Supporting Context-Awareness: A Taxonomic Review

Éamonn Linehan

Distributed Systems Group, Department of Computer Science, Trinity College Dublin, Dublin 2, Ireland.

Shiu Lun Tsang

Distributed Systems Group, Department of Computer Science, Trinity College Dublin, Dublin 2, Ireland.

Siobhán Clarke

Distributed Systems Group, Department of Computer Science, Trinity College Dublin, Dublin 2, Ireland.

Abstract

Context-aware applications realise the vision of ubiquitous computing by utilising information gathered from their environment to automatically adapt behaviour. To support the development of such applications, researchers have developed infrastructures, architectures, middlewares, and toolkits. This paper presents a taxonomy based on a review of these support infrastructures for context-aware computing. The taxonomy is organised as a set of categories, each of which is responsible for classifying context-related functionality at the software component level. It provides the nomenclature of terminology used to describe the features that characterise infrastructures for context-aware computing. The taxonomy supports the design of context-aware systems and provides a convenient way to compare and contrast middleware support for context-awareness.

Key words: Architecture, Context-Aware, Infrastructure, Taxonomy
1991 MSC: 68N01, 68M99

Email addresses: Eamonn.Linehan@cs.tcd.ie (Éamonn Linehan), ShiuLun.Tsang@cs.tcd.ie (Shiu Lun Tsang), Siobhan.Clarke@cs.tcd.ie (Siobhán Clarke).

URLs: https://www.cs.tcd.ie/Eamonn.Linehan/ (Éamonn Linehan), https://www.cs.tcd.ie/ShiuLun.Tsang/ (Shiu Lun Tsang), https://www.cs.tcd.ie/Siobhan.Clarke/ (Siobhán Clarke).

1 Introduction

The vision of ubiquitous computing, according to Weiser, is one of many small devices, seamlessly integrated into the environment, and working "invisibly" in the background to aid users with their everyday tasks (75). The aim is to provide applications with the ability to gather information from their surroundings and to use this "context" to automatically adapt application behaviour. Pascoe et al. (62) concluded that "the core of a supporting infrastructure for context-aware computing should be a context information service". The responsibilities of such a service would be to gather, model, and provide contextual information to context-aware applications to allow them automatically adapt application behaviour.

This desire to produce a "supporting infrastructure" for context-aware computing has led to a focus on developing middlewares, frameworks, toolkits, and infrastructures [1] (all of which support the development of context-aware applications). In this paper we collectively refer to these systems as types of context-aware infrastructure, where we define infrastructure as a set of technologies that act as a foundation for context-aware applications. Riva uses the term "context provisioning middleware" to label this same group of systems (65). These infrastructures support context-aware computing by providing "uniform abstractions of common functionality and reliable services for common operations" in order to make it easier to develop robust applications even on a diverse and constantly changing set of devices and sensors (40). However, many context-aware infrastructures have been devised in an "improvised" fashion, with developers choosing methods that are quick and easy over creating generalisable and reusable components (75). As a result, there are a large number of context-aware infrastructures with none widely adopted.

There exists a set of common components that form the core of a context-aware infrastructure. These core components are the building blocks of ubiquitous computing software. However, there exists a lack of convention in labelling these building blocks, making direct comparisons between context-aware infrastructures difficult. This inevitably leads to increased naive reimplementation of context-awareness functionality, redirecting development effort to infrastructural issues and slowing advances in context-aware application development and deployment.

Our experience in developing a context-aware infrastructure combined with a broad review of existing context-aware infrastructures has allowed us to identify a common system of names and terms for these building blocks. From this system of names a systematic taxonomic classification of functionally

[1] See Hong et al. for a discussion on the differences between these terms (40).

equivalent components based on the similarities in design and processing is possible.

This paper provides a taxonomic classification of the components of existing context-aware infrastructures. By classifying their features, we provide a means to compare different infrastructures based on the components and functionality they support. The taxonomy is presented as a set of functional categories, organised hierarchically according to component composition relationships. That is, each recognisable taxonomic unit is sub-divided based on a natural classification of the functionality it contains.

The taxonomy has a number of objectives: it supports design by providing a checklist of components a context-aware infrastructure may need; it supports implementation by distinguishing the set of approaches (and technologies) adopted for the realisation of context-aware functionality; it supports discussion and feedback by providing a common terminology by which components and functions are referred allowing developers to identify synonymous components across projects; it aids researchers who are new to the field, providing an overview of the features that characterise a context-aware infrastructure.

The remainder of the paper is structured as follows. Sections 2 presents related work. In Section 3, we provide an overview of how the taxonomy should be interpreted. Section 4, 5 and 6 describe branches of the taxonomy. In Summary 7 we discuss the taxonomy's contribution and give examples of the conflicting use of terminology that it can reconcile.

2 Related Work

The first survey of the core features of context-awareness was by Pascoe (61) (although Schilit had previously identified classes of context-aware applications (68)). The taxonomies of Pascoe and later Abowd (1) classify individual applications based on their features. The vocabularies presented in these papers described the generic features of context-aware applications but can not be applied to the development of infrastructure-based support for context-awareness.

The survey of context-aware mobile computing by Chen & Kotz (17) investigates context-aware applications in terms of how they sense and model contextual information. From this survey, a discussion of the infrastructure required to support context-aware applications was formed. In contrast, our taxonomy stems from what is now a more mature field. Our taxonomy is not solely concerned with mobile context-aware computing and it does not highlight future research directions but provides a classification of what has been accomplished

to date.

Anagnostopoulos et al. (2) report on both the software architectures and context modeling issues in context-aware computing. Their report includes a "reference model for context-aware system architecture" which is presented as a use case diagram. Each case in the diagram represents a generic class of context-aware functionality. Like our taxonomy, this reference model unifies some of the terminology in use. However, their report is mainly concerned with context modelling and representation issues, whereas our taxonomy covers a wider range of functionality to support context-awareness.

UbiqStack (55) is the first example of a high level taxonomy of context-aware infrastructure components. UbiqStack proposes a convention for labeling classes of building blocks commonly used to construct context-awareness infrastructures. This high level taxonomy has been used as a starting point for our work. We adopt a number of the infrastructure component classes they identified to form categories of functionality in our taxonomy (Registration and Discovery, Context Management & Computation Sharing). However, in contrast to UbiqStack, our taxonomy is not predicated on the belief that complex context-aware infrastructures can be developed by selecting appropriate subsystems from each taxonomy category.

In 2006 Baldauf published a "Survey on Context-Aware Systems" (4). In this survey, a five layer conceptual design framework is presented and used to explain the different elements common to context-aware infrastructures. Where our taxonomy is, as far as possible, implementation independent, Baldauf et al. chooses a ubiquitous computing environment based on Chen's architectural approaches to acquiring context (19) and limits the survey and classification to systems that have adopted similar environment architectures. Our taxonomy is a finer grained analysis of the common elements at a component level and includes infrastructures irrespective of their architecture.

The most recent related work was the "general software infrastructure for ubiquitous computing" by de Costa et al. in 2008 (21). This work shares our goal of "helping the [ubiquitous computing] community develop and assess middleware and frameworks". In contrast to our taxonomy, de Costa et al. identify the challenges in ubiquitous computing. Based on these challenges a set of requirements for the components that should be present in a context-aware infrastructures are defined. Hence, their set of required components is aspirational rather than based on existing infrastructures. As such, it may be used to identify required features in future infrastructures.

This paper provides a more complete picture of the components [2] that may be

[2] We define components to be any general feature of context-awareness infrastructures defined along a functional boundary. These components are referred to as

included in context-aware infrastructures along with a hierarchical taxonomy categorising functionally equivalent components and how they are composed.

3 Overview of the Taxonomy

The taxonomy is the result of our experience in developing a framework for building context-aware mobile applications (27) and an investigation into supporting software infrastructures for context-awareness. The categories of the taxonomy are organised hierarchically. Each category constitutes a distinct branch of the taxonomy, representing a grouping of a common context-related functionality. Branches are subdivided to reflect more specific processes. Rectangular nodes in the taxonomy diagrams represent branches which we further divide, while oval nodes represent leaf nodes and as such group functionality in which we have not identified a decomposition. The order in which the categories of the taxonomy are presented is intended to reflect the context data flow through a typical context-awareness support infrastructure. Each category of the taxonomy is presented in context of its neighbours and includes examples of instances of components in the literature implementing the classified functionality along with any synonyms and implementation options.

The taxonomy divides the processes involved in supporting context-awareness into three categories: *Adaptation, Administration* and *Gathering* (Fig. 1). These categories form a grouping of context-related functionality along high level boundaries. The categories are structured in a data-flow fashion, with control passing from one category to the next as context enters the infrastructure (*Gathering*) and conceptually travels through *Administration* and *Adaptation* processes, eventually leading to application adaptation. *Security & Privacy* (including trust and authentication) is a crosscutting concern that applies to functionality in all three sub categories of context-aware infrastructure. In general, such concerns "are best addressed by each individual component in a way that is meaningful to that component" (55).

3.1 Security & Privacy

Traditional concerns regarding security and privacy are amplified in context-aware applications which are predicated on access to a wide range of sensitive data, and involve ad-hoc collaborations between unknown entities[3]. Privacy

components, modules or layers in the literature.

[3] See Langheinrich (50) for a more thorough characterisation of the privacy problem in ubiquitous computing.

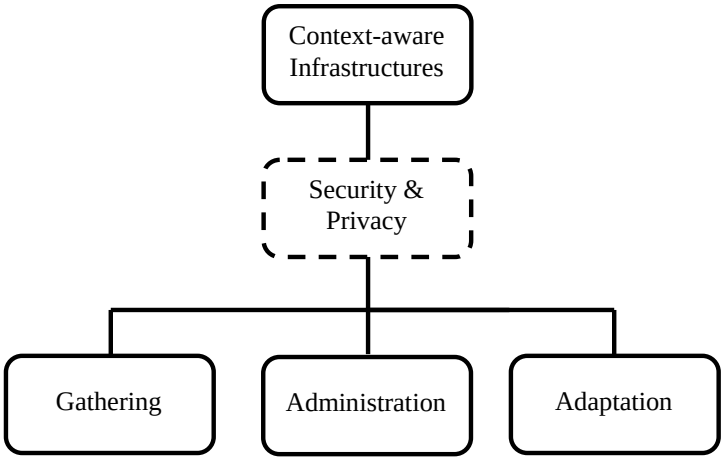

Fig. 1. Context-aware infrastructure taxonomy root.

is reported to be the "most often-cited criticism" of ubiquitous computing (41). Because of this it has been argued that support for privacy constraints must be provided by context-aware infrastructures and not applications (51). However, traditional approaches to security and privacy are too rigid, often depending on the existence of enterprise borders which disappear in the ubiquitous computing world.

Security and Privacy concerns can not be isolated as a single block of functionality within existing context-aware infrastructures. Instead, provisions are made within other components. For this reason, the taxonomy considers Security and Privacy concerns to be crosscutting and as such, illustrates them as an assistant rather than subordinate element in the taxonomy (Fig. 1). Where support for Security & Privacy has appeared in context-aware infrastructures, it has generally been based on addressing one or more of the three topics: Confidentiality, Trust or Identity [4].

Confidentiality: Confidentiality in context-aware infrastructures concerns enabling secure associations to be created between devices in order to protect privacy sensitive data in an untrusted computing environment. The most basic support for confidentiality is to protect the communication channels. Merino (49) features encryption of raw sensor data to support personal security. The use of *physically constrained channels* has also been proposed to provide for confidentiality by securing spontaneous interactions (46; 70). In addition data-centric protection mechanisms [5] based on the application of digital rights management technologies to context elements have been developed (26; 41). These techniques are based on perceiving context information as intellectual property and enforcing privacy and ownership. CoBrA

[4] A study of all the possible security and privacy approaches is beyond the scope of this paper. Yamabe et al. (76) surveys emerging security and privacy enhancing technologies in ubiquitous computing environments.

[5] Also referred to as containment-aware security in literature.

(19) adopts a policy-based approach to protect privacy, allowing users to control the granularity of the information that is shared through the use of a policy ontology.

Trust: One approach being explored in providing decentralised security management to mobile entities is based on the human notion of trust (12). In such a scenario, a formalised human notion of trust (featuring partial information and uncertainty) can be used to design security mechanisms for ubiquitous computing.

Identity: Identity management can provide for trust by enabling access control and authentication (60). Hoffmann (39) proposed a user-centric identity management framework for context-aware mobile services, which makes it possible for the user to control personal data. JCAF (5) uses digital signatures to provide authentication and verification of identity, enabling context access control to be implemented. Solar, Gaia and Context Tailor (18; 67; 51) also implement role-based access control privacy models. Conversely, the absence of any form of identity management has been proposed as a means of protecting privacy (77). Anonymity of users and devices may be provided through pseudonyms (15). IrisNet (56) supported anonymity by actively anonymising streams from sensors to remove any personally identifiable information, for example, detecting and masking peoples faces in video streams.

4 Gathering

A recurring challenge for context-aware applications is to gather information on the state of their environment. This information is required to enable context-aware applications to appropriately adapt to any relevant events in its surrounding environment. *Gathering* is the process of collecting information from the environment. It includes services for the discovery of context sources (peer devices, sensors or network services), the matching of context sources to system needs, and subsequent retrieval of context data.

The taxonomy has a *Gathering* category that may be used to classify components with these responsibilities. The *Gathering* category is further decomposed into: *Discovery & Registration*, *Communication*, *Acquisition* and *Data Formatting*. The complete process of gathering a single piece of context may involve invoking functionality from each of these categories.

Existing context-aware infrastructures provide functionality similar to the *Gathering* class we have defined. The PACE infrastructure (35) for example features a *context gathering layer* with equivalent functionality. This same functionality has been termed *raw data retrieval* in a similar layered conceptual framework by Baldauf et al. (4).

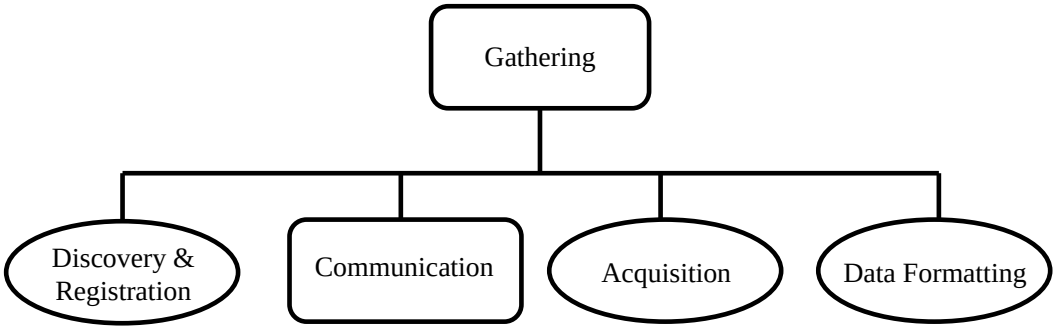

Fig. 2. The Gathering category of the taxonomy

4.1 Discovery & Registration

A problem for many ubiquitous computing systems is how to deal with mobility. This is particularly evident in environments where heterogeneous, distributed mobile devices are communicating using various wireless networking technologies with limited range. This highly dynamic networking environment is characterised by serendipitous connection and disconnection with no guarantee of reconnection with peers. For context-aware applications to gather contextual information, they must meet the challenges of discovering and interacting with the relevant sources of context. Discovery & Registration is the process of finding relevant context sources in a previously unseen environment. It includes both device and service discovery.

When a device enters a ubiquitous computing environment of which it has no prior knowledge, a process on that device must probe the environment for other devices. In the essay by Hong et al. (41), proximity-based discovery is cited as one of the basic context-aware infrastructural services. This functionality is included in Hermes (27), which provides a discovery component that uses IP Multicast to broadcast messages to peers, advertising context information. The Context Toolkit incorporates *discoverer* within its component-based framework (25). Gaia (67) also includes a *discovery service* that uses a similar heartbeat mechanism of sending periodic presence notifications on a device presence channel.

In addition to proximity-based devices discovery, service discovery discovers invokable services. Service discovery is the binding of data and computation services to a network address. These services are then made available to other devices. This approach is taken in NeXus (58), which describes an *area service register* that that servers a particular geographic area. The Strathclyde Context Infrastructure (71) uses a *ranges* metaphor to achieve a similar geographic partitioning of context servers.

Discovery & Registration functionality can also be implemented via a registry-based discovery service, that mediates between service requesters and devices

offering services by caching the service descriptions and bindings. Registry-based discovery protocols such as those implemented by Jini, RMI and CORBA can be used by context-aware infrastructures (14). The Context Toolkit (25) featured a centralised registry at a known network address and port with which all widgets, aggregators, and interpreters register. The registry then pings each registered component at a pre-specified frequency to ensure that each component is functioning correctly. Guide (28) provides a context database that serves as a registry for available context services and classifies services according to context type.

Synonyms: registration & discovery (55), area service register (58), service locating service (34), discovery service (23), range discovery (59), Spontaneous Interaction (69).

4.2 Communication

In order to facilitate communication between peer devices and sensors over various network interfaces (when acquiring context information), context-aware applications require functions that are responsible for the transporting of data between hosts. In particular, applications will be required to connect and transfer data with heterogeneous devices, manage different network interfaces, and deal with the various data formats that are used by different context sources. Communication is the process of establishing a connection for transferring data with context sources in the environment. This includes functionality to transport data, manage addressing, and support various message protocols.

It should be noted that the *Communication* category does not classify possible inter-layer or inter-component communication mechanisms within a context-aware infrastructure. The goal of Communication related functionality is to mask the complexity of interacting with multiple operating systems and network communication stacks.

Infrastructure components with functionality belonging to the Communication category of the taxonomy have appeared in context management systems, labelled with various synonymous terminology. Technology for Enabling Awareness (TEA) (16) includes a *signal layer* which is concerned with maintaining communication channels with sensors. The Cocoa architecture (6) applied the label, *communication drivers layer* to its communication functionality.

The taxonomy further subdivides Communication into three subcategories: Transport, Addressing and Message Protocols (Fig. 3).

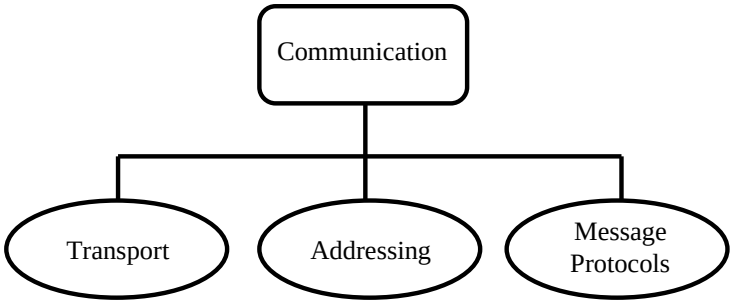

Fig. 3. The Gathering ▷ Communication category of the taxonomy

4.2.1 Transport

Transport classifies components that handle connecting to devices and the transporting of data between devices. This includes the transport of sensor data between sensors, peers, and hosts over short range low bandwidth radio interfaces or over high bandwidth wired links. Transport components are at the heart of any infrastructure's communication functionality and are often implemented in a plugable manner. It is common for context-aware infrastructures to draw on distributed system middleware technologies to supply this functionality.

Transport components are can be either synchronous or asynchronous. Synchronous transport mechanisms deliver data immediately to its recipient (do not store and forward). Examples of synchronous transport mechanisms include distributed object technologies and streaming middlewares. CORBA is extended by the Gaia middleware for active spaces (67) with a *presence service*. A streaming middleware is proposed by UbiqStack (55) as a transport mechanism. RPC technologies such as SOAP are used by Active Campus (33) and Java RMI in JCAF (5).

Asynchronous transport mechanisms deliver messages at irregular intervals and are often forwarded with a time delay. Such transport mechanisms include event-based middlewares [6] and other message passing schemes. STEAM (52) is an event-based communication middleware used by MoCoA (69) and the Sentient Object Model (SOM) (8) to transport context. EasyLiving (11), Context Fabric (Confab) (41), Hermes (27) and Active Campus (33) all use message passing over HTTP to transport context. Guide (20) uses a broadcast-based message protocol to deliver read-only location specific content in a round robin manner to all devices within range of a transmitter.

[6] The event-based communication paradigm provides anonymous, loosely coupled, many-to-many communication between application components via asynchronous event notifications.

4.2.2 Addressing

In an environment where there are many devices communicating over different network interfaces with network addresses that are constantly changing, there is a need to uniquely address a device. IPv6 and MobileIP go some way to tackling this problem in IP networks but the same techniques do not apply to other network interfaces. Also, it is becoming common for devices to have multiple network interfaces, each with their own address. A device addressing mechanism enables devices to be addressed independently of the underlying network. An example is *InConcert*, the communication middleware solution implemented by Easy Living (11), which uses a naming and lookup service to achieve machine independent addressing based on *Instance IDs*. The Context Toolkit (25) uses programming-language independent handles to address individual widgets and Gaia (67) relies on the naming services provided by CORBA to address objects across devices.

4.2.3 Message Protocols

The format in which context is transported is dictated by the messaging protocol bound to the transport middleware in use. The Multi-User Publishing Environment (MUPE) (73) is a multi-user context-aware application platform that features the *context exchange protocol*, an XML-based messaging API. A similar XML-based messaging protocol is adopted in Hermes (27) and HyCon (9). In other cases, messaging protocols are defined to facilitate simple serialisation of context. In such cases the format of messages is tightly coupled to the method used to model the context information (See Section 5.2). Examples include the *context modeling language* of PACE (35), the XML *context tuples* defined by the Confab architecture (41) and the SGML representation in the Stick-e document framework by Brown (10)

4.3 Acquisition

Once context sources are discovered and a communication channel has been established, the next task is to manage the reception of data. Challenges attributed to this task include: providing a uniform interface to receive multiple context types from multiple context sources; eliminating irrelevant sources by semantically matching the types of context desired by the application to the context types supplied by context providers; deciding which set of all discovered context providers to use; deciding on the need for data acquisition, trading off the cost of communicating with context providers against the cost of deriving context; and deciding on the frequency of data acquisition.

Acquisition is the collection of processes involved in receiving data from con-

text sources and forms the third branch of the *Gathering* category of the taxonomy. Acquisition components abstract the details of how context is sensed and hide the low level concerns associated with information retrieval. For Example, the Context Tailor (51) uses a *context service* for managing its acquisition. The context service provides a set of *context drivers* designed to interface with a single type of context source. A *dispatcher* is responsible for matching context types desired by an application with the corresponding *context driver*. Aura has a *contextual information service* (43) that provides an acquisition architecture consisting of a *query synthesizer* that provides clients with the ability to construct rich queries. The aim is efficient acquisition, enabling clients to acquire information from a variety of sources without incurring large communication and computation expenses. The SOCAM (34) and Gaia (64) architectures both provide a set of *context providers* responsible for interacting with context sources. Different context providers abstract the low level details of different context sources. Similarly Dey defines *widgets* and *aggregators* for acquisition functionality (25). Functionally synonymous components in the literature include:

Synonyms: signal layer (16), smart sensor layer (49), sensory capture (8), sensor service (14), context event service (41), context acquisition (6), sensor listeners (29), sensor and terminal layer (9), adaptor layer (38), environment module (62), realm containment modeling layer (26).

4.4 Data Formatting

Crucial in easing the development of context-aware applications is the provision of software components which abstract away from physical device protocols, and support the conversion of data into higher-level symbolic representations. In general, context-aware infrastructures address the challenge of Data Formatting by providing a formatting process that transforms all acquired data into a uniform format, enabling it to be internally processed, manipulated, represented and understood. *Data Formatting* classifies functionality that converts low level information acquired from devices and sensors into an interpretable format.

In some infrastructures, data formatting functions are combined with acquisition processes. CASPEr (26) performs its acquisition and data formatting at its *realm containment modeling layer*. JCAF (5) implements a *context monitor* component to both acquire data and to translate data into Java objects, while the Stick-e document framework (10) provides *SeSensor* components for acquiring and converting of sensor data into its SGML representation. Other infrastructures provide a separate distinct entity dedicated to data format-

ting. The framework developed by Henricksen (35) has a *context reception* layer that converts acquired context into a *fact-based* internal representation. Finally some infrastructures combine data formatting and aggregation functionality (administration category of the taxonomy) into a single function (8; 69; 6; 47; 19; 34; 42; 41). Section 5.1.2 provides more information on reasoning.

Synonyms: context service (51), source nodes (18), senselets (56), acquisition (27), entity modelling (33), citron worker (76), federation layer (57), smart sensor layer (49), measurements layer (37), data layer (9), sensor reporting component (33).

5 Administration

Context-aware applications are likely to receive multiple types of information from a variety sources. Once received, the challenge is to interpret, manipulate, and store this information. *Administration* is the process of managing context information. It includes services for aggregating data, for representing and persisting information in an easily accessible manner and for making computational sharing decisions.

The Administration category of the taxonomy groups all functionality related to the management of context information. This is the second high level category of the taxonomy, residing between Gathering (Section 4) and Adaptation (Section 6). Conceptually, context acquired by Gathering functions are delivered to functions of Administration for storage and manipulation. Context is then accessed by Adaptation functionality for making application behaviour decisions.

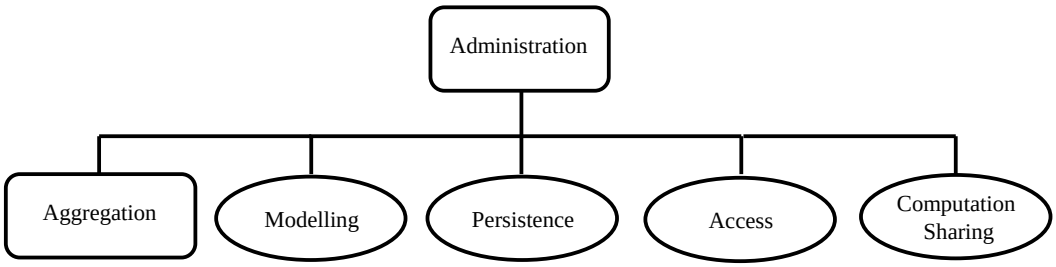

Fig. 4. The Administration category of the taxonomy

The Administration category of the taxonomy (Fig. 4) is further decomposed into five subcategories, all related to the management of context: *Aggregation*, *Modelling*, *Persistence*, *Access* and *Computation Sharing*.

5.1 Aggregation

When data arrives from a context source, it is generally unusable by an application in its external state. Acquired data may be inaccurate, imprecise, ambiguous, or meaningless. The challenge of context-aware applications is to combine and scrutinise acquired information in order to better manage any inherent uncertainty and to interpret information for use in application decision making.

Aggregation is the process of managing uncertainty in acquired context information. It includes services filtering unnecessary information, for interpreting information and for augmenting information with meaningful meta-data. The *Aggregation* category of the taxonomy classifies functionality related to managing context uncertainty. Aggregation has been referred to as sensor or information fusion in literature (8). Fig. 5 shows three sub classifications of *Aggregation*: *Filtering*, *Reasoning* and *Augmentation*. Filtering and Reasoning classify functions that involve processing contextual data and make use of probabilistic reasoning techniques such as Bayesian networks, fuzzy logic, and neural networks. Augmentation classifies processes that involve tagging context with additional, often quality related, information.

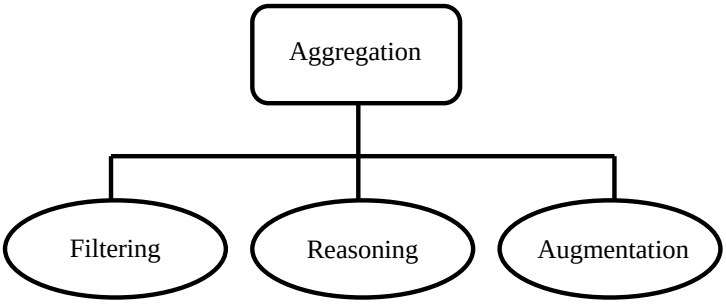

Fig. 5. The Administration ▷ Aggregation category of the taxonomy.

5.1.1 Filtering

Data acquired from context providers such as sensors and devices is inherently noisy and imperfect. Consequently, many frameworks provide a layer of functionality dedicated to addressing this problem. Probabilistic reasoning techniques are generally used for filtering and validating context by identifying redundant and contradictory data. That is, a client may choose to disregard certain context if it exceeds any particular quality thresholds such as accuracy, precision, certainty, or confidence. We term such processes Filtering. Existing infrastructures support various contrasting methods for performing this function. For example, the SOCAM (34) architecture provides a *context interpreter* which contains a set of *context reasoners*, which are responsible for resolving context conflicts and maintaining the consistency of information. The Sentient

Object Model (8) provides a *sensory capture* component that uses Bayesian networks to validate data and handle uncertainty. *Context providers* in Gaia (64) validate context using fuzzy logic or first order logic. The context management infrastructure presented by Anagnostopoulos et al. (3) includes a *filtering process* that merges contextual data based on inter-ontology semantic similarity.

Synonyms: smoothing (69), context inference engine (19), context refiner (42), service-specific-filtering (56), smart sensor layer (49), fusion layer (37).

5.1.2 Reasoning

Data derived from sensors certain sensors can be far removed from the level of abstraction employed by context management. Therefore processing is required to raise its level of abstraction (deduce higher level, more meaningful information by fusing or relating lower level context data). For example, detecting that a meeting is taking place from calendar, location, and noise context. We classify components responsible for this process as Reasoning.

Gaia's *context infrastructure* (64) provides various reasoning techniques. Applications can apply rules written in one of several logic-based formats such as first order logic, temporal logic, description logic, higher order logic, fuzzy logic etc. The architecture also supports machine learning techniques including Bayesian networks, neural networks, and reinforcement learning.The Location Stack (37) provides an *arrangements* and *contextual fusion* layer for reasoning. These layers provides an engine for probabilistically reasoning about the relationships between objects to deduce higher level context details. The *sensory capture* component of the Sentient Object Model (8), and the *fusion service* of MUSE (14) both use Bayesian networks to perform reasoning.

Synonyms: context interpreter (25; 34), automatic path creation (41), context reasoning engine (19), context inference service (23), situation modelling (33), context merging service (47), context transformers and context aggregators (5), transforming or merging (3), context reasoners (54), context abstractors (48), context agents (28).

5.1.3 Augmentation

Context-aware infrastructures will augment acquired context with additional data or meta-information, describing its quality. Meta-information is used by *Adaptation* components (Section 6) when determining the context to use for making decisions. For example, the Context Tailor (51) tags context with a time stamp to indicate data freshness and a confidence value. The *contextual*

information service of Aura (43) supports accuracy, confidence, update time, and sample interval meta-attributes. The *context refiner* of Context Shadow (42) and context exchange protocol of MUPE (73) tag information with a certainty factor and a timestamp to indicate the freshness of data. In HyCon (9) context is augmented with meta-data describing creation and modification timestamp.

5.2 Modelling

"Much of UbiComp relies on having access to a consistent view of the world so that applications might correctly actuate themselves" (55). Context models form the core of any context management infrastructure by facilitating the construction of such a "view of the world". They are critical to the effectiveness of a context-aware application as they structure the context that drives application behaviour. The structure determines not only how context will be stored but also the formats in which it is exchanged, and has implications for semantic interoperability, query languages, context data storage, context interpretation, extensibility and context visualisation.

Modelling classifies the functions that represent context information so that it can be easily stored, accessed and exchanged. Seven classes of context modelling functionality are supported by current context-aware infrastructures. Theses are: Graphical, Logic-based, Ontological, Object-oriented, Tuple-based, Hierarchical and Geometric.

Graphical context models include a diagrammatic representation, specifying the structure of a context model. Unlike other modelling techniques, graphical modelling is a design-time process. As such, a context model that is graphical in nature requires a translation to some other form of representation during development. Henricksen et al. (36) comments that traditional data modelling or object-oriented modelling techniques are "neither natural nor appropriate for describing context". Alternatively they propose a graphical notation referred to as the *Context Modeling Language* (CML) (35; 66).

Logic-based models add context as facts and extract context via expressions or rules. They generally have a high degree of formality and are often tightly coupled to a context reasoning approach (Section 5.1.2). Logic-based models provide context in a form that can be directly operated upon by mathematical expressions. Gaia (67) is an example of a context-awareness infrastructure that implements its reasoning (Section 5.1.2) using first-order logic operations such as quantification, implication, conjunction, disjunction, and context predicate negation. Similarly Gray & Salber (31) propose a *sensed context model* that uses a formal first-order predicate logic representation. These logical represen-

tations facilitate the composition of individual context expressions into more complex sensed context expressions. Context in the Sentient Object Model (8) is represented as a set of discrete logical facts, following a logic-based approach to context representation.

Ontological context models provide a formal specification that captures the semantics of context information. Ontological models (also called semantic context models) have come to be considered the most appropriate method of modelling context (72; 34). The Ambisense context middleware (47) divided context into five subcategories (*task, social, personal, spatio-temporal,* and *environmental*) that were integrated to form a domain ontology. A two level ontology is used to model context in SOCAM (34). The top level of this hierarchy is a generalised ontology while the lower layer contained domain specific ontologies which were divided into several subdomains.

Object-oriented context models define context using object-oriented programming principles: abstraction, inheritance, polymorphism, composition, and aggregation. Object-oriented context models leverage programming language constructs. The resulting unification of the infrastructure's logic and data model development into a single environment can lead to shorter development cycles. Object-oriented context models generally contain objects modelling entities (people, places, objects) that contain context items as attributes or subobjects. This is how JCAF (5), Hydrogen (38), and Mobility and ADaptation enAbling Middleware (MADAM) projects (54) organise their object-oriented context models.

Tuple based context models that are based on a flexible unit of data encapsulation that contains a number of attribute and value mappings and functions as a single record. Tuple-based (alternatively referred to in literature as key-value pair) models facilitate the construction of simple record-based persistence mechanisms and can be queried using template-based queries. Schilit (68) represents context as *dynamic objects* which are collections of key value pairs. The Context Based Storage (44) system includes a *logical context data model* that represents context as tuples using four concepts: *entities, attributes, relationships, and groups*. Tuples have also been chosen to represent context where the context is to be made available to multiple processes via a blackboard-based context access or distribution system. Examples of projects taking a tuple-based modelling approach for this reason include Context Shadow (42), Citron (76) and the Confab infrastructure (41).

Hierarchical context models represent context using a tree-like structure of context types. Hierarchical representations of context facilitate the construction of context containers that implement efficient searching. An example of such a hierarchical model is the context model used by Cocoa (6). Cocoa bases its hierarchy structure on the definition of context categories by Abowd

et al. (1). CASPEr (26) featured a containment-based model of the world. The CASPEr model is built around the concept of a containment tree, which models the world "by nesting containers to represent higher level entities in a structured manner". The Context Toolkit (25) introduced hierarchical categories of context information: identity, location, status and time. These basic categories were chosen to facilitate simple inference, or derivation, of context information from a single known piece of context information.

Geometric context models represent context as attributes of physical entities that have a dimension. The physical entities are represented by a set of vector coordinates. The vector coordinates may be organised into shapes such as point, line, and area features and attribute records may be associated with individual shapes. Such a model is appropriate where a detailed model of the real world is important. Geometric models facilitate geometric operations such as overlap, intersection, containment and distance, which can be used as part of a spatial reasoning process to determine higher level contexts or trigger behaviour. Guide (20) makes use of a geometric model that supports route guidance and provides information about specific physical locations. EasyLiving (11) from Microsoft Research models its context with the *EasyLiving geometric model* in which all entities were modeled as *extents*.

These modelling categories are not mutually exclusive. It is common for context management systems with an object-oriented or ontological model to also have a graphical representation of the model associated with them. Similarly geometric and ontological models can be represented as objects. In addition to these classifications of modelling approaches, a small number of projects apply informal and unstructured models to context representation. An example is CoolTown (45), which models context as web pages intended for human consumption. Yet others are missing a model entirely, representing context simply as raw sensor data (56; 74; 37).

5.3 Persistence

Context-aware applications often rely on historical information when making decisions. Storing information for future retrieval and ensuring appropriate data replication for fault tolerance are important context-related challenges.

Persistence is the process of storing context in non-volatile memory, in a manner that maintains the semantics it is afforded by the context model (Section 5.2), and in a way that facilitates efficient retrieval. *Persistence* functions may structure data differently to Modelling for reasons of performance and generally exist as a separate category in context-aware infrastructures. Persistence functions are generally tightly coupled to context models. The choice of a con-

text model often determines the appropriate data persistence mechanism. Like Modelling, functionality classified by the Persistence category of the taxonomy often have a strong influence on the functionality that the taxonomy classifies under Access (Section 5.4). This is because database systems are often chosen to implement Persistence components. The query languages and APIs native to these systems are exploited directly or extended to provide access to context. Persistence functionality is implemented by components referred to in literature as *context repositories*, *model repositories* (3) or *containers* (53). There are two common approaches to persistence that are taken by existing infrastructures: Centralised and Distributed.

Centralised persistence stores context on a single host, usually on the device that generated or gathered the context. Centralised context persistence is generally achieved via a database management system or bespoke record-based storage mechanism. Both spatial and relational database management systems have been used, often in a plugable manner, where individual database providers can be substituted. MiddleWhere (63) uses a spatial database to persist contextual information. PACE (35), Active Campus (33) and CASS (29) store context, application and user data, domain knowledge, and behaviour in a relational database. HyCon (9) implements a *data layer* responsible for converting its object-oriented data model for persistence in a relational database at the *storage layer*. SOCAM (34) includes a *context database,* a relational database, in which ontological knowledge and context information is stored. The Context Toolkit's (25) *widgets* include such a plugable storage mechanism.

Distributed persistence copies context data to multiple distributed devices. In this case the context is stored or replicated on many devices and is available to the context-aware infrastructure components on each device. This is generally achieved through some form of blackboard-based communication system or distributed database. Distributed context persistence is appropriate where there are no privacy issues with sharing context between devices and helps an infrastructure achieve scalability, robustness and availability. Distributed context persistence can be leveraged at a higher level to facilitate the collaboration of devices through context sharing. Distributed databases are a common approach to providing distributed persistence. CAPNET's *context based storage* (44) is a distributed physical data store. Gaia (67) includes a *context file system* implemented as a hybrid database and file system with an architecture composed of mount and file servers. Another method of distributed persistence is the blackboard-based approach. This approach behaves like a distributed shared memory with a flexible data representation. Processes running on all devices have shared access to the context. Citron (76) uses a tuple space as its blackboard-based data model to store context. The Context Shadow system (42) implements its blackboard-based data store using *TSpaces*. Functionality synonymous with Persistence support in the literature include:

Synonyms: context persistence service (23), spatial model server (58), context repository (49), context based storage (44), storage layer (9), model repositories (3), containers (53).

5.4 Access

Much of the success of context-aware applications is owed to how fast they can react to changes in the environment. Prompt retrieval of necessary information is therefore critical to their successful operation.

Access is the process of retrieving context information, either directly from sensors, from a context model representation, or from persistent storage. Many infrastructures provide a uniform method for accessing context, regardless of their representation structure or storage location. The Access category of the taxonomy classifies the context-aware infrastructure components that are responsible for providing applications and other infrastructural components access to context. There are three common approaches to implementing *Access* functionality by existing context-aware infrastructures: Unique Identifiers, Programmatic Routines and Query Languages.

Unique Identifiers are used to access context where context elements can be located by an associated identifier. Merino (49) provides access to objects on their local and distributed context servers using a *URN*. In Solar (18), context is stored as a set of directories and is accessed using a sequence of context labels from the root directory. Gaia (67) and CAPNET (44) also use logical paths to access context and enable context to be accessed based on specific context attributes similar to the SQL *where* clause.

Programmatic routines provide a set of predefined procedural calls for context access, usually in the form of an API, masking details about how data is accessed from the user. These routines are tied to a particular programming language. For example, JCAF (5) provides a Java event-based API, enabling developers to use *EntityListeners* as hooks for extending the frameworks functionality. Components register themselves with the *ContextService* as *EntityListeners*. A similar approach is supported in the Context Toolkit (25) for accessing context from *aggregators* and *interpreters* with a *sendToSubscribers()* function sending new context to subscribing parties.

Query Languages in contrast provide specifically designed languages for accessing context. Unlike Programmatic Routines, Query Language access requires the developer to have some knowledge about the structure of stored data. However they do facilitate the decoupling of context access from the programming language and internal data representation. The type of Persistence mechanisms that are used by infrastructures will have a defining influence on how

context is accessed, and this is particularly true for Query Language Access. CASS (29), PACE (35) and Active Campus (33) store context in relational databases and hence use SQL as the query language to access and manipulate information. The Contory middleware features a SQL-like context query language (65). NeXus (57) provides an *Augmented World Model Query Language* (AWQL) to access context from its *augmented world model*. Access to data in CASPEr (26) is through operations supported by the *data analyser* and *type warden* components implemented in its XML storage subsystem. Standard XML tools such as DOM, SAX, or XPath are supported for data access.

Synonyms: context query service (23), Structured I/O (32), context service (51), environment module (62), context management (27).

5.5 Computation Sharing

Many ubiquitous computing applications contain mobile components. Mobile, context-aware applications are limited by processing and battery power of the devices they run on. To overcome this problem there is growing support for infrastructure functionality designed to improve the user experience of context-aware applications without adversely effecting the battery life of mobile devices. This is achieved by sharing or offloading certain computation processes to other devices.

Computation Sharing is the process of delegating computation to other devices. The Computation Sharing category of the taxonomy classifies functions that enable applications to make use of remote computational resources, enabling ubiquitous computing applications adapt to the dynamism of resources on mobile devices and the mobility of the user. There are two main approaches supported by some current context-aware infrastructures for Computation Sharing: Proxying and Migration.

Proxying delegates computation tasks to other devices which perform them on their behalf and return the results. For example, Aura (30) terms its computation sharing functionality *cyber foraging*. Aura's cyber foraging environment can make use of servers located near the client device, called *surrogates*, to assist in its tasks. It is a remote execution environment designed for pervasive applications that monitors four resources: *CPU*, *network*, *battery*, and *cache state*; and uses this information to decide how to optimise device performance. The Strathclyde Context Infrastructure (71) takes advantage of its peer-to-peer environment by using peer resources as proxies for processing context information.

Migration, in contrast, is a mechanism of moving computation to another operating environments where processes are long-lived and results are not ex-

pected to be returned. The *migration* service in One.world (32) is an example of computation sharing functionality we term Migration. One.world provides the ability to move or copy an environment and its contents to another device. Migration makes use of *serialisation* to capture an applications environment and execution state. *Deserialisation* occurs on the target device with the environment and execution state restored. CARMEN (7) supports migration of processes when mobile entities move with its *shadow proxy*.

6 Adaptation

Weiser's vision for ubiquitous computing is a future where computers are widespread and integrated with the environment (75). For this vision to be realised applications need the ability to automatically adapt to changes in an environment. The nature of a dynamic environment means that context-aware applications are required to function under a myriad of different situations and conditions and adapt accordingly to any changes.

Adaptation is the process of altering the behaviour of the application in response to relevant context changes in the environment. Many of the surveyed context-aware infrastructures only provide functions for Gathering and Administration with Adaptation functionality delegated to the application itself (25; 16; 43; 38; 19; 5; 42; 56; 76; 49; 33; 58). Infrastructures that do provide Adaptation functionality usually do so either by providing support for Rules or for Machine Learning algorithms.

Rules support enables applications to define set of context to behaviour mappings indicating behaviour that should occur in a given context. based on event-condition-action model (24). Rules are specified by developers during the implementation phase, and application behaviour is restricted to actions specified in rules. The Context Tailor (22) provides an API containing *contextual*, *learning classification*, *temporal*, and *statistical* components for specifying rules, termed *patterns*. The *pattern activator* matches context retrieved from the *context service* with conditions stated in patterns retrieved from the *pattern repository* to determine the actions to trigger. In the framework by Korpipää et al. (48), a *rule script engine* defines rules in XML format. The engine matches the condition clauses in rules with the current context retrieved from the *context manager*. CASPEr (26) provides a policy model for its adaptation. It uses the *ponder* policy language for specifying its policy rules. The policy editor, policy specification interface, and policy manager, TFFST module and policy deployment module are components of the framework dedicated to managing policies and adaptation. The CARISMA infrastructure provides adaptation support in the presence of conflicting rules (13). It supports a *reflection* based adaptation process that enables *intra-profile* and *inter-profile*

rule conflicts to be resolved.

Machine Learning provides support for probabilistic techniques that enable the application to be trained, learning how to behave in different situations. Ambisense (47) uses the case-based reasoning for adaptation. The premise of case-based reasoning is to retrieve a known context or case to decide what action to take in the current context. Cases along with related context and solution information will be stored or classified for future retrieval and comparisons. Ambisense uses a two-tier reasoning mechanism: *on-line* and *off-line*. On-line reasoning will classify new cases on the user's mobile device while off-line reasoning makes use of persistent storage on the user's home system to classify and group common cases. Gaia (64) also supports machine learning-based adaptation. Bayesian approaches, neural networks, various clustering algorithms, and reinforcement learning are supported so that applications can automatically adapt themselves to changing situations. Learning can be performed offline with training examples or online as the application is in operation.

Functionality synonymous with Rule based adaptation support in the literature include:

Synonyms: inference engine (29), trails (27), context-aware mobile services (34), policy (3), adaptation control (28), adaptation manager (54), context scripts (73), migration policies (7), intentions (37).

7 Summary

In this paper we have presented a taxonomy that provides a generalisation of the capabilities of current context-aware infrastructures in order to aid future research, design, and discussion of context-aware systems. Through the taxonomies categories, an overview of the features that characterise a context-aware infrastructure is provided. The paper provides a classification of existing context-aware infrastructures using the taxonomy. This classification provides a clearer understanding of different infrastructures, the components and functionality they support, and where components and processes are located. This information can be used to help identify limitations of existing approaches and make decisions on where it may be appropriate to reuse approaches in the future. The nomenclature of terminology used to describe the features that characterise middleware support for context-awareness has been provided. We have shown there to be significant problems in the way infrastructures use terminology. For example, what one author refers to as "context provisioning" (65), another calls "context service" (51). Yet another author uses the term "context management" (27) which naively seems similar but in this case is used

to label a system that is responsible solely for what the taxonomy classifies as *Administration*. Similarly, the "adaptor layer" (38) in the hydrogen project provided *Acquisition* functionality, whereas, the "adaptation manager" (54) in Mikalsen et al. provides *Adaptation* functionality. The taxonomy allows the identifications of synonymous components across projects, supporting the design of context-aware systems and providing a convenient way to compare and contrast middleware support for context-awareness.

Acknowledgements

This work is supported in part by the Irish Research Council for Science, Engineering and Technology (IRCSET) and Intel Corporation.

References

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, P. Steggles, Towards a better understanding of context and context-awareness, in: HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, Springer-Verlag, London, UK, 1999.

[2] C. Anagnostopoulos, A. Tsounis, S. Hadjiefthymiades, Context awareness in mobile computing: A survey, in: Mobile HCI '04, Mobile and Ubiquitous Information Access, 2004.

[3] C. Anagnostopoulos, A. Tsounis, S. Hadjiefthymiades, Context management in pervasive computing environments, in: International Conference on Pervasive Services, 2005. ICPS '05., 2005.

[4] M. Baldauf, S. Dustdar, F. Rosenberg, A survey on context-aware systems, International Journal of Ad Hoc and Ubiquitous Computing.

[5] J. E. Bardram, The java context awareness framework (jcaf) - a service infrastructure and programming framework for context-aware applications., in: H.-W. Gellersen, R. Want, A. Schmidt (eds.), Pervasive, vol. 3468 of Lecture Notes in Computer Science, Springer, 2005.

[6] P. Barron, V. Cahill, Using stigmergy to co-ordinate pervasive computing environments, in: WMCSA '04: Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'04), IEEE Computer Society, Washington, DC, USA, 2004.

[7] P. Bellavista, A. Corradi, R. Montanari, C. Stefanelli, Context-aware middleware for resource management in the wireless internet, IEEE Transactions Software Engineering 29 (12).

[8] G. Biegel, V. Cahill, A framework for developing mobile, context-aware applications, in: PERCOM '04: Proceedings of the Second IEEE Inter-

national Conference on Pervasive Computing and Communications (PerCom'04), IEEE Computer Society, Washington DC, USA, 2004.

[9] N. O. Bouvin, B. G. Christensen, K. Grønbæk, F. A. Hansen, HyCon: a framework for context-aware mobile hypermedia, Hypermedia 9 (1) (2003) 59–88.

[10] P. J. Brown, The stick-e document: A framework for creating context-aware applications, in: In Proceedings of Electronic Publishing, vol. 8, 1996.

[11] B. Brumitt, B. Meyers, J. Krumm, A. Kern, S. A. Shafer, Easyliving: Technologies for intelligent environments, in: HUC, 2000.

[12] V. Cahill, E. Gray, J.-M. Seigneur, C. D. Jensen, Y. Chen, B. Shand, N. Dimmock, A. Twigg, J. Bacon, C. English, W. Wagealla, S. Terzis, P. Nixon, G. di Marzo Serugendo, C. Bryce, M. Carbone, K. Krukow, M. Nielsen, Using trust for secure collaboration in uncertain environments, IEEE Pervasive Computing 02 (3) (2003) 52–61.

[13] L. Capra, W. Emmerich, C. Mascolo, Carisma: Context-aware reflective middleware system for mobile applications, IEEE Transactions on Software Engineering 29 (10) (2003) 929–945.

[14] P. Castro, R. Munz, Managing context data for smart spaces, IEEE Personal Communications 7 (5) (2000) 44–46.

[15] C. Chatfield, R. Hexel, User identity and ubiquitous computing: User selected pseudonyms, in: Workshop on UbiComp Privacy, The Seventh International Conference on Ubiquitous Computing (Ubicomp 05), 2005.

[16] D. Chen, A. Schmidt, H.-W. Gellesen, An architecture for multi-sensor fusion in mobile environments, in: Proceedings International Conference on Information Fusion, Sunnyvale, CA, USA, 1999.

[17] G. Chen, D. Kotz, A survey of context-aware mobile computing research, Tech. Rep. TR2000-381, Dartmouth College, Hanover, NH, USA (2000).

[18] G. Chen, D. Kotz, Solar: Towards a flexible and scalable data-fusion infrastructure for ubiquitous computing, In Workshop on Application Models and Programming Tools for Ubiquitous Computing at the Third International Conference on Ubiquitous Computing (UbiComp 2001), 2001.

[19] H. Chen, T. Finin, A. Joshi, An intelligent broker architecture for context-aware systems, in: Adjunct Proceedings of Ubicomp 2003, Seattle, Washington, USA, 2003.

[20] K. Cheverst, N. Davies, K. Mitchell, A. Friday, Experiences of developing and deploying a context-aware tourist guide: the guide project, in: MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking, ACM Press, New York, NY, USA, 2000.

[21] C. A. da Costa, A. C. Yamin, C. F. R. Geyer, Toward a general software infrastructure for ubiquitous computing, IEEE Pervasive Computing 7 (1) (2008) 64–73.

[22] J. S. Davis, D. M. Sow, M. Blount, M. R. Ebling, Context tailor: Towards a programming model for context-aware computing, 1st International

ACM Workshop on Middleware for Pervasive and Ad-Hoc Computing, 2003.

[23] R. de Freitas Bulcao Neto, M. da Graca Campos Pimentel, Toward a domain-independent semantic model for context-aware computing, in: LA-WEB '05: Proceedings of the Third Latin American Web Congress, IEEE Computer Society, Washington, DC, USA, 2005.

[24] D. L. de Ipia, An eca rule-matching service for simpler development of reactive applications, Middleware 2001 vol. 2 no. 7.

[25] A. K. Dey, D. Salber, G. D. Abowd, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, Human-Computer Interaction (HCI) Journal 16 (2-4) (2001) 97–16.

[26] B. Dragovic, Casper: Containment-aware security for pervasive computing environments, Doctor of philosophy, St John's College, University of Cambridge (March 2006).

[27] C. Driver, E. Linehan, S. Clarke, A framework for mobile, context-aware trails-based applications: Experiences with an application-led approach, in: Workshop 1 ("What Makes for Good Application-led Research in Ubiquitous Computing?"), Pervasive 05, 2005.

[28] C. Efstratiou, K. Cheverst, N. Davies, A. Friday, An architecture for the effective support of adaptive context-aware applications, in: MDM '01: Proceedings of the Second International Conference on Mobile Data Management, Springer-Verlag, London, UK, 2001.

[29] P. Fahy, S. Clarke, Cass: Middleware for mobile, context-aware applications, in: Workshop on Context Awareness at MobiSys, 2004.

[30] D. Garlan, D. Siewiorek, A. Smailagic, P. Steenkiste, Project aura: Toward distraction-free pervasive computing, IEEE Pervasive Computing 1 (2) (2002) 22–31.

[31] P. D. Gray, D. Salber, Modelling and using sensed context information in the design of interactive applications, in: EHCI '01: Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction, Springer-Verlag, London, UK, 2001.

[32] R. Grimm, J. Davis, E. Lemar, A. MacBeth, S. Swanson, T. A. Steven Gribble, B. Bershad, G. Borriello, D. Wetherall, Programming for pervasive computing environments, Tech. Rep. UW-CSE-01-06-01, University of Washington Department of Computer Science and Engineering, Seattle, Washington, USA (2001).

[33] W. G. Griswold, R. Boyer, S. W. Brown, T. M. Truong, A component architecture for an extensible, highly integrated context-aware computing infrastructure, in: ICSE '03: Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society, Washington, DC, USA, 2003.

[34] T. Gu, H. Pung, D. Zhang, A middleware for building context-aware mobile services, in: IEEE Vehicular Technology Conference, Milan, Italy, 2004.

[35] K. Henricksen, J. Indulska, A software engineering framework for context-

aware pervasive computing, in: PERCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04), IEEE Computer Society, Washington, DC, USA, 2004.

[36] K. Henricksen, J. Indulska, A. Rakotonirainy, Modeling context information in pervasive computing systems, in: Pervasive '02: Proceedings of the First International Conference on Pervasive Computing, Springer-Verlag, London, UK, 2002.

[37] J. Hightower, B. Brumitt, G. Borriello, The location stack: A layered model for location in ubiquitous computing, in: WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications, IEEE Computer Society, Washington, DC, USA, 2002.

[38] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, W. Retschitzegger, Context-awareness on mobile devices - the hydrogen approach, in: HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9, IEEE Computer Society, Washington, DC, USA, 2003.

[39] M. Hoffmann, User-centric identity management in open mobile environments, in: Workshop on Security and Privacy in Pervasive Computing, Second International Conference on Pervasive Computing (Pervasive 04), Vienna, Austria, 2004.

[40] J. I. Hong, J. A. Landay, An infrastructure approach to context-aware computing, in: In Human-Computer Interaction, vol. 16, 2001.

[41] J. I. Hong, J. A. Landay, An architecture for privacy-sensitive ubiquitous computing, in: MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services, ACM Press, New York, NY, USA, 2004.

[42] M. Jonsson, P. Werle, C. G. Jansson, Context shadow: An infrastructure for context aware computing, in: Proceedings of Artificial Intelligence in Mobile System, 2003.

[43] G. Judd, P. Steenkiste, Providing contextual information to pervasive computing applications, in: PERCOM '03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, IEEE Computer Society, Washington, DC, USA, 2003.

[44] S. Khungar, J. Riekki, A context based storage system for mobile computing applications, SIGMOBILE Mob. Comput. Commun. Rev. 9 (1) (2005) 64–68.

[45] T. Kindberg, J. Barton, A web-based nomadic computing system, Comput. Networks 35 (4) (2001) 443–456.

[46] T. Kindberg, K. Zhang, Information Security, vol. 2851/2003 of Lecture Notes in Computer Science, chap. Validating and Securing Spontaneous Associations between Wireless Devices, Springer Berlin / Heidelberg, 2003, pp. 44–53.

[47] A. Kofod-Petersen, M. Mikalsen, Context: Representation and Reasoning – Representing and Reasoning about Context in a Mobile Environment,

Revue d'Intelligence Artificielle 19 (3) (2005) 479–498.

[48] P. Korpipää, E.-J. Malm, I. Salminen, T. Rantakokko, V. Kyllönen, I. Känsälä, Context management for end user development of context-aware applications, in: MDM '05: Proceedings of the 6th international conference on Mobile data management, ACM Press, New York, NY, USA, 2005.

[49] B. Kummerfeld, A. Quigley, C. Johnson, R. Hexel, Merino: Towards an intelligent environment architecture for multi-granularity context description, Workshop on User Modeling for Ubiquitous Computing, Pittsburgh, USA, 2003.

[50] M. Langheinrich, Personal privacy in ubiquitous computing – tools and system support, Ph.D. thesis, ETH Zurich, Zurich, Switzerland (May 2005).

[51] H. Lei, D. M. Sow, I. John S. Davis, G. Banavar, M. R. Ebling, The design and applications of a context service, ACM SIGMOBILE Mobile Computing and Communications Review 6 (4) (2002) 45–55.

[52] R. Meier, V. Cahill, Exploiting proximity in event-based middleware for collaborative mobile applications., in: DAIS, 2003.

[53] F. Meneses, Context management for heterogeneous administrative domains, in: PERVASIVE, Second International Conference on Pervasive Computing, Linz / Vienna, Austria, 2004.

[54] M. Mikalsen, J. Floch, N. Paspallis, G. A. Papadopoulos, P. A. Ruiz, Putting context in context: The role and design of context management in a mobility and adaptation enabling middleware, 7th International Conference on Mobile Data Management (MDM'06) 0 (2006) 76.

[55] M. Modahl, B. Agarwalla, S. Saponas, G. Abowd, U. Ramachandran, Ubiqstack: A taxonomy for a ubiquitous computing software stack, vol. 10, Springer-Verlag, London, UK, 2005.

[56] S. Nath, Y. Ke, P. B. Gibbons, B. Karp, S. Seshan, Irisnet: An architecture for enabling sensor-enriched internet service, Tech. Rep. IRP-TR-03-04, Intel Research Pittsburgh (June 2003).

[57] D. Nicklas, M. Bernhard, On building location-aware applications using an open platform based on the nexus augmented world model, Software and Systems Modeling 3 (4).

[58] D. Nicklas, M. Großmann, T. Schwarz, S. Volz, B. Mitschang, A model-based, open architecture for mobile, spatially aware applications, in: SSTD '01: Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases, Springer-Verlag, London, UK, 2001.

[59] P. Nixon, F. Wang, S. Terzis, Programming structures for adaptive ambient systems, in: ISICT '03: Proceedings of the 1st international symposium on Information and communication technologies, Trinity College Dublin, 2003.

[60] P. Osbakk, N. Ryan, A Privacy Enhancing Infrastructure for Context-Awareness, UK-UbiNet webpage, position Paper for the 1st UK-UbiNet Workshop, Imperial College, London, UK. (September 2003).

[61] J. Pascoe, Adding generic contextual capabilities to wearable computers, in: ISWC '98: Proceedings of the 2nd IEEE International Symposium on Wearable Computers, IEEE Computer Society, Washington, DC, USA, 1998.

[62] J. Pascoe, N. Ryan, D. Morse, Issues in developing context-aware computing, in: HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, Springer-Verlag, London, UK, 1999.

[63] A. Ranganathan, J. Al-Muhtadi, S. Chetan, R. Campbell, M. D. Mickunas, Middlewhere: A middleware for location awareness in ubiquitous computing applications, in: Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware, Springer-Verlag New York, Inc., New York, NY, USA, 2004.

[64] A. Ranganathan, R. H. Campbell, A middleware for context-aware agents in ubiquitous computing environments, in: ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil, 2003.

[65] O. Riva, Contory: A middleware for the provisioning of context information on smart phones, Middleware 2006 (2006) 219–239.

[66] R. Robinson, K. Henricksen, J. Indulska, Xcml: A runtime representation for the context modelling language, in: PERCOMW '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops, IEEE Computer Society, Washington, DC, USA, 2007.

[67] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, K. Nahrstedt, A middleware infrastructure for active spaces, vol. 1, IEEE Educational Activities Department, Piscataway, NJ, USA, 2002.

[68] B. N. Schilit, A context-aware system architecture for mobile distributed computing, Ph.D. thesis, Columbia University (May 1995).

[69] A. Senart, R. Cunningham, M. Bouroche, N. O'Connor, V. Reynolds, V. Cahill, Mocoa: Customisable middleware for context-aware mobile applications, in: Distributed Object Applications 2006, 2006.

[70] F. Stajano, R. J. Anderson, The resurrecting duckling: Security issues for ad-hoc wireless networks, in: Proceedings of the 7th International Workshop on Security Protocols, Springer-Verlag, London, UK, 2000.

[71] G. Stevenson, P. Nixon, R. I. Ferguson, A general purpose programming framework for ubiquitous computing environments, Ubisys: System Support for Ubiquitous Computing Workshop, UbiComp, Seattle, Washington., 2003.

[72] T. Strang, C. Linnhoff-Popien, A context modeling survey, in: Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England, 2004.

[73] R. Suomela, E. Rasanen, A. Koivisto, Mupe context programming manaual, Online (September 2006).

[74] R. Want, A. Hopper, V. Falcão, J. Gibbons, The active badge location

system, ACM Trans. Inf. Syst. 10 (1) (1992) 91–102.
[75] M. Weiser, Ubiquitous computing, Computer 26 (10) (1993) 71–72.
[76] T. Yamabe, A. Takagi, T. Nakajima, Citron: A context information acquisition framework for personal devices, in: RTCSA '05: Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05), IEEE Computer Society, Washington, DC, USA, 2005.
[77] A. Zugenmaier, A. Hohl, Anonymity for users of ubiquitous computing, in: 2nd Workshop on Security in Ubiquitous Computing, Ubicomp 03, Seattle, Washington, USA, 2003.