Sensor Search Techniques for Sensing as a Service Architecture for the Internet of Things

Charith Perera, *Student Member, IEEE*, Arkady Zaslavsky, *Member, IEEE*, Chi Harold Liu, *Member, IEEE*, Michael Compton, Peter Christen, and Dimitrios Georgakopoulos, *Member, IEEE*

***Abstract*— The Internet of Things (IoT) is part of the Internet of the future and will comprise billions of intelligent communicating "things" or Internet Connected Objects (ICOs) that will have sensing, actuating, and data processing capabilities. Each ICO will have one or more embedded sensors that will capture potentially enormous amounts of data. The sensors and related data streams can be clustered physically or virtually, which raises the challenge of searching and selecting the right sensors for a query in an efficient and effective way. This paper proposes a context-aware sensor search, selection, and ranking model, called CASSARAM, to address the challenge of efficiently selecting a subset of relevant sensors out of a large set of sensors with similar functionality and capabilities. CASSARAM considers user preferences and a broad range of sensor characteristics such as reliability, accuracy, location, battery life, and many more. This paper highlights the importance of sensor search, selection and ranking for the IoT, identifies important characteristics of both sensors and data capture processes, and discusses how semantic and quantitative reasoning can be combined together. This paper also addresses challenges such as efficient distributed sensor search and relational-expression based filtering. CASSARAM testing and performance evaluation results are presented and discussed.**

***Index Terms*— sensors, search and selection, indexing and ranking, semantic querying, quantitative reasoning, multi-dimensional data fusion.**

I. Introduction

THE number of sensors deployed around the world is increasing at a rapid pace. These sensors continuously generate enormous amounts of data. However, collecting data from all the available sensors does not create additional value unless they are capable of providing valuable insights that will ultimately help to address the challenges we face every day (e.g. environmental pollution management and traffic congestion management). Furthermore, it is also not feasible due to its large scale, resource limitations, and cost factors. When a large number of sensors are available from which to choose, it becomes a challenge and a time consuming task to select the *appropriate*[1] sensors that will help the users to solve their own problems.

The sensing as a service [1] model is expected to be built on top of the IoT infrastructure and services. It also envisions that sensors will be available to be used over the Internet either for free or by paying a fee through midddleware solutions. Currently, several middleware solutions that are expected to facilitate such a model are under development. OpenIoT [2], GSN [3], and xively (xively.com) are some examples. These middleware solutions strongly focus on connecting sensor devices to software systems and related functionalities [2]. However, when more and more sensors get connected to the Internet, the search functionality becomes critical.

This paper addresses the problem mentioned above as we observe the lack of focus on sensor selection and search in existing IoT solutions and research. Traditional web search approach will not work in the IoT sensor selection and search domain, as text based search approaches cannot capture the critical characteristics of a sensor accurately. Another approach that can be followed is that of metadata annotation. Even if we maintain metadata on the sensors (e.g. stored in a sensor's storage) or in the cloud, interoperability will be a significant issue. Furthermore, a user study done by Broring et al. [4] has described how 20 participants were asked to enter metadata for a weather station sensor using a simple user interface. Those 20 people made 45 mistakes in total. The requirement of re-entering metadata in different places (e.g. entering metadata on GSN once and again entering metadata on OpenIoT, etc.) arises when we do not have common descriptions. Recently, the W3C Incubator Group released Semantic Sensor Network XG Final Report, which defines an SSN ontology [5]. The SSN ontology allows describing sensors, including their characteristics. This effort increases the interoperability and accuracy due to the lack of manual data entering. Furthermore, such mistakes can be avoided by letting the sensor hardware manufactures produce and make available sensor descriptions using ontologies so that IoT solution developers can retrieve and incorporate (e.g. mapping) them in their own software system.

Based on the arguments above, ontology based sensor description and data modeling is useful for IoT solutions. This approach also allows semantic querying. Our proposed

Manuscript received July 11, 2013; revised August 7, 2013; accepted September 13, 2013. Date of publication September 20, 2013; date of current version November 26, 2013. This work was supported in part by SSN TCP, CSIRO, Australia and in part by the ICT OpenIoT Project, which is co-funded by the European Commission under the Seventh Framework Program under Grant FP7-ICT-2011-7-287305-OpenIoT. The associate editor coordinating the review of this paper and approving it for publication was Prof. Elena Gaura.

C. Perera, A. Zaslavsky, M. Compton, and D. Georgakopoulos are with the Information and Communication Centre, Commonwealth Scientific and Industrial Research Organisation, Canberra 2601, Australia (e-mail: charith.perera@csiro.au; arkady.zaslavsky@csiro.au; michael.compton@csiro.au; dimitrios.georgakopoulos@csiro.au).

C. H. Liu is with IBM Research, Beijing 100193, China (e-mail: chiliu@cn.ibm.com).

P. Christen is with the Research School of Computer Science, The Australian National University, Canberra 0200, Australia (e-mail: peter.christen@anu.edu.au).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSEN.2013.2282292

[1]We describe the term *appropriate* in Section III.

solution allows the users to express their priorities in terms of sensor characteristics and it will search and select appropriate sensors. In our model, both quantitative reasoning and semantic querying techniques are employed to increase the performance of the system by utilizing the strengths of both techniques.

In this paper, we propose a model that can be adopted by any IoT middleware solution. Moreover, our design can be run faster using *MapReduce* based techniques, something which increases the scalability of the solution. Our contributions can be summarized as follows. We have developed an ontology based context framework for sensors in IoT which allows capturing and modeling context properties related to sensors. This information allows users to search the sensors based on context. We have designed, implemented and evaluated our proposed CASSARAM model and its performance in a comprehensive manner. Specifically, we propose a Comparative-Priority Based Weighted Index (CPWI) technique to index and rank sensors based on the user preferences. Furthermore, we propose a Comparative-Priority Based Heuristic Filtering (CPHF) technique to make the sensor search process more efficient. We also propose a Relational-Expression based Filtering (REF) technique to support more comprehensive searching. Finally, we propose and compare several distributed sensor search mechanisms.

The rest of this paper is structured as follows: In Section II, we briefly review the literature and provide some descriptions of leading IoT middleware solutions and their sensor searching capabilities. Next, we present the problem definitions and motivations in Section III. Our proposed solution, CASSARAM, is presented with details in Section IV. Data models, the context framework, algorithms, and architectures are discussed in this section. The techniques we developed to improve CASSARAM are presented in Section V. In Section VI, we provide implementation details, including tools, software platforms, hardware platforms, and the data sets used in this research. Evaluation and discussions related to the research findings are presented in Section VII. Finally, we present a conclusion and prospects for future research in Section VIII.

II. Background and Related Work

Ideally, IoT middleware solutions should allow the users to express what they want and provide the relevant sensor data back to them quickly without asking the users to manually select the sensors which are relevant to their requirements. Even though IoT has received significant attention from both academia and industry, sensor search and selection has not been addressed comprehensively. Specifically, sensor search and selection techniques using context information [6] have not been explored substantially. A survey on context aware computing for the Internet of Things [6] has recognised sensor search and selection as a critical task in automated sensor configuration and context discovery processes. Another review on semantics for the Internet of Things [7] has also recognised resource (e.g., a sensor or an actuator) search and discovery functionality as one of the most important functionalities that are required in IoT. Barnaghi et al. [7] have highlighted the need for semantic annotation of IoT resources and services. Processing and analysing the semantically annotated data are essential elements to support search and discovery [7]. This justifies our approach of annotating the sensors with related context information and using that to search the sensors. The following examples show how existing IoT middleware solutions provide sensor searching functionality.

Linked Sensor Middleware (*LSM*) [8], [9] provides some sensor selection and searching functionality. However, they have very limited capabilities, such as selecting sensors based on location and sensor type. All the searching needs to be done using SPARQL, which is not user-friendly to non-technical users. Similar to *LSM*, there are several other IoT middleware related projects under development at the moment. *GSN* [3] is a platform aiming at providing flexible middleware to address the challenges of sensor data integration and distributed query processing. It is a generic data stream processing engine. *GSN* has gone beyond the traditional sensor network research efforts such as routing, data aggregation, and energy optimisation. *GSN* lists all the available sensors in a combo-box which users need to select. However, GSN lacks semantics to model the metadata. Another approach is Microsoft *SensorMap* [10]. It only allows users to select sensors by using a location map, by sensor type and by keywords. *xively* (xively.com) is also another approach which provides a secure, scalable platform that connects devices and products with applications to provide real-time control and data storage. This also provides only keyword search. The illustrations of the search functionalities provided by the above mentioned IoT solutions are presented in [11]. Our proposed solution CASSARAM can be used to enrich all the above mentioned IoT middleware solutions with a comprehensive sensor search and selection functionality.

In the following, we briefly describe some of the work done in sensor searching and selection. Truong et al. [12] propose a fuzzy based similarity score comparison sensor search technique to compare the output of a given sensor with the outputs of several other sensors to find a matching sensor. Mayer et al. [13] considers the location of smart things/sensors as the main context property and structures them in a logical structure. Then, the sensors are searched by location using tree search techniques. Search queries are distributively processed in different paths/nodes of the tree. Elahi et al. [14] propose a content-based sensor search approach (i.e. finding a sensor that outputs a given value at the time of a query). *Dyser* is a search engine proposed by Ostermaier et al. [15] for real-time Internet of Things, which uses statistical models to make predictions about the state of its registered objects (sensors). When a user submits a query, *Dyser* pulls the latest data to identify the actual current state to decide whether it matches the user query. Prediction models help to find matching sensors with a minimum number of sensor data retrievals. Very few related efforts have focused on sensor search based on context information. Perera et al. [11] have compared the similarities and differences between sensor search and web service search. It was found that context information has played a significant role in web service search (especially towards web services composition). According to a study in Europe [16], there are over 12,000 working and useful Web services on the

Web. Even in such conditions, choice between alternatives (depending on context properties) has become a challenging problem. The similarities strengthen the argument that sensor selection is an important challenge at the same level of complexity as web services. On the other hand, the differences show that sensor selection will become a much more complex challenge over the coming decade due to the scale of the IoT.

De et al. [17] have proposed a conceptual architecture, an IoT platform, to support real-world and digital objects. They have presented several semantic ontology based models that allow capturing information related to IoT resources (e.g. sensors, services, actuators). However, they are not focused on sensors and the only context information considered is location. In contrast, CASSARAM narrowly focuses on sensors and considers a comprehensive set of context information (see Section IV-F). Guinard et al. [18] have proposed a web service discovery, query, selection, and ranking approach using context information related to the IoT domain. Similarly, TRENDY [19] is a registry-based service discovery protocol based on CoAP (Constrained Application Protocol) [20] based web services with context awareness. This protocol has been proposed to be used in the Web of Things (WoT) domain with the objective of dealing with a massive number of web services (e.g. sensors wrapped in web services). Context information such as hit count, battery, and response time are used to select the services. An interesting proposal is by Calbimonte et al. [21], who have proposed an ontology-based approach for providing data access and query capabilities to streaming data sources. This work allows the users to express their needs at a conceptual level, independent of implementation. Our approach, CASSARAM, can be used to complement their work where we support context based sensor search and they provide access to semantically enriched sensor data. Furthermore, our evaluation results can be used to understand the scalability and computational performance of their working big data paradigm as both approaches use the SSN ontology. Garcia-Castro et al. [22] have defined a core ontological model for Semantic sensor web infrastructures. It can be used to model sensor networks (by extending the SSN ontology), sensor data sources, and the web services that expose the data sources. Our approach can also be integrated into the *uBox* [23] approach, to search *things* in the WoT domain using context information. Currently, *uBox* performs searches based on location tags and object (sensor) classes (types) (e.g. hierarchy local/class/actuator/light).

The Table I summarises the different research efforts that have addressed the challenge of sensor search. Table I lists the efforts and the number of sensors used in their experiments.

TABLE I
NUMBER OF SENSORS USED IN EXPERIMENTAL EVALUATIONS OF DIFFERENT SENSOR SEARCH APPROACHES

Approach	Number of sensors used in experiments
Truong et al. [13]	42
Elahi et al. [14]	250
Ostermaier et al. [15]	385
Mayer et al. [13]	600
Calbimonte et al. [24][2]	1400
LSM [9]	100,000

III. PROBLEM DEFINITION AND MOTIVATION

The problem that we address in this paper can be defined as follows. Due to the increasing number of sensors available, we need to search and select sensors that provide data which will help to solve the problem at hand in the most efficient and effective way. Our objective is not to solve the users problems, but to help them to collect sensor data. The users can further process such data in their own ways to solve their problems. In order to achieve this, we need to search and select sensors based on different pieces of context information. Mainly, we identify two categories of requirements: point-based requirements (non-negotiable) and proximity-based (negotiable) requirements. We examined the problem in detail in [11] by providing real world application scenarios and challenges.

First, there are the point-based requirements that need be definitely fulfilled. For example, if a user is interested in measuring the temperature in a certain location (e.g. Canberra), the result (e.g. the list of sensors) should only contain sensors that can measure temperature. The user cannot be satisfied by being providing with any other type of sensor (e.g. pressure sensors). There is no bargain or compromise in this type of requirement. Location can be identified as a point-based requirement. The second is proximity-based requirements that need be fulfilled in the best possible way. However, meeting the exact user requirement is not required. Users may be willing to be satisfied with a slight difference (variation). For example, the user has the same interest as before. However, in this situation, the user imposes proximity-based requirements in addition to their point-based requirements. The user may request sensors having an accuracy of around 92%, and reliability 85%. Therefore, the user gives the highest priority to these characteristics. The user will accept sensors that closely fulfil these requirements even though all other characteristics may not be favourable (e.g. the cost of acquisition may be high and the sensor response may be slow). It is important to note that users may not be able to provide any specific value, so the system should be able to understand the user's priorities and provide the results accordingly, by using comparison techniques.

Another motivation behind our research are statistics and predictions that show rapid growth in sensor deployment related to the IoT and Smart Cities. It is estimated that today there about 1.5 billion Internet-enabled PCs and over 1 billion Internet-enabled mobile phones. By 2020, there will be 50 to 100 billion devices connected to the Internet [25]. Furthermore, our work is motivated by the increasing trend of IoT middleware solutions development. Today, most of the leading midddleware solutions provide only limited sensor search and selection functionality, as mentioned in Section II.

We highlight the importance of sensor search functionality using current and potential applications. Smart agriculture [26] projects such as Phenonet [27] collects data from thousands of sensors. Due to heterogeneity, each sensor may have different context values, as mentioned in Section IV-F. Context

TABLE II
COMMON ALGORITHMIC NOTATION TABLE

Symbol	Definition
$\mathbb{O}$	*Ontology* consists of sensor descriptions and context property values related to all sensors
$\mathbb{P}$	*UserPrioritySet* contains user priority value for all context properties
$\mathbb{Q}$	*Query* consists of point-based requirements expressed in SPARQL
N/N_{All}	Number of sensors required by the user / Total number of sensors available
$\mathbb{S}_{Filtered}$	This contains the results of the query $\mathbb{Q}$
$\mathbb{S}_{Results}$	*ResultsSet* contains selected number of sensors
$\mathbb{S}_{Indexed}$	*IndexedSensorSet* store the index values of the sensors
$\mathbb{M}$	Multidimensional space where each context property is represented by a dimension and sensors are plotted
$\mathbb{UI}$	*UserInput* consists of input values provided by the users via the user interface
$\mathbb{SC}/SC$	Values of all the sliders / Value of a slider
$\mathbb{P}^w$	This contains user priority value converted into weights using normalization
p_i/p_i^w	Value of i^{th} context property / Value of i^{th} context property in normalized form
$\mathbb{CP}/CP$	*ContextPropertySet* consists of all context information / value of i^{th} context property
$\mathbb{NCP}$	NormalizedContextPropertySet
M	Margin of error
S_j	This is the j^{th} sensor
$CP_i^{S_j}$	CP value of i^{th} property of j^{th} sensor.
CP^{ideal}	CP values of the ideal sensors that user prefers

information can be used to selectively select sensors depending on the requirements and situations. For example, CASSARAM helps to retrieve data only from sensors which have more energy remaining when alternative sensors are available. Such action helps to run the entire sensor network for a much longer time without reconfiguring and recharging. The sensing as a service [28] architectural model envisions an era where sensor data will be published and sold through the cloud. Consumers (i.e., users) will be allowed to select a number of sensors and retrieve data for some period as specified in an agreement by paying a fee. In such circumstances, allowing consumers to select the sensors they want based on context information is critical. For example, some consumers may be willing to pay more for highly accurate data (i.e., highly accurate sensors) while others may be willing to pay less for less accurate data, depending on their requirements, situations, and preferences.

IV. CONTEXT-AWARE SENSOR SEARCH, SELECTION AND RANKING MODEL

In this section, we present the proposed sensor selection approach step by step in detail. First, we provide a high-level overview of the model, which describes the overall execution flow and critical steps. Then, we explain how user preferences are captured. Next, the data representation model and proposed extensions are presented. Finally, the techniques of semantic querying and quantitative reasoning are discussed with the help of some algorithms. All the algorithms presented in this paper are self-explanatory and the common algorithmic notations used in this paper are presented in Table II.

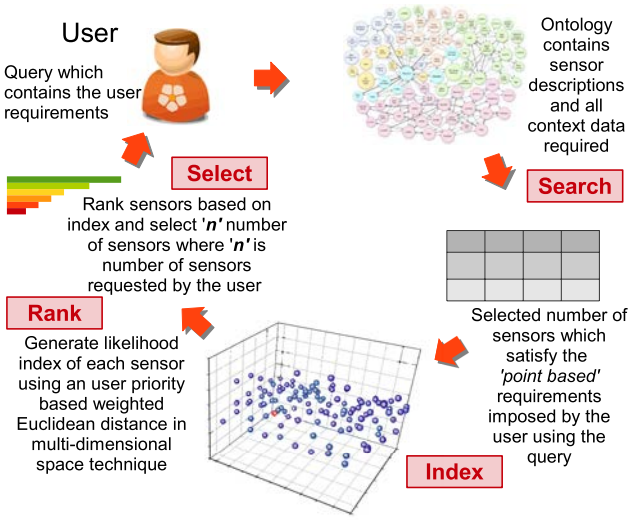

Fig. 1. High level Overview of CASSARAM.

A. High-Level Model Overview

The critical steps of CASSARAM are presented in Fig. 1. As we mentioned earlier our objective is to allow the users to search and select the sensors that best suit their requirements. In our model, we divide user requirements into two categories (from the user's perspective): *point-based requirements* and *proximity-based requirements*, as discussed in Section III. Algorithm 1 describes the execution flow of CASSARAM. At the beginning, CASSARAM identifies the point-based requirements, the proximity-based requirements, and the user priorities. First, users need to select the point-based requirements. For example, a user may want to collect sensor data from 1,000 temperature sensors deployed in Canberra. In this situation, the sensor type (i.e., temperature), location (i.e., Canberra) and number of sensors required (i.e., 1,000) are the point-based requirements. Our CASSARAM prototype tool provides a user interface to express this information via SPARQL queries. In CASSARAM, any context property can become a point-based requirement. Next, users can define the proximity-based requirements. All the context properties we will present in Section IV-F are available to be defined in comparative fashion by setting the priorities via a slider-based user interface, as depicted in Fig. 2. Next, each sensor is plotted in a multi-dimensional space where each dimension represents a context property (e.g. accuracy, reliability, latency). Each dimension is normalized [0,1] as explained in Algorithm 3. Then, the Comparative-Priority Based Weighted Index (CPWI) is generated for each sensor by combining the user's priorities and context property values as explained in Section IV-E. The sensors are ranked using the CPWI and the number of sensors required by the user is selected from the top of the list.

B. Capturing User Priorities

This is a technique we developed to capture the user's priorities through a user interface, as shown in Fig. 2. CASSARAM allows users to express which context property is more important to them, when compared to others. If a user does not want a specific context property to be considered in the indexing process, they can avoid it by not selecting the

Algorithm 1 Execution Flow of CASSARAM

Require: $(\mathbb{O})$, $(\mathbb{P})$, $(\mathbb{Q})$, (N), $(\mathbb{M})$.
Output: $\mathbb{S}_{Results}$
$\mathbb{S}_{Filtered} \leftarrow$ queryOntology$(\mathbb{O}, \mathbb{Q})$
if cardinality$(\mathbb{S}_{Filtered}) < N$ **then**
  return $\mathbb{S}_{Results} \leftarrow \mathbb{S}_{Filtered}$
else
  $\mathbb{P} \leftarrow$ capture user priorities$(\mathbb{UI})$
  $\mathbb{M} \leftarrow$ Plot sensors in multidimensional space$(\mathbb{S}_{Results})$
  $\mathbb{S}_{Indexed} \leftarrow$ calculate CPWI$(\mathbb{S}_{Results}, \mathbb{M})$
  $\mathbb{S}_{Results} \leftarrow$ rank sensors$(\mathbb{S}_{Indexed})$
  $\mathbb{S}_{Results} \leftarrow$ select sensors$(\mathbb{S}_{Results}, N)$
  return $\mathbb{S}_{Results}$
end if

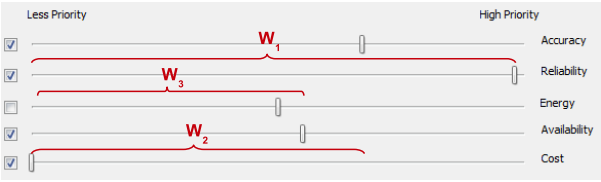

Fig. 2. A weight of W_1 is assigned to the *reliability* property. A weight of W_2 is assigned to the *Accuracy* property. A weight of W_3 is assigned to the *availability* property. A weight of W_4, the default weight, is assigned to the *cost* property. High priority means always favoured, and low priority means always disfavoured. For example, if the user makes *cost* a high priority (more towards the right), that means CASSARAM tries to find the sensors that produce data at the lowest cost. Similarly, if the user makes *accuracy* a high priority, that means CASSARAM tries to find the sensors that produce data with high accuracy.

check-box correlated with that specific context property. For example, according to Fig. 2, *energy* will not be considered when calculating the CPWI. This means the user is willing to accept sensors with any *energy* consumption level. Users need to position the slider of each context property if that context property is important to them. The slider scale begin from 1, which means no priority (i.e., the left corner). The highest priority can be set by the user as necessary with the help of a *scaler*, where a higher scale makes the sliders more sensitive (e.g. $10^2 = 1$ to $100, 10^3, 10^4$). Algorithm 2 describes the user priority capturing process.

As depicted in Fig. 2, if the user wants more wieght to be placed on the reliability of a sensor than on its accuracy, the reliability slider need to be placed further to the right than the accuracy slider. A weight is calculated for each context property. Therefore, higher priority means higher weight. As a result, sensors with high reliability and accuracy will be ranked highly. However, those sensors may have high costs due to the low priority placed on cost.

C. Data Modeling and Representation

In this paper, we employed the Semantic Sensor Network Ontology (SSN) [5] to model the sensor descriptions and context properties. The main reasons for selecting the SSN ontology are its interoperability and the trend towards ontology usage in the IoT and sensor data management domain.

Algorithm 2 User Priority Capturing

Require: $(\mathbb{UI})$, $(\mathbb{SC})$
Output: $\mathbb{P}^w$
$\mathbb{P} \leftarrow$ extract user priorities$(\mathbb{UI})$
$SC_{Highest} \leftarrow$ get maximum priority$(\mathbb{SC})$
$SC_{Lowest} \leftarrow$ get minimum priority$(\mathbb{SC})$
$SC_{Range} \leftarrow SC_{Highest} - SC_{Lowest}$
for each context property priority $p_i \in \mathbb{P}$ **do**
  $p_i^w \leftarrow (p_i \div S_{Range})$
  if $p_i^w \geq 0$ **then**
    add p_i^w to $\mathbb{P}^w$
  else
    continue
  end if
end for
return $\mathbb{P}^w$

A comparison of different semantic sensor ontologies is presented in [29]. The SSN ontology is capable of modeling a significant amount of information about sensors, such as sensor capabilities, performance, the conditions in which it can be used, etc. The details are presented in [5]. The SSN ontology includes the most common context properties, such as accuracy, precision, drift, sensitivity, selectivity, measurement range, detection limit, response time, frequency and latency. However, the SSN ontology can be extended *unlimitedly* by a categorization with three classes: *measurement property*, *operating property*, and *survival property*. We depict a simplified segment of the SSN ontology in Fig. 3. We extend the *quality class* by adding several sub-classes based on our context framework, as listed in Section IV-F. All context property values are stored in the SSN ontology in their original measurement units. CASSARAM normalizes them on demand to [0,1] to ensure consistency. Caching techniques can be used to increase the execution performances. Due to technological advances in sensor hardware development, it is impossible to statically define upper and lower bounds for some context properties (e.g. battery life will be improved over time due to advances in sensor hardware technologies). Therefore, we propose Algorithm 3 to dynamically normalize the context properties.

D. Filtering Using Querying Reasoning

Once the point-based requirements of the user have been identified, they need to be expressed using SPARQL. Semantic querying has weaknesses and limitations. When a query becomes complex, the performance decreases [30]. Relational expression based filtering can also be used; however, using it will increase the computational requirements. Further explanations are presented in Section V-B. Any of the context properties identified in Section IV-F can become point-based requirements and need to be represented in SPARQL. This step produces $\mathbb{S}_{Filtered}$, where all the sensors satisfy all the point-based requirements.

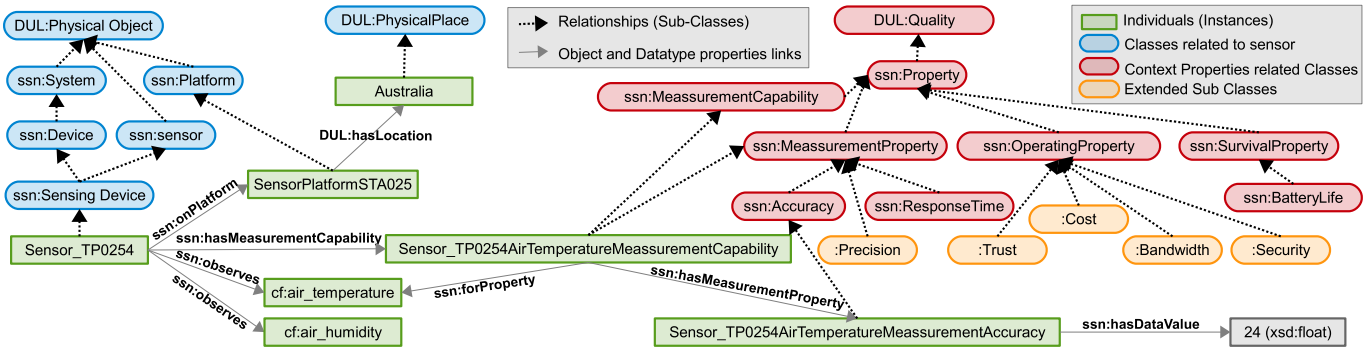

Fig. 3. Data model used in CASSARAM. In SSN ontology, *sensors* are not constrained to physical sensing devices; rather a sensor is anything that can estimate or calculate the value of a phenomenon, so a device or computational process or combination could play the role of a sensor. A *sensing device* is a device that implements sensing [5]. *Sensing device* is also a sub class of *sensor*. By following above definition, our focus is on sensors. CF (Climate and Forecast) ontology is a domain specific external ontology. DOLCE+DnS Ultralite (DUL) ontology provides a set of upper level concepts that can be the basis for easier interoperability among many middle and lower level ontologies. More details are provided in [5].

Algorithm 3 Flexi-Dynamic Normalization

Require: $(\mathbb{CP})$, $(\mathbb{S})$, (cp_i),
Output: $\mathbb{NCP}$
$cp_i^{S_j} \leftarrow$ receive new property value*
$cp_i^{highest} \leftarrow$ retrieve highest$(\mathbb{CP})$
$cp_i^{lowest} \leftarrow$ retrieve lowest$(\mathbb{CP})$
if $cp_i^{highest} < cp_i^{S_j}$ **then**
  $cp_i^{highest} \leftarrow cp_i^{S_j}$
  for each $cp_i^{S_j} \in \mathbb{CP}, \mathbb{S}$ **do**
    update$(\mathbb{NCP}) \leftarrow [\frac{(cp_i^{S_j} - cp_i^{lowest})}{(cp_i^{highest} - cp_i^{lowest})}]$
  end for
else
  update$(\mathbb{NCP}) \leftarrow [\frac{(cp_i^{S_j} - cp_i^{lowest})}{(cp_i^{highest} - cp_i^{lowest})}]$
end if
return $\mathbb{NCP}$
*sensors registered in the IoT middleware

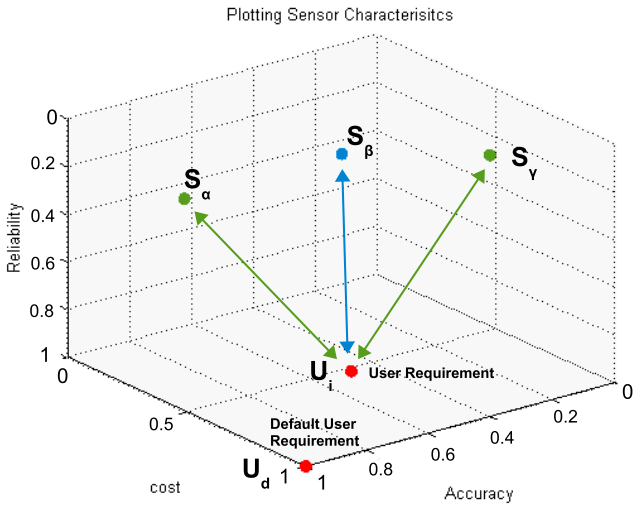

Fig. 4. Sensors plotted in three-dimensional space for demonstration purposes. S_α, S_β, and S_γ represent real sensors. U_i represent the user preferred sensor. U_d represent the default user preferred sensor. CPWI calculate weighted distance between $S_{j=\alpha||\beta||\gamma}$ and $U_{i||d}$. Shortest distance means sensor will rank higher because it is close to the user requirement.

E. Ranking Using Quantitative Reasoning

In this step, the sensors are ranked based on the proximity-based user requirements. We developed a weighted Euclidean distance based indexing technique, called the Comparative-Priority Based Weighted Index (CPWI), as follows.

$$(CPWI) = \sqrt{\sum_{i=1}^{n} \left[W_i (U_i^d - S_i^\alpha)^2\right]}$$

First, each sensor is plotted in multi-dimensional space where each context property is represented by a dimension. Then, users can plot an ideal sensor in the multi-dimensional space by manually entering context property values as illustrated in Fig. 4 by U_i. By default, CASSARAM will automatically plot an ideal sensor as depicted in U_d (i.e., the highest value for all context properties). Next, the priorities defined by the user are retrieved. Based on the positions of the sliders (in Fig. 2), weights are calculated in a comparative fashion. Algorithm 4 describes the indexing process. It calculates the CPWI and ranks the sensors using reverse-normalised techniques in descending order. CASSARAM selects N sensors from the top.

F. Context Framework

After evaluating a number of research efforts conducted in the quality of service domain relating to web services [31], mobile computing [32], mobile data collection [33], and sensor ontologies [5], we extracted the following context properties to be stored and maintained in connection with each sensor. This information helps to decide which sensor is to be used in a given situation. We adopt the following definition of *context* for this paper. "Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." [34]. CASSARAM has no limitations on the number of context properties that can be used. More context information can be added to the following list as necessary. Our context framework comprises availability, accuracy, reliability, response time, frequency, sensitivity, measurement range, selectivity, precision, latency, drift, resolution, detection limit, operating power range, system

Algorithm 4 Comparative-Priority Based Weighted Index

Require: $(\mathbb{P}^w)$, $(\mathbb{CP})$, $(\mathbb{S}^{Indexed})$, $(\mathbb{P}^{S_j})$, $(\mathbb{UI})$
Output: $\mathbb{S}_{Ranked}$
$CP^{ideal} \leftarrow$ proximity based requirements$(\mathbb{UI})$
plot on multi-dimensional space(CP^{ideal})
for each sensor $S_j \in \mathbb{S}$ **do**
  plot on multi-dimensional space(CP^{S_j})
end for
Indexing Formula (for S^α) = $\sqrt{\sum_{i=1}^{n}\left[W_i(U_i^d - S_i^\alpha)^2\right]}$
for each sensor $s^j \in \mathbb{S}$ **do**
  $\mathbb{S}^{Indexed} \leftarrow$ calculate index$(\mathbb{P}^{S_j}, \mathbb{P}^w)$
end for
$\mathbb{S}_{Ranked} \leftarrow$ reversed normalized ranking*$(\mathbb{S}^{Indexed})$ *i.e.: lowest value is ranked higher which represents the weighted distance between use preferred sensor and the real sensors
return $\mathbb{S}_{Ranked}$

(sensor) lifetime, battery life, security, accessibility, robustness, exception handling, interoperability, configurability, user satisfaction rating, capacity, throughput, cost of data transmission, cost of data generation, data ownership cost, bandwidth, and trust.

V. Improving Scalability and Efficiency

In this section, we present three approaches that improve the efficiency and the capability of CASSARAM. First, we propose a heuristic approach that can handle a massive number of sensors by trading off with accuracy. Second, we propose a relational-expression based filtering technique that saves computational resources. Third, we tackle the challenge of distributed sensor search and selection.

A. Comparative-Priority Based Heuristic Filtering (CPHF)

The solution we discussed so far works well with small number of sensors. However, model becomes inefficient when the number of sensors available to search increases. Let us consider an example to identify the inefficiency. Assume we have access to one million sensors. A user wants to select 1,000 sensors out of them. In such situation, CASSARAM will index and rank one million sensors using proximity-based requirements provided by the user and select top 1,000 sensors. However, indexing and ranking all possible sensors (in this case one million) is inefficient and wastes significant amount of computational resources. Furthermore, CASSARAM will not be able to process large number of user queries due to such inefficiency. We propose a technique called Comparative-Priority Based Heuristic Filtering (CPHF) to make CASSARAM more efficient. The execution process is explained in Algorithm 5. The basic idea is to remove sensors that are positioned far away from user defined ideal sensor and reduce the number of sensors that need to be indexed and ranked. Fig. 5 illustrates the CPHF approach with a sample scenario. The CPHF approach can be explained as follows. First, all the eligible sensors are ranked in descending order of the highest

Algorithm 5 Comparative-Priority Based Heuristic Filtering

Require: $(\mathbb{O})$, $(\mathbb{P})$, $(\mathbb{Q})$, (N), $(M\%)$
Output: $\mathbb{S}_{Filtered}$
$\mathbb{S} \leftarrow$ query ontology$(\mathbb{O}, \mathbb{Q})$
$\mathbb{P}^w \leftarrow$ get weighted priorities$(\mathbb{P})$
$\mathbb{P}^{Percentages} \leftarrow$ convert weights to percentages$(\mathbb{P}^w)$
$N_{All} \leftarrow$ total numberof available sensors$(\mathbb{O}, \mathbb{Q})$
$N \leftarrow$ required number of sensors$(\mathbb{UI})$
$N_{Removable} \leftarrow (N_{All} - N)$
$\mathbb{P}^{Percentages}_{ordered} \leftarrow$ descending order$(\mathbb{P}^{Percentages})$
for each priority percentage $p \in \mathbb{P}^{Percentages}_{ordered}$ **do**
  $\mathbb{S}_{Filtered} \leftarrow$ Query $\mathbb{S}^{Filtered}$ and ordered by p
  Remove $N_{Removable} \times (100 - M)$ sensors from bottom.
end for
return $\mathbb{S}_{Filtered}$

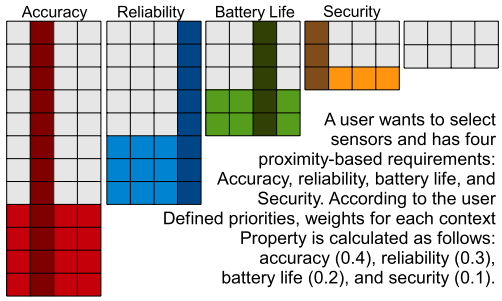

Fig. 5. Visual illustration of Comparative-Priority Based Heuristic Filtering.

weighted context property (in this case accuracy). Then, 40% (from $N_{Removable}$) of the sensors from the bottom of the list need to be removed. Next, the remaining sensors need to be ordered in descending order of the next highest weighted context property (in this case reliability). Then, 30% (from $N_{Removable}$) of the sensors from the bottom of the list need to be removed. This process needs to be applied for the remaining context properties as well. Finally, the remaining sensors need to be indexed and ranked. This approach dramatically reduces the indexing and ranking related inefficiencies. Broadly, this category of techniques are called *Top-K selection* where top sensors are selected in each iteration. The efficiency of this approach is evaluated and discussed in Section VII.

B. Relational-Expression Based Filtering (REF)

This section explains how computational resources can be saved and how to speed up the sensor search and selection process by allowing the users to define preferred context property values using relational operators such as $<, >, \leq$, and $\geq$. For example, users can define an upper bound, lower bound, or both, using relational operators. All context properties defined by relational operators, other than the equals sign $(=)$, are considered to be semi-non-negotiable requirements. According to CASSARAM, non-negotiable as well as semi-non-negotiable requirements are defined using semantic queries. Let us consider a scenario where a user wants to select sensors that have 85% accuracy. However,

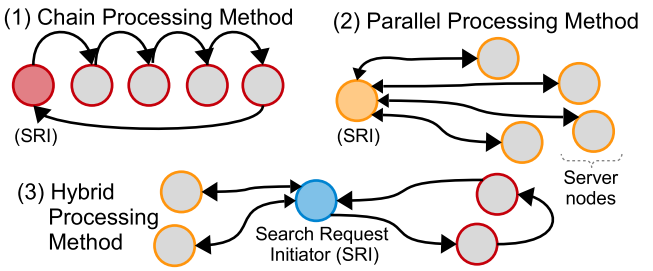

Fig. 6. Distributed Processing Approaches for CASSARAM.

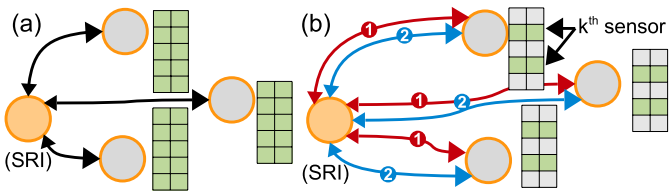

Fig. 7. Optimization: (a) wihout k-extension and (b) with k-extension.

the user can be satisfied by providing sensors with accuracy between 70% and 90%. Such requirements are called semi-non-negotiable requirements. Defining such a range helps to ignore irrelevant sensors during the semantic querying phase without even retrieving them to the CPWI generating phase, and this saves computational resources. Even though users may define ranges, the sensors will be ranked considering the user's priorities by applying the same concepts and rules as explained in Section IV. The efficiency of this approach is evaluated in Section VII.

C. Distributed Sensor Searching

We have explained how CASSARAM works in an isolated environment without taking into consideration the distributed nature of the problem. Ideally, we expect that not all sensors will be connected to one single server (e.g., a single middleware instance). Similarly, it is extremely inefficient to store complete sensor descriptions and related context information in many different servers in a redundant way. Ideally, each IoT middleware instance should keep track of the sensors that are specifically connected to them. This means that each server knows only about a certain number of sensors. However, in order to deal with complex user requirements, CASSARAM may need to query multiple IoT middleware instances to search and select the suitable sensors. Let us consider a scenario related to the smart agriculture domain [26]. A scientist wants to find out whether his experimental crops have been infected with a disease. His experimental crops are planted in fields distributed across different geographical locations in Australia. Furthermore, the sensors deployed in the fields are connected to different IoT middleware instances, depending on the geographical location. In order to help the user to find the appropriate sensors, CASSARAM needs to query different servers in a distributed manner. We explored the possibilities of performing such distributed queries efficiently. We identified three different ways to search sensors distributively, depending on how the query/data would be transferred over the network (i.e., path), as depicted in Fig. 6. We also identified their strengths, weaknesses, and applicability to different situations.

1) Chain Processing: Data is sent from one node to another sequentially as depicted in Fig. 6(a). First, a user defines his requirements using an IoT middleware instance (e.g. GSN installed in a particular server). Then, this server becomes the search request initiator (SRI) for that specific user request. The SRI processes the request and selects the 100 most appropriate sensors. Then, the information related selected sensors (i.e. the unique IDs of the sensors and respective CPWIs) is sent to the next server node. The second node (i.e., that next node) merges the incoming sensor information with the existing sensor descriptions and performs the sensor selection algorithm and selects the 100 best sensors. This pattern continues until the sensor request has visited all the server nodes. This method saves communication bandwidth by transferring only the most essential and minimum amount of data. In contrast, due to a lack of parallel processing, the response time could be high.

2) Parallel Processing: The SRI parallelly sends each user search request to all available nodes. Then, each sensor node performs the sensor searching algorithm at the same time. Each node selects the 100 most appropriate sensors and returns the information related selected sensors to the SRI. In circumstances where we have 2500 server nodes, the amount of data (2500×100) received by the SRI could be overwhelming, which would waste the communication bandwidth. The SRI processes the sensor information (2500×100) and selects the final 100 most appropriate sensors. This approach becomes inefficient when N becomes larger.

3) Hybrid Processing: By observing the characteristics of the previous two methods, it is obvious that the optimal distributed processing strategy should employ both chain and parallel processing techniques. There is no single method that works efficiently for all types of situations. An ideal distributed processing strategy for each situation needs to be designed and configured dynamically depending on the context, such as the types of the devices, their capabilities, bandwidth available, and so on.

We can improve the efficiency of the above methods as follows. In the parallel processing method, each node sends information related to N sensors to the SRI as depicted in Fig. 7(a). However, at the end, the SRI may only select N sensors (in total) despite its having received a significant amount of sensor related information ($N \times number of nodes$). Therefore, the rest of the data $[(N \times number of nodes) - N]$ received by the SRI would be wasted. For example, let us assume that a user wants to select 10,000 sensors. Assuming that there are 2500 server nodes, the SRI may receive a significant amount of sensor information ($10,000 \times 2500$). However, it may finally select only 10,000 sensors. We propose the following method to reduce this wastage, depicted in Fig. 7(b).

In this method, the SRI forwards the search request to each server node parallelly, as depicted in step (1) in Fig. 7b. Each node selects the 10,000 most appropriate sensors. Without sending information about these 10,000 sensors to the SRI, each server node sends only information about the kth sensor (the UID and CPWI of every kth sensor). (I.e., If $k = 1,000$, then the server node sends only the 1000th, 2000th, 3000th,

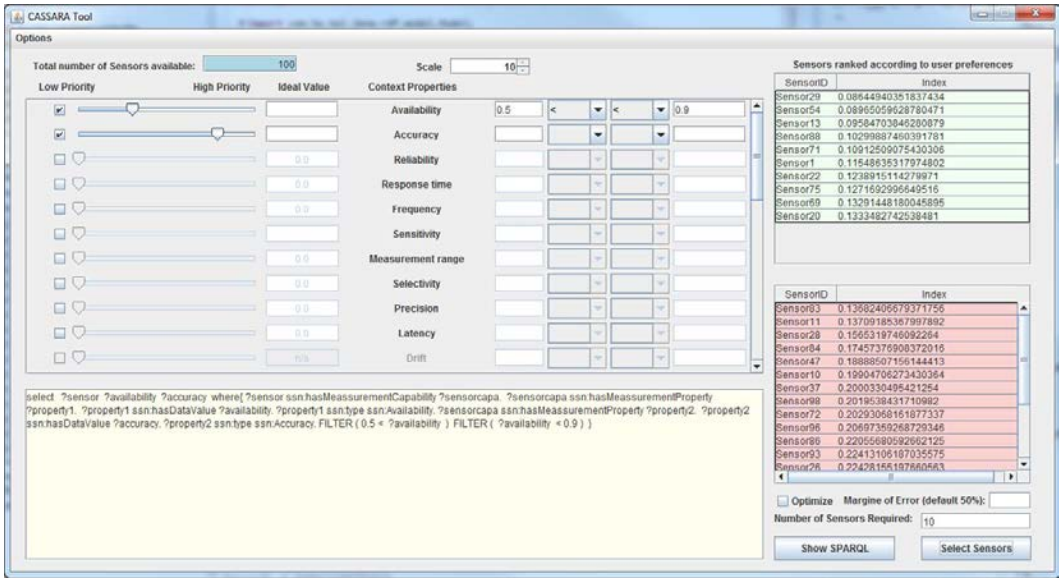

Fig. 8. First, users need to select, in the UI, the context properties about which they are concerned. Then, users need to set the scale. The slider becomes more sensitive when the scale is increased. Next, the slider attached to each context property needs to be positioned to express its priority. The ideal value related to each context property can be entered. The values can be entered in native measurement units (e.g., accuracy in percentage, latency in milliseconds). All the values are normalized by CASSARAM. The default is *'best possible'* (i.e., highest accuracy, lowest cost, lowest latency). Later, users can decide whether to use the optimization functionality or not, by selecting that option. Users can also define the margin of error as a percentage (the default is 50%). Based on the user's preferences, CASSARAM generates the SPARQL appropriately. Finally, users need to specify the number of sensors they require.

…10,000th sensors). Therefore, instead of sending 10,000 records, now each server node returns only 10 records. Once the SRI receives the sensor information from all the server nodes, it processes and decides which portions need to be retrieved. Then, the SRI sends requests back to the server nodes and now each node returns the exact portion specified by the SRI (e.g. the 5th server node may return only the first 2000 sensors instead of sending 10,000 sensors) as depicted in (2). In this method, k plays a key role and has a direct impact on the efficiency. k needs to be chosen by considering N as well as other relevant context information as mentioned earlier. For example, if we use a smaller k, then information about more sensors would be sent to the SRI during step (1), but with less wastage in step (2). In contrast, if we use a larger k, then less information would be sent to the SRI during step (1), but there would be comparatively more wastage in step (2). Furthermore, machine learning techniques can be used to customize the value of k for each server node, depending on the user's request and context information, such as the types of the sensors, energy, bandwidth availability, etc. The suitability of each approach is discussed in Section VII-B.

VI. Implementation and Experimentation

In this section, we describe the experimental setup, datasets used, and assumptions. The experimental scenarios we used are explained at the end. The discussions related to the experiments are presented in Section VII.

We analysed and evaluated the proposed model using a prototype, called *'CASSARA Tool'*, which we developed using Java. The user interface of *"CASSARA Tool"* is presented in Fig. 8 with a self-explanatory description. The data was stored in a MySQL database. Our tool allows capturing user preferences and the priorities of the various context properties of the sensors. We used a computer with an Intel(R) Core i5-2557M 1.70GHz CPU and 4GB RAM to evaluate our proposed model. We also reproduced the experimentations using a higher-end computer with more CPU and RAM and the results showed that the graphs are similar in shape though the exact values are different. In order to perform mathematical operations such as a Euclidean distance calculation in multi-dimensional space, we used the Apache Commons mathematics [35] library. It is an open source optimized library of lightweight, self-contained mathematics and statistics components, addressing the most common problems not available in the Java programming language. As we used a Semantic Sensor Ontology (SSN) [5] to manage the sensor descriptions and related data, we employed open source Apache Jena API [36] to process and manipulate the semantic data. Our evaluation used a combination of real data and synthetically generated data. We collected environmental linked data from the Bureau of Meteorology [37] and data sets from both the Phenonet project [27] and the Linked Sensor Middleware (LSM) project [8], [9]. The main reasons for combining the data were the need for a large amount of data and the need to control different aspects (e.g., the context information related to the sensors needed to be embedded into the data set, because real data that matches our context framework is not available in any public data sets at the moment) to better understand the behaviour of CASSARAM in different IoT related real world situations and scenarios where real data is not available. We make the following assumptions in our work. We assume that the sensor descriptions and context information related to the sensors have already been retrieved from the sensor manufacturers in terms of ontologies, and been into the SSN ontology. Similarly, we assume that the context

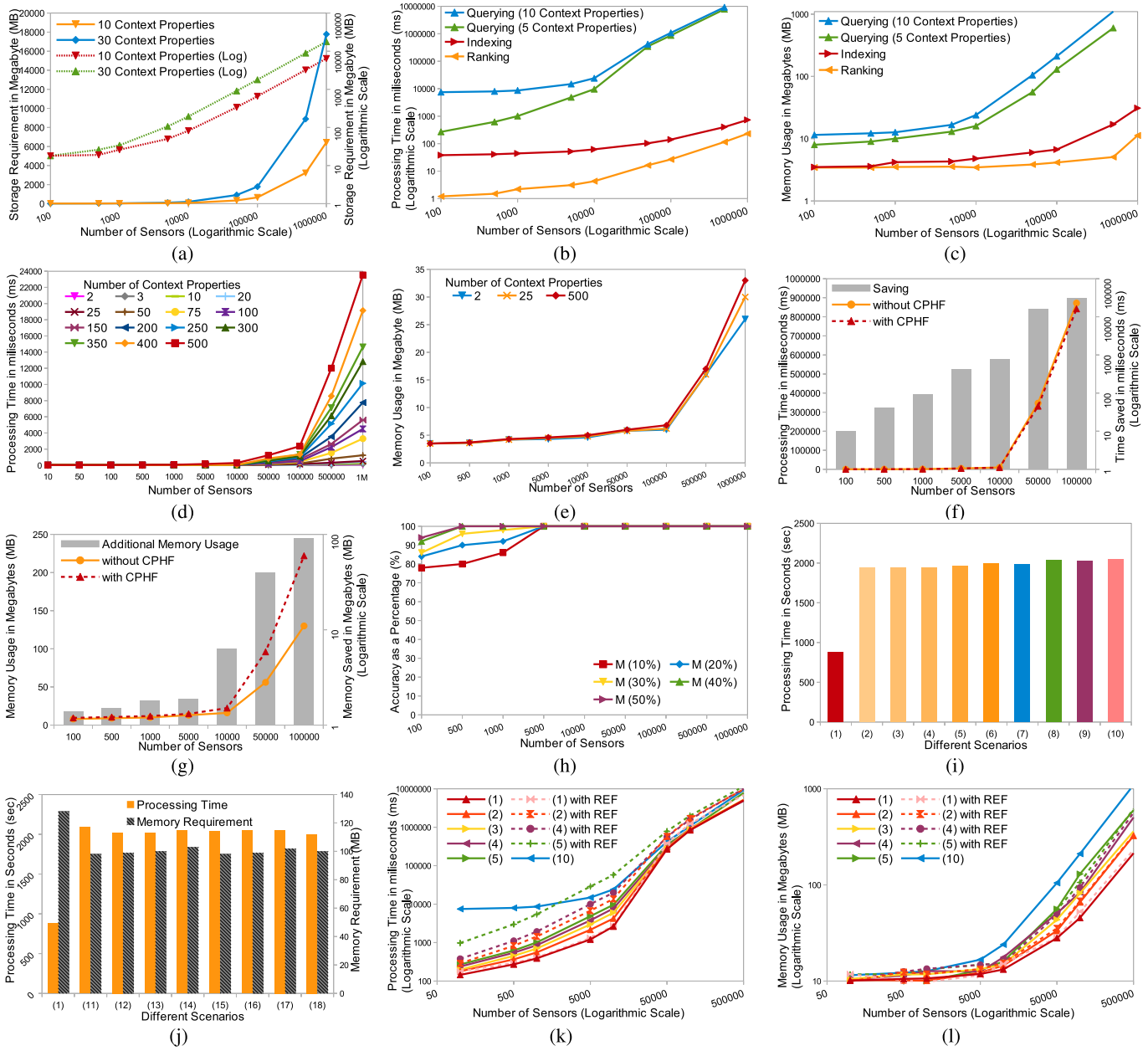

Fig. 9. Experimental results of CASSARAM. Processing time, memory usage, accuracy under different conditions (e.g. number of sensors, number of context properties) are measured in order to examine CASSARAM's efficiency and scalability.

data related to the sensors, such as accuracy, reliability, etc., have been continually monitored, measured, managed, and stored in the SSN ontology by the software systems. In order to evaluate the distributed processing techniques, we proposed an experimental test involving four computational nodes. All the nodes are connected to a private organizational network (i.e., The Australian National University IT Network). The hardware configurations of the three additional devices are as follows: (1) Intel Core i7 CPU with 6GB RAM, (2) Intel Core i5 CPU with 4GB, and (3) Intel Core i7 with 4G. The details are presented in Section VII-B.

We evaluated the performance of CASSARAM using different combinations of relational operators, such as $<, >, =, \leq, \geq$. The scenarios numbered in Figs. 9i–9l correspond to the scenario numbers listed below. All the experiments retrieve five context properties. (1) Do not use any relational operator. (2) 1 out of 5 context properties are restricted by $\geq$ (e.g., the accuracy is to be greater than 80%) (3) 2 out of 5 (e.g., the accuracy is to be greater than 80% AND reliability greater than 85%), (4) 3 out of 5, (5) 4 out of 5. All 5 context properties are restrained (6) by $\geq$, (7) by $\leq$, (8) by $=$, (9) by $<$, (10) by $>$. (11) 1 out of 5 context properties are restricted by two relational operators (e.g., the accuracy is to be greater than $\geq$ 80% AND less than $\leq$ 95%), (12) 2 out of 5, (13) 3 out of 5, (14) 4 out of 5; All 5 context properties are restrained (15) by $\leq$ and $\geq$, (16) by $<$ and $>$. We increased the number of restrictions imposed using additional relational operators. (17) defined two ranges for each context properties (e.g., (accuracy $\geq$ 80% AND $\leq$ 95%) OR (accuracy $\geq$ 50% AND $\leq$ 60%)). (18) defined three ranges.

VII. Evaluation and Discussion

We evaluated CASSARAM using different methods and parameters as depicted in Figs. 9i–9l. In this section, we explain the evaluation criteria which we used for each experiment and discuss the lessons we learned. Fig. 9a shows how the storage requirement varies depending on the number of sensor descriptions. We stored the data according to the SSN ontology, as depicted in Fig. 3. We conducted two experiments where we stored 10 context properties and 30 context properties from the context framework we proposed in Section IV-F. To store one million sensor descriptions, it took 6.4 GB (10 context properties) and 17.8 GB (30 context properties). It is evident that the storage requirements are correlated with the number of triples: a single triple requires about 0.193 KB storage space (for 100,000+ sensors). Though storage hardware is becoming cheaper and available in high capacities, the number of context properties need to store should be decided carefully in order to minimize the storage requirements, especially when the number of sensor is in the billions.

Fig. 9b shows how much time it takes to select sensors as the number of sensors increases. Each step (i.e., searching, indexing and ranking) has been measured separately. Semantic querying requires significantly more processing time than indexing and ranking. Furthermore, as the number of context properties retrieved by a query increases, the execution time also increases significantly. Furthermore, it is important to note that MySQL can join only 61 tables, which only allows retrieving a maximum of 10 context properties from the SSN ontology data model. Using alternative data storage or running multiple queries can be used to overcome this problem. Similarly, it is much more efficient to run multiple queries than to run a single query if the number of sensors is less than 10,000 (e.g., 8 ms to retrieve 5 context properties and 24 ms to retrieve 10 context properties when querying 10,000 sensors). In addition, Fig. 9c shows how much memory is required to select sensors as the number of sensors increases. It is evident that having more context properties requires having more memory. The memory requirements for querying do not change much up to 10,000 (ranging from 10 MB to 25 MB). When the number of sensors exceeds 10,000, the memory requirements grow steadily, correlated with the number of sensors. In comparison, indexing and ranking require less memory.

Fig. 9d shows the processing time taken by the sensor indexing process as the number of context properties and the number of sensors increase. Reducing the number of sensors needing to be indexed below 10,000 allows speeding up CASSARAM. The processing time starts to increases significantly after 100,000 sensors. Similarly, Fig. 9e shows the memory usage by the sensor indexing process as the number of context properties and sensors increases. Even though the memory requirements increase slightly, the actual increase is negligible when the number of sensors is still less than 100,000. After that, the memory requirements increase substantially, but are still very small compared to the computational capabilities of the latest hardware. Furthermore, the number of context properties involved does not have any considerable impact during the indexing process. The differences only become visible when the number of sensors reaches one million. Still, the memory required by the process is 30 MB. Java garbage collection performs its task more actively when processing large numbers of sensors, which makes the difference invisible.

Fig. 9f and 9g compare the time taken by the sensor selection process and the memory it requires, with and without the CPHF algorithm, as the number of sensors increases. The number of sensors that the user requires is kept at 50 in all experiments ($N = 50$). Five context properties are retrieved, indexed, and ranked. The complexity of CPHF (due to the SPARQL subqueries) has not affected significantly the total processing time of CASSARAM. Instead, CPHF has saved some time in the indexing and ranking phases. In contrast, CPHF requires more memory when querying, due to its complexity. However, it requires significantly less memory when transferring data to the next phase for indexing. Therefore, CPHF is efficient as it does not require holding millions of pieces of sensor information in multiple phases in CASSARAM. Furthermore, CPHF returns only a limited number of sensors whereas the non-CPHF approach returns all sensors available to CASSARAM, which consumes more resources including more processing time and a significant amount of memory and temporary storage. Fig. 9h shows how the accuracy changes when the Margin of Error (M%) value changes in the CPHF algorithm and the number of sensors increases. The scenario presented in Fig. 5 has been evaluated. The accuracy of the CPHF approach increases when the margin of error (M) increases. However, a lower M leads CASSRAM towards low resource consumption. Therefore, there is a trade-off between accuracy and resource consumption. The optimum value of M can be dynamically learned by machine learning techniques based on which context properties are prioritized by the users in each situation and how the normalized weights are distributed between the context properties.

In Fig. 9i and Fig. 9j, we evaluated how processing time and memory requirements change when relational expressions are used during the semantic querying phase. We tested different scenarios with and without relational expressions (e.g. $<, >, =, \leq, \geq$) as described at the end of Section VI. For all experiments, we queried 100,000 sensors. When at least one relation operator is used in SPARQL, the processing time and the memory requirements increase by 100%. However, neither the number of relational operators used nor the type of relational operators used make any impact on either processing time or memory requirements. Therefore, it is efficient to use multiple relational operators (as much as possible) so as to reduce the number of sensors retrieved by the querying phase. This helps to reduce the amount of data needing to be handled in the other phases.

Finally, in Figs. 9k and 9l, we extensively evaluated how REF affects the processing time and memory requirements in CASSARAM, as the number of sensors and context properties increases. As we mentioned earlier, REF adds more processing overhead, which affects the processing time and memory. There is a significant difference in processing time when the number of sensors needing to be queried is less than 100,000. However, when the number of sensors increases beyond 100,000, the difference becomes insignificant. In contrast,

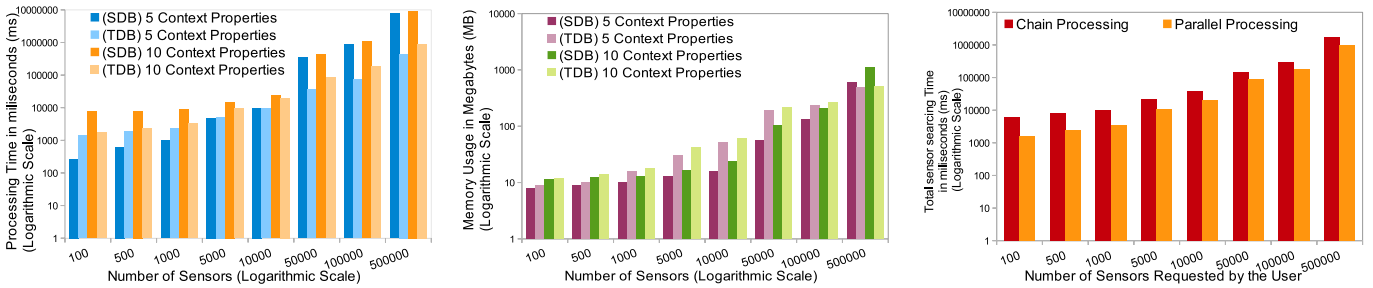

Fig. 10. Results of alternative storage usage and distributed sensor searching: (a) comparison of the processing times taken by both the Jena SDB/MySQL and the Jena TDB approach, (b) comparison of memory usage by the SDB and TDB approaches, and (c) comparison of the times taken by two different distributed searching techniques, namely, chain processing and parallel processing.

the differences in memory requirements are negligible when the number of sensors is less than 10,000: but it starts to become visible after that. Furthermore, the processing time increases significantly after 10,000 sensors. We also learned that allocating more memory for CASSARAM can speed up the entire sensor selection process.

In contrast, CASSARAM can also be used under limited resources though it takes a much longer time to respond. According to the extensive evaluations we conducted, it is evident that CPHF and REF techniques can be used to improve the efficient of CASSARAM. Even though this paper is specifically focused on sensor selection in the IoT domain, the proposed model and the concepts we employed can be used in many other domains, such as web service selection. Furthermore, the results we obtained through these evaluations are also applicable to any other approach that employs an ontology model similar to the SSN ontology and requires a large number of records. Even though we tested our solution with millions of sensor descriptions, in practice it is highly unlikely that millions of sensors would connect to a single middleware instance. Practically, IoT middleware solutions will store data in a distributed manner in different instances, and need to be searched in a distributed fashion, as explained in Section V-C. By parallel processing, the amount of time it takes to process millions of sensor data descriptions can be reduced drastically.

A. Evaluating Alternative Storage Options

In the evaluations conducted earlier (Figs. 9a–9l), we used Jena SDB/MySQL-backed RDF storage to store the data. In order to evaluate the performance of CASSARAM when using alternative storage options, we here employ a Jena TDB-backed approach (jena.apache.org/documentation/tdb). In Fig. 10c, we compare the processing times taken by both the Jena SDB/MySQL and the Jena TDB approach. Furthermore, in Fig. 10b, we compare the memory usage by the SDB and TDB approaches. According to the Berlin SPARQL Benchmark [30], Jena TDB is much faster than Jena SDB. We also observed similar results both in 5 context data processing as well as in 10 context data processing. Specifically, Jena TDB is 10 times faster than SDB when processing 10 context properties, where the dataset consists of half a million sensor descriptions. The Jena SDB approach consumed less memory than the Jena TDB approach when the dataset was less than 100,000 sensor descriptions. However, after that, the Jena TDB approach consumes less memory than the Jena SDB. Specifically, Jena TDB uses 50% less memory than Jena SDB when processing 10 context properties, where the dataset consists of half a million sensor descriptions. Therefore it is evident that Jena TDB is more suitable when the number of sensor descriptions goes beyond 100,000.

Despite the differences we observed in our evaluation, there are several factors that need to be considered when selecting underlying storage solutions. As evaluated on the Berlin SPARQL Benchmark, there are several other storage options available, such as Sesame (openrdf.org), Virtuoso TS, Virtuoso RV, and D2R Server [30]. Jena TDB offers faster load times and better scale, but has the worst query performance. Sesame seems better all-round for low data sizes assuming infrequent loads. In contrast, Jena SDB provides moderate performance, offering load times, query performance, and scalability between the Jena TDB and Sesame. Based on these evaluations, at the time at which this paper was written, there is no superior solution that has all good qualities. Due to the lack of extensive usage and the short existence of Sesame, SDB/MySQL can be seen as a better choice especially when considering database functionalities such as backup, concurrent and parallel processing. As we do not expect frequent loading/ unloading of datasets such as sensor descriptions, it is evident that SDB outperforms TDB in query processing (excluding data loading) [30]. As we expect more updates (transactions) to occur, SDB would be a better choice.

B. Evaluating Distributed Sensor Searching

We evaluated distributed sensor searching using a private network that consists of four computational nodes. We compare two different distributed sensor search techniques, namely, chain processing and parallel processing with/without k-extensions, which we discussed in Section V-C. The results are presented in Fig. 10c. Each node consists of a dataset of one million sensor data descriptions. The four datasets are different from each other. Five context properties are considered for the evaluation and the context information is stored using Jena TDB. First, we discuss the techniques from the theoretical perspective.

Let us define some of the notations which will be used in the following discussion: n = number of computational nodes (in our experiments $n = 4$), N = number of sensors requested by the users, S_i = number of sensor descriptions stored in

TABLE III

THE AMOUNT OF REDUNDANT DATA COMMUNICATION SAVED BY THE PARALLEL SENSOR SEARCH WITH k-EXTENSION STRATEGY

k value \ Number of sensors requested by the users (N)	100	500	1,000	5,000	10,000	50,000	100,000	500,000	1,000,000
10	-60.7	-60.5	-60.3	-58.7	-56.7	-40.5	-20.2	141.6	344.0
100		-5.9	-5.7	-4.1	-2.1	14.1	34.3	196.2	398.5
500			-1.1	0.5	2.5	18.7	38.9	200.8	403.1
1000				0.8	2.8	19.0	39.3	201.1	403.5
5000					0.9	17.1	37.3	199.2	401.5
10000						14.1	34.3	196.2	398.5
50000							10.1	172.0	374.3
100000								141.6	344.0
500000									101.2

in Megabytes (MB)

the ith computational node, r = size of a single sensor description record (i.e., storage requirements), $t_{i,j}^{net}$ = time taken for network communication between the computational nodes i and j, t_i^{pro} = time taken to query the computational node i, merge the indexed results with the incoming results, and select the final number N of sensors. The total time taken by chain-based distributed sensor searching can be defined as:

$$Total_{chain} = \sum_{i=1}^{n} t_i^{pro} + \sum_{i=1}^{n-1} t_{i,i+1}^{net} + t_{n,1}^{net}. \quad (1)$$

The total time taken by parallel distributed sensor searching can be defined as:

$$Total_{parallel} = max\left\{i = [2..n] : t_i^{pro} + t_{1,i}^{net}\right\}. \quad (2)$$

According to the results, it is evident that parallel processing is more efficient than chain processing in terms of the total processing time. However, parallel processing is inefficient in other aspects, such as network communication and bandwidth consumption. Therefore, we proposed k-extension to address this issue. The evaluation of the k-extension approach is presented in Table III. In this experiment, we measured how much data communication can be saved (i.e., due to elimination of redundant data communication that occurs in parallel processing without k-extension) by using different k values under different N values. We measured the *guaranteed minimum*[2] amount of data communications (measured in Megabytes) that can be saved.

In Table III, positive values (marked in green) indicate the minimum amount of data communication saved using the k-extension. Although negative values (marked in orange/red) indicate no guaranteed savings, some situations (marked in orange) have a high chance of saving redundant data communication compared to others. Equation (3) can be used to calculate the guaranteed minimum amount of data saving by using k- extensions.

$$Total_{Saving} = \sum_{i=2}^{n} S_i r - \left\{[\sum_{i=2}^{n} \frac{S_i}{k} + N + (k-1)n] \times r\right\}, \quad \text{IF } (k < N). \quad (3)$$

[2]Depending on the dataset and the context information stored in each node, the parallel processing technique with k-extension will be able to save more data communication than the guaranteed minimum level.

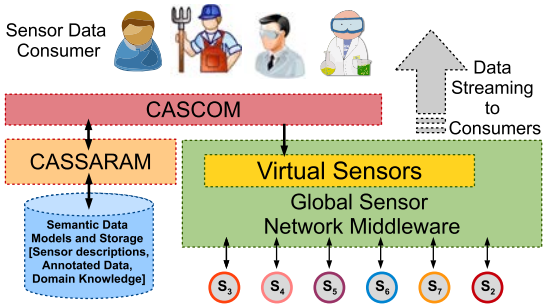

Fig. 11. CASSARAM in Action.

Let us consider different scenarios where chain and parallel processing can be used. Chain processing is suitable for situations where saving computational resources and bandwidth is more critical than response time. A parallel processing method *without k-extension* is suitable when response time is critical and N is fairly small. *k-extension* requires two communication rounds: communication radios need to be opened and closed twice. Such a communication pattern consumes more energy [38], especially if the computational devices are energy constrained. Therefore, transmitting data at once is more efficient. However, this recommendation becomes invalid when N becomes very large (10,000+). Our experiments clearly show that k-extensions can be used to improve the efficiency of the parallel sensor searching approach, especially when N is large. The ideal value of k needs to be determined based on N, n, and S_i.

C. Application

In this section, we show where CASSARAM fits in the big picture (Figure 11). Sensor data consumers are expected to interact with a model called *Context Aware Sensor Configuration Model* (CASCoM) [39]. Details explanation of CASCoM is out of the scope of this paper. Consumers are facilitated with a graphical user interface, which is based on a question-answer (QA) approach, that allows to express the requirements. Users can answer as many questions as possible. CASCoM searches and filters the tasks that the user may wants to perform. From the filtered list, users can select the desired task (e.g. environmental pollution detection). CASCoM searches for different programming components that allow to capture the data stream required by consumers

(i.e. sensor data required to detect environmental pollution). CASCoM tries to find sensors that can be used to produce the inputs required by the selected data processing components. To achieve this task, CASCoM employs CASSARAM. Once the required sensor types are identified (and if multiple sensors are available), CASSARM graphical user interface is provided to the consumers to define their priorities. Later, the final set of sensors and data processing components are composed together. Required wrappers [40] and the virtual sensor [3] are generated and sent to GSN by CASCoM. Finally, GSN starts streaming data to the consumer as defined in the virtual sensor.

VIII. Conclusions and Future Research

With advances in sensor hardware technology and cheap materials, sensors are expected to be attached to all the objects around us, which will increase the number of sensors available to be used. This means we have access to multiple sensors that would measure a similar environmental phenomenon. Such circumstances force us to choose between alternatives. We need to decide which operational and conceptual sensor-related context properties are more important than others.

In this paper, we showed how the context information related to each sensor can be used to search and select the sensors that are best suited to a user's requirements. We selected sensors based on the user's expectations and priorities. As a proof of concept, we built a working prototype to demonstrate the functionality of our CASSARAM and to support the experimentations using realistic applications. We also highlight how CASSARAM helps achieve our broader sensing-as-a-service vision in the IoT paradigm. CASSARAM allows optimizing the sensor data collection approaches by selecting the sensors in an optimized fashion. For example, CASSARAM can be used to find out which sensors have more energy and collect data only from those sensors. This helps to run the entire sensor network for a much longer time without reconfiguring. We explored three different techniques that improve the efficiency and scalability of CASSARAM: comparative-priority based heuristic filtering, relational-expression based filtering, and distributed sensor searching. We evaluated the performance of the proposed model extensively. In the future, we plan to incorporate CASSARAM into leading IoT middleware solutions such as GSN, SenseMA, and OpenIoT, to support an automated sensor selection functionality in distributed environments. We will also investigate how to improve the efficiency of CASSARAM using cluster-based sensor search and heuristic algorithms that incorporate machine learning techniques.

Acknowledgement

The Authors acknowledge help and contributions from The Australian National University, Canberra, Australia.

References

[1] A. Zaslavsky, C. Perera, and D. Georgakopoulos, "Sensing as a service and big data," in *Proc. Int. Conf. ACC*, Jul. 2012, pp. 21–29.

[2] (2012). *Open Source Solution for the Internet of Things into the Cloud* [Online]. Available: http://www.openiot.eu

[3] K. Aberer, M. Hauswirth, and A. Salehi, "Infrastructure for data processing in large-scale interconnected sensor networks," in *Proc. Int. Conf. Mobile Data Manag.*, May 2007, pp. 198–205.

[4] A. Broring, F. Bache, T. Bartoschek, and C. P. Elzakker, "The SID creator: A visual approach for integrating sensors with the sensor web," in *Advancing Geoinformation Science for a Changing World*, S. Geertman, W. Reinhardt, and F. Toppen, Eds. New York, NY, USA: Springer-Verlag, 2011, pp. 143–162.

[5] (Jun. 2011). *Semantic Sensor Network XG Final Report: W3C Incubator Group Report* http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/

[6] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the Internet of Things: A survey," *IEEE Commun. Surveys Tuts.*, vol. PP, no. 99, pp. 1–41, May 2013.

[7] P. Barnaghi, W. Wang, C. Henson, and K. Taylor, "Semantics for the Internet of Things: Early progress and back to the future," *Int. J. Semantic Web Inf. Syst.*, vol. 8, no. 1, pp. 1–21, Jan. 2012.

[8] Digital Enterprise Research Institute, (2011). *Linked Sensor Middleware (LSM)* Galway, Ireland [Online]. Available: http://lsm.deri.ie/

[9] D. L. Phuoc, H. N. M. Quoc, J. X. Parreira, and M. Hauswirth, "The Linked Sensor Middleware—Connecting the real world and the Semantic Web," in *Proc. ISWC*, Oct. 2011, pp. 1–22.

[10] S. Nath, J. Liu, and F. Zhao, "Sensormap for wide-area sensor webs," *Computer*, vol. 40, no. 7, pp. 90–93, Jul. 2007.

[11] C. Perera, A. Zaslavsky, P. Christen, M. Compton, and D. Georgakopoulos, "Context-aware sensor search, selection and ranking model for Internet of Things middleware," in *Proc. IEEE 14th Int. Conf. MDM*, Jun. 2013, pp. 314–322.

[12] C. Truong, K. Romer, and K. Chen, "Fuzzy-based sensor search in the Web of Things," in *Proc. 3rd Int. Conf. IoT 2012*, 2012, pp. 127–134.

[13] S. Mayer, D. Guinard, and V. Trifa, "Searching in a web-based infrastructure for smart things," in *Proc. 3rd Int. Conf. Internet Things*, 2012, pp. 119–126.

[14] B. M. Elahi, K. Romer, B. Ostermaier, M. Fahrmair, and W. Kellerer, "Sensor ranking: A primitive for efficient content-based sensor search," in *Proc. Int. Conf. Inf. Process. Sensor Netw.*, 2009, pp. 217–228.

[15] B. Ostermaier, K. Roalter, L. Oandmer, F. Mattern, M. Fahrmair, and W. Kellerer, "A real-time search engine for the Web of Things," in *Proc. 2nd Int. Conf. IoT*, Dec. 2010, pp. 1–8.

[16] J. Domingue and D. Fensel, "Toward a service web: Integrating the Semantic Web and service orientation," *IEEE Intell. Syst.*, vol. 23, no. 1, pp. 86–88, Dec. 2009.

[17] S. De, T. Elsaleh, P. Barnaghi, and S. Meissner, "An Internet of Things platform for real-world and digital objects," *Scalable Comput., Pract. Exper.*, vol. 13, no. 1, pp. 45–57, 2012.

[18] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-based Internet of Things: Discovery, query, selection, and on-demand provisioning of web services," *IEEE Trans. Services Comput.*, vol. 3, no. 3, pp. 223–235, Jul./Sep. 2010.

[19] T. A. Butt, I. Phillips, L. Guan, and G. Oikonomou, "TRENDY: An adaptive and context-aware service discovery protocol for 6LoWPANs," in *Proc. 3rd Int. Workshop Web Things*, 2012, pp. 2:1–2:6.

[20] Z. Shelby, "Embedded web services," *IEEE Wireless Commun.*, vol. 17, no. 6, pp. 52–57, Dec. 2010.

[21] J.-P. Calbimonte, H. Jeung, O. Corcho, and K. Aberer, "Enabling query technologies for the Semantic Sensor Web," *Int. J. Semant. Web Inf. Syst.*, vol. 8, no. 1, pp. 43–63, Jan. 2012.

[22] R. Garcia-Castro, O. Corcho, and C. Hill, "A core ontological model for Semantic Sensor Web infrastructures," *Int. J. Semantic Web Inf. Syst.*, vol. 8, no. 1, pp. 22–42, Jan. 2012.

[23] N. Namatame, Y. Ding, T. Riedel, H. Tokuda, T. Miyaki, and M. Beigl, "A distributed resource management architecture for interconnecting Web-of-Things using uBox," in *Proc. 2nd Int. Workshop Web Things*, 2011, pp. 4:1–4:6.

[24] J.-P. Calbimonte, H. Jeung, O. Corcho, and K. Aberer, "Semantic sensor data search in a large-scale federated sensor network," in *Proc. 4th Int. Workshop Semantic Sensor Netw.*, Oct. 2011, pp. 1–29.

[25] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelffle, "Vision and challenges for realising the Internet of Things," European Commission—Information Society and Media DG, Luxembourg, Germany, Tech. Rep., Mar. 2010, (ISBN 978-92-79-15088-3, doi:10.2759/26127).

[26] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "CA4IOT: Context awareness for Internet of Things," in *Proc. IEEE Int. Conf. Internet Thing*, Nov. 2012, pp. 775–782.

[27] Commonwealth Scientific and Industrial Research Organisation (CSIRO), (2011). *Phenonet: Distributed Sensor Network for Phenomics Supported by High Resolution Plant Phenomics Centre, CSIRO ICT Centre, and Csiro Sensor and Sensor Networks TCP*, Australia, [Online]. Available: http://phenonet.com
[28] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a service model for smart cities supported by Internet of Things," in *Proc. Trans. ETT*, 2013, pp. 1–13, doi: 10.1002/ett.2704.
[29] M. Compton, C. Henson, H. Neuhaus, L. Lefort, and A. Sheth, "A survey of the semantic specification of sensors," in *Proc. 2nd Int. Workshop Semantic Sensor Netw. 8th Int. Semantic Web Conf.*, Oct. 2009, pp. 1–16.
[30] C. Bizer and A. Schultz, "The Berlin SPARQL benchmark," *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 2, pp. 1–24, 2009.
[31] S. Ran, "A model for web services discovery with QoS," *SIGecom Exchanges*, vol. 4, no. 1, pp. 1–10, Mar. 2003.
[32] D. Chalmers and M. Sloman, "A survey of quality of service in mobile computing environments," *IEEE Commun. Surveys Tuts.*, vol. 2, no. 2, pp. 2–10, Apr. 1999.
[33] C. Perera, P. P. Jayaraman, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Mosden: An Internet of Things middleware for resource constrained mobile devices," in *Proc. 47th HICSS*, Jan. 2014, pp. 1–10.
[34] 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. 1st Int. Symp. Handheld Ubiquitous Comput.*, 1999, pp. 304–307.
[35] Apache Foundation, (2011). *Commons Math: The Apache Commons Mathematics Library*, Forest Hill, MD, USA [Online]. Available: http://commons.apache.org/math/
[36] Apache Software Foundation, (Nov. 2010). *Apache Jena*, Forest Hill, MD, USA [Online]. Available: http://jena.apache.org/
[37] Australian Government, Bureau of Meteorology, (2012). *Experimental Environmental Linked-Data Published by the Bureau of Meteorology*, Melbourne, Australia [Online]. Available: http://lab.environment.data.gov.au/
[38] C. Perera, A. Zaslavsky, P. Christen, A. Salehi, and D. Georgakopoulos, "Capturing sensor data from mobile phones using global sensor network middleware," in *Proc. IEEE 23rd Int. Symp. PIMRC*, Sep. 2012, pp. 24–29.
[39] C. Perera, A. Zaslavsky, M. Compton, P. Christen, and D. Georgakopoulos, "Semantic-driven configuration of Internet of Things middleware," in *Proc. 9th Int. Conf.SKG*, Oct. 2013, pp. 1–8.
[40] C. Perera, A. Zaslavsky, P. Christen, A. Salehi, and D. Georgakopoulos, "Connecting mobile things to global sensor network middleware using system-generated wrappers," in *Proc. 11th ACM Int. Workshop Data Eng. Wireless Mobile Access*, May 2012, pp. 23–30.

Charith Perera (S'12) received the B.Sc. (Hons.) degree in computer science from Staffordshire University, Stoke-on-Trent, U.K., in 2009 and the M.B.A. degree in business administration from the University of Wales, Cardiff, U.K., in 2012. He is currently pursing the Ph.D. degree in computer science with Australian National University, Canberra, Australia. He is with the Information Engineering Laboratory, ICT Centre, CSIRO. His current research interests include the Internet of Things, pervasive and ubiquitous computing with a focus on sensor networks, and context aware computing. He is a member of the Association for Computing Machinery.

Arkady Zaslavsky (M'92) is the Science Leader of the Semantic Data Management science area with the Information Engineering Laboratory, ICT Centre, CSIRO. He is an Adjunct Professor with Australian National University, Canberra, Australia, a Research Professor with Lawrence Technological University, Southfield, MI, USA, and an Adjunct Professor with the University of New South Wales, Kensington, Australia. He is currently involved in and is leading a number of European and national research projects. He received the M.Sc. degree in applied mathematics majoring in computer science from Tbilisi State University, Georgia, Russia, in 1976 and the Ph.D. degree in computer science from the Moscow Institute for Control Sciences, U.S.S.R. Academy of Sciences, Moscow, Russia, in 1987. He has published more than 300 research publications. He is a Senior Member of ACM and a member of IEEE Computer and Communication Societies.

Chi Harold Liu (M'10) is a Staff Researcher with IBM Research, Beijing, China. He received the Ph.D. degree from Imperial College London, London, U.K. and the B.Eng. degree from Tsinghua University, Beijing, China. His current research interests include the Internet of Things, big data analytics, mobile computing, and wireless ad hoc, sensor and mesh networks. He received the Distinguished Young Scholar Award in 2013, the IBM First Plateau Invention Achievement Award in 2012, and the IBM First Patent Application Award in 2011, and was interviewed by EEWeb.com as the Featured Engineer in 2011. He published widely in major conferences and journals and owned ten EU/U.S./China patents. He has served as the General Chair of international workshops with IEEE SECON in 2013, IEEE WCNC in 2012, and ACM UbiComp in 2011.

Michael Compton is a Research Scientist with the Information Engineering Laboratory of CSIRO's ICT Centre. Since joining the ICT Centre in 2006, he has been with the Information Security and Privacy Team in the pHealth and Sensor Networks themes, and in the Data Services for Sensor Networks project in the sensor networks theme, and now works in the Semantic Frameworks for the Hydrological Sensor Webs project in the IWIS theme. His current research interests include using logic and specification to model and reason about systems and algorithms and as parts of systems that execute declarative descriptions of required processing. He currently works with Semantic Web technologies for security, sensors, and data integration. He received the B.I.T. (Hons.) degree from the Australian National University, Canberra, Australia, in 2000 and the Ph.D. degree from the University of Cambridge, Cambridge, U.K., in 2007.

Peter Christen is an Associate Professor with the Research School of Computer Science, Australian National University, Canberra, Australia. He received the Diploma degree in computer science engineering from ETH Zürich, Zürich, Switzerland, in 1995 and the Ph.D. degree in computer science from the University of Basel, Basel, Switzerland, in 1999. His current research interests include data mining and data matching (entity resolution). He is especially interested in the development of scalable and realtime algorithms for data matching, and privacy and confidentiality aspects of data matching and data mining. He has published over 80 papers, including in 2012 the book *Data Matching* (Springer), and he is the Principal Developer of the Febrl (Freely Extensible Biomedical Record Linkage) open source data cleaning, deduplication, and record linkage system.

Dimitrios Georgakopoulos (M'91) is a Research Director with the CSIRO ICT Centre, where he heads the Information Engineering Laboratory that is based in Canberra and Sydney. He is an Adjunct Professor with Australian National University, Canberra, Australia. Before coming to CSIRO in October 2008, he held research and management positions in several industrial laboratories in the U.S. From 2000 to 2008, he was a Senior Scientist with Telcordia, Bellcore, NJ, USA, where he helped found Telcordias Research Centers in Austin, TX, USA, and Poznan, Poland. From 1997 to 2000, he was a Technical Manager with the Information Technology Organization of Microelectronics and Computer Corporation (MCC), and the Chief Architect of MCC's Collaboration Management Infrastructure Consortial Project. He has received the GTE (Verizon) Excellence Award and two IEEE Computer Society Outstanding Paper Awards, and was nominated for the Computerworld Smithsonian Award in Science. He has published more than 100 journal and conference papers.