# Towards a Scalable Architecture for Building Digital Twins at the Edge

**Khaled Alanezi**
Department of Computing
College of Basic Education,
PAAET, Kuwait
kaa.alanezi@paaet.edu.kw

**Shivakant Mishra**
Computer Science Department
University of Colorado,
Boulder, USA
mishras@colorado.edu

## ABSTRACT

This paper presents a system architecture for building digital twins at the edge. In particular, it addresses two major challenges: system scalability and complexity. It introduces two novel architectural components: a Context Aware Communication Component (CACC) that addresses the scalability issue in communication between physical and virtual environments, and second, an application-agnostic methodology, a service registry component for integrating digital twin with EdgeAI (DT-EdgeAI integration) at the edge. The paper describes this architecture in detail and provides a preliminary prototype implementation.

## Author Keywords

Digital Twin Architecture; EdgeAI; Scalability; Complexity

## CCS Concepts

**•Computing methodologies → Distributed computing methodologies;**

## INTRODUCTION

In recent years, computer systems researchers are observing a surge in the number of architectures utilizing the concept of a digital-twin (henceforth DT) especially in IoT. An IoT deployment can benefit from DTs by creating a digital replica for the IoT environment consisting from virtual replicas of IoT nodes, communications between them and the environment they are deployed in. This digital replica of the complex IoT system can then be utilized for analyzing the performance of the system, monitoring the environment for problems and studying the impact of alternative mitigation scenarios before applying them. Take for example the impact of a downtime for a system upgrade in a factory or closing an intersection in a city which can be studied by means of the virtual DT environment before actually performing the change physically.

At present, digital twins for IoT environment have mainly been approached through cloud-driven technology for domain specific architectures [1, 2, 3]. In this approach, the digital replica is implemented in the cloud and the communication layer involves communication from the physical environment to the cloud over the Internet. However, this approach of building digital twins in the cloud suffers from several critical problems: (1) Communication between the IoT sensors on the field and cloud incurs high latency. This is a critical problem particularly for digital twins because low communication latency between physical and digital entities is of paramount importance to ensure a close, near realtime synchronization between physical environments PE and virtual environments VE. (2) Computing on the cloud incurs high bandwidth cost, since all the raw sensor data needs to be transmitted over the Internet, and modern IoT systems can comprise of thousands and even tens of thousands of sensors transmitting continuously rich IoT environmental data. (3) Several IoT systems such as smart agriculture are often deployed in remote areas where Internet connectivity is at best intermittent. In such scenarios, it is challenging and sometimes even infeasible to maintain a high-fidelity virtual replica of a physical system where VE is implemented in the cloud.

*SEC'23,* December 6-9, 2023 Wilmington, DE, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 979-8-4007-0123-8/23/12...$15.00
DOI: 10.1145/3583740.3626807

To address these problems, there is a need for research in building digital twins at the edge. In this scenario, Edge Artificial Intelligence (EdgeAI) is a prime component. By means of EdgeAI, machine-learning models are trained using large volumes of data on the cloud and pushed to the edge of the network near the data sources to enable low-latency access to classification tasks. These concepts of big-data, IoT, edge computing and DTs did not emerge in isolation however. For example, DTs are seen as a technology that followed the advances in IoT, big-data and AI [4].

Typically, an IoT system can be mapped into three layers as can be seen in Figure 1. The figure also lists possible alternative implementation scenarios for three example IoT applications in each of the three layers. First, the perception layer is responsible for sensing and actuation. Take for example gathering medical information about a patient in case of an Internet of Medical Things (IoMT) solutions or from a robot in a factory in Industrial Internet of Things (IIoT). Second, the edge layer provides access to localized AI models (edge AI) near the data sources. Applying AI on the edge ensures low-latency access to time-sensitive AI models such as real-time control in a factory or irrigation control in a smart farm. Third, by having a birds-eye-view for multiple IoT deployments, the cloud

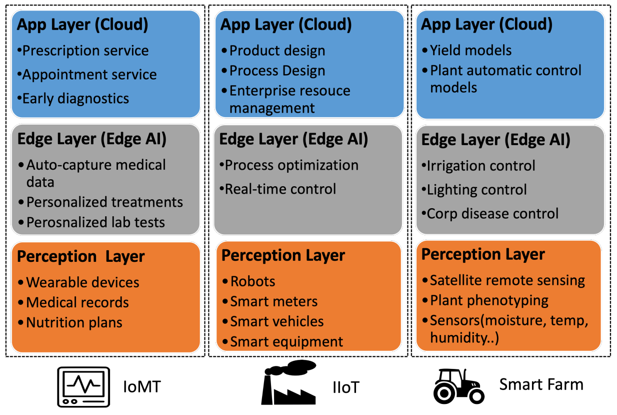

**Figure 1. Motivating Scenarios**

environment can develop AI models trained on big data such as generic early diagnosis models for patients. These trained models can then be pushed to the edge layer for localized use. Indeed, the availability of a DT replica will benefit these scenarios for several reasons. First, a DT data model must strive to only replicate necessary sensory data for the Edge AI model thereby saving communication energy to prolong the IoT node lifetime. Second, communication bandwidth and storage at the cloud will also be saved by minimizing the amount of data reaching into the cloud servers. Finally, the DT data model must also be designed so as to limit location specific data reaching into the cloud for better privacy and security.

Despite showing clear benefits, developing IoT architectures fully utilizing the above concepts remains far from easy due to technical and scalability challenges. An IoT architecture is a complex distributed system and deploying applications on top of the described perception-edge-cloud architecture is inherently difficult. In addition, utilizing advanced concepts such as DTs and EdgeAI further exacerbates the problem of complexity. Next, modern IoT applications now-a-days are routinely comprised of tens of thousands of sensors transmitting rich sensor data over a wireless medium with limited bandwidth. As a result, both the communication between the physical and virtual environments and performing complex computations at the edge become scalability bottlenecks. To tackle these issues, we propose a modularized architecture for developing digital twins at the edge allowing developers to focus on the IoT application and digital twin logic.

In particular, we present an architecture that specifically addresses the the scalability and complexity issues. This architecture incorporates two important features: First a Context Aware Communication Component (CACC) that addresses the scalability issue in communication between physical and virtual environments, and second, an application-agnostic methodology, a service registry component for integrating digital twin with EdgeAI (DT-EdgeAI integration) at the edge. The paper describes this architecture and provides some preliminary performance numbers.

## PROPOSED ARCHITECTURE

The proposed architecture as shown in Figure 2 utilizes the model of a **cloudlet server** [5]. In this model, a server is placed near the edge nodes to provide low-latency access to computation resources such as storage and CPU cycles. The **IoT Node** is a microcontroller with sensors and actuators installed on board as required by the IoT application. Data from IoT nodes' sensors in the physical environment is replicated to the corresponding DTs in the virtual environment through the communication layer. This replication happens by means of the **Context-Aware Communication and Control (CACC)** component which employs various techniques for ensuring the scalability of the architecture when dealing with large volumes of sensory data. We describe the CACC component in details in Section 3.

The **Digital Twin Service** is responsible for handling the complete lifecycle of DTs in this architecture including DT creation, state update, state inquiry, code triggers, and DT termination. Note here that we employ Eclipse Ditto [6] which provides all of the aforementioned DT functionality. Ditto uses a Device-as-a-Service approach in which RESTful **DT APIs** are exposed whenever a DT is created. These APIs can then be used to mirror dynamic state data of the DT whenever its physical counterpart is updated in the physical environment. Note here that created DTs can be of DT instance (DTI) type or a DT aggregate (DTA) type [7]. DTI is a virtual replica focusing on a single aspect of the physical object whereas DTA includes multiple DTIs to create an exact virtual replica covering all aspects of the physical object. The DT service also provides **Service Triggers**. Those are callback methods that are invoked whenever a change happens to a particular DT based on the DT identifier.

While the digital twin service is responsible for maintaining a digital replica of the physical environment, we need an architectural component to link the digital replica with appropriate edgeAI service to analyze the digital replica state and trigger appropriate actuation in the physical environment via CACC based on this analysis. To this end, our architecture introduces the **Service Registry** component.

The service registry component integrates the digital replica with appropriate edgeAI service in an application-agnostic manner. It provides a lookup service to discover and call Edge AI components available on the edge server. We use Eureka server [8] to implement this service. Whenever an Edge AI microservice such as the components listed in Figure 1 is deployed, the host, port number and name of the service are registered with the Eureka server. Such action enables the service triggers in the Digital Twin Service to search for services and invoke them by name without worrying about their implementation details. Using this feature, a DT can define in its static metadata the services that must be called whenever a change in its state occurs. Accordingly, the service triggers look for those services and invoke them programmatically.

Finally, as a result of executing the edge AI components specific interventions must be introduced to physical environment. These interventions are planned by the CACC component

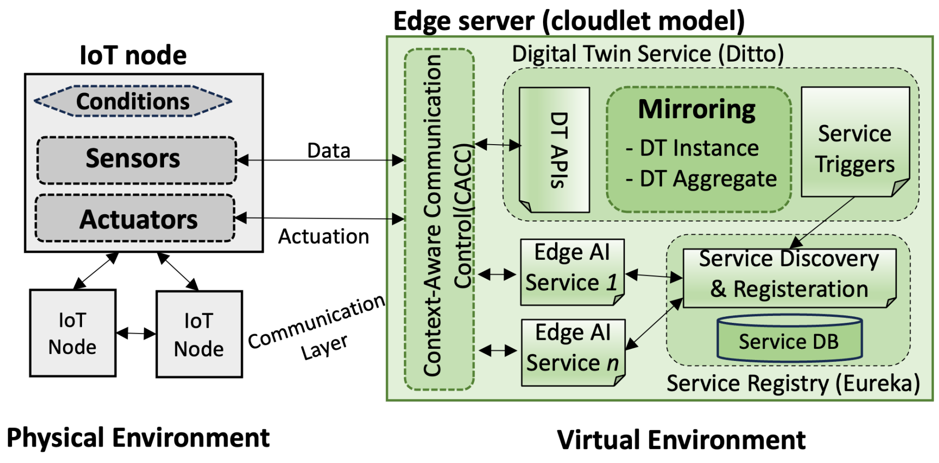

**Figure 2. Digital Twin Architecture Spanning the Physical and Virtual Environments**

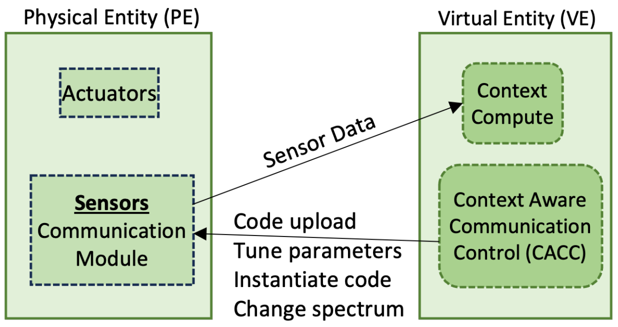

**Figure 3. Context Aware Communication Control**

based on the Edge AI input and subjected to the environment by means of calling the necessary sensors and actuators.

## CONTEXT AWARE COMMUNICATION CONTROL (CACC)

Communication component between the physical IoT environment and its DT at the edge is a critical component in a DT architecture providing the bidirectional transfer or sharing of data between them, including quantitative and qualitative data (related to material, manufacturing, process, etc.), historical data, environmental data, and most importantly, real-time data. This component is the key to (1) maintaining near real time synchronization between the physical environment and its digital twin, and (2) ensuring a timely actuation of appropriate actions in physical environment based on prediction/analysis at the edge. So, a key challenge in building DTs at the edge is managing the communication component under heavy load with limited network bandwidth to ensure that physical environment and its DT remain well-synchronized and timely actuation occur. The key issue here is how do we handle large volume of raw sensor data being pushed from physical environment in face of limited network bandwidth? A plethora of flow control techniques exist for both wireline and wireless protocols to address congestion problem by automatically slowing down the sender to prevent it from overwhelming the receiver. However, these flow control techniques are either insufficient or ill-suited for building DTs at the edge for a number of reasons. First, a DT is designed to provide appropriate actuations based on the current and predicted future behavior, which it learns from analyzing the real time sensor and other data it receives from physical environment. To accurately compute the current context, it may need to prioritize the transmission of sensor data from one part of physical environment over the other parts under specific conditions. For example, in case of an unrest in a smart city, it would be preferable to receive higher quality sensor data much more frequently from the location where the unrest is taking place than from other locations. Traditional flow control techniques do not directly support such differentiated flow controls. Second, in bandwidth saturation situations, DTs will prefer to receive application-dependent pre-processed data from the source instead of receiving raw sensor data at a lower pace (possibly) with dropped packets. For example, to detect animal intrusion in smart agriculture when network bandwidth is being saturated, it would be preferable to receive raw motion sensor data and only pre-processed image data (instead of original images) to conserve bandwidth (See [9]). Finally, to prevent bandwidth saturation or focus on one part of physical environment that requires special attention, DTs may even require to change the spectrum (if feasible) over which specific data is transmitted. To support such unique requirements for DTs at the edge, there is a need for a system level support that provides *dynamic and fine-grained* control of communication to the DT infrastructure, so that it can dynamically control the type of data transmitted, rates at which it is transmitted and the spectrum over which it is transmitted. The key insight here is that the DT infrastructure can compute a highly accurate context of the IoT environment by analyzing the latest sensor data and other historical data. It is in the best position to determine the best way to manage communication using the current context and application semantics and requirements. Providing dynamic and fine-grained control of communication to the DT infrastructure will potentially lead to a best possible synchronization between physical environment and its DT, and enable a much faster and accurate actuation. To this end, we propose a *context aware communication control* (CACC)

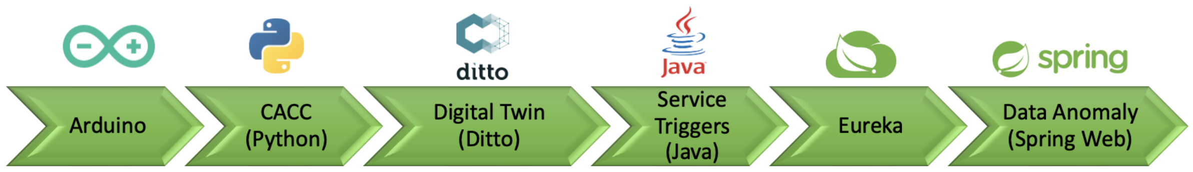

**Figure 4. Chain of Calls Representing the Implemented Prototype.**

component running at the edge server that can dynamically control data transmission software in the sensing devices in the physical environment to cope up with current context. CACC can provide a variety of features including (See Figure 3):

1. System support for both pull and push mechanisms, wherein based on the current context, CACC can dynamically mandate at runtime whether to pull data from PE or have PE push data either event-driven or at some regular intervals.
2. System support for CACC to be able to dynamically manage/alter the software controlling the data communication between physical and digital environments. In particular, provide system support for
   - uploading new data transmission code/code updates at physical environment;
   - tuning various data transmission parameters at physical environment such as transmission rates;
   - instantiating new code that is already present at physical environment; and
   - changing the spectrum (if feasible) over which the data is transmitted.
3. System support for differentiated data transmission mechanisms based on the current context, wherein CACC can dynamically control data transmission mechanism of individual sensors or a group of sensors to allow, for example, transmission mechanisms from one set of sensors to be different than from other set of sensors.

We note here that the aforementioned adaptation methods differentiate our DT framework from the many proposed frameworks in literature. Context-aware replication is necessary since replicating all data in an IoT-Edge-Cloud architecture is neither practical not feasible.

## EXPERIMENTS AND RESULTS

### Prototype Implementation

We implemented a proof of concept for the architecture representing the various components described in Section 2. Note that we have utilized some of the components from our DT-based solution we have proposed in our previous work [10] to detect data anomalies in IoT. The implemented prototype is a chain of calls for the different components as shown in Figure 4. In the beginning, an Arduino board contacts the CACC component running on the edge server. We choose to implement the connectivity between the IoT node and the edge server using WiFi. However, the CACC component, which is

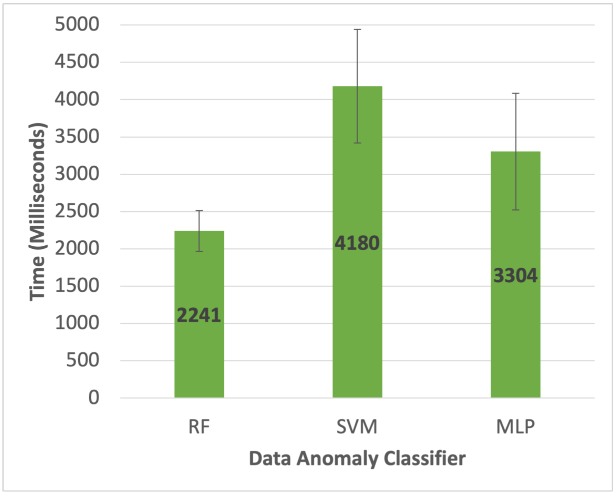

**Figure 5. Average Time Between Updating DT State and Performing Data Anomaly Detection on the Edge for Three Data Anomaly Classifiers**

implemented using Python, can switch to different connectivity measures as per the context as described in Section 3. Upon receiving the request to update the DT state from the Arduino board, the CACC component calls the RESTful APIs provided by Ditto to update the DT dynamic state data. Here, state data represents four sensors we have installed on the Arduino board to detect any data anomalies namely in temperature, humidity, loudness and light sensors. Now, since Ditto is configured to watch for any state changes for the DT by means of the DT identifier, Ditto SDK client implemented in Java calls Eureka server to check if any of the services corresponding to the DT (listed in the DT static metadata) are registered on the edge server. If the service is available, it is called to perform the needed classification.

In our implementation, we utilized the data anomaly service which we trained using three machine-learning classifiers (see [10] for details) namely Random Forests (RF), Support Vector Machines (SVM) and Multilayer Perceptron (MLP). We report in Figure 5 the time within the edge server measured from receiving a DT state update via Wi-Fi at the CACC until the data anomaly classification result is returned from the data anomaly service at the end of the chain. This time includes updating the DT state, triggering the service lookup, locating the data anomaly service by means of the Eureka server and finally calling the data anomaly service and receiving the classification results back. The total elapsed time is shown in the diagram for the three classifiers.

We can see from Figure 5 that the utilized machine-learning classifier has impact on the overall performance with RF producing the minimum overall time of 2.2 seconds and SVM producing the highest delay of 4.1 seconds. Note that this time includes the overhead from the classification operation as well as the overhead of the chain of calls between the different components. Also, note that MLP represented a middle ground between the two scenarios when it comes to time performance. Nonetheless, tolerating these delay numbers is application-dependent and thus must be carefully visited by the solution architects while designing the solution. In general, optimization of the architecture can happen by optimizing the calls between the components and the EdgeAI components performance which we will be exploring in the future.

## Full-Fledged Implementation Plans

The goal of the implemented prototype described in Section 4.1 was only to reflect on the feasibility to integrate the various open-source components needed to implement the architecture. As a future work, we plan to perform full implementation for the architecture to study the design choices for each of the components especially the CACC component and the service discovery component. First, for the CACC component, we plan to study mitigation scenarios that can be implemented when the architecture is confronted with large volumes of telemetry data or in case of extreme environment conditions requiring special handling. By means of simulation, we plan to subject the architecture to these special situations and apply solutions such as code uploads and/or parameter tuning depending on the context. Consequently, we can test the ability of such mitigation measures to ensure the architecture scalability under extreme conditions. We also plan to test the architecture with different IoT applications to gauge the benefit of the adaptation scenarios under different conditions. First, we plan to implement a smart agriculture IoT scenario. This scenario typically involves collection of scalar data from remote locations with intermittent connectivity. In addition, we plan to implement a smart city scenario representing the other extreme which requires dealing with multimodal sensor data (such as audio and video data) collected via Wi-Fi from buildings and public squares.

Second, for the service discovery component, we plan to integrate large number of Edge AI services pertaining to different IoT application such as smart city, IoMT, IIoT, smart building and smart agriculture with the architecture. Afterwards, we will gauge the complexity of developing various IoT applications while utilizing DTs in conjunction with the automatic service discovery provided by Eureka. In addition, we plan to simulate large workloads on these services to test the ability of the Eureka server to perform load balancing by switch between different microservices instances when confronted with high workloads. In addition, we plan to introduce multiple instances of the Eureka server itself so as to avoid bottleneck scenario for the discovery service.

## CONCLUDING REMARKS

This paper presented an edge-based architecture utilizing DTs to support IoT applications. The presented architecture relies on two novel components to address scalability and complexity issues in IoT. First, the Context-Aware Communication Component (CACC) responsible for scalability while integrating between the physical environment (PE) and the virtual environment (VE). Second, the service discovery component responsible for application-agnostic deployment of EdgeAI services. This component ensures reduced IoT application development complexity and introduces modularity and scalability when integrating with EdgeAI services. A prototype implementation using open source components for the architecture on an edge server shows clear benefits and great practicality. The presented architecture is currently undergoing full scale implementation to evaluate its ability to handle the challenges of complexity and providing scalability while serving various IoT applications's workloads.

https://www.overleaf.com/project/64cb88c8fc2519b619c0b2e7

## REFERENCES

[1] S. D. Okegbile, J. Cai, C. Yi, and D. Niyato, "Human digital twin for personalized healthcare: Vision, architecture and future directions," *IEEE network*, 2022.

[2] M. Jafari, A. Kavousi-Fard, T. Chen, and M. Karimi, "A review on digital twin technology in smart grid, transportation system and smart city: Challenges and future," *IEEE Access*, 2023.

[3] Y. Wang, Z. Su, S. Guo, M. Dai, T. H. Luan, and Y. Liu, "A survey on digital twins: architecture, enabling technologies, security and privacy, and future prospects," *IEEE Internet of Things Journal*, 2023.

[4] L. Raes, P. Michiels, T. Adolphi, C. Tampere, A. Dalianis, S. McAleer, and P. Kogut, "Duet: A framework for building interoperable and trusted digital twins of smart cities," *IEEE Internet Computing*, vol. 26, no. 3, pp. 43–50, 2021.

[5] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[6] E. Foundation, "Open source framework for digital twins., eclipse ditto," https://eclipse.org/ditto/.

[7] M. M. Rathore, S. A. Shah, D. Shukla, E. Bentafat, and S. Bakiras, "The role of ai, machine learning, and big data in digital twinning: A systematic literature review, challenges, and opportunities," *IEEE Access*, vol. 9, pp. 32 030–32 052, 2021.

[8] I. VMware, "Spring cloud netflix - eureka server," https://cloud.spring.io/spring-cloud-netflix/.

[9] J. Miao, D. Rajasekhar, S. Mishra, S. Nayak, and R. Yadav, "A fog-based smart agriculture system to detect animal intrusion," Tech. Rep., 2023, arxiv:2308.06614.

[10] K. Alanezi and S. Mishra, "An iot architecture leveraging digital twins: Compromised node detection scenario," https://tinyurl.com/5n9ahxhr.