Context Storage for M2M Scenarios

Mário Antunes
Instituto de Telecomunicações
Universidade de Aveiro
Aveiro, Portugal
Email: mario.antunes@av.it.pt

Diogo Gomes
Instituto de Telecomunicações
Universidade de Aveiro
Aveiro, Portugal
Email: dgomes@av.it.pt

Rui Aguiar
Instituto de Telecomunicações
Universidade de Aveiro
Aveiro, Portugal
Email: ruilaa@av.it.pt

***Abstract*—As the number of environmental sensors grows, it becomes increasingly difficult to manage, store and process all these sources of information. Several context representation schemes try to standardize this information, however none of them have been widely adopted. Instead of proposing yet another context representation scheme, we discuss efficient ways to deal with this diversity of representation schemes. We defined the basic requirements for flexible context storage systems, proposed an implementation and compared our implementation against two other approaches. Our solution provides more value than the remaining solutions without suffering a significant decrease in performance.**

***Keywords*—Internet of things, M2M, context information**

I. Introduction

Mobile devices, from tablets to environmental sensor networks, produce massive amounts of data. The number of mobile devices is growing. According to the ICT Knowledge Transfer Network (ICT KTN), the number of mobile devices is expected to increase worldwide from 4.5 billion in 2011 to 50 billion by 2020 [1]. The data generated by these devices, if properly processed, can be used to infer the context of entities (entity in this context can be a person, a device, a room, etc.).

In M2M (machine to machine) scenarios, an entity's context can be used to provide added value: improve efficiency, detect abnormal conditions or advertise information. The following example illustrates the importance of context information in M2M scenarios. Take into account several sensors (e.g. air and leaf humidity, air and soil temperature sensors) in a green house. Currently several platforms are able to automatically water the plants. However, by combining the data from all the sensors it is further possible to detect the plants growth and even if the plants are being attacked by any infestation or suffering any disease. The same platform for automatic watering is not able to cope with this added information.

For scenarios like the previous to become reality, it is necessary to develop a way to store and process such diverse machine context information. The common definitions of context information [2], [3] do not provide any insight about its structure. In fact, each device can share context information with a different structure. For example, sensory information and location information can be used to characterize an entity context, yet the two can have very different structures. Standard context management platforms commonly store context information in relational databases. In order to take full advantage of a relational database it is necessary to map context information into the relational model. We can devise a mapping process easily only if there is a common context representation.

The main objective of context representation is to standardize the process of sharing context information through several services. Context-aware platforms strongly benefit from a uniform environment: the storage process is easier (the information follows a known structure) and the analysis of the information becomes simpler. Multiple context representations have been proposed, such as ContextML[4], SensorML[5] COBRA-Ont[6], OASIS XDI[7] and OASIS XLIFF[8]. All these representations try to solve the same problem, but each representation is quite different and incompatible with the other. None of the above mentioned representations have been widely accepted either by the academia or the industry. Usually, each context-aware platform defines his own context representation based on the platform scenarios. This breaks compatibility between platforms and limits the quantity of context information that can be used in M2M applications.

It is possible (but unlikely), that in the future a context representation standard will be widely adopted. Until then, context-aware platforms have to deal with multiple context representations. The work presented in this paper addresses this problem and analyses possible representation schemes independent of storage solutions.

The remainder of the paper is organized as follows. In Section II we analyse how context information can be organized and define the basic requirements for context storage systems. Three context storage systems are analysed in Section III. Section IV contains implementation details of our context storage system. The results of the quantitative evaluation performed are in Section V. Finally the discussion and conclusions are presented in Section VI.

II. Context organization

Context information is an enabler for deeper and further data analysis, requiring the integration of an increasing number of information sources. As previously mentioned, nowadays no widely accepted context representation scheme exists; instead there are several approaches to deal with context information.

We could define a completely new context representation. This context representation would be optimized for that specific context-aware platform, but then the platform could never easily process other context sources.

978-1-4799-0059-6/13/$31.00 ©2013 IEEE

978-1-4799-2003-7/14/$31.00 ©2014 IEEE

One could adopt one of the existing context representation schemes. In that case, the context-aware platform supports all the compatible information sources. However, this limits the quantity of context sources, due to the diversity of context representations available, and complexity scaling.

Another possibility would be developing several ontologies in order to standardize context storage. The context-aware platform supports a specific data model, and for each context representation there is an ontology that converts the representation into the internal data model. In computer science, ontology is "an explicit and formal specification of a conceptualization" [9]. An example of this approach is COEUS [10]. This type of platform supports several context representations, yet it is necessary to define a new ontology for each representation.

Or finally, we can accept the diversity of context representation as a consequence of economic pressures, and prepare for this inevitability. In this paper we analyse representation independent storage solutions as the proper technical approach to this situation.

Before discussing possible representation independent storage solutions, let us point out important characteristics of context information. The common definitions of context information [2], [3] are so broad that any information related to an entity can be considered context information. A representation independent storage solution must aim to generalize the storage process while providing discriminating retrievals. Generalizing the storing process is considerable easy: several databases can store binary blobs or textual fields. The main challenge is classifying context information in a way that provides discriminative retrieval and does not force a specific representation at storage time.

Clay Shirky [11], Gabriela Avram [12] and Thomas Gruber [13] point out that top-down classifications induce a bias into their model of the world. According to the authors, the signal loss brought about by the unification process of top-down classification is enormous. Top-down classifications limit the dimension along which one can make distinctions, and local choices at the leaves are constrained by global categorizations in the branches. It is therefore inherently difficult to put things in their hierarchical places, and the categories are often forced. The authors explain that probabilistic models on top of bottom-up (user-centred) characterization produce better results than binary schemes built on top of top-down classification. Moreover, bottom-up (user-centred) characterization is massively dimensional, and there is no global consistency imposed by current practice.

According to the authors, the best solution to classify context information is through bottom-up characterization. Although sensor information is not usually tagged by users (it is automatically generated by mobile devices and sensors), we can model the tagging process. One possibility is to model the tagging process as keyword extraction [14]–[16]. A keyword is a sequence of one or more words that provide a compact representation of a document's content. Ideally, keywords represent in condensed form the essential content of a document.

Based on this analysis, we define three requirements that a context storage must fulfil: generalize storing process, discriminative retrieval and the ability to scale. The first two requirements complement each other. Due to the lack of a widely adopted context representation, a context database must have the capacity to store any type of context information. However this generalized storing process must be balanced with a discriminative retrieval process. Discriminative retrieval process means that information is retrieved based on concepts instead of simple words. In order words, the ideal context database must store and accurately pinpoint any piece of context information associated with any type of sensor.

Discriminative retrieval process can be achieved through different methods. Two of the most common methods are through semantic web, and using an information retrieval (IR) system. Semantic web methods require structured information (mainly expressed in RDF or OWL) which in the scenario of context information is a disadvantage. As discussed above, bottom-up classification (free form distributed tagging) is the best solution to classify context information. Hence, a context database must provide a discriminative retrieval based on IR systems.

As previously mentioned, tags can be modelled with keywords (discriminative terms). Information retrieval systems use discriminative terms to index documents, commonly through a tf-idf (term frequency-inverse document frequency) [17] statistical framework. Tf-idf is a numerical statistic which reflects how important a word is to a document in a collection or corpus. An IR allows a user to retrieve documents based on discriminative terms (similar to a search on a search engine). Moreover, it is possible to enrich a IR system with semantic information. Sensors commonly publish information in a semi-structured representation (e.g. XML, YAML, JSON, BSON). With a custom document analyser, the IR system is able to correctly parse semi-structured representations. The relevant terms are selected from the set of "keys" and not the respective values. The initial set of relevant terms is semantically enriched through a custom concept graph or semantic services (Alchemy API[1]). With this addition, and a new similarity metric that takes into account the semantic value of the relevant terms, users can search for concepts instead of simple words. In sum, information retrieval systems fulfils the discriminative retrieval requirement.

The third and final, ability to scale, is related with the sheer number of sensors. The number of sensors is rapidly increasing. As a consequence, the quantity of context information is also increasing, and a context database must be robust to this increase. Simple replications allows a database to scale retrieve operations. To scale write operations it is necessary a distributed database through several nodes, each one containing a set of the whole database (horizontal partitioning/sharding [18]).

III. Context storage

Databases can be divided into two major groups: relational and NoSQL databases [19]. There are several types of NoSQL databases, such as graph based, document store, key-value, and object based. For this paper we are only interested in document store and key-value NoSQL databases. The remaining types of NoSQL databases adopt a stricter data model which is not compatible with flexible context information, as previously

[1] http://www.alchemyapi.com/

described. Within this section we analyse three context storage approaches. The first solution uses a conventional relational database and the other two use NoSQL databases (document store and key-value database respectively).

Relational databases, in general, are not completely suitable for storing context information. None of the key requirements for complex M2M environments are completely fulfilled. It is possible to develop a context storage solution using relational databases: context information can be stored in a single table, some relational databases already support full-text search and the majority of the databases support replication. Sensor information is stored as text fields (key-value approach), using a single table with two columns: the first column holds a unique identifier and the second column holds the document. The second column is then indexed by the full-text search engine.

Sensory information is written in a single table, without taking advantage of the relational model. The full-text engine, present in relational databases, is rather limited. It is a simple index that contains keywords instead of primary or external keys. The search is based on keyword similarity and not concept similarity. The full-text engine lack several features present in a more flexible information retrieval system.

As previously mentioned, sensors transmit information in a textual format. Sensory information is in fact a textual document, as such it can be stored in a **document store database**. Document store databases are databases designed to store, retrieve and manage documents. In this context, documents are semi-structured data, typically encoded in XML, YAML, JSON or BSON. Documents, for this type of databases, are similar to records in a relational database. However, they are less rigid and do not follow a standard schema. Document-oriented databases typically provide a query mechanism based on IR systems or custom indexes.

Compared with a conventional relational database this approach is an improvement, but it is not ideal. A document store database implies conversions of document representations, whenever the source document representation is different from the document representation. The query mechanism is also limited. Although based on IR systems the indexes are commonly created automatically based on the document structure. Therefore, in order to retrieve documents it is necessary to known the structure of the documents. As previously mentioned the representation of context information for M2M is unknown, hence these indexes are not suitable in the scenario of context store.

As a counterpart to these approaches, we propose a modular context database that provides a key value database and an information retrieval interface. Our implementation combines LevelDB (key-value database) with Lucene (information retrieval system). Details about the implementation are in section IV.

The key-value database accepts any document in any representation without relying in the relation model or require document conversion. The main idea of a modular context database is that several key-values databases and information retrieval systems can be plugged-in with minimal effort. Hence the integration between the key-value database and the informational retrieval system is not limited by any constrains. It is also important to mention that our implementation only stores the keys for the documents, and as such the complete document is only stored once.

The solution scalability is not directly related with the storage's type. It is closely related with the storage implementation, as such it is not discussed in this section. Table I summarizes which requirements are fulfilled by the different approaches to context databases.

TABLE I. REQUIREMENTS FULFILLED BY EACH SOLUTION.

Databases/ Requirements	Generalize store	Discriminative retrieval
Relational Database	Partial	Partial
Document Store	**Full**	Partial
Proposed Solution	**Full**	**Full**

IV. IMPLEMENTATION

In this section we discuss important details about our solution. The context storage is divided into 3 different components as depict in Fig. 1. The storage and index components store and index context information respectively, the router communicates with the index and storage components in order to fulfil each operation. All the components communicate with each other through message passing.

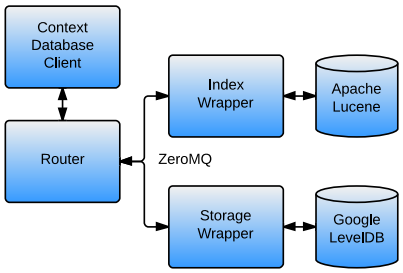

Fig. 1. Propose context storage architecture.

The components communicate with each other using the ZeroMQ[2] socket library. ZeroMQ supports several transportation methods: TCP sockets, inter-process communication and inter-thread communication. Messaging passing allows the application to be distributed through several machines and each component can be written in any programming language, without being restricted by the router component. This strategy is then specially suitable for the diversity of environments in M2M applications.

Although the modular solution in our architecture has several advantages, it can also produce sub-optimal solutions. Conceptually it is easier to devise storage and index component as two independent components. The router component (central component) decomposes each operation as a sequence of independent operations from the remaining components. As an example, a search operation is decomposed as a search operation over the index component and a retrieve operation

[2]http://www.zeromq.org/

over the storage component. The search operation over the index component returns the relevant documents' keys. Using the documents' keys, the retrieve operation over the storage component returns the relevant documents. The search operation's performance can be optimized, so a single component does not suffer the communication's overhead. This modular architecture allows us to develop applications that are more scalable, for example several search operations can run in parallel with minimal locks. The ability to scale is one of the key requirements of a context storage system. Hence, it is more important to optimize the solution for simultaneous operations.

In order to decrease coupling, each component follows two design patterns: abstract factory and command. Each operation is encapsulated as a message object and each message object is in turn encapsulated as an action object. The abstract factory creates an action object from message objects, this way the translation is transparent to the component. The command design pattern makes the process of executing an action transparent to the component. Each action object implements a execute method that contains the necessary steps to execute the respective action. Another advantage of the command pattern is multitasking, each operation of the component is independent of each other, so multiple requests can be performed at the same time using different threads (as depicted in Fig. 2). The combination of abstract factory and command design pattern allows us to achieve a modular storage system with focus on parallelism instead of single request performance.

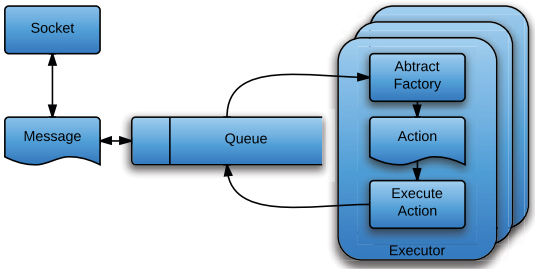

Fig. 2. General architecture of the components.

The router component exposes an interface that is a combination of a key-value and information retrieval interface. It exposes four methods: insert, retrieve search and delete documents. Each one of these methods is implemented as a set of operations of the storage and index components. The decomposition of operations improves the performance of the platform in multi-threaded scenarios.

The index component is mainly an information retrieval system, responsible for indexing and searching relevant documents. It was prototyped in Java, using Apache Lucene[3] at its core. This component was developed with special attention to parallelism. The IndexWriter class was expanded to support periodical commits (safe store in the disk) with a background thread. The component also uses near-real-time search[4]. This feature allows an index changes to be visible to a new searcher with fast turnaround time.

[3]lucene.apache.org/core/

[4]blog.mikemccandless.com/2011/06/lucenes-near-real-time-search-is-fast.html

The storage component is mainly a NoSQL key-value database responsible for storing the documents. It was prototyped in C++, using LevelDB [5] as its core library. LevelDB is one of the fastest key-values databases currently available, developed by Google and inspired by Google BigTable [20]. LevelDB, apart from being efficient, supports compression through Google Snappy [6]. Compression is an important feature for M2M scenarios, and improve efficiency. The structure of the context representation scheme tends to be constant within devices. As such, compression can reduce significantly the size of the database.

V. Performance Evaluation

In Section III we evaluated qualitatively three context storage approaches. Some approaches do not fulfil all the requirements completely, however all the mentioned approaches are sufficient for addressing at least some M2M scenarios. Within this section we evaluate quantitatively the three solutions. Instead of measuring their performance with a synthetic test, we evaluated the performance of the storage systems with a simulation of a M2M smart agriculture scenario. A M2M simulation provides more information about the storage systems' performance in a real world scenario than a synthetic test.

From the several relational databases available we selected PostgreSQL for illustration. PostgreSQL is one of the most used relational databases, and supports full-text search natively. Full-text search is supported through two different types of indexes: GIN (Generalized Inverted Index) and GiST (Generalized Search Tree). GIN index lookups are faster than GiST, but it takes more time to build a GIN than a GiST index. This database natively only supports replication (version 9 above). Replication improves the read performance at the expense of the write performance. There are some 3rd party clustering proxies, however their main objective is high availability and not write performance.

Santos et al. [21] proposed to store context information in a document store database. The authors developed a context database combining a document store database (CouchDB) with a information retrieval system (Lucene). CouchDB only stores documents as JSON objects. The authors used Lucene (through a external plugin named couchdb-lucene) to index the contents of the database. Due to limitations in plugin designs several features present in information retrieval systems are not available (its not possible to add custom components, or modify the behaviour of the default components). CouchDB supports replication but does not support partitioning. There are some 3rd party proxies that offer clustering capabilities for CouchDB. However, the proxy is not capable of interacting with the external indexer (the search request is direct routed to CouchDB itself) making it difficult to have a external indexing system and a distributed version of CouchDB at the same time.

We developed a smart agriculture simulation based on the LOFAR-agro project [22], [23]. The LOFAR-agro was the first large-scale experiment in precision agriculture in the Netherlands. This project addressed the protection of a potato crop against *phytophthora*. The experiment was composed

[5]code.google.com/p/leveldb/

[6]code.google.com/p/snappy/

of 150 sensor nodes, each node measuring temperature and relative humidity. Every 10 minutes each node sends a message containing a measurement (set operation). The normal farming routine requires that a treatment advice is available every morning, when the activities for the day are planned. In other words, every morning it is necessary to retrieve all the data from the previous day (search operation).

TABLE II. SIMULATION'S PARAMETERS

Simulation parameters	
Number of nodes	150
Set rate	every 10 minutes
Number of clients	1
Search rate	once a day
Duration	one month (30 days)
Warm up	one week (7 days)

TABLE II summarizes the parameters that where used in our simulation. For efficiency reasons we accelerated the simulation 1000 times (it took 43 minutes instead of 30 days). This increase in speed can be interpreted as a load effect of 1000 crops (it has a similar effect) and thus also illustrates scalability of the system.

The simulation ran in a virtual machine with the following specifications: 8 GB of memory RAM and 4 CPUs with a 2,4 GHz clock speed. The virtual machine had a Linux operating system (kernel version 3.2.29), PostgreSQL (9.2.1), CouchDB (1.2.1) and the proposed solution (based on LevelDB 1.9 and Lucene 4.1). PostgreSQL supports two kinds of full text indexes (Gist and Gin), both indexes were evaluated in our simulation.

At turns, each context storage concept was evaluated as the storage model for M2M using the above mentioned simulation. The virtual sensors sent documents in JSON. This way there is no document transformation penalty for the CouchDB based solution. It is worth mentioning that a request queue was used to connect the databases with the simulation program, in order to simulate the asynchronous behaviour of the sensors. The write operations are asynchronous, yet the search operations are not. A search operation is performed everyday by the farming advisor program, and the program waits for the reply before analysing the data. We measured the duration of each operation inside the context database, the maximum size of the request queue and the CPU load. The duration of each operation is a standard measure of performance. The other two performance measures provide insight about the databases scalability.

TABLE III. PERFORMANCE EVALUATION

Context Database	Write (time seconds)	Search (time seconds)	Request Queue (max. size)
Relational Database (GIN)	0.0006	0.23	141
Relational Database (GiST)	0.0004	0.29	156
Document Store	0.0242	41.28	40876
Proposed Solution	0.0024	5.41	40

The average write and search time are summarized in TABLE III. The relational database had the lowest average write and search time (which correspond to the highest performance). On the other hand, the solution based on document-store was completely outperformed by the remaining solutions.

Our solution stood in the middle of the performance evaluation. It is important to mention that our solution was devised to achieve good scalability. The request queue maximum size provides some insight about the solution's ability to scale. The queue grows as more and more requests arrive and are not fulfilled. Our solution has the smallest queue size showing the highest processing capability. The document-store solution had a queue size at least two orders of magnitude greater than the remaining solutions. One of the most important features of CouchDB is supposedly concurrency, yet in this simulation it had the worst performance in terms of scalability. Another evidence that, in this simulation, the previous solution has a poor ability to scale is the high CPU load measured (see Fig. 3): the previous solution had an average CPU load of 60%, while the remaining solutions have an average around 15%. At close inspection we notice that this solution took much longer than the other solutions to finish the simulation.

In the beginning, our solution had a rather high CPU load. The initial high CPU load is a consequence of Lucene creating and filling their buffers, but after some time the CPU load decreased to a consistent 15%.

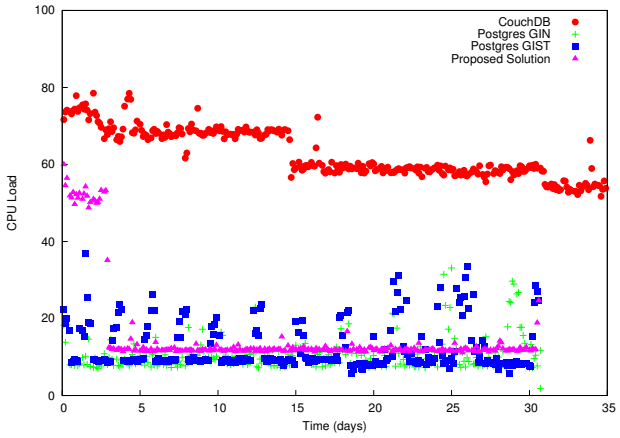

Fig. 3. CPU load

To summarize, the relational database achieved the best performance in this evaluation. The difference in performance of the two PostgreSQL indexes is marginal, at least in the considered M2M scenario. Although it does not completely fulfil context storage requirements for generic M2M, as discussed in Section III, it is sufficient for this specific scenario.

On the other hand, the document store based solution had the worst performance in our simulation. The integration with the information retrieval system is not optimal. In a document-oriented database, every document is decomposed, analysed and stored. Which means that every document is decomposed and analysed twice (by the CouchDB itself and the Lucene plug-in). As a result, document-oriented databases are not suitable for real-time M2M scenarios.

Our solution provides a better integration between a in-

formation retrieval system and a database for generic M2M scenarios. The evaluation proves that is possible to fulfil context storage requirements without significant performance degradation.

VI. Discussion and Conclusions

As the number of sensors increase, it becomes increasingly difficult to store, process and analyse context information. There are countless context representation schemes, however none of them have been widely either adopted by the academia or the industry. Within this paper, we defined the basic context storage requirements and proposed a new context storage architecture for generic M2M applications

Let us consider a more complex but relevant scenario, where the farming field has more sensors' types and a local weather station. With the combined sensory information it is possible to detected several crop infestations, meteorological conditions, optimize the watering process and the crop growth. Each one of the above mentioned conditions are not related with a single sensor. It can only be detected with the combination of context information from multiple sensors. Without loss of generality let us only consider crop infestations detection. Several crop infestations can be detected based on weather information, soil pH, temperature and humidity, leaf moisture etc. In a ideal context storage system, information related with a specific crop infestation should be automatically tagged with an appropriate tag (e.g. the name of the crop infestation). The information published by the sensors does not mention any crop infestation, it only contains measurements of the environment. As discussed in Section III the search mechanism must be enhanced, allowing users to search with concepts instead of simple words. It is quite difficult to add these functionalities to relational databases.

Three context storage approaches were analysed and evaluated both qualitatively and quantitatively. Qualitatively, our solution fulfils a broader set of requirements than the remaining. Quantitatively, relational databases achieved the best performance for the simulated scenario. Its important to mention that relational databases are not suitable for more complex M2M scenarios. Our simulated scenario only has one context representation, from one environment.

Relational databases (with full-text search) are sufficient for the current specific M2M scenarios. But due to three key factors, relational databases are not viable approaches for context storage systems. The three factors are: the lack of a widely accepted context representation scheme, the ever growing number of context sources and the inherent complexity of novel M2M scenarios. Context storage systems based on bottom-up (user-centred) characterization are a long term, but necessary solution. The research presented in this paper intends to be a step in the right direction for widespread M2M communications support.

Acknowledgement

This work has been partially funded by the Portuguese Innovation Agency/National Strategic Reference Framework (AdI/QREN) under grant agreement No. 2011/021580 (APOLLO project) and by project Cloud Thinking (CENTRO-07-ST24-FEDER-002031), co-funded by QREN, "Mais Centro" program.

References

[1] U. F. I. S. Group, "Future internet report," ICT Knowledge Transfer Network, Tech. Rep., May 2011.

[2] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Proc. of the 1st international symposium on Handheld and Ubiquitous Computing*, 1999, pp. 304–307.

[3] T. Winograd, "Architectures for context," *Hum.-Comput. Interact.*, vol. 16, no. 2, pp. 401–419, December 2001.

[4] M. Knappmeyer, S. L. Kiani, C. Frà, B. Moltchanov, and N. Baker, "Contextml: a light-weight context representation and context management schema," in *Proc. of the 5th IEEE international conference on Wireless pervasive computing*, 2010, pp. 367–372.

[5] M. Botts and A. Robin, "Opengis sensor model language (sensorml) implementation specification," OGC, Tech. Rep., July 2007.

[6] H. Chen, T. Finin, and A. Joshi, "An ontology for context-aware pervasive computing environments," *The Knowledge Engineering Review*, vol. 18, pp. 197–207, Setember 2003.

[7] M. Sabadello and D. Reed, "Oasis xri data interchange," https://www.oasis-open.org/committees/xdi/charter.php, accessed: 22-07-2013.

[8] B. Schnabel, T. Comerford, and D. Filip, "Oasis xml localisation interchange file format," https://www.oasis-open.org/committees/xdi/charter.php, accessed: 22-07-2013.

[9] T. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," in *International Journal Human-Computer Studies*, vol. 43, no. 5-6, November 1993, pp. 907–928.

[10] P. Lopes and J. L. Oliveira, "Coeus: Semantic web in a box for biomedical applications," *Journal of Biomedical Semantics*, vol. 3, no. 1, p. 11, 2012.

[11] C. Shirky, "Ontology is overrated: Categories, links, and tags," http://shirky.com/writings/ontology_overrated.html, May 2005, accessed: 22-07-2013.

[12] G. Avram, "At the crossroads of knowledge management and social software," *Electronic Journal of Knowledge Management*, vol. 4, no. 1, pp. 1–10, January 2006.

[13] T. Gruber, "Ontology of folksonomy: A mash-up of apples and oranges," *International Journal on Semantic Web and Information Systems*, vol. 3, no. 2, pp. 1–11, 2007.

[14] A. Hulth, "Improved automatic keyword extraction given more linguistic knowledge," in *Proc. of the 2003 conference on Empirical methods in natural language processing*, 2003, pp. 216–223.

[15] R. Mihalcea and P. Tarau, "Textrank: Bringing order into texts," in *Conference on Empirical Methods in Natural Language Processing*, 2004.

[16] S. Rose, D. Engel, N. Cramer, and W. Cowley, *Automatic Keyword Extraction from Individual Documents*. John Wiley and Sons, Ltd, March 2010, pp. 1–20.

[17] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of Documentation*, vol. 28, pp. 11–21, 1972.

[18] S. Ceri, M. Negri, and G. Pelagatti, "Horizontal data partitioning in database design," in *Proc. of the 1982 ACM SIGMOD international conference on Management of data*, 1982, pp. 128–136.

[19] N. Leavitt, "Will nosql databases live up to their promise?" *Computer*, vol. 43, no. 2, pp. 12–14, February 2010.

[20] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in *Proc. of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, 2006, pp. 15–15.

[21] N. Santos, O. Pereira, and D. Gomes, "Context storage using nosql," in *Conferência sobre Redes de Computadores*, 2011.

[22] J. Thelen, D. Goense, and K. Langendoen, "Radio wave propagation in potato fields," in *1st Workshop on Wireless Network Measurements*, vol. 2, 2005, pp. 331–338.

[23] A. Baggio, "Wireless sensor networks in precision agriculture," in *ACM Workshop on Real-World Wireless Sensor Networks (REALWSN 2005)*, 2005.