Mobile Information Systems 9 (2013) 1–17
DOI 10.3233/MIS-2012-0144
IOS Press

Energy conservation in mobile devices and applications: A case for context parsing, processing and distribution in clouds

Saad Liaquat Kiani[a], Ashiq Anjum[b], Nik Bessis[b,*], Richard Hill[b] and Michael Knappmeyer[a,c]
[a]*Faculty of Engineering and Technology, University of the West of England, Bristol, UK*
[b]*School of Computing and Mathematics, University of Derby, Derby, UK*
[c]*Faculty of Engineering and Computer Science, University of Applied Sciences Osnabrück, Osnabrück, Germany*

Abstract. Context information consumed and produced by the applications on mobile devices needs to be represented, disseminated, processed and consumed by numerous components in a context-aware system. Significant amounts of context consumption, production and processing takes place on mobile devices and there is limited or no support for collaborative modelling, persistence and processing between device-Cloud ecosystems. In this paper we propose an environment for context processing in a Cloud-based distributed infrastructure that offloads complex context processing from the applications on mobile devices. An experimental analysis of complexity based context-processing categories has been carried out to establish the processing-load boundary. The results demonstrate that the proposed collaborative infrastructure provides significant performance and energy conservation benefits for mobile devices and applications.

Keywords: Energy conservation, mobile computing, cloud computing, XML processing

1. Introduction

Context-aware systems have distinct functional elements that perform the role of data acquisition, context synthesis, context storage and dissemination. Context consumption and production applications/services are usually assisted via *context servers or brokers* within the context provisioning systems to coordinate the flow of context queries/subscriptions and responses/notifications. Context-consuming applications usually aggregate simple context information (location, time, etc.) about entities (users, devices, etc.) into complex contextual information e.g. user activity. This aggregation may involve varying levels of reasoning and parsing of contextual data. While desktop based context-consuming applications have adequate processing ability available for such purposes, the parsing, processing and reasoning about contextual information presents a significant burden upon on smart mobile devices. In this paper, we explore the possibility of delegating such tasks from device based context consumers to a Cloud-based service and evaluate the benefits of the approach in terms of processing time improvement and (energy) cost reduction on such devices.

*Corresponding author: Prof. Nik Bessis, School of Computing and Mathematics, University of Derby, Kedleston Road, Derby, DE22 1GB, UK. Tel.: +44 1332592108; Fax: +44 1332597741; E-mail: n.bessis@derby.ac.uk.

Our analysis of the focal issues is carried out within the scope of the Context Provisioning Architecture [12], which is a federated broker based system for provisioning of contextual information from Cloud-based context providing services to mobile and desktop based applications. The collection and dissemination of contextual information from Cloud-based services (context brokers and providers) contribute to the notion of 'context-aware Clouds'. However, modelling and representation of real world concepts into machine understandable format, for reasoning and consumption by applications, is a non-trivial task. Predominantly, semi-structured modelling approaches have been used for this purpose e.g. XML based schemas and Ontologies. The complexity involved in processing semi-structured data is both time and space dependent. For example, the complexity of XPath query processing increases with respect to both the size of the query and the XML document containing the information. Gottlob et al. [7] report that common implementations of XPath engines require exponential time with respect to the size of the queries in the worst case and that even the simplest queries take quadratic time in the size of the data. This complexity is significant when context consuming and producing applications execute on smart mobile devices. Furthermore, another constraint is the limited energy available to battery-powered smart mobile devices and any complex processing imposes an extra burden on the finite energy resources of the devices.

A natural solution to the issue of parsing and processing of context is to offload such tasks onto resource-capable computing systems such as those hosted in a Cloud infrastructure. But the additional communication steps incur further costs in terms of time taken to process information and energy used during communication. The querying complexity and document size boundary at which the offloading of query processing starts being beneficial in terms of time and energy costs needs to be investigated. Furthermore, offloading simple queries on small sized documents may not provide time/energy conservation benefits as compared to complex queries on large size documents. The relative benefits may also vary due to different processing capabilities of heterogeneous devices and platforms, reinforcing the notion that optimising energy consumption is a multi-objective problem [17]. This article is targeted towards investigating these issues and questions in the Context Provisioning Architecture that utilises an XML schema based model, titled ContextML [13], for modelling contextual information. The answers to the questions posed earlier will provide an assessment of the degree of query complexity and document size to be *efficiently* managed by modern devices during context consumption and processing. Furthermore, our approach presents a mechanism for context provisioning within the Cloud infrastructure itself, supporting context-aware applications and services.

Before describing our context provisioning architecture and the experimental framework that is used for studying and evaluating the issues discussed above, we describe the background and work related to the problem domain being addressed in this paper in Section 2. Description of the salient features of the federated broker [11,12] based context provisioning system, including our context modelling schema and Cloud based Context Processing Service is provided in Section 3. Thereafter, Section 4 describes the experimental evaluation of reduced context processing on the device, via offloading to a Cloud service, and discusses the experimental results of varying scenarios. We conclude this article in Section 5 by elaborating the general future direction and specific targets planned for further exploration of the issues discussed in this article.

2. Background and related work

Korpipää et al. [15] have specified some basic requirements for designing a contextual 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). XML and Ontology based context modelling and representation schemes generally provide adequate coverage of these properties, with each scheme offering scenario and usage dependent strengths and weaknesses e.g. XML Schema based approaches are inherently simpler and light-weight in comparison to ontology based approaches but lack the generality offered by ontologies in terms of modelling a wider spectrum of domain knowledge. The simplicity of XML based modelling facilitates portability across heterogeneous devices e.g. designing uniform user–device interaction mechanisms [19]. Ontologies have been used for describing human-domain concepts on mobile devices [9] and Ontology as a Service (OAAS) has also been proposed [5] for extracting and merging ontological concepts in the Clouds. However, in this article we will only consider the processing of contextual information represented using XML based schemas.

XPath [3], proposed by the World Wide Web Consortium is a widely adopted XML query language for selecting nodes from XML document trees. A drawback of XPath based query processing is that it leads to an exponential time complexity [7,20]. The complexity of an XPath query is largely dependent on the predicates and assertions specified in the query e.g. attributes and elements to match, and the size/depth of the document being processed. Efficient implementations of XPath processing have been proposed in literature that achieve polynomial (combined) complexity [7]. However, such improved implementations have not been integrated in widely deployed XPath engines such as those available in Java, Android and .NET platforms.

A consequence of the complexity of XPath processing is that it is costly, in terms of time and space, to process large documents and/or complex queries, especially on mobile devices. Context data has a high degree of temporal significance; a context datum may be valid at one point in time and invalid at the next. Processing delays may invalidate the integrity of processed information. Hence, it is important to process contextual data as soon as possible and within the temporal bounds of the data itself. The computational complexity of executing XPath queries on mobile devices also affects the limited energy resources. This issue is one of the main thrusts of this paper and we aim to identify the XPath query processing and offloading scenarios that lead to energy conservation on mobile devices by reduction of the processing burden. Before describing the experimental framework to analyse the XPath query processing costs, in terms of time and energy consumption on modern mobile devices, it is necessary to describe our federated broker based context provisioning system, titled the Context Provisioning Architecture, and the ContextML schema.

3. Context provisioning architecture

3.1. Consumer-Broker-Provider model

The Context Provisioning Architecture is based on the producer (provider)-consumer model in which context related services 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 if and when it is available.

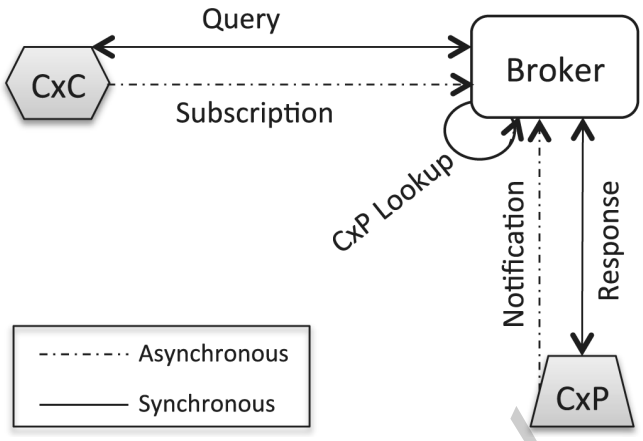


Fig. 1. Basic mobile broker based context provisioning component interaction.

Context provider – The Context Provider (CxP) component provides contextual information. A CxP gathers data from a collection of sensors, network services or other relevant sources. A CxP may use various aggregation and reasoning mechanisms to infer context from raw sensor, network or other source data. A CxP provides context data only 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 among 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. A number of useful applications have been developed based on this architecture. Further details of this architecture and industrial trials are described in [14,21].

Context consumers and providers register with a broker by specifying its communication end point and the type of context they provide or require. This in turn enables the brokering function in which the context broker can lookup a particular context provider that a context consumer may be interested in (e.g. based on the type of context being requested). The broker can cache recently produced context, in order to exploit the principle of locality of reference, as done routinely in internet communications to improve overall performance. A distinguishing feature of this architecture is the federation of multiple context brokers to form an overlay network of brokers (Fig. 2), which improves scalability of the overall system and provides location transparency to the local clients (CxCs and CxPs) of each broker. This federation of context brokers is achieved with a coordination model that is based on routing of context queries/subscriptions and responses/notifications across distributed brokers, discovery and lookup functions and is described in detail in our earlier work [11].

3.2. ContextML

ContextML is an XML based schema for the representation of contextual information. The defining principle in ContextML is that context data relates to an entity and is of a certain scope (Fig. 3). The entity may be a user, a username, a SIP or email address etc., and scope signifies the type of context data e.g. weather, location, activity and user preferences.

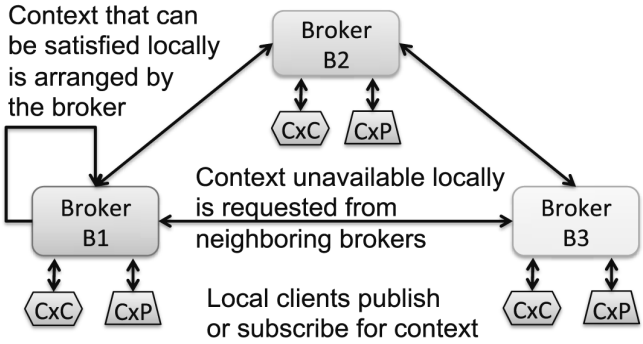


Fig. 2. Simplified view of the mobile broker based interaction.

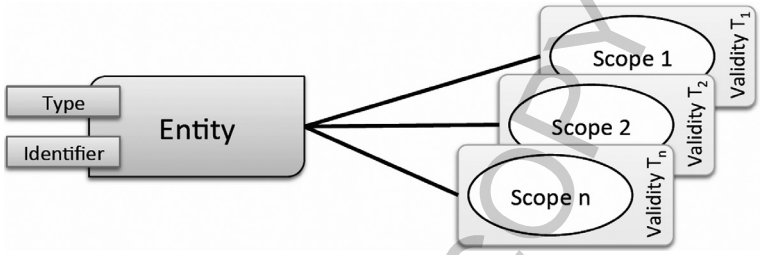


Fig. 3. Entity-scope relationship. An entity can have many context scopes associated with it. Each scope instance has a validity period after which the specific context instance is considered invalid.

A specific context instance in ContextML is called a *context element* and contains the actual context data and also context meta-data. Context data is represented in context parameters, which are name-value pairs, arrays of context parameters, or in structures that are a collection of context parameters and context parameter arrays. The model of the context element is depicted in Fig. 4 and the corresponding ContextML schema is also listed in Fig. 5.

In addition to the representation of contextual data, ContextML also contains a specification for control messages between components, subscriptions and notifications, component advertisements and routing related messages that are utilised in the overall system for coordination of context exchange. The ContextML Parser has been implemented as a Java library for Java SE, EE and the Android platforms that can be used by context producing and consuming applications for the processing of contextual information and other messages encoded in ContextML. The model of the contextual data-related elements is depicted in Fig. 4, detailed discussion about various dimensions of ContextML is presented in an earlier work [13].

3.3. Context processing service

As discussed above, CxCs request context by subscribing with their CxBs. The CxBs look up the relevant CxPs and forward the subscriptions to the CxPs, which reply with the context data that satisfies the particular context subscription. The CxBs maintain subscription tables and forward incoming notifications to the subscribing consumers. A context consuming application usually subscribes to multiple context scopes in our architecture and has to infer complex context by processing multiple context messages. Keeping in mind the increasingly involved role of mobile devices in human-computer interaction and ubiquitous computing, the Context Provisioning Architecture allows the context consuming applications on mobile devices to submit XPath queries to be applied to context notifications generated

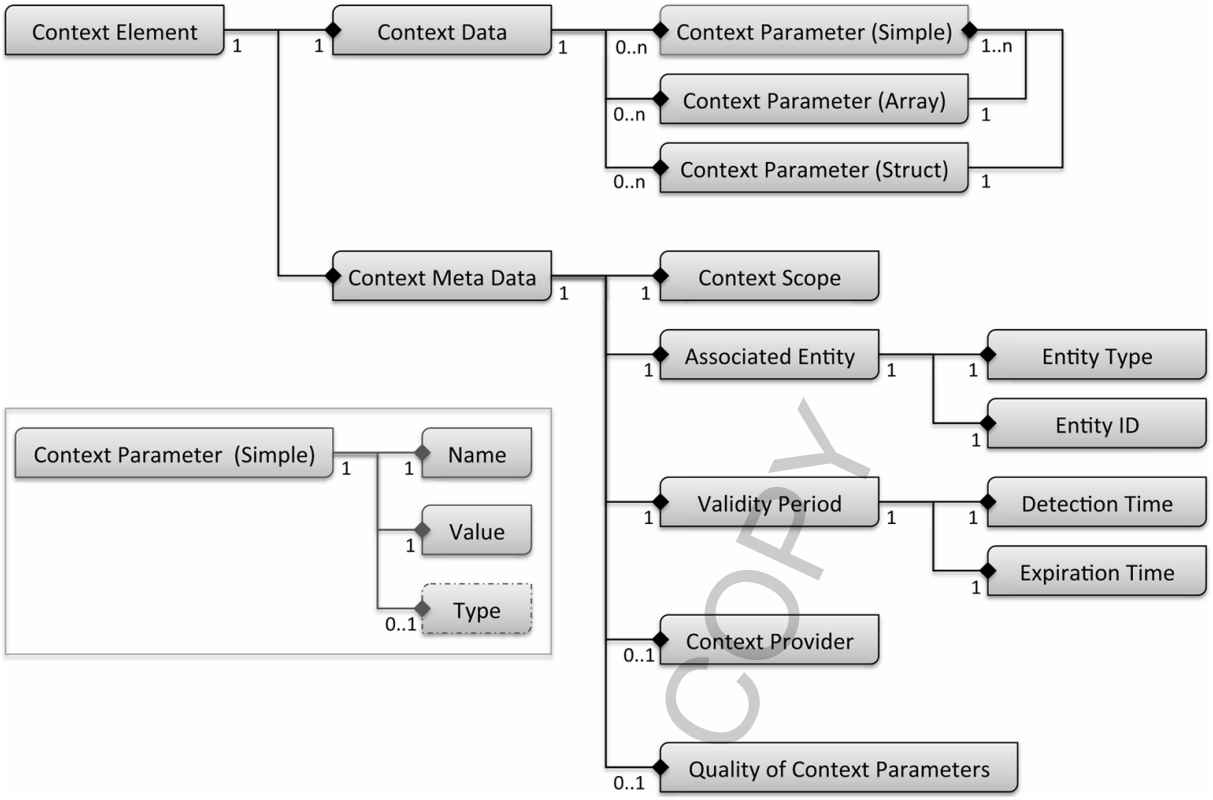


Fig. 4. ContextML model of the *ctxEl* element.

in response to their subscriptions as well. Furthermore, the context consumers can submit contextual data and processing queries to the context brokers in order to decrease their processing burden. The context brokers provide these facilities through a Context Processing Service (CPS), which exposes RESTful HTTP based interfaces for CxCs to submit queries for processing over contextual data. The query and context data can be provided by the CxCs during the invocation of CPS or via the CxB (cp. Fig. 6). Upon completion of query processing, the CPS returns the results to the submitting client. It is expected that this model of offloading complex processing tasks should be beneficial in terms of time and space complexity on mobile devices. Consequently, such a mechanism would also affect the energy consumption on mobile devices and result in better utilisation of this critical resource.

Despite the simplistic offloading mechanism's obvious benefits, there are critical questions to be addressed. Firstly, the trade-off between the efficiency improved by offloading complex tasks, and that compensated by addition of the remote communication process, is not yet evident and quantified. Secondly, all contextual processing is not complex. The processing complexity of XPath, being used in our analysis, is dependent both on the size of the document (ContextML based information), its structure (depth of the document tree) and the complexity (number of parameters) of the XPath expression/query. The processing cost of simple XPath expressions on small sized ContextML documents may be less than that incurred when offloading to a remote Cloud based service due to addition of the communication cost incurred during data transfer from device to the Cloud service. The size of the ContextML document, the complexity of the XPath query, the processing time, the communication cost and energy consumption are the main parameters that need to be evaluated in order to establish the realistic benefits of offloading

```
<xs:element name="ctxEl">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="contextProvider">
      </xs:element>
      <xs:element name="entity" minOccurs="0">
        <xs:complexType>
          <xs:attribute name="id" type="xs:string" use="required"/>
          <xs:attribute name="type" type="xs:string" use="required"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="scope" type="xs:string">
      </xs:element>
      <xs:element name="timestamp" type="xs:dateTime">
      </xs:element>
      <xs:element name="expires" type="xs:dateTime">
      </xs:element>
      <xs:element name="dataPart">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="par" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="parS" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="parA" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Fig. 5. ContextML schema for the context data element.

complex query processing from mobile devices to the Context Processing Service. The following section describes our experimental analysis of these factors and its results in detail.

4. Context processing offloading analysis

4.1. Experimental framework

Figure 7 depicts the system architecture of the Context Provisioning Architecture relevant to the focus of this discussion. The experimental framework consists of a CxB deployed on a network host that is a compute node of an OpenStack [4] based cloud platform. The CxB itself executes on a Glassfish [8] application server. A CxC is deployed on an Android based mobile device [1] (Google Nexus One). The device and the CxB are connected through an IEEE 802.11g based WLAN access point, which provides the public network interface to the OpenStack compute node. The CxB exposes RESTful HTTP based interfaces to the CxC for requesting context from CxPs and submitting contextual queries for processing to the CPS. Context queries and responses are exchanged between the interacting components in the form

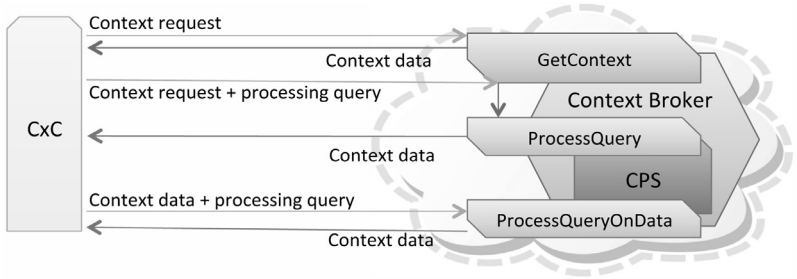


Fig. 6. RESTful HTTP interfaces exposed by the CxB and CPS.

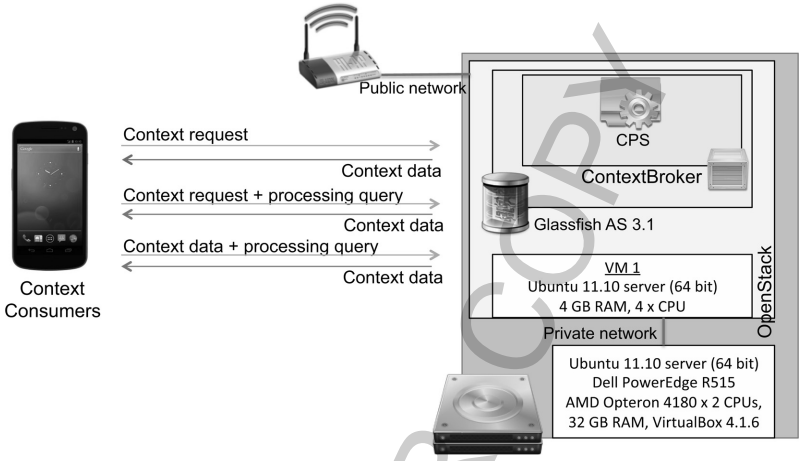


Fig. 7. Mobile system's architecture with focus on components relevant to the experimental framework.

of ContextML encoded messages over the RESTful HTTP interfaces. The endpoints of the CxB interfaces are manually hardcoded into the mobile device based CxC for the purpose of this experiment. A dynamic discovery mechanism (for example one based on QoS parameters of the Cloud-based services [2]) can be used in a real wold setup. In order to reduce the variable parameters from the experimental framework, CxPs are not utilised rather the CxB is provided with stored context notifications messages spanning 10 context scopes and 10 entities. The contextual data has been recorded from real-world deployments of various context providers of the Context Provisioning Architecture and includes contextual scopes of user location (GPS), weather, proximity, activity, wi-fi (nearby access points), presence (nearby bluetooth devices), user profile, device status, device settings and device location (cellular). The document sizes of the ContextML encoded context information range from 2 KB to 32 KB; Figs 8 and 9 show example ContextML snippets from such documents.

The CxC on the mobile device is also provided with these ContextML documents for use when the cost of local device-based processing is to be evaluated. The CxC is also configured to issue context queries spanning these documents. The queries are divided into four categories of complexity, consisting of:

1. *QT1*, an XPath expression that searches for a fixed string value of parameter *n* in any element i.e. //*[contains(@n,'string')]
2. *QT2*, an XPath expression that searches for two different string values of any parameter in any element i.e. //*[contains(.,'string')] | //*[contains(.,'string')]

```
<contextML>
 <ctxEls>
  <ctxEl>
   <contextProvider id="DevCxP-354957030977071" v="0.8" />
   <entity id="354957030977071" type="imei" />
   <scope>wf</scope>
   <timestamp>...</timestamp>
   <expires>...</expires>
   <dataPart>
    <par n="wfList">...</par>
    <parA n="wfDevices">
     <parS n="wfDevice">
     <par n="wfName">SLK</par>
     <par n="wfBssid">0024B29A4D18</par>
     <par n="wfType">[WPA-PSK-TKIP]</par>
     <par n="wfSignal">-77</par>
     <par n="wfOpen">false</par>
     </parS>
    </parA>
   </dataPart>
  </ctxEl>
 </ctxEls>
</contextML>
```

Fig. 8. ContextML snippet from a document containing contextual information about the detected and known Wi-Fi access points in proximity to an entity.

3. *QT3*, an XPath expression that searches for a particular combination of parameter *n* values along an element path /parS/par i.e. //parS[contains(@n,'string')] /par[contains(@n,'string')]
4. *QT4*, an XPath expression that retrieves all parameter *n* values from the document i.e. //*[@n]

4.2. Results

Figure 10a illustrates the mean query satisfaction times of all types of queries on different ContextML document sizes when they are processed locally on the mobile device. It is evident that QT1, QT2, QT3 and QT4 provide an increasing order of complexity, with the mean satisfaction times increasing significantly as the document sizes increase. Figures 10b, 10c, 10d and 10e illustrate the reduction in mean query processing times when QT1, QT1 + QT2, QT1 + QT2 + QT3 and all types of query processing are offloaded to the CPS, respectively. The communication time between the device and the CPS is minimal as the experiments are performed in an isolated WLAN.

Figure 11 aggregates the various scenarios and demonstrates the gradual reduction in mean query processing times as subsequent query types are offloaded to the CPS. The graph shows that the relevant benefits, in terms of improved processing time, increase with the increase in the document size and the complexity of the queries. We can observe from Fig. 10 that the there is a minor improvement in mean query processing time between the cases when QT1 is offloaded to the CPS and when QT1 + QT2 are offloaded. However, the improvement is greater when complex query types (QT3 and QT4) are offloaded as well. These trends are visible from the slope of the lines representing each experiment

Table 1
Query processing times

Query processing location†	Total processing time across all document sizes (2K–32K) [ms]
All QT‡ on Device	2314
QT1 CPS	2068.25
QT1+QT2 CPS	1708.25
QT1+QT2+QT3 CPS	1144.75
QT1+QT2+QT3+QT4 CPS	246.25

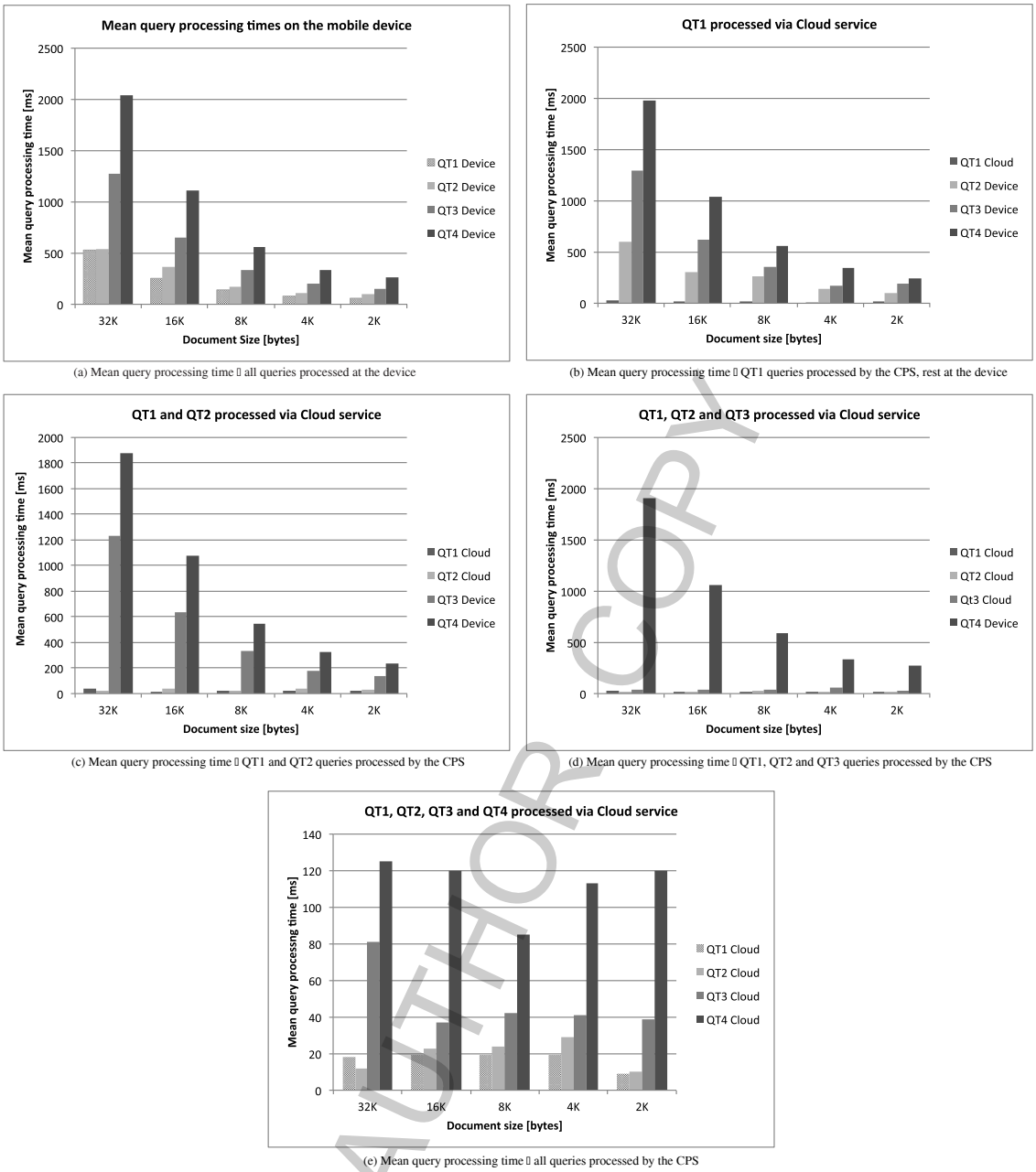
†Device is the mobile device used in the experiment, whereas CPS is the Context Processing Service deployment on the Cloud.
‡QT stands for Query Type – the four query types used in the experiments are described in Section 4.1.

```
<contextML> <ctxEls> <ctxEl>
   <contextProvider id="BPCxP" v="1.0" />
   <entity id="3P31-Frenhay-UWE" type="room" />
   <scope>btPresence</scope>
   <timestamp>...</timestamp>
   <expires>...</expires>
   <dataPart>
    <parA n="entities">
     <par n="btAddress">00:23:76:79:C7:33</par>
     <par n="btAddress">00:23:76:B0:34:A1</par>
     <par n="btAddress">00:23:76:05:38:CA</par>
    </parA>
    <parS>
     <parA n="civilAddress">
      <par type="streetAdress">3P31, Frenchay Campus, Coldharbour Lane,
        University of the West of England, Bristol, BS16 1QY, UK</par>
      <par n="name">3P31</par>
      <par n="street">Coldharbour Lane</par>
      <par n="adminCode">BS16 1QY</par>
      <par n="locality">Frenchay Campus</par>
      <par n="adminUnit">Univeristy of the West of England</par>
      <par n="city">Bristol</par>
      <par n="countryCode">UK</par>
     </parA>
     <parA n="geographicalAddress">
      <par n="latitude"></par>
      <par n="longitude"></par>
     </parA>
    </parS>
   </dataPart>
  </ctxEl> </ctxEls> </contextML>
```


Fig. 9. ContextML snippet from a document containing contextual information in the *presence* scope (Bluetooth based).

set in Fig. 11. Table 1 further quantifies the total time taken when a certain experiment set is carried out across all document sizes. The results show that offloading complex query types to the CPS provide greater improvements in the overall mean query processing time.

The reduction in processing burden also affects the energy consumption on the device. Figure 12 illustrates the energy consumption plot during different query processing scenarios. PowerTutor [22]



(a) Mean query processing time – all queries processed at the device

(b) Mean query processing time – QT1 queries processed by the CPS, rest at the device

(c) Mean query processing time – QT1 and QT2 queries processed by the CPS

(d) Mean query processing time – QT1, QT2 and QT3 queries processed by the CPS

(e) Mean query processing time – all queries processed by the CPS

Fig. 10. Query processing times – experimental results.

is used for calculating the energy used by individual applications (broker, consumers and providers) on the device. PowerTutor is an application for Google phones that displays the power consumed by major system components such as CPU, network interface, display, etc. and different applications. This application allows software developers to see the impact of design changes on power efficiency. PowerTutor calculates the phone's breakdown of power usage with an average of 1% error over 10-second intervals as well as the worst case error over 10 seconds is 2.5% [22, p. 8].

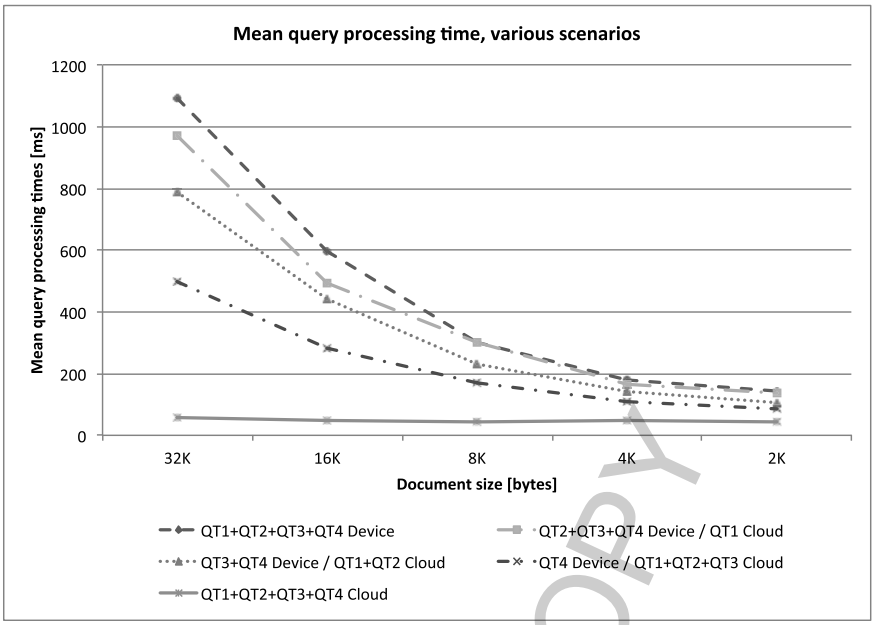


Fig. 11. Mean query processing times – various scenarios.

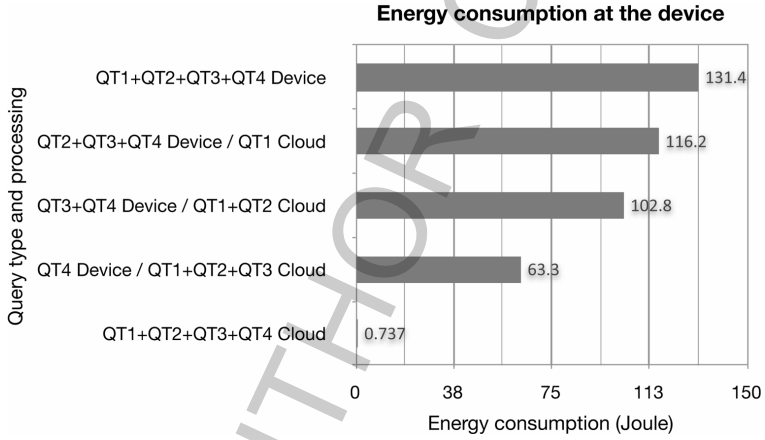


Fig. 12. Energy consumption at the mobile device – various scenarios.

In these experiments only the energy used by an application in utilising the CPU and Wi-Fi is considered when calculating its energy consumption signature. Executing all query sets on the device consumes 131.4J in our experiment scenario. Offloading QT1 and QT2 to the CPS results in a reduction of about 11.5% energy cost at each step, whereas the offloading of QT1, QT2 and QT3 to the CPS results in 38.4% reduction in energy consumption when compared to offloading only QT1 and QT2 queries. Expectedly, offloading all queries to the CPS results in minimal energy consumption at the device; 0.74J in our experiments. These relative energy consumption measurements follow a similar trend as in case of mean query processing time (Fig. 11, Table 1) i.e. energy consumption improves when complex query types are offloaded to the CPS. These results signify the benefit of our approach in terms of energy conservation on mobile devices involved in context consumption and processing.

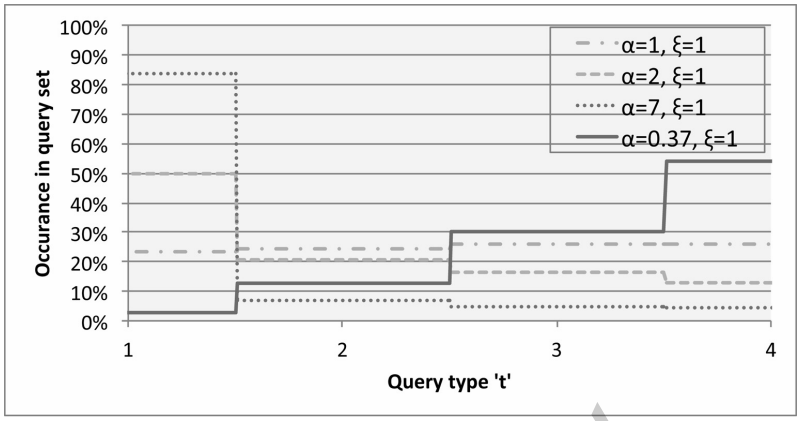


Fig. 13. Pareto distributions of query types with $\alpha = 0.37$, 1, 2 and 7. *T* is the set of possible query types. The Pareto *shape* parameter α affects the bias towards one of the query types in the set *T*.

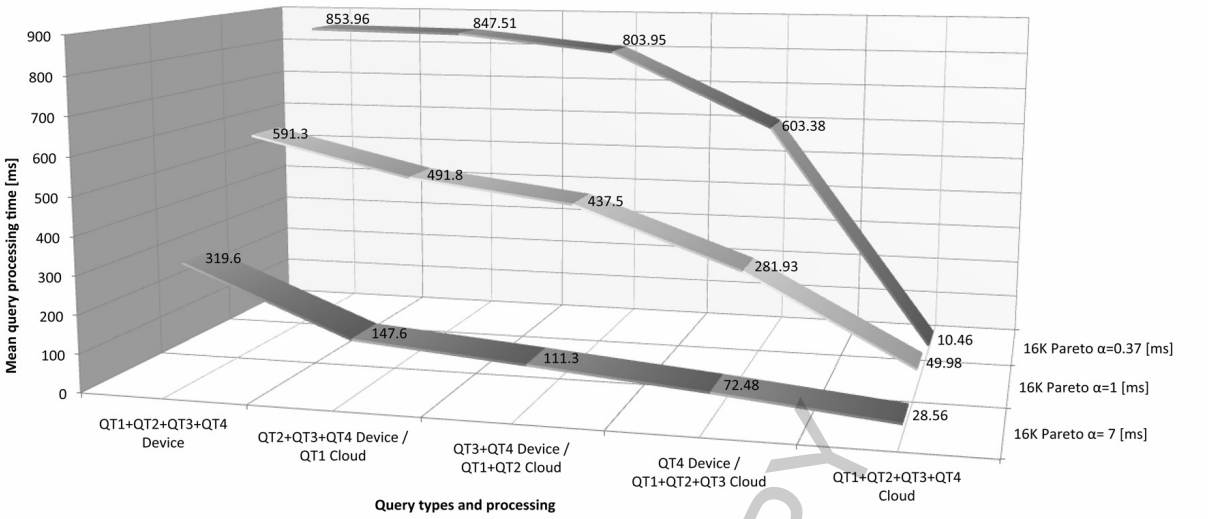
The queries in each complexity category are uniformly distributed in the experiments. In realistic scenarios, the distribution of query types during a certain duration may be biased towards complex or simplistic query types i.e. a non-uniform distribution of different query types in the overall query set. In order to analyse the effect of non-uniform distribution of contextual queries in our setup, we have repeated the experiments by non-uniformly distributing the type of context queries in the overall query set. We can model a non-uniform distribution of query types by using the Pareto distribution to influence the likelihood of a particular query type in the overall query set. During the execution of further experiments, the next query type q_n is selected by the Pareto distribution with a shape α and a scale ξ as shown in Eq. (1).

$$q_n = \left\lceil |T| \cdot \left(\frac{\xi}{randomUniform\,(0,1]} \right)^{-\alpha} \right\rceil \tag{1}$$

Figure 13 illustrates the different distributions obtained by varying the shape α (keeping the Pareto scale parameter ξ constant). The Pareto scale $\alpha = 0.37$ creates a scenario where almost half the queries consist of QT4 queries (with less than 5% QT1), $\alpha = 1$ creates a nearly uniform distribution, whereas $\alpha = 7$ creates a long tail distribution with the majority of queries ($> 80\%$) from the QT1 category. The results of the earlier experiments with these distributions are plotted in Fig. 14.

When all queries are processed on the device, the query set consisting of a greater percentage of QT4 (relatively complex queries, $\alpha = 0.37$) results in the largest mean query satisfaction time (853.96 ms). Within this distribution, the offloading of QT1 and QT2 to the CPS does not result in much improvement in the mean query satisfaction time because queries in these categories occur less than 15% of the time (see Fig. 13) and they are also less time consuming in comparison to QT3 and QT4 (see Fig. 10a). There is a notable decrease in the mean query satisfaction time with this series as QT4 is also offloaded to the CPS. This can be explained by the fact that QT4 is the most complex query type being considered in the experiments and is the most time consuming to process on the device (see Fig. 10a).

The series with $\alpha = 1$ illustrates the results of nearly equal distribution of different types of queries in the experiment and results are identical to those discussed earlier (cf. Fig. 11). The series with $\alpha = 7$ illustrates the results of the scenarios where the majority of queries consists of QT1, with QT2, QT3 and QT4 queries occurring only about 17% of the time. Since QT1 type queries are easily



	QT1+QT2+QT3+QT4 Device	QT2+QT3+QT4 Device / QT1 Cloud	QT3+QT4 Device / QT1+QT2 Cloud	QT4 Device / QT1+QT2+QT3 Cloud	QT1+QT2+QT3+QT4 Cloud
16K Pareto α= 7 [ms]	319.6	147.6	111.3	72.48	28.56
16K Pareto α=1 [ms]	591.3	491.8	437.5	281.93	49.98
16K Pareto α=0.37 [ms]	853.96	847.51	803.95	603.38	10.46

Fig. 14. Mean query processing times over 16 kilobyte sized documents with $\alpha = 0.37$, 1 and 7.

(quickly) processable on the device, the mean query satisfaction times are much lower for these scenarios, improving further as subsequent query categories are offloaded to the CPS. Naturally, a sudden decrease in the mean query satisfaction time is observed as QT1 queries are offloaded to the CPS because most of the queries in this experiment iteration are of the type QT1. These results are in line with the expected trend that offloading queries to the CPS results in improved mean query satisfaction time. Moreover, this improvement is not effected by the *non-uniform* distribution of queries (based on types) in the overall query set. However, the improvement in the mean query satisfaction time is dependent on not only the complexity of the queries but also on the relative ratios of different query types in the query set.

5. Conclusion and future work

This article has briefly described the Context Provisioning Architecture, focusing upon the analysis of ContextML query processing on a smart mobile device, and experimentally evaluating the benefits of offloading complex query processing to a Context Processing Service.[1] The RESTful HTTP based access of the CPS and the CxBs can ideally be utilised by deployment in a Cloud infrastructure to exploit the benefits of scalability in increasing load scenarios, providing device independence, and improved reliability. Our experimental framework has considered XPath based queries in four categories of complexity and demonstrated the computational cost associated with such processing on a mobile device. The experiments evaluated the improvements in query processing times, as the processing

[1]This article extends the results reported earlier in [10]. The previous version of the paper reported preliminary results limited to a uniform distribution of different query types in the analysis. This article expands the scope of the analysis by also considering non-uniform query type distribution, which has been modelled through a discretised Pareto distribution.

of queries in various complexity categories is offloaded to the CPS. The results have demonstrated significant improvement in the overall query processing times. This improvement has a direct effect on the energy consumption of the mobile device during the query processing. Furthermore, we have repeated the experiments with a non-uniform distribution of different query types in the overall query set and reported the results that are consistent with earlier observations.

In our experimental setup, we have only considered device-Cloud communication over the Wi-Fi interface of the device, whereas a real-world scenario may involve a greater portion of communication being carried through the 3G interface on the device, which has different speed and energy consumption characteristics. This factor may influence the energy consumption and improved processing time related benefits that we have quantified in this paper. We have also not considered the mobility of context consumers in our experimental setup, though the Context Provisioning Architecture supports the mobility of such components from one context broker to another. However, our existing mobility management mechanism could be improved through pro-active caching as proposed in [6].

Furthermore, the queries considered in these experiments represent quite a limited subset of possible XPath queries that can be issued over ContextML documents and represent fairly simplistic expressions. XPath based queries that are much complex than these, such as those that consider both ancestor and descendant relationships, can be analysed. XPath query optimisation techniques, such as semantic query transformations [16] can also be explored to optimise the processing of complex queries in our system. Moreover, our analysis has been restricted to fixed sized ContextML documents during the experiments. While this restriction reduces the number of variable factors in the analysis, it is not expected to occur too often in a real world scenario and hence further analysis with variable ContextML documents will improve the generalisability of our results. We are also considering the implications of real-world factors on the performance of our experimental setup e.g. the location of mobile clients, network conditions and other similar constraints affecting the query processing times [18]. These issues are the main focus of our future work.

References

[1] M. Butler, Android: Changing the mobile landscape, *IEEE Pervasive Computing* **10** (2011), 4–7.

[2] C.-C. Chuang and J. Kao, Shang, Adjustable flooding based discovery with multiple QoSs for cloud services acquisition, *Int J Web Grid Serv* **7** (May 2011), 208–224.

[3] J. Clark and S. DeRose, XML path language (XPath) version 1.0, *W3C Recommendation*, 16, 1999, http://www.w3.org/TR/xpath/.

[4] R. Clark, Creating a rackspace and NASA Nebula compatible cloud using the OpenStack project, in: *AGU Fall Meeting Abstracts*, volume 1, 2010, p. 01.

[5] A. Flahive, D. Taniar and W. Rahayu, Ontology as a service (OaaS): a case for sub-ontology merging on the cloud, *The Journal of Supercomputing*, pp. 1–32. 10.1007/s11227-011-0711-4.

[6] A. Gaddah and T. Kunz, Extending mobility to publish/subscribe systems using a pro-active caching approach, *Mobile Information Systems* **6** (December 2010), 293–324.

[7] G. Gottlob, C. Koch and R. Pichler, Efficient algorithms for processing XPath queries, *ACM Transactions on Database Systems* **30**(2) (2005), 444–491.

[8] D. Heffelfinger, M. Verma, K. Lewis and I. Books24x7, *Java EE 5 Development using GlassFish Application Server*, Packt Pub., 2007.

[9] S. Jan, M. Li, G. Al-Sultany, H. Al-Raweshidy and I.A. Shah, Semantic file annotation and retrieval on mobile devices, *Mobile Information Systems* **7** (April 2011), 107–122.

[10] S.L. Kiani, A.M. Anjum, N. Bessis, R. Hill and M. Knappmeyer, Context parsing, processing and distribution in clouds. in: *Proceedings of IEEE INCoS-2011: Third International Conference on Intelligent Networking and Collaborative Systems*, Fukuoka, Japan, November 2011. IEEE.

[11] S.L. Kiani, M. Knappmeyer, N. Baker and B. Moltchanov, A federated broker architecture for large scale context dissemination, in: *Proceedings of the International Conference on Computer and Information Technology*, Los Alamitos, CA, USA, 2010, pp. 2964–2969. IEEE Computer Society.

[12] S.L. Kiani, B. Moltchanov, M. Knappmeyer and N. Baker, Large-scale context-aware system in smart spaces: Issues and challenges, in: *Baltic Congress on Future Internet and Communications – Special Session on Smart Spaces and Ubiquitous Solutions*, Riga, Latvia, February 2011.
[13] M. Knappmeyer, S.L. Kiani, C. Frá, B. Moltchanov and N. Baker, A light-weight context representation and context management schema, in: *Proceedings of IEEE International Symposium on Wireless Pervasive Computing*, May 2010, pp. 367–372.
[14] M. Knappmeyer, R. Tönjes and N. Baker, Modular and extendible context provisioning for evolving mobile applications and services, In *18th ICT Mobile Summit*, Santander, Spain, June 2009.
[15] P. Korpipää, J. Kela and E.J. Malm, Managing context information in mobile devices, *IEEE Pervasive Computing* **2**(3) (2003), 42–51.
[16] D. Le, S. Bressan, D. Taniar and W. Rahayu, Using semantics for XPath query transformation, *International Journal of Web and Grid Services* **6**(1) (2010), 58–94.
[17] C. Li and L. Li, Energy efficient resource management in mobile grid, *Mobile Information Systems* **6** (April 2010), 193–211.
[18] Z. Mammeri, F. Morvan, A. Hameurlain and N. Marsit, Location-dependent query processing under soft real-time constraints, *Mobile Information Systems* **5** (August 2009), 205–232.
[19] A. Santangelo, A. Gentile, G. Vella, N. Ingraffia and M. Liotta, XPL – the extensible presentation language. *Mobile Information Systems* **5** (April 2009), 125–139.
[20] P. Wadler, Two semantics for XPath. Technical report, Bell Labs, 2000, http://homepages.inf.ed.ac.uk/wadler/papers/xpath-semantics/xpath-semantics.pdf.
[21] M. Zafar, N. Baker, B. Moltchanov, J.M. Goncalves, S.L. Kiani and M. Knappmeyer, Context management architecture for future internet services, in: *ICT Mobile Summit 2009*, Santander, Spain, June 2009.
[22] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R.P. Dick, Z.M. Mao and L. Yang, Accurate online power estimation and automatic battery behavior based power model generation for smartphones, in: *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES/ISSS '10, New York, NY, USA, 2010, pp. 105–114. ACM.

Dr. Saad Liaquat 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. He has recently been awarded PhD in Computer Science from UWE. 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 lecturer in the School of Computing and Mathematics at the University of Derby. His areas of expertise include Distributed, Parallel and Concurrent Systems (mainly Cloud and Grid Computing). He has worked on a number of scientific and industrial projects (notable contributions include FP7 funded TRASNFORM project, FP7 funded neuGrid project, FP6 funded Health-e-Child project, NSF/DOE funded CAIGEE project and US Aid funded GAE project) in collaboration with leading industrial and academic partners in Europe and USA. He has more than 10 years of experience in academic and industrial research and has more than 50 publications to his credit.

Dr. Nik Bessis is currently a 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. Prof. Bessis has published over 120 papers, won 2 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, Prof. Bessis 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.

Dr. Richard Hill is Head of Subject, Computing and Mathematics, and a Reader in Intelligent Systems at the University of Derby. His research interests include Agent Oriented Software Engineering, collective intelligence, emergence in systems, and enterprise Cloud Computing. Dr. Hill has published over 100 articles and edited several books and journal special issues.

Dr. Michael Knappmeyer studied Computer Engineering and Computer Science at the University of Applied Sciences Osnabrück (UASO) and received his Diplom-Informatiker (FH) degree in August 2006. Afterwards he joined the Mobile Communication Research Group at UASO as a Research Associate. He has participated in the European FP6 IST project C-MOBILE and the FP7 ICT project C-CAST. In the latter he led the context reasoning related activities. Michael has recently defended his PhD dissertation at the University of the West of England, which concentrates on a Context Provisioning Middleware with Support for Evolving Awareness. His interests include smart spaces, context modelling, reasoning and mobile multicast/broadcast.