MIDDLEWARE 101

WHAT TO KNOW NOW AND FOR THE FUTURE

ALEXANDROS GAZIS AND ELEFTHERIA KATSIRI

In computer science, systems are typically divided into two categories: software and hardware. However, there is an additional layer in between, referred to as *middleware*, which is a software "pipeline," an operation, a process, or an application between the operating system and the end user. This article aims to define middleware and reflect on its necessity, as well as address controversies about when and where it applies. It also explores the application of middleware in emerging technologies such as cloud computing and the IoT (Internet of Things), as well as future middleware developments.

The term was introduced in the early 1980s. It encompasses complex software solutions that modernize legacy systems—typically mainframes—through new features such as software and application components. Initially, it was solely used to expand the layer separating the network and application layers. Subsequently, its use expanded to serve as the layer above the operating system and network layer, and below the application layer. This means that middleware could now facilitate the generic communication between the application component and the distributed network.

Through middleware, a programmer has the option to implement a decentralized solution instead of having to interact and analyze different components.[18]

In recent literature[3,12,14,16] multiple definitions have been used, depending on the field of research. On the one hand, both a software and a DevOps engineer would describe middleware as the layer that "glues" together software by different system components; on the other hand, a network engineer would state that middleware is the fault-tolerant and error-checking integration of network connections. In other words, they would define middleware as communication management software. A data engineer, meanwhile, would view middleware as the technology responsible for coordinating, triggering, and orchestrating actions to process and publish data from various sources, harnessing big data and the IoT (Internet of Things). Given that there is no uniform definition of middleware, it is best to adopt a field-specific approach.

The main categories of middleware are as follows:[11]

➡ Transactional. Processing of multiple synchronous/asynchronous transactions, serving as a cluster of associated requests from distributed systems such as bank transactions or credit card payments.

➡ Message-oriented. *Message queue* and *message passing* architectures, which support synchronous/asynchronous communication. The first operates based on the principle that a queue is used to process information, whereas the second typically operates on a publish/subscribe pattern where an intermediate broker facilitates the communication.

➡ Procedural. *Remote* and *local* architectures to connect, pass, and retrieve software responses of asynchronous communications such as a call operation. Specifically, the first architecture calls a predetermined service of another computer in a network, while the second interacts solely with a local software component.

➡ **Object-oriented.** Similar to procedural middleware, however, this type of middleware incorporates object-oriented programming design principles. Analytically, its software component encompasses object references, exceptions, and inheritance of properties via distributed object requests. It is typically used synchronously, because it needs to receive a response from a server object to address a client action. Importantly, this type of middleware can also support asynchronous communication via the use of (multi) threads and generally concurrent programming.

Academics have further segregated middleware depending on the application module it serves, such as database, web server, etc. There are several types of middleware, falling into these key categories: reflective, agent, database, embedded, portal, and device (or robotics).[4,15]

First, *reflective* middleware constitutes components that are specifically designed to "easily operate with other components and applications," while *agent* middleware has multiple components that operate on complex domain-specific languages and laws.

Second, *database* middleware focuses on DB-to-DB or DB-to-apps communication—either natively or via CLIs (call-level interfaces)—while *embedded* middleware acts as the intermediary for embedded integration apps and operating-system communication.

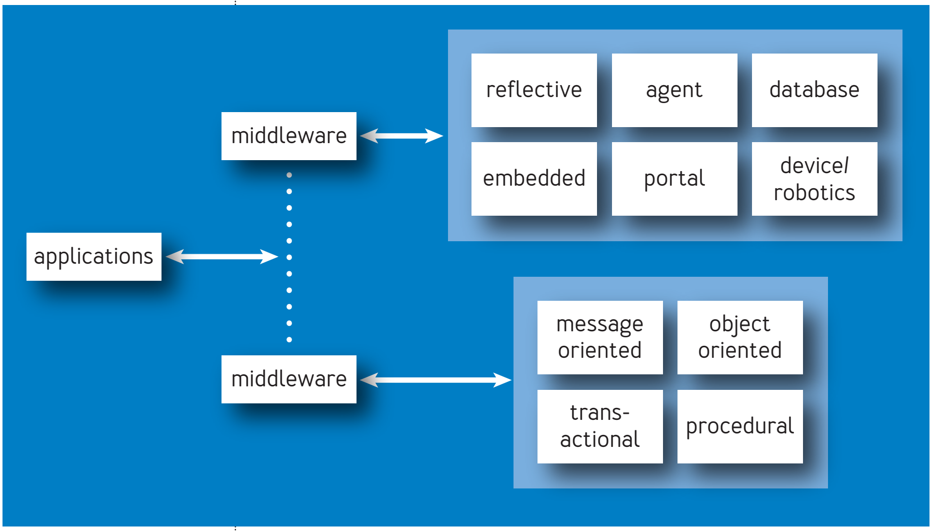
Third, *portal* middleware creates a context-management tool in a composite, single-screen application, while *device* (or *robotics*) middleware simplifies the integration of specific device operating systems or robotic hardware and firmware.

The first categorization is broader, emphasizing architecture operating principles, while the second categorization is application-driven. For this reason, the first segregation is preferable to define middleware accurately per architecture integration instead of its application properties. All types of middleware are presented in figure 1.

THE USE OF MIDDLEWARE

In developing an application, the three necessary elements to consider are scalability, maintenance, and automation.

FIGURE 1: **TYPES OF MIDDLEWARE**



First, developers avoid horizontal scaling, which is just adding resources to expand the capabilities of the main system. They strive for workload partitioning—optimally distributing job scheduling over the overall network. As for maintenance, the separation of concerns principle is very important for developers, both to make each entity reusable (modularity) and to bundle its properties (encapsulation). (Typical modular examples include the Linux kernel, since the code base can be altered [added/removed], and a LEGO set where different elements can be used multiple times to build a system.) Moreover, developers focus on automating operations to reduce errors and make an application available 24/7 or ad hoc (i.e., on-demand provisioning).

Middleware can act as a facilitator to achieve scalability, maintenance, and automation. Specifically, it adds a layer that simplifies complex systems into small integrations, allowing for their association with the network of distributed resources.[2] This means that middleware provides agility during software development while simultaneously decreasing the time for a full software cycle; it also provides developers with easier future scaling.

Moreover, middleware can support rapid prototyping by incorporating the “fail fast, succeed faster” principle. It allows developers to apply, adopt, and evaluate business changes instantly. Middleware can also reduce project cost and generally promote entrepreneurship and innovation.[9]

MIDDLEWARE’S CAPABILITIES

Before the widespread use of the Internet and the adaptation of high-speed connections, most applications were developed as single-tiered, independent software solutions. This software was “monolithic,” built to serve a specified purpose and activity, and thus not designed to connect and interact with other applications and software components. Single-tiered software needed a complex middleware solution either to share information with different modules, such as client/server, or to deliver a request from host/resource-management software.

After the Internet revolutionized the way developers operate, an increasing number of transactions driven by multiple computing devices connected to a large, distributed computer network (also referred to as the Internet of Things). *Distributed computing* introduced SOA (service-oriented architecture) instead of monolithic applications. Specifically, SOA consists of multitiered software solutions that implement the separation of entities and services, thus breaking down each component into microservices. This is achieved by decreasing the system’s complexity and further increasing its modularity. In this scenario, middleware considers each entity as unique and autonomous. Thus, future modifications are addressed to a specific service (module) and not to the overall system’s components.

This middleware “is an approach for developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.”[13]

Middleware is closely connected to APIs (application protocol interfaces), serving as the tier or a software bundle for different APIs used by a programmer. This means that middleware can simplify sophisticated applications so that the developer focuses on not only the communication of components but also the business logic and the systems' interaction. This is an important aspect in the era of IoT, since APIs are the main gateway to connect devices and send information without errors.

IOT MIDDLEWARE

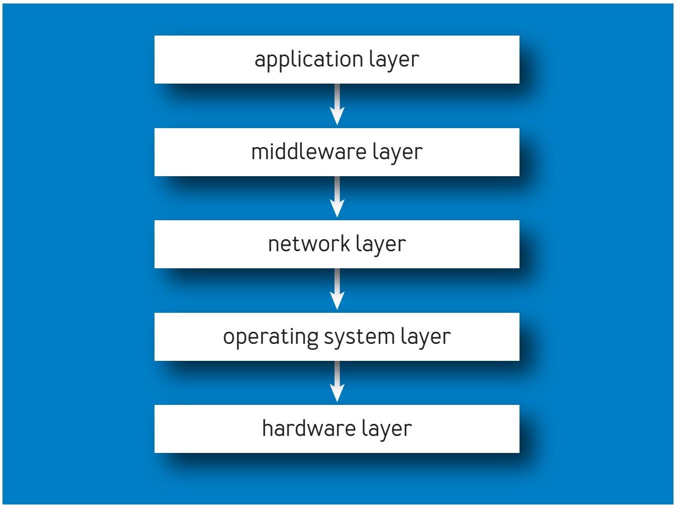
The term *IoT* describes a large network of interconnected devices that gather realtime data fused by multiple smart sensory devices.[15] To achieve that, electronic devices (mobile/tablet/computer) send data to an external service hosted in a cloud or edge-computing infrastructure. Recent research has focused on developing an IoT network that will not only interact with its surroundings, but also act autonomously without requiring a user's intervention. Under this scope, IoT researches pervasive/ubiquitous computing as the future of computing applications.[10] "Computers" are no longer associated with single devices or a network of devices. *Pervasive computing* is defined as "the entirety of situative services originating in a digital world, which are perceived through the physical world."[8]

Moreover, the architectural principles to develop an IoT application include a review of security, energy consumption and monitoring, reliability, interpretability, and communication.[19] As mentioned, middleware provides an abstract tier for all these functions. Based on its software licensing, it can be categorized as corporate-maintained, open-sourced, or device-specific (for microcomputers/actuators such as Raspberry Pi or Arduino).

Figure 2 depicts how IoT middleware typically handles its operations based on the following separation of concerns:

- The hardware layer (also known as the edge layer) includes all the sensory devices, as well as the sensor network in which they operate. This tier is responsible for gathering and processing the available data.
- The operating system's layer (i.e., the access gateway tier) performs the necessary data-transformation operations for information to be extracted and loaded accordingly.
- The network layer (i.e., the Internet layer) focuses on sending data to the next layer by securing a continuous, safe, and nondisruptive communication stream.
- The middleware layer handles the message communication protocols and services. Specifically, this layer checks systems for operation and data-transmission failures, in addition to providing access protocols for the application.

FIGURE 2: **IOT MIDDLEWARE LAYERS**



- Finally, the application layer is solely responsible for providing services (typically by API) to the end user by enabling the broadcast of services to various application endpoints (e.g., different developers and departments).

A review of some of the most promising open-source projects regarding IoT middleware[6] highlights the following: OpenIoT for sensor systems in the cloud; FIWARE for translating protocols of communication between devices; LinkSmart (formerly known as Hydra) for fast deployment and high scalability of data storage and machine learning; DeviceHive for IoT abstraction of automation layers regarding communication, control, and management; and ThingSpeak for industrial IoT frameworks regarding smart applications.

Similarly, IBM, AWS (Amazon Web Services), Microsoft Azure, Google, and Oracle have developed corporate middleware.[1] Based on the highlighted projects, several middleware frameworks focus on automating either a specific task or a core business activity process.

THE FUTURE: CLOUD CONTAINERS AND MICROSERVICES

While developers use virtualization (layering of resources into infrastructure), hypervisor (interpreters of the operating system), guest operating systems (with their own kernels), and applications, middleware promoted a decentralized deployment in a single multipurpose environment. This became evident following the exponential increase of *containers*—software environments that can be rapidly and easily deployed multiple times via the same server (host) in an isolated environment, also referred to as a *sandbox*. Like Java's motto, "Write once, run anywhere," containers are an independent software environment with unique code, libraries, runtime, and dependencies. Middleware tiers have also shifted from virtualization to containerization for the same purpose of optimizing communication and abstracting the communication protocols to develop a software pipeline.

From the scope of a developer,[7] the shift to cloud-computing solutions means that less coding is required, since most of the work in the cloud infrastructure is performed "under the hood." In other words, several aspects of distributed programming and enterprise development previously handled by a local middle tier can now be handled remotely. More specifically, common issues to be tackled include scaling, resilience observability, resource management, and continuous integration and delivery. This means that enterprises will limit the number of middleware developers. Instead of deployment, they will focus on architecture and application development.

CONCLUSION

Middleware can be used during several phases of the software cycle—from its architecture and development to its deployment. The perpetual need for the digital transformation of businesses (from monolithic to microservice implementations) has showcased that middleware is here to stay. Whether segregating a sophisticated software component into smaller services, transferring data between computers, or creating a general gateway for seamless communication, you can rely on middleware to achieve communication between different devices, applications, and software layers. Moreover, there is a need to educate new developers about middleware and highlight its importance through modern education techniques and learning systems.[5,17]

Following the increasing agile movement, the tech industry has adopted the use of fast waterfall models to create stacks of layers for each structural need, including integration, communication, data, and security. Given this scope, it is no longer important to study the potential expansion of cloud or data services. Emphasis must now be on endpoint connection and agile development.

This means that middleware should not serve solely as an object-oriented solution to execute simple request-response commands. Middleware can incorporate pull-push events and streams via multiple gateways by combining microservices architectures to develop a holistic decentralized ecosystem.

References

1. Agarwal, P., Alam, M. 2020. Investigating IoT middleware platforms for smart application development. *Smart Cities—Opportunities and Challenges* 58, 231–244. Springer; https://link.springer.com/chapter/10.1007/978-981-15-2545-2_21.
2. Al-Jaroodi, J., Nader, M. 2012. Middleware is STILL everywhere!!! *Concurrency and Computation: Practice and Experience* 24(16), 1919–1926; https://dl.acm.org/doi/abs/10.1002/cpe.2817.
3. Becker, C., Julien, C., Lalanda, P., Zambonelli, F. 2019. Pervasive computing middleware: current trends and emerging challenges. *CCF Transactions on Pervasive*

Computing and Interaction 1(1), 10–23. Springer; https://link.springer.com/article/10.1007/s42486-019-00005-2.

4. Bishop, T.A., Ramesh, K. K. 2003. A survey of middleware. In *Proceedings of 18th International Conference on Computers and Their Applications*, 254–258; https://www.researchgate.net/publication/221205414_A_Survey_of_Middleware.
5. Ciufudean, C., Buzduga, C. 2020. Digital engineering education applications. *Transactions on Adanvces in Englineering Education* 17, 10–14; http://www.doi.org/10.37394/232010.2020.17.2.
6. da Cruz, M.A.A., Rodrigues, J.J.P.C., Sangaiah, A.K., Muhtadi, J.A., Korotaev, V. 2018. Performance evaluation of IoT middleware. *Journal of Network and Computer Applications* 109(C), 53–65; https://dl.acm.org/doi/abs/10.1016/j.jnca.2018.02.013.
7. Farahzadi, A., Pooyan, S., Rezazadeh, J., Farahbakhsh, R. 2018. Middleware technologies for cloud of things: a survey. *Digital Communications and Networks* 4(3), 176–188; https://www.sciencedirect.com/science/article/pii/S2352864817301268.
8. Ferscha, A. 2009. Pervasive computing. In *Hagenberg Research*, ed. B. Buchberger et. al., 379–431. Springer; https://link.springer.com/chapter/10.1007/978-3-642-02127-5_9.
9. Helland, P. 2021. Fail-fast is failing... fast! *acmqueue* 19(1); https://dl.acm.org/doi/10.1145/3454122.3458812.
10. Hong, J. 2017. The privacy landscape of pervasive computing. *IEEE Pervasive Computing* 16(3), 40–48; https://ieeexplore.ieee.org/document/7994573.
11. IBM Cloud Education. 2021. What is middleware; https://

www.ibm.com/cloud/learn/middleware.

12. Zhang, J., Ma, M., Wang, P., Sun, X.D. 2021. Middleware for the Internet of Things: a survey on requirements, enabling technologies, and solutions. *Journal of Systems Architecture: The EUROMICRO Journal* 117(C); https://dl.acm.org/doi/abs/10.1016/j.sysarc.2021.102098.
13. Lewis, J., Fowler, M. 2014. Microservices: a definition of this new architectural term; https://martinfowler.com/articles/microservices.html.
14. Medeiros, R., Fernandes, S., Queiroz, P.G.G. 2022. Middleware for the Internet of Things: a systematic literature review. *Journal of Universal Computer Science* 28(1), 54–79; https://lib.jucs.org/article/71693/.
15. Pinus, H. 2004. Middleware: past and present a comparison. Seminar paper. https://www.researchgate.net/publication/265203339_Middleware_Past_and_Present_a_Comparison.
16. Salazar, G.D.S., Venegas, C., Baca, M., Rodríguez, I., Marrone, L. 2018. Open middleware proposal for IoT focused on Industry 4.0. *IEEE 2nd Colombian Conference on Robotics and Automation (CCRA)*, 1–6; https://ieeexplore.ieee.org/document/8588117.
17. Salem, A.B.M., Mikhalkina, E.V., Nikitaeva, A. Y. 2020. Exploration of knowledge engineering paradigms for smart education: techniques, tools, benefits and challenges. *Transactions on Advances in Engineering Education*, 1–9; http://www.doi.org/10.37394/232010.2020.17.1.
18. Schantz, R.E., Schmidt, D.C. 2002. Middleware. In *Encyclopedia of Software Engineering*, ed. J. J. Marciniak. Wiley and Sons; https://onlinelibrary.wiley.

com/doi/10.1002/0471028959.sof205.

19. Singh, K.J., Kapoor, D.S. 2017. Create your own Internet of Things: a survey of IoT platforms. *IEEE Consumer Electronics Magazine* 6(2), 57–68; https://ieeexplore.ieee.org/document/7879392.

Alexandros Gazis *is a researcher and Ph.D. candidate in the department of electrical and computer engineering, Democritus University of Thrace in Xanthi. He is a member of the Technical Chamber of Greece and works as a software engineer at Eurobank S.A., specializing in core banking systems and mainframe development. He has published articles on artificial intelligence, big data, pervasive computing, web data analytics, remote sensing, and neural networks. His research focuses on the Internet of Things via wireless sensor networks, cloud computing, and middleware development for pervasive computing.*

Eleftheria Katsiri *is an assistant professor (with tenure) in the department of electrical and computer engineering, Democritus University of Thrace in Xanthi, where she teaches operating systems, structured programming, HCI (human-computer interaction), and distributed and parallel systems. Since 2014, she has been adjunct researcher of the Athena Research and Innovation Center. Her research interests are in the areas of high-level, programming tools for context-awareness in ubiquitous computing, IoT and wireless sensor networks, and distributed and parallel systems with realtime requirements.*