Smart Application Development and Deployment over the IoT Computing Continuum

Carlos Kamienski[1]

[1]Federal University of ABC (UFABC)
Santo André – SP – Brazil

Abstract. *Smart applications increasingly rely on the Internet of Things (IoT) and the computing continuum to collect, process, and act on distributed data streams. However, current development and deployment practices remain fragmented, requiring manual configuration to map application functions to heterogeneous infrastructure, limiting portability and dynamic reconfiguration. We present DATUM, a framework for engineering distributed smart applications using a monolith-distributed programming approach. Developers write logic once, and DATUM compiles and distributes functions across the continuum. A drone-delivery service demonstrates how functions are deployed across sensors, edge, cloudlet, and cloud environments. Results show the potential for systematic, portable, and scalable deployment of IoT-continuum applications, supporting future automation and standardization.*

Resumo. *Aplicações inteligentes dependem cada vez mais da Internet das Coisas (IoT) e do contínuo computacional para coletar, processar e atuar sobre dados distribuídos. No entanto, práticas atuais de desenvolvimento e implantação são fragmentadas e exigem configuração manual para mapear funções da aplicação a camadas heterogêneas, limitando portabilidade e reconfiguração. Este artigo apresenta o DATUM, um framework para engenharia de aplicações distribuídas baseado em programação monolítica distribuída. A lógica é escrita uma vez, enquanto o DATUM compila e distribui funções ao longo do continuum. Um serviço de entrega por drones ilustra a distribuição entre sensores, névoa, cloudlet e nuvem. Os resultados demonstram potencial para implantação portátil e escalável de aplicações no continuum IoT, apoiando futuras iniciativas de automação e padronização.*

1. Introduction

Smart applications are emerging as key enablers of data-driven services across domains such as smart cities, precision agriculture, healthcare, and industry. These systems collect, process, and act on continuous streams of data generated in the physical environment, leveraging distributed computing resources spanning sensors, edge devices, cloudlets, and cloud platforms. This vision has given rise to the concept of the IoT computing continuum, a unified environment where computation and intelligence seamlessly transition between highly constrained devices and large-scale cloud infrastructure. Such environments must accommodate heterogeneous hardware, diverse communication paths, and strict requirements for latency, mobility, privacy, and resilience.

Despite advances in cloud-native technologies and edge-computing frameworks, engineering smart applications for the continuum remains intricate and labor-intensive.

Current practices often rely on manual decision-making to determine how software components should be distributed across different execution layers. Developers must navigate a fragmented ecosystem of tools, middleware, orchestration mechanisms, and deployment pipelines, each tailored to specific environments. As a result, application portability, systematic deployment, and automatic reconfiguration across heterogeneous IoT infrastructures remain limited. This gap between conceptual continuum architectures and practical deployments restricts scalability, maintainability, and the widespread adoption of distributed smart services.

To address these limitations, this paper introduces DATUM, a framework that provides an integrated model for the development, compilation, deployment, and execution of distributed smart applications. DATUM adopts a monolith-distributed programming paradigm, in which application logic is authored as a single program and subsequently compiled into distributed execution units. These units can then be placed across continuum stages according to functional roles, resource constraints, and runtime needs. The framework formalizes abstractions such as the D-Script, D-Graph, D-Node, and D-Continuum, establishing a structured foundation for building and evolving smart applications independently of the underlying infrastructure. By separating application design from deployment decisions, DATUM promotes transparency, portability, and adaptability across heterogeneous IoT environments.

To illustrate the applicability of the approach, a smart drone-delivery service is modeled as a representative use case. This scenario demonstrates how application functions related to navigation, sensing, communication, coordination, and decision-making can be systematically allocated across sensors, edge nodes, cloudlets, and cloud resources. The example highlights the benefits of unified development and controlled distribution in supporting complex, latency-sensitive, and collaborative IoT services.

The main contributions of this work are:

- A unified architecture and lifecycle for developing, compiling, deploying, and executing smart applications over the IoT computing continuum.
- A monolith-distributed programming model, enabling single-source development with transparent function distribution.
- Formal abstractions for representing continuum resources, execution nodes, and distributed application graphs.
- A modeled use case demonstrating practical placement of distributed services across sensors, edge, cloudlet, and cloud layers.
- A foundation for automation and standardization, promoting systematic, portable, and scalable smart-application deployment.

2. Related Work

Recently, the existence of a processing, storage, and communication continuum between sensors and the cloud, comprising a diverse set of infrastructure elements, has become clear [Moreschini et al. 2022]. The IoTinuum (IoT Computing Continuum) is a framework for understanding the choices and tradeoffs in developing and deploying smart applications [Kamienski et al. 2024]. It covers the end-to-end path of data acquisition, processing, storage, and transmission, from devices through intermediate stages to the user interface. The IoTinuum model comprises six stages that span from the physical world

to the user interface. At the foundation, S1-Thing includes devices, sensors, and actuators that sense the environment and actuate changes, converting analog signals into digital actions. Closest to them is S2-Mist, a system of low-power, field-deployed single-board computers that act as radio gateways and often rely on energy harvesting. Further upstream, S3-Fog consists of more capable edge nodes—such as tower servers, equipment in farm offices, or telecom cabinets—located tens of meters to kilometers away, providing stable power and sheltered deployment. The S4-Cloudlet stage hosts micro-datacenters positioned at telecom access points or industrial facilities to offer localized compute near the core. At its core, S5-Cloud comprises large-scale public or private data centers with vast computing and storage resources. Finally, S6-App represents the user endpoint where smart applications run, and users interact with the system.

The Internet of Things (IoT) introduces additional complexity to software development due to its inherent distribution and the inclusion of a massive number of heterogeneous devices (sensors and actuators) in its functionality and hardware architecture [Borelli et al. 2020b]. Developing software architectures for IoT thus involves interacting with various software components that play distinct roles. Although there are already some initiatives to create these architectures, they still require widespread acceptance within the software developer community [Zyrianoff et al. 2020]. Specific architectural patterns are needed to develop an IoT smart application, which are classified into seven categories [Borelli et al. 2020a]: data endpoint, data ingestion, data interaction, data integration, data storage, data processing, data visualization, and data security.

The deployment of smart applications for IoTinuum requires the efficient orchestration of resources allocated to services to achieve the intended quality levels [Oliveira et al. 2024]. Orchestration involves fulfilling the application needs by carefully matching services, resources, and workloads, including microservice placement, deployment, and migration.

IoTDeploy [Oliveira et al. 2024] is a strategy for static and dynamic service migration across IoTinuum, along with a set of derived tools to implement it. An IoT smart application comprises various services (i.e., microservices) that may be deployed at different stages for distinct installations. Using a particular node within the continuum stage involves deploying services. If this stage is not used in a given installation, this service must be deployed elsewhere in another stage. IoTDeploy extends the traditional continuous integration and continuous delivery (CI/CD) approach by implementing a pipeline for deploying different configurations of the same application across the continuum.

3. Challenges in Application Development and Deployment

Application development for the IoTinuum presents several challenges. First, the distributed nature of the IoT ecosystem, the massive number of heterogeneous connected devices, and their mobility patterns increase development complexity. Second, while a microservice architecture offers advantages over monolithic designs, developers must still define the number, roles, and boundaries of microservices and manage their communication. An increasing number of microservices leads to more complex orchestration. Third, microservices must run across highly diverse environments, from constrained devices to powerful cloud servers. Although containers (e.g., Docker) facilitate portability, they remain limited on specific resource-constrained devices.

Application deployment across the IoTinuum introduces additional challenges. Applications composed of multiple microservices must span infrastructure from S1-Thing to S5-Cloud, requiring efficient resource orchestration to meet quality-of-service expectations. Deployment configurations may vary depending on the available infrastructure (e.g., with or without fog nodes). Furthermore, deployments can be static (where services are placed at specific stages per installation) or dynamic (where services migrate between stages, e.g., from S3-Fog to S5-Cloud) based on runtime conditions, while preserving data integrity.

These challenges are tightly interconnected: development influences deployment, as complex microservice-based designs demand sophisticated static and dynamic deployment strategies; conversely, deployment constrains development, since the need to operate across the IoTinuum drives finer-grained service decomposition. Thus, a unified approach that jointly addresses development and deployment across the IoTinuum becomes essential.

These challenges are tightly interdependent: development shapes deployment, as building applications composed of multiple microservices demands sophisticated static and dynamic deployment strategies; conversely, deployment shapes development, since the need to operate across the diverse stages of the IoTinuum pushes developers toward increasingly complex microservice-based designs. Consequently, a unified solution that integrates development and deployment across the IoTinuum becomes essential.

4. Distributed Applications for the IoT Computing Continuum

$DATUM$ (Distributed Applications for the IoTinuum) [Kamienski et al. 2025] is a framework for the integrated development and deployment of distributed smart applications for the IoTinuum. The $DATUM$ Architecture ($D\text{-}Architecture$) introduces a structured view of $DATUM$ considering the challenges in the smart application development and deployment. $D\text{-}Architecture$ is contextualized by its environment, called $D\text{-}Environment$.

4.1. DATUM Architecture and Lifecycle

The big picture of $D\text{-}Architecture$ is represented by $D\text{-}Environment$ (Figure 1), a computing environment where a $DATUM$-enabled application runs in the dataplane and an external controller. $D\text{-}Environment$ is centered around the Life Cycle, the Applications, the Platform, the Continuum, and the Controller.

$D\text{-}LifeCycle$ represents the $D\text{-}Architecture$ development and deployment life cycle. Figure 2 depicts $D\text{-}LifeCycle$ where rectangles represent processes ($D\text{-}Process$) and ellipses represent artifacts ($D\text{-}Artifact$). $D\text{-}LifeCycle$ comprises three processes (or phases): $D\text{-}Develop$: the $DATUM$ Development Process; $D\text{-}Deploy$: the $DATUM$ Deployment Process; $D\text{-}Execute$ and $D\text{-}Monitor$: the $DATUM$ Execution and Monitoring Processes.

4.2. D-Application: DATUM Smart Applications

A Smart Application is a runtime instance of all software components deployed within a distributed computing infrastructure (hardware and communication), addressing a specific area or vertical, such as cities, agriculture, healthcare, or industry. Using various

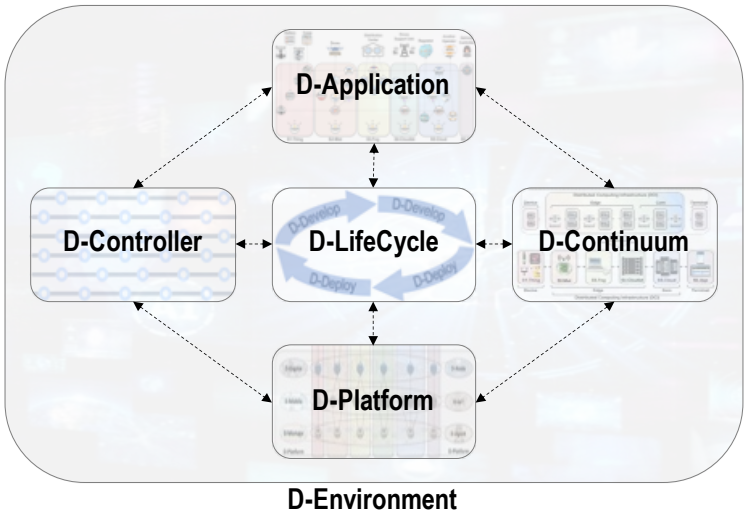


Figure 1. DATUM Environment for Smart Applications

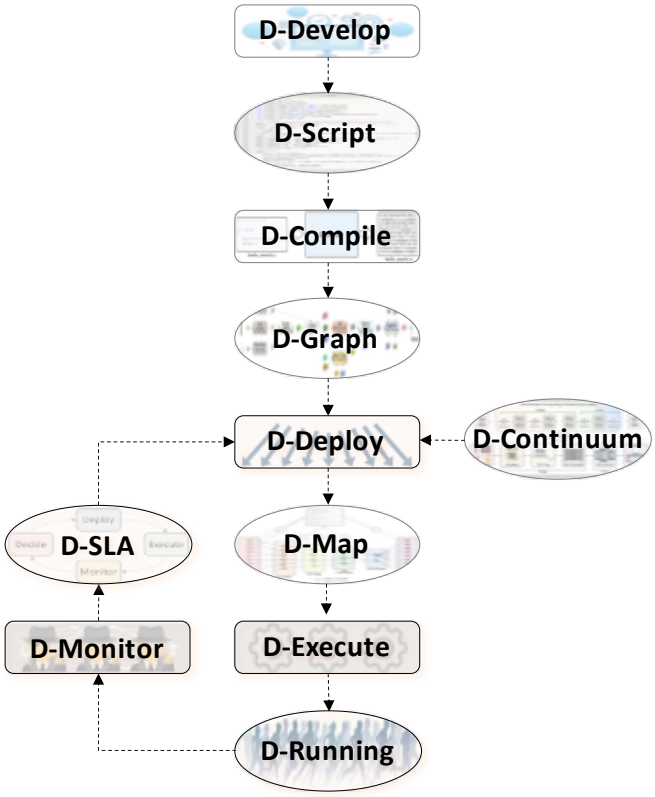


Figure 2. DATUM Application Development, Deployment, Execution, and Monitoring Life Cycle

technologies combined, such as IoT, cloud computing, AI/ML, and network softwarization, enables the development of a myriad of new smart applications.

$D\text{-}Application$ is the application logic for $D\text{-}Architecture$ comprising all pieces of code needed to implement the required functionality for a smart application. Like a datum is a single piece of data, the abstract $DATUM$ code is a single, logically centralized

entity that can be deployed and physically distributed across various nodes at different stages of the $D\text{-}Continuum$. These codes enable the development and deployment of diverse Smart Applications (e.g., smart cities or smart farming) with varying requirements (e.g., latency or privacy) and use cases spanning different categories (e.g., federated learning or autonomous operation).

The key feature of $D\text{-}Application$ is its centralized nature: it is developed as a monolith composed of a set of functions distributed across the continuum via static (at compilation time) or dynamic (at deployment time) processes. In principle, any $DATUM$ piece of code can be deployed over any $D\text{-}Continuum$ stage unless it plays a role that depends on specific hardware or location (e.g., reading sensor data).

The $D\text{-}Application$ centralized code is called $D\text{-}Script$ which is a set of functions called $D\text{-}Function$. The management of a $D\text{-}Application$ belongs to the $DATUM$ Life Cycle ($D\text{-}LifeCycle$) and involves the processes for application development ($D\text{-}Develop$) and deployment ($D\text{-}Deploy$).

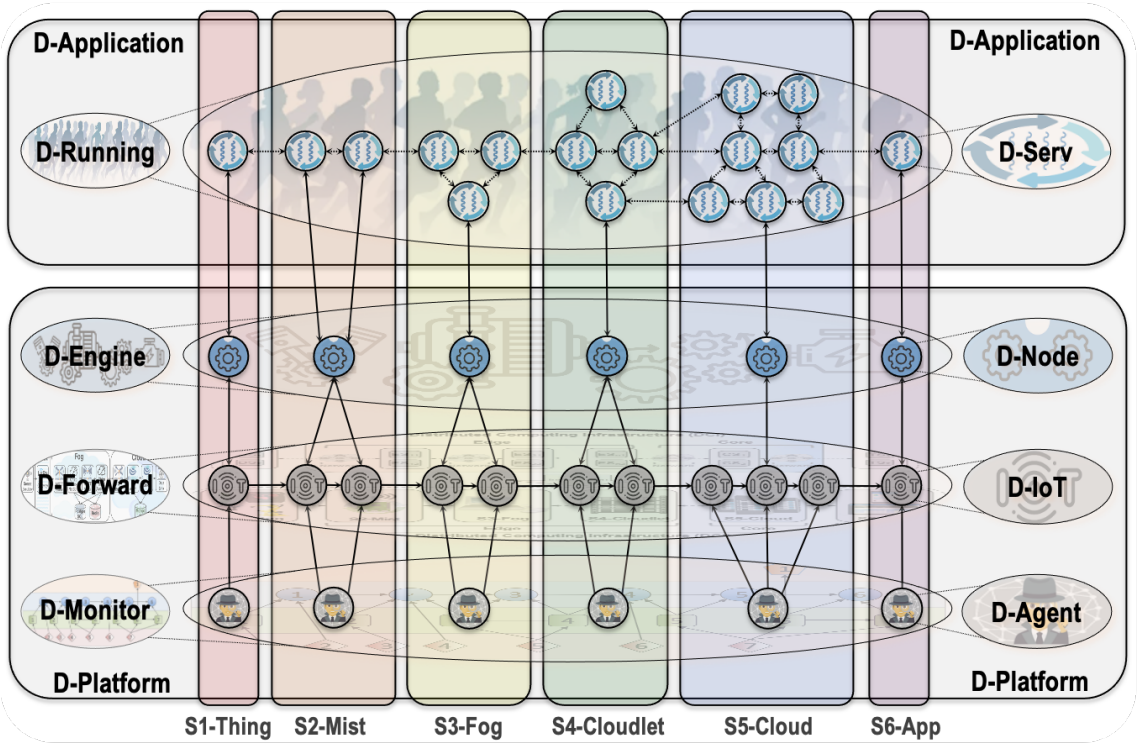


Figure 3. The DATUM Architecture

4.2.1. D-Platform: The DATUM Platform

$D\text{-}Platform$ is the $DATUM$ underlying platform providing the environment that executes a $D\text{-}Application$. $D\text{-}Platform$ is composed by three sub-platforms, $D\text{-}Engine$, $D\text{-}Forward$, and $D\text{-}Monitor$. $D\text{-}Engine$ is the $DATUM$ distributed Engine that runs $D\text{-}Application$ over $D\text{-}Continuum$, comprised of a set of $D\text{-}Node$. A $D\text{-}Node$ has multiple functions in $D\text{-}Architecture$, particularly running $D\text{-}Graph$ composed of a set of $D\text{-}Serv$. The implementation of $D\text{-}Node$ varies depending on the language used for

$D\text{-}Serv$. For example, in case WebAssembly (WASM) [Rossberg 2024] is selected as the $D\text{-}Serv$ code (called $D\text{-}Code$), a $D\text{-}Node$ may be a WASM runtime, e.g., WasmTime [Alliance 2024], augmented with communication and management capabilities.

$D\text{-}Forward$ is an IoT compound platform (or middleware) that forwards data from S1-Thing to S5-Cloud and back, composed of a set of $D\text{-}IoT$ running on different $D\text{-}Continuum$ stages. The function, placement, and chaining of different $D\text{-}IoT$ over $D\text{-}Continuum$ make the end-to-end data path that interconnects the set of $D\text{-}Serv$ belonging to a $D\text{-}Graph$. The multiple $D\text{-}IoT$ composing $D\text{-}Forward$ may have different natures, such as brokers (e.g., MQTT or Zenoh) or LoRaWAN servers, which may or may not be connected to IoT Platforms, such as FIWARE or ThingsBoard.

4.2.2. D-Controller: The $DATUM$ Controller

$D\text{-}Controller$ is a control plane entity that manages all $DATUM$ centralized control-plane processes, mainly $D\text{-}Deploy$ described further in Section 6. The role of $D\text{-}Controller$ is similar to that of an SDN controller, which, by the way, is also used in IoT and fog/edge computing to address IoT's main challenges. Similarly to SDN, $D\text{-}Controller$ is logically centralized but can be physically distributed.

5. D-Develop: the DATUM Development Process

$D\text{-}Develop$ is the process of $D\text{-}Application$ software development that generates a $D\text{-}Script$ as its main outcome. IoT Smart Application developers may rely on a variety of methods and tools to perform the $D\text{-}Develop$ process, which is orthogonal to $D\text{-}Architecture$. Therefore, we consider these methods to be related but outside the scope of $DATUM$.

5.1. D-Script: the DATUM Script

$D\text{-}Script$ is a monolith-distributed-style development for $D\text{-}Continuum$that generates serverless services as an output [De Palma et al. 2023]. The application development process is similar to a monolith, with FaaS (Function as a Service) functions connected. $D\text{-}Script$ comprises a set of $D\text{-}Function$ that run as distributed pieces of code. $D\text{-}Function$ may have annotations representing non-functional requirements, such as SLA/QoS and security/privacy.

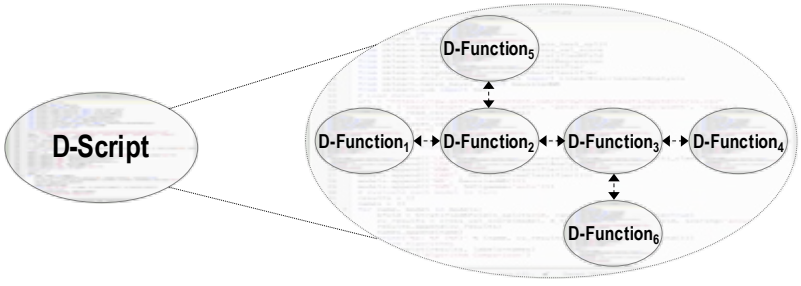


Figure 4. D-Script and D-Function

5.2. D-Graph: The DATUM Graph

A $D\text{-}Graph$ is an application graph comprising a set of communicating $D\text{-}Serv$. Formally, a $D\text{-}Graph$ is defined as $G = (V, E)$ where V is a set of $D\text{-}Serv = (D\text{-}Serv_1, \ldots, D\text{-}Serv_s)$, $s = |D\text{-}Serv|$ and E is a set of service invocations (aka $D\text{-}Call$) between pairs of $D\text{-}Serv$ in the form $D\text{-}Call_{ij} = D\text{-}Serv_i \rightarrow D\text{-}Serv_j$, $i, j = 1..s$. A $D\text{-}Serv$ is a serverless service that runs on different $D\text{-}Continuum$ stages, respecting its role. For example, a $D\text{-}Serv$ that collects sensor data is intended to run in S1-Thing. Also, different granularities of $D\text{-}Serv$ may compose a $D\text{-}Application$ such as:

- Microservices: a traditional view of the microservice architecture, but with FaaS.
- Nanoservices: fine-grained microservices specialized for a single type of hardware resource [Wang et al. 2021]
- Picoservices: tiny services that run in severely resource-constrained devices.
- Netservices: In-network services that run in switches and network interfaces via data plane programming languages, such as P4, and collaborate with distributed application processing in the IoTinuum.

$D\text{-}Code$ is the $D\text{-}Serv$ executable code in an architecture-independent language that any $D\text{-}Node$ can run in a $DATUM$ Application. A strong candidate is WebAssembly (WASM) [Rossberg 2024], initially developed for web browsers, but increasingly proposed for applications running in the continuum [Kakati and Brorsson 2023, Zhang et al. 2024].

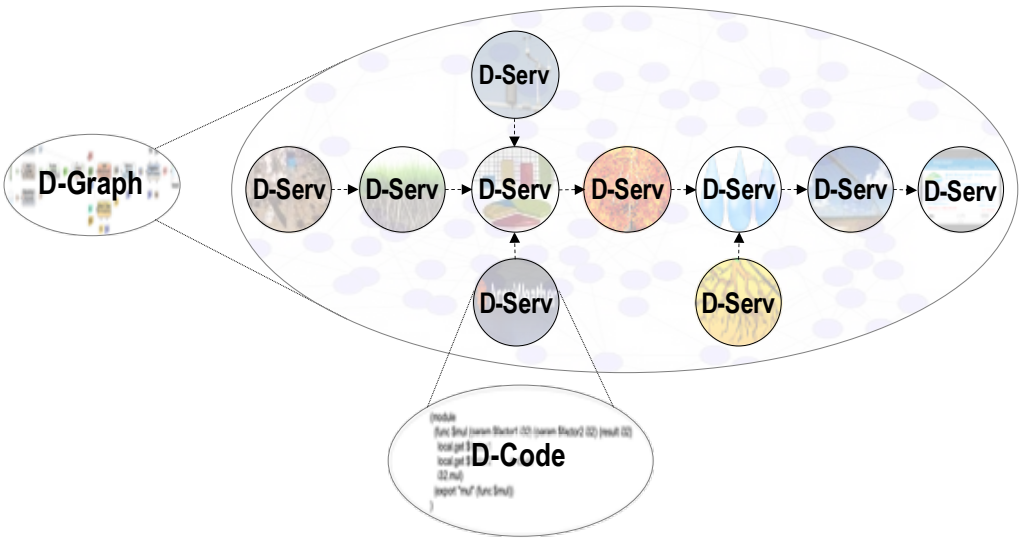


Figure 5. D-Graph, D-Serv and D-Code

5.3. D-Compile: The DATUM Compiler

$D\text{-}Compile$ compiles a $D\text{-}Script$, which contains a set of $D\text{-}Function$, into a $D\text{-}Graph$, which contains a set of $D\text{-}Serv$. In addition to the straightforward task of generating $D\text{-}Code$ from $D\text{-}Function$ $D\text{-}Compile$ also maps $D\text{-}Function$ to $D\text{-}Serv$ according to different styles, requirements, preferences, and constraints. Figure 6 depicts different $D\text{-}Function$ to $D\text{-}Serv$ mapping styles, which may be as straightforward as $1:1$, but

one $D\text{-}Function$ may be split into N $D\text{-}Serv$ $(1:N)$, several M $D\text{-}Function$ may be grouped to form a single $D\text{-}Serv$ $(M:1)$, or even a group of M $D\text{-}Function$ may be mapped to a different group of N $D\text{-}Serv$ $(M:N)$.

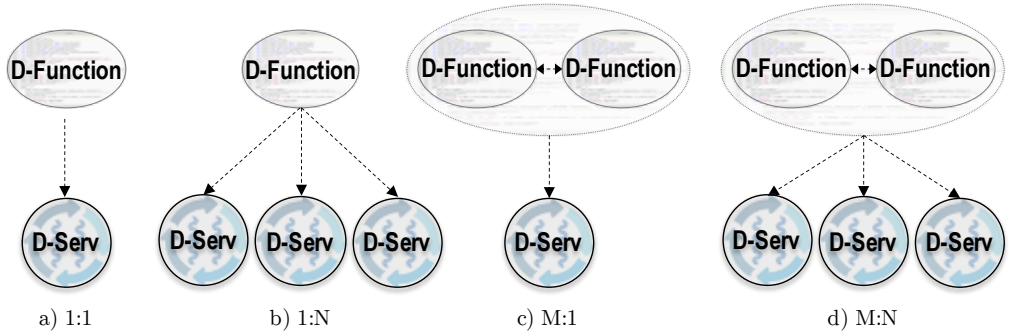


Figure 6. Mappings from D-Function to D-Serv

6. D-Deploy: The DATUM Deployment Process

$D\text{-}Deploy$ involves the process of $D\text{-}Application$ deployment and dynamic redeployment over $D\text{-}Continuum$.

6.1. D-Continuum: The $DATUM$ Continuum

$D\text{-}Continuum$ is the specification of the supported IoTinnum configuration for a specific $D\text{-}Application$, comprising stages, substages, and elements, together with their characteristics, constraints, and requirements.

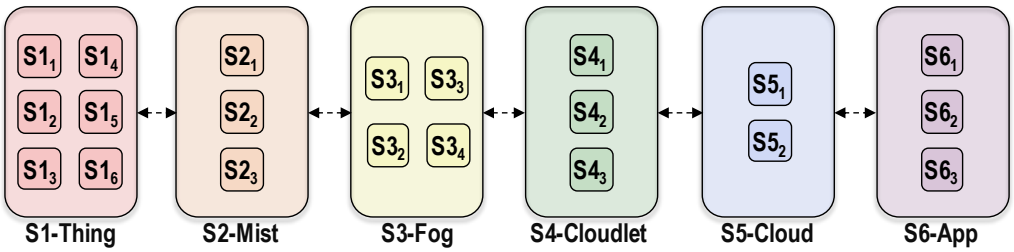


Figure 7. $D\text{-}Continuum$**:** $D\text{-}Stage$ **and** $D\text{-}Substage$

- $D\text{-}Stage$: Represent different locations, distance to devices, and number of elements.
- $D\text{-}Substage$: Represent categories of computing equipment for the same $D\text{-}Stage$, for example: a) different devices implementing S1-Thing, such as Arduino and ESP32, or b) different geographical coverage, such as areas of a city or fields of a farm.
- $D\text{-}Element$: any instance of physical equipment or device that runs the $DATUM$ Architecture in a $D\text{-}Stage$. For example, if a $D\text{-}Application$ uses 100 ESP32 for S1-Thing, each device is a $D\text{-}Element$.

As an example, Figure 8 depicts an S1-Thing D-Stage with two D-Substage Arduino and ESP32, each with four D-Elements.

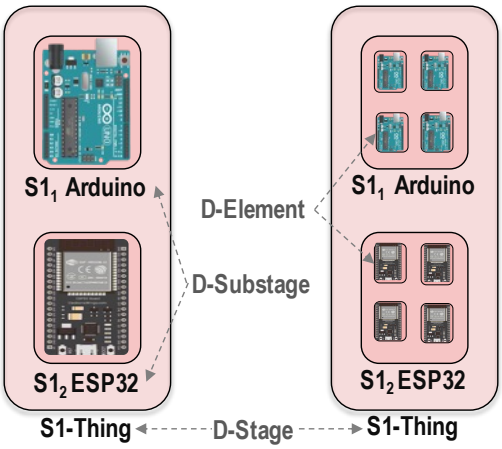


Figure 8. $D\text{-}Continuum$**: example of** $D\text{-}Stage$**,** $D\text{-}Substage$**, and** $D\text{-}Element$

6.2. D-Node vs. D-Serv

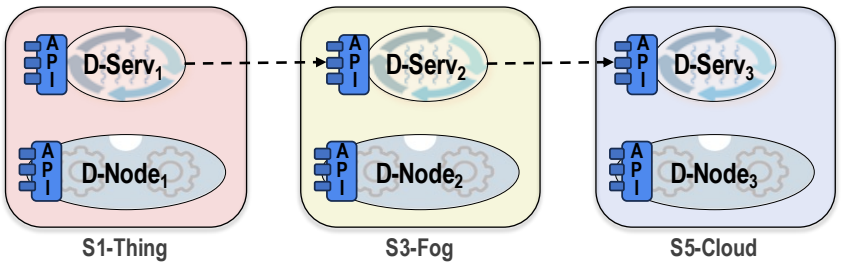
Each $D\text{-}Node$ is associated with a specific $D\text{-}Continuum$ $D\text{-}Stage$ or $D\text{-}Substage$. Since $D\text{-}Continuum$ stages are not equal concerning their resources, $D\text{-}Node$ must comply with the underlying hardware (e.g., a $D\text{-}Node$ running on S1-Thing differs from another $D\text{-}Node$ running on S5-Cloud).

Figure 9 depicts the three ways a group of $D\text{-}Serv$ can communicate with each other. Figure 9a) shows a scenario in which all $D\text{-}Serv$ have APIs that others can access, so communication is always direct between them. Figure 9b) shows the scenario in which $D\text{-}Serv$ lacks APIs, so all communication must go through the $D\text{-}Node$. Also, Figure 9c) depicts a hybrid situation in which some $D\text{-}Serv$ have APIs, while others must rely on their $D\text{-}Node$.

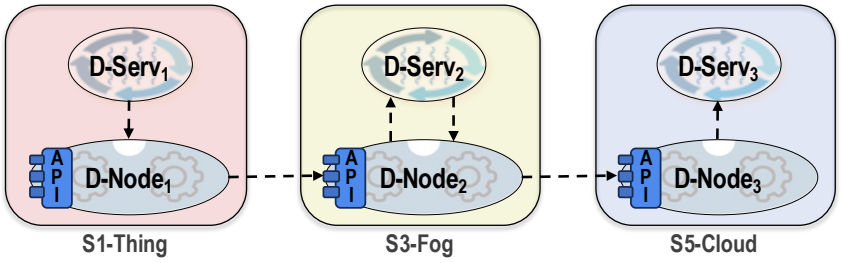
7. Use Case: Drone Delivery Service

A smart drone delivery application utilizes Unmanned Aerial Vehicles (UAVs), also known as drones, to deliver packages in urban areas, replacing traditional terrestrial vehicles [de Oliveira et al. 2023]. Fig. 10 illustrates a drone delivery service utilizing a 6-stage IoTinuum, where the distributed computing stages collaborate to facilitate efficient and safe flights. Drones are modeled at the edge as mobile S2-Mist components that aggregate built-in sensors in S1-Thing (inside the drone), such as a camera, GPS, and Light Detection and Ranging (Lidar) for detecting other drones. Drones can communicate with the distribution center control (S3-Fog) using Wi-Fi for takeoff and landing sequencing [Soares et al. 2023, Soares et al. 2025]. Also, they can communicate with drone support units (DSU) provided by the 5G (or even 6G in the future) operator in the S4-Cloudlet. In this use case, the S4-Cloudlet is crucial for enabling drones to avoid collisions and share relevant information with each other. Distribution centers (S3-Fog) can also communicate with DSUs (S4-Cloudlet) to coordinate the delivery service. End-users and management personnel access the service via the S6-App. S3-Fog and S4-Cloudlet communicate with S5-Cloud to collaborate on various activities, including collision avoidance.

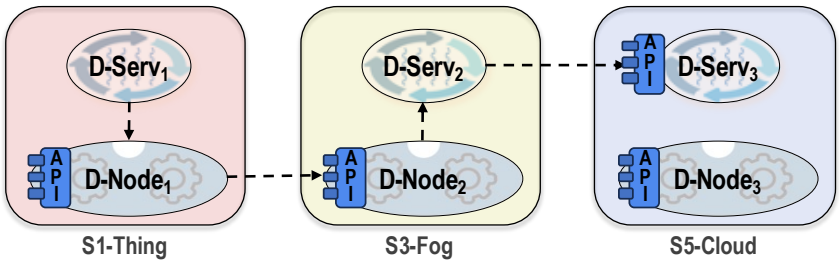
We model the issue of collision avoidance in a smart drone delivery service to exemplify the use of the IoTinuum and $D\text{-}Architecture$can be a valuable tool for understanding the challenges of smart applications for urban computing. As this scenario



a) Direct communication between D-Serv via D-Serv^{API}



b) Indirect communication between D-Serv via D-Node^{API}



c) Hybrid Direct/Indirect communication between D-Serv via D-Serv^{API} andD-Node^{API}

Figure 9. D-Serv Direct and Indirect Communication

represents a future service that is not yet implemented, only the set of $D\text{-}Serv$ belonging to $D\text{-}Application$ is modeled. Fig. 11 expands the basic modeling of Fig. 10 with examples of $D\text{-}Serv$ needed in each stage for delivery management and collision avoidance activities. Services needed in all stages are:

- S1-Thing: control services for sensors and actuators: Engine control, Cargo control, Lidar collector, Battery collector;
- S2-Mist: services running in the drone's single-board computer that control the flight, operate the delivery, and perform collision avoidance: Flight Control, Delivery Operation, Battery Control, Takeoff and Landing in Distribution Center, Detour, Collaborative Detour, Collision Avoidance;
- S3-Fog: Manages the operations of the distribution center, i.e., delivery control and takeoff and landing sequencing: Delivery Schedule, Takeoff and Landing Management, Landing Operation, Takeoff operation;
- S4-Cloudlet: Processes functions to help drones engage in collaborative collision avoidance: Collaboration Management, Swarm, Consensus, Collision Avoidance Management;
- S5-Cloud: Contains functions for delivery and collision avoidance manage-

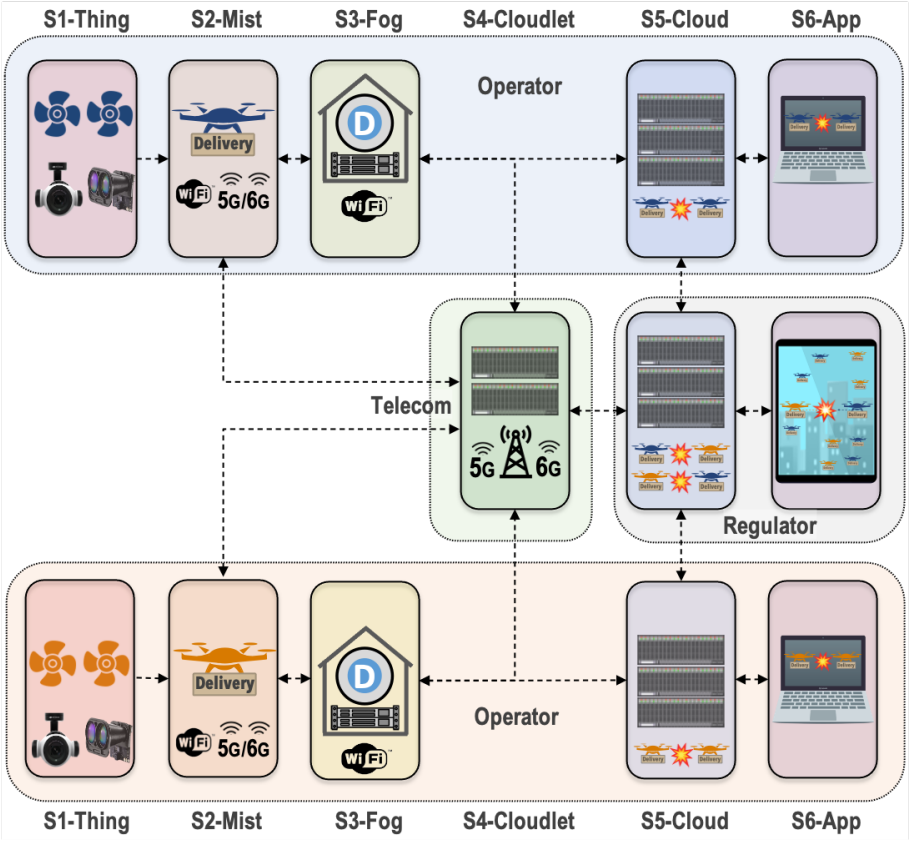


Figure 10. IoTinuum for Smart Drone Delivery

ment: Delivery Management, Collaboration Management, Path Planning, Collision Avoidance Management, Airspace Policy, Logistic Operator;

- S6-App: presents a graphical interface where a Delivery Controller supervises the delivery management activities and interacts with Delivery Mgmt in S5-Cloud.

8. Considerations for Internet Standardization

The evolution of the IoT computing continuum and the DATUM framework demand architectural and operational guidance to support seamless deployment, execution, and adaptation of distributed smart applications. While the current Internet architecture, protocols, and service orchestration mechanisms provide many foundational capabilities, there is still limited standardization addressing how computation, data, and control should be consistently distributed from constrained devices to cloud and application layers. Inspired by the architectural direction outlined in RFC 9556 [Hong et al. 2024], which frames the computing continuum as an integral part of the Internet model, this work proposes to advance a complementary view focused specifically on smart-application deployment and execution across heterogeneous IoT environments.

To support this vision, an Internet-Draft is being prepared to formalize conceptual models, terminology, and architectural components for function distribution, deployment, and execution across the IoT continuum. This draft aims to establish common principles for service decomposition, placement, portability, and lifecycle control that reflect operational realities encountered in multi-layer IoT environments. The intent is not to prescribe

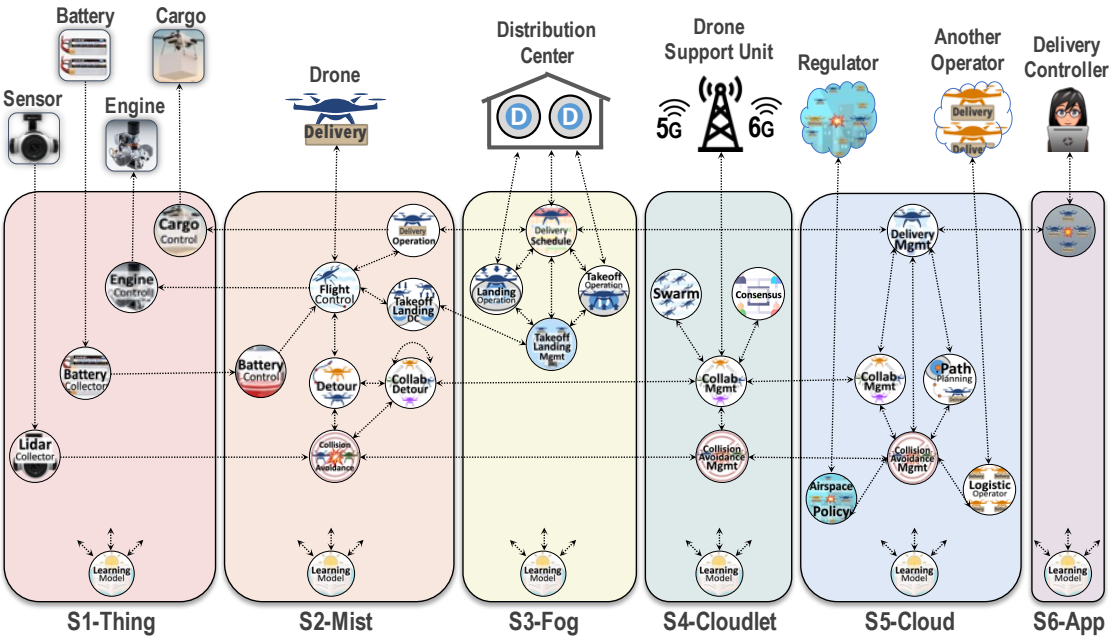


Figure 11. DATUM for Smart Drone Delivery

specific implementation mechanisms or technologies, but rather to provide an architectural foundation and vocabulary upon which interoperable mechanisms, open interfaces, and future protocols may be built.

By defining a reference model and clarifying deployment-oriented roles across the continuum, this effort contributes to the broader goal of ensuring that IoT systems evolve in a manner consistent with Internet architectural principles, promoting interoperability, openness, and innovation while supporting a scalable, programmable application deployment ecosystem.

9. Conclusion

This paper introduces DATUM, a framework that enables unified development, compilation, and deployment of smart applications across the IoT computing continuum through a monolith-distributed programming model and structured abstractions for application logic and infrastructure. A drone-delivery scenario illustrated how DATUM distributes functions across sensors, edge, cloudlets, and the cloud, demonstrating its potential to support systematic, scalable continuum-based deployments. Future work includes implementing and validating the framework in real environments, incorporating automated function placement and orchestration, and advancing toward reference models and standards for IoT continuum application deployment.

References

Alliance, B. (2024). Wasmtime: A fast and secure runtime for webassembly. Accessed on June 17, 2024.

Borelli, F., Biondi, G., Horita, F., and Kamienski, C. (2020a). Architectural software patterns for the development of iot smart applications. *arXiv preprint arXiv:2003.04781*.

Borelli, F. F., Biondi, G. O., and Kamienski, C. A. (2020b). Biota: A buildout iot application language. *IEEE Access*, 8:126443–126459.

de Oliveira, F. M., Bittencourt, L. F., Bianchi, R. A., and Kamienski, C. A. (2023). Drones in the big city: Autonomous collision avoidance for aerial delivery services. *IEEE Transactions on Intelligent Transportation Systems*.

De Palma, G., Giallorenzo, S., Trentin, M., and Vjerdha, G. (2023). A framework for bridging the gap between monolithic and serverless programming. In *Microservices 2023: 5th International Conference on Microservices*, pages 1–7. Microservices Community.

Hong, J., Hong, Y., de Foy, X., Kovatsch, M., Schooler, E., and Kutscher, D. (2024). Internet of things (iot) edge challenges and functions: Rfc 9556. *Internet Research Task Force (IRTF)*.

Kakati, S. and Brorsson, M. (2023). Webassembly beyond the web: A review for the edge-cloud continuum. In *2023 3rd International Conference on Intelligent Technologies (CONIT)*, pages 1–8. IEEE.

Kamienski, C., Zyrianoff, I., Bittencourt, L., and Di Felice, M. (2024). Iotinuum: The iot computing continuum. In *21st International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT 2024)*, pages 1–6.

Kamienski, C. A., Zyrianoff, I., Xue, L., and Felice, M. D. (2025). Distributed smart agriculture monitoring over the IoT computing continuum. In *2025 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*, Italy.

Moreschini, S., Pecorelli, F., Li, X., Naz, S., Hästbacka, D., and Taibi, D. (2022). Cloud continuum: the definition. *IEEE Access*, 10:131876–131886.

Oliveira, F. B., Di Felice, M., and Kamienski, C. (2024). Iotdeploy: Deployment of iot smart applications over the computing continuum. *Internet of Things*, 28:101348.

Rossberg, A. (2024). Webassembly specification - release 2.0 (draft 2024-01-17).

Soares, L. D. P., De Oliveira, F. M., Kamienski, C. A., and Bittencourt, L. F. (2023). Drone edge management system (drems): Sequencing drone takeoff and landing. In *2023 10th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 114–121. IEEE.

Soares, L. D. P., De Oliveira, F. M. C., Kamienski, C. A., and Bittencourt, L. F. (2025). Edge4drone: Managing landings and takeoffs in high-density distribution centers. In *IEEE INFOCOM 2025-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE.

Wang, X., Yuan, T., Huang, Y.-J., and van Renesse, R. (2021). Disaggregated applications using nanoservices. In *Second Workshop On Resource Disaggregation and Serverless (WORDS 2021)*. ACM.

Zhang, Y., Liu, M., Wang, H., Ma, Y., Huang, G., and Liu, X. (2024). Research on webassembly runtimes: A survey. *arXiv preprint arXiv:2404.12621*.

Zyrianoff, I., Heideker, A., Silva, D., Kleinschmidt, J., Soininen, J.-P., Salmon Cinotti, T., and Kamienski, C. (2020). Architecting and deploying iot smart applications: A performance–oriented approach. *Sensors*, 20(1):84.