Size Efficient Big Data Sharing Among Internet of Things Devices

Sungmin Cho and Christine Julien
The Center for Advanced Research in Software Engineering
The University of Texas at Austin
Email: {smcho, c.julien}@utexas.edu

***Abstract*—The Internet of Things (IoT) connects smart objects so they can share information in a network to provide context-sensitive services. The amount of shared information will increase, likely dramatically, as more and more smart objects join the network and disseminate their contextual information. In this paper, we explain how smart IoT devices can share a large amount of context information using much less storage space and communication bandwidth. We use the versatile and simple JSON format at the application interface to allow applications to define their context descriptions, but we convert this JSON format into size-efficient yet equivalent probabilistic data structures for storage or communication. The loss of schema information in the conversion is compensated for by introducing a *schema summary*, which incorporates a hierarchical structure, and a state machine that recovers the schema information from the relationship among elements in a summary.**

I. Introduction

Through connected smart objects that generate, process, and share context information, the Internet of Things (IoT) can provide time- and location-sensitive services. Consider a situation where smart vehicles enhance safety by sharing sensed information such as location, speed, and direction with other nearby vehicles. As another example, sellers and buyers in an open-air market can share interests or advertisements to help match sellers with offerings to buyers who are interested in those offerings. In these and similar situations, the amount of context information shared can increase, sometimes exponentially, with an increasing the number of participants. Sharing such massive amounts of context information is prohibitively expensive for battery-powered IoT devices and can be burdensome even for resource-rich devices.

To address the potential downsides in sharing such large amounts of context information, one could use a proprietary format to store the context information. For example, only values could be stored to reduce the context size. However, this approach raises two issues: (1) a lack of *flexibility* due to the limited ability to store only specific data types and (2) decreased *interoperability* due to the limitation that only devices that can decode the proprietary format can exchange information. In the IoT, any smart devices may need to share any kind of information with other devices in the network network, so these could be critical issues. Compressing context information could be another solution, but compressing data does not always result in the needed size reduction.

In this paper, we present an approach that can reduce context size, sometimes dramatically, when sharing or storing context information, without degrading the application's flexibility and interoperability. Our approach uses the JSON (JavaScript Object Notation) format to store context information. This JSON formatted data is, in turn, encoded into a probabilistic data structure [1] to reduce the context size for communication or storage. In our previous research, we demonstrated that such an approach can achieve up to an87.42% reduction in context size [2]. However the use of the probabilistic data structure in our prior work comes with two disadvantages: (1) when encoded schema of the data structure, i.e., the list of encoded data labels, is not readily available and (2) being a probabilistic data structure, a retrieved data value may provide false information, i.e., the value could result from a false positive that indicates a label is a member of the data structure when it is not. We address these issues by introducing (1) a *schema summary*—a context representation that can also stores the identity of its contents using its schema entity-relationship [3] and (2) an Augmented Transition Network (ATN) [4]—a state machine that recovers the schema and corresponding values from a probabilistic data structure using the relationship.

Using our approach, applications can benefit from the JSON data format—a gateway to various analytic, database, and visualization tools—as an input context representation. When users aggregate massive context information to share among devices within an IoT network or to store locally, they can convert the JSON representation to probabilistic data structures that are size-reduced but equivalent representations. These probabilistic data structures can be recovered back to the JSON representation anytime with a ATN state machine. Our contributions are:

- we introduce a technique to recover the schema of a context, and thus to recover the JSON format data, from the probabilistic data structure using ATN and a schema summary (Section III); and
- we calculate and assess the probabilities of false-positives in recovering the JSON format data from probabilistic data structures in order to demonstrate that our approach can reduce the total amount of context size dramatically without degrading the data quality (Section IV); and
- we augment our ATN approach with a technique to further enhance data quality, specifically detecting false-positive values in a floating-point type (Section III-C).

II. Context Summary Representations

In this paper, we define context information as the situational information that can characterize the situation of a person, place, or objects [5]. Context information represents knowledge used to reason about a surrounding environment and to build shared perspectives [6] among smart objects in an IoT network. Various models can be used to exchange or store contextual information [7]. We use a "key-value pair" model wherein a context summary is defined as a set of attributes; each attribute is a (label, value) pair. A schema of a context summary is simply the set of labels in the summary. To represent context information for applications, we use the JSON data exchange format due to its high performance and low resource consumption in processing the information [8]. Furthermore, the JSON format is directly supported in many programming languages, application libraries and tools, web service interfaces, and protocols [8], [9].

In our previous research [2], we gave examples of how a context summary is represented in JSON; we explained the CHITCHAT data type to show how the values in the summary can be succinctly described; and we showed various additional context representations that differ in their tradeoffs related to summary size, data quality, and flexibility. In this section, we summarize the highlights of this prior work; all of the examples and tables are from the prior work.

A. Context Summary Examples in the JSON Format

Consider a "market example" wherein people use a hyper-localized search to share information about the environment and individual's interests in order to find the best matches between buyers and sellers. Below is a JSON example that captures the situation when a book lover visits an open-air book fair and shares her interest in modern art books.

```
{
  "latitude": [31, 25, 38, 2],
  "longitude": [-17, 42, 11, 0],
  "date":[2016, 10, 09],
  "time": [10, 21],
  "leave time": [15, 21],
  "interest category 1": "toys",
  "interest item 1": "German Steiff Teddy Bear",
  "interest category 2": "paper",
  "interest item 2": "Roylco R15286 Antique Paper",
}
```

The JSON format can be also used for machine-to-machine or device-to-device type sensor data exchange. This is a "sensor example" from a building.

```
{
  "latitude": [30, 25, 38, 5],
  "longitude": [-17, 47, 11, 0],
  "time": [11, 21],
  "date":[2016, 10, 11],
  "device id": 11,
  "number of sensor": 3,
  "sensor 1 name": "temperature",
  "sensor 1 value": 28,
  "sensor 1 unit": "C",
  "sensor 2 name": "light",
  "sensor 2 value": 121,
  "sensor 2 unit": "lux"
}
```

These context summary examples show how JSON can be used for high quality and descriptive attributes.

B. CHITCHAT Data Types

JSON uses only a string-type for representing stored values. In some cases, this can waste storage, particularly for integers, floating point numbers, and arrays of them. In the example of the second scenario, the "sensor 2 value" uses three bytes for storing the value "121" in a string type when one byte is sufficient for storing the integer value. This wasted storage could be ignored when only a small number of context summaries is shared, but when huge numbers of context summaries are exchanged, this overhead cannot be ignored. To reduce the context summary size, we introduced CHITCHAT data types [2] tailored to the pervasive computing context; specific examples are shown in Table I. We support multiple integral types and assume only single precision (32 bit) floating point numbers (a higher precision is not commonly required for context). We use Pascal-style strings, with the length as the first element. We also define special types to aid in efficiently packing data, e.g., a "level" type that scales from 1 and 10. Some data types use multiple bits for the encoding; dates include a year (7 bits), month (4 bits), and day (5 bits) and times have hours (5 bits) and minutes (6 bits). Both latitude and longitude are expressed in degrees (± 90 for latitude and ± 180 for longitude), minutes, seconds, and sub-seconds.

TABLE I: Example data types for contexts

Type	Bits	Bytes	Range	Encoding
Byte	8	1	(-128, 127)	
Float	32	4		IEEE 754
String	$n \times 8$	n		Pascal
Level	4	1	(1,10)	
Date	16	2		(7,4,5)
Time	11	2		(5,6)
Latitude	27	4		(8,6,6,7)
Longitude	28	4		(9,6,6,7)

C. CHITCHAT Context Summary Representations

In our prior work, we analyzed various context representations for a context summary. Each representation has different advantages and disadvantages. We summarize the properties of each context representation in three groups.

1) JSON Summaries: A JSON summary is the CHITCHAT's interface for users (humans, applications, and devices). The JSON summary can be compressed to reduce size; we could achieve a 29% and 34% size reduction from our market and sensor examples, respectively.

A labeled summary is a JSON summary that uses CHITCHAT data types to more compactly store values. The size reduction compared to the JSON summary is 38% and 48% respectively for the examples. Both the JSON and labeled summaries keep the schema as part of their representations.

2) Proprietary Summaries: For proprietary summary formats, where the schemas of the summary are already known, we can reduce the context summary size by not storing labels; such a complete summary trades off flexibility for size-efficiency as only the context values that match the already-shared schema can be stored in the summary. The size reduction is 67% and 85% for the two examples.

3) Probabilistic Data Structure Summaries: Bloomier Filters [1] are probabilistic data structures that deal with arbitrary function mapping from a label (e) to a CHITCHAT type value ($f(e)$) in a table that can then be queried by an application using a target label as the input to the query. Bloomier Filters have expressiveness limitations because the table width constrains the CHITCHAT types that can be stored. For example, with two bytes in table width, only values of types that require up to two bytes can be stored. To address this issue, we introduced an FBF (Folded Bloomier Filter) data structure that can express any CHITCHAT types by "folding" the value across multiple entries in the table. A CBF (Complete Bloomier Filter) is an FBF with enhanced flexibility that allows the values in a summary to be updated; this results in a larger context size since the table size must be extended to allow for the added flexibility.

Table II shows the size reduction from a JSON summary, the largest but most flexible, and size increase from a complete summary, the smallest but least flexible, for both FBF and CBF context summaries. The first two rows demonstrate the size efficiency of the market and sensor examples. The next two rows show the results with a context summary that contains only string values ("strings") and one with only numerical values ("stringless"). The size reduction rate is more pronounced for "strings" than for "stringless", which matches the pattern in that the sensor example, which has fewer string type values, shows an increased size reduction relative to the market example, which has more string type values.

TABLE II: Size efficiency

Example	Reduction (%) vs. JSON		Increase (%) vs. complete	
	FBF	CBF	FBF	CBF
market	76.42	67.45	6.38	46.81
sensor	80.47	74.22	6.38	40.43
strings	46.55	28.16	4.49	40.45
stringless	87.42	84.91	-4.76	14.29

As both FBF and CBF summaries are probabilistic data structures, there is always the risk of returning false positive values in response to application queries. A false positive occurs when querying a context summary returns a "junk" value for an attribute that was not inserted. To mitigate false positives, we use filters to constrain reasonable values based on application semantics [10] by filtering out false-positives using a pair of filter. First, the *innate filter* checks the data range or encoding format to detect impossible values. Second, the *correlated filter* uses the relationship among labels in the summary to identify unlikely values.For example, two correlated labels—*longitude* and *latitude*—establish a location. The probability that a summary is wrongly identified to contain both *longitude* and *latitude* labels is considerably lower than the case identified to contain only one of the labels.[1] Table III from our previous experiments [2] shows how these filters can remove false positives effectively.

[1]We omit a third *situational filter* that removes false positives using situational information, as we do not use this filter in this work.

TABLE III: False positive (*fp*) probabilities (FBF)

Type	$fp_{innate}(\%)$		$fp_{correlate}(\%)$	
	theory	exp.	theory	exp.
Boolean	0.39	0.38		
Level	4.29	4.28	0.0037	0.0034
Float	99.99	99.99	0.085	0.082
Latitude	1.5	1.5	0.045	0.051
Time	2.2	1.9	0.12	0.11

III. JSON CONTEXT SUMMARY CONVERSION

A JSON summary is the most versatile representation from the CHITCHAT context representations. Once contextual information is captured and saved in a JSON summary, the summary can be used as an input to various application tools including database, analytic, and visualization tools. Any information in the JSON summaries—schemas and values—can be retrieved, analyzed, combined, and processed to make new context summaries. However, these benefits are not free; a JSON summary requires more storage space and communication bandwidth for sharing context information than other representations. For storage and communication, JSON summaries can be converted to other context representations to reduce the summary size, but the conversion can be unidirectional; if a JSON summary is converted into a schema-less representation, there may be no way to recover it back to a JSON summary.

In this section, we explain how JSON summaries can be structured to enable the bidirectional conversion to and from schema-less representations. Specifically, we describe the conditions and mechanisms of how the schema of probabilistic data structures (FBF and CBF summaries) can be retrieved to build JSON summaries; we also describe the techniques to improve the false-positive detection rate to make the conversion more accurate and reliable.

A. Schema Summaries

The correlated filter is effective in detecting false positives; as shown in Table III above, after applying the correlated filters, the probability of related members being false positive becomes minuscule. We can use this feature to recover the schema from probabilistic context summaries. To do this, we assume a library of *schema descriptions* that define commonly used schemas, using relationships among the attributes found in the schema. Then, recovering a schema becomes equivalent to finding a match of a received summary against these stored schema descriptions; if the summary matches one of the stored schema descriptions, the list of attributes can be recovered. Specifically, we define a *schema summary* to be an FBF or CBF whose correlated filters define a relational structure constraining the schema members; this relational structure is the *schema description*. Associating a schema description with a context summary is not the same as reserving space for particular attributes in the structure as in the case of the CBF, nor is it the same as *requiring* specific attributes to appear in every context summary. Instead, the schema description

specifies a network of relationships that are required among attributes *if* they are present in the summary.

Consider a market that is adverting a special discount. The content of the context summary communicating the advertisement may be required to match the following format: *Market M has a special discount event on day D at time T, and the location is L*. Represented pictorially, this summary has the relatively generic *notification* structure shown in Fig. 1. As captured by a CHITCHAT context summary, only the leaf nodes (shown in gray in Fig. 1) are members of the summary's schema. When a context summary has this structure, even when the application uses an FBF summary for dissemination, any receiver can recover the schema of the summary, i.e., the complete list of the attributes that should be represented in the summary.

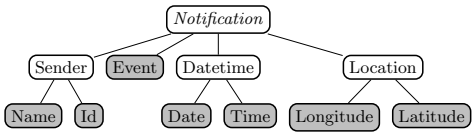

Fig. 1: Notification structure

To make the use of a schema description even more flexible, some members of the schema description can be tagged as optional, and some members can have aliases. For instance, for the notification schema description, we can make the *Id* member optional and the *Event* member to have an alias *Advertisement*. In this case, a context summary with a schema of (*name*, *advertisement*, *date*, *time*, *latitude*, *longitude*) can also be recovered by matching against the notification schema description. CHITCHAT's language for representing a schema description also supports repetition. For example, consider a schema description that includes a name attribute followed by several sensor values, such as (*name*, *sensorName1*, *sensorValue1*, *sensorName2*, *sensorValue2*, ...), using repetition can significantly reduce the amount of space needed to represent the schema description, e.g., (*name*, (*sensorName*, *sensorValue*)+); it also enhances the flexibility of this schema description by enabling it to match against any context summary of the right structure, regardless of the concrete number of sensor values represented.

B. Augmented Transition Network

A schema summary is an ordinary FBF or CBF summary in that it does not explicitly carry its schema information as part of the summary. Instead, the schema summary adheres to some previously known schema description (from a library of such schema descriptions that are known *a priori*). A schema description specifies expected relationships among elements stored in the summary using an Augmented Transition Network (ATN) [4].

An ATN is a state machine with states and registers. It starts from a starting state and makes a transition on any valid input; any input and related transition history are recorded in the register. When all the inputs are consumed, the final state should be reached to recover the schema of a summary. One difference from a traditional ATN that passively waits for input to make a transition is that the ATN actively queries a received context summary to determine whether the label that causes the next transition is in the summary.

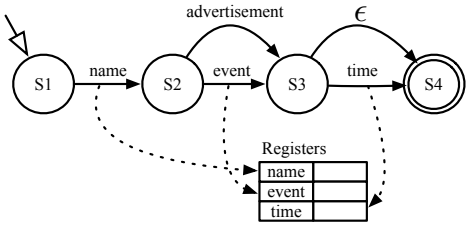

Fig. 2: ATN example

Fig. 2 shows the ATN for the schema description:

$$(name, event|advertisement, time?)$$

The notation $event|advertisement$ indicates that the attribute label $advertisement$ can be used as an alias for *event*, while the ? following $time$ indicates that the *time* attribute is optional. In Fig. 2, the initial state (S1) is marked with an arrow, and the final state (S4) is indicated by a double circle. As CHITCHAT processes the summary, each transition occurs and the ATN retrieves a (label, value) pair and records it in a register. In this particular ATN, the first transition requires the *name* attribute, while the second checks for either the *advertisement* or *event* attribute. The ϵ transition between S3 and S4 accounts for the fact that the *time* attribute is optional.

C. Enhancing False Positives Detection Rate

The assumption underlying using an ATN to recover a schema is that the false positive filters detect all false-positives with high probability. From Table III, we can confirm that most data types satisfy this assumption (e.g., only 1.5% of latitude values that pass the innate filter are not actual latitude values),except for the floating point type. However for the floating point type, which uses the IEEE 754 standard [11], the fp_{innate} (i.e., the number of false positives that remain after applying the innate filter) is close to 100%, since any random number can be decoded as a valid floating point number with rare exception. The IEEE 754 format can represent very small numbers from 10^{-38} to very large ones, up to 10^{38} with varying precision. To ensure the format's ability to cover the very small numbers with high precision, approximately 50% of the values that can be represented are between -1.0 and 1.0.

This issue can be addressed considering that the floating point numbers commonly used to describe human context information are not in this range of extremely small values, we can enhance the innate filter's false positive detection rate. For instance, when encoding a floating point number, we could add 1.0 to any positive value and subtract -1.0 from any negative value to make ensure that no value between -1.0 and 1.0 is placed in an FBF or CBF. Upon decoding, we can identify any floating point values in the range of -1.0 to 1.0 as a false

positive. We can control the amount of the shift to tune the tradeoff between the false positive detections and the range of floating point values allowed. We can detect further false positives by specifying a minimum and/or maximum values.

Obviously, these limits place constraints on the utility of the floating point type, which we attempt to balance alongside the space savings and the ability of the false positive filters to recover. With a shift value of 1.0, and range limit from -100.0 to 100.0, we can reduce the fp_{innate} of float values to approximately 4% in Table III.

IV. Evaluations

In a real world scenario where a receiver attempts to match a received summary against all of the ATNs corresponding to stored schema descriptions, it is possible that the receiver will identify one type of description as another. In this section, we calculate and evaluate (1) the probability that one type of a schema description is incorrectly identified as another; (2) the size reduction rate when JSON summaries are converted into other summaries; and (3) the recovery rate with which these summaries can be converted back to JSON summaries.

For experimental purposes, we define three schema descriptions to describe the context sharing situations we could encounter frequently (Listing 1). We introduce variables to express the structural nature of these schema descriptions.

Listing 1: CHITCHAT schema structures

```
datetime = (date, time)
location = (longitude, latitude)

news = event | advertisement
sender = (name, id?) | ("gchat id")

notification = (sender news datetime location)
dataStream = (sender datetime count value unit
    comment)
sensors = (count datetime location? (name id value
     unit)+)
```

The `notification` schema description matches the one given in Section III-A and depicted in Fig. 1. This schema description can be used for sharing a context summary that captures a one-time event such as an advertisement. The `dataStream` schema description is for data that is continuously updated. Finally, the `sensors` schema description is for aggregating `count` number of sensor values.

A. False Positive Probabilities

First, we assessed the theoretical and experimental false positive rates for these schema descriptions; while a schema summary will always be recovered correctly, it may be possible that some summary that is in fact *not* a schema summary will be recovered as one, by chance. Theoretically, the probability of detecting a summary as a `notification` schema description is $2.95 \times 10^{-13}\%$, as a `dataStream` description is $1.21 \times 10^{-15}\%$, and as a `sensor` description is $1.21 \times 10^{-15}\%$. In practice, we could find no false identification with 100,000 randomly generated context samples. This result shows that in real-world situations, there is practically no possibility that a non-schema summary is identified as a schema summary.

TABLE IV: Maximum True or False Positive Probabilities (%)

	Notification	Data Stream	Sensor
Notification	100.0	1.37×10^{-8}	1.01×10^{-12}
Data Stream	3.33×10^{-6}	100.0	1.62×10^{-12}
Sensor	5.74×10^{-7}	2.13×10^{-8}	100.0

We measured the probability that a schema description is identified correctly (which, as expected, is 100%), and and the probability that one type of schema summary is wrongly identified as another type. While the probability of one schema summary being randomly classified using a different (incorrect) schema description is higher than the probability of classifying a randomly generated summary as a schema summary, the probabilities of incorrect classifications are all fleetingly small. For these experiments, we used 100,000 randomly generated context summaries that did not adhere to any schema description and 100,000 schema summaries for each of the three schema descriptions. Table IV shows the results; the first column reports the probability that a randomly generated summary is accepted as a schema summary for each of the three schema descriptions.

B. Size Reduction and Recovery Rate

In the next evaluation, we assumed a scenario where a user needs to share a large number of context summaries in an IoT environment. Depending on the application, the user can use schema summaries or the summaries with unique schema structures (non-schema summaries). Likewise, the user can also choose what context representations the JSON summaries are converted into. We experimented with three option groups (six options in total with two types per each group), listed in Table V. The *quality* group options favor high quality data and use only JSON summary representations. The *small* group options favor size reduction and uses only the options that guarantee maximum size reduction: complete summaries or FBF summaries. Finally, the *intelligent* option group favors a maximum recovery rate to JSON summaries with possible size reduction.

We created 10,000 randomly generated context summaries to simulate the condition that IoT devices share massive information; for the first experiment, only schema summaries (each using one of the three schema descriptions above) were used. For the second and third experiments, 60% and 0% of the context summaries were schema summaries, respectively. In Table VI, we measured (1) the size reduction (Red) achieved, using the quality type 1 option as a basis and (2) the recovery (Rec) rate, which measured the percentage of the context summaries were recovered to JSON summaries. The results show that the *quality* group options produced perfect recovery

TABLE V: Experimental options

Option	Type	Context Representation Usage
quality	1	JSON
	2	labeled
small	1	FBF
	2	complete
intelligent	1	FBF for schema summary, otherwise JSON.
	2	FBF for schema summary, otherwise label.

rates with very little or no size reductions, while the *small* group options produced maximum size reductions but, as expected, had limited recovery rates. The *intelligent* group options show size reductions proportional to the percentage of schema summaries with perfect recovery.

TABLE VI: Reduction & Recovery rate (%)

	100% schema		60% schema		0% schema	
	Red	Rec	Red	Rec	Red	Rec
quality1	0.0	100.0	0.0	100.0	0.0	100.0
quality2	30.82	100.0	28.19	100.0	23.29	100.0
small1	61.92	100.0	65.79	60.0	73.34	0.0
small2	67.14	0.0	70.96	0.0	78.60	0.0
intelligent1	61.92	100.0	41.26	100.0	0.0	100.0
intelligent2	61.92	100.0	49.04	100.0	23.29	100.0

V. Related Work

Context and context-aware computing, and context representations [12] have been extensively surveyed [13]. In our own prior work, we expressed context as a combination of local and shared information [6] to build global views from sharing context information, and we provided a basic framework for sharing succinct context information in a pervasive computing network [2], [14]. In this work we use the same basic structure but substantially enhance both the data structure and algorithms to enable significant space gains without degrading the quality of the context representation.

The schema summary is conceptually similar to database research related to modeling and describing the business domain to capture relationships among schema entities [3]. A semantic model [15] also has a similar basis as the schema summary in that both express a conceptual model by using specifications of relationships, abstractions, and constraints to capture meanings among entities. ATN [4] was developed as a formal description of natural language used to efficiently parse a grammatically correct structure, but it is used in many other fields that require simple and fast structure parsing [16].

VI. Conclusion

In this paper, we described the benefits (versatility and simplicity) and an issue (size inefficiency) in using JSON for sharing a large amount context information among IoT devices. We explained our solutions in addressing the size issue by converting JSON summaries into size efficient probabilistic data structures for storage or communication. For recovering the converted the data structures back to JSON summaries, we introduced schema summaries, which incorporate pre-defined relational schema structures, and an ATN state machine order to recover JSON summaries using the structural relationships. In our experiments, we demonstrated that schema summaries can be recovered back to JSON summaries effectively and without errors in real-world situations to reduce total summaries for storage or communication bandwidth. We expect this approach is particularly useful in an IoT environment, where various devices participate in sharing context information by allowing context representations with different benefits and effective and accurate conversion among them.

Acknowledgements

This work was funded, in part, by a Samsung GRO and the NSF (#CNS-1218232). The views and conclusions are those of the authors and not of the sponsoring agencies.

References

[1] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal, "The Bloomier filter: an efficient data structure for static support lookup tables." in *Proc. of the 15th Annual ACM-SIAM Symp. on Discrete Algorithms*, 2004, pp. 30–39.

[2] S. Cho and C. Julien, "CHITCHAT: Navigating tradeoffs in device-to-device context sharing." in *Proc. of the 2016 IEEE Int'l. Conf. on Pervasive Computing and Communications*, 2016, pp. 1–10.

[3] P. P.-S. Chen, "Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned," Louisiana State University, Tech. Rep., Feb. 2002.

[4] S. C. Shapiro, "Generalized Augmented Transition Network Grammars for Generation from Semantic Networks." *American Journal of Computational Linguistics*, 1982.

[5] A. K. Dey, "Understanding and Using Context." *Personal and Ubiquitous Computing*, vol. 5, no. 1, pp. 4–7, 2001.

[6] C. Julien, A. Petz, and E. Grim, "Rethinking context for pervasive computing: Adaptive shared perspectives," in *Proc. of the 12th Int'l. Symp. on Pervasive Systems, Algorithms and Networks*, 2012, pp. 1–8.

[7] T. Strang and C. Linnhoff-Popien, "A context modeling survey," in *Proc. of the 1st Int'l. Workshop on Advanced Context Modelling, Reasoning And Management*, 2004.

[8] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of JSON and XML Data Interchange Format: A Case Study." in *CAINE*, 2009.

[9] N. Kobayashi, M. Ishii, S. Takahashi, Y. Mochizuki, A. Matsushima, and T. Toyoda, "Semantic-JSON: a lightweight web service interface for Semantic Web contents integrating multiple life science databases," *Nucleic Acids Research*, vol. 39, pp. W533–W540, Jun. 2011.

[10] S. Cho and C. Julien, "The Grapevine Context Processor: Application support for efficient context sharing," in *Proc. of the 2nd ACM Int'l. Conf. on Mobile Software*, 2015, pp. 68–71.

[11] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008*, pp. 1–70, Aug 2008.

[12] M. Perttunen, J. Riekki, and O. Lassila, "Context representation and reasoning in pervasive computing: A review," *Int'l. J. of Multimedia and Ubiquitous Computing*, vol. 4, no. 4, pp. 1–9, 2009.

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

[14] C.-L. Fok, E. Grim, and C. Julien, "Grapevine: Efficient situational awareness in pervasive computing environments," in *Proc. of PerCom Workshops*, Mar. 2012, pp. 475–478.

[15] J. Peckham and F. Maryanski, "Semantic data models," *ACM Comput. Surv.*, vol. 20, no. 3, pp. 153–189, Sep. 1988. [Online]. Available: http://doi.acm.org/10.1145/62061.62062

[16] W. A. Woods, "Transition network grammars for natural language analysis." *Comm. of the ACM*, vol. 13, no. 10, pp. 591–606, 1970.