2010 10th IEEE International Conference on Computer and Information Technology (CIT 2010)

A Federated Broker Architecture for Large Scale Context Dissemination

Saad Liaquat Kiani*, Michael Knappmeyer†*, Nigel Baker*, Boris Moltchanov‡
*Bristol Institute of Technology, University of the West of England, Bristol BS16 1QY, UK
{saad2.liaquat, nigel.baker}@uwe.ac.uk
†University of Applied Sciences Osnabrück, P.O. Box 1940, 49009 Osnabrück, Germany
m.knappmeyer@fh-osnabrueck.de
‡Telecom Italia Lab, via G.Reiss Romoli, 274, 10148-Torino, Italy
boris.moltchanov@telecomitalia.it

***Abstract*—Dissemination of context data from context producing entities to context consuming entities is a fundamental functional task of context aware systems. Broker based approaches have been successfully demonstrated in a number of prototype context-aware systems. With the increase in sensing capabilities of mobile devices, such devices are not merely consumers of context information any more but also have the ability to be *providers* of context gathered through integrated sensors. In such a provider-consumer model, where context dissemination is aided by a central broker, device-broker interaction can become a communication and computation bottleneck in presence of multiple context providers and consumers in mobile devices. In this paper we present a theoretical model for a large scale context-aware system based on a federation of multiple context brokers and discuss the concept of a mobile broker to facilitate the participation of mobile devices in context provision and consumption efficiently. The utilisation of an asynchronous event-based publish-subscribe paradigm is focused as a key element.**

I. Introduction

A context-aware communication system usually comprises of several context management functionalities. Most important are acquisition and provision of contextual information related to an entity (e.g. user, device, environment, network etc). Within this simple definition, we can divide the system components involved in the context management into either context consumers or context providers or a combination thereof. Given this basic role model, a small scale system could work with direct communication between context providers and consumers but for large scale systems, network boundaries, mobility and other factors give rise to the necessity of having other communication mechanisms, e.g. broker assisted communication between the consumers and producers. In a broker based model, providers of context register their capabilities with the broker and consumers can query a central broker about the type of context provider they are looking for. For example, a location context provider may provide realtime location of users based on radio signal strength and received cell identifiers. A context consuming application on a user device can use this location context to recommend restaurants to the user around lunch time. This context consuming application can also use other context providers to aggregate user context and deliver sophisticated context based services, e.g. weather and user calendar context providers can be used to make more appropriate restaurant recommendations. Pervasiveness of mobile devices makes it evident that users will mostly interact with the context aware system through these devices. Moreover, modern mobile devices are increasingly being fitted with various sensors, e.g. accelerometers, GPS receivers, light sensors, magnetometers etc, and can act as providers of context information in addition to their traditional role of hosting context consuming applications. This increased capability of mobile devices allows for the realisation of complex usage scenarios as more user context can be made available by utilizing the on-device sensors. For example, consider a network service that builds multicast groups based on user location for efficient delivery of multimedia content. Such a service can query user interests and preferences from a *user profile* provider and location from GPS based location providers being executed on user devices. The type of content may further be customized based on the users' current activity, which can be acquired from an *activity context provider* being executed partly on the user device and partly on a performant server deployed in the system infrastructure. The activity context provider can use the on-board accelerometers, noise and light sensors, and running applications on the device to estimate the current activity context of the user [9][10]. The provider-consumer interaction is depicted in Fig. 1. Building on this complex scenario, we

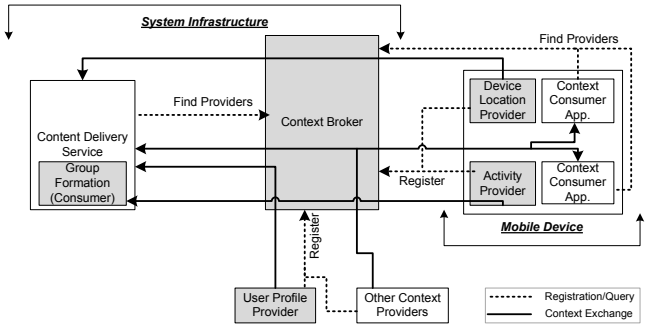

Figure 1. Complex broker based consumer-provider interaction.

can deduce that a number of context providers and consumers

978-0-7695-4108-2/10 $26.00 © 2010 IEEE
DOI 10.1109/CIT.2010.495

on the user device will be interacting with a context broker on the network for acquisition and provision of context data within the context aware system and beyond (e.g. third party services). If each device level context provider and consumer were to handle broker bound communication themselves, it would increase not only the computation cost but also the development cost for new providers and consumers. Given the inherent mobility and somewhat intermittent connectivity of mobile devices, each provider and consumer may have to carry out extra life-cycle management tasks as well. To mitigate the effects of these issues, we propose a mobile broker that can work in federation with those context brokers being deployed in the system infrastructure. The mobile context broker will be executed on the user mobile device and facilitate the device level context providers and consumers in retrieving and providing context to infrastructure components through a well defined publish-subscribe mechanism. Before we discuss this communication paradigm(Section III) which is used to design the broker federation model (Sections IV, V, VI, VII), it is important to describe our broker architecture (Section II).

II. Broker Architecture for a Large Scale Context Provisioning System

In order to design a large context-aware system that is able to support emerging services, a large variety of context information needs to be created, maintained and distributed. Although in context-aware systems, domain knowledge is very much tied to the application, building context aware applications from scratch is not practical. Therefore one of the aims of our architecture is to decouple context from applications to enable reuse. A well-known paradigm of combined context-aware service and networked service model is the producer(provider)-consumer model. Network components may take the roles of a Context Provider or may take the role of information sinks, i.e. Context Consumers [2][3]. These basic entities may interconnect by means of Peer-to-Peer (P2P) techniques [4] or by a context *broker* providing a directory and lookup service. Broker architecture in its various forms exists as a middleware technology that manages communication and data exchange between objects or entities. Chen et. al.[5][6] present a Context Broker Architecture (CoBrA), which is an agent based architecture for supporting context-aware systems in smart spaces. Formal representation of context information is an integral part of any context management architecture. Our architecture uses an XML based representation, entitled ContextML [7] to represent context information. An overview of the three main components of our architecture [1] is given in the following paragraphs. A more detailed description of all components of the architecture is provided in [11].

A. Context Consumer

A Context Consumer (CxC) is an entity (e.g. a context based application) that uses context data. A CxC can retrieve context information by sending a subscription request to the Context Broker (CxB) and context information is delivered asynchronously once it is available or when it changes. A synchronous method of getting context information also exists where a CxC requests the Context Broker (CxB) for a particular CxP and queries the CxP directly.

B. Context Provider

A Context Provider (CxP) is a component whose task is to provide context information. A CxP gathers data from a collection of sensors, network, services or other relevant sources. The CxP uses various filtering, aggregation and reasoning mechanisms to infer context from raw sensor, network or other source data. A CxP provides context data only further to a specific invocation or subscription and is specialized on a particular context domain (e.g. location).

C. Context Broker

Context Broker (CxB) is the main component of the architecture. It works as a handler and aggregator of context data and as an interface between architecture components. Primarily the CxB has to control context flow among all attached components which it achieves by allowing CxCs to subscribe to context information and CxPs to deliver notifications. For facilitating synchronous (on-demand) CxC context queries, CxB also provides a Context Provider Lookup Service by maintaining entries of context providers registered with the broker, their communication endpoints, and their capabilities. A number of useful applications have been developed based

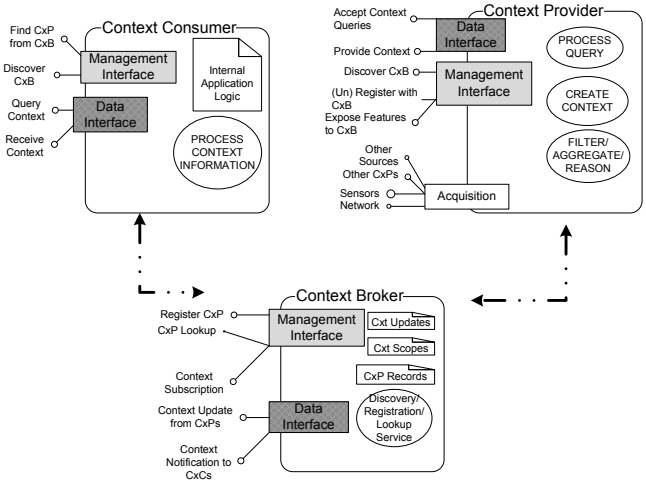

Figure 2. Context management components interaction.

on this simple architecture in synchronous mode of operation and employing a single context broker. A depiction of the core system components described above is presented in Fig. 2 and further details on this architecture and industrial trials are described in [2][1]. Some issues in this basic architecture have been identified in the areas of communication and scalability. Primarily, having a central broker limits the scalability and robustness of the overall system and limits the reach of

[1]This work is supported in part by the European ICT project "C-CAST" - Context Casting - (Contract-No. ICT-216462).

the context aware system. Moreover, synchronous communication between the main system components gives rise to blocking calls, delays and, with the involvement of mobile devices, reliability issues. These issues are being tackled by employing a publish-subscribe communication mechanism for asynchronous communication and developing a model for federation of context brokers to meet the scalability challenge.

III. Publish Subscribe Communication Model

There are a number of well-established distributed communication protocols e.g. Message passing, Remote Procedure Call (RPC), Notifications , Distributed shared memory paradigm [12], Message queuing [13], etc. Publish-subscribe is an asynchronous messaging paradigm where publishers of messages (or events) do not send their messages to specific receivers (subscribers) directly. Rather, published messages are characterized (e.g. into classes, types, channels) without knowledge of what (if any) subscribers there may be. Subscribers express interest in one or more classes or types, and only receive messages that are of interest, without knowledge of what (if any) publishers there are. The messages or events are posted and subscribed to a third entity usually referred to as the broker or event service. This decoupling of publishers and subscribers can allow for greater scalability and a more dynamic network topology. Communication between distributed entities, especially those involving mobile entities, puts greater demands on decoupling between the producers and consumers of information (events). Decoupling in the following three categories is crucial for a distributed communication mechanism.

1) *Space*: The interacting components are not required to know each other in advance. The CxP (publisher) publishes information through an event/information service and the CxC (subscriber) receives information indirectly through that service.
2) *Time*: The interacting components do not need to be actively participating in the interaction at the same time, i.e. the CxP might publish some information while the CxC is disconnected and the CxC might get notified about the availability of some information while the original CxP is disconnected.
3) *Synchronization*: CxPs are not blocked while producing information, and CxC can be asynchronously notified of the availability of information while performing some concurrent activity, i.e. the provision and consumption of information does not happen in the main flow of control of the interacting parties.

This decoupling is important to cater for as it increases scalability by removing explicit dependencies between the interacting participants. Removing these dependencies strongly reduces coordination requirements between different components and results in a well adapted distributed communication infrastructure. There are a number of reasons that make publish-subscribe a good choice for our system:

1) *Decoupling* CxPs produce context independently of any requirements of CxCs. They do not stop producing context if no CxCs are present or interested. CxCs do not inform CxPs when they need context (they subscribe with the CxBs). This mutual independence hints at strong decoupling and is only truly provided by the publish-subscribe paradigm.
2) *Subscription Management* In simple notification systems, subscriptions are managed by publishers, i.e. consumers register their interest in a particular sort of information (topics, channel, content) with publishers and once a produced set of information matches a subscription, the publisher sends the information to consumers who had subscribed for that. In our system, this mechanism places extra burden on CxPs to manage individual subscriptions and keep track of CxCs that will become quite difficult in presence of mobile CxCs and mobile CxPs. Moreover, it will also increase development effort for each CxP. In contrast, publish-subscribe does not dictate the providers to manage subscriptions themselves; rather a central event service, i.e. the broker, manages the subscriptions. Consumers subscribe with the event service and the task of filtering events from providers and delivering to consumers is carried out by the broker.
3) *Context Consumers* Context consumers differ from simple event consumers by usually requiring information from a number of providers to formulate higher-level context. In absence of an event service, if a certain context consumer has to subscribe to multiple context providers, it would have to first of all discover and then subscribe with a number of providers individually. This is much more complex than a scenario where a context consumer only needs to register for its entire context related subscriptions with one event service. Publish-subscribe enables context consumers to take the simpler route.
4) *Mobility* Following the discussion in the previous paragraphs, subscription management, both at context providers and consumers, becomes much more complex in the presence of mobile entities and absence of an event service. In that case, mobile terminals on which context producers and consumers are executing will have to take extra measures to manage subscriptions and notification between consumers and producers over the network. If a publish-subscribe enabling broker is available, much of the synchronization and management tasks will be handled by that component, relieving providers and consumers on the mobile terminal of these burdensome tasks.

IV. Broker 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 providers and consumers attached to different brokers can interact seamlessly. To achieve this, a simple event system can be implemented by an overlay network of distributed brokers for relaying subscriptions and notifications. Our system model for an overlay network of brokers working in a federation is based on the model presented by Muehl et. al. in [8] and has been extended for context subscription and notification. The conceptual development of this model with the novel provision for a mobile broker is presented in the following sections.

A. System Model

As described in [8], 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 $L_{B_i} = \{\kappa_1, \kappa_2, ..., \kappa_n\}$ 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 $N_{B_i} = \{\eta_{i1}, \eta_{i2}, ..., \eta_{in}\}$ and $N_{B_i} \subset \beta$ where β is the set of all brokers in the system.

B. Subscriptions

A subscription σ contains a stateless logical expression that is applied to a notification ν, i.e. $\sigma(\nu) \longrightarrow (true, false)$. 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 $\{\nu | \sigma(\nu) = true\}$.

C. Subscription Routing Tables

Brokers communicate with their neighbours by asynchronous communication. Each broker B_i maintains a subscription routing table T_{B_i} containing routing entries. Each routing entry is a pair $(\sigma, \mathcal{D})$ consisting of a subscription σ and a destination client $\mathcal{D} \in \kappa \cup N_B$. Hence each router maintains routing entries only for its local clients and neighbouring routers and not of the whole system entities.

D. Notifications

A client κ_i registers with the broker by providing its credentials and additional information which allows the broker to communicate with the client asynchronously for delivering notifications. The broker exposes two interfaces namely $pub(Notification\ \nu)$ and $sub(Subscription\ \sigma)$ that allow the clients to publish or subscribe to events. The broker uses a $notify(Notification\ \nu)$ message itself to deliver notifications to local clients. Moreover, it uses a message $forward(Notification\ \nu)$ to forward notifications to neighbouring brokers (brokers who have clients subscribed for the current notification).

In the following sections, we describe how this model operates for 1, 2 and *n* number of brokers.

V. Single Broker Case

The local client κ_1 of broker B_1 subscribes with the broker with subscription σ_1 using the $sub(Subscription\ \sigma_1)$ broker interface (Fig. 3). The broker saves this subscription in its subscription routing table and then determines that the local client κ_2 is capable of producing information which can, in theory, satisfy the subscription. Broker 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 subscription σ_1 is satisfied, κ_2 produces a notification ν_1 and sends it to the broker via the $pub(Notification\ \nu_1)$ broker interface. The broker consults its routing table T_{B_1} and forwards the notification to the clients with matching subscriptions, in this case κ_1.

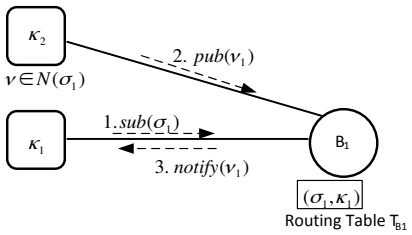

Figure 3. Subscription and notification in case of a single broker.

VI. A Case of Two Brokers

Consider this case with the help of Fig. 4 where local client κ_1 of broker B_1 subscribes with B_1 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 (ϕ denotes this entry is not for un-subscription):

$$tableUpdate(B_1, \sigma_1, \phi) \tag{2}$$

This causes the broker B_2 to update its routing table with the entry (σ_1, B_1). Broker B_2 has two registered clients K_2 and K_3. B_2 forwards σ_1 to K_2 considering it to be a potential source of matching notifications

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

κ_1 is a local client of B_1. Therefore B_1 uses the $notify(\nu_1)$ procedure to notify the client with the notification ν_1.

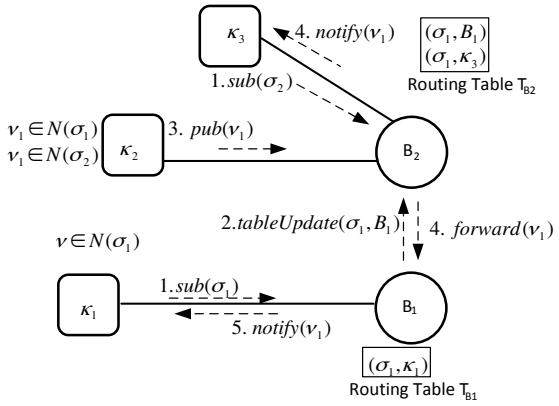

Figure 4. Subscription and notification in case of a two brokers.

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:

$$\Gamma_{B_2}(\nu_1) = \{\kappa_1, \kappa_3\} \tag{4}$$

for the notification ν_1. For the local client κ_3, B_2 will invoke $notify(\kappa_3, \nu_1)$ locally. The other match κ_1 is a remote client according to T_{B_2} and B_2 will invoke (3). For the local client κ_1 of B_1, B_1 will then invoke $notify(\kappa_1, \nu_1)$ locally.

If at some point, κ_1 sends an unsubscribe request to B_1, B_1 sends the following message to its neighbouring broker B_2:

$$tableUpdate(B_1, \phi, \sigma_1) \tag{5}$$

and B_2 removes the entry (B_1, σ_1) from its subscription routing table (ϕ means not a subscription request).

VII. A Case of *n* Brokers

Before considering the case of a broker federation consisting of an arbitrary number of brokers in the federation, let us extend the subscription model 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 unsubscription set χ). The case for *n* brokers builds on the previous case incrementally such that the subscription table updates are propagated throughout the broker federation. In simple routing configuration, subscription routing updates are initiated in response to clients subscribing or un-subscribing. The subscription routing updates reach all brokers in the federation allowing them to update their routing 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:

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

$$T_{B_i} \longleftarrow T_{B_i} \setminus \{(\sigma_j, \kappa_i) \mid \sigma_j \in \chi\} \tag{7}$$

$$\forall(\eta \in N_{B_i}),\ M_\sigma \longleftarrow \{(\sigma_j, \eta) \mid \eta \in N_{B_i} \setminus \kappa_i \wedge \sigma_j \in \varsigma\} \tag{8}$$

$$\forall(\eta \in N_{B_i}),\ M_\nu \longleftarrow \{(\sigma_j, \eta) \mid \eta \in N_{B_i} \setminus \kappa_i \wedge \sigma_j \in \chi\} \tag{9}$$

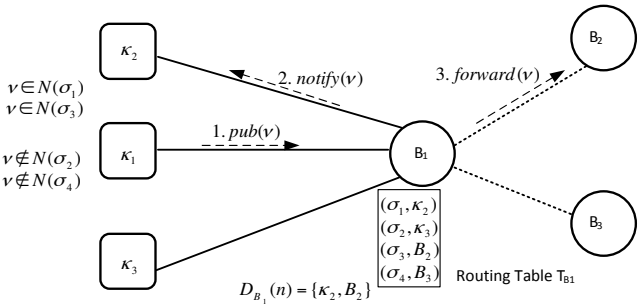

Figure 5. Subscription and notification in *n* number of federated brokers.

Eq. 6 and 7 show addition of subscriptions in subscription set ς and addition of un-subscriptions in un-subscription set χ in the routing table of the broker. For each neighbouring broker η in N_{B_i}, Eq. 8 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 except the source κ_i. Eq. 9 describes the similar mechanism for un-subscriptions.

For notifications, the brokers use the $notify(Client\ \kappa, Notification\ \nu)$ procedure as before 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 in section VI, if a broker receives a $forward(Notification\ \nu)$ 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 forwards the notification to is determined by entries in the subscription routing table. Matching subscription for a single destination $\mathcal{D}$ can be defined as:

$$\mathcal{T}^{|\mathcal{D}} \equiv \{\sigma \mid \exists(\sigma, \mathcal{D}) \in T_B\} \tag{10}$$

Consider Figure 5. κ_1 publishes a notification ν. Assume that ν_1 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 to its neighbour B_2 due to entry (σ_3, B_2).

VIII. Mobile Broker

As stated in Section 1, complex scenarios will involve a number of context providers and consumers on the mobile device in addition to applications that consume context. To mitigate the effects of issues that arise due to device mobility, intermittent connectivity, broker bound communication of all device level context consumers and providers, we propose a context broker for mobile devices that will perform context brokering at the device in coordination with network brokers. Figure 6 illustrates the introduction of a mobile broker into the overlay network of brokers. Using the procedure described in Section IV, the mobile broker becomes part of the overlay network of brokers. The exchange of subscription tables ensures

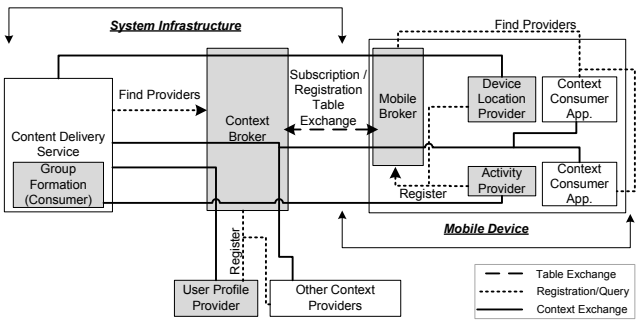

Figure 6. Mobile Broker as part of the broker federation.

that each context consumer or provider only queries its local broker for information about a provider or subscription and the rest of the network is transparent to that component. The mobile broker monitors its connectivity with the network and returns an *unavailable* response during disconnection which is interpreted by its clients as a *resend query after interval t* message. A wait queue could be implemented at the mobile broker instead of this mechanism but that induces a *blocking* behaviour in the system. Note that handling of disconnection in this manner is valid not only for mobile brokers but also system infrastructure brokers as well.

Joining and leaving of clients can be incorporated in the federation model in the following manner:

- Each broker B_i manages a set L_{B_i} containing B's current set of local clients. Initially L_{B_i} is empty.
- If a client κ_i calls one of the interface operations (pub, sub, unsub) and κ_i is not in L_{B_i}, then κ_i is added to L_{B_i}.
- If a client wants to leave the system, it calls the new *leave* interface operation. This operation removes κ_i from L_{B_i} and cancels all active subscriptions.

The availability of a brokering element on the mobile device reduces the subscription and notification management overhead for the device level context consumers and providers. It can also be used to cache recently produced context by context providers executing on the device. Notifications and subscriptions for a particular context scope can be matched/-generated by the mobile broker itself without invoking the context providers. This arrangement has the potential to reduce computational load on the individual context providing components in the mobile device.

IX. Discussion and Future Work

In this paper we have presented a theoretical model that is based on federation of context brokers to address scalability and mobility issues faced in developing and experimenting with a single-broker system that relied on synchronous communication between the system components. We have also introduced a concept of mobile broker that becomes part of the broker federation while executing on the mobile device and facilitates communication (subscription and notification) between context producers/consumers on the mobile device and rest of the context aware system. The federation of brokers is interlinked through a publish-subscribe mechanism.

A number of technologies exist that can be harnessed to realise the concepts presented here, namely CORBA Event and Notification Service [16], Java Message Service [17], WS Eventing and WS Notification [15], IBM WebShphere MQ [14], TIBCO Rendezvous [18], to name a few. Each has its own strengths and weaknesses which were not the focus of this discussion. Various aspects of the theoretical model presented here have room for improvement. Specifically, the subscription routing mechanism is simplistic and does not tackle topology changes, cyclic topologies, composite events and congestion control. Muehl et. al [8] suggest a number of improvements over this simplistic approach that target network based routing entities (brokers) but not mobile brokers. These issues are being targeted in our future work.

References

[1] Zafar, M., Baker, N., Moltchanov, B., Goncalves, J. M., Liaquat, S., and Knappmeyer M.: "Context Management Architecture for Future Internet Services". ICT MobileSummit 2009, Santander, Spain. June 2009.
[2] Moltchanov, B., Knappmeyer, M., Licciardi, C. A.: "Context-Aware Content Sharing and Casting", 12th ICIN, Bordeaux, France, October 2008
[3] Goix, L. et al.: "Situation Inference for Mobile Users: A Rule Based Approach". In International Conference on Mobile Data Management, 2007. pp. 299-303
[4] van Kranenburg, H. et al.: "A Context Management Framework for Supporting Context-aware Distributed Applications". IEEE Communications Magazine, 44(8), 67-74
[5] Chen, H., Finin, T. and Joshi, A.: "An Intelligent Broker for Context-Aware Systems". Adjunct Proceedings of Ubicomp 2003
[6] Baldauf, M., Dustdar, S. and Rosenberg, F.: "A Survey on Context-aware Systems". International Journal of Ad Hoc and Ubiquitous Computing, 2(4), 263-277
[7] "A Light-Weight Context Representation and Context Management Schema". Michael Knappmeyer, et. al., IEEE International Symposium on Wireless Pervasive Computing, Modena, Italy, May 5-7, 2010
[8] Muehl, G., Fiege, L. and Pietzuch, P. R.: Distributed Event-Based Systems, Springer-Verlag, Germany. 2007
[9] Fábián, A., Győrbíró, N., and Hományi, G.: "Activity recognition system for mobile phones using the MotionBand device", 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications, Innsbruck, Austria, 2008
[10] Choudhury et. al.: "The Mobile Sensing Platform: An Embedded Activity Recognition System", IEEE Pervasive Computing, April-June 2008, Volume: 7, Issue: 2. On page(s): 32-41
[11] Knappmeyer, M., Tonjes, R., Baker, N.: "Modular and Extendible Context Provisioning for Evolving Mobile Applications and Services", ICT Mobile Summit 2009, Santendar, Spain, June 2009
[12] Tam, M., Smith, J. M., Farber, D. J.: "A taxonomy-based comparison of several distributed shared memory systems", ACM Operating Systems Review, vol. 24, pages 40-67, 1990
[13] Blakeley, B. Harris, H., Lewis, J.: "Messaging and Queuing Using the MQI, Concepts and Analysis, Design and Development". McGraw-Hill, New York, NY, USA, 1995, ISBN:0-07-005730-3
[14] IBM Corporation: "MQSereis: An introduction to messaging and queuing", 1995
[15] Graham, S., Hull, D. , Murray, B.: "Web Services Base Notification (WS-BaseNotification) 1.3." http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf
[16] Object Management Group: "CORBA Event Service Specification version 1.2", 2004
[17] Sun Microsystems Inc.: "Java Messaging Service Specification 1.1." http://java.sun.com/products/jms/docs.html
[18] TIBCO Inc.: "TIBCO Rendezvous", http://www.tibco.com/resources/software/messaging/rendezvous_ds.pdf (accessed 3 June, 2009).