Intelligent Information Sharing among IoT Devices: Models, Strategies, and Language

Samuel Sungmin Cho*, Myungkyu Song†, Christine Julien‡
*Northern Kentucky University, †University of Nebraska at Omaha, ‡The University of Texas at Austin
Email: *chos5@nku.edu, †myoungkyu@unomaha.edu, ‡c.julien@utexas.edu

***Abstract*—Various smart objects are connected in the invisible and intelligent Internet of Things (IoT) network to share information. However, considering the heterogeneity of Internet of Things devices, connecting devices with differences in capabilities–processing, storage, energy, and communication bandwidth–and programming methods–language, compiler, and tools–can burden developers with the complexities caused by interoperability. This paper proposes a solution to address this interoperability issue when sharing information among IoT devices. We model information sharing as the communication between a sender and a receiver with constraints from application requirements. We explore tradeoffs from constraints to propose three strategies to hide technical details of various data representations to meet their application needs. We propose the CHITCHAT Information Sharing Language (CISL) for developing IoT applications so that IoT developers can focus solely on their applications by delegating the control of information sharing details to the features that the CISL language provides.**

I. INTRODUCTION

The Internet of Things (IoT) aims to create an invisible and intelligent network fabric with smart objects called IoT devices. These devices can sense the environment, process the acquired information, and communicate among themselves directly or indirectly through a device-to-device or Internet connection. Using these connections, IoT programmers can build applications to provide services that can fundamentally change society as a whole [1], [2].

However, programming IoT devices requires multiple levels of complexity due to their inherent diversity. The first level of complexity is from the different programming environments. IoT devices are supposed to be operated on different hardware and software. For example, sensor devices are programmed using low-level programming tools. Also, these devices have limitations in computing power, energy, and communication bandwidth. However, the device that receives information from these devices may have no such limitations and may be programmed using high-level tools.

The next level of complexity comes from application requirements. The constraints of each IoT device dictates priorities of each device. As an example, consider an IoT application where a server collects temperature information from a massive number of wireless sensors to monitor a wildfire. The sensor devices will prioritize minimal energy consumption due to their battery-powered operations. However, for the server that collects the temperature information from sensors and uploads the aggregated information through the Internet, reducing the data size can result in large savings in storage and bandwidth. For IoT application programmers these complexities need to be addressed so that they can focus solely on the applications.

In this paper, we present an information sharing model, related strategies, and a programming language for IoT application developers to alleviate the problems caused by these complexities. We model a conceptual information sharing as a communication between a sender and a receiver, and both of them have different constraints. The sender encodes information and sends the information through a communication channel (1) with a minimal cost of communication and (2) in a way that the receiver can decode the information received without a loss of data quality. This research uses our previous work of CHITCHAT context representation models [3]. In the research, we presented various encoding representations that have tradeoffs in size, flexibility, encoding energy, and data quality. However, from an IoT application developers' perspective, it is still difficult for them to understand the varieties of representations and to find the optimal representation to meet their application requirements. Therefore, we introduce four strategies that can be used to meet various requirements.

The conceptual information sharing model and related strategies are implemented in the CHITCHAT Information Sharing Programming (CISL) language. The language provides the tools to control the life-cycle of IoT information from the generation, processing, storing, and to sharing. Programmers can use the high-level CISL language to describe the application behavior, and the CISL compiler translates CISL code into application code targeted for various IoT programming environments.

Our contributions are as follows:

- We propose the CHITCHAT information sharing model among IoT devices. IoT programmers can choose the strategy for their applications without understanding technological details.
- We introduce the CHITCHAT programming model and the CHITCHAT Information Sharing Programming language (CISL). This programming model is based on the conceptual model. Programmers can use the CISL and its compiler to generate the application code for IoT devices to facilitate the their development of information sharing applications.
- We assess the benefits of using our tools. We compare the size reduction, information quality degradation, and

relative energy consumption in encoding with different strategies based on real-world scenarios.

The rest of this paper starts with the conceptual context sharing model in Section III. We introduce the CHITCHAT programming model in Section IV followed by the examples of CISL in Sub-section IV-A. Our experiment results are analyzed and explained in Section V.

II. RELATED WORK

The CHITCHAT programming model is a model based aggregation of tools for IoT application developers to control the life-cycle of contexts. Similar research includes a model-based autonomic context management system (ACoMS) [4] that provides the context life-cycle control functions, but the CHITCHAT programming model focuses mainly on the structures and services for context shared among IoT devices. The Context Toolkit [5] also provides services that mediate between the environment and the application with context widgets; the CHITCHAT programming model is similar in providing services to the application developers, but CHITCHAT does not provide low level features. Mobile Sensor Data Processing Engine (MODSEN) [6] is a plug-in-based middleware for resource constrained IoT mobile devices.

Context information can reshape the knowledge [7]; The CISL expands this view in that the degree of the reshaped knowledge can even be controlled by sharing pre-knowledge, which is also knowledge that can be shared. Further, the information can be treated from database to process the knowledge effectively using database technologies and tools. Our view of context as a schema-less database entry is explored in NoSQL databases [8], [9].

CISL is a domain-specific language (DSL) for expressing how the IoT information is shared among IoT devices. Other domains also leverage DSLs for heterogeneous environment of the IoT platform. García et al. [10] propose a DSL for abstracting the application generation problem in an IoT platform. Their DSL allows the creation of IoT applications for interconnected heterogeneous objects: sensors, mobile devices, etc. Given the domain of the problem and its properties, the DSL helps a user to interconnect many objects without any programming knowledge by supporting an abstraction of the base programming language. Salihbegovic et al. [11] present a DSL that helps to develop IoT applications on complex communication, protocols and operating systems. The DSL is used for formal representation as a meta-model to describe a set of visual notations for IoT applications. In contrast to their approaches, CISL focuses on the abstraction of the information sharing models and the special constructs for domain-specific strategies to be described declaratively.

III. THE CHITCHAT INFORMATION SHARING MODEL

We model the sharing of information among IoT devices as sending knowledge, i.e., encoded information, from a sender to a receiver. The contents of the knowledge can be queried to be retrieved and the inner representation of the knowledge is hidden outside. In this model, information is represented as a dictionary: a set of a pair of (key, value). The representation of information, i.e., the encoding method of the dictionary, can be different depending on the devices' application requirements.

Fig. 1 shows our conceptual model for storing and sharing information in IoT devices. In this model, sharing information involves two intelligent agents with different capabilities and priorities: a sender ① and a receiver ②. Knowledge ③ is shared via some available communication channel ④ between the sender and receiver. We define pre-knowledge ⑤ as any information shared among agents *a priori* and post-knowledge ⑥ as any information reconstructed by the receiver from the combination of the received knowledge and pre-knowledge. The cost of communication ⑦ is the energy consumed in the course of sharing information. This energy cost can come from many sources, including energy consumption for communication (which is proportional to the size of the information shared) or energy consumption in processing information to reduce its representation size. Some agents with severely limited bandwidth may make reducing the size the top priority, with a goal to reduce the cost of communication ⑧. However this approach may require more energy to encode the information ⑨ at the sender and to decode it ⑩ at the receiver.

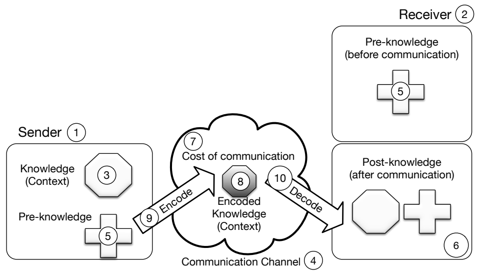

Fig. 1: A conceptual context sharing and storage model

Priorities in sharing information entail tradeoffs. To analyze the tradeoffs, we start by considering several strategies that represent possible alternatives in sharing contexts under various requirements. As an example, we can sacrifice the precision of data in a context by using a single precision floating point number (4 bytes) instead of double precision (8 bytes) to reduce communication costs by communicating with fewer bytes. Alternatively, we may consume more energy in encoding and decoding by compressing the data to achieve a smaller size. If each device already shared the keys in the dictionary (pre-knowledge), we can reduce the communication burden by sending only the values to be recovered from the sender by combining the already shared keys and newly received values. In this case, we trade flexibility (because the keys must be shared *a priori*) for size (and therefore communication cost). Likewise, privacy, latency, andexpressiveness can all be possible tradeoffs. In this paper, we consider only the following factors in tradeoff analysis when sharing and storing information:

Size: The number of bytes used to capture a dictionary.

Encoding energy: The power consumed in processing (encoding, decoding) a dictionary.

Flexibility: The ability to add, remove, and update a value in a context summary.

Data quality: The fidelity with which the data at the receiver matches the sent information.

These factors are related to each other; when we prioritize one factor, other factors can be affected positively or negatively. For example, reducing size using a compression algorithm requires more energy than not compressing. Data quality and size are similarly related. In addition to examples like the floating point precision given above, we can reduce data quality using techniques that run the risk of introducing *false positives* into dictionaries [12], [13]. The CHITCHAT information representation model provides an abstraction to query a context summary for its stored context values, using a value's associated key. A correct value is expected to be returned from the given key, and the result should be empty when there is no matching value of the label. In the case that some random value is returned instead, we call the value a false positive, and the presence of false positives results in degraded data quality. In our previous research, we introduce context summary structures that can dramatically reduce the size of stored context information with a tradeoff in a slight increase in the probability of a false positive, i.e., when the context summary is queried for a given label that was *not* inserted, the query returns untrue value [3], [12].

We categorize our information sharing and storage model using three strategies to analyze the tradeoffs among the factors under each strategy. Fig. 2 shows the "High cost, high quality (HCHQ)" strategy, which is representative of the current state-of-the-art strategies that rely on text-based representations such as XML. In this strategy, the sender sends the knowledge (information) without any loss in data quality; the sender also does not assume any pre-knowledge from the receiver. As a result, users can get the best data quality, but have a high energy consumption for communication and a larger footprint for storage due to the large size required to represent the information.

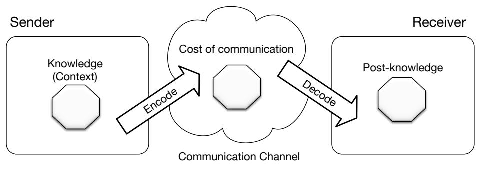

Fig. 2: High cost, high quality strategy

Fig. 3 shows the "Low cost, low flexibility strategy (LCLF)" strategy, which lies at the other end of the spectrum. It is representative of state-of-the-art approaches used in proprietary applications. This strategy assumes pre-knowledge in the receiver. The sender can send only partial knowledge (information) to reduce the size (and therefore the cost) of the information shared. In this strategy, the energy consumption is low due to the small size, but there are constraints in flexibility as the receiver can only know the information that it can recover using its pre-knowledge. This strategy is the opposite of HCHQ strategy.

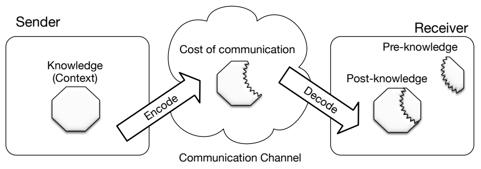

Fig. 3: Low cost, Low flexibility strategy

The HCHQ and LCLF strategies mark the two ends of the spectrum for analyzing the tradeoffs in sharing and storing information. In this paper, we introduce a "Tunable strategy", in which users can tune the aforementioned trade-off factors to meet application constraints while achieving a balance between size and flexibility. In Fig. 4, the sender assumes no pre-knowledge, giving flexibility and expressiveness. Information is encoded ① in a dictionary to reduce the size, and the information in the dictionary retrieved using a query interface ②. Senders can control the size reduction, data quality, and energy consumption with encoding parameters. This strategy can reduce the context summary size without affecting flexibility. However, when the queried result is false positive, data quality ③ is affected.

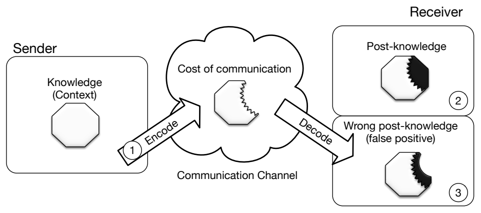

Fig. 4: Tunable strategy

The overall tradeoff relationships when considering the four strategies are (roughly) summarized in Table I. In this table, the '$+$' indicates a factor that we consider to be a positive, while the '$-$' implies a negative factor. In this table, the HCHQ and LCLF strategies compensate each other except in terms of the data quality. Users can tune parameters for their applications in the tunable strategy so that (1) they can consume less energy in communication with possible energy consumption in encoding a dictionary ($+^*$) and (2) they can get generally excellent performance with possible and controllable sacrifice in data quality ($+^{**}$).

TABLE I: Strategies and tradeoff factors

	HCHQ	**LCLF**	Tunable
Size	$-$	$+$	$+$
Energy	$-$	$+$	$+^*$
Flexibility	$+$	$-$	$+$
Data Quality	$+$	$-$	$+^{**}$

IV. THE CHITCHAT PROGRAMMING MODEL

In this section, we present the CHITCHAT programming model to demonstrate how the conceptual model shown in the previous chapter can be implemented for application programmers. The CHITCHAT programming model focuses on the interoperability among IoT devices. For IoT application developers, interoperability is one of the main issues because (1) IoT devices have varying capabilities in data acquisition, data processing, communication bandwidth, and energy storage and (2) the software tools for IoT devices require different configurations, environments, and services when building applications.

The first conceptual integrity for interoperability in the CHITCHAT programming model is that 'every information is represented as a dictionary'. The key is a string type data, but the value can be any data type to represent the nature of the information to share among IoT devices. For example, it can be a floating point type value to represent sensor data, a boolean type value to represent the state of a system, or a string type value to store name or address. It can also be a user-defined type to compactly represent information.

The second conceptual integrity for interoperability in the CHITCHAT programming model is the common set of operations. Fig 5 shows the seven operations that programmers can use for controlling the life cycle of information they share.

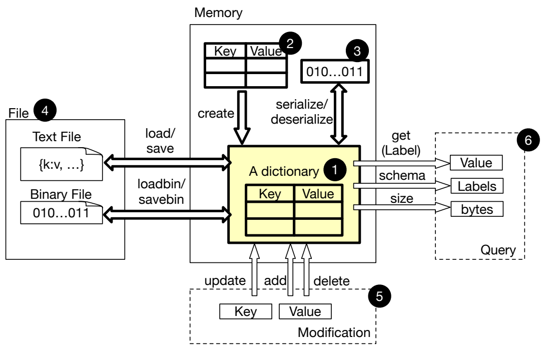

Fig. 5: CHITCHAT information sharing conceptual model

The information is represented as a dictionary ❶. The dictionary is created from any information ❷ selected to be shared or stored. For communication, the dictionary should be transformed into a serialized bit stream. Likewise, the received bitstream needs to be transformed into a dictionary ❸. The dictionary can be stored and retrieved both in a text or a binary file ❹. When the dictionary needs to be modified, keys or values can be added, updated, or deleted ❺. The information in the dictionary can be retrieved with a query operation ❻.

Programmers can use the same CHITCHAT programming model to program any IoT devices. This interoperability is possible because (1) the CHITCHAT programming model is implemented in the CISL, and (2) programmers need to specify the strategy that each IoT device should use when they program in the CISL.

For example, when programmers need to generate information sharing code for mobile devices using the CISL, they can tag **@android** to specify that the CISL generate the target code for Android devices. The default option, **@server**, does not generate any target code as the CHITCHAT language can be executed as a script on high-end servers. Programmers can also specify the strategy for the device. For example, when they need to use the HCHQ strategy for the Android device, they can tag **@strategy(HCHQ)**. When the application requirements demand small dictionary size, they can tag **@strategy(tunable, size)** to generate the code that encodes and decodes the dictionary with maximum size efficiency. The tag **@strategy(tunable, encoding energy)** can be used to generate the dictionary with minimum encoding energy.

A. CISL Language Design and Semantics

The CISL is a declarative, domain-specific language that was designed to be easy to learn, use, and understand. Fig. 6 shows the syntax definition of the CISL that defines the ability to add, remove, and update a value in the information summary. Fig. 7 demonstrates the semantics of the information sharing operations using set operations. Adding new summaries to existing ones is described using $\cup$, the set union operator. Deleting summaries is described using $\setminus$, the set difference operator. Updating summaries is described using $\rightarrow$, which designates existing elements being modified. D/Serializing summaries is described using $\Rightarrow$, which indicates transformation from/to a byte array.

We followed a minimalistic approach, introducing new constructs only if absolutely necessary, thus lowering the learning curve for the programmer; however, more complex features of the CISL, including annotations and inheritance, have been reduced to save space and streamline the presentation.

A set of original/created summaries, $S = \{s_1, s_2, \ldots, s_n\}$
A set of serialized summaries, $S^* = \{s_1^*, s_2^*, \ldots, s_n^*\}$
A set of updated summaries, $S^! = \{s_1^!, s_2^!, \ldots, s_n^!\}$
A set of added summaries, $S^+ = \{s_1^+, s_2^+, \ldots, s_n^+\}$
A set of deleted summaries, $S^- = \{s_1^-, s_2^-, \ldots, s_n^-\}$

$\langle s \rangle p$ denotes a summary s of program p
$\langle S \rangle p$ denotes a set of summaries S of program p

Fig. 6: Syntax definition

$$\text{AddSummary}(\langle S \rangle p, \langle S^+ \rangle p) = \cup_{i \in S} \langle i \rangle p \ \cup \ \cup_{j \in S^+} \langle j \rangle p$$
$$\text{DeleteSummary}(\langle S \rangle p, \langle S^- \rangle p) = \cup_{i \in S} \langle i \rangle p \setminus \cup_{j \in S^-} \langle j \rangle p$$
$$\text{UpdateSummary}(\langle S \rangle p, \langle S^! \rangle p) = \cup_{i \in S} \langle i \rangle p \rightarrow \ \cup_{j \in S^!} \langle j \rangle p$$
$$\text{SerializeSummary}(\langle S \rangle p) = \cup_{i \in S} \langle i \rangle p \Rightarrow \ \cup_{j \in S^*} \langle j \rangle p$$
$$\text{DeserializeSummary}(\langle S^* \rangle p) = \cup_{i \in S^*} \langle i \rangle p \Rightarrow \ \cup_{j \in S} \langle j \rangle p$$

Fig. 7: The information sharing operations.

B. High-Level CISL Programming

Using high-level abstraction, programmers can build their IoT applications without knowing configurations details. All they need to know is the strategy that each device should use and the target device where the generated code is executed.

Values and Dictionaries. The `value` keyword is used to define values; it requires parameters specifying the type(s) of the value(s) used the definition. Applications can create dictionaries and load them into memory using the `dictionary` keyword in the CISL language.

High Level Scripts. The CISL language provides various control structures including loops and selections. These structures, values, and dictionaries are used in functions to implement the features that IoT applications specify.

Listing 1 shows an CISL function example that calculates the average temperature from an array of dictionaries. Programmers can use the CISL provided functions such as 'get' and 'contains'. The features that were explained in Section IV are available in the form of functions.

```
function getAverageTemperature(dictionaries:dictionary[])
     : temperature = {
  var temp: temperature = 0.0; count: int = 0;

  for (dictionary in dictionaries) {
    if (contains(dictionary, "temperature")) {
      temp = get(dictionary("temperature"))
      count = count + 1
    }
    return (temp / count)
}
```

Listing 1: Function definition example (computes average temperature)

A CISL script is a unit of programming. Each CISL script has definitions and functions to describe the behavior of the program to implement a programmer's IoT application requirements. Listing 2 is an example that uses the function we defined in Listing 1 to alert users when the average temperature is more than 50°C. The dictionaries that contain the temperature information are in a serialized bytes (stream), so the stream input should be converted into an array of dictionaries using the deserialize CISL function. This script will generate Android source code from the **@android** tag. Also, this script will generate all the necessary code to make the dictionary size as small as possible, but without losing flexibility in information representation, because of the **@strategy(tunable, size)** tag.

```
@android
@strategy(tunable, size)
var averageTemp: temperature
var dictionaries: dictionary[]
dictionaries = deserialize(stream)
averageTemp = getAverageTemperature(dictionaries)
if (averageTemp > 50)
  alert("Abnormal temperature)
```

Listing 2: Script example (detects abnormal temperature)

C. Low Level CISL Programming

The CISL low level programming enables programmers to use the exposed core features of the CISL programming language.

Types and Filters. A type in CISL describes the innate properties of a value in a dictionary. In addition to the data types supported by CISL, programmers can define their data types for their applications. Applications that use CISL also define false-positive filters that constrain legal values of attributes in the information summaries. In our previous research [3], we explained that various situational filters can detect practically all false positives to enhance data quality dramatically. Programmers who have the requirements of (1) high data quality and (2) a small dictionary can add their filters to remove possible data degradation.

Script example. CISL provides low-level functions for programmers to control the life-cycle of the information sharing activities among IoT devices. Fig. 3 shows an example low level script that defines a function that converts a feature rich format, JSON, encoding into the representation in size efficient format, FBF, using a low level function such as 'toFbf' and 'toJson'.

```
function from_json_to_fbf(json_file:file, binary_file:
     file) = {
  dictionary = load(json_file); fbf = toFbf(dictionary)
  binary = serailize(fbf); save(binary, binary_file)
}
function transform_dictionary(
  binary_file:file, schema_description:file, json_file:
       file) = {
  dictionary = load(binary_file)
  json = toJson(summary, schema_description); save(json,
       json_file)
}
```

Listing 3: Script that uses low level functions

V. CISL Experiments

In this section, we simulate how developers build IoT applications that share information intelligently among IoT devices. The developers use the CISL language without knowing implementation details of various dictionary representations, but they are aware of the application requirements. They use four different strategy tags: @strategy(tunable, size), @strategy(tunable, encoding energy), @strategy(HQHS), and @strategy(LCLF) to meet their application requirements from device constraints. We compare size efficiency, false positive probabilities (data quality), and relative energy consumption for encoding using these strategies. We show how the CISL language facilitate intelligent interoperability among IoT devices.

For experiments, we used the same real-world information sharing situation as in our previous work [3]. In the research, we used a basic epidemic information dissemination protocol to simulate our open-air market. We used a city, modeled on Manhattan, with four open-air markets [3]. We tested with three different test dictionary sets. The first dictionary contains situational information, including markets summaries, describe inventory descriptions, opening hours, locations, and discounts, to be shared among both book sellers and buyers [3]. The second test uses the dictionary from our research [14]. In the research, we modeled streaming sensor data—count, date, time, location, name, id, value, and unit—to monitor the health of a structure. Application developers write a single CISL code to generate target languages. We use @android tag and @server tag to emulate the communication between

Android mobile devices and server devices. We measured the size reduction rate and data quality degradation rate from various options for sharing information among IoT devices.

Table II shows the size reduction when each strategy is compared with the HCHQ strategy. In our previous research, we showed that we can use internal table size to control the dictionary size and data quality [3] [15]. For the turnable strategy of maximum size reduction (size option), we set a smaller (width of 2) internal table size. For the tunable strategy of minimum encoding energy (encoding energy option), we used a large (width of 6) internal table size. It is difficult to measure the energy consumption accurately when encoding a dictionary [3], so we measure the time for encoding a dictionary to estimate the relative energy consumption.

The LCLF strategy shows the maximum size reduction. For tunable strategy with size option, we obtained a relatively worse data reduction rate compared to LCLF. However, considering the constraints of using LCLF, the size option gives benefits that can compensate for the reduced size reduction. With an encoding energy option, the overall size reduction is smaller than the results with a size option. However, energy consumed for encoding is dramatically smaller: generally, it took seconds to have a maximum size reduction rate when encoding with size option but milliseconds with an encoding energy option.

TABLE II: Size reduction rate comparison with HCHQ.

	LCLF	**Tunable**	
		size	encoding energy
Test1	74.2%	71.3%	60.1%
Test2	84.5%	76.7%	65.6%

Table III shows the data quality degradation rate of different strategies. There is no data quality degradation for the HCHQ strategy. However, with the LCLF strategy, we can decode information from received dictionaries only with pre-knowledge as we explained in Fig. 3. This explains why the degradation rate in a 'pre (knowledge) column' is 0%. Otherwise, the degradation rate becomes 100%, as in the data in a 'non-pre (knowledge) column', because we cannot decode information from dictionaries. Tunable strategies' data degradation is calculated from false positive probabilities as we explained in Fig. 4. The false positive probabilities are extremely low—practically zero—both with size and encoding energy options. In our experiments, we do not have any data degradation with a tunable option.

TABLE III: Data quality degradation rate

	HCHQ	**LCLF**		**Tunable**	
		pre	non-pre	size	encoding energy
Test1	0%	0%	100.0%	3.49×10^{-14}%	4.80×10^{-15}%
Test2	0%	0%	100.0%	2.71×10^{-15}%	3.73×10^{-16}%

These experiments show that IoT developers can use the CISL language for their applications that should meet different requirements such as small data size, high data quality, or small encoding energy, with simple tags added in their programs. The CHITCHAT programming model can alleviate coding burdens for application programmers.

VI. Conclusion and Future Work

The emerging technology of IoT has the potential to change not only people's lives, but also the industry, services, and society as a whole. However, context sharing among heterogeneous IoT devices requires different approaches than sharing context among homogeneous devices, because IoT devices have widely differing capabilities and priorities. In this work, we provide a conceptual model and strategies in sharing information among heterogeneous IoT devices to propose a homogeneous approach for IoT application developers. We propose a programming model, the CHITCHAT information sharing language (CISL), and its semantics to be used to implement the functionalities for sharing information. We showed how programmers can use the conceptual strategies with the CISL language to make the applications that meet the program requirements, and we assessed the tradeoffs of each strategy by analyzing the size reduction and data quality degradation quantitatively. As the future work for improving the CHITCHAT approach and language, we plan to conduct empirical studies with professional developers in industry settings requiring an adequate share of context knowledge in scalable, heterogeneous environment.

References

[1] J. Chase, "The evolution of the internet of things," Texas Instruments, White Paper, 2013.

[2] D. Evans, "The Internet of Things ," Cisco Internet Business Solutions Group, White Paper, Apr. 2011.

[3] S. Cho and C. Julien, "CHITCHAT - Navigating tradeoffs in device-to-device context sharing." in *PerCom'16: Proceedings of the 2016 IEEE International Conference on Pervasive Computing and Communications*. IEEE, 2016, pp. 1–10.

[4] P. Hu, J. Indulska, and R. Robinson, "An Autonomic Context Management System for Pervasive Computing," in *PerCom*.

[5] D. Salber, A. K. Dey, and G. D. Abowd, "The Context Toolkit - Aiding the Development of Context-Enabled Applications." in *CHI'99*. New York, New York, USA: ACM Press, 1999, pp. 434–441.

[6] C. Perera, P. P. Jayaraman, A. Zaslavsky, D. Georgakopoulos, and P. Christen, "MOSDEN: An Internet of Things Middleware for Resource Constrained Mobile Devices," in *HICSS'14*.

[7] C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca, "Context Information for Knowledge Reshaping," *International Journal Web Engineering Technology*, vol. 5, no. 1, pp. 88–103, May 2009.

[8] C. Băzăr and C. S. Iosif, "The Transition from RDBMS to NoSQL."

[9] mongoDB, "Top 5 Considerations When Evaluating NoSQL Databases," *White Paper*, pp. 1–9, Oct. 2013.

[10] C. G. García, B. C. P. G-Bustelo, J. P. Espada, and G. Cueva-Fernandez, "Midgar: Generation of heterogeneous objects interconnecting applications. a domain specific language proposal for internet of things scenarios," *Computer Networks*, vol. 64, pp. 143–158, 2014.

[11] A. Salihbegovic, T. Eterovic, E. Kaljic, and S. Ribic, "Design of a domain specific language and ide for internet of things applications," in *MIPRO 2015*.

[12] E. Grim, C.-L. Fok, and C. Julien, "Grapevine - Efficient situational awareness in pervasive computing environments." in *PerComW*.

[13] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal, "The Bloomier filter: an efficient data structure for static support lookup tables." in *SODA'04*.

[14] S. Cho and C. Julien, "Size Efficient Big Data Sharing Among Internet of Things Devices," in *PerCom Workshop BICA*.

[15] S. Cho, "Navigating Tradeoffs in Context Sharing Among the Internet of Things," Ph.D. dissertation, The University of Texas at Austin, The University of Texas at Austin, Aug. 2016.