Distrib Parallel Databases (2007) 21:85–111
DOI 10.1007/s10619-006-7005-5

Sharing hierarchical context for mobile web services

Christoph Dorn · Schahram Dustdar

Published online: 8 December 2006

Abstract Context has the potential to enhance Web services in mobile environments to a great extent. Yet, challenges such as bandwidth restriction or dynamic changes require considerations that need to be reflected in the distribution of context. In this paper, we present a novel technique to share and control access to context. In structuring context information according to levels of granularity, we achieve context propagation at the relevant levels of detail while protecting the privacy of the user. Introducing the *Context Access control, Subscription and Query Language* (CASQL), we enable fine-grained subscriptions and control over context in our proposed *Context Sharing Architecture* (CoSAr). A collaborative work scenario accompanies our approach for a unified way of accessing context information.

1 Introduction

Providing Web services for and on mobile devices has not only become a major field in the research community but also opened great business opportunities. Bringing the Service-oriented Architecture to mobile devices enables nomadic users to access conventional services that were formerly limited to static clients. Thus, this architecture of loosely coupled components, based on standardized protocols and description techniques featuring platform independence can realize pervasive computing—anytime, anywhere, any device. However, this evolution is not only about invoking services from mobile clients but also about providing services within the mobile network. As

Recommended by: Djamal Benslimane and Zakaria Maamar

C. Dorn (✉) · S. Dustdar
VitaLab, Distributed Systems Group, Institute of Information Systems, Vienna University of Technology, Vienna, Austria
e-mail: dorn@infosys.tuwien.ac.at

S. Dustdar
e-mail: dustdar@infosys.tuwien.ac.at

both service requestor and service provider become part of a dynamic environment, new challenges arise. Hence, we will first motivate the use of context in mobile environments. This will consist of discussing the dynamic nature of mobile services and the importance of sharing context information. The subsequent problem statement will discuss the encountered issues in more detail.

1.1 The mobile environment

Figure 1 depicts an environment that consists of a mobile and a static part. We argue that a wire-line network needs to be part of such a dynamic environment as communication happens not only within the wireless part but also to and from the static one. However, possible scenarios in our environment need not rely on such an infrastructure (e.g., mobile ad-hoc collaboration). In the scope of this paper, we do not differentiate between different types of mobile devices, but instead we state that we are aware of the heterogeneous nature of potentially participating appliances such as smart phones, personal digital assistants (PDAs), and laptops. Furthermore, our mobile environment features the three main interaction patterns we identified as important to our analysis. They give an estimate of the complexity of mobile Web service communication.

1. The wireless network hosts both requestor and provider. This usually occurs in (mobile) ad-hoc networks. However, infrastructure might exist that aids interaction between peers.
2. The requestor is mobile whereas the provider is situated in the wired part. This is probably the most common setting, opening up wired services also to nomadic users.
3. The requestor is located in the wired part and invokes a service on a mobile device. Examples are push services or applications for tracking purposes.

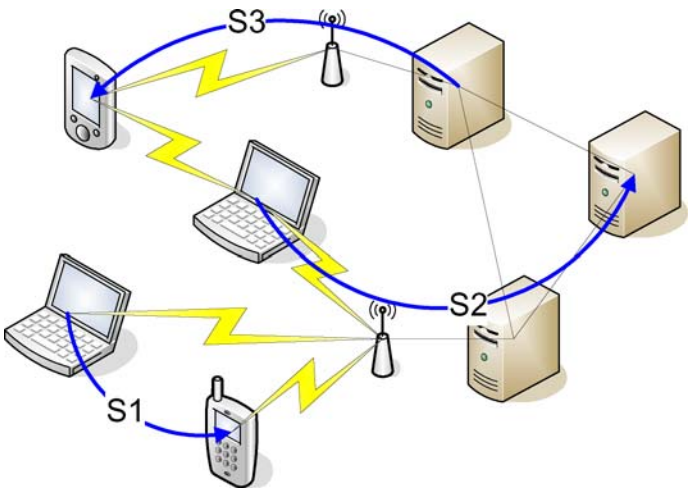

Fig. 1 Mobile Environment: also a wired segment is part of a mobile environment as nomadic clients access static services and vice versa. The three main interaction scenarios are denoted as S1 to S3

1.2 Mobile Web services

To outline where and how context provides a benefit, we need to go into further detail. From the view of fulfilling a task by using one or several services, the following dimensions help to understand vital (mobile) service characteristics.

- Service Execution Time: A service can be executed immediately or scheduled for later invocation.
- Service Execution Occurrence: A service can be used just once, on a regular basis, or event-triggered.
- Service Uniqueness: The nature of the task determines if an exact service is needed (such as a personal calendar service) or any service from a given group (e.g., any image processing service able to convert from gif to jpeg) or all available services of a given type (e.g., all current chat services during a team meeting).

These dimensions visualize the complexity found in the service-oriented architecture. While they pose less of a concern in static networks, the dynamic nature of mobile networks requires thorough considerations. At this point, we introduce the concept of context to mobile Web services.

1.3 Service-oriented context

A single definition of context does not exist nor would it be sensible. Bazire and Brézillon [1] collected 150 definitions from various areas of research. A widely adopted definition in the domain of computer science by Dey and Abowd [2] 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.

We need to modify this rather general definition to better suit a service-centric view. Hence, we propose the following adaptation.

> 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 a service as well as in-between services, including the user and services themselves.

In addition, we like to emphasize the main points of our definition. In the domain of service-oriented computing where there exists no longer just a monolithic application, the definition of context is not purely focused on the interaction between a user and an application but also covers the interaction between services. Subsequently, context also describes the situation—the embedding—of a service, service instance, and service platform. Especially in a mobile environment, location can be used to describe the context of the service client—representing the user—and the service instance, possibly but not necessarily related to another user. Hence, two classes of context information arise: user-related context and service-related context.

Assuming that services are invoked to reach a goal, we add another class: task-related context. This rather conceptual class contains context information that

describes the overall goal, the current task, related tasks, and the role of the invoking party to allow services to behave accordingly. In this case, the context neither describes the user nor the service but links a certain task to its social situation. To put it differently, services can be invoked on behalf of a role (independently of its current bearer) or a group—ranging from a small team to a large community.

The following list provides some examples for the three context classes.

- User-related context: preferences, location, activity, presence, available devices, and communication capabilities.
- Service-related context: location, system capabilities (e.g. processing power, bandwidth, storage capacities, or available main memory), network hop distance to same, similar, or related services, position within a workflow.
- Task-related context: community, team, roles, goal, task, task patterns, related tasks.

Comparing service-related to task-related context, we can observe that the latter describes information at a conceptually higher level. For example, related tasks and patterns can be mapped to workflows. Yet this task-related context is of importance, as workflows heavily rely on role definitions and group structure and not on individual entities alone [3]. The four dimensions of context information *location*, *identity*, *time*, and *activity*, identified by Dey [4], can be applied to all three above introduced context classes.

1.4 Empowerment through context

Having discussed the basic nature of context, we can now present the benefits of context in the area of mobile Web services. Above, we have pointed out, that the presented dimensions such as service execution time, execution occurrence and service uniqueness are of less concern in static networks than mobile ones. This results from the fact that a mobile network is more prone to changes than a static one. These changes can be anticipated and reacted upon more appropriately if decisions are based on context information. The different context information classes as outlined above cover the whole service flow starting at the user, stretching across service client, composite services, service platforms, and finally reaching basic service providers. Examples at various stages include service response adaptation as results are tailored to the user's device and activity. Trying to find the most stable service, the client might look up the nearest service by applying location information in the form of network hops or GPS coordinates.[1] A composite service utilizes time as context information to predict load peaks on employed services. Simple services anticipate load-distribution by observing the number of similar services available. In addition, service execution platforms incorporate context information about the mobile environment to proactively detect changes and notify services to reconfigure. While these examples are rather generic, a greater variety of context types and granularity would be specific to actual applications. In short, in static networks we can plan and provide services well in advance, whereas in mobile environments we need context information to achieve this as late as possible and as early as sensible.

[1] We are aware, that location is not the only relevant context when determining the most suitable mobile service that remains available to allow a successful invocation.

All these examples have the common need for exchanging context information as this information resides distributed amongst the mobile network participants. Taking the number of internet-enabled mobile devices in use worldwide and having experienced the awkwardness in using the mostly context-unaware applications and services, the need for enabling context usage becomes evident. Consequently, rendering context sharing efficient and widely employable is one major concern. Hence, we are not going to present a new context framework as a number have already been proposed—some presented in the section on related work—but instead focus on sharing aspects.

The subsequent section gives an in-depth view on the faced challenges (Section 2) including a motivating scenario followed by an introduction to the central ideas of our paper (Section 3). Thereafter, we present our approach to these challenges. This includes introducing levels of context details to reduce the amount of traffic as only information of the required granularity is transferred. In addition, we suggest the notion of dominant context information to further reduce transmitted data. Moreover, a hybrid push-pull-based context sharing mechanism instead of a conventional one enables context updates only being propagated if the local context of the interested nodes indicates the update as relevant. Subsequently, Section 4 presents the integration of these techniques into our *Context Sharing Architecture* (CoSAr). Describing in detail our *Context Access control, Subscription and Query Language* (Section 5), we move along to introduce our proof-of-concept implementation (Section 6). Section 7 provides an evaluation focusing also on aspects concerning mobility before sections on related work (Section 8), future research (Section 9) and final remarks (Section 10) complete our paper.

2 Problem statement

By focusing on sharing, we provide mechanisms and insights to allow also users working on devices not especially suited for context sensing, processing, or storing to benefit from the power of remote context information.

In the Introduction, we have pointed out the distributed nature of context in mobile environments and the need for sharing such information. Yet, there are some challenges that influence the sharing mechanism to a great extent.

To begin with, mobile devices are rather heterogeneous when it comes to processing power, storage capacity, available memory, and communication capabilities. A sharing mechanism thus needs to consider these differences.

Furthermore, the available transmission technologies such as Bluetooth, WLAN, WIMAX, GSM, GPRS, UMTS, or HSDPA shape the amount of exchanged context information to a great extent as the available bandwidth is very low or costly, the transmission range is limited, or any combination thereof. Hence, reducing the amount of transferred information is another requirement.

As users move around, switch on and off their devices, a sharing algorithm must be able to cope with unreliable links and a dynamically changing environment.

In contrast to conventional, infrastructure-bound services, mobile Web services are free to move around, thus generating and consuming context changes themselves. Consequently, context is not only limited to the invoking party but also embraces the providing one, thus additionally increasing the amount of context data.

Besides device relocation and service reconfigurations, ever-changing activities, interactions, goals and situations produce an enormous amount of context data spread all over the network. As context producer and consumer—including intermediaries such as context brokers—are likely to be distributed but require such information, a mechanism to define clearly what context information is of relevance (and thus needs to be exchanged) is necessary.

Finally, for sharing context information we also need a means to manage access. Especially as context is in many cases directly user-related, a fine-grained mechanism provides privacy.

In the following subsection, we will further outline these challenges and corresponding requirements embedded in a scenario.

2.1 Motivation scenario

This motivating scenario describes a situation taken from the domain of distributed collaboration: Alice is member of a distributed mobile team for project CollabContext. Her colleagues—Bob, Carol, and Dave—work on the same project, but are employed at different companies working from their offices, on the move, and from home. Given the heterogeneous organisational environment, the team members mostly provide the necessary collaborative services on their respective devices. As each member is involved in several other projects at the same time, their schedule is tight and their calendars filled. In order to remain up-to-date of all team activities, they agree to use a distributed awareness service that provides availability and device information. In addition, members switch between activities related to different projects, move around, change their devices, communicate by means of VoIP or mobile phones and engage in video conferences or chats.

At one point, Alice wishes to arrange a meeting with her team members. She accomplishes this task by invoking a *Meeting Scheduling and Agreement Web service*. This composite service possesses enough logic to arrange a meeting but needs to access further services for acquiring calendar data, checking availability, executing a scheduling algorithm, and resolving arising date conflicts. Figure 2 visualizes all involved components and displays the steps made in the following description. There are a number of context types involved, namely: activity (describing work environment, project, activities, and artefacts), reachability (which device and communication means are available to contact a user), and device status (system load and capabilities). These three types are structured as hierarchies and serve as the underlying data structure used in our prototype.

1. Alice invokes the *Meeting Scheduling and Agreement Web service* stating the members and corresponding project.
2. The meeting service then contacts the shared team *Calendar Web service* (2a) to retrieve the calendars of all participating members (including Alice) and a *Awareness Web service* (2b) to check for their current reachability. We assume the awareness service has subscribed to all members, respectively their devices for high-level availability and device status context information. Currently, Dave's laptop and PDA as well as Carol's smartphone are online, while Bob is unavailable for the moment.

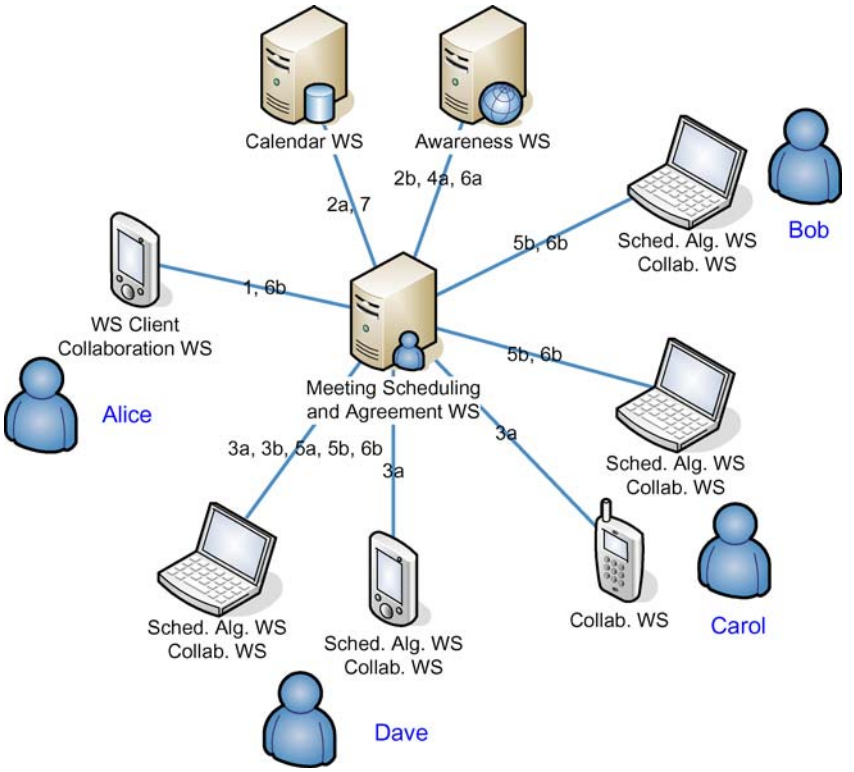

Fig. 2 Meeting scheduling scenario in a mobile environment. The client and the collaboration services reside on mobile devices, whereas the composite *Meeting Scheduling and Agreement Web service* as well as the *Calendar Web service* and the *Awareness Web service* are abstract services and can be implemented either distributed or centrally provided by the infrastructure. Lines represent information flow between nodes. The numbers on the lines indicate the temporal aspect as given by the textual description of this scenario. We abbreviated the labels for *Scheduling Algorithm Web service* and *Collaboration Web service* on the mobile devices and omitted the links between each device and the *Awareness Web service* for clarity reasons

3. Next, the service queries all available devices for their system load and capabilities (3a) and finally invokes the *Scheduling Algorithm Web service* on Dave's laptop (3b), which is experiencing the least load.
4. In the meantime, the availability service notifies (4a) the meeting service that Bob is available now and Alice has become offline. In addition, Carol changes from her smartphone to her laptop, yet this information is not propagated as also the awareness service has not subscribed at such level of granularity.
5. The *Scheduling Algorithm Web service* detects (5a) a conflict that requires human intervention to be solved. As Alice is still offline, the meeting service cannot contact all necessary members. Hence, it subscribes (5b) to activity information concerning the whole team at a very coarse-grained level, as all members prefer to be contacted when at work and not during their freetime. Thus when they are at home, neither reachability nor device status information is available, even their devices might be connected.

6. Once Alice reports back, the awareness service notifies (6b) the meeting service that all members are online. Thus using fine-grained reachability information directly from all connected devices a *Collaboration Web service* on the best suited device for each participant brings together all involved team members to agree on the proposed date or another date. As the *Collaboration Web service* knows about the goal type of communication (task-related context information) it can choose the right means of communication: in this case synchronous chat.
7. After the four have agreed on the meeting details, the team calendar is updated and the meeting service terminates.

Once we consider the huge amount of information on activities, devices status and reachability that would be transferred between nodes without any granular structuring, the benefit of using hierarchical context information combined with a hybrid sharing mechanism becomes evident. Imagine location information used within each company, that is irrelevant in such a distributed team but nevertheless shared, as there is no mean to control context flow.

Furthermore, this scenario highlights the two ways how context information is retrieved. The composite meeting service subscribes and queries context information at different levels of granularity. So, on the one hand it is interested in change events (for which it receives notifications) and on the other hand it retrieves context facts once relevant changes have occurred. Especially in such a dynamic, non-deterministic environment, using either purely pull or push mechanism for both situations will result in more traffic than necessary.

Thus in the following section, we demonstrate how a set of mechanism combined offer a flexible and powerful solution to these challenges.

3 Accessing and sharing context

Our contribution consists of three key concepts—namely *Context Granularity*, *Context Dominance*, and *Granularity-Aware Context Sharing*—that cover context data modelling, context relevance ordering, and context distribution.

3.1 Context granularity

The idea of structuring context hierarchically is not new. For example, Ferscha et al. [5] introduced a symbolic location containment hierarchy consisting of concrete and abstract places (such as campus, 1st floor, or room1 in the former case and faculty1, department1, or desk in the latter case). We argue that hierarchical structuring of context information should not be application specific. Instead, we propose to map the representation of a given context dimension to a level of granularity in a corresponding hierarchy. That is to say, context information is modelled as required—by means of an ontology, graph, tuple, etc. Yet, it needs to be accessible at various levels of detail. Example hierarchies related to people could be focusing on location (town, district, street, floor, room), abstract places (indoors, faculty, department, desk), presence (not available, lunch, lunch with colleagues, lunch with Mr. X and Mrs. Y), time (in the future, next week, Monday, 12:00), or activity (work, project X, document

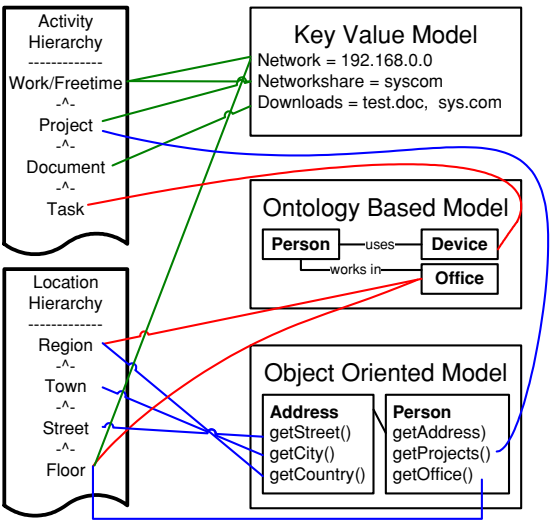

Fig. 3 Providing context access by means of hierarchies: in this example activities and locations are accessible at different levels of granularity. The three exemplary models visualize how different context forms map to the same hierarchy. A *model-to-hierarchy mapping*—indicated by means of the lines between models and hierarchies—determines how context information flows between the respective forms of representation. This mapping is specific for each problem domain as the links contain semantic information

Y, task Z). In the case of services, an exemplary hierarchy could describe the load on a service execution platform: Available Services, high load, 80% load, Executed Services. Figure 3 contains such an exemplary mapping.

It is neither sensible nor possible to provide all available context information in such a way. Only information subject to frequent changes should be structured this way to allow for a fine-grained access and update mechanism. Consequently, the further up in a hierarchy an update occurs, the more significant it is. Yet, those data that rule out hierarchical structuring or data always used at the same level of detail (e.g., user preferences) need not necessarily be excluded but rather form a hierarchy merely consisting of a single level.

Providing context information in such a way enables better context reuse between peers using different context semantics. As long as the vocabulary is the same, both peers—featuring different task specific high-level reasoning—can draw their own conclusions from the shared set of context information. Subsequently, hierarchies can be seen as interfaces of context models.

3.2 Context dominance

Context dominance describes the concept of ordering context information according to importance. Taking our scenario as an example, the participants define the context state "Freetime" of context type *Activity* as dominant over context information of type *Reachability* and *DeviceStatus* as they would like to remain uninterrupted regardless of changing devices. Therefore, occurring updates on context information concerning these two context types are not propagated in case of such activity status. Context dominance is expressed as a set of rules that are restricted to a device, a user, or are globally applied to all participating services involved in fulfilling a given task. Moreover, dominance rules become active depending on context values at other granularity levels and hierarchies. Thus, one context type might be dominant over a second one

under certain conditions but subordinate otherwise. We present the structure of such rules in Section 5.

3.3 Granularity-aware context sharing

With our third contribution, we introduce a hybrid, granularity-aware context sharing mechanism. Context information is neither purely pushed nor purely pulled between nodes but a combined, hybrid technique is applied. Pushing context information results in unnecessary traffic as updates are always propagated at the most detailed level—event though the receiving node finds that data useless at the given time or granularity. On the other hand, pulling context information either causes much network load (in case of small polling intervals without any updates available) or delays delivery if polling happens rarely.

We combine and enhance these mechanisms in two ways. On the one hand, pushing context is based on well understood subscribe/notify techniques. The difference to conventional publish/subscribe systems is the ability of subscribers to define in greater detail the events they would like to be notified about by means of conditions. Thus, the context requestors need not rely on predefined events. In our case, these events are context updates at certain granularity levels. Two classes of subscriptions are available: (1) basic subscription defining merely the level of detail and optionally the value of which context changes are of relevance and (2) composed subscriptions that introduce conditions based on values of other context type. Consequently, our subscribe/notify mechanism requires access to context information at different granularity levels as introduced above. We present our *Context Access control, Subscription and Query Language* (CASQL) in further detail below where we also give some examples.

Besides a push mechanism, the pull-based one needs integration as we argue: relevance of context information is dependent on the actual context. Subsequently, as context updates of a given granularity arrive, the overall context can change which might result in further context information at a different level of detail being required. At this point, the receiver pulls additional—more detailed—context information from the context provider and optionally updates the subscription rules. The motivating scenario demonstrates this strategy: subscribing to coarse-grained availability information combined with retrieving fine-grained device status on demand. Yet, obtaining this information is not limited to the node providing the update in the first place but includes all available context sources. As any incoming context information might lead to inconsistencies within a hierarchy, we need a mechanism to deal with such conflicting context data. The *Hierarchy Adapter* in Section 4 provides such a means.

We merged these three ideas in our *Context Sharing Architecture* (CoSAr) and *Context Access control, Subscription, and Query Language* (CASQL) presented in the following sections.

4 Context Sharing Architecture (CoSAr)

In this section, we present the integration of our techniques and mechanisms into a mobile Web service execution platform in form of a logical layer. Our proposed *Context Sharing Architecture* consists of a *Context Publish/Subscribe* (CPS) component, a

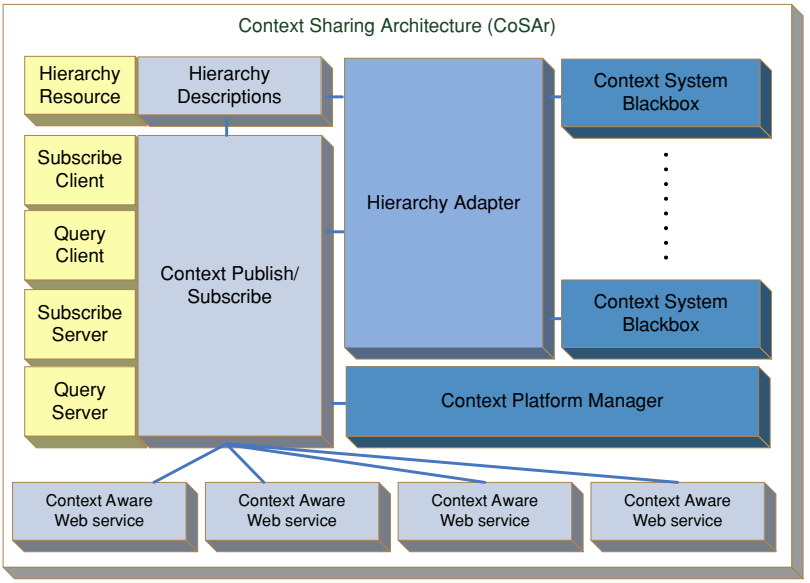

Fig. 4 Context Sharing Architecture (CoSAr) for a mobile Web service execution platform: CoSAr consists of a *Context Publish/Subscribe* component, *Hierarchy Adapter*, *Hierarchy Descriptions*, *Platform Context Manager* and *Context System Blackbox*

Platform Context Manager (PCM), *Hierarchy Descriptions* (HD), a *Hierarchy Adapter* (HA) and one or more *Context System Blackboxes* (CSB). These main components are visualized in Fig. 4. Together they provide capabilities to Web services for accessing local and remote context information. For our scenario, we envision all participating devices to feature a complete (or partial on small-scale devices) implementation of this architecture.

The Conxtext Publish/Subscribe (CPS) component is the main module responsible for sharing context information in a push- and pull-based manner. Pull-based retrieval is realized by a separate Web service interface using the same data-structure as notification messages. The main tasks are twofold. On the one hand, the *Context Publish/Subscribe* component processes incoming context (subscription and query) requests, and on the other hand, it queries and subscribes to remote context information on behalf of local Web services. In the latter case, the remote part needs not be a context sharing layer of the same kind, but can be any context source able to process subscription or query requests (specified in CASQL). The CPS module has access to a set of rules to decide which context information may be subscribed to. Specifically, the Privacy Rules and active Dominance Rules determine the extent to which context is shared. Both sets contain rules—written in CASQL—that are added and controlled by the user, via policies, or by means of Web services. Furthermore, the module sustains a list with remote and local subscriptions. Finally, the CPS shares the common set of available context *Hierarchy Descriptions* with the *Hierarchy Adapter*. There are two remotely accessible interfaces. The *Subscribe Server* serves as an endpoint for incoming subscribe requests, whereas the *Query*

Server handles incoming query requests. For local services, the *Subscribe Client* and *Query Client* are source of outgoing requests and queries as well as destination for incoming notifications.

Hierarchy Descriptions (HD) define the remotely available hierarchies and locally supported ones. As the *Hierarchy Resource* interface enables retrieval of supported context hierarchies, remote Web services can check whether the required context information can be obtained via subscriptions. In addition, also the *Context Publish/Subscribe* module can check if subscription requests are likely to be accepted. In both cases, the necessary messages run between the CPS modules, which provide the acquired hierarchies to the local Web services.

The Hierarchy Adapter (HA) provides a bridge between *Context System Blackboxes* and the given hierarchies. It translates the query and subscription statements into the appropriate representations. Thus, it maps the given granularity levels to the internal data structure and capabilities of the context system. In the other direction, the HA module analyses context data (in form of events, notifications, etc.) from the context system to determine the corresponding hierarchy and level of granularity. Subsequently, the *Hierarchy Adapter* generates the appropriate CASQL event description and forwards it to the CPS module for further processing. Depending on the mapping between hierarchies and context system, an input from either side (CPS or CSB) can result in multiple output statements on the other side. As the mapping mechanism quite heavily depends on the semantics of the specific hierarchy and the underlying context model of the CSB, it is not possible to describe a generic algorithm for all context representation forms. However, an approach for mapping context information from an object based model to a generic hierarchy is depicted in Fig. 5 and works as follows. Each object in the CSB that maps partially or even completely to a hierarchy defines one or more parameters that are part in possibly several mappings. A mapping describes how a parameter from the object model fits to a value of a specific hierarchy. On the hierarchy side, each hierarchy has levels that link to parent and child level by means of an aggregation method.

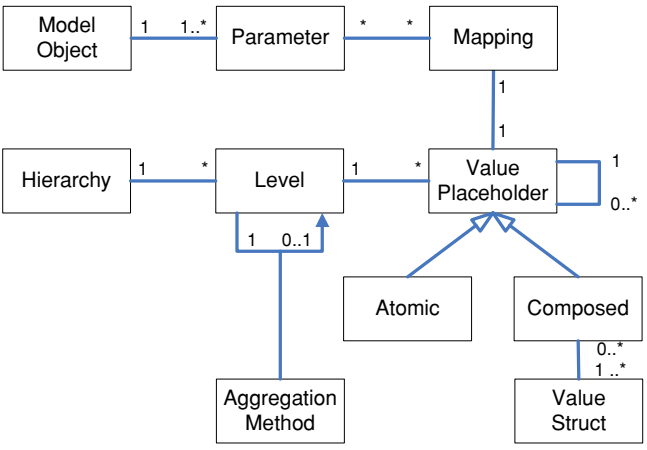

Fig. 5 A generic approach to map hierarchies to objects and vice-versa

A level consists of value placeholders. These values either can be simple, atomic values or composed ones that are further described in the value struct. Hence, if a change event in the CSB occurs, the corresponding parameter is looked up and the mapping(s) retrieved. Subsequently, the value of a hierarchy and level is created and a CASQL event fired. Vice-versa, from an incoming notification, the HA extracts the value, looks up the corresponding mapping and inserts the data for the fitting parameters.

Conflicting values within a hierarchy are dealt with in two components. Either the context system alone determines which piece of context information is discarded or the *Hierarchy Adapter* assists in this task. As the the HA is unaware of the semantic meaning of the contextual data, it cannot detect conflicts by itself. However, values in a hierarchy feature a confidence value at each level, where more coarse-grained values are more confident than fine-grained ones. Thus, the CSB is able to apply a threshold that defines when lower level context information overrides higher ones. Otherwise, the value at the higher level takes precedence. Another feature of the HA is partial hierarchy coverage which describes the incomplete mapping of context information and hierarchy levels as well as values. Here, some complete levels or individual values are not considered, as corresponding context information is either not available or of unsupported type. Thus, some levels of granularity are either neglected or merged with hierarchically higher ones. More detailed mechanisms of information matching are outside the scope of this paper. We will address them in our future work. Moreover, also whole hierarchies are unaccounted for at the *Hierarchy Adapter*, if they describe context information that Web services subscribe to at remote nodes. Yet another aspect is the HA's support for multiple, heterogeneous context systems, each requiring a separate mapping. However, each hierarchy can only contain information from one context system as otherwise the HA would not know which CSB will produce the events for a given subscription request. Hence, it cannot request that information from the correct CSB.

The Platform Context Manager (PCM) acts as a small-scale context system providing service specific context information. It analyses service requests to establish who is invoking a service, what QoS properties are available or required, what endpoints are accessed from local services, which services are deployed and what instances of them are currently running. This context information allows classifying context requestors according to participation in composite services. Hence, the PCM pre-defines several roles to permit Privacy and Dominance rules based on this information. For example, a service instance can restrict access to context information to services participating in fulfilling a composite task. Thus, it needs not state the exact type or address as defining the role is sufficient. Moreover, in such a scenario, if one of these services no longer participates in the composition, it also looses its right to access context information. The first part of Table 1 lists the available roles, which relate to the service dimensions presented in the introductory section. In addition, the PCM also provides abstract events related to service compositions—not to be confused with context change events. Start or endpoints of service invocation allow restricting context access to a specific, a-priori unknown time span. In this way, the PCM supplies service context information to enable more efficient and effective use of general context information. Nevertheless,

Table 1 Platform context manager roles and events

	Name	Description
Roles	Composition	All services that participate in a composite task execution
	Requestor	The (remote) service or client that invoked the corresponding local one, part of Composition
	Provider	Services that are used by the local one, also part of Composition
	Interface	All services (remote or local) that implement the same interface
	Identical	All services instances of the same local service
	Local	All local services
	Remote	All remote services
Events	Invocation start	The moment, a service method was invoked
	Invocation end	The moment, a service finished
	Conversation start	The moment a conversation is started
	Conversation end	The moment a conversation is finished

information regarding service platform load are not located at the PCM but made available by means of a regular service and a corresponding hierarchy.

The Context System Blackbox (CSB) contains all elements of a context management system such as the ones introduced in the section on related work. For our purpose, it is sufficient to handle the system as a blackbox assuming that it includes components such as context reasoner, context processor, context storage/database, and context sensors (or interfaces to access sensors). In case of several CSBs, we assume each of them remains unaware of the other ones. Furthermore, we expect applications—or services in our case—to access the context system blackbox natively. Hence, the context sharing layer enables context retrieval from the local CSB by means of the *Hierarchy Adapter* or directly. The latter case is, however, subject to future research.

Web Services become context-aware as they access context locally via the *Hierarchy Adapter* or remotely by means of the *Context Publish/Subscribe* component. The services themselves normally act as mere context consumers, however, the HA also enables adding context information to the CSBs. This capability is required not only to forward remote context events to the subscribing Web service but also to feed this information into the local context system(s). In addition, Web services possess a set of Privacy rules and private Dominance rules to adapt their subscriptions and update the CPS's active Dominance rules.

To provide a better understanding of the *Context Sharing Architecture*, the following examples of information flow describe the interaction between the components in more detail.

Service instantiation results in a new Web service instance provided with relevant context information coming from the invoking party and the local context system. This includes Dominance rules and Privacy rules. Subsequently, the service analyses the current context and adapts these rules. The service checks upon start-up whether the required context hierarchies are provided. By comparing the structure of available and required hierarchies, the service determines the compatibility and tries to detect if existing deviations prevent it from functioning properly. Such an algorithm is briefly outlined in Annex A. In our scenario, upon invocation the

meeting service checks with the awareness service if their hierarchies are compatible. In this case, as subscriptions to the awareness service are at a very coarse-grained level only the top values need to fit.

Outgoing Context Subscriptions are initiated by a Web service. It provides one or more rules in CASQL, which represents the subscription request. In accessing the *Context Publish*/*Subscribe* module, the service also states whether the required context information is found locally or remotely. If in lack of this information, the CPS module determines the correct context source by checking available hierarchies, information embedded in the requests and information coming from the *Platform Context Manager*. In case of a remote location, the CPS subscribes at its remote counterpart. For example, the awareness possesses no context information about Alice, Bob, Carol, and Dave locally, so it requests the CSP to subscribe to such information remotely.

Incoming Context Events are augmented with context information from the *Platform Context Manager* to establish related services and rules. In case such context information is stored locally, the context changes are integrated in the corresponding *Context System Blackbox* by first passing the *Hierarchy Adapter*. Otherwise, only the subscribing Web service obtains the context event. Depending on the content of the received context change, the service might find minor and major adaptations of Privacy rules and current subscriptions necessary. In turn, major incoming events can result in the CSB or Web service publishing context changes.

Incoming Context Subscriptions in CASQL are first checked against the existing Privacy rules for permission. The *Context Publish*/*Subscribe* module also verifies by means of the Dominance rules whether a subscription will ever receive any notifications. Context information from the *Platform Context Manager* describes which roles the subscribing party has in relation to invocation, composition and policies. In case of a valid subscription, the *Hierarchy Adapter* parses the supplied rule to extract the matching condition and maps it to a request for the *Context System Blackbox*. The CSP collocated with the awareness service checks the subscription request by the meeting service and accepts it as it subscribes on behalf of the team member Alice.

Outgoing Context Change Event. Upon receiving a context change event from the CSB, the HA creates the corresponding CASQL representation and forwards this to the CPS module. There the subscriptions are matched against the event to determine which ones fit. As in the meantime services might have changed Privacy and Dominance rules, no longer valid subscriptions requests are deactivated and thus excluded from the event matching process. Finally, the CPS component transmits context changes for all matching subscriptions. Referring to the scenario, the CSP collocated with the awareness service receives context changes from the team members and forwards these data at the requested level to the meeting service.

Remote Service Invocation includes context transfer as the local composite Web service provides relevant context information to the invoked services. Primarily Dominance rules (and to a lesser extent Privacy rules) determine which context is significant and thus needs to be pushed. Besides context, the invocation of remote services also includes the transfer of Privacy and Dominance rules if required.

5 Context access control, subscription and query language—CASQL

As we stated above, for efficiently using context we need access control, subscription mechanisms and querying tools. These three requirements form the main focus in our *Context Access control, Subscription and Query Language* (CASQL). Instead of designing a new syntax for our language, we rely on XML and XML-Schema. This enables the use of available tools for creating, validating, transforming, and general processing of context information in XML documents.

Central to our language is the hierarchical representation of context information. An XML-Schema document describes the general structure of this information, consisting of hierarchy details, level details and value details. This skeleton provides a basis for domain specific hierarchies, which then include specific levels and details on the used context information format. Such precise hierarchies serve as contracts between interacting services as they thus agree on supported levels and context format. The following elements are part of the generic hierarchy structure:

Hierarchy. The top level element is a *hierarchystruct* node which has an identifier for referencing from other documents, a version to allow for adapting and evolving hierarchies, a name, a human readable description and information on the depth of levels. In addition, the type field defines what kind of entity the hierarchy describes. The three applicable values correspond to Dey's et al. [4] classification of context entities, namely persons, places, and things.

Level. Each hierarchy consists of a number of *levelstruct* elements which represent context granularity. Each such level has an identifier, name and human readable description. Furthermore, links to parent and child levels establish a double-linked list that can include further levels later. In the simplest case, each level contains one value. Several values are possible, thus establishing a tree by linking to the parent value. Yet, we expect the number of branches and thus the complexity of the tree to remain small.

Value. Finally, a value describes how context events are structured. Allowing several values in each level enables extending the hierarchy at a later point without requiring existing services to adapt. This also supports different context states at a rather generic level featuring different context representations at lower levels, hence reducing the need to handle two separate hierarchies and at the same time enabling a more fine-grained context subscription.

The components of CASQL encompass XML schema descriptions of how queries, subscriptions, notifications (also used for query results) dominance and privacy rules are structured.

Dominance rules. As explained above, Dominance rules consist of statements that describe under which conditions one hierarchy is dominant over another one. For this purpose, a dominance rule defines the condition when it is valid (the dominant context) and the hierarchy that is dominated. Specifying not only a hierarchy but also levels and values enables to restrict sharing to a specific subtree of the hierarchy. Further parameters describe optional conditions on valid time and entity. Listing 1 shows one dominance rule taken from the scenario.

Privacy rules. In accordance with the context structure from the basic hierarchies, Privacy rules also build upon optional conditions regarding confidence, context value, timestamp and entity. More important, such a set is not restricted to single role, group or service but bound to a list of those. The Platform Context Manager manages the generally valid roles listed in the first part of Table 1. Additional ones are specific to services, and thus are user or task defined. They remain outside the PCM's context scope; hence, the respective services need to provide the mapping of roles and groups to entities.

Subscriptions define the entity (or role), level, and value for which to receive notifications. Optionally, it is possible to state a minimum confidence value, transition type (if an entity has reached a certain state, or left it), notification type (whether to receive an initial notification about the current state or just future events) and detail type (which part of a hierarchy: only values at the exact given level, above, below or all). The meetings service's subscription to the team members concerning their activity status is given in Listing 2.

Queries are similar to subscriptions except for the missing confidence value and notification type. For our scenario, we have identified queries and subscriptions as listed in Table 2.

```
<DominanceRule xmlns="http://ns1/vimocos/dominance" id="dr1">
 <dominant>
    <hierarchyId>ns2.activity.ActivityHierarchy</hierarchyId>
        <levelId>L1</levelId>
        <valueId>ns2.activity.Environment</valueId>
        <value>
            <Env:Environment xmlns:Env="http://ns1/vimocos/activity">
                <Env:envType>FREETIME</Environment:envType>
            </Env:Environment>
        </value>
    </dominant>
    <subordinate>
     <hierarchyId>ns2.vimocos.reachability.ReachabilityHierarchy</hierarchyId>
     <levelId>L1</levelId>
     <valueId>ns2.vimocos.reachability.TopStatus</valueId>
    </subordinate>
 <entity>Alice</entity>
</DominanceRule>
```

Listing 1 Example dominance statement: if the activity of Alice at level *Environment* is *Freetime* then do not propagate any *Reachability* information. Namespaces are substituted for line length reasons: ns1: www.vitalab.tuwien.ac.at and ns2: at.ac.tuwien.vitalab.vimocos

```
<Subscription xmlns:ns2="http://ns1/vimocos/sharing"
    detailtype="UPPERINCL"
    notificationtype="ALL"
    transitiontype="TO" xmlns="">
    <ns2:entity>Alice</ns2:entity>
    <ns2:hierarchyId>ns2.activity.ActivtiyHierarchy</ns2:hierarchyId>
    <ns2:levelId>L3</ns2:levelId>
    <ns2:valueId>ns2.activity.Project</ns2:valueId>
    <ns2:minConfidence>50</ns2:minConfidence>
</Subscription>
```

Listing 2 Example subscription statement: request notifications for any events about Alice that concern her Project status. In any case (events also occurring at higher levels), just return context from level three upwards and only if the confidence is at least 50. Namespaces are substituted for line length reasons: ns1: www.vitalab.tuwien.ac.at and ns2: at.ac.tuwien.vitalab.vimocos

Table 2 Subscriptions and Queries from the motivating scenario. Subscriptions and queries are based on the level and not on exact values, as this is sufficient

Nr	From	To	S/Q	Hierarchy	Level	Type
0a	Awareness	Alice, Bob, Carol, Dave	Sub	Reachability	L1	exact
0b	Awareness	AlicePDA, [. . .], DaveLaptop	Sub	DeviceStatus	L1	exact
2b	Meeting	Awareness about Alice, Bob, Carol, Dave	Query	Reachability	L1	exact
3a	Meeting	DaveLaptop, DavePDA, CarolSmartphone	Query	DeviceStatus	L3	lowerincl
5b	Meeting	Alice, Bob, Carol, Dave	Sub	Activity	L3	upperincl
6b	Meeting	Alice, Bob, Carol, Dave	Query	Reachability	L1	lowerincl

Notifications. Eventually subscriptions lead to notifications. These represent one possible path trough the respective hierarchy tree. Hence, at each level, there resides only one value object. A notification consists of value objects that refer to their respective level and hierarchy. Beside the specific context content, each value features context metadata such as confidence, context source, and timestamp. Both subscriptions and queries result in notifications being returned. In the case of context events at a higher level than subscribed to, also notifications are created as the context has consequently changed also at all lower levels.

The source of Privacy and Dominance rules is manifold. Some rules will be specific to the current platform, others shared amongst collaborating services. In addition, some will be user-defined, or come pre-installed with the service. These can then be refined for each service instance.

For each component we presented in the scope of CASQL, there exists an XML-Schema for validation. Moreover, example XML hierarchies, schemas, subscriptions, queries, and notifications are available for download at our project site.[2]

6 Implementation

We chose the open source Knopflerfish OSGi[3] platform as an environment for our proof-of-concept implementation for several reasons. To begin with, it provides the possibility to realize the individual components of our architecture as bundles that allow for dynamic installation, update, and removal without influencing the other bundles. Secondly, OSGi is able to run on mobile devices such as PDAs (using for example IBMs J9[4] Java virtual machine) and Laptops thus reducing the need to write a separate application for each class of mobile devices. In addition, OSGi support is about to be incorporated in Smartphones as well. With regard to Web service support,

[2] Project site link: http://www.vitalab.tuwien.ac.at/projects/vimocos

[3] http://www.knopflerfish.org/index.html

[4] http://www-306.ibm.com/software/wireless/wctme/bundle.html

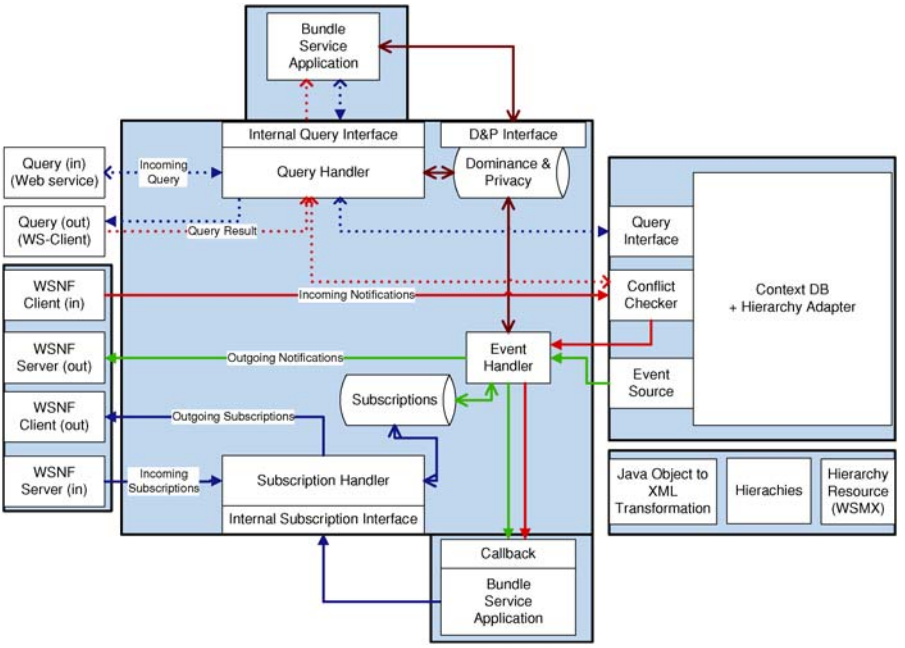

Fig. 6 The internal message flow of the Context Publish Subscribe component and WS-Notification and OSGi interfaces

the Knopflerfish project provides a plug-in bundle[5] based on Axis 1 that allows any bundle interface to be made available as a Web service.

For demonstrational reasons, we opted to change the nature of the hierarchy adapter to include a context database. This reduces the complexity of the initial version while preserving the benefits that come with the hierarchical structuring of context. Otherwise, our implementation reflects the CoSAr architecture. Figure 6 displays the internals of the context sharing bundle including the available internal and external interfaces as well as additionally required bundles.

The Context Sharing Component is split into a generic WS-Notification Bundle and a context specific bundle to allow reusing the WS-Notification sources. Also the hierarchies and all XML-to-Java and vice-versa mapping code are combined in a separate bundle. The database bundle provides the events and query interface and thus can be exchange with a true hierarchy adapter and back-end context system at any time. Making use of the intra-OSGi container communication facilities, the scenario Web services access the internal interfaces of the context sharing component rather than using the Web service based ones. To evaluate our architecture, we implemented the services from the motivating scenario as well as a simulator to control the generation of context change events. Having provided a first prototype, we are going to evaluate our architecture in the next section.

[5] https://www.knopflerfish.org/svn/knopflerfish.org/trunk/osgi/bundles_opt/soap/readme.html

Table 3 Context sharing protocol message size: numbers are in bytes

Message type	Size
Subscription Request	1200
Subscription Response	810
Unsubscribe Request	690
Unsubscribe Response	690
Notification Envelope	900
Query Request	710
Query Response Envelope	400

The values for Notification and Query Response messages exclude the actual context payload.

7 Evaluation

We base our evaluation on a series of test runs where we observed the messages flow to derive the average size for each type given in Table 3. We then demonstrate the benefit of our architecture by calculating the reduction of transferred context data in the following three areas.

1. The hybrid approach of query and subscription to context information reduces protocol overhead as opposed to pure pull or pure push-based solutions.
2. Subscriptions based on levels and values reduce the amount of unnecessary (because too detailed) context information.
3. Specifying which part of a hierarchy to transfer upon a request or notification further limits the transferred amount of context.

As we are dealing with context information that changes in a non-deterministic way, we will not compare our hybrid approach with a purely pull based mechanism that relies on polling for detection of changes. In the domain of mobile collaboration, bandwidth is too expensive to apply polling and the frequency of events is too unpredictable to select an appropriate polling interval. Hence, in respect to point (1) we limit our comparison to pure publish/subscribe systems. Our hybrid approach reduces the message overhead by substituting queries for short-lived subscription. Based on the data from Table 3, the protocol overhead of a subscribe roundtrip (consisting of a subscribe request, response and one notification) is nearly three times as high as a simple query request response message exchange (2910 bytes to 1100 bytes). Context information itself is not included in either case. The advantage of the pull mechanism is even higher if we take possible unsubscribe requests and responses into account. Nevertheless, we have to keep in mind that queries are only superior to subscriptions when it comes to retrieving existing facts rather than events.

When we discuss point (2) that focuses on the ability to specify the level of change and not only the whole context structure, we have to make some assumptions. For a hierarchy consisting of five levels and assuming the subscription to be evenly spread, Table 4 displays the difference between a hierarchy-aware subscription system and a regular one. In the first case, events happen equally on every level, where as in the second case the more likely situation of fine-grained changes happening more often that coarse-grained ones is presented. Under the former assumptions, level-based subscription saves 40%, while under the latter assumptions the number of notifications is reduced by 53%.

Table 4 Context events produced by a level-based subscription mechanism and a regular one

Level	Sub	Events	Nfy w/	Nfy w/o	Improvement
Case 1					
L1	1	1	1	5	
L2	1	1	2	5	
L3	1	1	3	5	
L4	1	1	4	5	
L5	1	1	5	5	
Total			15	25	40%
Case 2					
L1	1	1	1	15	
L2	1	2	3	15	
L3	1	3	6	15	
L4	1	4	10	15	
L5	1	5	15	15	
Total			35	75	53%

Subscriptions are evenly spread across levels (one at each level). In addition, in case (1) events are evenly occurring whereas in case (2) events at lower levels are linear more likely than at higher levels.

Reducing bandwidth usage does not stop at level-based subscription. As our mechanism enables subscriptions and queries to specify which part of the hierarchy to transmit, the amount of data is further limited. For our three hierarchies, we have listed in Table 5 the average context content size for events at each level. To obtain these data, we created random (within a certain scope of choice) hierarchy data for four (respectively six) entities.[6] Then, queries at each level and data type were issued and the response size collected. We then aggregated the value of each level from the available entities and test runs. For queries and subscriptions in our scenario (as listed in Table 2), we achieved an improvement of 29% up to 76% of payload reduction. As push and pull of context information is based on the same data structure, these numbers are valid for notifications and query result messages.

Further reductions occur from the use of Dominance Rules. However, as the rules and respective application are domain specific as well as the affected context events non-deterministic, no improvement results are possible to predict.

In general, the right choice of subscriptions and queries as well as the required level and return type greatly influences the amount of data transmitted and exhibits a lot of potential for improvement beyond the presented results.

7.1 Discussion

Although running in an OSGi platform, our proof-of-concept implementation might not seem fit for a mobile environment on the first sight. It could be argued, that it is too heavyweight for mobile, resource constraint devices. This is certainly true if certain aspects remain unconsidered during the system's deployment phase. The presented framework presents a complete view on what components *can* be available on a certain device, but need not necessarily be.

[6] The entities were: Alice, Bob, Carol, Dave as well as AlicePDA, BobLaptop, CarolLaptop, CarolSmartphone, DavePDA, and DaveLaptop, respectively.

Table 5 Average context query results in bytes for Activity hierarchy, Reachability hierarchy and DeviceStatus hierarchy

	Full	Exact	Lowerincl	Upperincl
Activity				
L1	3368	636	3368	636
L2	3368	783	2958	1193
L3	3368	675	2442	1642
L4	3368	1068	1953	2484
L5	3368	1111	1111	3368
Reachability				
L1	2724	639	2724	639
L2	2724	615	2318	1026
L3	2724	831	1932	1624
L4	2724	1334	1334	2724
DeviceStatus				
L1	2508	1043	2508	1043
L2	2508	674	1705	1477
L3	2508	692	1271	1929
L4	2508	818	818	2508

Mobile devices featuring sufficient processing power and storage capacity such as Laptops require no adaptation of the presented software. Compared to hybrid peer-to-peer networks, such machines resemble super nodes that provide context information and services to clients that are more lightweight. These in turn require solely implementing the context subscription mechanism without the need for a context system or hierarchy adapter. On the other end of the scale reside pure context servers, which provide context information without subscribing to any. These nodes are also the most likely to feature access to several context systems.

A further issue concerns XML processing. Regular tools for parsing, validating, analyzing, or transforming come with a large overhead. For example, keeping a complete DOM tree in a smart phone's memory is not an option. Instead, techniques such as a typed, pull-based XML parser need to be employed on these devices. However, this will be part of future implementations.

Clearly, optimizing context sharing and processing alone does not address all mobility concerns. Web service related aspects such as service discovery, selection, and composition remain outside the scope of this paper. Furthermore, service continuity and message reliability are not addressed in this paper. On the one hand, we argue that latter issue does not present a major concern. Each participant simply relies on its local view rather than trying to establish a consistent one on a global level which in highly dynamic environments might not be possible (with a reasonable amount of effort). On the other hand, there is work available already dealing with such problems. Papers by Lee and Fox [6], Jorstad et al. [7], Chakaborty et al. [8], Maamar et al. [9], or Zahreddine and Mahmoud [10] address some of these areas.

Furthermore, in using hierarchies we require context to be available at a rather aggregated state. This reduces the applicability of our framework concerning context sensor-networks or sharing of sensor data. However, we argue that sharing context makes more sense at a high aggregation level that is most useful to end-users.

Finally, we emphasise that the presented architecture is not intended to replace a context system but should rather be seen as an add-on/plug-in/partial substitute for the distribution mechanism. As such, it enables a more efficient sharing of context. By using a SOA-based approach, we decouple the context distribution from the process of sensing, aggregating, reasoning and storing. Greater decoupling also allows context sources to be reused and exchanged more easily. This cannot be said of the context frameworks presented in the following section.

8 Related work

Context-awareness has been the focus of many research efforts. Most of the available toolkits focus on gathering, aggregating and providing context information—few of them in a distributed manner. Some of the following frameworks would present potential candidates for integration with our context-aware service platform.

Baldauf et al. [11] analyze representational applications concerning architecture, context model and context lifecycle to generate a broad overview on context-aware systems.

Biegel and Cahill [12] present a framework for developing mobile, context-aware applications. They also introduce the concept of a context hierarchy. However, their hierarchy has the notion of a task tree rather than structuring context information into various levels of detail. Similar to our approach, components of their framework (sentient objects) are context producers and consumers at the same time. Communication, however, happens only in one direction, whereas our approach allows for more customized context transfer as we apply a hybrid push-pull based mechanism. Finally, the authors also include a form of context dominance. Yet, their focus is not on context information relevance, but on excluding rules that are of no importance in the current context state.

Web Service Context (WS-Context) [13] is a specification proposed by OASIS to describe the context of an activity—composed of several Web services. WS-Context defines methods to pass context by value or just by reference. Thus, the receiving service can obtain the actual information from the *context manager service*. Context information itself can be structured hierarchically as WS Context includes an optional element, which refers to the general parent context. Yet, WS Context provides no means to access context at a specific level of granularity or to navigate down towards context child-elements. Furthermore, context information is merely propagated along the chain of invoked Web services, lacking capabilities for subscribing to context changes.

The service-oriented context-aware middleware (SOCAM) by Gu et al. [14] utilizes a two level ontology to model context information. An upper-level ontology describes generic concepts whereas the lower-level ontologies define specific domains. The SOCAM framework includes push- and pull-mechanisms for retrieving context information as we do. However, such information is only gathered but not forwarded to other services and solely provided to the applications build on top of SOCAM. This restricts the applicability for composite services. Furthermore, access to context information lacks the notion of granularity.

Although applying agents, the CoBrA framework by Chen et al. [15] relies on a centralized context broker architecture. As we argued above, centralized storage of context is not an efficient method for managing and processing context in a mobile

environment. Furthermore, sharing of context, granularity, and context dominance is not explicitly supported.

Costa et al. [16] designed a platform for mobile context-aware applications. Context information is shared by subscribing to this platform using the WASP Subscription Language (WSL). Yet, their subscription language lacks the support for information granularity. Furthermore, WSP depends on the proposed object-based context model, thus restricting its use.

The Solar middleware by Chen and Kotz [17] provides a platform for context-aware mobile applications consisting of one star and several planet nodes. Client applications need not collect, aggregate, or process context themselves but subscribe to context changes at the central star. This component computes an event flow graph and distributes the context processing steps across the available planet nodes to reuse existing context processing operators and filters. This approach differs from ours as we propose to subscribe to changes and transmit context only to the nodes actually requiring this information. Besides, context granularity is not supported.

To the best of our knowledge, no context aware system explicitly supports context granularity. However, da Rocha and Endler [18] have recently proposed context granularity as an important part of distributed context-aware systems but they did not go beyond pointing out this aspect.

9 Future work

Open work and research is manifold. In the near future, major work effort will consist of designing and deploying tools to aid the developer in designing hierarchies, mapping hierarchies to context systems and support for defining dominance and privacy rules on top of those hierarchies. Furthermore, algorithms to match similar hierarchies, merging of hierarchies and automatic mapping of context models to hierarchies are on our agenda. In addition, providing hierarchies at a repository for downloading and sharing might prove worth realizing. At the same time, we will analyse the integration of granularity levels into the whole context life cycle.

Another promising area of research will be context learning. We believe that by enabling services to learn from previous situations, context information will not only facilitate better-adapted services but also pro-active service aggregation. Analyzing which services required what context under certain conditions just will be the beginning. Elaborating on context granularity combined with rules and learning will enable services to anticipate situations and enforce appropriate access to resources without user interference. A first step into that direction will consist of describing hierarchies by means of ontologies.

10 Conclusion

This paper discussed issues of context sharing in mobile environments. After identifying current challenges, we elaborated on the aspect of distribution. The aim in highly dynamic systems is reducing the amount of transferred context updates. The techniques we introduced achieve this goal by means of context hierarchies and a fine-grained notification mechanism. This includes structuring context informa-

tion according to levels of granularity to limit change notification to the required amount of details. We define hierarchies as problem specific views on a given context model. Hence, we leave the utilized context system unchanged and merely provide a hierarchy-to-model mapping for extracting and subscribing to information. In addition, the concept of context dominance enables relevance-based context transfer. A scenario from the domain of distributed collaboration served as a test-bed to outline the benefits of context granularity and dominance.

In the second part of our work, we combined these approaches to form CoSAr, a context sharing architecture. Our newly formed *Context Access control, Subscription and Query Language* (CASQL) based on XML standards allows for fine-grained context control. Thus, not only reduces granularity-based context sharing bandwidth usage but also improves on privacy issues that could not be solved without hierarchical structuring. Evaluation based on a prototype implementation showed potential for significant reduction in transferred context.

```
boolean matchHierarchy(hierarchy expH, hierarchy availH)
{
    global list foundBackupValues = empty;
    for each (expH.levels level)
    {
        if (level.contains (value.isRequired or
                            value.isOptionalBackup or
                            value.isBackup)
        {
            if ( checkLevel(level, availH.rootLevel) == false)
                return false;
        }
    }
    return true;
}

boolean checkLevel(level expL, level availL)
{
    if (availL == null)
        return false
    if (expL.id == availL.id)
        return checkValues(expL.values, availL.values)
    else
        checkLevel(expL, availL.child)
}

boolean checkValues(list expValues, list availValues)
{
    List reqValues = expValues(isRequired)
    List opBackupValues = expValues(isOptionalBackup)
    List backupValues = expValues(isBackup)
    List notListed = (reqValues.subSetOf(availValues)
    if (notListed = not empty)
        return false;
    foundBackupValues.append(reqValues.subSetOf(backupValues)
    List notListed2 = (opBackupValues.subsetOf(availValues)
    for each (notListed2.value value)
    {
        if  (foundBackupValues.doesNotContain(value))
            return false;
    }
    return true;
}
```

Listing 3 A Hierarchy Matching Algorithm

We conclude that describing context through granularity hierarchies provides a means that requires further research but already now enables efficient handling of major challenges in context-aware dynamic environments.

Appendix A—Hierarchy matching algorithm

The hierarchy matching algorithm takes two arguments as input: the expected hierarchy (as a tree of all known values) and the complete available hierarchy (also as a tree of values). The value nodes of the expected hierarchy tree are marked as optional, required, or optional with backup value. In the last case, the specific value need not be available if there is a given parent value available instead. Those backup values are separately marked as such. The pseudo code in Listing 3 outlines our matching algorithm.

Acknowledgments Part of this work was supported by the Austrian Science Fund (FWF) under grant P18368-N04 Project OMNIS and EU STREP Project inContext (FP6-034718).

References

1. M. Bazire and P. Brézillon, "Understanding context before using it," in Modeling and Using Context: 5th International and Interdisciplinary Conference CONTEXT 2005, 2005, pp. 29–41.
2. A. Dey and G. Abowd, "Towards a better understanding of context and context-awareness," in Workshop on the What, Who, Where, When, and How of Context-Awareness at CHI 2000, 2000.
3. S. Dustdar, T. Hoffmann, and W. van der Aalst, "Mining of ad-hoc business processes with Teamlog," Data and Knowledge Engineering, vol. 55, no. 2, pp. 129–158, 2005.
4. A. Dey, Providing Architectural Support for Building Context-Aware Applications, PhD thesis, Georgia Institute of Technology, 2000.
5. A. Ferscha, C. Holzmann, and S. Oppl, "Context awareness for group interaction support," in MobiWac '04: Proceedings of the Second International Workshop on Mobility Management & Wireless Access Protocols, New York, NY, USA, ACM Press, 2004, pp. 88–97.
6. S. Lee and G. Fox, "Wireless reliable messaging protocol for web services (WS-WRM)," in IEEE International Conference on Web Services, 2004, pp. 350–357.
7. I. Jørstad, T. van Do, and S. Dustdar, "A service continuity layer for mobile services," in IEEE Wireless Communications and Networking Conference, 2005, pp. 2300–2305.
8. D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha, "Service composition for mobile environments," Mobile Networks and Applications, vol. 10, no. 4, pp. 235–451, 2005.
9. Z. Maamar, Q.Z. Sheng, and B. Benatallah, "On composite web services provisioning in an environment of fixed and mobile computing resources," Information Technology and Management, vol. 5, pp. 251–270, 2004.
10. W. Zahreddine and Q. Mahmoud, "An agent-based approach to composite mobile web services," in 19th International Conference on Advanced Information Networking and Applications, 2005. AINA, 2005, pp. 189–192.
11. M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," International Journal of Ad Hoc and Ubiquitous Computing, 2006 forthcoming.
12. G. Biegel and V. Cahill, "A framework for developing mobile, context-aware applications," in Second IEEE Annual Conference on Pervasive Computing and Communications. PerCom 2004, pp. 361–365, 2004.
13. M. Little, E. Newcomer, and G. Pavlik, Web Service Context Specification (WS-Context), OASIS, 2004.
14. T. Gu, H. K. Pung, and D.Q. Zhang, "A middleware for building context-aware mobile services," in 59th Vehicular Technology Conference, VTC 2004, pp. 2656–2660, 2004.

15. H. Chen, T. Finin, and A. Joshi, “An ontology for context-aware pervasive computing environments,” Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, vol. 18, no. 3, pp. 197–207, 2003.
16. P.D. Costa, L.F. Pires, M. van Sinderen, and J.P. Filho, “Towards a service platform for mobile context-aware applications,” in 1st International Workshop on Ubiquitous Computing—IWUC 2004, pp. 48–61, 2004.
17. G. Chen and D. Kotz, “Solar: An open platform for context-aware mobile applications,” in First International Conference on Pervasive Computing (Short Paper), pp. 41–47, 2002.
18. R.C.A. da Rocha and M. Endler, “Context management in heterogeneous, evolving ubiquitous environments,” IEEE Distributed Systems Online, vol. 7, no. 4, 2006.