Problems in Context-Aware Semantic Computing

doi:10.3991/ijxx.vxix.xxx (Please do not delete this line)

O.A. Nykänen[1] and A. Rivero Rodriguez[1]
[1] Tampere University of Technology, Tampere, Finland

***Abstract*—Acknowledging the user context, e.g., position and activity, provides a natural way to adapt applications according to the user needs. How to actually capture and exploit context, however, is not self-evident and it is tempting to assign the related responsibilities to individual context-consuming applications. Unfortunately, this confuses the user, complicates application development and hinders context-aware semantic computing as a research discipline. In this article, we outline context-aware semantic computing research topics and the state-of-the-art mobile application development frameworks of special interest to us, acknowledging best practices for accessing and modeling sensor context. From the integrated point of view, context-aware semantic computing is demonstrated in terms of a software component called context engine. In order to better understand how theory is tied with practice, we also introduce a simple context engine prototype. Finally, we use the research background and the empirical setting to discuss the significant problems and relevant research directions in context-aware semantic processing.**

***Index Terms*—Context Engine, Context-Aware Services, Mobile Computing, Semantic Computing**

I. INTRODUCTION

Acknowledging the user context provides a natural way to focus user attention and the use of resources in applications. For instance in mobile applications, user position, time, calendar activity, and task establish a convenient starting point for filtering, organizing, and providing access to relevant information and tools.

Pioneering research in context-aware computing research dates back to early 1990s [7]. Since then, studying and building context-aware systems have been first tackled in application-specific manner, then in terms of reusable toolkits, and finally, on infrastructure level [19]. For various reasons, however, the rate of infrastructure-level deployment and adoption in production systems is still catching up. Significant technological progress has been made, in particularly in mobile applications [34]. Simply looking at application volumes, it is fair to say that contemporary mobile sensor frameworks establish the de facto technology driver for context-aware computing.

Still, despite the research results and technological advancements, it is not self-evident how context should be realized and what is the role of sophisticated context-aware computing in the application ecosystem(s). From the perspective of context-aware computing, the current sensor APIs and frameworks provide rather low-level access to sensor information, which in practice suggests that each application deals with the context as it sees fit. This confuses users and hinders the development of more abstract context-aware computing.

A modern reincarnation of the middleware for context-aware computing is a system called a context engine. In brief, the main task of a context engine is to filter and refine the contextual clues, e.g., for recommendation applications [30].

Through this notion of the context engine, context-aware computing gets closely affiliated with a multidisciplinary research topic called semantic computing [39]. In brief, semantic computing is about computing with (machine processable) descriptions of content and (user) intentions [22]. Aligned with Semantic Web technologies [46], this provides the methodological and technological baseline for modeling, understanding, and computing with the user context (cf. e.g. [43]). Significant research problems, however, need still to be properly addressed before the promise of context-aware semantic computing can be fulfilled.

In this article, we outline context-aware semantic computing research topics and the state-of-the-art mobile application development frameworks of special interest to us, acknowledging best practices for accessing and modeling sensor context. From the integrated point of view, context-aware semantic computing is demonstrated in terms of a software component called context engine. In order to better understand how theory is tied with practice, we also introduce a simple context engine prototype. Finally, we use the research background and the empirical setting to discuss the significant open problems and relevant research directions in context-aware semantic processing.

The main contribution of this article is to review the related sensor and context modeling research in order to systematically characterize the role of context-aware semantic computing in (mobile) applications, and to use this setting to discuss the related significant research and engineering questions.

Considering (our) future research, we believe that context-aware semantic computing will have an increasingly significant impact in application development. In addition to mainstream mobile computing, perhaps two of the most prominent application areas with a large volume of industrial applications include Web of things and the Industrial Internet paradigm [48] [14].

Our current work stems from the ongoing Marie Curie ITN research project MULTI-POS, Multi-technology positioning professionals (Grant agreement no. 316528, 2012-2016) where we study context-aware semantic processing.

The rest of this article is organized as follows: in Section 2, we outline the background of our work, highlighting the current technology driver and the best practices for modeling context. In Section 3, we present context engine architecture and a simple prototype implementation. Equipped with the research background and implementation experience, we then discuss the related open research and engineering questions in Section 4. Finally, in Section 5 we conclude the article.

II. BACKGROUND

Context can refer to any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object [6].

Contextual information may include physical information such as accelerometer data, virtual information such as calendar events, recognized patterns such as observed user activities, and predictions such as weather forecasts. In the abstract sense, context can be used to reduce the computational complexity of problem solving by restricting the search space – in turn decreasing the number of irrelevant end user choices.

A. Related Research and Basic Concepts

The term context-aware (computing) appeared first time in early 1990s, with the beginning of context-aware system research [7]. In addition to solely computing with respect to time and place, context-aware systems can capture many other things as well, such as places, things, commitments, and user knowledge and preferences [30]. A typical application area is context-aware search, which includes the phases of data acquisition, context reasoning and state updates, and contextualized output [44].

The main components of a context-aware system include context providers and context-aware services, perhaps associated with service locating services or brokers [19]. In applications, the computing context, the user context, and the physical context are often differentiated [7]. Processing contextual information is carried out by a component called context interpreter, and the relevant data is stored in a context database. The basic activities include context assertion, i.e. making contextual information available, and context retrieval, i.e. exploiting the context in an application [30]. Reasoning with the context is typically based on logic programming [5][20].

In brief, we may identify three complementary approaches on how the context providers acquire contextual information [7][35][8]:

- *Direct sensor access*, where sensor information is directly read from the sensor APIs.

- *Middleware infrastructure*, which introduces a layered architecture that enhances reusability and provides concurrent sensor access. Instead of accessing directly the raw data from sensors, an intermediate software layer manages sensorial data.

- *Context server*, which in addition allows gathering information from remote data sources and distributing the costs of measurements and computations.

In any case, direct sensor access is not usually feasible since sensor access needs to be encapsulated for multi-tasking, concurrency etc.

In principle, context-computing tasks may be delegated to a software component called context engine [30]. For purposes of this article, we say that a *context engine* is a software component, which integrates and refines the generalized (sensor) context, the related services, and the user preferences, for the benefit of individual (user) applications. Note that the term context broker is sometimes used for a similar architecture [8].

Typical tasks of a context engine include acting as a local context provider, providing logical context interpretation, accessing external context providing services, and managing an archived sensor information database e.g. for minimizing battery consumption and user preferences. Note that these tasks typically exceed the boundaries of individual applications.

Acknowledging the close relationship between context and sensor information, the notion of a "sensor" is typically generalized. We may acknowledge at least three different types of sensors providing contextual data [4]:

- *Physical sensors* are the most frequently used sensors, capable of capturing physical data (e.g. position, orientation, and acceleration).

- *Virtual sensors* provide contextual information from applications and services. Virtual sensors may further be based on local or external data sources (e.g. user calendar vs. weather service).

- *Logical sensors* provide new contextual information by combining and computing information from physical and virtual sensors.

Considering past research, known context-aware frameworks and systems include Context Broker Architecture (CoBrA), Context-Awareness Sub-Structure (CASS), CORTEX, Gaia, Context Management Framework, and Context Toolkit, which have introduced many of the elements related to context-aware computing [4]. Besides query requests, (logical) reasoning my also be founded on event-based processing [33].

Today, vendor-specific physical sensor middleware frameworks establish the major technology driver in mainstream context-aware computing. This has a major impact both in application development and in the current strategies of modeling context.

B. Current Technology Driver: Physical Sensor Context

A nice overview of the current state-of-the-art sensor technologies can be compiled by looking at the widespread mobile platforms, Android and iOS, and considering the various Web-based cross-platform development tools.

Android developers can make use of contextual information in several ways [1]. The first approach is using the Android Sensor Framework, which includes the motion sensors (e.g. accelerometers), environmental sensors (e.g. temperature) and position sensors (e.g. orientation sensor). It is also possible to access location information with Location API and other additional location services, such as Geofence API to alert user or applications when the user is entering a certain region.

iOS developers can access similar kinds of sensor information [2], with the chief exception of using Objective-C instead of Java.

In addition to device-specific interfaces, various browser APIs are also being developed. Accepting the obvious challenges in generalizing the sensor context of different operating systems, an interesting research perspective on context providers is established by cross-platform tools. These abstract the details of the various platforms, aiming to allow implementation of an application and its user interface for several mobile platforms more efficiently [34]. Table 1 lists the most popular cross-platform development tools, pointing out what sensor information is currently available.

TABLE I.
APIS SUPPORTED BY MAIN CROSS-PLATFORM DEVELOPMENT TOOLS (ADAPTED FROM [34])

Tool \ API	*Rhodes (JS)*	*Phone-Gap (JS)*	*Mo-Sync (JS)*	*Mo-Sync (C, C++)*	*Dragon-Rad*
Accelerometer		X	X		
Barcode	X	X			X
Bluetooth	X	X		X	
Calendar	X	X	X	X	X
Camera	X	X		X	
Capture		X	X	X	X
Compass		X	X		
Connection		X	X	X	
Contacts	X	X			X
Device	X	X	X	X	X
File	X	X	X	X	
Geolocation	X	X	X	X	X
Menu	X				X
NFC	X	X	X	X	X
Notification	X	X	X	X	
Screen Rot	X	X		X	
Storage	X	X	X	X	X

The need for standard access to application context has been also acknowledged by the related standardization organizations, namely the World Wide Web Consortium (W3C). In particular, the standardization of the so-called Open Web Platform includes several browser APIs that can be used in device and application independent manner for acquiring context [47].

It is interesting to observe that in most applications, developers must access and exploit sensor information directly, i.e. without the explicit notion of context engine. Further, the sensor information is mostly related to particular mobile device; any negotiation with additional context providing servers takes place in application-specific manner and is not directly supported by the (sensor) toolkits.

C. *Modeling Context*

Even if the mobile development frameworks do not yet provide integrated means for context-aware computing, various theoretical modeling approaches exist. We may identify several major strategies for modeling context [4][7], including *key-value models*, *object-oriented models*, and *ontology-based models*. Further, context can be defined in various ways [11].

Currently, there is no commonly agreed standard model or systems for sensing contextual information from various sources to enable reuse across various middle-ware systems and frameworks [4]. Ontology-based models, however, seem to offer many desirable properties such as information alignment, dealing with incomplete or partially understood information, domain-independent modeling, and formally working with context model of varying level of detail [8]. Adopting an context ontology standard might be beneficial but require global consensus on the matter.

Perhaps the most widely known sensor ontology is the W3C Semantic Sensor Network (SSN) ontology. SSN was developed based on reviewing 17 existing sensor or observation-centric ontologies [9]. In order to normalize the ontology and support its adoption with other ontologies, the SSN ontology is aligned with the general DOLCE Ultra Lite upper ontology, providing concepts such as PhysicalObject, Event, Situation, and Region.

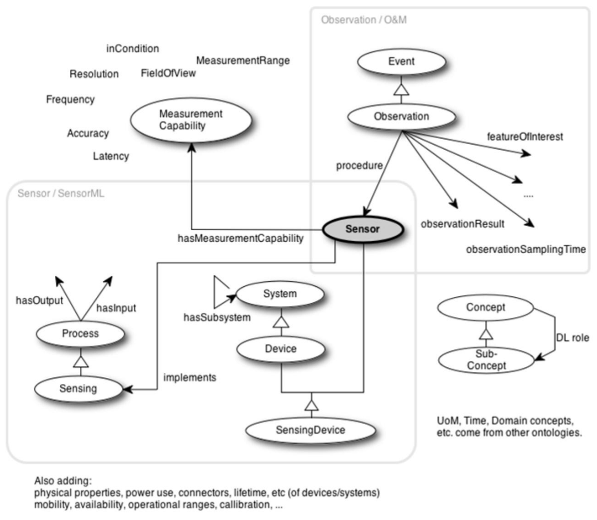

Figure 1. Overview of the SSN ontology structure prior to its modularization and alignment [29]

According to the SSN ontology, sensors may have properties such as accuracy in certain conditions, or may be deployed to observe a particular feature (see Figure 1) [29]. While abstractions or extensions are applicable, the SSN ontology in practice emphasizes the aspects of physical sensor networks.

Some context ontologies, however, by design do acknowledge the generalized logical (sensor) context. A prime example is the Service-Oriented Context-Aware Middleware (SOCAM) architecture, which aims providing efficient infrastructure support for building context-aware services in pervasive computing environments [19].

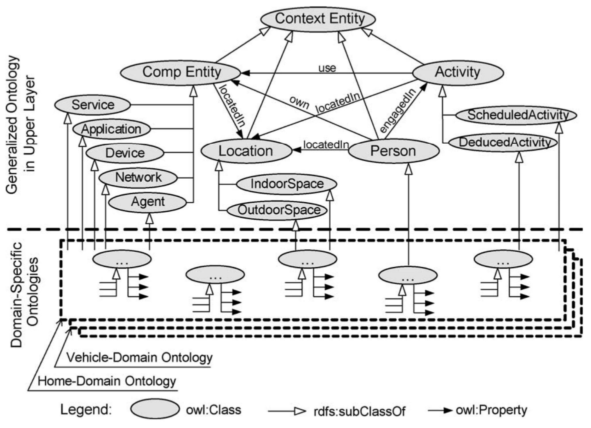

Figure 2. Class hierarchy of the upper (SOCAM) ontology [19]

In SOCAM, context modeling is carried out in OWL ontologies based on two-level information architecture: the general context concepts are captured in the common upper ontology and application-specific concepts in domain ontologies (see Figure 2). This approach suggests using upper-level context ontology, in addition to general top-level alignment ontology, for integrating various kinds of domain ontologies, suitable for explaining their role in providing context.

It is worth observing that both of the referred ontologies above are static by design: They provide a fixed structure for observations (etc.) that is assumed to be true and which does not change overtime. Indeed, a considerable practical challenge lies in managing imprecise, uncertain, or evolving information. While the significance of this topic is widely acknowledged in the related research [41][3], related standardization is still underway [28].

III. CONTEXT ENGINE

It is quite difficult to study context-aware semantic computing and context engines based on very abstract definitions. To make discussion more concrete, let us next first specify a certain kind of context engine and then illustrate the chief properties of a related prototype implementation. The context engine architecture is novel but of course influenced by the aforementioned, related research.

A. *Main Properties and Abstract Architecture*

In brief, a context engine accepts the overlapping responsibilities and tasks of the local context provider and (logical) context interpretation, which typically exceed the boundaries of individual applications. The essential tasks of a context engine include providing context information to the applications via various logical queries in terms of a standard I/O interface, and managing user preferences.

The chief communication mechanism between the context engine and the applications is the context ontology. Any domain-specific knowledge is captured in terms of references to domain-specific ontology modules and user preferences.

Individual applications do not necessarily have to fully understand the knowledge base of the context engine for making simple queries or asking questions about the current context, and vice versa. For instance, a simple telephony application might only need to know whether the user's activity status is currently "working" and if the user is in a business meeting or not.

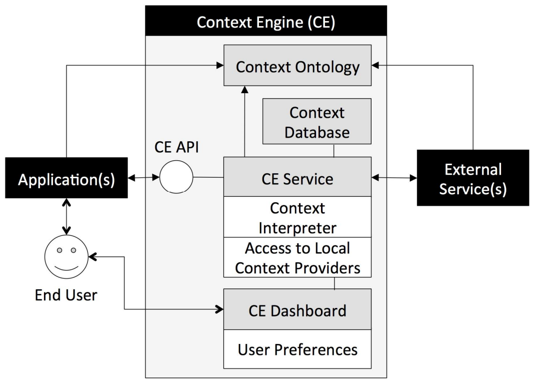

Figure 3. Simplified Context Engine Architecture

Note that domain-specific knowledge, i.e. how to actually utilize context in applications is not a responsibility of the context engine (cf. Figure 2). It does not have to understand application specific ontologies either. When needed, any extralogical computation (including heuristics, predictions, etc.) can be delegated to other services.

Simplified context engine architecture is depicted in Figure 3. In brief, the end user interacts with an application, which executes user activities and accesses contextual information through the context engine. Typical user applications include information management and communication applications, such as calendar, messaging and telephony applications, and novel software agents.

The context engine implements the context engine (service) Application Programing Interface (API). When context-aware semantic processing is needed, the user application requests context engine services. To fulfill these requests, the context engine has access to local context providers and possibly to external services. In addition to asking individual sensor values, a context interpretation query might ask the context engine to interpret and infer additional information about a given context, e.g. asking the known weather prediction (or archived value) for a given place at a given time. This might involve requests to external services.

To provide internal (sensor) context archives, the context engine maintains a context database. This can be used, e.g., to analyze and optimize context engine behavior. Note that the applications may also depend on external services in their internal design.

From the perspective of the end user, the context engine also manages global user preferences that are taken into account in context-aware semantic computing. For instance, the user might prefer not accepting certain kinds of phone calls outside the office hours. For this purpose, the context engine provides the user a dashboard GUI, for defining appropriate context engine settings – or explaining how to extract the information from sensor data. The user preferences might be considered as a rule system that refers to the context ontology and user tasks.

The end user dashboard might also be used for providing extra or overriding information, e.g. to check out how context affects a particular applications, or for overriding physical sensor context (perhaps "lying").

Notably extensions to the context engine include an event listener service (e.g. notify application when specific contextual event takes place) and shortcuts for certain kinds of commonly needed queries. More complex context engines might also extend the related knowledge bases and add extralogical services to the content engine I/O interface.

B. *Experimental Context Engine Environment*

We anticipate that eventually, a context engine (of a mobile device) is a service provided by an appropriate sensor framework, including an operating system level utility similar to personal details or privacy settings. When Internet connectivity can be assumed, the main alternative is providing context engine as a webized service.

Further, considering the current mobile application ecosystem(s), it seems likely that context engines are a business for large and established Internet service and application providers, simply due user base, credibility and critical mass of applications.

In the meantime, however, it is instructional to outline a research prototype deployable to a particular device that allows us to study both the concept and implementation of context-aware semantic computing. This allows us also to learn from the developer and the user experience, and enables discussing context-aware semantic computing research questions in a concrete setting.

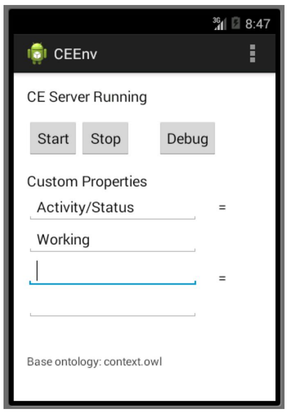

Figure 4. A context engine dashboard prototype

Figure 4 presents the main view of a Java-based context engine dashboard prototype running in an Android emulator. The user interface includes the essential functionality to start and stop the context engine service and to provide custom properties to the context ontology. Note that in this case, the context engine service has been physically deployed in a mobile device; a design stance that we will later challenge since the applications only need some access to the API and the dashboard.

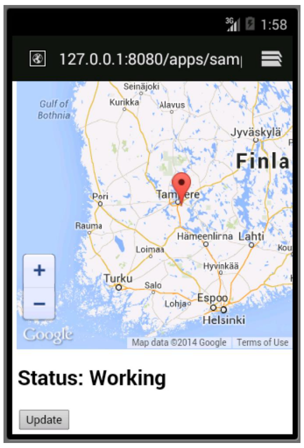

Figure 5. A sample browser application accessing the context engine

Indeed, from the implementation and deployment perspective, a major design decision lies in exposing the CE API to the applications. In an Android environment, a standard approach would be to deploy the context engine as a CE background service bound to a CE dashboard activity equipped with a graphical user interface. In this architecture, any application that wishes to utilize CE services would have to either bind to the CE service, or communicate with the CE activity via the so-called intent messaging. While this approach is clearly the most powerful one within the Android environment, it would require that each application is a native Android application which somewhat complicates experimental development.

For research purposes, we have adopted an alternative implementation strategy. In our case, the context engine includes a web server that allows publishing the CE API over HTTP, based on the NanoHttpd server implementation [15]. This allows prototyping the context engine quite flexibly, and supports experimenting and analyzing the context engine in various real and in simulated environments.

In itself, a sole context engine is of course not useful. Figure 5 depicts a sample Javascript application executed within the default browser of the Android emulator. In brief, the application depicts user location using the Google Maps API [17] and shows the activity status. Note that due to the default browser's Javascript security restrictions, the application needs to be downloaded through the localhost.

Behind the scenes, the application communicates with the context engine prototype via the HTTP-based Context Engine API, which allows accessing the local, built-in (generalized) sensor information. These include a subset of the Android sensor API and the custom properties communicated via the dashboard interface.

In cases when key-value sensor data is not sufficient, contextual information can be semantically bound together via the sample context OWL ontology. Put another way, the classes of the context ontology can be populated by the individual sensor information retrieved from the environment. In principle introducing any of the sensor APIs (cf. Table 1) is also straightforward.

In a production environment, controlling the applications' access to context information would require additional management controls. Recall that when installing a native android application, similar information is asked from the user, e.g. for granting access to user location or contacts.

Since the Google Maps API is used in rendering the map view, some user data is exposed to Google by the sample application. The (user of the) context engine can either choose to accept this, or refuse using the application altogether. While several map providers exist, it seems likely that all free services include terms that allow the service providers to collect usage data in order to improve the user experience and to provide value-added services to their customers.

Once started, the context engine provides applications two ways to access context-aware processing services. The first approach is straightforwardly asking specific, most recent raw sensor information using a HTTP GET request. In this case, the context ontology is only used as a sort of information architecture, for application and context engine (key-value) communication. The second, more powerful approach is formulating a query in SPARQL, using an Android port of the Jena framework [13]. With the help of reasoner services, e.g. transitive or OWL reasoning, this allows logical context (ontology-based) interpretation beyond mere syntactic queries.

When compared to using the built-in Android sensor API – in addition to the interface design – a major feature of the context engine prototype is that it can provide a single entry point to all sensor information. This allows analyzing context-aware processing, refining and optimizing the use of contextual information, and considering various implementation strategies, above the level of individual applications.

In addition to accessing the explicit context providers, the context engine can also exploit the usage patterns of the applications to infer the properties of the current context. For instance, with proper training data, the current status (Working) might be statistically inferred with certain degree of belief from the user logs so that the user would not have to explicitly enter the status at all.

IV. SIGNIFICANT RESEARCH PROBLEMS

The research background suggests that there is a theoretical need for a context engine component that mixes the responsibilities of context-aware and semantic computing. Empirical work verifies that this is also doable in several types of common application systems, including sensor networks and mobile machine platforms.

In the general case, however, several practical and theoretical challenges still remain, including:

Deployment. Deploying a context engine requires not only providing it to a device but also exposing it to applications – and attracting application developers' interest in using it.

In principle, access to a context engine can be provided on the operating system level (such as the Android Sensor Framework [1]), cross-platform development tool level (such as PhoneGap [34]), browser platform level (such as the navigator browser API [45]), on Internet service level (such as Google API [17]), or as a "yet another" HTTP service (such as our Context Engine prototype).

The device independent approaches provide obvious flexibility of devices and platforms but might require Internet connectivity, lack device-specific features, and raise privacy concerns.

Efficiency. In a production environment, it is not self-evident how the context ontology should be populated since accessing sensor information consumes processing and energy resources. The context engine should thus somehow optimize its performance, typically in terms of a trade-off between accuracy and costs [37]. One approach is to use a context database to cache recent sensor readings and/or to choose the cheapest matching sensors for better performance [31]. Further, deploying a complete query endpoint or a reasoner into a mobile device introduces its own overhead [13][5].

A potential solution is exploiting external computing services. In cases when the role of external context providers is particularly significant or Internet access must anyway be assumed, this might in fact suggest delegating certain context engine responsibilities to an external service altogether (cf. [12]). Relying onto external providers is also in line with the business logic of the major Internet service and product providers (c.f. [18]).

Note, however, that deploying e.g. the reasoning and the database modules of the context engine as an external service does not completely remove the need for local components with local computing costs [34]. This is particularly true when accessing local sensors.

Privacy. From a very practical point of view, the main utility of the context engine lies in the fact that it provides (within device or user session) a "centralized" access point to context information. This allows it to provide individual applications far better and more abstract information about the user context than each application could possibly do on its own. Note that in addition to using the explicitly registered context providing services, the context engine may also keep logs of the regular applications usage, e.g. to statistically infer contextual information.

When detailed personal information is managed, potential privacy issues are of course raised [8]. While dependency on external services may significantly help in providing more efficient context engine services of better quality, the obvious challenge lies in controlling and managing access to this information [24].

The baseline level of privacy is established by the related technologies and policies, such as submitting and storing sensitive information anonymously and securely [23]. A significant dependency lies in managing user preferences and matching these with the end user agreements when using various kinds of applications and services. Note that involves not only the context engine but also the individual application components (cf. [17]).

Understandability. In principle, it is not technically too difficult to provide a sophisticated rule system so that users could quite flexibility assert rules for acquiring and exposing their personal information to applications and managing how context is used in applications. For instance, consider adding a complete user preference rule component to our context engine. Such system, however, would be close to full-fledged logic programming and might be quite difficult use and understand in full detail [21].

More on the end user side, the adaptation due context-aware processing is another potential issue since it can be very difficult to users to recognize which parts of the application were adapted due context-aware computing in the background. To minimize the problems, adaptation might be visualized and analyzed during design [16]. Quality of contextual information is also a potential issue and may require managing additional metadata for quality control [26].

Problems of adaptive systems are well understood in personalized search systems, where increased adaptation may e.g. guide the decision making of the inexperienced users, but be perceived as too restrictive by expert users [25]. This seems to insist a tradeoff between the level of application adaptation and user control.

Semantics. Finally, while logical queries based on sensor information using a fixed ontology suffice for many tasks, a fundamental challenge is introduced by the very notion of context itself: Some contextual properties can be derived from others and thus, context is not a fixed concept in the first place [10].

For instance, the user location (e.g. Office) can sometimes be reliably used to (statistically reason and) predict the user activity status (e.g. Working), and vice versa.

Further, sensor and other information sources evolve over time, which should also be taken into account in semantic modeling. In particular, when regulations or organizational processes undergo changes at the workflow level, so does the notion of context. This may involve introducing new terms explaining context, or worse; using the old terms with a new meaning.

Thus, the design of the context ontology should ideally reflect the fact that some contextual properties may depend on each other, and that the context ontologies evolve over time. Alternatively, ontologies can also be used to support and evaluate the quality of statistical reasoning [36]. While using an alignment or top-level ontology seems indeed necessary, it may not be sufficient unless further semantics required in the evolution (e.g. same as, broader than) and statistical reasoning (e.g. evidence for, statistically independent) are encoded as well. Indeed, semantically modeling the aforementioned challenges, include a variant of the frame problem [38] and schema-level information evolution [32], which might well be called the hard problems of semantic computing.

V. CONCLUSION

Access to contextual information provides computational advantage in theory and in practice. In this article, we have outlined key elements of contemporary context-aware semantic computing. To make the discussion more concrete, we have also introduced a simple context engine prototype environment.

Intuitively, the insight of context-aware semantic computing is quite clear: information about the proper context can significantly improve the user experience by enabling the design of more efficient applications and help in optimizing the related computation beneath. However, when the related theoretical and engineering dependencies are analyzed in more detail, the single objective of context-aware semantic computing gets broken down into several, evidently competing design requirements [40].

Instead of a single problem, we thus have many. To address this observation, we have acknowledged several significant research questions in the area, including deployment, efficiency, privacy, understandability, and semantics.

Looking at the specific research problems related to context engine implementation, the topics of efficiency and access, understandability, and privacy deserve special attention. In principle, a local installation of the context engine gives best control over user privacy. In practice, however, the design choices and the user agreements of individual applications may easily invalidate this assumption. Local installation also means computation and memory overhead, and of course increases the risk of a single-point failure.

When Internet connectivity can be assumed, the idea of decentralizing the context interpretation and sensor (etc.) database management tasks seems like a viable design stance. This also potentially provides the context engine the ability to coordinate e.g. pattern recognition, classification, and context ontology evolution activities among users groups and sharing and reusing sensor data, effectively providing more efficient and better user experience. It seems likely, however, that this takes place at the expense of user privacy, even if it might offer only a limited access to the local sensors.

Strictly from the semantic computing point of view, the question how to properly model the related semantics, coined by the context ontology, is also highly relevant. Even quite simple use case scenarios point out that assuming fixed context ontology is an oversimplification, and that evolution at the level of domain-specific context ontology components have to be assumed at some point. Further, when learning, classification, and prediction algorithms are taken into account, it seems rather obvious that particular sensor information may appear either in the role of "physical" or "logical" sensor, e.g. in relationship with most recent sensor data and a particular prediction algorithm. This suggests introducing also evidence-based relationships (etc.) in the context ontology.

Thus, due to the complexity of the topic, it is unrealistic to assume that a single best solution exists for context engines and hence context-aware semantic computing in general. Instead, one must be satisfied with special-purpose approaches, e.g., finding a compromise between easy deployment and privacy, and between expressivity and understandability. From the perspective of context engine standardization, this of course requires prioritizing the design objectives, and/or acknowledging several context engine profiles and modes.

We believe that the large-scale adoption of context-aware semantic computing is inevitable, and is likely to take place in terms of the mainstream Internet service and product providers. Either way, context-aware semantic computing will have profound impact in applications.

REFERENCES

[1] Android Developer. Location and Sensors APIs. Available at http://developer.android.com/guide/topics/sensors/index.html

[2] Apple Developer. Features – iOS Technology Overview. Available at https://developer.apple.com/technologies/ios/features.html

[3] J.C. Augusto, J. Liu, P.J. McCullagh, H. Wang, and Y. Jian-Bo, "Management of uncertainty and spatio-temporal aspects for monitoring and diagnosis in a Smart Home," *International Journal of Computational Intelligence Systems*, 1(4) 2008, 361-378.

[4] M. Baldauf, S. Dustdar, and F. Rosenberg, "A Survey on Context Aware Systems," *Int. J. Ad Hoc Ubiquitous Computing* 2, no. 4, 2007, 263–277.

[5] K. Broda, K. Clark, R. Miller, and A. Russo, "SAGE: A Logical Agent-Based Environment Monitoring and Control System," *Proceedings of the 3rd European Conference on Ambient Intelligence*, AmI-09, 2009, pp. 112-117, Springer.

[6] J. Brown, P., Nigel Davies, Mark Smith, and Pete Steggles, "Towards a Better Understanding of Context and Context-Awareness," *Handheld and Ubiquitous Computing*, edited by Hans-W. Gellersen, pp. 304–307. Lecture Notes in Computer Science 1707. Springer Berlin Heidelberg, 1999.

[7] G, Chen and D. Kotz, "A survey of context-aware mobile computing research," Technical Report TR2000-381. Dartmouth College, 2000.

[8] H. Chen and T. Finin, "An ontology for a context aware pervasive computing environment," *IJCAI Workshop on Ontologies and Distributed Systems*, Acapulco MX 2003.

[9] M. Compton, P. Barnaghi, L. Bermudez, R. Garcia-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W.D. Kelsey, D. Le Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A., Sheth, and K. Taylor, "The SSN Ontology of the W3C Semantic Sensor Network Incubator Group," *Journal of Web Semantics*, 2012.

[10] U. Christoph and J. von Stülpnagel, "Context Detection on Mobile Devices," *Second Workshop on Context-Systems Design, Evaluation and Optimisation (CoSDEO 2011)*, in conjunction with the 24th International Conference on Architecture of Computing Systems (ARCS) in Como, Italy, February 22nd - 25th, 2011.

[11] A. Dey and G. Abowd, "Towards a better understanding of context and context-awareness," *Workshop on the What, Who, Where, When and How of Context-Awareness* at CHI 2000, 2000.

[12] O. Droegehorn, "Optimizing background-communication of mobile Devices and Sensors to drive End-User Services," *IEEE 22nd International Symposium on Personal, Indoor and Mobile Radio Communication*, 2011.

[13] e-Lite. Apache Jena on Android. Available at http://elite.polito.it/jena-on-android

[14] Forschungsunion, "Securing the future of German manufacturing industry: recommendations for implementing the strategic

initiative INDUSTRIE 4.0," Final report of the Industrie 4.0 Working Group. Forschungsunion & Acatech, 2013.

[15] GitHub. NanoHttpd. Available at https://github.com/NanoHttpd/nanohttpd

[16] J. C. Georgas, A. van der Hoek, and R. N. Taylor, "Using Architectural Models to Manage and Visualize Runtime Adaptation," *Computer*, October 2009.

[17] Google. Google Maps API. Available at https://developers.google.com/maps/

[18] Google. Google+ API. Available at https://developers.google.com/+/api/

[19] T. Gu, P. Hung Keng, and Z. Da Qing, "A Service-Oriented Middleware for Building Context-Aware Services," *Journal of Netw. Comput. Appl.* 28, no. 1, 1-18, 2005.

[20] M. Hatala, R. Wakkary, and L. Kalantari, “Ontologies and rules in support of real-time ubiquitous application,” *Journal of Web Semantics*, Special Issue on ”Rules and ontologies for Semantic Web”, vol. 3, no. 1, pp. 5–22, 2005.

[21] A. Hoffmann, *Paradigms of Artificial Intelligence: A Methodological & Computational Analysis*. Springer-Verlag, August 1998.

[22] *International Journal of Semantic Computing*. World Scientific Publishing. Available at http://www.worldscientific.com/page/ijsc/aims-scope

[23] P. Jagtap, A. Joshi, T. Finin, and L. Zavala, "Preserving Privacy in Context-Aware Systems," *Fifth IEEE International Conference on Semantic Computing*, 2011.

[24] X. Jiang and J. A. Landay, "Modeling Privacy Control in Context-Aware Systems," *IEEE Pervasive computing*, pp. 59-63, 2002.

[25] A. Kamis and M.J. Davern, "Personalizing to Product Category Knowledge: Exploring the Mediating Effect of Shopping Tools on Decision Confidence," *Proceedings of the 37th Hawaii International Conference on System Sciences*, IEEE, 2004.

[26] E. Kim and J. Choi, "A Context Management System for Supporting Context-Aware Applications," *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, 2008.

[27] A. Kofod-Petersen and M. Mikalsen, “Representing and Reasoning about Context in a Mobile Environment,” *Revue d’Intelligence Artificielle*, vol. 19, no. 3, pp. 479–498, 2005.

[28] K.J. Laskey, K.B. Laskey, P.C.G. Costa, M.M. Kokar, M. Trevon, and T. Lukasiewicz, " Uncertainty Reasoning for the World Wide Web," W3C Incubator Group Report 31 March 2008. Available at http://www.w3.org/2005/Incubator/urw3/XGR-urw3/

[29] L. Lefort, C, Henson, and K. Taylor, "Semantic Sensor Network XG Final Report," W3C Incubator Group Report 28 June 2011. Available at http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/

[30] P. Mehra, "Context-Aware Computing: Beyond Search and Location-Based Services," *IEEE Internet Computing* 16, no. 2, 12 – 16, 2012.

[31] S. Nath, "ACE: exploiting correlation for energy-efficient and continuous context sensing," *In MobiSys'12*, June 25-29, UK, pp. 29-42, 2012.

[32] O. Nykänen, "Semantic Web for Evolutionary Peer-to-Peer Knowledge Space," In Birkenbihl, K., Quesada-Ruiz, E., & Priesca-Balbin, P. (Eds.) Monograph: Universal, Ubiquitous and Intelligent Web, *UPGRADE, The European Journal for the Informatics Professional*, Vol. X, Issue No. 1, February 2009, ISSN 1684-5285, CEPIS & Novática. Available at http://www.upgrade-cepis.org/issues/2009/1/upgrade-vol-X-1.html

[33] T. Patkos, I. Chrysakis, A. Bikakis, D. Plexousakis, and G. Antoniou, "A Reasoning Framework for Ambient Intelligence," *SETN 2010*, pp. 213-222.

[34] M. Palmieri, I. Singh, and A. Cicchetti, "Comparison of Cross-Platform Mobile Development Tools," *16th International Conference on Intelligence in Next Generation Networks (ICIN)*, pp. 179–186, 2012.

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

[36] D. Riboni, "Towards the Combination of Statistical and Symbolic Techniques for Activity Recognition," *IEEE Pervasive Computing and Communications*, 2009.

[37] N. Roy, A. Misra, C. Julien, S. K. Das, and J. Biswas, "An Energy-Efficient Quality Adaptive Framework for Multi-Modal Sensor Context Recognition," *IEEE International Conference on Pervasive Computing and Communications* (PerCom), Seattle, March 21-25, 2011.

[38] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[39] P. Sheu, H. Yu, C.V. Ramamoorthy, A.K. Joshi, and L.A. Zadeh, *Semantic Computing*. Wiley-IEEE Press, 2010.

[40] W. Sitou, and B. Spanfelner, "Towards Requirements Engineering for Context Adaptive Systems," *31st Annual International Computer Software and Applications Conference*, 2007.

[41] U. Straccia, "Foundations of Fuzzy Logic and Semantic Web Languages," Chapman & Hall, USA:CRC Press, 2014.

[42] T. Strang and C. Linnhoff-Popien, "A Context Modeling Survey," *Workshop on Advanced Context Modelling, Reasoning and Management*, In UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England, 2004.

[43] A. Toninelli, R. Montanari, L. Kagal, and O. Lassila. "A Semantic Context-Aware Access Control Framework for Secure Collaborations in Pervasive Computing Environments," *Proceedings of the 5th International Semantic Web Conference*, Springer, pp. 5-9, 2006.

[44] E. Yndurain, D. Bernhardt, and C. Campo, "Augmenting Mobile Search Engines to Leverage Context Awareness," *IEEE Internet Computing* 16, no. 2, pp. 17–25, 2012.

[45] WebPlatform.org. APIs. Available at http://docs.webplatform.org/wiki/apis

[46] W3C Data Web Activity – Building the Web of Data. World Wide Web Consortium (W3C). Available at http://www.w3.org/2013/data/

[47] W3C Standards. World Wide Web Consortium (W3C). Available at http://www.w3.org/standards/

[48] W3C WOT. Web of Things Community Group. World Wide Web Consortium (W3C). Available at http://www.w3.org/community/wot/

AUTHORS

Dr. O. A. Nykänen works as Adjunct Professor at the Tampere University of Technology, Department of Mathematics, Tampere, Finland (e-mail: ossi.nykanen@tut.fi). His research interests include semantic computing, information modeling and scientific visualization, (computer-supported) mathematics and education, and the related applications. In addition to his research and higher education activities, Dr. Nykänen is the Manager of the World Wide Web Consortium (W3C) Finnish office.

M.Sc. A. Rivero Rodriguez works as Researcher at the Tampere University of Technology, Department of Mathematics, Tampere, Finland (e-mail: Alejandro.rivero@tut.fi), within the Marie Curie ITN research project MULTI-POS. His research interests include context-awareness, semantic modelling/computing, and e-learning.

Manuscript received 13 May 2014. This work was supported in part by the Marie Curie ITN research project MULTI-POS, Multi-technology positioning professionals (Grant agreement no. 316528, 2012-2016).

Published as submitted by the author(s).