. α and ω are projections and specify concrete relations between predecessor and successor context providers.

More specifically, an orchestration tree $O = (V, E, \alpha, \omega)$ can be described as follows:

1. V is defined as the non-empty set of a primary context provider P_i and its corresponding complementary context providers $\mathbb{C}_i^P$: $V := \{P_i\} \cup \mathbb{C}_i^P$
2. R is the defined as the set of relations r that exist between the context providers contained in V where r is defined as the tuple $r \in \{P_i\} \times \mathbb{C}_i^P \cup \mathbb{C}_i^P \times \mathbb{C}_i^P$
3. $V \cap R = \emptyset$

4. $\alpha : R \to V$ and $\omega : R \to V$ are projections where $\alpha(r)$ is the starting or direct *predecessor* context provider and $\omega(r)$ the ending or direct *successor* context provider of the relation r. $\alpha(r)$ and $\omega(r)$ are adjacent to each other.
5. $\forall r \in R : \alpha(r) \neq \omega(r)$

With this definition, orchestration trees can be represented in an adjacency tableau as depicted in Figure 5 that represents the relations r_i together with preceding and adjacent context providers $P_i \in \mathbb{P}$ and $C_j \in \mathbb{C}$.

	α	ω
r_1	P_1	C_1
r_2	P_1	C_2
r_3	P_2	C_5
r_4	C_2	C_3
r_5	C_2	C_4

Fig. 5. Example of an adjacency tableau for eight context providers

The set $V_i^O \in O_i^P$ contains the primary context provider $P_i \in \mathbb{P}$ as well as all compatible complementary context providers orchestrated in the orchestration tree O_i^P. Therefore, let $\mathbb{C}_i^P$ be the set of complementary context provider $C_j \in \mathbb{C}$, which are orchestrated in an orchestration tree O_i^P where $\mathbb{C}_i^P \subseteq \mathbb{C}$. Hence, $V_i^O \in O_i^P$ can be defined as the union of the primary context provider $P_i \in \mathbb{P}$ and the set of compatible context providers $C_j \in \mathbb{C}_i^P$:

$$V_i^O := \{P_i\} \cup \mathbb{C}_i^P \text{ where } \mathbb{C}_i^P \subseteq \mathbb{C} \tag{2}$$

As a consequence, the set of all complementary context providers $\mathbb{C}$ can also be defined as the union of the partial sets of complementary context providers $\mathbb{C}_i^P$ orchestrated in an orchestration tree O_i^P and the set of providers C_k that are also deployed on a system but not part of any orchestration tree O_i^P:

$$\mathbb{C} := \bigcup_{i=1}^{|\mathbb{P}|} \mathbb{C}_i^P \cup C_k \text{ with } \bigcap_{i=1}^{|\mathbb{P}|} \mathbb{C}_i^P = \emptyset \tag{3}$$

Since an orchestration tree O_i^P contains only distinct elements, where a context provider P_i or C_j can only be part of one orchestration tree, we can state that all sets V_i^O and V_j^O are pairwise disjunct:

$$\forall V_i^O \in O_i^P, V_j^O \in O_j^P, i \neq j : V_i^O \cap V_j^O = \emptyset \tag{4}$$

Further, let $\mathbb{O}$ denote the union of all orchestration trees O_i^P with $P_i \in \mathbb{P}$ and $|\mathbb{O}| = |P|$, and V_i^O the set of all context providers orchestrated within an orchestration tree O_i^P. Therefore, V_i^O can be defined as:

$$\mathbb{O} := \bigcup_{i=1}^{x} O_i^P \text{ and } \bigcap_{i=1}^{x} V_i^O = \emptyset \text{ where } x = |\mathbb{P}| \tag{5}$$

For building an orchestration tree, the orchestration framework analyzes the data description of each context provider and builds an *adjacency matrix.* The adjacency matrix represents a graph $G = (V, E)$ with $V =$

$\{v_1, ..., v_n\}, n = |\mathbb{A}|$ where vertices v_i represent context providers orchestrated in an orchestration tree. Then, E can be represented in a $n \times n$-matrix, where

$$a_{ij} = \begin{cases} 1 \rightarrow \{v_i, v_j\} \in R \\ 0 \rightarrow otherwise \end{cases} \tag{6}$$

The adjacency matrix servers as a basis for building the orchestration trees O_i^P of primary context providers P_i and associated complementary context providers $C_j \in \mathbb{C}_i^P$. It contains a separate row and column for every element $P_i \in \mathbb{P}$ and $C_j \in \mathbb{C}$. Each element in the matrix corresponds to a distinct tuple (P_i, C_j) or (C_j, C_k) from the cartesian products $\mathbb{P} \times \mathbb{C}$ and $\mathbb{C} \times \mathbb{C}$. In case there is a relation between P_i and C_j or C_j and C_k, the matrix coefficient is set to 1 at the corresponding position. All other coefficients remain at 0.

Figure 6 depicts an example of an adjacency matrix containing three primary context providers $P_{1,...,3}$ and five complementary context provider $C_{1,...,5}$. Each row represents a particular context provider where the matrix coefficients in each column indicate a relationship to the context provider given in each column in case its value is set to 1. Likewise, 0 indicates no relationship r between the context provider of the current row and the context provider of the current column. Primary context providers can not be connected among each other, i.e., primary context providers always have complementary context providers as successors. Complementary context providers in contrast can be connected among each other in a non-cyclic way.

	P_1	P_2	P_3	C_1	C_2	C_3	C_4	C_5
P_1	0	0	0	1	1	0	0	0
P_2	0	0	0	0	0	0	0	1
P_3	0	0	0	0	0	0	0	0
C_1	0	0	0	0	0	0	0	0
C_2	0	0	0	0	0	1	1	0
C_3	0	0	0	0	0	0	0	0
C_4	0	0	0	0	0	0	0	0
C_5	0	0	0	0	0	0	0	0

$$= \begin{pmatrix} 0&0&0&1&1&0&0&0 \\ 0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&1&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \end{pmatrix}$$

Fig. 6. Example of an adjacency tableau and corresponding matrix for eight context providers

As an example, let there be three primary context providers $P_{1,...,3}$ and five complementary context providers $C_{1,...,5}$ deployed within the framework. By analyzing the data descriptions exposed by each context provider, the adjacency matrix depicted in Figure 6 can be created based on the compatibility indicators computed by the orchestration framework. The first three rows represent the adjacent complementary context providers $C_j \in \mathbb{C}$ of P_1, P_2, and P_3 that take their context models as input. For instance, P_1 has the two complementary context providers C_1 and C_2 as direct successors that are able to refine and augment the context model delivered by P_1.

The values in row 4 show the direct successors of the first complementary context provider C_1. Since the matrix coefficients in the entire row are set to 0, there exist no relationships to other complementary context providers. C_2 in line 5 for instance has two adjacent context providers C_3 and C_4, which both do not have any further successors that take their context models as input for their acquisition tasks. By analyzing the given adjacency matrix, the orchestration trees O_i^P depicted in Figure 7 can be derived for each primary context provider $P_{1,...,3}$[12]. Those orchestration trees represent atomic units of context acquisition workflow schemes.

[12] The orchestration tree O_3^P for context provider P_3 can be considered a special case of an orchestration tree that only contains a root element and no further adjacent elements.

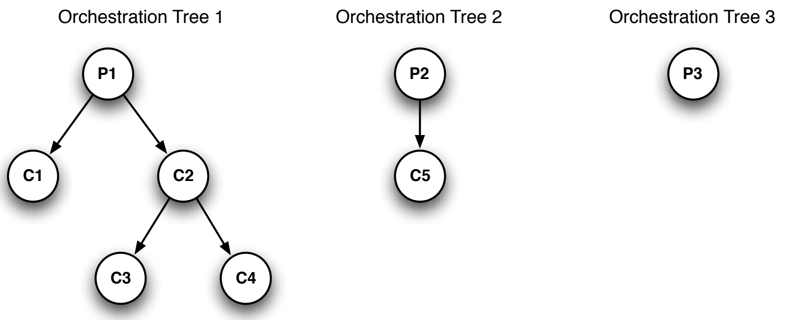

Fig. 7. Orchestration graphs derived from the adjacency matrix

3.5 Context Model Aggregation

The following paragraph provides an overview of the context acquisition workflows from a data-centric view and illustrates the process of building a global context model, the so-called *context configuration* from the context providers' context models.

Let M_i^P be the context model created by a primary context provider P_i, and M_j^C the context model created by a complementary context provider C_j. Since there is a 1:1 relation between a context provider and the context model it creates, we can define a projection $A_k \longmapsto M_k^A$ with $A_k \in \mathbb{A}$ between any context provider and its context model. A context model M is represented as an RDF graph consisting of a finite set of resources represented by their URIs and denoted as $\mathbb{U}$, a finite set of blank nodes $\mathbb{B}$, and a finite set of literals $\mathbb{L}$[13]. Those elements are the building blocks of an RDF statement which is represented as a triple $\mathcal{T} = (\mathcal{S}, \mathcal{P}, \mathcal{O})$ consisting of a *subject* $\mathcal{S}$ which is either a URI reference or a blank node, a *predicate* $\mathcal{P}$ which is a URI reference, and an *object* $\mathcal{O}$ that can either be a URI reference, a blank node, or a literal. A context model M itself can be defined as an element from the power set $\mathbb{M}$ that is defined as the cartesian product of the union of $\mathbb{U}$ and $\mathbb{B}$, $\mathbb{U}$, and the union of the sets $\mathbb{U}$, $\mathbb{B}$, and $\mathbb{L}$:

$$M \in \mathbb{M} \text{ where } \mathbb{M} := \mathcal{P}(\mathbb{U} \cup \mathbb{B} \times \mathbb{U} \times \mathbb{U} \cup \mathbb{L} \cup \mathbb{B}) \tag{7}$$

An orchestration tree O_i^P for a primary context provider P_i is considered as an atomic unit, which—when all acquisition tasks of the contained context providers have finished—provides an aggregated context model M_i^O that is composed of the context models of contained context providers. Since O_i^P denotes the orchestration tree of a primary context provider $P_i \in \mathbb{P}$, let M_i^O denote the compounded context model acquired by the context providers orchestrated within the orchestration tree O_i^P. As there is a 1:1 relationship between an orchestration tree $O_i^P \in \mathbb{O}$ and the compounded context model M_i^O acquired by its containing context providers, we can define the projection $O_i^P \longmapsto M_i^O$. Further, let $\mathbb{M}_i^C$ be the set of context models acquired by all $C_j \in \mathbb{C}_i^P$. Hence, the compound context model M_i^O of orchestration tree O_i^P is defined as

$$M_i^O := \{M_i^P\} \cup \bigcup_{j=1}^{|\mathbb{C}_i^P|} M_j^C \in \mathbb{M}_i^C \tag{8}$$

When the context models $M_k^A \in \mathbb{M}$ of all context providers being part of all orchestration trees $O_i^P \in \mathbb{O}$ were acquired successfully, the context configuration CC can be created. Therefore, let CC be the union

[13] For the purpose of this model, no distinction is made between plain and typed literals (cf. [54]).

of all compounded context models M_i^O of the orchestration trees O_i^P where $i = |\mathbb{O}|$. Then, we can define CC as the result of a function $f(x)$ that processes the compounded context models of all orchestration trees $O_i^P \in \mathbb{O}$ and infers additional information by applying reasoning techniques.

$$CC := f(M_i^O | i = 1 \dots |\mathbb{O}|) \tag{9}$$

The orchestration tree O_i^P of a primary context provider $P_i \in \mathbb{P}$ delivers an aggregated context model M_i^O that is compounded of the context models of the constituting context providers M_i^P and $\mathbb{M}_i^C$. The context configuration is then built by the context dispatcher by integrating, aggregating, and consolidating those compounded context models.

3.6 Processing Workflow

For processing contextual information represented as RDF graphs, we apply concepts from graph theory and distributed transaction management to maintain context accuracy, consistency, and completeness while guaranteeing computational efficiency in terms of required resources and processor load. An orchestration tree O_i^P is executed within a *context acquisition workflow* denoted as WF_i^P that serves as a workflow scheme for the execution of the contained context providers and monitors acquisition progress. At any point in time, multiple context acquisition workflows can be active while there is a 1:1:1-relationship between a primary context provider P_i, the corresponding orchestration tree O_i^P, and its workflow scheme WF_i^P. Every workflow scheme is considered an *atomic unit* where the containing context providers acquire their context models independently from context providers contained in other workflow schemes. This concepts resembles the idea of the *atomicity* and *isolation* properties defined in the ACID paradigm for database transactions [78]. In that sense, the context models acquired within the course of their corresponding acquisition workflows are considered as a single transaction where only the corresponding compound model $M_i^O \subseteq CC$ is updated as a whole in the context configuration CC.

Information about adjacent providers $C_j \in \mathbb{C}_i^P$ are requested from the orchestration framework that returns a context provider's direct descendants. Those complementary context providers are executed in self-contained, dedicated, and independent control threads called *context acquisition control threads* denoted as $PCT(C_j)$ that control and coordinate acquisition tasks and set the corresponding entries in the model manager when the control thread finishes, i.e., when a context update has been acquired. Context acquisition control threads are instantiated on demand in separate threads whenever the context of a preceding context provider has changed. The acquisition processes of the involved context providers are completely decoupled and independently executed in form of a *single atomic transaction* [78].

Algorithm 1 describes the relevant steps in the acquisition workflow carried out by the context dispatcher for processing and aggregating context descriptions to built a global context model. The context dispatcher regularly checks whether there are new context descriptions available. A context description D_k^A for a context provider A_k can be defined as the quadruple $D = (M, \rho, \sigma, \tau)$ where M represents the context model M_k^A of the corresponding context provider A_k, ρ and σ represent status information and a status code, and τ a time stamp when the context model was created. Therefore, a context description D_k^A can be considered a projection of a context model M_k^A for a given context provider A_k at a given time τ: $M_k^A \longmapsto D_k^A(\tau)$.

In case a new context description D_k^A is available, the context dispatcher retrieves the new context description and stores it in the model manager. For each context description D_k^A, the corresponding context provider A_k is requested from the registry. The context dispatcher then checks the context description D_k^A and calculates a unique value that reflect certain characteristics and the current status (see Table 2) of the corresponding context provider A_k. Therefore, we have defined some basic conditions derived from the requirements of the proposed mobile context framework (see Section 3.1). Each condition is assigned a unique value of the form 2^n where n refers to a certain condition. This allows to represent unique states by simply summarizing

Input: ContextDescription D_i
Output: -

```
while ContextDispatcher is running do
    if all acquisition workflows have finished then
        collect and aggregate context models $M_j^A$ where $1 \leq j \leq |\mathbb{A}|$ ;
        create ContextConfiguration $CC$ ;
        notify ReplicationManager ;
        reset ModelManager ;
    end
    if ContextDescriptionQueue contains new ContextDescriptions then
        forall the ContextDescriptions $D_i \in$ ContextDescriptionQueue do
            obtain context description $D_i$ from ContextDescriptionQueue ;
            retrieve corresponding context provider $A_j \leftrightarrow D_i$ ;
            calculate status value for $D_i$ ;
            if $A_j \in \mathbb{P}$ and $M_j^A \notin$ ModelManager then
                store $M_j^A$ in ModelManger ;
                if hasSuccessor($A_j$) then
                    $WF_j^A \leftarrow$ new ContextAcquisitionWorkflowThread ;
                    forall the Successors $C_k \in \mathbb{C}_j^A$ do
                        $PCT(C_k) \leftarrow$ new ContextAcquisitionControlThread ;
                        $PCT(C_k)$.start() ;
                    end
                end
            end
            else if $A_j \in \mathbb{C}$ and $A_j \in WF_x^P$ then
                store $M_j^A$ in ModelManger ;
                if hasSuccessor($A_j$) then
                    forall the Successors $C_k \in \mathbb{C}_j^A$ do
                        $PCT(C_k) \leftarrow$ new ContextAcquisitionControlThread ;
                        $PCT(C_k)$.start() ;
                    end
                end
            end
            else if $A_j \in \mathbb{P}$ and $M_j^A \in$ ModelManager and exists($WF_x^P, x \neq j$) then
                replace $M_j^A$ in ModelManager ;
                if hasSuccessor($A_j$) then
                    $WF_j^A \leftarrow$ new ContextAcquisitionWorkflowThread ;
                    forall the Successors $C_k \in \mathbb{C}_j^A$ do
                        $PCT(C_k) \leftarrow$ new ContextAcquisitionControlThread ;
                        $PCT(C_k)$.start() ;
                    end
                end
            end
        end
    end
end
```

Algorithm 1: Algorithm for processing context descriptions in pseudo code

Table 2. Conditions for determining a context provider's state

Condition	Value	Description
A	2^0	checks whether a context model has been sent by the context provider and thus is already contained in the model manager.
B	2^1	indicates whether the context provider has adjacent complementary context providers that are orchestrated within a context acquisition workflow.
C	2^2	indicates whether an acquisition workflow thread that coordinates the execution of the context providers orchestrated within that workflow has already been initiated for the given primary context provider.
D	2^3	indicates whether the given complementary context provider is part of an already instantiated acquisition workflow.
E	2^4	indicates whether the corresponding entry for a given context provider in the model manager is locked, meaning that no context model propagation is possible until the corresponding workflow thread has finished.
F	2^5	checks whether the given context provider is an active, that is, primary context provider.
G	2^6	checks whether the corresponding context acquisition workflow or control thread has been finished depending on the context provider's type.

the distinct property values that match for a given provider. For instance, if a context provider is primary typed and has adjacent providers but no acquisition workflow has been initiated yet, its status value is 34 $(0 * 2^0 + 1 * 2^1 + 0 * 2^2 + 0 * 2^3 + 0 * 2^4 + 1 * 2^5 + 0 * 2^6)$. If a condition evaluates to true, the corresponding value is added to the provider's status value that determines how a context provider as well as its context model and adjacent providers are to be processed. In accordance to this value, the corresponding processing steps (see Table 3) will be executed; if the context description D_k^A was sent by a primary context provider P_i whose context model M_i^P has not been stored in the model manager yet, it will be extracted from the context description D_i^P and stored in the model manager. If P_i has complementary context providers $C_j \in \mathbb{C}_i^P, 1 \leq i \leq \left|\mathbb{C}_i^P\right|$ associated to it in an orchestration tree O_i^P, a new instance of an context acquisition workflow WF_i^P is created. The direct descendants of P_i (cf. the adjacency matrix depicted in Figure 6) are requested from the orchestration framework and the context acquisition control threads $PCT(C_j)$ which executes and controls the acquisition tasks will be created and initiated.

If A_k is a primary context provider with $A_k \longmapsto P_i$ has already committed its context model M_i^P but has no direct descendants, that is, no relations to complementary context providers and there are other context acquisition workflows WF_k^P where $i \neq k$ running in parallel, its already stored context model M_i^P can be replaced by the new one in the model manager without violating consistency requirements. The same applies to primary context providers P_l whose context acquisition workflows are finished, that is, all associated complementary context providers $C_j \in \mathbb{C}_i^P, 1 \leq k \leq \left|\mathbb{C}_x^P\right|$ have delivered their context models, but other context acquisition workflows are still running. In this case, a new context acquisition workflow WF_l^P will be initiated and the previous context models can be replaced by new (updated) models.

In case a context acquisition workflow WF_i^A has finished, it notifies the workflow manager, which then checks whether there exist further context acquisition workflows WF_j^A where $j \neq k$. If all acquisition workflows have finished, the workflow manager notifies the context dispatcher that a new context configuration CC can be built. The context dispatcher then starts the merging and consolidation process by collecting all updated and unaltered context models M_k^A. Updated context models are retrieved from the model manager whereas unaltered context models are retrieved from the context description queue. Those models will then be aggregated (merged) where consolidation and reasoning rules are applied by a lightweight rule reasoner. After the context configuration is built, it will be forwarded to the replication manager to initiate the replication tasks of the deployed data providers.

Table 3. Excerpt of the calculation and processing matrix for context providers including descriptions and rules

Condition	A	B	C	D	E	F	G		
Value	2^0	2^1	2^2	2^3	2^4	2^5	2^6	Sum	Description & Rule
#1	0	1	0	0	0	1	0	**34**	This is a primary context provider with adjacent context providers but no context acquisition workflow has been created yet; **rule**: *create a new context acquisition workflow for the given context provider*
#2	0	1	0	1	0	0	0	**10**	This complementary context provider is part of a context acquisition workflow and has adjacent complementary context providers attached to it that further augment its context model. **rule**: *retrieve complementary context providers and initiate acquisition tasks*
#3	0	1	0	0	0	0	0	**2**	This is a complementary context provider that has further complementary context providers attached to it as successors that take its context model as input data for their acquisition tasks. **rule**: *retrieve complementary context providers and initiate acquisition tasks*
#4	0	0	0	1	0	0	0	**8**	This is a complementary context provider that is part of a context acquisition workflow. It has no adjacent providers attached to it and its context model isn't stored in the model manager yet. **rule**: *store model in model manager*
#5	1	1	1	0	1	1	0	**55**	This is the status of a primary context provider with adjacent complementary context providers that has already delivered a context model stored in the model manager. The corresponding context acquisition workflow is still in progress since not every complementary context provider contained in the corresponding orchestration tree has finished its acquisition task yet. **rule**: *refuse the context provider's updated context model*
#6	1	1	0	0	1	1	1	**115**	A primary context provider whose context acquisition workflow has already finished sends an updated context model. Since there are other acquisition workflows currently running, its updated context model can be accepted and a new context acquisition workflow can be initiated. **rule**: *reset corresponding entries in model manager, store context model and initiate new context acquisition workflow*
#7	1	0	0	0	0	1	1	**97**	A primary context provider without adjacent complementary context providers as sent an updated context model. **rule**: *update corresponding entires and store context model in model manager*
...	...	...	...	...	...	...	...	...	...

4 Implementation

4.1 The Google Android Platform

Android is an open software stack for mobile devices consisting of a Linux-based operating system kernel, a middleware, key applications, and a set of API libraries for accessing native system functions and natively deployed sensors [79]. It was released in 2007 under the auspices of the Open Handset Alliance[14], a coalition of 79 technology and mobile companies[15] and has gained noticeable attentions since it represents a new generation of mobile application development platform and software development kit [80]. One of the key architectural features of the Android platform is the open communication infrastructure where application can reuse functionality and exchange data in a controlled and flexible way. Android includes a highly specialized virtual machine, called Dalvik VM that was designed for low-powered handheld devices as those devices "*lag behind their desktop counterparts in memory and speed by eight to ten years*" [81]. In contrast to conventional Java virtual machines which use a stack-based architecture for data storage, the Davlik VM is built upon a register-based architecture and transforms generated Java classes into a performance and memory optimized Dalvik specific file format.

Android includes a lightweight and powerful relational SQLite database that offers dedicated libraries for its utilization within application and services to store data persistently. Such data can be shared across applications in a controlled manner using Android's inter-process communication model. In contrast to the hard-wired application models of desktop operating systems, Android offers an *intend-based* application model that allows an application to specify a certain kind of functionality it requires for data processing where the operating system chooses the application that best matches. Access to core-system libraries and functions is offered via native APIs that can be used by both native and non-native applications. Android employs an equal, non-prioritized execution policy for native and non-native applications that are executed in the same runtime, and offers a complete multithreading environment where applications can place extensive computational tasks in separate threads [82].

This environment brings the following options for deploying a framework that provides the desired functionality necessary for an efficient acquisition, management, storage, and dissemination of context information:

– A service can constantly scan the environment for exploitable context sensors and ubiquitous devices and integrate them dynamically in a context framework.
– A context framework can be realized as a background service that is executed transparently and autonomously from the user while not affecting any of their tasks nor requiring explicit user attention.
– The concept of content providers can be deployed for the controlled and fine-grained provision of replicated data as well as for the provision and utilization of context data.
– Broadcast receivers can notify the user about relevant events in a non-disruptive manner while providing specialized configuration views in case explicit user inputs are necessary, e.g, for integrating a recently discovered ubiquitous sensor.
– The open and uniform architecture of Android allows for a broad utilization of locally deployed sensors where native APIs offer uniform and controlled access to those peripheral hardware.

4.2 Context and Data Providers

All the basic data of context and data providers such as name, identifier, data description, and status is held in the `AbstractBasicProvider` class and specified by the `IBasicProvider` interface. The `AbstractContextProvider`

[14] Open Handset Alliance: `http://www.openhandsetalliance.com/`
[15] see `http://www.openhandsetalliance.com/oha_faq.html`

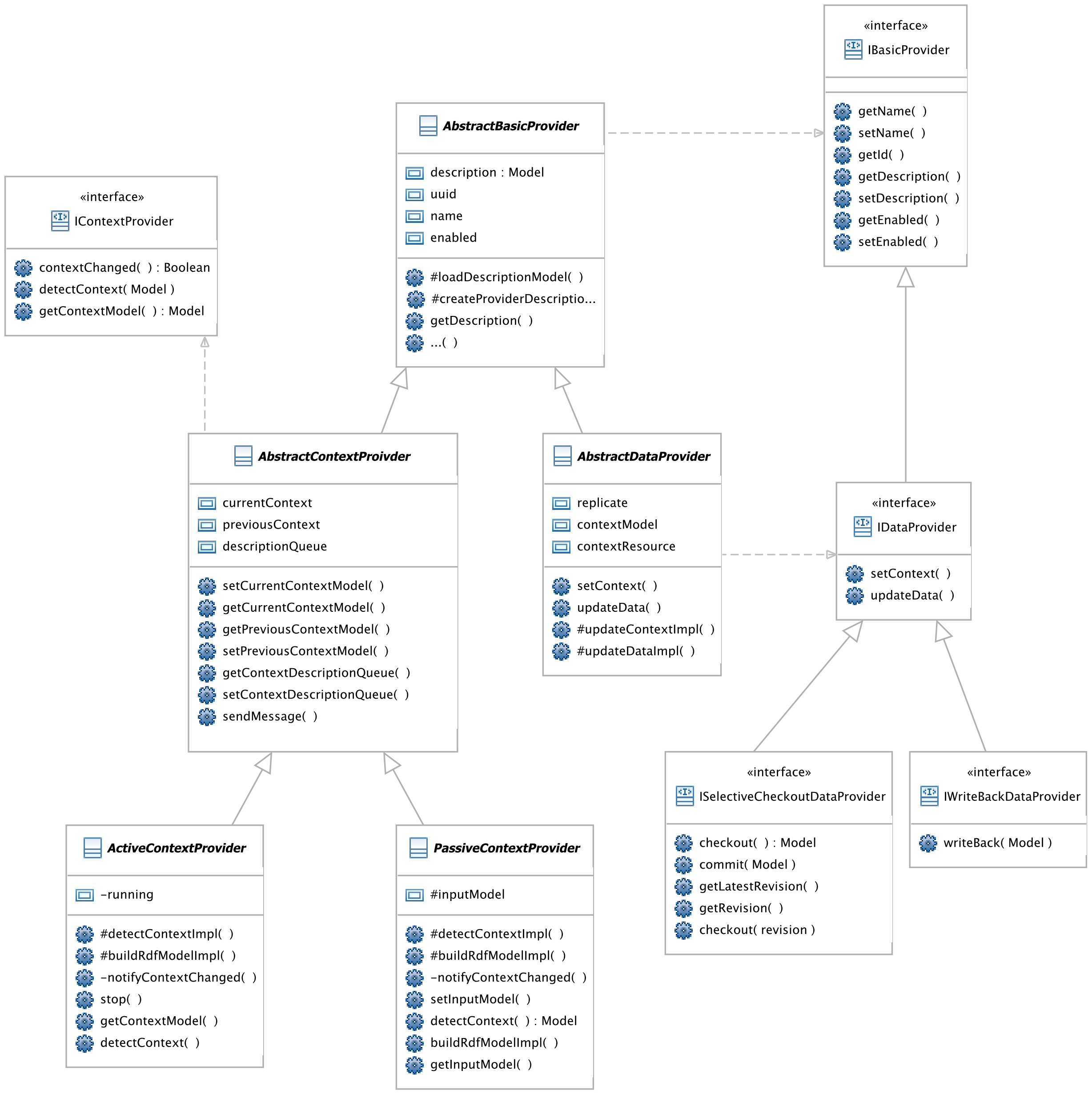

Fig. 8. Structured class diagram for context and data providers

class implements methods regarding the communication with the framework as well as for the creation and management of the context providers' models. The method declarations defined in the `IContextProvider` interface represent a minimum functionality a context provider must expose and are implemented in the sub classes or delegated to concrete context provider implementations (e.g. `contextChanged()` method). Primary context providers must be inherited from the abstract class `ActiveContextProvider` whereas complementary context provider are deduced from the abstract class `PassiveContextProvider`. Both classes are by itself subclasses of `AbstractContextProvider` and implement specific behavior for each provider type. Both methods specify the abstract methods `detectContextImpl()` which contains the concrete acquisition logic and `buildRdfModelImpl()` which holds the code for building the RDF context models. These abstract methods must be implemented in each context provider individually. Primary context providers acquire context-relevant data independently from any requests in a proactive and autonomous manner whereby they run in a separate thread where their super class includes the infrastructure for their control. Passive context providers initiate their context acquisition tasks when they were notified by the `ContextDispatcher` about the availability of a required context input model.

The generic functionality of data providers are implemented in the abstract class `AbstractDataProvider` that is a subclass of `AbstractBasicProvider`. It contains member variables for storing the context configuration and specifies the two abstract method declarations `updateContextImpl()` and `updateDataImpl()` that must be implemented by individual data provider. The `updateContextImpl()`-method allows concrete data providers to implement an individual processing logic for analyzing the context configuration. The actual replication code must be implemented in the `updateDataImpl()`-method, which is called by the replication manager. We decided to decouple the replication process so that a data provider can perform additional processing before it initiates its replication workflow. The two interfaces `ISelectiveCheckoutDataProvider` and `IWriteBackDataProvider` contain method declarations for more complex data replication tasks, e.g., for replicating only subsets of data sets [83] or bidirectional synchronization of replicated data.

So far, a number of concrete context providers were implemented:

- `GPSContextProvider` is an active (primary) context provider that utilizes the device's internal GPS module for retrieving GPS coordinates. It can further be configured to use a WiFi-based location ascertainment method. The GPS context provider returns its data as latitude and longitude coordinates using the W3C WGS84 vocabulary[16].
- `GeonamesContextProvider` is a passive (complementary) context provider that is able to interpret any geographical coordinates expressed using the W3C WGS84 vocabulary and requests the *GeoNames.org* web service in order to resolve the GPS coordinates to a concrete geographical location identified by a GeoNames `URI`. Therefore it is an ideal complement to the `GPSContextProvider`, whose output is enriched with logical location concepts in addition to physical GPS coordinates; however, it can in general be used to process any GPS coordinate information regardless of its source.
- `GoogleCalendarContextProvider` is an active (primary) context provider that retrieves appointments from the user's calendar within a configurable time span (e.g., 72 hours) starting from the current time. For all events, a set of metadata are extracted and added to the context model, including the event's names and start/end times, location descriptions associated to the events, and information about each event's attendees. It uses terms from the popular Dublin Core vocabulary[17] as well as the NEPOMUK calendar ontology[18] for modeling extracted metadata.
- `LocalAddressbookContextProvider` is an active (primary) context provider that exposes the contacts found in the use's local address book to make them available for relevant data providers. Properties like name, email addresses, phone numbers, postal addresses, and instant messaging contact IDs are extracted and inserted into the context model, where they are expressed using the popular FOAF vocabulary[19].

[16] `http://www.w3.org/2003/01/geo/`
[17] `http://purl.org/dc/terms/`
[18] `http://www.semanticdesktop.org/ontologies/2007/04/02/ncal/`
[19] `http://xmlns.com/foaf/0.1/`

```
// handle location updates
// (called by Android location manager)
public void onLocationChanged(Location location) {
   this.location = location;
}

@Override
// detect context model
// (called by ActiveContextProvider)
protected void detectContextImpl() {
   // read data from the location object (location object is provided by )
   double lat = this.location.getLatitude();
   double lon = this.location.getLongitude();
   String gpsProviderType = this.location.getProvider();


   // clear context model
   this.getCurrentContextModel().removeAll();

   // create new context resources
   Resource context = getCurrentContextModel().createResource();
   context.addProperty(RDF.type, Vocabulary.Mobisem.CONTEXT);
   Resource location = getCurrentContextModel().createResource();
   context.addProperty(Vocabulary.Mobisem.CURRENT_LOCATION, location);

   // attach coordinates to location resource
   location.addProperty(Vocabulary.Geo.LAT, lat);
   location.addProperty(Vocabulary.Geo.LONG, lon);
   if (gpsProviderType != null)
      location.addProperty(Vocabulary.Mobisem.LOCATION_PROVIDER_TYPE, gpsProviderType);
}
```

Fig. 9. Code snippet of `GPSContextProvider`, converting location data into an RDF-based context model

Typically, context providers use any kind of input data and convert it to an RDF model that represents this data in machine-processable form. Ideally, context providers use standardized, publicly available, and community-accepted vocabularies for this purpose in order to enable interoperability between components from different manufacturers. As an example, Figure 11 shows how the `GPSContextProvider` builds a simple RDF model representing containing GPS coordinates obtained from the device's location service. In this example, the context provider registers itself as a location listener in the Android location subsystem; in consequence, the `onLocationChanged()` method is called every time the device's location changes. In this case the context provider buffers the most recent `Location` object. In turn, the `detectContextImpl()` method is regularly invoked by the `ActiveContextProvider` class. It creates a resource of type `mobisem:Context` and uses the property `mobisem:currentLocation` to associate the context with another resource, representing the actual location. This second resource is described with two properties for the latitude and longitude values, as well as one property for the type of the used location provider, if applicable.

In our motivating example (cf. Section 1.2) a component that provides location information can be used to update information while John is on travel. It can be used for a variety of purposes, including the provisioning of general information about his location. It could be combined with information about John's personal interests in order to suggest points to visit in his spare time, or it could be combined with his calendar information in order to suggest public transport routes to the address where his next meeting takes place.

```
@prefix mobisem: <http://www.mobisem.org/2009/01/context#> .
@prefix geo:     <http://www.w3.org/2003/01/geo/wgs84_pos#> .

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

Fig. 10. Context description as returned by the `GPSContextProvider` (Turtle notation)

Data providers are built upon the abstract classes and interfaces of context provides, `AbstractBasicProvider` and `IBasicProvider`; they actually expose very generic and common interfaces for managing their basic data as depicted in Figure 8. Data providers are responsible for handling all data replication tasks. They retrieve the context configuration from the replication manager which signals them to initiate their replication tasks based on their analysis of the relevant entities contained in the context configuration. Each data provider wraps a specific data source and replicates data to the local SQLite data base deployed on the Android platform. The framework does not expose any restrictions on the data sources to be wrapped by data providers; they can wrap a remote file system object, a remote data base, web services and web applications respectively, or generate data themselves. 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 in the triple store. The only requirement exposed is that replicated data must be available as RDF. The following concrete data providers were implemented so far:

- `DBpediaLocationDataProvider` is able to process geo coordinates as well as Geonames ids of location resources (which are taken from the global context model). For all resources of type `geonames:Feature`, the data provider retrieves relevant location information from DBpedia.org by querying the DBpedia SPARQL endpoint for relevant resources, based on their label. For all returned resources, descriptive triples are stored on the mobile device.
- `MobiSemDataProvider` is the default data provider for replicating data from a (versioned) MobiSem repository [83] to the mobile device. Through this client-/server-based infrastructure, any data source that can be expressed as RDF triples (e.g., file systems, relational data bases, etc.) can be exposed as versioned RDF graph repository, and can be replicated to a mobile device. For this purpose, a REST-style protocol has been developed that is used to specify additional metadata (like context information or the current graph version) in the form of additional HTTP headers. In addition to serving and versioning RDF graphs, the MobiSem repository is able to perform ranking and selection of RDF triples based on a context model that may be sent from the client alongside a checkout request. In this case, only parts of RDF graphs are replicated to the mobile device, saving transmission and storage capacity [83]. The `SelectiveHomebaseDataProvider`, in addition to providing support for such partial replicas, enables the client to write changes to replicated data back to the remote repository, which then takes care of merging and conflict detection [83].
- `SindiceTopResultsDataProvider` extracts possible search terms from the global context model by analyzing all literals found therein. It tokenizes the literals and constructs a list of most frequent keywords, which are in turn sent to the `Sindice.com` Semantic Web indexing service. From there it retrieves the top results and converts them into an RDF model, which is then stored on the mobile device. Optionally, the data provider is able to retrieve more information about the top results by de-referencing the resource URIs, according to Linked Data principles. Any data retrieved via this option is also stored on the mobile device's data store.

To illustrate the basic functionality of a data provider, we describe the `DBpediaLocationDataProvider` in more detail. It consists of two main methods: the first one, `updateContextImpl()`, is called by the data provider's implementation base class (`AbstractDataProvider`) whenever it receives an updated context model from the context dispatcher. In this method, the data provider analyzes the context model and

```
// analyze the current context model (stored in this.contextModel)
@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 statements of these features
      StmtIterator si2 = this.contextModel.listStatements(featureResource, null, (RDFNode) null);
      while(si2.hasNext()) {
         // check if a label property is attached
         Statement s = si2.nextStatement();
         if( s.getObject().isLiteral() && ! this.currentResourceLabels.contains(s.getString())) {
            this.currentResourceLabels.add(s.getString());
         }
      }
   }
}

// update data from the remove 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("{" +
         "?concept rdfs:label \"" + featureLabel + "\"@en . " +
         "?concept rdf:type dbpedia-owl:Place . " +
      "} UNION \n");
   }
   queryBuffer.append("{} }");

   // send query to DBpedia
   HttpClient client = new DefaultHttpClient();
   String url = this.serviceUri + "?query=" + URLEncoder.encode(queryBuffer.toString());
   HttpGet method = new HttpGet(url);
   method.addHeader("Accept", "text/plain");   // accept only N-TRIPLES
   try {
      // execute request
      HttpResponse response = client.execute(method);

      // read model into targetModel (which is then further processed by the context framework)
      targetModel.read(response.getEntity().getContent(), "N-TRIPLE");
   }
   catch (Exception e) {
      // error handling
   }
}
```

Fig. 11. Code snippet of `DBpediaLocationDataProvider`, querying DBpedia for data about location resources

iterates over all resources of type `geonames:Feature`; i.e., all logical locations. For all these resources it extracts the labels and stores them in a list of resource labels. In the second method, `updateDataImpl()` (which is again called by the implementation base class whenever the data provider is requested to update data) a SPARQL `DESCRIBE` query is constructed that incorporates all resource labels (in this example we assume the user is interested only in information in English). Additionally it restricts the query to resources of type `dbpedia-owl:Place`. This query is sent to the DBpedia query endpoint, and the results are read into the `targetModel` variable, which is subsequently processed by the context dispatcher.

Since DBpedia provides a large amount of information about locations (including names, descriptions, geographic and statistical data, as well as links to related persons, buildings, and events) such a data provider can be of great use in the case of our motivating example (cf. Section 1.2). John will be enabled to browse relevant information about the places that he will visit during his next trip without the need to actually search for it. By matching location information with John's interests (which could be derived from his Web browsing history, or tags he has used to annotate his photos) the system could recommend points of interests to him; by combining location data with publicly available personal profiles (e.g., from FOAF data) John could be notified of people that live in cities John is going to visit, and based on their interests it could suggest meetings with potential customers or cooperation partners.

4.3 Utilization and Data Provision

Replicated data are provided to other applications through an RDF content provider which is automatically instantiated and started by the Android when the provider is first requested. Content providers are treated as *singleton classes* but can communicate with multiple `ContentResolver`-objects across system processes and applications. This is the reason why the operations a content provider offers must be implemented in a thread-safe manner to avoid side-effects and data inconsistencies.

Content providers usually expose their data using a table-based data model, where columns represent specific types of data and rows a single record. Each data set a content provider exposes is identified via a unique and public-accessible URI called the *content URI* that follows a well-defined scheme. A content URI always starts with the `content://` scheme to signal the system that the data is exposed by a content provider followed by an authority part that specifies the content provider, e.g., `org.mobisem.rdfprovider`, a path segment that can be used to address specific types of data in case a content provider exposes multiple data sets, e.g., `/binary` for a binary serialization, and an optional ID segment that allows for addressing specific records, e.g, the URI or name of a data replica.

Figure 12 provides a example of storing an RDF graph to the database using the `RDFContentProvider`'s `insert()`-method, which supports three different types of insertions: (1) inserting a complete graph, (2) adding a partial graph to an existing one, and (3) adding single triples. The URI determines which insert operation is to be performed; for instance, by specifying the URI `content://org.mobisem.rdfprovider/graphs`, a complete RDF graph is to be stored in the database. If only a partial graph is to be added to an existing graph, the `content://org.mobisem.rdfprovider/graph` URI can be used instead. In case a graph already exists under the given URI, the existing graph will be overwritten. We assume that the new graph is an updated replica that results from a context update. All the data to be stored in the database must be wrapped in a μJena model and serialized as a byte array.

Line 2 stores the graph's URI (indicated by the `GRAPH_NAME` variable) in the `ContentValues` using the name key (indicated by the `RDFContentProvider.ContentValues.NAME` constant). In the next line, the RDF graph is serialized as a byte array before it is added to the `ContentValues`. Finally, the `insert()`-method of the `RDFContentProvider` is requested using the URI for adding a complete graph together with the graph data stored in the `ContentValues`-object. In case storage was successful, the graph's *URI* is returned.

```
ContentValues values = new ContentValues();
values.put(RDFContentProvider.ContentValues.NAME, GRAPH_NAME);
values.put(RDFContentProvider.ContentValues.MODEL, serializeModelAsByteArray(model));
URI uri = this.mResolver.insert(RDFContentProvider.CONTENT_URI_GRAPH, values);
```

Fig. 12. Example of storing an RDF graph using the `RDFContentProvider`'s `insert()`-method

The `query()`-method allows to query for data replicas stored in the local database. Depending on the content URI, the `RDFContentProvider` either returns a collection of all data replicas' names together with the amount of triples included in each graph, an N-Triple-based representation of the triples contained in a specific replica, or a serialized μJena model of a replica that can be parsed for further processing. Figure 13 depicts a code fragment for retrieving a byte array serialized data replica that can directly transformed to an in-memory RDF model.

```
1  Cursor c = this.mResolver.query(
2     RDFContentProvider.CONTENT_URI_BINARY, null, REPLICA_UUID, null, null);
3  assertTrue(c.moveToFirst());
4  Model m = transformToRdfModel(c.getBlob(0));
```

Fig. 13. Example of transforming a data replica to a workable in-memory RDF model

The data replica retrieved from the triple store is stored in a cursor-object that allows for iterating over the data contained in the result set. Therefore, the cursor is moved to the first entry (line 3) and the byte array-based serialization of the requested data replica is passed to a transformation method that parses the byte stream and creates an in-memory model of the data replica (line 4).

In general, the `RDFContentProvider` also allows for adding and removing triples (update)[20] as well as deleting all replicas stored in the database, a specific replica, or all the triples of a specific replica's resource.

5 Conclusion and Future Work

In this chapter, we have introduced the concepts of Semantic Web-enhanced context-aware computing for mobile systems and presented the MobiSem context framework that provides an infrastructure for intelligently assisting mobile users by selectively replicating RDF data from remote data sources to a mobile device according to their current and future information needs and the different contexts they are operating in. Since research in context-aware computing and mobile systems shows that context and context-awareness should be central parts of future mobile information systems, we drafted an application scenario that outlines how Semantic Web-enhanced context-aware computing can support mobile users in fulfilling their information needs that also serves as a motivating example.

This chapter provided an overview on how context is used in information system and discussed problems and limitations of current context-aware computing approaches in information systems. Areas were presented where the introduction of concepts and technologies from the Semantic Web can make substantial contributions in representing, processing, and managing contextual information. For this purpose, concepts from graph theory, distributed transaction management, and the Semantic Web were adopted in order to

[20] Updates to RDF graphs usually consist of two sets: one set that contains the triples to be added to an RDF graph and another set that contains the triples to be removed from a graph.

demonstrate that a synthesis of those fields can serve as an architectural infrastructure for the efficient acquisition, management, and processing of contextual information directly on a mobile device. In consequence, dependencies to external systems can be reduced, and security and privacy issues are reflected since private data do not need to be transferred outside the mobile system.

The proposed architecture allows for a controlled acquisition, aggregation, and consolidation of contextual information taking into account mobile operating system peculiarities. At the same time it guarantees consistency, accurateness, and completeness among contextual data and acquisition workflows. It employs a loose coupling between context acquisition and data provisioning components, which is gained by applying semantic technologies (data models, vocabularies, inference) to interpret and process context information. Practical insights into implementation details of context and data providers were discussed, and concrete examples were shown on how context-relevant data can be acquired from locally deployed sensors, represented using semantic vocabularies, and utilized by external applications.

Although this framework demonstrates that semantic technologies can make substantial contributions in realizing a mobile context-aware infrastructure assisting users in satisfying their mobile information needs, there are still some open issues that need to be addressed in future research. For instance, the integration of dynamically discovered context sources is a challenge most context-management frameworks face, especially in mobile and ubiquitous environments. We therefore suggest to direct future research towards finding and integrating such sources and develop common interfaces and protocols for this purpose. For this, Semantic Web technologies are a candidate because they have been proven to be an appropriate infrastructure for the explicit representation and exchange of data semantics. Those semantics can be used not only for describing contextual information, but also for sensor interfaces to facilitate discovery and integration in running systems, as well as the exchange of contextual information. 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.

Further, our framework could be extended to include feedback loops that would allow for adjusting context acquisition and aggregation tasks according to data provisioning needs. Further, more sophisticated reasoning capabilities are missing, which we plan to address in future work. A context framework as such should be aware of the operating system conditions to adapt itself to changing conditions. Finally, support for efficient and complex inferencing techniques is still an open issue on mobile systems and should be addressed in future research.

References

1. Timothy Sohn, Kevin A. Li, William G. Griswold, and James D. Hollan. A diary study of mobile information needs. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 433–442, New York, NY, USA, 2008. ACM.
2. Maryam Kamvar and Shumeet Baluja. A large scale study of wireless search behavior: Google mobile search. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 701–709, New York, NY, USA, 2006. ACM.
3. Anind K. Dey. Understanding and Using Context. *Personal Ubiquitous Comput.*, 5(1):4–7, 2001.
4. B.N. Schilit and M.M. Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22–32, Sep/Oct 1994.
5. Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, London, UK, 1999. Springer-Verlag.
6. Christos B. Anagnostopoulos, Athanasios Tsounis, and Stathes Hadjiefthymiades. Context awareness in mobile computing environments. *Wirel. Pers. Commun.*, 42(3):445–464, 2007.
7. Anind K. Dey. *Providing architectural support for building context-aware applications.* PhD thesis, Georgia Institute of Technology, 2000.

8. Damien Fournier, Sonia Ben Mokhtar, Nikolaos Georgantas, and Valérie Issarny. Towards ad hoc contextual services for pervasive computing. In *MW4SOC '06: Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW4SOC 2006)*, pages 36–41, New York, NY, USA, 2006. ACM.
9. Sung Woo Kim, Sang Hyun Park, JungBong Lee, Young Kyu Jin, Hyun-Mi Park, Amy Chung, SeungEok Choi, and Woo Sik Choi. Sensible appliances: applying context-awareness to appliance design. *Personal Ubiquitous Comput.*, 8(3-4):184–191, 2004.
10. G. Biegel and V. Cahill. A Framework for Developing Mobile, Context-aware Applications. pages 361–365, March 2004.
11. Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen. There is More to Context than Location. *Computers and Graphics*, 23:893–901, 1998.
12. M. Kaenampornpan and B. Ba Ay. An Intergrated Context Model: Bringing Activity to Context. In *Workshop on Advanced Context Modelling, Reasoning and Management - UbiComp*, 2004.
13. Hong-Siang 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.
14. N. S. Ryan, J. Pascoe, and D. R. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications in Archaeology 1997*, British Archaeological Reports, Oxford, October 1998. Tempus Reparatum.
15. Anind K. Dey, Gregory D. Abowd, Mike Pinkerton, and Andrew Wood. Cyberdesk: A framework for providing self-integrating ubiquitous software services. In *ACM Symposium on User Interface Software and Technology*, pages 75–76, 1997.
16. Anind K. Dey. Context-aware computing: The cyberdesk project. In *AAAI 1998 Spring Symposium on Intelligent Environments*, pages 51–54, Palo Alto, 1998. AAAI Press.
17. Richard Hull, Philip Neaves, and James Bedford-roberts. Towards situated computing. In *In Proceedings of The First International Symposium on Wearable Computers*, pages 146–153, 1997.
18. P. J. Brown. The stick-e document: a framework for creating context-aware applications. In *Proceedings of EP'96, Palo Alto*, pages 259–272. also published in it EP–odd, January 1996.
19. Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277, 2007.
20. 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.
21. J. Euzenat, J. Pierson, and F. Ramparany. Dynamic Context Management for Pervasive Applications. *The Knowledge Engineering Review*, 23(1):21–49, 2008.
22. Paul Dourish. What We Talk About When We Talk About Context. *Personal Ubiquitous Comput.*, 8(1):19–30, 2004.
23. 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.
24. 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.
25. Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In *Pervasive '02: Proceedings of the First International Conference on Pervasive Computing*, pages 167–180, London, UK, 2002. Springer-Verlag.
26. P. Korpipaa, J. Mantyjarvi, J. Kela, H. Keranen, and E.J. Malm. Managing Context Information in Mobile Devices. *Pervasive Computing, IEEE*, 2(3):42–51, 2003.
27. 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.
28. Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2):97–166, 2001.
29. 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.
30. S. Boehm, J. Koolwaaij, M. Luther, B. Souville, M. Wagner, and M. Wibbels. Introducing IYOUIT. *The Semantic Web - ISWC 2008*, pages 804–817, 2008.
31. K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for Distributed Context-Aware Systems. In *OTM Conferences (1)*, 2005.

32. C. Huebscher and A. McCann. An Adaptive Middleware Framework for Context-aware Applications. *Personal Ubiquitous Comput.*, 10(1):12–20, 2005.
33. 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.
34. Ivo Widjaja and Sandrine Balbo. Spheres of role in context-awareness. In *OZCHI '05: Proceedings of the 17th Australia conference on Computer-Human Interaction*, pages 1–4, Narrabundah, Australia, Australia, 2005. Computer-Human Interaction Special Interest Group (CHISIG) of Australia.
35. Nicholas A. Bradley and Mark D. Dunlop. Toward a multidisciplinary model of context to support context-aware computing. *Hum.-Comput. Interact.*, 20(4):403–446, 2005.
36. Thomas Strang and Claudia L. Popien. A context modeling survey. In *UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 31–41, Nottingham, September 2004.
37. Jadwiga Indulska and Peter Sutton. Location management in pervasive systems. In *ACSW Frontiers '03: Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003*, pages 143–151, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.
38. Fei Xia, Alex V. Yakovlev, Ian G. Clark, and Delong Shang. Data communication in systems with heterogeneous timing. *IEEE Micro*, 22:58–69, November 2002.
39. M. Wojciechowski and J. Xiong. Towards an open context infrastructure. *Proceedings of the Workshop on Context Awareness for Proactive Systems (CAPS'06)*, pages 125–136, 2006.
40. Harry Chen. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems.* PhD thesis, University of Maryland, Baltimore County, December 2004.
41. Harry Chen and Anupam Joshi. An Ontology for Context-Aware Pervasive Computing Environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, 18(3):197–207, May 2004.
42. Marko Luther, Yusuke Fukazawa, Matthias Wagner, and Shoji Kurakake. Situational reasoning for task-oriented mobile service recommendation. *The Knowledge Engineering Review*, 23(1):7–19, 2008.
43. Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger. Context-awareness on mobile devices - the hydrogen approach. In *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9*, page 292.1, Washington, DC, USA, 2003. IEEE Computer Society.
44. Guoray Cai and Yinkun Xue. Activity-oriented context-aware adaptation assisting mobile geo-spatial activities. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, pages 354–356, New York, NY, USA, 2006. ACM.
45. Bonnie A. Nardi, editor. *Context and consciousness: activity theory and human-computer interaction.* Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.
46. Kari Kuutti. *Activity theory as a potential framework for human-computer interaction research*, pages 17–44. Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.
47. Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artif. Intell.*, 86(2):269–357, 1996.
48. Janice Redish and Dennis Wixon. The human-computer interaction handbook. chapter Task analysis, pages 922–940. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 2003.
49. Dan Diaper. *Task Analysis for Human-Computer Interaction.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1990.
50. Andrew Carton, Siobhan Clarke, Aline Senart, and Vinny Cahill. Aspect-oriented model-driven development for mobile context-aware computing. In *SEPCASE '07: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments*, page 5, Washington, DC, USA, 2007. IEEE Computer Society.
51. Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
52. T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax (RFC 3986).* Network Working Group, January 2005.
53. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616).* Network Working Group, 1999.
54. Graham Klyne and Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax (W3C Recommendation 10 February 2004).* World Wide Web Consortium, 2004.
55. David Becket. *RDF/XML Syntax Specification (W3C Recommendation 10 February 2004).* World Wide Web Consortium, 2004.
56. David Beckett and Tim Berners-Lee. *Turtle – Terse RDF Triple Language (W3C Team Submission 14 January 2008).* World Wide Web Consortium, http://www.w3.org/TeamSubmission/turtle/, 2008.

57. Dan Brickley and R.V. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema (W3C Recommendation 10 Februar 2004).* World Wide Web Consortium, 2004.
58. Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (W3C Recommendation 27 October 2009).* World Wide Web Consortium, October 2009. Available at `http://www.w3.org/TR/owl2-syntax/`.
59. Harold Boley, Gary Hallmark, Michael Kifer, Adrian Paschke, Axel Polleres, and Dave Reynolds. *RIF Core Dialect (W3C Recommendation 22 June 2010).* World Wide Web Consortium, June 2010. Available at `http://www.w3.org/TR/rif-core/`.
60. Eric Prud'hommeaux and Andy Seaborne. *SPARQL Query Language for RDF (W3C Recommendation 15 January 2008).* World Wide Web Consortium, 2008.
61. Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data — The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3), 2009.
62. Stefan Zander and Bernhard 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.
63. D. Preuveneers, J. Van den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, and K. De Bosschere. Towards an Extensible Context Ontology for Ambient Intelligence. In Panos Markopoulos, Berry Eggen, Emile Aarts, and James L. Crowley, editors, *Second European Symposium on Ambient Intelligence*, volume 3295 of *LNCS*, pages 148 – 159, Eindhoven, The Netherlands, Nov 8 – 11 2004. Springer.
64. Stefan Zander and Bernhard Schandl. Context-driven RDF Data Replication on Mobile Devices. *Semantic Web – Interoperability, Usability, Applicability*, under review. available at `http://www.semantic-web-journal.net/content/new-submission-context-driven-rdf-data-replication-mobile-devices`.
65. P. Korpipää, E. Malm, I. Salminen, T. Rantakokko, V. Kyllönen, and I. Känsälä. Context management for end user development of context-aware applications. In *MDM '05: 6th int'l conf. on Mobile data management*, pages 304–308. ACM, 2005.
66. F. Ramparany, J. Euzenat, T. H. F. Broens, A. Bottaro, and R. Poortinga. Context management and semantic modelling for ambient intelligence. Technical Report TR-CTIT-06-52, Enschede, April 2006.
67. T. Heath, M. Dzbor, and E. Motta. Supporting User Tasks and Context: Challenges for Semantic Web Research. *Proc. ESWC2005 Workshop on End-User Aspects of the Semantic Web (UserSWeb)*, 2005.
68. Tom Heath, Enrico Motta, and Martin Dzbor. Context as a foundation for a semantic desktop. In Stefan Decker, Jack Park, Dennis Quan, and Leo Sauermann, editors, *Proc. of Semantic Desktop Workshop at the ISWC, Galway, Ireland, November 6*, volume 175, November 2005.
69. J. Euzenat. Alignment Infrastructure for Ontology Mediation and Other Applications. In *MEDIATE2005*, volume 168, pages 81–95, 2005.
70. George H. Forman and John Zahorjan. The Challenges of Mobile Computing. *Computer*, 27(4):38–47, 1994.
71. 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.
72. C. B. Anagnostopoulos, Y. Ntarladimas, and Stathes Hadjiefthymiades. Situational computing: An innovative architecture with imprecise reasoning. *Journal of Systems and Software*, 80(12):1993–2014, 2007.
73. Thomas Springer, Patrick Wustmann, Iris Braun, Waltenegus Dargie, and Michael 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.
74. Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs. *Journal of Web Semantics*, 3(4):247–267, 2005.
75. Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. Sw-store: a vertically partitioned dbms for semantic web data management. *The VLDB Journal*, 18(2):385–406, 2009.
76. Alice Hertel, Jeen Broekstra, and Heiner Stuckenschmidt. RDF Storage and Retrieval Systems. On-line, 2008.
77. Dan Brickley and Libby Miller. Foaf vocabulary specification 0.91, 2007.
78. Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1992.
79. Android developer's guide, 12 2010.
80. Rick Rogers, John Lombardo, Zigurd Mednieks, and Blake Meike. *Android Application Development: Programming with the Google SDK.* O'Reilly, Beijing, 2009.
81. Sayed Y. Hashimi, Satya Komatineni, Dave MacLean, Sayed Y. Hashimi, Satya Komatineni, and Dave MacLean. *Pro Android 2.* Apress, 2010.
82. Reto Meier. *Professional Android 2 application development.* Wiley Pub., 2010.
83. Bernhard Schandl. Replication and Versioning of Partial RDF Graphs. In *Proceedings of the 7th European Semantic Web Conference (ESWC 2010)*, 2010.