2016 IEEE First International Conference on Internet-of-Things Design and Implementation

A Modular Approach to Context-Aware IoT Applications

Jagannathan Venkatesh, Christine Chan, Alper Sinan Akyurek, Tajana Simunic Rosing
Computer Science and Engineering
University of California, San Diego
La Jolla, CA, USA
{jvenkate, csc019, aakyurek, tajana}@eng.ucsd.edu

***Abstract*—The Internet of Things (IoT) refers to an environment of ubiquitous sensing and actuation, where devices are connected to a distributed backend infrastructure. It offers the opportunity to access a large amount of input data, and process it into contextual information about different system entities for reasoning and actuation. State-of-the-art IoT applications are generally black-box, end-to-end application-specific implementations, and cannot keep up with timely resolution of all this live, continually updated, heterogeneous data. In this work, we propose a modular approach to these context-aware applications, breaking down monolithic applications into an equivalent set of functional units, or context engines. By exploiting the characteristics of context-aware applications, context engines can reduce compute redundancy and computational complexity. In conjunction with formal data specifications, or ontologies, we can replace application-specific implementations with a composition of context engines that use common statistical learning to generate output, thus improving context reuse. We implement interconnected context-aware applications using our approach, extracting both user activity and location context from wearable sensors. We compare our infrastructure to single-stage monolithic implementations, demonstrating a reduction in application latency by up to 65% and execution overhead by up to 50% with only a 3% reduction in accuracy.**

Keywords – contextf-aware computing; internet of things

I. INTRODUCTION

Sensor networks and ubiquitous sensing are evolving into the Internet of Things (IoT) – the collection of sensing and actuation backed by the existing and growing Internet infrastructure [1]. This invalidates many re-IoT assumptions about this area – that compatibility and control over the sensors in the system can be assumed [2], or that applications use a manageable amount of raw data. The number of available sensing and actuation devices has grown rapidly in the last few years [3], for a truly pervasive sensing/actuation environment. In addition, ubiquitous connectivity and cloud storage have largely mitigated the primary research issues in the pervasive sensing fields. Communication and storage reliability issues focus on the application layer: IoT applications operate on changing inputs and available compute nodes as sensors and devices move through an application's domain.

IoT applications drive automated actuation, such as in smart environments, distributed microgrids, or user-centric automation. These context-aware applications operate on dynamically changing, ontologically-defined data called context – data whose type, range, and sources are specified in an interface. The current state of the art tailors end-to-end implementations of each application to their initial infrastructure and platforms, which hampers adaptation to the evolving number and types of sensing devices in deployment`. Smaller, simpler functional units fulfilling intermediate steps towards an overall application can alleviate scalability issues. Additionally, the state of the art [4] [5] places the burden of data processing on every individual application. This is inefficient when multiple applications process the same data using similar computation (e.g. both workplace and home automation process a user's GPS-based location and occupancy). Reliance on application-specific code also squanders the potential for designing and reusing general-purpose machine learning for multiple applications. In this work, we identify a novel approach to context-aware IoT applications: a general-purpose functional unit (context engine) which drives data processing for a single output context variable. We identify the theoretical advantages of modular applications for complexity and reduction of compute redundancy, and exploit the specification of data through ontologies and the single-output design of context engine to drive general-purpose machine learning. The context engines, implemented in middleware, each translate one level of heterogeneous input data into a single higher-level usable context, and are composed to form an equivalent application. We apply this approach and the context engine implementation to a case study: location monitoring and user activity using wearable sensors. We show 62% reduction in execution overhead and 50% reduction in high-order operations.

II. RELATED WORK

In the IoT, data collected through various devices goes through several levels of abstraction, combination, or distillation to produce context in discrete semantic states. This higher level context is used for visualization (e.g. quantified self [4], vehicular safety [6]) and actuation (smart spaces [7], ubiquitous computing [8], medicine [9], e-learning [5] [10]). In exchange for raw data precision, discretized context enables reasoning and data reuse. State of the art IoT development approach is end-to-end, not allowing data sharing and hence resulting in a disorganized data space. This necessitates the use of *ontologies*, or formal data representations, to maintain a unified, regulated data representation [11]. Ontologies are already in use in IoT systems such as smart spaces [7] [12] and semantic services

978-1-4673-9948-7/16 $31.00 © 2016 IEEE
DOI 10.1109/IoTDI.2015.13

[11] [13]. Ontologies focused on context representation include the Context Modeling Language (CML) [14], which attributes context to physical or virtual entities (objects) and provides domains associated with them (roles). It allows for dynamically changing input data by partitioning the input space into *dimensions* that can be used for reasoning.

Machine learning facilitates transforming input data into output actuation. K-means clustering is a prevalent way to automatically relate low-level data into high-level context [4]. Reinforcement learning (RL) allows users to reinforce and guide the system towards better accuracy. Madhu et al. [5] use constraint reasoning to and RL to find optimal custom reminders for impaired users. Rashidi et al. [15] perform unsupervised learning over low-level sensor event sequences to extract patterns for high-level activities. They focus on specific cases for the smart home over a known set of activities, but in a highly domain-specific manner.

Our takeaways from the related work are 1) IoT applications provide a multi-input multi-output (MIMO) interface between sensing and actuation, 2) applications operate in a dynamic sensor space: mobile sensing devices (e.g. wearables) and compute units (e.g. mobile phones) enter and leave the domain of a particular application [16], 3) ontologies are leveraged to provide platform independent data organization [12].

We identify two major challenges with the current view which we address in this paper: 1) Significant processing redundancy: applications using the same input may repeat the same computation. For example, user occupancy serves both home security and grid automation applications, and may be independently computed by both. *Our approach aims to expose this computation for reuse.* 2) Increase in complexity with input and output spaces: As application data grows, so does the computational cost of machine learning (ML) algorithms, whose complexity is dominated by inputs size. This in turn forces application-specific implementations. *By using smaller functional unit and enforcing a single-output approach, our system design facilitates the use of ML.*

III. System Design

Our main contribution is the design and implementation of an alternate approach to IoT applications: a hierarchy of multiple-input-single-output (MISO) functional units to improve reasoning while reducing data redundancy across applications and accomplishing the same functionality as MIMO applications. Exposing intermediate data reduces the complexity and redundancy of applications in the larger infrastructure. These improvements may come at the cost of accuracy, but we both quantify the additional error and illustrate how it can be minimized. Modularization generates intermediate context that can be shared among applications (see Figure 1.b). The smaller hierarchical functional units represent a simpler data translation compared to the overall application, facilitating the use of general-purpose machine learning to perform data transformation and reduce application-specific code.

IoT applications consume raw, noisy data about both physical and virtual entities from heterogeneous sources and process them in applications into usable information. They also extract and produce context: high-level abstracted data. In the IoT, context tends to be human-centric interpretations (e.g. location) that are important to many applications [17]. Black-box application implementations mask intermediate processing output from other applications, which leads to compute redundancy. Our hierarchy of functional units in place of monolithic implementations trades off compactness for versatility. A single application is broken up into multiple functional units. Although this serialization can increase organizational complexity and latency of a highly compact algorithm, it can also expose intermediate output for reuse by other applications, thus reducing compute redundancy. We prove that it also decreases overall compute complexity and reduce input processing and functional order when certain conditions are met (Section III.A).

A. Context Engine Architecture

We first specify the input space of each context engine as a set of context variables. We define a context variable as the individual input or output data unit, and leverage the variable's space representation using ontologies to define the domain/range of the variable, and subsequently, the context engine.

While current monolithic applications may have internal modularity and parallelism, they are hidden from the rest of the system. The MIMO implementation of a current IoT application can be explicitly broken down into several context engines, which decompose its internal functionality into smaller, MISO functional units. The composition of the context engines produces the same outputs as the original application (see Figure 1.b). This reduces the complexity of each context engine, as each performs less processing than the single-stage engine and requires fewer inputs and produces fewer outputs. For IoT applications, this increases the overall versatility, as the now-visible functional units

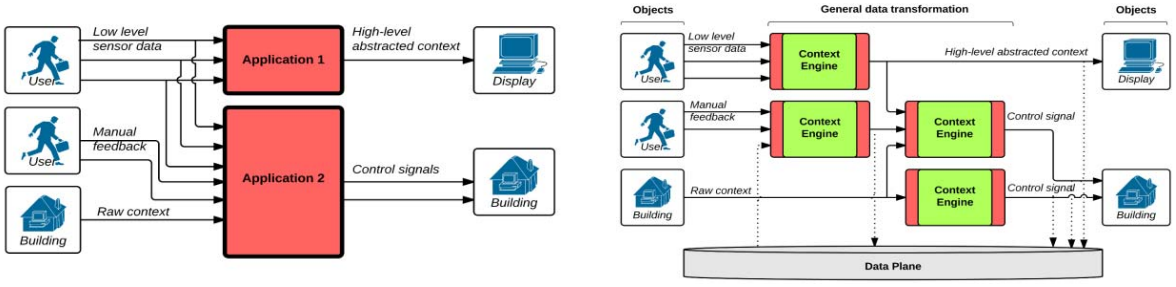

Figure 1. (a) The current state-of-the-art: monolithic application implementation. (b) Our implementation: Applications publish intermediate context for reuse. Functional units (context engines) are multi-in-single-output performing general statistical learning.

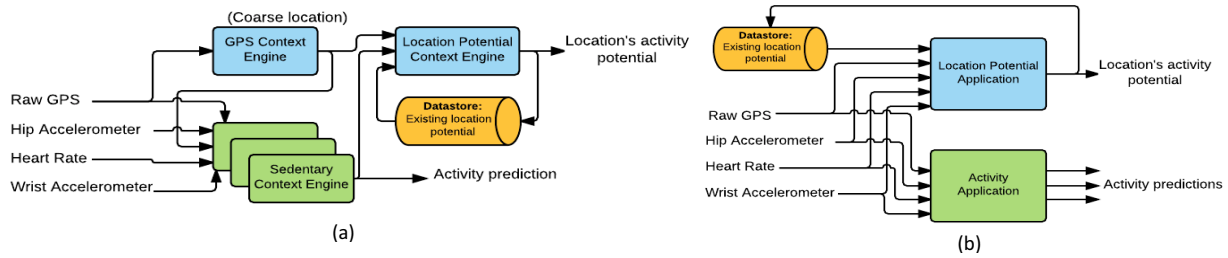

Figure 2. (a) Sequential context engine applications and (b) consolidated applications for user activity and location potential detection.

and the resulting dependency graph can be deterministically or automatically distributed and parallelized among available compute nodes by the IoT management system.

This sequential hierarchical approach raises questions about the complexity overhead, latency, and accuracy of breaking down a possibly compact application. We validate our approach by proving that the overall computational complexity of the architecture is actually reduced with a marginal impact on output accuracy. We have previously demonstrated the use of matrix-based stochastic learning models to perform the data translation within the context engine [21]. We leverage TESLA, a model designed for forecasting, as our context engine's ML implementation. It provides efficient model generation: $O(n^{\alpha})$, where n is the number of inputs and α is the function order of the Taylor expansion. The generic function of this expansion is:

$\sum_{i=0}^{n} C_i x_i$ (1st order), $\sum_{i=0}^{n}\sum_{j=0}^{i} C_{ij} x_i x_j$ (2nd order) etc. (6)

where C_{ij} represents individual coefficients learned once observations are determined, and $x_0 = 1$ (the constant coefficient). The resulting equation is $Ax = B$, where A is the row matrix of input observations; x is a column vector of coefficients, and B is a column vector of output observations correlating with the corresponding row of A, and solved by least squares estimation. We use TESLA to derive complexity and error in the theorems below.

Theorem I. Dividing a context engine into multiple context engines decreases the total computational complexity of a nonlinear system.

Proof. We start with a general representation of a context engine: *N* number of inputs and a computational complexity order α for a maximum computational overhead N^{α}. We divide the single engine into two stages, where there are multiple engines with an arbitrary number of inputs of A. The number of engines of the first step becomes $\frac{N}{A}$. The second stage takes the outputs of the first stage to provide the final output. The total complexity overhead of this system is $\frac{N}{A}A^{\alpha} + (\frac{N}{A})^{\alpha}$. The conditions where the two-stage has a lower complexity than the single engine are:

$$\frac{N}{A}A^{\alpha} + (\frac{N}{A})^{\alpha} < N^{\alpha} \rightarrow A^{2\alpha-1}N + N^{\alpha} < A^{\alpha}N^{\alpha}$$

$$A^{\alpha-1}A^{\alpha} < N^{\alpha-1}(A^{\alpha} - 1) \rightarrow (\frac{N}{A})^{\alpha-1}(1 - \frac{1}{A^{\alpha}}) > 1 \quad (1)$$

Although the selection of A is arbitrary, there are two limiting conditions: A must be an integer and the number of context engines must be an integer $(\frac{N}{A})$. Thus, the minimum for A is 2 and the maximum is $\frac{N}{2}$, i.e. 2 engines. We do not consider *A=1* or *A=N*, as neither creates multiple context engines. The final inequality is the multiplication of two terms. The first term is minimized when $A = \frac{N}{2}$ and results in $2^{\alpha-1}$. The second term is minimized when $A = 2$ and results in $1 - 2^{-\alpha}$. This provides a lower bound for the result: $2^{\alpha-1}(1 - 2^{-\alpha}) = 2^{\alpha-1} - \frac{1}{2}$. If this bound satisfies the inequality, the multiplication result must also satisfy it.

$$(\frac{N}{A})^{\alpha-1}(1 - \frac{1}{A^{\alpha}}) > 2^{\alpha-1} - \frac{1}{2} > 1 \rightarrow \alpha > log_2 3 \sim 1.6 \quad (2)$$

This proves that if the complexity order of the system is greater than 1.6 (e.g. for 2nd and greater integer function orders), any arbitrary division of the single engine results in a decrease in computational complexity. The corollary is:

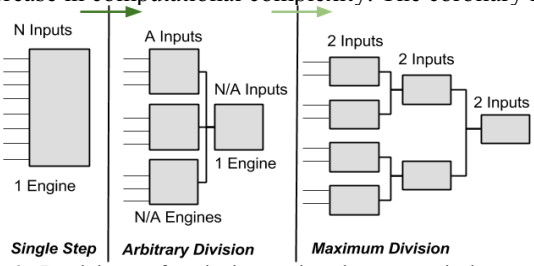

Figure 3. Breakdown of a single-step into lower-complexity equivalent reductions, with minimum complexity occurring with maximum division.

Theorem II: The complexity of a system of context engines is minimized when each individual engine contains 2 inputs.

Proof. Theorem I shows that dividing an engine decreases complexity if the system has a complexity order greater than 1. The number of context engines is $\frac{N}{A}$, which gets its maximum value at $A = 2$.

While context-aware applications do not necessarily fit perfectly into a system of two-input engines, the more we reduce inputs into each CE, the greater the reduction in overall system complexity.

We then study the accuracy change between sequential and consolidated applications. We use a general representation of the machine learning algorithm in a context engine (e.g. polynomial model [18]). While a general formulation depends on the functional order and application, we illustrate the accuracy change between 2-input context engines and a single 4-input context engine in Figure 4.

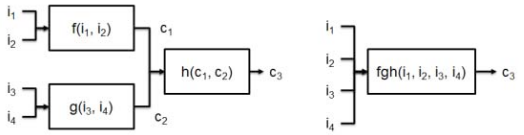

Figure 4. Functional comparison of sequential (left) and single-stage (right)

We can compare the two implementations through their transformation functions. The second-order Taylor expansion of the data transformation (Section III.B) is f:

$$f(i_1,i_2) = f_0 + f_1 * i_1 + f_2 * i_2 + f_{11} * i_1^2 + f_{22} * i_2^2 + f_{12} * i_1 * i_2 \quad (3)$$

where f_{ij} are the corresponding coefficients. The other context engines ($g, h,$ and fgh) have corresponding expansions. However, because h is composed of the outputs of f and g, it can be represented as a function of the f_{ij} and g_{ij} coefficients and the initial input variables i_1 to i_4. This is directly comparable to fgh, which is also a function of the initial inputs and a set of coefficients represented as λ_{ij} (e.g. $\lambda_0 + \lambda_1 * i_1 + \ldots$). $h(f,g)$ evaluates to fgh exactly except for the ratio of g_1 and g_2, which matches two *different* pairs of coefficients of fgh:

$$\frac{g_1}{g_2} = \frac{\lambda_{13}}{\lambda_{14}}, \frac{g_1}{g_2} = \frac{\lambda_{23}}{\lambda_{24}} \quad (4)$$

This means that the sequential context engine will have the same accuracy as the single-stage only when the ratio of λ_{13} to λ_{14} matches the ratio of λ_{23} to λ_{24}. If the ratios do not match, 1 out of the 4 coefficients in this ratio cannot be represented, though the other 3 in the ratio as well as the remaining 7 coefficients in the equation are all represented accurately. This is modeled as an error factor δ in the λ_{24} coefficient, which contributes $\delta * i_2 * i_4$ *truncation error*.

We also quantify the impact of input signal noise on the sequential context engine compared to the single-stage approach. We model each input with a zero-mean additive white Gaussian: $x_i + w_i$, a common expression of sensor noise [19]. The resulting noise propagates through the applications. In the sequential case, f and g both propagate the original noise to h. The truncation error of the sequential context engine is now compounded by additional noise:

$$\delta * i_2 * i_4 + \delta * w_2 * i_4 + \delta * w_4 * i_2 + \delta * w_2 w_4. \quad (5)$$

The first term is the truncation error we previously quantified; the second and the third terms are scaled Gaussian values due to noise; and the last term is a chi-squared distribution also derived from noise. The significance of the error terms entirely depends on the relationship between the cross-product input terms i_2 and i_4. If the output context is highly dependent on the cross products, the weight of the noise and truncation terms will pose significant error. Simply selecting highly correlated input terms for context engines - an intuitive choice nonetheless - will mitigate truncation error, as the impact of the missing cross-coefficient terms is minimized.

IV. Case Study: User Activity

For our case study, we investigate applications that leverage user activity prediction. User activity is important across several domains: connected/reactive spaces, the smart grid, social behavior understanding, etc. all provide output actuation (e.g. grid demand-response, home automation). We therefore investigate both applications that output user activity predictions and applications that use activity to provide output actuation for a particular space/domain. In particular, the latter application is used to determine the potential for a location to be used for activity or exercise.

Input Data: We use sensor data collected from wearable sensors from the UCSD Personal Activity and Location Measurement System (PALMS) [22], which provides high-fidelity wearable data such as from fitness trackers: hertz GPS and heart rate data, and 30Hz data from wrist and hip accelerometers for 40 individuals, with the activity annotated through observation of the individuals. The annotations fall into four categories: the activity; the posture of the participant; whether it constitutes social interaction; indoor/outdoor. The activity is chosen from a set: eating, TV, leisure, sports, exercise. Posture and indoor/outdoor are binary values associated with each activity.

Applications: We consider two applications: one specific to the users wearing the devices, and one specific to the spaces users are moving in. The user-centric application is activity prediction: translating raw sensor data into high-level activity definitions. The location-centric application is health potential – the potential for a particular location to raise the user's heart rate. While the former application's output is specific to the user, the latter can help guide users towards better behavior: identify the possibility of taking stairs or improving their daily energy expenditure. This forms a consolidated IoT environment – users whose sensor data impacts the spaces around them, and conversely, smart spaces that provide actuation to the users that enter them.

Data Translation & Outputs: The state of the art data translation is straightforward: separate applications take in all the available data as input, and produce the activity potential and activity prediction, respectively (see Figure 2(b)). Our approach modularizes the problem into three sets of context engines: generating coarse GPS location, detecting activities, and identifying each location's activity potential. In keeping with the MISO principle, we allocate a separate context engine for each activity. We also have separate activity detection engines for each of the 40 users, leveraging the fact that the users' fitness trackers are embedded devices that can detect personalized activity.

As GPS information is important, but the raw GPS data is too fine-grained for either application, we introduce a GPS context engine, which outputs a coarser latitude/longitude reading showing a larger physical space. This intermediate variable is defined as a latitude/longitude pair, albeit with less fine-granularity. The reference data to train this context engine is derived by spatially clustering the raw GPS data and using the northwest point of each cluster.

The output prediction is a boolean for each potential activity. Similarly, a location's activity potential is a boolean. Both values are trained using available annotated data, or ground truth, from PALMS. The location's activity potential is stored in the datastore (a key-value cloud database) with the location as the lookup key, and is fed back into the context engine as another input context.

Context Engine Setup: Figure 2 illustrates the configurations for both the sequential context engines (a) and the current state of the art single-stage application (b). The single-stage applications, as monolithic black-boxes, require all the input data. The sequential approach can be designed more judiciously. The GPS context engine's output supplants the raw GPS latitude/longitude data in both

applications, though each activity's context engine requires the speed and satellite count from the raw GPS data. The GPS context engine, with one input, has *O(1)* complexity.

The second stage of the sequential activity application transforms the available input from the original data sources and the GPS engine to a boolean representation of the respective activity. Each of these second-stage context engines have n inputs and a computational complexity of $O(n^a)$ for generating output, where a is the function order. The third stage is the entire location potential application, as it *only* uses intermediate data produced by the other context engines (including feedback from itself). As activity detection context engines can grow in number as more activities are added, the computational complexity is $O(kn^a)$, where k is the number of detectable activities.

The single-stage applications (Figure 2(b)) are used to compare complexity and accuracy against our approach. They are also run using a general data transform defined by a polynomial function order. Since both applications take in all the available inputs, their complexity is similar to the second-stage context engines: $O(n^a)$. As each context engine uses different inputs and generates different outputs, we test each with TESLA up to 3rd-order functions, after which accuracy improvements are marginal. From the PALMS dataset, we extract contiguous time-series data, interpolating each as necessary to provide correlated training and test samples. To test the impact of the number of samples on functional order, we vary the number of samples up to 8,000 (two days), and test against 4,000 (one day) samples.

A. Results

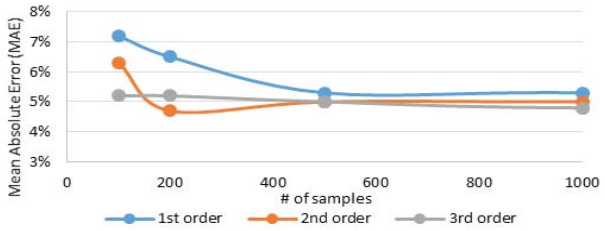

Figure 5. MAE of GPS context engine over function order and sample size

Complexity: The GPS context engine is the first stage for both sequential applications, as its output is consumed by following stages. The single engine was tested across different functional orders of the TESLA algorithm, where it performs well with first-order, with marginal improvements for second- and third-order functions (Figure 5), for a complexity of *O(1)*. The activity context engines are used by both applications. Since some activities are more readily predictable than others, their functional orders vary from 1st (*Indoor*) to 2nd (*Sedentary, Bike, Exercise, In Vehicle*), and 3rd (*Walk, TV, Eating*). Figure 6 illustrates the change in the mean absolute error with each change in functional order.

Some statistical learning results were inaccurate even when taken up to 3rd order. For example, *eating* shows marked similarities to *walking*, with the exception of speed, and as such, reports false positives even at 3rd order. *Walking* incurs accuracy issues when indoors, due to reduced GPS accuracy. This is compounded by treating each correlated input set as an instance, whereas *walking* is better represented by a time-series. Implementing TESLA in this manner is outside the scope of this work, but the methodology can be found in [18].

Other activities can be determined with lower function order: most of the other activities are accurately predicted with 2nd order polynomials, and determining that the user is *indoors* can be determined with 1st order. The worst-case complexity for the set of activity context engines is $O(kn^3)$. In contrast, location potential context engine can generate accurate output (9.2% MAE) by a 2nd order function $O(n^2)$.

Each single-stage application uses several TESLA algorithms on all inputs, simulating general-purpose MIMO learning, with a time complexity of $O(kn^3)$. Even with the same worst-case complexity, our approach has an advantage in execution overhead: All context engines recalculate the output whenever a new input observation is recorded, but the 2nd stage of our approach reacts only to changes in coarse location, which is much less frequent than the correspondingly subtle changes to the raw GPS data. Table 1 highlights the computations performed by the single-stage and sequential applications over the test set.

Table 1. Execution overhead based on iteration count for the context engines associated with the In-Vehicle activity.

In Vehicle Prediction	**Number of computations**		**Total Latency**
	1st Stage $O(1)$	**2nd Stage $O(n^3)$**	
Sequential	3670	1823	0.24 sec
Single-Stage	--	3670	0.37 sec

The sequential application naturally performs more total computations than the single-stage approach. Still, it exhibits only 65% of the single-stage latency. This is in part because of distributed computing: the first stage consists of data-independent context engines that can be parallelized. More importantly, because of the modularized functional units, the sequential context engine offloads the processing of raw GPS to the low-overhead GPS context engine. The single-stage context engine has no choice but to perform over 3000 $O(n^3)$ computations. The sequential application

Figure 6. Mean absolute error (MAE) for each activity context engine across different function orders.

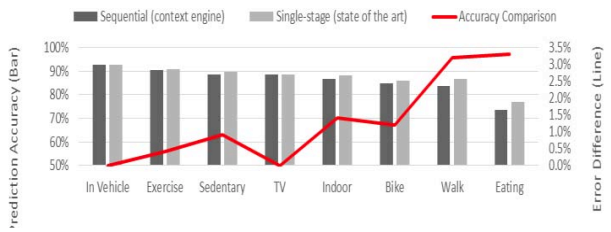

Figure 7. Accuracy comparison between the sequential and the single-stage activity applications (bar graph) with the delta in accuracy (line graph).

requires fewer than 50% of the time- and compute-intensive $O(n^3)$ computations, and consequently, can complete the work 35% faster. This difference in latency carries over to all other activity context engines, with a speedup of up to 2.6x for the *Indoor* activity engine, which has only a 1st order function. Table 2 compares the latency of the context-engine-based applications normalized against single-stage.

Table 2. Latency of the sequential applications grouped by function order

Function Order (Context Engine)	Norm. Latency
1st (Indoor)	0.38
2nd (Sedentary, Bike, Exercise, In Vehicle)	0.49
3rd (Walk, TV, Eating)	0.65

Accuracy: Figure 7 compares the accuracy of each of the individual activity context engines against the consolidated activity application. The application sets have very similar accuracy numbers, with the highest error difference between them being 3.3% (for *eating*). This is explained by the high similarity in their input sets, with the only difference being the use of the intermediate output of the GPS context engine. Since only a small number of the original cross-coefficients in the statistical learning algorithm are missing, the resulting difference in error is low. The consolidated location potential application is significantly different from the equivalent context engine. This is an ideal example for comparison, as instead of the raw input data, the context engine relies on only *intermediate* data – the outputs of the activity and GPS engines. Table 3 quantifies the accuracy comparison of sequential and single-stage applications.

Table 3. Output accuracy comparison for Location Potential between the sequential context engine and the single-stage application

Application	Output Prediction Accuracy
Sequential	79.9%
Single-Stage	74.2%

Despite using only intermediate data, there is only a 5.7% reduction in output accuracy. This reduction illustrates one of the conclusions from Section III.A: organizing inputs and outputs of the modular application appropriately reduces truncation error. Intuitively, the location's activity potential is related to the activities that are performed in that location, and as such, using the existing intermediate output provides reasonable accuracy compared to reusing the original data.

Application Extension: The location potential application shows the extensibility of our approach. The single-stage approach (Figure 2(b)) needed an entirely new $O(n^3)$ application with $n=5$ to produce the same output as a $O(n^2)$ application with $n=3$ using already-generated output in the sequential approach (Figure 2(a)). Growing an ecosystem that makes use of existing intermediate output reduces the computation needed for new applications.

V. CONCLUSION

We motivate and design a novel approach to IoT applications. State of the art implementations are monolithic [1], with computational redundancy between applications and increased aggregate compute complexity. Our modular framework exploits common computational processes between applications to expose shareable intermediate context. Decomposing a single-stage functional unit into a sequence of smaller ones increases reusability, reduces complexity at a minor cost in accuracy, and facilitates general data transformation. We implement a statistical learning technique that exploits ontological data to construct and train a model. A case study using our approach in different context-aware applications demonstrates the versatility and efficient data reuse of the context engine, and up to 65% latency improvement with minimal accuracy loss.

ACKNOWLEDGEMENTES

This work was supported by STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, through TerraSwarm center and NSF grant CNS-1446912. The DIAL data was collected with funding from National Cancer Institute Grant # 1U54CA155435.

REFERENCES

[1] C. Perera et al, "Context Aware Computing for the Internet of Things: A Survey," *IEEE Communications, Surveys, & Tutorials,* 2013.
[2] M. Friedewald and O. Raabe, "Ubiquitous computing: an overview of technology impacts," *Telematics and Informatics,* 2011.
[3] J. Hammer and T. Yan, "Poster: A virtual Sensing Framework for Mobile Phones," in *Proceedings of MobiSys*, 2014.
[4] J.-H. Hong et al, "Conamsn: A context-aware messenger using dynamic bayesian networks with wearable sensors," *Expert Systems with Applications,* 2010.
[5] S. K. Madhu et al, "An Ontology-based Framework for Context-Aware Adaptive E-Learning System," in *ICCI*, 2013.
[6] Lee and K. et al., "AMC: Verifying User Interface Properties for Vehicular Applications," in *Proceedings of MobiSys*, 2013.
[7] H. Chen et al, "An Ontology for Context-Aware Pervasive Environments," *The Knowledge Engineering Review,* 2004.
[8] S. Lee and K. C. Lee, "Context-prediction performance by a dynamic bayesian network: Emphasis on location prediction in ubiquitous decision support environment," *Expert Systems w. Applications,* 2012.
[9] M. Rudary et al, "Adaptive cognitive orthotics: combining reinforcement learning and constraint-based temporal reasoning," in *ICML*, 2004.
[10] Google, "Google Now," Google, 2014. [Online]. Available: http://www.google.com/landing/now/. [Accessed 10 November 2014].
[11] S. Staab and R. Studer, Handbook of Ontologies, Springer Science and Busines, 2010.
[12] T. Gu et al, "An Ontology-based Context Model in Intelligent Environments," in *CNDS*, 2004.
[13] W. e. a. Wang, "A comprehensive ontology for knowledge representation in the internet of things," in *IEEE TrustCom*, 2012.
[14] M. Nebeling et al, "XCML: providing context-aware language extensions for the specification of multi-device web applications," *WWW,* 2012.
[15] P. Rashidi et al, "Discovering activities to recognize and track in a smart environment," *IEEE TKDE,* 2011.
[16] S. Bandyopadhyay, et al, "A Survey of Middleware for Internet of Things," *Recent Trends in Wireless and Mobile Networks,* 2011.
[17] C. Perera et al, "Context Aware Sensor Configuration Model for Internet of Things," in *ISWC*, 2013.
[18] B. O. Akyurek et al, "TESLA: Taylor expanded solar analog forecasting," in *SmartGridComm*, 2014.
[19] S. Zahedi and C. Bisdikian, "A framework for QoI-inspired analysis for sensor network deployment planning," in *WICON*, 2007.
[21] J. Venkatesh et al: "A Context-Driven IoT Middleware Architecture," *TechCon,* 2015.
[22] K. Ellis, et al, "Identifying Active Travel Behaviors in Challenging Environments Using GPS, Accelerometers, and Machine Learning Algorithms," in *Frontiers in Public Health*, 2014.