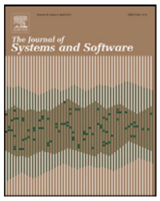
The Journal of Systems and Software 86 (2013) 1107–1123

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss



Federated broker system for pervasive context provisioning

Saad Liaquat Kiani[a], Ashiq Anjum[b], Michael Knappmeyer[c], Nik Bessis[b,*], Nikolaos Antonopoulos[b]

[a] Faculty of Environment and Technology, University of the West of England, Bristol, UK
[b] School of Computing and Mathematics, University of Derby, Derby, UK
[c] Ratiodata IT-Lösungen & Services GmbH, Münster, Germany

ARTICLE INFO

Article history:
Received 21 September 2012
Received in revised form
24 November 2012
Accepted 27 November 2012
Available online 6 December 2012

Keywords:
Context provisioning
Federated broker model
Distributed systems design

ABSTRACT

Software systems that provide context-awareness related functions in pervasive computing environments are gaining momentum due to emerging applications, architectures and business models. In most context-aware systems, a central broker performs the functions of context acquisition, processing, reasoning and provisioning to facilitate context-consuming applications, but demonstrations of such prototypical systems are limited to small, focussed domains. In order to develop modern context-aware systems that are capable of accommodating emerging pervasive/ubiquitous computing scenarios, are easily manageable, administratively and geographically scalable, it is desirable to have multiple brokers in the system divided into administrative, network, geographic, contextual or load based domains. Context providers and consumers may be configured to interact only with their nearest, relevant or most convenient broker. This setup demands inter-broker federation so that providers and consumers attached to different brokers can interact seamlessly, but such a federation has not been proposed for context-aware systems. This article analyses the limiting factors in existing context-aware systems, postulates the design and functional requirements that modern context-aware systems need to accommodate, and presents a federated broker based architecture for provisioning of contextual information over large geographical and network spans.

1. Introduction

Ubiquitous computing is one of the latest steps in evolution of computing paradigms, which models integration of information processing into everyday effects and activities without explicit involvement of users. Weiser's vision (Weiser, 1991) of the proximate future depicts a ubiquitous world where interconnected smart entities are able to provide information on '*anything, any time, anywhere*'. Since the inception of this concept nearly two decades ago, ubiquitous computing research has been dealing with the possibilities of future; its progress has faced not only technological challenges but is also concerned with anticipation of future trends of human behaviour. Central to the vision of the ubiquitous computing environment is the processing and communication of information between smart entities. The information may relate to inhabitants of the environment, smart appliances or physical characteristics of the environment itself and is labelled as *context*. Context-aware systems aim to use the knowledge of user and environment context to *proactively* provide services relevant to the user's situation.

Context management toolkits and middleware frameworks have been developed to assist in developing context-aware applications and extending their functional range, e.g. Hallsteinsen et al. (2012). A context-aware system usually comprises several context management functionalities. Most important are acquisition, reasoning and distribution of contextual information related to entities (user, device, environment, network, etc.). Within this simple definition, we can divide the system components involved in context provisioning into context managers, context consumers, context providers or a combination thereof. Furthermore, we define the term *context provisioning* to encompass the set of functions that facilitate the communication and coordination of contextual information amongst the context consumers, providers and managers. The provisioning of contextual information in context-aware systems has usually been carried out through context brokers, which facilitate *context-consuming* components in the system to retrieve context from *context-provisioning* components. The mechanism of context provisioning through context brokers, shortcomings in existing realisations of this mechanism and possible improvements to accommodate emerging ubiquitous computing scenarios form the core focus of this article.

* Corresponding author at: Room E509, Kedleston Road, University of Derby, Derby, UK. Tel.: +44 01332592108; fax: +44 01332597741.
E-mail addresses: saad2.liaquat@uwe.ac.uk (S.L. Kiani), a.anjum@derby.ac.uk (A. Anjum), michael.knappmeyer@ratiodata.de (M. Knappmeyer), n.bessis@derby.ac.uk (N. Bessis), n.antonopoulos@derby.ac.uk (N. Antonopoulos).

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2012.11.050

A number of prototypical context-aware systems have been developed that showcase context-awareness in one domain or the other, but large-scale context provisioning and adoption of context-aware applications and services has proved elusive so far due to multi-faceted challenges in this area. One of the main challenges is the diversity of settings in which the context-awareness related research takes place, highlighted by the heterogeneity of devices and fluid pattern of human behaviour that evolves with the changing technological landscape. Although laboratory settings do provide a controlled environment, the reality remains that the communication and processing infrastructure is inherently heterogeneous across the complete range of devices. Moreover, the consumers and producers of contextual information (applications and services) will be spread across large geographical and network spans as smart ubiquitous environments take hold in the digital ecosystem. This facet of smart environments, coupled with the increasing pervasiveness of mobile devices, their increasing role in human-computer interaction and the inherent mobility of device-based context consuming and producing components, points towards an increasingly distributed and large-scale provisioning function of context-aware systems.

Existing context-aware systems are not ideally placed to meet the discussed domain challenges and facilitate their use in the emerging ubiquitous computing scenarios. Prominent shortcomings in existing systems include (1) the predominant trend of utilising a central context management component, e.g. a context broker, for coordinating context-awareness related functions, (2) a dominant focus on designing for a static topology of the interacting distributed components, (3) the presumption of a single administrative domain or authority and context provisioning within a single administrative, geographic or network domain, (4) a limited support for accommodating mobility of context providing and consuming components, and (5) a lack of standardisation with respect to a simple, flexible and extensible context model that can accommodate contextual information exchange between heterogeneous actors.

The common design approach of utilising a centralised managing component for coordinating the flow of contextual information between context providers and consumers results in an architecture that places a functionally critical burden on the management component. A consequence of this design approach is the managing component becomes the hub of all functional activities, which increases the coupling between context related clients and services. Not considering the possibility of dynamic changes in the topology of interconnected components, designing a system which assumes all operating procedures will fall under the control of a *single administrator*, and a data model that does not facilitate the contextual information exchange between *heterogeneous actors* (devices, sensors, other digital artefacts, etc.) of varying computational capabilities are the major shortcomings of existing context provisioning systems. Furthermore, existing systems target context provisioning of contextual information within a limited geographic or network domain, which disregards the practical constraints (availability, authorisation, quality of service, etc.) that come into play when a context-aware system has to operate in a real-world setting where consumers and providers of contextual information are likely to be distributed across large geographical spans and multiple network domains. The limitations of existing systems with respect to the scale of their coverage are compounded by the increasing mobility of modern users of such systems, which results in mobility of the context providing and consuming components that execute on user devices. The increasing participation of devices and sensors of varying computational capabilities in context-awareness related functions requires a context model that is not only simple enough for the lowest common denominator amongst the heterogeneous devices, but flexible and extensible enough to accommodate newer domain concepts as and when they become part of the contextual knowledge base. Due to the narrow domain focus of most of the existing context-aware systems, there is also room for improvement in terms of standardising such a context model.

This article describes a *federated broker based context provisioning system* that aims to address the listed challenges and overcome the shortcomings in existing context-aware systems. The principle theme of our system is based on the distribution of the context management responsibility amongst an arbitrary number of inter-connected and cooperative context brokers. Such a federated broker model has not been demonstrated in the domain of context-aware systems. Each context broker in our system, prototyped as the *Context Provisioning Architecture*, manages a subset of clients (context consumers and providers), thus reducing the functional burden on individual context brokers. The distributed brokers form an overlay network of brokers (*federation*) by exchanging information about their individual clients and facilitating their clients in exchanging contextual information with clients of remote context brokers. The segregation of brokers in the federation improves not only the load scalability in the overall system, but is also well suited for providing administrative scalability whereby different brokers, and hence their local clients, can be associated with different administrative authorities. The federated context brokers provide asynchronous communication interfaces to their clients and neighbouring brokers for exchanging contextual information and administrative information related to their registered clients and their capabilities. In addition to decoupling the interacting components, this communication model also accommodates the topology changes due to mobile or disappearing/re-appearing components (brokers or their clients) by propagating such changes across the federated brokers. Furthermore, an XML based context model is utilised to specifically address the simplicity, flexibility, extensibility and heterogeneity constraints of this domain discussed earlier in the section.

We present the work related to the core problem areas in Section 2. Based on the related work, we also categorise the key issues in state of the art, which serve as our motivation for the development of new concepts and the creation of an innovative federated broker model for a distributed context provisioning system. The functional description of our proposed architecture, the context model, methods for facilitating the inclusive role of mobile devices and the inter-broker federation model are presented in Sections 3–6. We provide an analysis and evaluation of our model, in light of key issues and challenges, in Section 7 and conclude the discussion in Section 8.

2. Related work

Context-aware systems can be characterised by the set of functions they perform, which include data acquisition, context synthesis, context storage, context coordination and communication (provisioning) and serving as an application/service platform. The execution of these functions, under a shared model of context and its representation, has mostly been realised using context broker based architectures. In the *context broker* approach, the broker performs the functions of collecting and synthesising context from sensor data and other information sources. Clients of the broker access it remotely to access context or raw data to process locally. This approach is useful for accommodating low-capability clients by sharing the processing burden and sensing resources. The context broker architecture allows remote components to access context but the data acquisition function is restricted by the limitation in communication range of sensors and other information sources. To overcome this shortcoming, the context broker architecture has evolved further in terms of distribution of its constituent components, e.g. distribution of the data acquisition function into multiple remote data acquisition modules that acquire data from their assigned sources and push it to the central server. This further distribution of functionality has transformed context-aware systems into truly distributed systems where each function may execute on different hosts in a network and a central broker coordinates the flow of information and control between these components. Other factors that have influenced the adoption of this architecture include availability of a large number of distributed information sources and sensors, dedicated reasoning components, mobility of modern day users and abundance and increased usage of mobile devices.

Distributing functionality into separate components in such systems results in greater flexibility and changeability. However, if each distributed component handles communication with other components itself, the system faces several dependencies and limitations. For example, the system becomes dependent on the communication mechanism used, clients need to know the location of servers before they can function fully, services are needed to provide discovery and lookup of other components and usually force distributed components to be restricted to a common implementation technology. To overcome such limitations and provide a design time solution for the potential issues raised by the constraints, Buschmann et al. (1995) proposed the Broker architectural pattern. Broker architecture in its various forms exists as a middleware technology that manages communication and coordination between distributed objects or software components. Brokering components have also been employed in context-aware systems, CoBrA (Chen, 2004), MobiLife (Kernchen et al., 2006) and C-CAST (Moltchanov et al., 2010) being prominent demonstrations of the use of context brokers. However, the use of brokers in these systems is different from the concept of brokers advocated by the broker pattern. The main difference lies in the amount of functional responsibility assigned to these brokers, e.g. in case of CoBrA, the broker is an agent that contains a context acquisition module, context knowledge base, context reasoning engine and a privacy management module. Similarly, the context broker in the C-CAST system is responsible for storing context, providing querying, entity resolution, context caching and user management services in addition to context query resolution. With this collection of functional responsibilities the context brokers in these systems resemble full-fledged context servers containing most of the context-aware middleware rather than representing the lightweight brokering role envisaged by the broker architectural pattern. The possibility of federating multiple *active spaces* together has been proposed in the Gaia middleware but realisation of the concept has not been demonstrated (Román et al., 2002, p. 9).

The federation of context brokers effectively distributes the set of context providing and consuming clients in the system between multiple context brokers. Because the clients normally interact only with their local broker, their contextual queries (subscriptions) and responses (notifications) have to be routed across the overlay network of brokers. The routing of these subscriptions and notifications is based on their content and hence *content-based* routing protocols can be effectively utilised for this purpose. *Flooding* is the most basic form of routing in which a broker forwards a notification received from a local client or a neighbouring broker to all other neighbours. Flooding based routing is simpler to implement but is not optimal. The main advantage of flooding is the increased reliability provided by this routing method. The notification messages are sent at least once to every broker, guaranteeing that the notifications will reach at least the local broker of the intended destination client. In addition, a notification message will reach the destination through the shortest possible path. However, flooding is very wasteful in terms of the total bandwidth utilised in the network, e.g. a notification message may only have one destination but it is sent to every broker, increasing the maximum load placed upon the network. *Simple filter-based routing* (Mühl, 2002, p. 47) improves upon the trivial flooding technique by updating the broker routing configurations in response to subscribing and unsubscribing clients. When a client sends a subscription message to its local broker, the broker floods the information to its neighbouring brokers. Subsequently any notification that matches a subscription is only sent to the relevant broker(s) based on the information in the flooded subscription. Therefore, in contrast to flooding based routing, only subscriptions are flooded into the overlay network of brokers while notifications are forwarded only to matching brokers. Using *advertisements*, which can limit the propagation of subscriptions to those brokers where matching notifications are potentially produced, the efficiency of the *simple filter-based routing* approach can be enhanced. This extension is employed in the Context Provisioning Architecture. Other inter-broker routing mechanisms that may be used include identity-based routing, covering-based routing and merging-based routing (Carzaniga, 1998).

Another issue that arises in federated broker based systems is that of mobility of components (clients, brokers), which affects the topology of the overall network. Research in subscription and notification routing systems has mostly focussed on static topologies (Podnar and Lovrek, 2004) where mobility concerns are delegated to the applications (consumers, producers of events), e.g. in JEDI, which uses explicit *moveIn* and *moveOut* operations to relocate clients. Hence, mobility management is delegated to the applications/clients and not controlled by the brokers. Podnar and Lovrek (2004) propose an approach for managing mobility in which the brokers store persistent notifications until their validity period expires. Huang and Garcia-Molina (2004) have also suggested algorithms for managing mobility in single and multiple broker based systems but such algorithms have not been incorporated and analysed in context-aware systems.

The context model used in a federated broker based system, in which individual brokers may be under different administrative domains involving heterogeneous clients, is critical to the interoperability of the overall system. Korpipää et al. (2003) have specified some basic requirements for designing a context model in terms of simplicity (for easy manipulation and reasoning), flexibility (modification of existing concepts), generality (range of concepts that can be modelled) and expressiveness (encompassing the properties of the modelled concepts). ContextML, the XML based model used in the Context Provisioning Architecture, meets these requirements to an acceptable degree. ContextML schema has its roots in the earlier efforts of the Context Representation Framework (MobiLife Project, 2006) of the MobiLife project. Further extension to the original ContextML model has been carried out in the Context Casting (C-CAST) project; our earlier work (Knappmeyer et al., 2010b) describes the basic ContextML model used in C-CAST. Additions to ContextML carried out in the C-CAST project primarily focus on extending the model's capability in representing real world contextual concepts. We have further extended the basic ContextML model developed in earlier works to be used in a publish/subscribe system such that domain concepts can be subscribed to, published and notified.

The examination of state of the art shows that existing context-aware systems do not *holistically* address the issues of heterogeneity of involved actors, scalability, limited domain focus, simple and flexible context model, and federation between cross-domain brokers. This observation serves as the motivation for the development of new concepts and the creation of an innovative model for a federated broker based context-provisioning system. The following sections elaborate a functional description of our design for such a system, the context model, methods for facilitating the mobility of components, and discuss how these individual challenges are overcome.

3. Federated broker model for context provisioning

Based on identification of the key issues in designing context-aware systems capable of large-scale context provisioning, this section presents the design and architecture of the Context Provisioning Architecture. We present our case for using a broker based architecture and relate to the utilisation of brokers in distributed systems in general and context-aware systems in particular. Main components of the Context Provisioning Architecture are discussed and the context-modelling scheme used in the system is also explained. Specialised functions, such as caching and broker discovery are also described. We present a case for the federation of context brokers that specifies a coordination model for context exchange between components distributed across the brokers in the federation. Special cases of disappearing brokers and mobile clients are also highlighted.

The Context Provisioning Architecture is based on the producer–consumer model in which context related components take the roles of context providers or context consumers. These basic entities are interconnected by means of context brokers that provide routing, event management, query resolution and lookup services. The following paragraphs describe these three main components of the architecture.

Context consumer A context consumer (CxC) is a component (e.g. a context based application) that uses context data. A CxC can retrieve context information by sending a subscription to the context broker (CxB) or a direct on-demand query and context information is delivered when and if it is available.

Context provider The context provider (CxP) component provides context information. A CxP gathers data from a collection of sensors, network/cloud services or other relevant sources. A CxP provides context data only further to a specific invocation or subscription and is usually specialised in a particular context domain (e.g. location).

Context broker A context broker (CxB) is the main coordinating component of the architecture. It works as a facilitator between other architectural components. Primarily the CxB has to control context flow amongst all attached components, which it achieves by allowing CxCs to subscribe to context information and CxPs to deliver notifications.

A depiction of the core system components described above is presented in Fig. 1, emphasising the complementary provision of synchronous and asynchronous context related communication facilities. Our discussion in this article will focus only on the asynchronous context communication (subscriptions and notifications) in the Context Provisioning Architecture, which is better suited for utilisation in the federated broker based setup. Synchronous queries and responses involve blocking mode of communication

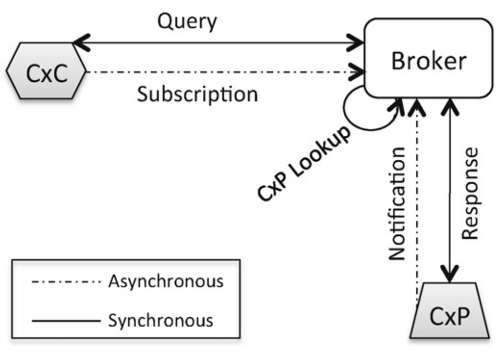


Fig. 1. Functional components of the Context Provisioning Architecture.

and are likely to induce delays when context has to be looked up across broker boundaries. In practical implementation of the Context Provisioning Architecture, we only utilise synchronous queries and responses when the information about communication endpoints is already known, e.g. hardcoded CxP address in a CxC. A number of useful applications have been developed based on this architecture. Further details of this architecture and industrial trials are described in Knappmeyer et al. (2010a) and Zafar et al. (2009).

The design of the CxB is based on the set of functions it provides to the context consumers and providers. These functions are listed below:

Client registration and lookup CxCs/CxPs register with a broker by specifying communication end points and the type of context they require/provide. This function in turn enables the brokering function in which the broker can direct a CxC requesting a particular type of context to the correct provider(s).

Subscription and notification A CxC subscribes with the broker specifying the type and instance of context it requires and the duration for which the subscription remains valid. The broker can forward the subscription to the appropriate provider or filter context produced by a provider in order to satisfy the subscription. The broker notifies the consumer on availability of the subscription-satisfying context, or the context is directly communicated to the consumer by the provider.

Caching The broker can cache recently produced context in order to exploit the locality of reference, as done routinely in internet communications, to improve the overall performance.

Federation The distributed brokers collectively form an overlay network of brokers that manage local clients (consumers and providers). This federation of context brokers is achieved through a coordination model that is based on routing of subscriptions and notifications across distributed brokers, discovery and lookup functions and is described in detail in Section 3.2.

The context broker offers these functions to CxPs and CxCs by exposing well-defined interfaces. Clients of a broker only communicate with the local broker. Queries, subscriptions and notifications are routed between brokers using a publish/subscribe communication paradigm. This model is described later in Section 3.2, a detailed theoretical model is also provided in our earlier work (Kiani et al., 2010b). The Context Provisioning Architecture provides two context communication mechanisms; one based on asynchronous semantics and the other on synchronous communication semantics. The availability of two complementary mechanisms is aimed at accommodating the varying access patterns of heterogeneous context consuming and producing applications and services. Asynchronous event-based queries (subscriptions/notifications) allow a context consumer to utilise an event-based publish/subscribe function in the broker for context queries and responses. Complementarily, context consumers can synchronously query for a particular context scope by invoking the context provider over HTTP and encoding the request parameters in the HTTP URL directly. Such complementary pull (synchronous) and push (asynchronous) mechanisms are common in distributed systems, e.g. Bessis (2009). The *lookup* function in the broker is used by the consumers to find the communication endpoint of the relevant context provider, after which the query and response takes place between consumers and providers directly without any participation of the broker. The broker also provides a *proxy query* function to resolve scope dependencies in consumer queries that include multiple scopes. Instead of requiring the context consumers to query each dependent scope from different providers, the broker satisfies the scope dependencies by querying for the dependent scopes when it receives a query about a context scope that is dependent on other scopes, e.g. weather context of a user is dependent on the location scope.

3.1. ContextML based data model

The data model specifies the format of the communication and coordination that takes place between context consumers, providers and the brokers. Clients of a broker are described using attributes, which are used for registration with the broker. Subscription and notification further require that the acquired context be annotated with meta-information that allows categorisation and matching of context into specific instances that can be compared with subscriptions and queries. The Context Provisioning Architecture utilises an XML based ContextML schema for coordination and communication of context information. ContextML specifies the model for context information, context subscription/notification and control messages as well (a detailed description is provided in our earlier work Knappmeyer et al., 2010a). *Entity* and *scope* are the two main concepts that drive the data model. Each exchange of context data is associated with a specific *entity*, which can be a complex group of more than one entity. An entity is the subject of interest (e.g. user or group of users) which context data refers to and it is composed of two parts: a type and an identifier. The type refers to the category of entities, e.g. username (for human users), IMEI (for mobile devices), SIP URI (for SIP accounts), room (for a sensed room) and group (for groups of other entities, e.g. usernames or IMEI numbers). Specific context information in ContextML is defined as scope and is a set of closely related context parameters. Every context parameter has a name and belongs to a certain scope. Scopes can be atomic or aggregated in a union of different atomic context scopes. Examples of context scopes of varying degree of complexity include location, weather, activity, situation, cellular, and Wi-Fi network identification.

Whenever a CxC requests or subscribes to a specific context scope, it receives a response encoded in the ContextML schema element *ctxEl* when context is available. *ctxEl* contains information about where the context has been detected and encoded (CxP), which entity it is related to (entity), what scope it belongs to, and the actual context data in the *dataPart* element. A graphical description of this element, along with ContextML schema elements of context subscriptions (*cxtSubscr*) is given in Fig. 2. The elements *par*, *parS* and *parA* are simple constructs to store name-value pairs and attributed collections (arrays and structures) respectively.

3.2. The federation model

A real world deployment of a broker-based context-aware system may incorporate context providers and consumers that are geographically distributed. To reduce management and communication overheads, it is desirable to have multiple brokers in the system divided into administrative, network, geographic, contextual or load based domains. Context providers and consumers may be configured to interact only with their nearest, relevant or most convenient broker. But this setup demands inter-broker federation so that the providers and consumers attached to different brokers can interact seamlessly. A simple event system implemented by an overlay network of distributed brokers for relaying subscriptions and notifications can satisfy this requirement.

Our system model for an overlay network of brokers working in a federation is based on the model presented by Mühl et al. (2006) and has been extended for context subscription/notification with the use of client advertisements. Mühl et al. have specified a theoretical framework for routing subscriptions/notifications using different routing algorithms and have also presented the validity proofs of these algorithms. The federation model of the Context Provisioning Architecture is modelled on this theoretical framework, which we have realised using ContextML based subscriptions, notifications and other control messages (e.g. advertisements, discussed later in this section). We have augmented the theoretical framework with caching and broker discovery functions. Specifically, the *broker discovery and registration service* (BDRS) enables the management of situations in which components disappear/reappear (whether planned or unplanned due to network issues) at the same or different communication endpoints. Furthermore, the theoretical framework specified by Mühl et al. only considers the case of mobile clients but not of mobile broker components, whereas the coordination model of the Context Provisioning Architecture enables the integration of mobile brokers in the framework as well. These additional features are discussed in detail after we describe the basic model in the following paragraphs. Each broker in the federation maintains registration tables for context consumers and providers and uses these tables to route context subscriptions and notification between the clients. Clients are required to send a keep-alive advertisement message periodically once they have registered with a context broker. Failure to receive keep-alive messages from clients results in automatic unregistration from the broker, i.e. they are considered to be offline. The conceptual development of the overall model is presented in the following sections.

System model The system model consists of a set of cooperating brokers that are arranged in a topology that is restricted to be acyclic. Each broker B_i manages a mutually exclusive set of local clients $L_{B_i} = \{\kappa_1, \kappa_2, \ldots, \kappa_n\}$ and $L_{B_i} \subset \mathcal{K}$ where $\mathcal{K}$ is the set of all clients in the system. The clients here refer to `CxCs` and `CxPs`. Each broker B_i is connected to a set of neighbouring brokers $N_{B_i} = \{\eta_{i1}, \eta_{i2}, \ldots, \eta_{in}\}$ and $N_{B_i} \subseteq \mathcal{B}$ where $\mathcal{B}$ is the set of all brokers in the system.

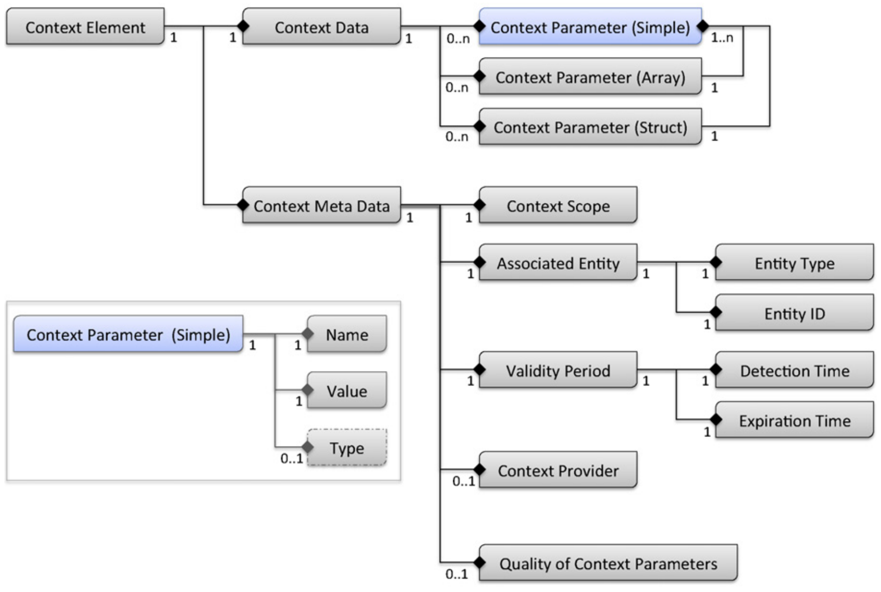
Subscriptions Subscription σ contains a stateless logical expression that is applied to a notification ν, i.e. $\sigma(\nu) \rightarrow (\mathit{true}, \mathit{false})$. A subscription can be given as a logical expression that consists of predicates that are combined by boolean or logical operators (and, or, not, >, =, etc.). Such operators can be used to impose constraints while defining subscriptions (e.g. attribute name = "weatherCondition"). Consider an attributed subscription that imposes a constraint on the value of a single attribute, e.g. age > 25. The subscription constraint can be defined as:

$$\gamma_i = (n_i, op_i, C_i) \tag{1}$$

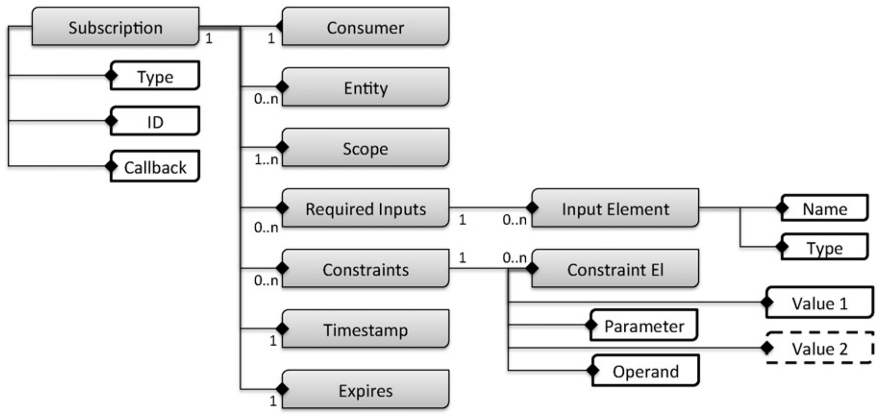
where n_i is the attribute name, op_i is a test operator and C_i is a set of constants that may be empty. The name n_i determines which attribute the constraint applies to. If a notification does not contain attribute named n_i then γ_i evaluates to false. A notification *matches* σ if $\sigma(\nu)$ evaluates to true. The set of matching notifications $N(\sigma)$ is defined as $\{\nu | \sigma(\nu) = \mathit{true}\}$.

Notifications The broker exposes two interfaces namely *pub(Notification ν)* and *sub(Subscription σ)* that allow the clients to publish or subscribe to events. The broker uses a *notify*(*Notification* ν) message itself to deliver notifications to local clients. Moreover, it uses a message *forward*(*Notificationν*) to forward notifications to neighbouring brokers (brokers who have clients subscribed for the current notification).

Client registration tables Each broker B_i maintains a client registration table R_{B_i}, which contains entries about its registered clients. A client κ_i registers with a broker by providing an advertisement that contains a unique



(a) Representation of context information in ContextML format (*ctxEl*)



(b) ContextML subscription schema element (*ctxSubscr*)

Fig. 2. A subset of ContextML schema elements. For brevity, only essential attributes are shown. (a) Representation of context information in ContextML format (*ctxEl*). (b) ContextML subscription schema element (*ctxSubscr*).

identifier I_κ and information about its communication endpoint URL_κ. In case the client is a CxP, the advertisement also contains the context *scope* served by the client. Neighbouring brokers exchange client registration tables amongst each other at regular intervals ΔX_R. Out-of-turn triggering of the client registration table update at a broker can occur when a new client registers with the broker so that the availability of a new client is immediately disseminated in the system.

Subscription tables Each broker B_i maintains a subscriptions table T_{B_i}, which contains entries about subscriptions related to its clients. Each entry in T_{B_i} is a pair $(\sigma, \mathcal{D})$ consisting of a subscription σ and a destination client $\mathcal{D} \in \kappa \cup N_B$. Hence each broker maintains subscription entries only for its local clients and neighbouring brokers and not of the whole system entities. When a client κ_j issues a subscription σ_k to the broker B_i that it is registered with, B_i adds an entry (σ_k, κ_j) to its subscriptions table T_{B_i}. Using the client registration table R_{B_i}, it determines which broker B_s can satisfy the subscription and updates B_s with the new subscription entry. B_s adds the entry (σ_k, κ_j) to its subscription table T_{B_s}.

4. Inter-broker coordination and federation

In the following sections, we describe how this model operates for one broker, two brokers and the general case of n number of brokers. It is assumed in the following discussions that the clients have already registered with their respective brokers and, in the case of two brokers, the brokers have already exchanged client registration tables, i.e. the brokers *know* which client/broker can satisfy a subscription pertaining to a particular scope.

For the trivial case of a single broker, consider that the local client κ_1 of broker B_1 subscribes with the broker with subscription σ_1 using the *sub*(*Subscription* σ_1) broker interface (Fig. 3). The broker saves this subscription in its subscription table and then determines that the local client κ_2 is capable of producing information that can satisfy the subscription. Broker B_1 forwards the subscription σ_1 to the local client κ_2. κ_2 monitors its produced data in case it matches any of the subscriptions it has received via the broker. If and when the subscription σ_1 is satisfied, κ_2 produces a notification ν_1 and sends it to the broker via the *pub*(*Notification* ν_1) broker interface. The broker consults its subscription table T_{B_1} and notifies the client that has the relevant subscription entry, in this case κ_1.

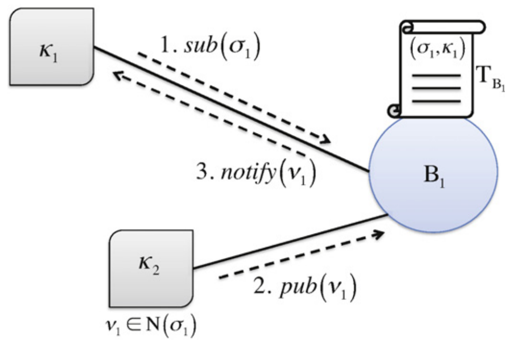


Fig. 3. Single broker coordination scenario.

4.1. A case of two brokers

Consider this case with the help of Fig. 4 where the local client κ_1 of broker B_1 subscribes with subscription σ_1. B_1 saves the entry (σ_1, κ_1) in its routing table T_{B_1} which was initially empty. It then sends the following message (table exchange) to its neighbouring broker B_2:

$$subTableUpdate(B_1, \sigma_1) \tag{2}$$

This message causes the broker B_2 to update its routing table with the entry (σ_1, B_1). Broker B_2 has two registered clients κ_2 and κ_3. B_2 forwards σ_1 to κ_2 considering it to be a source of matching notifications for this subscription by consulting its client registration table R_{B_2} and evaluating the *scope* entries in client advertisements; e.g. if the subscription is regarding weather updates of a certain area, then only a CxP that produces weather related context may be forwarded the subscription information; it may not be relevant to forward a weather related subscription to a client that produces context about proximity of a group of users. When κ_2 produces information that satisfies σ_1, it sends a notification ν_1 to B_2 along with the information that this notification satisfies the subscription σ_1, i.e. $\sigma_1(\nu_1) \rightarrow true$. B_2 analyses its subscription table T_{B_2} and finds entry (σ_1, B_1) and therefore forwards the notification ν_1 to B_1:

$$forward(B_1, \nu_1) \tag{3}$$

κ_1 is a local client of B_1. Therefore B_1 uses the *notify*(ν_1) procedure to notify the client with the notification ν_1.

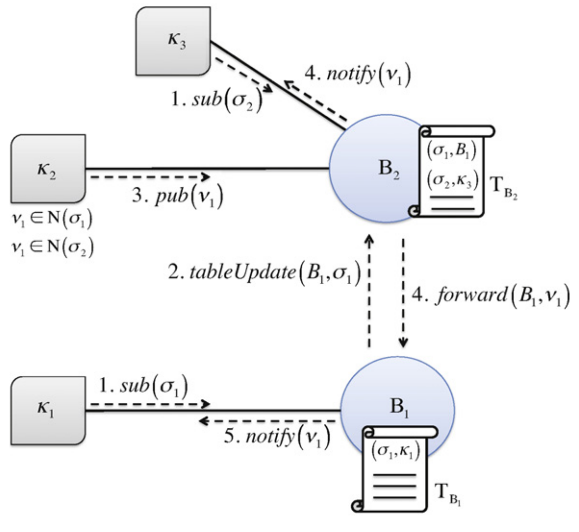


Fig. 4. Two broker coordination scenario.

Consider an additional subscription σ_2 received from the local client κ_3 of B_2, which is also satisfiable by κ_2. In this case, the subscription routing table T_{B_2} will contain an additional entry (σ_2, κ_3). Assuming that the notification ν_1 produced by κ_2 evaluates to true for both σ_1 and σ_2, B_2 will calculate the set of matching destinations as:

$$\Gamma_{B_2}(\nu_1) = \{B_1, \kappa_3\} \tag{4}$$

for the notification ν_1. For the local client κ_3, B_2 will invoke *notify*(κ_3, ν_1) locally. The other match B_1 is a remote broker and B_2 will invoke Eq. (3). For the local client κ_1 of B_1, B_1 will then invoke *notify*(κ_1, ν_1) locally.

4.2. The case of arbitrary number of brokers

Before considering the case of a broker federation consisting of an arbitrary number of brokers in the federation, the subscription model can be extended by allowing the clients to set up multiple subscriptions in one request, i.e. by declaring a set of subscriptions ς instead of a single subscription (and vice versa for un-subscription set χ). The case for n brokers builds on the previous case incrementally such that the subscriptions table updates are propagated throughout the broker federation. Using a simple routing mechanism, the subscription table updates are initiated in response to clients subscribing or un-subscribing. The subscription updates reach all brokers in the federation allowing them to update their subscription tables accordingly. If a client κ_i sends a subscribe $(\chi = \phi)$ or un-subscribe $(\varsigma = \phi)$ request to the parent broker B_i, following steps take place:

$$T_{B_i} \leftarrow T_{B_i} \cup \{(\sigma_j, \kappa_i) \mid \sigma_j \in \varsigma\} \tag{5}$$

$$T_{B_i} \longleftarrow T_{B_i} \setminus \{(\sigma_j, \kappa_i) \mid \sigma_j \in \chi\} \tag{6}$$

$$\forall(\eta \in N_{B_i}), \quad M_\sigma \longleftarrow \{(\sigma_j, \eta) \mid \eta \in N_{B_i} \setminus \kappa_i \wedge \sigma_j \in \varsigma\} \tag{7}$$

Eqs. (5) and (6) show the addition and removal of subscriptions in the broker table, respectively. If a broker cannot satisfy a subscription σ or subscription set ς locally, then it attempts to satisfy it by forwarding it to its neighbouring brokers, as shown in Eq. (7). For each neighbouring broker η in N_{B_i}, Eq. (7) returns a set M_σ of tuples $\{\sigma_j, \eta\}$ for each subscription σ_j in the subscription set ς. Each of the subscriptions is therefore forwarded to all neighbouring brokers (known to B_i) except the source client κ_i.

For notifications, the brokers use the *notify*(*Client* κ, *Notification* ν) procedure, as in simpler cases discussed earlier, to notify their local clients registered in L_B of any notifications they have received that match a particular client's subscription(s). In contrast to the case described in Section 4.1, if a broker receives a *forward*(*Notification* ν) message from a neighbour, it must evaluate if it needs to further forward the message to its other neighbours (in addition to notifying its local clients whose subscriptions match the notification). The list of neighbours a broker B_i forwards the notification to is determined by entries in the subscriptions table of that broker. Matching notification against a subscription $(\mathcal{T}^{\mathcal{D}})$ for a single destination $\mathcal{D}$ can be defined as:

$$\mathcal{T}^{\mathcal{D}} \equiv \{\sigma \mid \exists(\sigma, \mathcal{D}) \in T_B\} \tag{8}$$

i.e. a notification is forwarded from a broker to a destination $\mathcal{D}$ if a subscription entry exists in the subscriptions table of that broker. Following up on Eq. (8), all destinations $\mathcal{D}_B(\nu)$ to which a broker forwards a given notification ν is given by:

$$\mathcal{D}_B(\nu) \equiv \{\mathcal{D} \mid \mathcal{D} \in N_B \cup L_B \wedge \nu \in N(\mathcal{T}^{\mathcal{D}})\} \tag{9}$$

where $N(\mathcal{T}^{\mathcal{D}})$, is the set of matching notifications, i.e. a notification ν is forwarded to a destination $\mathcal{D}$ by a broker B if $\mathcal{D}$ is either a local client or a neighbouring broker with a valid subscription in the subscriptions table that is satisfiable by the notification ν.

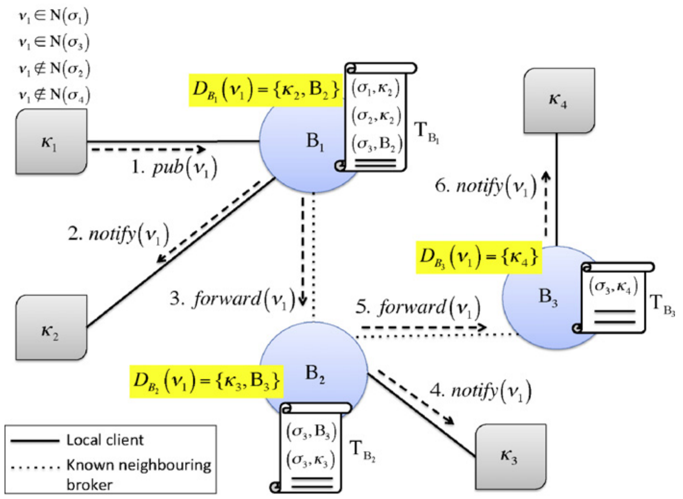


Fig. 5. Subscription and notification in *n* number of federated brokers.

Consider Fig. 5 where a network of three brokers exists with clients κ_1 and κ_2 of B_1, client κ_3 of B_2 and client κ_4 of B_3. Broker B_1 only has B_2 in its set of neighbouring brokers while B_2 has B_1 and B_3, and B_3 has B_2 in their neighbouring broker sets respectively (illustrated with dotted lines in Fig. 5). Consider a point in time where the subscription tables contain the following entries:

$$T_{B_1} = \{(\sigma_1, \kappa_2), (\sigma_2, \kappa_2), (\sigma_3, B_2)\} \tag{10}$$

$$T_{B_2} = \{(\sigma_3, B_3), (\sigma_3, \kappa_3)\} \tag{11}$$

$$T_{B_3} = \{(\sigma_3, \kappa_4)\} \tag{12}$$

Consider κ_1 publishes a notification ν_1 that matches subscriptions σ_1 and σ_3. Here, B_1 (of which κ_1 is a local client) delivers a notification received from κ_1 to its local client κ_2 due to entry (σ_1, κ_2) and forwards ν_1 to its neighbour B_2 due to entry (σ_3, B_2) (see Eq. (10)). At B_2, the client κ_3 is notified due to Eq. (11) and in addition the broker B_2 forwards the notification ν_1 to its neighbouring broker B_3 due to the entry (σ_3, B_3) in T_{B_2}. This step is the main difference from the case of two brokers; in the case of more than two brokers, each broker on receiving a notification consults its subscriptions table and the notification matching set M_ν in the notification to determine if the broker needs to forward the notification further or not. In case of two brokers, there was no such need when a broker received a notification from another broker as there were no more brokers involved. When the broker B_3 in this scenario receives the notification ν_1, it notifies its local client due to the entry (σ_3, κ_4) in T_{B_3} (see Eq. (12)). There are no more entries in T_{B_3} and therefore the broker B_3 is not required to forward the notification any further.

The coordination model described in the preceding sections uses *flooding* of subscriptions within the overlay network of brokers. In this coordination model, it was assumed that a broker will have the knowledge to determine which client or neighbouring broker can satisfy a subscription. In absence of this knowledge, the brokers have to *flood* the subscriptions to all their neighbouring brokers. Flooding is enforced due to lack of global knowledge about the characteristics of remote clients, i.e. which, if any, clients will be able to satisfy a subscription. The drawback of this approach is that subscriptions are forwarded regardless of whether or not matching notifications are potentially produced by clients of a particular broker. This drawback can be overcome if global knowledge exists in the broker network about the characteristics of local and remote clients that can assist in identifying *potential* candidates for subscription matching. This global knowledge can be induced in the system through a client advertisement procedure, which we describe in the following section.

5. Advertisements in the federation model

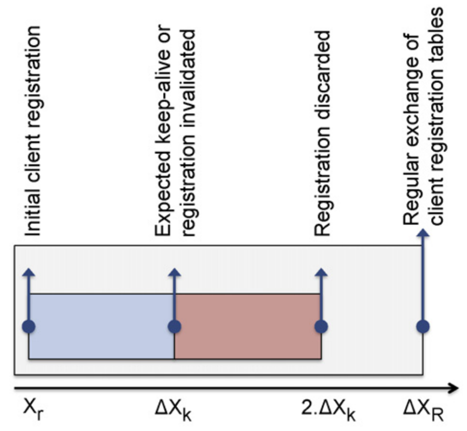
We extend our coordination model from Mühl et al.'s base model by using client *advertisements* via which all clients (context providers and consumers) register their characteristics with their local broker. Each client κ_i registers with its local broker using advertisement A_i. In case of a context provider type client advertisements contain, apart from other entries, the *scope* served by the client. The scope and other information in the client advertisements are used to calculate if a subscription and advertisement overlap. Formally a subscription and an advertisement overlap if the intersection of the set of scopes S_σ in a subscription and the set of scopes S_A is non-empty, i.e. *iff* $S_\sigma \cap S_A \neq \emptyset$.

A broker maintains a list of its registered clients (and those of neighbouring brokers) in a client registration table R_B. These tables are *normally* exchanged between neighbouring brokers at regular intervals (ΔX_R), building up the global knowledge about the type of clients registered with each broker in the broker federation. Out of turn update triggering of the registration tables is discussed later in this section. The global knowledge about association of clients with brokers and their characteristics is then used for forwarding subscriptions only to the relevant brokers, i.e. whose clients can potentially produce a matching notification, and flooding is avoided. Similarly, notifications can be directly routed to the context consuming clients as their registration entries contain information about their communication end point.

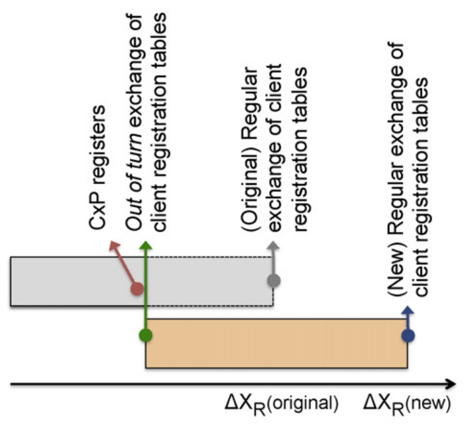
To safeguard against rogue clients, network disconnections and clients that disappear without unregistering properly, our coordination model requires that registered clients continue sending keep-alive advertisement messages to their local broker. Once a client registers with the broker at time x_r, keep-alive messages are expected within regular intervals Δx_k. If a client fails to send a keep-alive message after $x_r + \Delta x_k$, its registration is invalidated (but not removed). If a keep-alive or a new registration message from the same client is received at the broker within $2 \,.\, \Delta x_k$, its registration is considered valid again otherwise the registration is removed along with related subscriptions (see Fig. 6(a) for details). Conversely, there is possibility of a scenario where a client may consider itself properly registered while a broker may have discarded its registration, e.g. due to broker crash. In such a case, any active subscriptions of that client would also have been discarded. In such scenarios, if a broker receives a keep-alive message from a client it does not consider as registered, it replies with a negative acknowledgement. The correct client behaviour in this scenario is to re-register with the broker and re-issue any subscriptions that were previously active. Another case is that of transient failures of clients, e.g. a client entering the system and exiting. In such a scenario, the client may crash before completing its registration process, may register but issue no subscriptions and exit, may issue subscriptions but exit without unregistering, etc. In all such scenarios, the registration management protocol in our system will bring the overall system in to a correct state (with respect to client registrations and subscriptions) within $2\,\Delta x_k + 2\,\Delta X_R$ time period, i.e. the maximum time it takes for a client to be considered unregistered at a broker and the maximum of two client registration table exchange durations amongst brokers. Fig. 6 illustrates some additional client–broker registration scenarios.

5.1. Broker discovery and registration

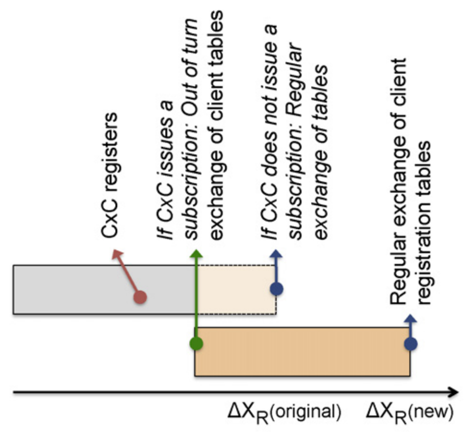
In the Context Provisioning Architecture, CxCs and CxPs need to discover their local broker in order to participate in context consumption and production. Our system provides multiple broker discovery mechanisms to accommodate clients of varying capabilities. These broker discovery mechanisms for clients, illustrated in Fig. 7, include manual discovery through an administrator provided


Initial client registration
Expected keep-alive or registration invalidated
Registration discarded
Regular exchange of client registration tables
X_r
ΔX_k
$2.\Delta X_k$
ΔX_R

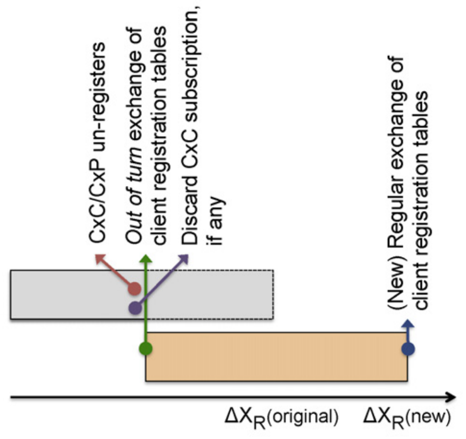
(a) Illustration of the client registration/keep-alive advertisements with respect to the exchange of client registration tables between the brokers.


CxP registers
Out of turn exchange of client registration tables
(Original) Regular exchange of client registration tables
(New) Regular exchange of client registration tables
ΔX_R(original)
ΔX_R(new)

(b) Illustration of the scenario that depicts the effect of a CxP registration on the client registration exchange intervals of a broker in the federation.

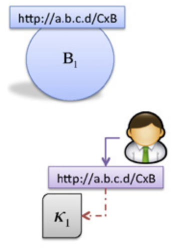

CxC registers
If CxC issues a subscription: Out of turn exchange of client tables
If CxC does not issue a subscription: Regular exchange of tables
Regular exchange of client registration tables
ΔX_R(original)
ΔX_R(new)

(c) Illustration of the scenario that depicts the effect of a CxC registration. Notice that the broker's original table exchange interval is only affected if the CxC issues a subscription.

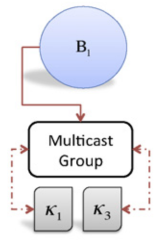

CxC/CxP un-registers
Out of turn exchange of client registration tables
Discard CxC subscription, if any
(New) Regular exchange of client registration tables
ΔX_R(original)
ΔX_R(new)

(d) Illustration of the scenario that depicts the effect of a CxP/CxC un-registration from a broker in the federation.

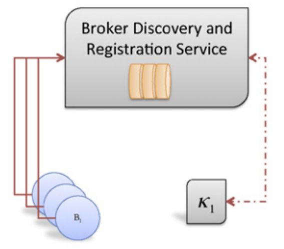
Fig. 6. Illustration of various client-broker scenarios with respect to the maintenance of the correct state of the broker federation. (a) Illustration of the client registration/keep-alive advertisements with respect to the exchange of client registration tables between the brokers. (b) Illustration of the scenario that depicts the effect of a CxP registration on the client registration exchange intervals of a broker in the federation. (c) Illustration of the scenario that depicts the effect of a CxC registration. Notice that the broker's original table exchange interval is only affected if the CxC issues a subscription. (d) Illustration of the scenario that depicts the effect of a CxP/CxC un-registration from a broker in the federation.



(a) Manual discovery of the broker by the client. The user or administrator provides the URL to the Context Broker



(b) Multicast based discovery of the broker by the clients



(c) Client discovers the broker by querying the Broker Discovery and Registration Service with which the brokers have registered their presence

Fig. 7. Various broker discovery mechanisms available to the clients in the Context Provisioning Architecture. (a) Manual discovery of the broker by the client. The user or administrator provides the URL to the context broker. (b) Multicast based discovery of the broker by the clients. (c) Client discovers the broker by querying the broker discovery and registration service with which the brokers have registered their presence.

URL, via multicast group communication and by utilising the broker discovery and registration service (BDRS). The context brokers are required to register with the BDRS upon startup. This service serves a dual purpose of allowing the clients to discover their nearest broker and allowing the brokers to discover their neighbouring brokers. In effect, the BDRS caters for the resource discovery problem in a large-scale system, in which categorisation of resources is a critical factor when distributed components manage subsets of client components in the system (Karaoglanoglou and Karatza, 2011). The BDRS facilitates the distributed brokers to form a federation and this mechanism is discussed in detail in the following section.

When a broker wishes to join the federation of context brokers, it registers with the BDRS by sending a broker advertisement message. Upon successful registration, the BDRS replies with the advertisements of existing registered brokers, which allows the broker to initialise its neighbouring broker set. The broker advertisements received from the BDRS also contain the communication endpoints of the brokers, which allows the brokers to establish communication connections. In order to leave the broker federation, a broker sends an un-registration request (advertisement message with unregister flag) to the BDRS, which removes the broker's registration entry and updates all the registered brokers about the unregistering broker. To safeguard against cases where a broker is unable to unregister and goes offline, registered brokers are required to send keep-alive advertisement messages to BDRS. This process is analogous to the one used in client–broker registration maintenance describe later. After registering with BDRS at time z_r, keep-alive messages are expected within regular intervals Δz_k. If a broker fails to send a keep-alive message after $z_r + \Delta z_k$, its registration is invalidated (but not removed). If a keep-alive or a new registration message from the same broker is received at the BDRS within $2\,\Delta z_k$, its registration is considered valid again otherwise the registration is removed. After removing the registration, the BDRS sends the latest broker registration information to the registered brokers. The duration $2\,\Delta z_k$ is called the registration grace period in our federation model.

A situation may occur such that a broker disconnects without unregistering or it unregisters with BDRS but the update from BDRS has not yet propagated in the broker federation. In such cases, if a network level broker B_j is unreachable for B_i, B_i removes all entries from the subscription and client registration tables that pertain to B_j. Mobile brokers are considered special cases, which are allowed to disappear with greater tolerance in the sense that knowledge of mobile clients and their subscriptions is not immediately discarded. Information related to mobile brokers (subscription and client registration tables) is maintained for the grace period within which a mobile broker may re-register ($2\,\Delta z_k$). When a mobile broker re-appears, it may have a different communication end point and is required to re-join the broker federation by re-registering with the BDRS. BDRS disseminates the updated advertisement of the mobile broker to all the brokers in the system. This update in case of re-registration within the grace period is not required in case of network brokers, which are assumed to have static communication endpoints.

6. Mobile Context Brokers in the federation

The federation model discussed so far describes the mechanism by which an overlay network of brokers facilitates the coordination and communication between a set of distributed clients. One of the key requirements of designing a dynamic, large-scale context-aware system is to incorporate and facilitate the mobility of components that take part in context production and consumption. A client may need to disconnect from its local broker for different reasons such as administrative issues or energy conservation factors in mobile devices. A disconnected client may re-join the same broker or a different one within the overlay network of brokers and new clients may join the system as well. This mobility of components and fluidity of the component set is managed in our coordination model by updating and propagating client registration tables amongst the brokers and updating the subscription tables according to the latest information in the client registration tables.

We further improve upon this coordination model for managing mobile clients by using a Mobile Context Broker, which executes on user mobile devices and provides brokering functions to context consumers and providers executing on the device. Mobile devices are likely to suffer from network connectivity disruptions due to their mobility and power constraints. In such scenarios all clients executing on such a device will disconnect, most likely without unregistering properly. Instead of a broker in the network infrastructure managing registrations of clients executing on mobile devices, the task is delegated to the *Mobile Context Broker* in our system. If and when a mobile device goes offline, whether properly by letting the clients unregister or without warning, it is only the disappearance of one broker (mobile) that has to be detected and propagated rather than a possibly large number of clients. Because the registration entries of individual clients contain their local broker's information, unavailability of that broker can easily be used to infer the same for its clients.

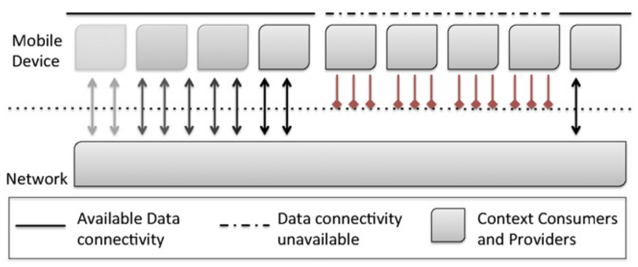
The Mobile Context Broker (MCxB) is a software component designed to execute on a mobile device as a background service that brokers context exchange between consumers and providers, hosted both on the device and the network. Device based context providers and consumers register their presence and requirements during execution to this broker and do not have to lookup each other individually. Moreover, during periods of dis-connected operation, which are still common in mobile devices and networks, these consumers and providers do not have to monitor device connectivity individually; this task is delegated to the Mobile Context Broker. Polling and waiting for events or context information to become available by consumer components is improved by applying the publish–subscribe communication paradigm and using the broker as an event service that manages notifications and subscriptions. These functions provided by the broker save valuable computation cycles and consequently reduce energy consumption. Moreover, the MCxB masks the effects of device mobility by coordinating with the network based context brokers while the device is on the move. By managing the mobility aspects of the device, the MCxB saves the context consumers and providers on the device from having to coordinate context acquisition and delivery as the communication end points of the device change during mobility. Further aspects of the MCxB result in reduction of the overall network bound traffic and execution cost of the CxCs and CxPs on the device. The MCxB participates in the federation of context brokers using the same federation model described earlier.

The mobility of the devices may induce changes in the communication endpoint used by the device for data communication, e.g. switching from one wireless Local Area Network (LAN) network to another, switching from GPRS to WLAN radio and moving across different mobile network carriers. The MCxB makes this mobility-induced change in communication endpoints transparent to the CxCs and CxPs executing on the mobile device. This transparency is achieved because the CxCs and CxPs are only concerned with communicating with the MCxB, which carries out any communication external to the device. The MCxB overcomes this mobility-induced issue by monitoring the changes in communication endpoints and subsequently informing its neighbouring brokers (via the BDRS) about its latest accessibility information. In absence of a broker on the device, each CxP and CxC would have to update a broker on the network individually, resulting in much more resource utilisation (processing and communication) than the case where a MCxB undertakes this task. Fig. 8 illustrates this connectivity transparency provided by the MCxB to its local clients.

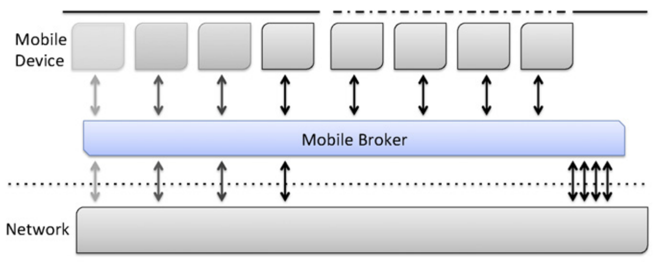
The mobility of devices may also bring about scenarios where data connectivity is unavailable and the context related clients face a disconnected operation scenario. The MCxB provides facilities to the CxCs and CxPs during disconnected operation in the form of storing their subscriptions and notifications for later forwarding, attempting to satisfy the context queries from a locally maintained cache and registering call-backs to inform interested clients when data connectivity is available again.

6.1. Bulk mode operation

The MCxB can operate in bulk query mode for low priority queries in which it forwards queries and responses to the network in bulk. This store and forward bulk mode is useful not only in saving network communication but is also utilised to manage queries and responses during periods of disconnected operation.


Mobile Device
Network
Available Data connectivity
Data connectivity unavailable
Context Consumers and Providers

(a) Communication operations of the device based consumers and providers during periods of data connectivity/disconnection in absence of a mobile broker. Disconnected periods of operation are shown with dotted lines at the top.



(b) Communication operations of the device consumers and providers during periods of data connectivity/ disconnection in presence of a mobile broker. Disconnection is transparent to the clients of the broker.

Fig. 8. Connectivity transparency provided by the Mobile Context Broker to its clients. (a) Communication operations of the device based consumers and providers during periods of data connectivity/disconnection in absence of a mobile broker. Disconnected periods of operation are shown with dotted lines at the top. (b) Communication operations of the device consumers and providers during periods of data connectivity/disconnection in presence of a mobile broker. Disconnection is transparent to the clients of the broker.

While operating in this mode, the MCxB examines the optional priority field in each subscription and if the priority is set to low, the broker adds the query to the bulk queue, which has a (configurable) limited capacity ω. A bulk queue is maintained for a maximum duration γ where γ is one half of the time $\mathcal{T}_x$ remaining in expiration of the subscription with the earliest expiry time.

$$\mathcal{T}_x = \min_{x\in[1,m]}(\mathcal{T}_{\sigma_1}, \mathcal{T}_{\sigma_2}, \ldots, \mathcal{T}_{\sigma_m}) \quad (13)$$

The half limit is chosen so as to leave adequate time for a response to reach the subscribing consumer. The duration γ is re-evaluated on addition of each low priority subscription to the bulk queue as shown in Eq. (14).

$$\mathcal{T}_x = \min_{x\in[1,m+1]}(\mathcal{T}_{\sigma_1}, \mathcal{T}_{\sigma_2}, \ldots, \mathcal{T}_{\sigma_m}, \mathcal{T}_{\sigma_{m+1}}) \quad (14)$$

$$\gamma = \frac{1}{2}\mathcal{T}_x \quad (15)$$

A bulk queue is immediately processed either when γ is reached or the number of subscriptions in the queue reaches the pre-defined bulk limit ω, i.e.

$$if((t_c \geq \gamma) \vee (m == \omega)) \quad (16)$$

where t_c is the current time. Fig. 9 illustrates the operation of the bulk queue with a limited queue capacity ω and incoming subscriptions with varying expiration times $\mathcal{T}_m$.

Another salient feature of the MCxB is that, in addition to the RESTful HTTP based interfaces, it also provides local clients inter-process communication (IPC) interfaces for context subscription, notification, registration and other functional tasks. IPC based communication between local processes on an Android based mobile device (implementation platform for the prototype MCxB) is an order of magnitude faster and less resource intensive than HTTP based communication between the processes (Kiani et al., 2011).

The features of MCxB described in the preceding paragraphs result in reduction in the functional burden of the context consumers and providers executing on the mobile devices. The reduction in functional burden has a direct correlation to the execution cost of these components, specifically the energy consumption in the mobile device while these components participate in context consumption, provision and brokering. The effects of the Mobile Context Broker, and its involvement in the federation of the brokers, on energy consumption in a mobile device are evaluated in the following section.

7. Evaluation and analysis

This section presents an analysis and an empirical evaluation of various facets of the federated broker based Context Provisioning Architecture. Both real-world deployments and simulation studies have been performed for empirically gauging the improvements offered by these features of the Context Provisioning Architecture.

There are various performance indicators that can be evaluated to compare the features of different context provisioning system architectures (Knappmeyer et al., 2011), e.g. load scalability, quality of service over time and quality of context. Within the focus of this article, we present an empirical evaluation of our federated broker architecture in comparison to single broker architecture. Our evaluation is focussed on the comparison of the mean query satisfaction times achieved with these different architectures under similar experimental conditions. Furthermore, we evaluate the effects of utilising the federated broker model on energy consumption in mobile devices in comparison to the scenario where broker federation is not utilised. The mean query satisfaction time T_{S_m} is defined as, for a finite number of context queries n, the mean time between the sending of the subscriptions by a consumer to a broker and the receipt of corresponding notification, i.e.

$$T_{S_m} = \frac{(T_{S_1} + T_{S_n} + \ldots + T_{S_n})}{n} = \frac{\sum_{i=1}^{n} T_{S_i}}{n} \quad (17)$$

where T_{S_m} is the satisfaction time of an individual subscription. One of the factors that effect the query satisfaction times in a broker based context provisioning system is the computational and I/O load on the broker component(s). Other factors include the availability of context, availability of and load on context providers, network conditions, etc. In our evaluation we vary the load on the broker(s) while attempting to keep the other factors constant as much as possible. Moreover, the experiment is set up in a manner such that the load on a broker is a function of incoming context queries, i.e. a higher the rate of arrival of context queries incurs a greater demand on the context coordinating functions of a context broker. With the help of this experimental framework, we intend to empirically evaluate the impact of a federated broker architecture, in comparison to a centralised broker architecture, on mean query satisfaction times under different broker load conditions. Furthermore, we will also analyse the mean time taken to disseminate a component's disappearance (e.g. CxPs or CxBs disconnecting with/without proper un-registration mechanisms) in the broker federation. The aim of this final analysis is to demonstrate the safety and stability features of the federation model.

7.1. Experiment setup

The experiment is set up with a fixed number of context providers (12) and consumers (6) while the number of brokers is one (non-federated) or three (federated) depending on the configuration being evaluated. All software components are initially

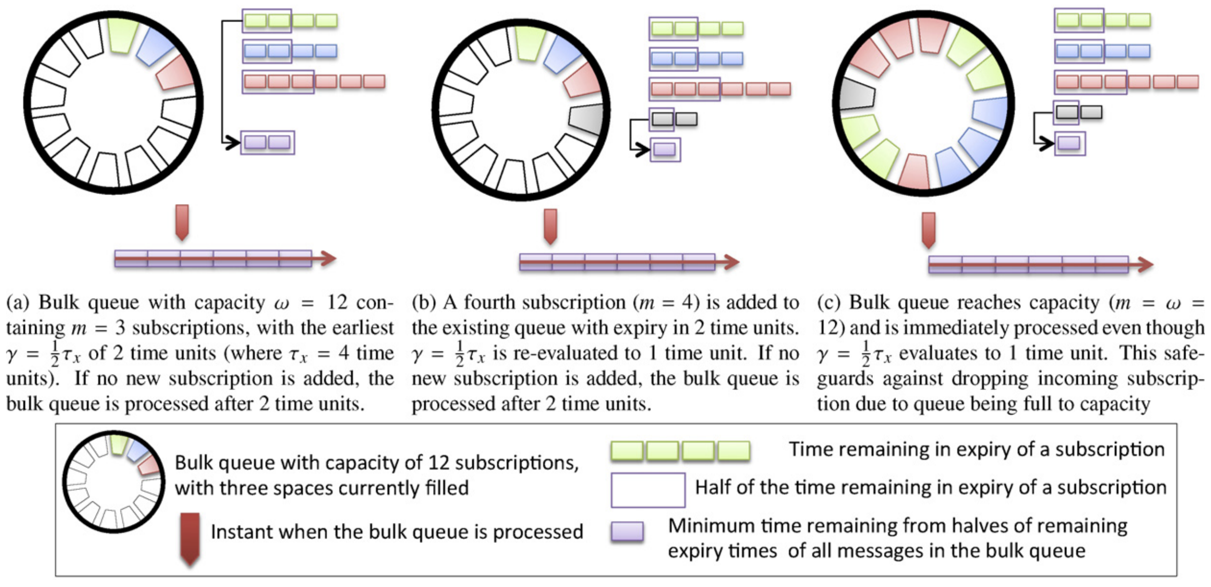

(a) Bulk queue with capacity $\omega = 12$ containing $m = 3$ subscriptions, with the earliest $\gamma = \frac{1}{2}\tau_x$ of 2 time units (where $\tau_x = 4$ time units). If no new subscription is added, the bulk queue is processed after 2 time units.
(b) A fourth subscription ($m = 4$) is added to the existing queue with expiry in 2 time units. $\gamma = \frac{1}{2}\tau_x$ is re-evaluated to 1 time unit. If no new subscription is added, the bulk queue is processed after 2 time units.
(c) Bulk queue reaches capacity ($m = \omega = 12$) and is immediately processed even though $\gamma = \frac{1}{2}\tau_x$ evaluates to 1 time unit. This safeguards against dropping incoming subscription due to queue being full to capacity
Bulk queue with capacity of 12 subscriptions, with three spaces currently filled
Instant when the bulk queue is processed
Time remaining in expiry of a subscription
Half of the time remaining in expiry of a subscription
Minimum time remaining from halves of remaining expiry times of all messages in the bulk queue

Fig. 9. Bulk queue operation in the Mobile Context Broker. (a) Bulk queue with capacity $\omega = 12$ containing $m = 3$ subscriptions, with the earliest $\gamma = (1/2)\tau_x$ of 2 time units (where $\tau_x = 4$ time units). If no new subscription is added, the bulk queue is processed after 2 time units. (b) A fourth subscription ($m = 4$) is added to the existing queue with expiry in 2 time units. $\gamma = (1/2)\tau_x$ is re-evaluated to 1 time unit. If no new subscription is added, the bulk queue is processed after 2 time units. (c) Bulk queue reaches capacity ($m = \omega = 12$) and is immediately processed even though $\gamma = (1/2)\tau_x$ evaluates to 1 time unit. This safeguards against dropping incoming subscription due to queue being full to capacity.

deployed on hosts located within a single LAN. Later, the experiments are repeated with components deployed across distributed networks. All context consumers are deployed on one host and context providers on a second host. Each broker is deployed on a separate host, i.e. three hosts in total are involved in experiment iterations relating to the centralised broker setup, while five hosts are involved when the setup is configured for federated broker evaluation. One of the main differences between the brokers used in the centralised and the federated broker setup is the caching facility, which is only employed in the brokers representing our Context Provisioning Architecture and not in the single, non-federated broker setup. This setup reflects that caching mechanisms have not been implemented in existing context-provisioning systems, either in a single or federated broker setup. Furthermore, in the single broker setup the context consumers and providers on the mobile device interact via the broker installed on the remote desktop host,

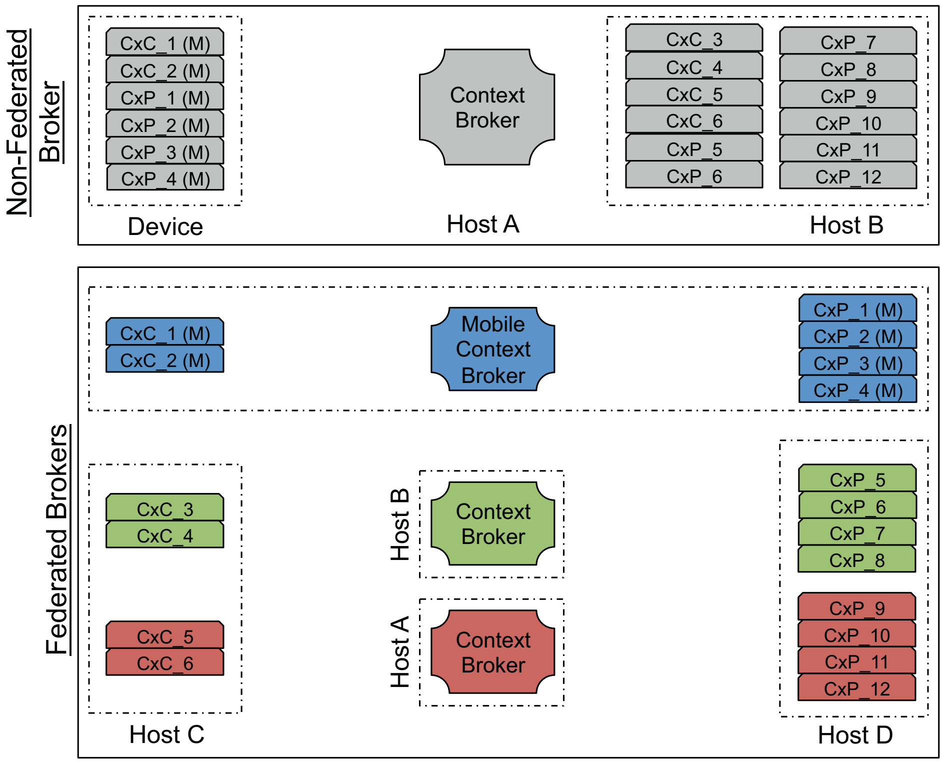


Fig. 10. Deployment setup of the federated and non-federated brokers experiments. The colours represent association of clients with a particular broker in the setup.

Table 1
Configuration of the hardware used in the experiment.

Host	Configuration
Desktop host A,B (Brokers)	Mac Pro, 2.66 GHz Quad-Core Intel Xeon CPU, 8 GB 1066 MHz DDR3 RAM, 802.11n WiFi, Mac OS X Server 10.7
Desktop host C,D (Consumers/Providers)	Mac Pro, 2.66 GHz Quad-Core Intel Xeon CPU, 8 GB 1066 MHz DDR3 RAM, 802.11n WiFi, Mac OS X Server 10.7
Mobile device	Google Nexus One, 1GHz Qualcomm QSD 8250 CPU, 802.11g WiFi, Android version 2.3.4
WLAN Access Point	Netgear WNR 2000 802.11n router

whereas in the federated broker setup, the device based consumers and providers interact with the local Mobile Context Broker. BDRS is available in the network and is deployed on one of the desktop hosts. The experiment deployment is illustrated in Fig. 10 and the hardware and software configuration of the computing hosts is shown in Table 1.

Each context provider in the experiment setup is responsible for provisioning of a single context scope. For purposes of our experiment it is assumed that the requested context is always available, i.e. the results will not be affected by availability of context information. The selection of a scope in subscriptions is uniformly distributed, i.e. there is an equal probability of a particular scope being queried across all context subscriptions during an experiment. Moreover, scopes are assumed to be atomic, i.e. satisfaction of a subscription about scope *x* does not depend on any other scope *y*. The variable parameter in the experiment iterations is the rate at which context consumers send subscriptions to the brokers, i.e. the query rate. The results reported later in this section are obtained by executing 2000 queries in each repetition. We have repeated the experiments for 100, 500, 1000, 1500, 2500 and 3000 queries and established that this selection provides an acceptable sample space as the results for 1000, 1500, 2500 and 3000 subscriptions have similar statistical tendencies. In the case where our sample space is less than 500 queries, the results show a significant variation that limits confident statistical inference from the results.

Arrival times of queries at a broker can be modelled as a Poisson process, which is defined by an exponential distribution. Modelling of such events as a Poisson process, a stochastic process in which events occur continuously and independently of one another, is well established (van Gelder et al., 2002). For calculation of the time instance at which to generate queries from context consumers in our experiment, we use a rate from the set 10, 20, 30, 40, 50, 60 queries per second. These query rates are used as the rate parameter λ to generate the time instance values when a context consumer should send the next query. An experiment iteration uses a constant rate, i.e. the rate of query does not change during an experiment run. These event generation times in the *constant rate* Poisson process of our experiment, for a given time interval $0 \leq t \leq \tau$, can be generated by using Algorithm 1.

Algorithm 1. Generation of event times in a Poisson process with constant rate λ

```
$a_0 \Leftarrow 0.0$
$n \Leftarrow 0$
while $a_n < \tau$ do
    $a_{n+1} \leftarrow Exponential(1/\lambda)$
    $n{+}{+}$
end while
return $a_1, a_2, \ldots, a_{n-1}$
```

7.2. Load scalability analysis

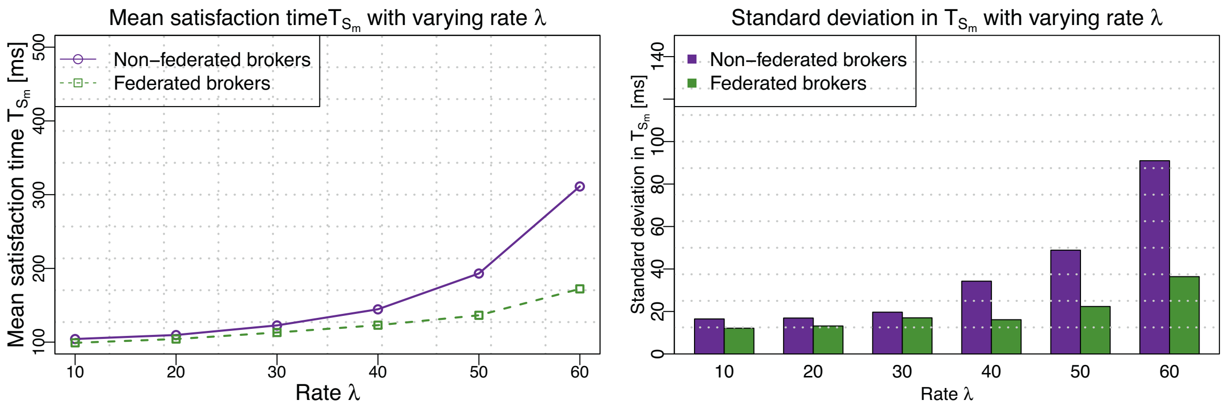
The experiments are carried out using the described setup by issuing context queries from the context consumers and recording the notification arrival times corresponding to each query. One set of experiments is carried out using the non-federated broker setup and the second set using the federated broker setup. Between successive iterations only the query rate λ is varied. On completion of 2000 queries, the mean satisfaction time for an iteration is calculated and individual query satisfaction times are also recorded for later analysis. The brokers' and clients' topology is assumed to be static and known before the start of the experiments, i.e. all clients of the brokers have registered with their local broker and the brokers have exchanged client advertisement related routing tables. Furthermore, the caching facility in the federated brokers is configured in a manner that it provides a maximum cache hit ratio of 15%, i.e. the broker monitors the number of subscriptions it has received and cache hits up to the current point in time and only returns a notification from the cache if the hit ratio is less than 15%. Our caching mechanism operates not only in desktop-based brokers but also in the Mobile Context Brokers. Caching related issues, specifically the effect of variable query rate, non-uniform scope selection, etc. on the cache hit ratio, are evaluated in detail in our earlier work (Kiani et al., 2010a). There remains further scope for research in this particular avenue, e.g. considering the patterns and contents of user queries (Drakatos et al., 2007), storage capabilities of mobile devices while utilising caches for performance optimisation and energy conservation (Mavromoustakis and Karatza, 2007).

The mean query satisfaction times for individual subscriptions with varying query rates λ are plotted in Fig. 11(a). Comparing the mean values of two setups, it is evident that an increase in the query rate results in delayed query satisfaction on average but more importantly it clearly illustrates that the adverse effect is much less pronounced in case of federated brokers setup. There are two predominant reasons for the better performance of the federated brokers setup. Firstly, the context providers are spread across three brokers, so context subscriptions are spread across three brokers, which reduces the overall load on all the brokers. Secondly, the federated brokers have a caching component which further effects the overall query satisfaction time by satisfying a subscription from the cache instead of forwarding it to another broker or provider.

In addition to the variation in mean query satisfaction time across all experiments, another notable observation is the standard deviation in the result sets. As Fig. 11(b) reveals, results of higher query rates show a marked increase in the standard deviation amongst the query satisfaction times. The spread in satisfaction times is more pronounced in case of non-federated broker experiments than federated broker experiments. Our earlier argument in the case of increased mean satisfaction time with increasing query rate also holds valid for this observation of standard deviation trend. Furthermore, the increasing trend in standard deviation of mean satisfaction times with increasing query rates also points out that the system is less likely to guarantee an optimal response time under increasing load. The high variance occurs due to the server not being able to process each query under similar load conditions. However, this deterioration of optimal response time guarantee is less in case of federated brokers than centralised broker setup.

The results and statistical inferences have demonstrated that a federated setup consisting of three brokers performs better than a centralised, single broker setup in terms of query satisfaction times under increasing load conditions. The distribution of subscription and notifications across three brokers naturally aids in distributing the load across the brokers. In our experimental setup the load is equally distributed across three brokers as the scope selection in subscriptions is uniformly distributed and each broker has the same number of providers/consumers as its clients. This uniform distribution may not be ideal in a real world scenario but in our experimental setup it helps us reduce the number of variable parameters and better study the parameters under observation.

Furthermore, these experiments have been carried out in a controlled environment of an isolated local area network and are not



(a) Mean query satisfaction time for rates $\lambda = \{10,20,30,40,50,60\}$ (b) Standard deviation in T_{S_m} for rates $\lambda = \{10,20,30,40,50,60\}$

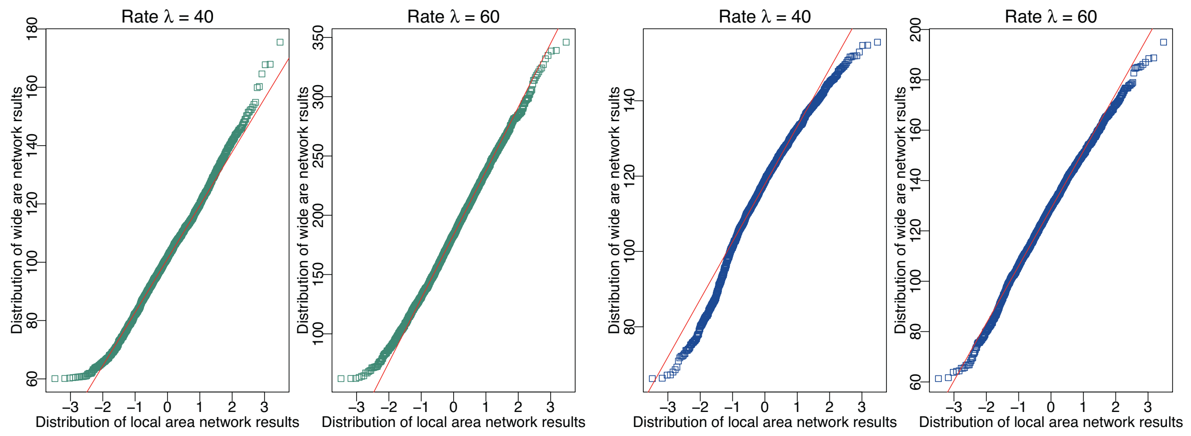
Fig. 11. Variation in mean query satisfaction time and standard deviation across all experiments. (a) Mean query satisfaction time for rates λ = {10, 20, 30, 40, 50, 60}. (b) Standard deviation in T_{S_m} for rates λ = {10, 20, 30, 40, 50, 60}.

affected by varying network conditions that are inherent in wide area networks. In order to ascertain the applicability of our deductions from these experiments in a real-world system, we have repeated the experiments by deploying the brokers and clients across a distributed network in university campus and residential settings across a city and recorded the same set of observations as specified earlier. We have observed from the results of repeating the experiments in a wider area network that both systems perform similarly to their local area network performances. Instead of including detailed results of wider area network experiments in this section, the *Q*–*Q* plots of a subset of these results, in comparison to their relevant local area network result subsets, are shown in Fig. 12. The *Q*–*Q* plots clearly demonstrate that the two distributions being compared are similar as the points in the *Q*–*Q* plot approximately overlap throughout the intervals, diverging only at the extremes.

7.3. Energy conservation through the Mobile Context Broker

In addition to the load distribution across federated brokers, another factor that reduces the degradation in mean query satisfaction times and variance is the caching facility of the federated brokers. Each broker maintains an individual cache and any cache hit results in an optimal satisfaction time for a query, i.e. a response to a subscription from the local broker's cache takes less overall time than response from a remote broker/provider or even a provider registered with the local broker. The cache-hit rate in our experiments is limited to a maximum of 15% in order to limit its impact on the variability of results. Further analysis of caching benefits in our architecture and the effect of scope distribution and query rates on cache-hit ratios is investigated in our earlier work (Kiani et al., 2010a).

In order to ascertain the effect of broker federation and the Mobile Context Broker on energy consumption in mobile devices, we have recorded the energy consumed by the context consumers, providers and the broker (if present) on the mobile device involved in the experiment. The total energy consumed by these components during each experiment iteration is divided by the number of context queries in that iteration to calculate the mean energy consumption per context query. PowerTutor (Zhang et al., 2010) is utilised for recording the energy consumption of executing processes on an Android based mobile device, which is an application for Android based devices that displays the power consumed by major system components such as CPU, network interface, display, etc. and different applications. PowerTutor calculates the phone's breakdown of power usage with an average of 1% error over 10-s



(a) Q-Q plot comparing the distributions of non-federated broker experiment results carried out within a local area network and wide area network (b) Q-Q plot comparing the distributions of federated brokers experiment results carried out within a local area network and wide area network

Fig. 12. *Q*–*Q* plots between the result distributions in local and wide area networks. (a) *Q*–*Q* plot comparing the distributions of non-federated broker experiment results carried out within a local area network and wide area network. (b) *Q*–*Q* plot comparing the distributions of federated brokers experiment results carried out within a local area network and wide area network.

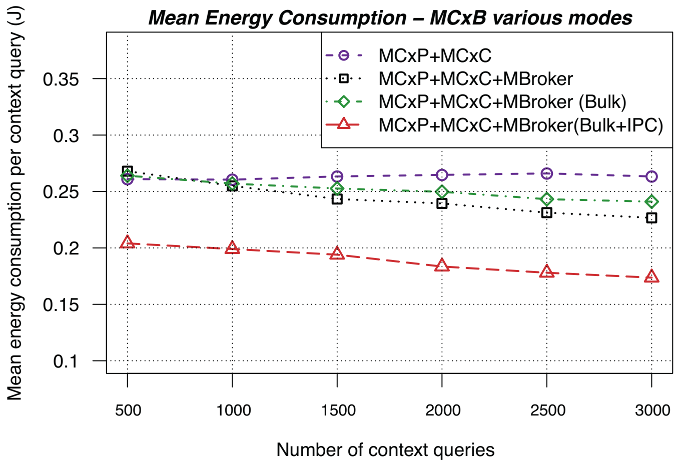


Fig. 13. Mean energy consumption per context query with MCxB in various modes: without MCxB, with MCxB and all communication over RESTful HTTP interfaces, with MCxB operating in bulk mode and all components using RESTful interfaces, and finally with MCxB in bulk mode and components using IPC communication.

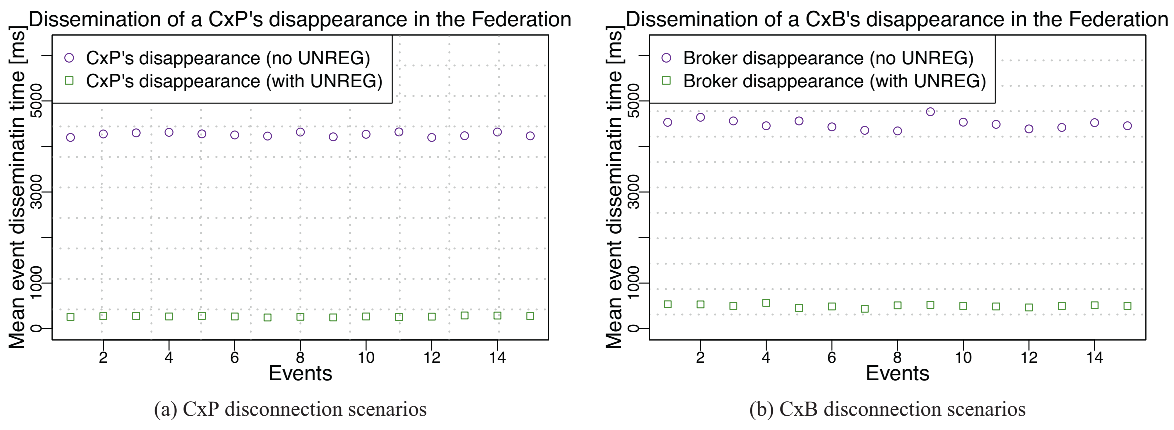
intervals while the worst case error over 10 s is 2.5% (cf. Zhang et al., 2010, p. 8). In these experiments only the energy used by an application in utilising the CPU and WiFi is considered when calculating its energy consumption signature. All results in the following section are mean values of five repetitions of individual experiment iterations (comparison of results from individual iterations show variations within ±3%).

The measurements of the mean energy consumption per query depicted in Fig. 13 demonstrate that reduction in energy consumption when MCxB is used in comparison to the case where MCxB is not used. This reduction in mean energy consumption occurs because the main functional burden of subscription and notification forwarding is delegated to the MCxB and the device based context consumers (MCxC) and providers (MCxP) are only issuing subscriptions and receiving notifications over the RESTful HTTP interfaces. Furthermore, MCxB maintains a context cache, which reduces network bound communication and thus conserves the device energy. When the MCxB is operated in the bulk mode, there is a minor increase in the mean energy consumption per context query. This is explained by the increased processing at the MCxB and the longer duration of execution of the experiment due to the slower query forwarding by the MCxB. However, when the broker is operated in the bulk mode and all device components communicate using IPC rather than RESTfull HTTP calls, the mean energy consumption reduces significantly, i.e. there is an approximately 30% reduction in the case of the experiment iterations with no MCxB and with MCxB operating in bulk mode and using IPC communication. This conservation of energy is an important factor for the efficiency of system operation and its real world usability, keeping in consideration the ever increasing demand for high resource availability and demand for progressively decreasing energy consumption in wireless devices/infrastructures (Mavromoustakis and Karatza, 2010).

7.4. Component disconnection management

The time taken by the disappearance of a component (broker or client) is of particular interest from a performance point of view. To establish the time taken by such an event to be disseminated within the federation, we have used the experiment setup described earlier and the context brokers and providers are scripted to disconnect periodically (i.e. operate in offline mode). They either go offline by unregistering correctly from their respective registration component (BDRS in case of the brokers and the local broker in case of the CxPs) or incorrectly (according to our federation model) without sending the un-registration message. All the hosts in the experiment are synchronised from a NTP server in order to establish a common time reference for measured values. The interval for both brokers' keep-alive message to the BDRS and that of the CxPs' to their local brokers is set to 2 s (for experimental purposes; 20 s interval is used in original experiment). The disappearance time is recorded by the components itself, while all other components, which receive the update containing a removed registration (from BDRS or a neighbouring broker), record the update's arrival time along with the ID of the component whose registration has been removed. These times are consolidated at the end of the experiment to establish the time taken for an event's information to disseminate within the federated broker (mean values are calculated across all times recorded by components). To simplify the setup, only one component goes offline at a time (re-joining the federation after 10 s) and subscriptions/notifications are not issued in this setup.

The calculated values are shown in Fig. 14. If a component disconnects correctly by sending an un-registration message to its registration component, the disconnection event is quickly propagated to the broker federation. This is evident both in the case of CxPs and the brokers in Fig. 14(a) and (b), respectively. However, if the disconnecting component goes offline without un-registering from its registration component, the registration component cannot detect the disconnection event immediately, and has to wait for the registration grace period (2 × 2 = 4 s) to expire before removing the disconnected component's registration entry and informing the neighbouring brokers. These results demonstrate the importance



(a) CxP disconnection scenarios (b) CxB disconnection scenarios

Fig. 14. Mean time taken to disseminate a component's disappearance in the broker federation. (a) CxP disconnection scenarios. (b) CxB disconnection scenarios.

of the correct un-registration behaviour to be employed by brokers and clients in the federation. More importantly, these results also demonstrate the safety and stability features of our federation model, i.e. even if a component does not disconnect by first un-registering, the system recognises the incorrect behaviour, which may or may not have been forced upon a component, and takes measures in order to keep the overall state of the system valid.

8. Conclusions and future work

This work investigates the issues involved in designing a large-scale context provisioning system and proposes a solution based on a federation of context brokers that use a publish/subscribe oriented context coordination and communication mechanism. Federation pattern, where two or more of a kind of service/system interoperate in a scalable manner, is well established in commercial event-based messaging systems where it is used to achieve scalability in general and redundancy in particular. However, federation of brokers or servers in context provisioning systems has not been demonstrated or analysed. Our federated broker based architecture is the first theoretical and practical demonstration of federated context brokers for large-scale context provisioning. This article establishes the theoretical foundation of an inter-broker routing framework for federating multiple context brokers together. The decoupling between context-providing and consuming components of the system is achieved through publish/subscribe communication semantics, which provides decoupling in all aspects of time, space and synchronisation and aid in better addressing the scalability demands in comparison to a centralised broker approach.

This work also presents the novel concept of a context-brokering component to manage and facilitate the modern role of mobile devices, i.e. consuming and providing context. We envision a continued evolution of the role of smart mobile devices in human–computer and human–smart space interactions that will be supported by technological advancements in the surrounding smart space ecosystem (sensors, services and communication infrastructure). The Mobile Context Broker is well suited to the evolving role of modern mobile devices in terms of participation in large-scale context provisioning. Its integration with the federation model also accommodates mobile clients that move between brokers in the federation. Moreover, it facilitates the reduction in functional burden on device based context consumers and providers, especially during periods of network disconnection and also facilitates energy conservation on devices that host context consumers and providers.

The federated broker model of the Context Provisioning Architecture has useful implications not only in coordination and communication of contextual information but also in the administration of the context provisioning process across organisation and administrative boundaries. In future ubiquitous environments with pervasive interconnected artefacts, the contextual landscape will be divided into multiple administrative domains due to ownership, management, access and privacy reasons. Existing broker based architectures that are unable to coordinate inter-domain context exchange with other instances of broker-managed context provisioning systems are ill-suited to be utilised in such environments. The Context Provisioning Architecture is well placed to serve as a building block for future context provisioning systems. Furthermore, the Mobile Context Broker can serve as a nomadic gateway for artefacts in smart environments that have limited communication range. Such artefacts can wait for a smart mobile device with a mobile broker to roam within its communication range and submit stored contextual data for dissemination via the Mobile Context Broker. Thus this novel component can be utilised in emerging modes of data dissemination, e.g. social dissemination in opportunistic networks (Ciobanu et al., 2011), specifically when context is defined from the viewpoint of users' mobile devices (Dobre, 2011).

An investigation into a possible extension of the inter-broker routing is planned, which takes the similarities between context subscriptions into account and adopts covering or merging-based routing. Covering and merging based routing have been proposed for general event based systems, but these do not consider the temporal bounds associated with the validity of events under consideration. Moreover, we are investigating a Cloud-based deployment of the federated broker architecture. We envision a not-so-distant application of this architecture over a Cloud infrastructure by organisations, such as telecom providers, for large-scale provisioning of contextual information. These Clouds may be public or private, and we propose that such Cloud instances can be federated together to coordinate cross-organisational boundary contextual information. Development of such techniques for integrating distributed and heterogeneous resources is useful in a number of social scenarios, e.g. disaster management, vehicular networks and smart cities (Bessis et al., 2011), where user and environment context is central to the focus of system. The Cloud platform can provide the requisite scalability, management, cost and performance benefits, and at the same time leverage the benefits of context federation promoted by the work presented article. The federation of public/private clouds is as yet a developing concept, but one that has a promising potential for application and success in the domain of context provisioning.

References

Bessis, N., Asimakopoulou, E., Xhafa, F., 2011. A next generation emerging technologies roadmap for enabling collective computational intelligence in disaster management. International Journal of Space-Based and Situated Computing 1 (1), 76–85.

Bessis, N., 2009. Model architecture for a user tailored data push service in data grids. In: IGI Global, http://dx.doi.org/10.4018/978-1-60566-364-7.ch012.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., 1995. Pattern-Oriented Software Architecture: A System of Patterns, Vol. 1. John Wiley & Sons, Ltd., West Sussex, England.

Carzaniga, A., 1998. Architectures for an event notification service scalable to wide-area networks. Ph.D. Thesis, Politecnico di Milano, Milano, Italy.

Chen, H., December 2004. An intelligent broker architecture for pervasive context-aware systems. Ph.D. Thesis, University of Maryland, Baltimore County.

Ciobanu, R.-I., Dobre, C., Cristea, V., 2011. A data dissemination algorithm for opportunistic networks. In: in: Proceedings of the 13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), September 26–29, 2011, pp. 299–305, http://dx.doi.org/10.1109/SYNASC.2011.56.

Dobre, C., 2011. CAPIM: a platform for context-aware computing. In: in: Proceedings of the 2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), Barcelona, Spain, October 26–28, 2011, pp. 266–272, http://dx.doi.org/10.1109/3PGCIC.2011.48.

Drakatos, S., Pissinou, N., Makki, K., Douligeris, C., 2007. A context-aware cache structure for mobile computing environments. Journal of Systems and Software 80 (7), 1102–1119, http://dx.doi.org/10.1016/j.jss.2006.10.027, Dynamic Resource Management in Distributed Real-Time Systems.

Hallsteinsen, S., Geihs, K., Paspallis, N., Eliassen, F., Horn, G., Lorenzo, J., Mamelli, A., Papadopoulos, G., 2012. A development framework and methodology for self-adapting applications in ubiquitous computing environments. Journal of Systems and Software 85 (12), 2840–2859, http://dx.doi.org/10.1016/j.jss.2012.07.052, Self-Adaptive Systems.

Huang, Y., Garcia-Molina, H., 2004. Publish/subscribe in mobile environments. Wireless Networks 10, 643–652 http://dx.doi.org/10.1023/B:WINE.0000044025.64654.65

Karaoglanoglou, K., Karatza, H., 2011. Directing requests in a large-scale grid system based on resource categorization. In: in: 2011 International Symposium on Performance Evaluation of Computer & Telecommunication Systems (SPECTS), IEEE, pp. 9–15.

Kernchen, R., Bonnefoy, D., Battestini, A., Mrohs, B., Wagner, M., Klemettinen, M., 2006. Context-awareness in MobiLife. In: Proceeings of the 15th IST Mobile Summit, IST Mobile Summit, Mykonos, Greece.

Kiani, S.L., Knappmeyer, M., Reetz, E., Baker, N., 2010a. Effect of caching in a broker based context provisioning system. In: Proceedings of the 5th European Conference on Smart Sensing and Context, Vol. 6446, LNCS, pp. 108–121.

Kiani, S.L., Knappmeyer, M., Baker, N., Moltchanov, B., 2010b. A federated broker architecture for large scale context dissemination. In: in: Proceedings of the 2nd Int'l Symp. on Advanced Topics on Scalable Computing, Bradford, UK.

Kiani, S.L., Moltchanov, B., Knappmeyer, M., Baker, N., 2011. Analysis of the energy conservation aspects of a Mobile Context Broker. In: Proceedings of the 11th International Conference and 4th International Conference on Smart Spaces and Next Generation Wired/Wireless Networking, NEW2AN'11/ruSMART'11, Springer-Verlag, Berlin, Heidelberg, pp. 26–37 http://www.dl.acm.org/citation.cfm?id=2033707.2

Knappmeyer, M., Kiani, S.L., Tönjes, R., Baker, N., 2010a. Modular context processing and provisioning: prototype experiences. In: Proceedings of the 4th ACM International Workshop on Context-Awareness for Self-Managing Systems, CASEMANS'10, ACM, New York, NY, USA, http://dx.doi.org/10.1145/1858367.1858375, 8:53–8:58.

Knappmeyer, M., Kiani, S.L., Frá, C., Moltchanov, B., Baker, N., 2010b. A light-weight context representation and context management schema. In: Proceedings of IEEE International Symposium on Wireless Pervasive Computing.

Knappmeyer, M., Kiani, S.L., Baker, N., Ikram, A., Tönjes, R., 2011. Survey on evaluation of context provisioning middleware. In: Proceedings of the Second Workshop on Context-Systems Design, Evaluation and Optimisation in conjunction with the 24th International Conference on Architecture of Computing Systems (ARCS), IEEE.

Korpipää, P., Kela, J., Malm, E.J., 2003. Managing context information in mobile devices. IEEE Pervasive Computing 2 (3), 42–51.

Mühl, G., 2002. Large-scale content-based publish/subscribe systems. Ph.D. Thesis, Darmstadt University of Technology, Germany.

Mühl, G., Fiege, L., Pietzuch, P., 2006. Distributed Event-Based Systems. Springer-Verlag, Berlin.

Mavromoustakis, C., Karatza, H., 2007. An optimal adaptive approach using behavioral promiscuous caching and storage-capacity characteristics for energy conservation in asymmetrical wireless devices. In: in: IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC 2007, IEEE, pp. 1–8.

Mavromoustakis, C.X., Karatza, H.D., 2010. Real-time performance evaluation of asynchronous time division traffic-aware and delay-tolerant scheme in ad hoc sensor networks. International Journal of Communication Systems 23 (2), 167–186, http://dx.doi.org/10.1002/dac.1054.

MobiLife Project, 2006. MobiLife CRF: Context representation framework. Tech. rep., http://www.lab.telin.nl/koolwaaij/showcase/crf/crf.html (last checked 01.05.12).

Moltchanov, B., Zafar, M., Baker, N., 2010. Distributed context management: Architecture and commercial trials. In: in: Proceedings of ICT Mobile Summit 2010, Florence, Italy.

Podnar, I., Lovrek, I., 2004. Supporting mobility with persistent notifications in publish/subscribe systems. In: in: Proceedings of the 3rd International Workshop on Distributed Event-Based Systems, Citeseer, p. 80.

Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H., Nahrstedt, K., 2002. A middleware infrastructure for active spaces. IEEE Pervasive Computing 1, 74–83, http://dx.doi.org/10.1109/MPRV.2002.1158281.

van Gelder, P., Beijer, G., Berger, M., 2002. Data Mining III, Vol. 6 of Management Information Systems. WIT Press Publishing, Statistical Analysis of Pageviews on Web Sites, p. 1032.

Weiser, M., 1991. The computer for the twenty-first century. Scientific American 265 (3), 94–104.

Zafar, M., Baker, N., Moltchanov, B., Jo ao Miguel Goncalves, S.L., Knappmeyer, M., 2009. Context management architecture for future internet services. In: in: ICT Mobile Summit 2009, Santander, Spain.

Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R.P., Mao, Z.M., Yang, L., 2010. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In: in: Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/software Codesign and System Synthesis, CODES/ISSS '10, ACM, New York, NY, USA, pp. 105–114 http://doi.acm.org/10.1145/1878961.1878982

Dr. Saad Liaquat Kiani is a senior lecturer in Networks and Mobile Computing at the University of the West of England (UWE), Bristol, UK. He is also a researcher at the Centre for Complex Cooperative Systems at UWE and has been involved in several EU projects, knowledge transfer partnerships and multi-disciplinary research and development projects. He received his BE degree from National University of Sciences and Technology, Pakistan, in 2003 and a masters degree in computer engineering from Kyung Hee University, South Korea, in 2005. His research interests focus around the technological issues and social impact of mobile computing, context-aware computing and distributed systems in general.

Dr. Ashiq Anjum is a senior lecturer in the School of Computing and Mathematics at the University of Derby, UK. Prior to this he was working at the department of computing, Imperial College London as a research associate. He has been working on various collaborative projects with CERN, Geneva, Switzerland for the last 10 years. His areas of research include concurrent, distributed and parallel systems. He is the principal investigator of the Stream Cloud project and in past has worked on a number of research projects that have been funded by various European, American and Asian funding agencies. He has more than 50 peer reviewed publications to his credit. Before starting an academic career, he worked for various multinational software companies for around 7 years.

Dr. Michael Knappmeyer is currently with Ratiodata IT-Lösungen & Services GmbH, an IT service provider, where he holds a position as mobile solution architect. He received his PhD (computer science) from the University of the West of England, Bristol, UK, in 2012. His thesis presented a Context Provisioning Middleware with Support for Evolving Awareness. In 2006 he received the Diplom-Informatiker (FH) degree from the University of Applied Sciences Osnabrück, Germany. As a research associate at Osnabrück, he participated in the European research projects "C-MOBILE" and "Context Casting (C-CAST)". In the latter he led the context reasoning related activities. His research interests include smart spaces, context modelling, reasoning and mobile device management.

Prof. Nik Bessis is currently the head of Distributed and Intelligent Systems (DISYS) research group, a professor and a chair of computer science in the School of Computing and Mathematics at University of Derby, UK. He is also an academic member in the Department of Computer Science and Technology at University of Bedfordshire (UK). His research interest is the analysis, research, and delivery of user-led developments with regard to resource discovery and scheduling, trust, data integration, annotation, and data push methods and services in dynamic distributed environments. These have a particular focus on the study and use of next generation technologies including grid and cloud computing for the benefit of various virtual organisational settings. He is involved in and leading a number of funded research and commercial projects in these areas. He has published over 170 papers, won 3 best paper awards and is the editor of several books and the Editor-in-Chief of the *International Journal of Distributed Systems and Technologies (IJDST)*. In addition, he is a regular reviewer and has served several times as a keynote speaker, conferences/workshops/track chair, associate editor, session chair and scientific program committee member.

Prof. Nikolaos Antonopoulos is currently the head of School of Computing and assistant dean (research) of the Faculty of Business, Computing & Law at the University of Derby. He has been carrying out research in computer science for over 15 years. The main focus of his research has been software mobile agents and Peer-to-Peer (P2P) networks. His work involves the design and optimisation of P2P architectures in terms of data traffic and latency in the context of resource and service discovery. He has been using the software agent abstraction in order to provide the architectures above with autonomy, adaptivity and efficiency through code mobility. His research interests include emerging technologies such as large scale distributed systems and peer-to-peer networks, software agent architectures and security.