Chapter 5

Middleware and IoT

5.1 An Overview of Middleware

There are several historical stories that linguistically unite humanity across the planet: the Tower of Babel, Enmerkar and the Lord of Aratta, Xelhua, and Toltecs. Middleware deals with the babble between distributed systems and has a similar objective in bringing linguistic or communicative unity to disparate technological systems.

The term *middleware* stems from distributed computing and refers to a set of enabling services such as standardized APIs, protocols, and infrastructure services for supporting the rapid and convenient development of distributed services and applications based on the client/server and later multitiered paradigm, which was essential for migrating single-tiered mainframe/terminal applications to multitiered architecture. Middleware is about integration and interoperability of applications and services running on heterogeneous computing and communications devices.

The services it provides, including identification, authentication, authorization, soft-switching, certification, and security,

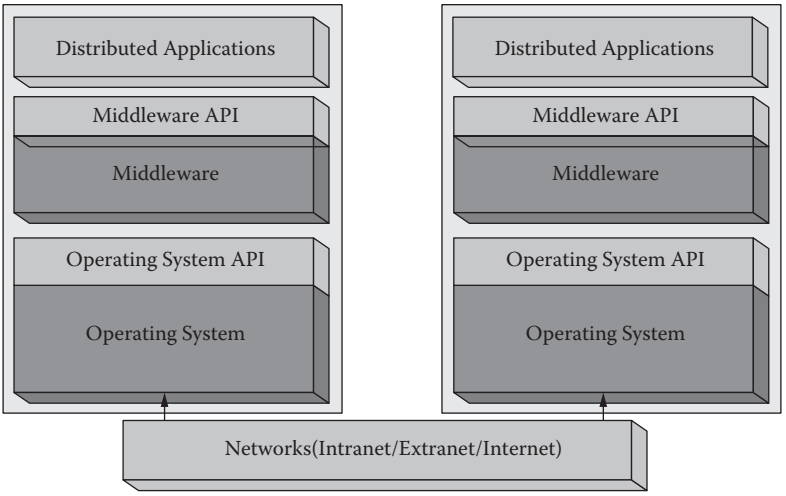


Figure 5.1 Omnipresent middleware.

are used in a vast range of global appliances and systems, from smart cards and wireless devices to mobile services and e-commerce. When the first distributed applications became widely used in the early 1990s, application developers were increasingly faced with a multitude of heterogeneous programming languages, hardware platforms, operating systems, and communication protocols, which complicated both the programming and deployment of distributed applications.

The term *middleware* refers to a layer that is arranged on top of operating systems and communications stacks and thus hides heterogeneity from the applications through a set of common, well-defined interfaces (Figure 5.1). In this way, the distributed client and server components of which an application is made up can be programmed in the same manner as if they were executed on the same host.

Middleware brings the following values to the table:

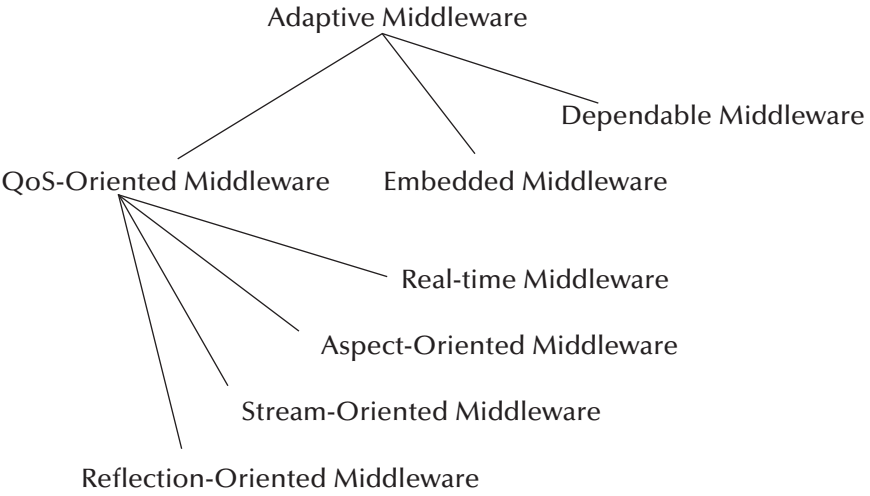
- Enables applications running across multiple platforms to communicate with each other
- Shields the developer from dependencies on network protocols, operating systems, and hardware platforms

- Is a software layer that lies between the operating system and the applications on each site of the system
- Hides heterogeneity and location independence
- Increases software portability
- Provides common functionality needed by many applications
- Aids application interoperability
- Aids scalability
- Helps integrate legacy facilities

Middleware is omnipresent and it exists nearly everywhere in an information and communications technology (ICT) system. Many kinds of middleware are described in related books [160,161]. A list of middleware is compiled below:

- Message-Oriented Middleware (MOM/MQ/JMS/ESB)
- CEP (complex event processing) Middleware (Tibco, Sybase)
- Adaptive and Reflective Middleware (TAO/DynamicTAO/OpenORB [80])
- Transaction Middleware (TPM/Tuxedo)
- Peer-to-Peer Middleware (JXTA)
- Grid Middleware (PVM/MPI/Schedulers)
- Model-Driven Middleware (CoSMIC)
- Games Middleware (Autodesk)
- Mobile Computing Middleware (OSA/Parlay/JAIN/OMA)
- Radio-frequency Identification (RFID) (Smart Cards) Middleware (Edgeware)
- Three-tiered Application Server Middleware (Weblogic, Websphere)
- Real-time CORBA Middleware (Real-time CORBA)
- High-Availability (Fault Tolerance) Middleware (Fault-Tolerant CORBA)
- Security Middleware (Siteminder)
- CATV/IPTV Middleware (MHP/GEM/OCAP) [181]
- RFID Edge Middleware (OATSystems, Sybase, Oracle, Tibco, SeeBeyond, IBM, SAP, Connectera, GlobeRanger, Manhattan Associates)

- Process-Oriented Middleware (WebMethods, SeeBeyond, Tibco, IBM, SAP, Oracle)
- Business-to-Business (B2B)-Oriented Middleware (SeeBeyond/Oracle, Tibco, webMethods)
- Middleware for Location-Based Services
- Surveillance Middleware



It's argued by some that with middleware proliferation these days, middleware is everywhere (there is also the concept of Everyware [19], an IoT software platform based on OSGi). It seems like every time that more than two applications need to be integrated, a piece of middleware has been deployed to handle the task. The trouble is that this has led to a lot of middleware sprawl because most of these middleware deployments are tactical, as opposed to being part of information technology (IT) strategy.

Middleware is also the software "glue" that helps programs and databases running on different computers to work together. Gartner formally defines middleware as: "Runtime system software that directly enables application-level interactions among programs in a distributed computing environment" [73].

The basis for nearly all middleware approaches was formalized by the International Organization for Standardization (ISO), which defined the common principles and structures of middleware in a framework known as Reference Model for Open Distributed Processing (RM-ODP). The main objective of ODP is to achieve distribution, interworking, and portability in an environment of heterogeneous IT resources and multiple organizational domains of different participants. ODP groups the functions of middleware into different transparency mechanisms, such as location, failure, persistence, transaction, and scalability. Each of them provides a number of APIs and services to the developer for masking the complexity associated with the respective functions.

The common principles of ODP have been adopted by many of the major middleware platforms, such as OSF DCE (Open Software Foundation's Distributed Computing Environment), Common Object Request Broker Architecture (CORBA), Java's Remote Method Invocation (RMI) and Java EE, .NET/DCOM of Microsoft, LAMP (Linux, Apache, MySQL, PHP/Perl/Python), and several approaches for web services. All of these provide several infrastructure services and support different communication patterns, for example, synchronous and asynchronous interactions.

A taxonomy of middleware functionality is outlined by Gartner [73] with three major categories: the integration middleware, the basic middleware, and the development and management tools. More than a dozen different functions that can be performed by middleware have been identified.

The integration middleware covers business- and application-oriented commonalities that include the following:

- Business process management
- Business rule engine/workflow
- Business event management
- Data routing and adapters

The basic middleware is the foundation, which applies to the Internet of Things (IoT) infrastructure also, and it can be further categorized as follows:

- Data management middleware: helps programs read from and write to remote databases or files. Examples of this kind of middleware include distributed and parallel file systems, such as Google File System, IBM GPFS, Network File System, and Windows, and also include the remote database access middleware, such as Open Database Connectivity or Java Database Connectivity libraries that are bundled into DBMSs such as IBM DB2, Oracle, and Microsoft SQL Server.
- Communication middleware: software that support protocols for transmitting messages or data between two points as well as a system programming interface (SPI) to invoke the communication service. More-advanced communication middleware (such as message-oriented middleware) also support safe (e.g., using strong security) and reliable (e.g., guaranteed once and only once) delivery of messages. Protocols and SPIs used in communication middleware can be proprietary (e.g., IBM WebSphere MQ/MQ-TT or Microsoft MSMQ) or based on industry standards such as ASN.1, DCE remote procedure call (RPC), CORBA/IIOP, Java Message Service (JMS), or web services (based on SOAP or REST). Today's communication middleware generally runs on Internet-based protocols such as HTTP (HTTPS), IP, SMTP, and so forth. It may implement higher level protocols, including industry standards (e.g., ebXML messaging and web services), and proprietary protocols (e.g., Oracle AQ), and it may run over the Internet or private networks. Communication middleware also includes *embedded middleware.* Research has been done on middleware and associated standard protocols for home automation and building controls [225,266]. Table 5.1 is a list of some emerging IoT middleware projects.

Table 5.1 Embedded IoT Middleware

IoT Middleware	*Features of Middleware*					
	Device Management	*Interoperation*	*Platform Portability*	*Context Awareness*	*Security and Privacy*	*Protocols*
HYDRA	Yes	Yes	Yes	Yes	Yes	Agnostic
ASPIRE	Yes	No	Yes	No	No	RFID
ISMB	Yes	No	Yes	No	No	RFID, etc.
UBIWARE	Yes	No	Yes	Yes	No	RFID/WiFi, etc.
UBISOAP	Yes	Yes	Yes	No	No	RFID/WiFi, etc.
UBIROAD	Yes	Yes	Yes	Yes	Yes	RFID/WiFi, etc.
GSN	Yes	No	Yes	No	Yes	RFID/WiFi, etc.
SMEPP	Yes	No	Yes	Yes	Yes	WiFi, etc.
SOCRADES	Yes	Yes	Yes	No	Yes	RFID, etc.
SIRENA	Yes	Yes	Yes	No	Yes	RFID, etc.
WHEREX	Yes	Yes	Yes	No	No	Agnostic
MQ-TT	Yes	Yes	Yes	Yes	Yes	Agnostic
Everyware	Yes	Yes	Yes	Yes	Yes	Agnostic
^{ez}M2M	Yes	Yes	Yes	Yes	Yes	Agnostic

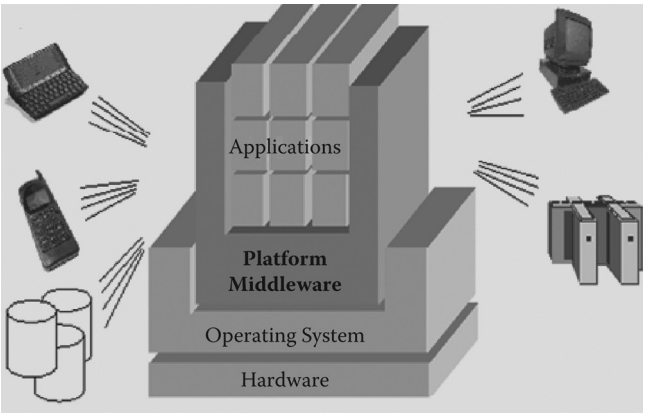


Figure 5.2 Platform middleware.

- Platform middleware: provides the runtime hosting environment (a container) for application components (see Figure 5.2). It uses embedded or external communication middleware to help programs interact with other programs. It also provides resource management services for hosting application modules at runtime (caching, starting, stopping, and multiplexing programs, load balancing, fault tolerance, access security, monitoring and management, distributed transaction processing, etc.). Platform middleware also provides interfaces to one or several forms of communication middleware (one-way messaging and request/reply). Platform middleware is well known today as *application servers* (JAVA EE or .NET Framework/COM+). However, historically, many other product categories have served as then-prevailing platform middleware. Examples include mainframe transaction processing monitors (TPMs such as IBM CICS), Unix-distributed TPMs (such as BEA Tuxedo; the author used to be part of the team), extended RPC implementations, extended object request brokers (ORBs) and object transaction monitors, DBMS stored procedures platforms, proprietary fourth-generation languages, and programmable web servers. Platform middleware has been evolving further in part

because of the growing interest in *portal* services such as personalization, multichannel access, and content management. Numerous vendors offer portal services as separate products such as BEA Weblogic Portal, Plumtree, Vignette, and others that are meant to complement web servers and application servers.

Middleware and the applications software built on top of it are becoming increasingly important in the networked device marketplace. For the nonnetworked device market, the profit is from the device product itself. For the networked device market, additional profits come from consumables, services, and contents. According to Harbor Research, after the "transition point," "the device itself becomes secondary to the value it brings to the customers. Connectivity become the means to cultivate an ongoing relationship." R. Achatz, chairman at Siemens Corporate Research, noted, "We have more software developers than Oracle or SAP, but you don't see this because it is embedded in our trains, machine tools and factory automation" [171]. The landscape of CapEx (Capital Expenditure) and OpEx (Operation Expenditure) is changing.

Device miniaturization, wireless computing, and mobile communication are driving ubiquitous, pervasive, and transparent computing. Supporting these rapidly evolving technologies requires middleware solutions that address connectivity-level, location-dependent, and context-dependent issues. Many companies have developed common application *platform middleware* frameworks for M2M or IoT applications, which will be discussed in more detail in Chapter 7. We will talk more about *communication middleware* in the following sections with regard to its association with M2M or IoT applications.

5.2 Communication Middleware for IoT

In a runtime environment, the DCM (device, connect, and manage) three-layer model can be further extended into more

layers depending upon the geographical scope of the area network (AN) from BAN to interplanetary Internet as listed below:

- Body (BAN)
- Personal (PAN)
- Near-me (NAN)
- Machine-to-machine, or M2M (MAN)
- Local (LAN)
  - Home (HAN)
  - Storage (SAN)
- Campus (CAN)
- Backbone
- Metropolitan (MAN)
- Wide (WAN)
- Internet
- Interplanetary Internet

In this section, we will talk about the extensions and enhancements of the existing technologies in the device and connect layers. If the IoT applications are to be extended from the current insolated Intranet or Extranet environments to the wide area as well as global Internet landscape, some fundamental changes in the networking systems have to be considered in a converged next-generation network (NGN) setting.

Some efforts such as the (open-source) Hydra project are under way to build a *unified communication network middleware* for IoT applications. Hydra [133] (networked embedded system middleware for heterogeneous physical devices in a distributed architecture) is a European Union–sponsored IoT open-source project (FP6 IST-2005-034891) that aims to reduce the complexity by developing service-oriented middleware.

5.2.1 MTC/M2M Middleware

The 3GPP (Third Generation Partnership Project) is a collaboration between groups of telecommunications associations

known as the Organizational Partners. The Organizational Partners are the European Telecommunications Standards Institute (ETSI), Association of Radio Industries and Businesses/Telecommunication Technology Committee (Japan), China Communications Standards Association, Alliance for Telecommunications Industry Solutions (North America), and Telecommunications Technology Association (South Korea). The project was established in December 1998.

The connect layer of DCM can be further divided into three layers based on 3GPP's efforts for GSM/WCDMA family (3GPP2 for CDMA family) cellular wireless M2M standardization: the M2M area network layer, the access/core network layer, and the external/Internet network layer, as depicted in the 3GPP/ETSI graphic in [230]. The M2M platform in the graphic is an IoT platform middleware at the "M" layer in the DCM value chain.

- M2M area network—provide wired or wireless connectivity between M2M devices and M2M gateways, such as personal area network
- M2M access/core network—ensure M2M devices interconnection from the gateways to the access/core communication network, such as GPRS/GSM (GGSN [Gateway GPRS Support Node], SGSN [Serving GPRS Support Node], etc.; WCDMA, and others
- External/Internet networks (long distance)—communicate between the 3GPP access/core network and the M2M middleware platform for applications, such as Internet, corporate WANs, and others

Even though 3GPP introduced the concept of the M2M area network and tries to cover RFID, wireless sensor network (WSN), and supervisory control and data acquisition (SCADA) application scenarios, it is applicable for GSM/WCDMA cellular M2M only. 3GPP's coverage/scope for the entire four-pillar IoT networking possibilities are limited. Other IoT applications, for

example, SCADA, may not use cellular networks at all. Those scenarios will be discussed later.

The concept of machine-type communication (MTC) was introduced by 3GPP [76]. MTC is the term 3GPP used for cellular M2M communication. It refers to communication without (or with limited) human intervention; data are input or generated by machines instead of humans, which can be significantly faster. Most future big data growth will be in the area of M2M machine-generated data, examples of which include

- Satellite-based telemetry application-generated data
- Location data such as RFID chip readings, global positioning system (GPS) output
- Temperature and other environmental sensor readings
- Sensor readings from factories and pipelines
- Output from many kinds of medical devices, in hospitals and homes alike

In 2009, Gartner estimated that data will grow by 650 percent in the following five years. Most of the growth in data is the by-product of machine-generated data, which could also create M2M data burst to the network systems. New communication middleware will play an important role in alleviating or protecting such overloads.

Current mobile networks are optimized for human-to-human communication, not for MTC. The following are some of the characteristics of MTC summarized by 3GPP (more shown in Table 5.2):

- Time tolerant—data transfer can be delayed
- Packet switched only—network operator shall provide PS service with or without a Mobile Station International Subscriber Directory Number (MSISDN)
- Online small data transmissions—MTC devices frequently send or receive small amounts of data
- Location-specific trigger—intending to trigger MTC device in a particular area, e.g., wake up the device

Table 5.2 MTC Characteristics

Characteristics Example Applications	*Data Volume*	*Quality of Service*	*Amount of Signaling*	*Time Sensitivity*	*Mobility*	*Server Installed Communication*
Smart energy meters	Low	Low	Intermediate	Very low	No	Yes
Road charging	Low	Low	Low	Low	Yes	No
eCall	Very low	Very high	Very low	Very high	Yes	No
Remote maintenance	Low	Low	High	High	No	Yes
Fleet management	Low	Low	Very high	Intermediate	Yes	Yes
Photo frames	Intermediate	Low	High	Low	No	Yes
Asset tracking	Low	Low	Very high	High	Yes	Yes
Mobile payments	Intermediate	Low	High	Very high	Yes	No
Media synchronization	High	Low	High	Intermediate	Yes	Yes
Surveillance cameras	Very high	Very high	Low	Very high	No	Yes
Health monitoring	High	High	High	Very high	Yes	Yes

- Group-based MTC features—MTC device may be associated with one group
- Extra-low power consumption—improving the ability of the system to efficiently service MTC applications

3GPP started the specification for MTC in early 2010; efforts are proposed as follows [66]:

- Provide network operators with lower operational costs when offering MTC services
- Reduce the impact and effort of handling large MTC groups
- Optimize network operations to minimize impact on device battery power usage
- Stimulate new MTC applications by enabling operators to offer services tailored to MTC requirements
- Prepare for number and IP address shortages

Below are issues with current telco networks for M2M:

- 3GPP SA1 has required solutions to cater for at least two orders of magnitude more devices compared with human to human.
- Shortage of telephone numbers.
- Shortage of IPv4 addresses.
- ISMI range seems large enough for most operators.

Network agnostic middleware approaches for matching application and service requirements with available network capabilities in the telecommunication domain are abundant:

- OSA-Parlay of 3GPP, Parlay-X
- JAIN (Java APIs for integrated networks)
- Open Mobile Alliance (OMA)
- Universal Plug and Play (UPnP)
- Devices Profile for Web Services (DPWS)
- Home Audio-Video Interoperability (HAVi)
- Jini and other middleware alternatives

It seems what the 3GPP's M2M effort lacks is specifying a unified middleware framework for all MTC networks. Middleware for networks is discussed in many works [78,79]. Sahin Albayrak et al. [77] emphasized that "we firmly believe that a new middleware architecture with innovative aspects in terms of: full support along the whole path rather than at the front and backend nodes, highly service aware networks, network aware services, and intelligent coordination and cooperation capabilities is the right answer to the upcoming challenges in next generation networks."

As networks evolve today, middleware based on the aforementioned OSA/Parlay, JAIN, and others for MTC is an area that requires more investigation and integration in the near future. In addition to the MTC optimization of the cellular wireless network, other optimizations or service enablement middleware (described in Chapter 3) are discussed [226,227] and their standardizations are also needed for M2M applications. Service enablement can be built as middleware that provides reliable and efficient connectivity for adjacent industry applications and to enable operators to

- Act as horizontal service providers across applications and industries
- Expand their role as managed service providers
- Capture maximum value as smart service providers

Nokia is one of the earliest vendors that offered M2M middleware. The Nokia M2M platform [228] is based on open, widely accepted middleware (built on CORBA) and communications architecture, and it supports standard GSM technology with a choice of wireless bearers. Open interfaces facilitate easy development, operation, and maintenance of various M2M applications and services, and provide an easy upgrade path for future technologies. IBM also built an MQ-TT (telemetry transport) middleware (http://mqtt.org/) for M2M applications over IP and non-IP networks.

Other kinds of M2M terminals are the CATV STB (set top box), globally executable MHP (GEM), and MHP (multimedia home platform, based on Java technologies) [128]. These are two of the middleware standards for cable TV, IPTV, Blu-Ray player terminals (embedded middleware), and head-end (platform middleware) applications. GEM, based on MHP, is also a recommended standard by ETSI and ITU.

STB-based home gateway terminal is also an important IoT/M2M application that has been developed for many years. Other middleware for STB M2M devices and head-end systems include Multimedia and Hypermedia Information Coding Expert Group (MHEG), Open Cable Application Program (OCAP), OpenTV, MediaHighway, Digital Video Broadcasting (DVB)-HTML, etc. All-IP convergence applications based on converged middleware will make the "triple network convergence" of China a reality.

In the digital home (or home automation, domotics) scenario, middleware technology refers to a layer of software that lies on top of a home device's or appliance's operating system. Middleware facilitates rapid development and increases scalability of a system and integration of services in digital homes. It bundles hardware and software into a single solution and provides transparent interaction between home systems and databases, enables unified user interfaces, reduces infrastructure requirements, and makes multiple services easier to manage. A typical digital home could have a number of home devices and appliances, which allows the physical interconnection of multiple systems and services. Home systems and services are inevitably supplied by different manufacturers and use a wide range of different protocols and standards for communication. The home systems and services must be interconnected seamlessly with a consistent middleware platform. An example of the integration architecture of middleware with various digital home services based on standards such as UPnP, DPWS, Jini, HAVi, and so forth is available [231].

5.2.2 SCADA Middleware

The concept of MAN (M2M area network) was introduced in 3GPP/ETSI's MTC specification. This concept also applies to other pillar segments of IoT. However, not all IoT applications will use a cellular network. In fact, most of the traditional SCADA applications have been using local wireline networks for communications. The remote terminal units (RTUs), programmable logic controllers (PLCs), or even process control systems (PCSs) communicate to the SCADA middleware server via gateways (similar to MAN but all wired) that aggregate data from different wired field buses. The SCADA system is accessed in a LAN environment (sometimes xDSL, cable, WiFi, or WiMax can be used) before it is integrated into the corporate back office system.

Considering that many of the field buses also support IP, such as Modbus TCP/IP, BacNet IP, and others, it is possible or easier than wireless networks to adopt an all-IP approach to implement SCADA applications. This approach has been used in some of the projects done by the author in building management systems. Figure 5.3 (redrawn based on concepts from [264]) depicts the role of SCADA middleware in such a scenario in more detail.

Companies providing such SCADA middleware products include the following:

- Central Data Control: CDC provides the software platform Integra, which utilizes data agents to translate protocols from different building system components into single management system.
- Elutions: Its Control Maestro product has a SCADA heritage. SCADA may be best known for industrial processes but is also deployed for infrastructure (water treatment plants, gas pipelines, etc.) as well as facility systems. Control Maestro is web-based, uses human–machine

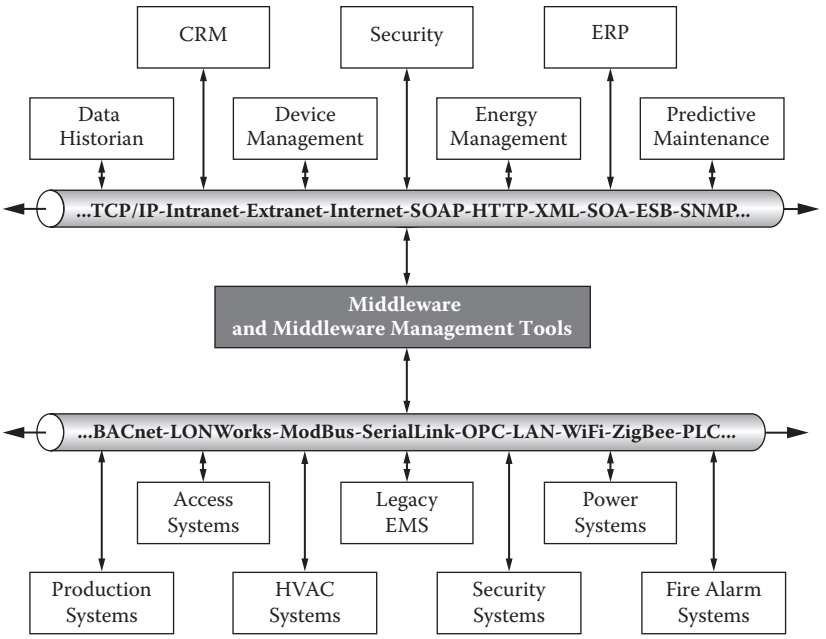


Figure 5.3 SCADA middleware architecture.

interfaces (HMI), and is able to deliver real-time and historical information.

- Richards Zeta: RZ's middleware solution is a combination of system controllers and software.
- Tridium: It provides the Niagara Java-based middleware framework and JACE hardware controllers. The Niagara platform provides protocol translation for a range of systems and the tools to build applications. Niagara has open APIs to all Niagara services and an extensible component model (XML) that enable development of applications by third parties. It also provides support for web-services data handling and communications with enterprise applications.

With the development of wireless technologies, systems have been developd that blend wireless with wired communication in SCADA applications. SensiLink™ is a middleware

and software suite from MeshNetics that links wireless sensor networks with SCADA systems. Sensor data collected from the nodes is channeled through RS232, RS485, USB, Ethernet, or GPRS gateway to the SensiLink server.

OPC middleware products are one of the important communications layer SCADA middleware that are designed to enhance any OPC standards-based applications. Originally, OPC was defined as a standardized solution for the recurring task of connecting PC-based SCADA/HMI applications with automation and process control devices. Today, the OPC standard has evolved into a robust data carrier able to transport entire enterprise resource planning documents and even video signals.

OPC is for Windows only (details about the standard is discussed in Chapter 6). Tridium is arguably the first SCADA middleware based on Java technology. Recent developments have integrated new technologies such as Java and iOS (application store) to build OS platform agnostic middleware for broader IoT applications; adopting new technologies for SCADA is a trend.

5.2.3 RFID Middleware

RFID networking shares a similar three-tiered communication architecture (as shown in Figure 5.4). RFID readers are the gateways similar to MAN. Data from the readers go to the corporate LAN and then are transmitted to the Internet as needed. However, just like the scenarios of M2M and SCADA, most current RFID systems stop at the corporate LAN level and are IoT systems only.

RFID middleware (including the edge middleware or edgeware) is currently no doubt the most well-defined, comprehensive, standardized middleware compared with the other three pillar segments of IoT. Before 2004, an RFID middleware-based system was defined by EPCglobal, which included:

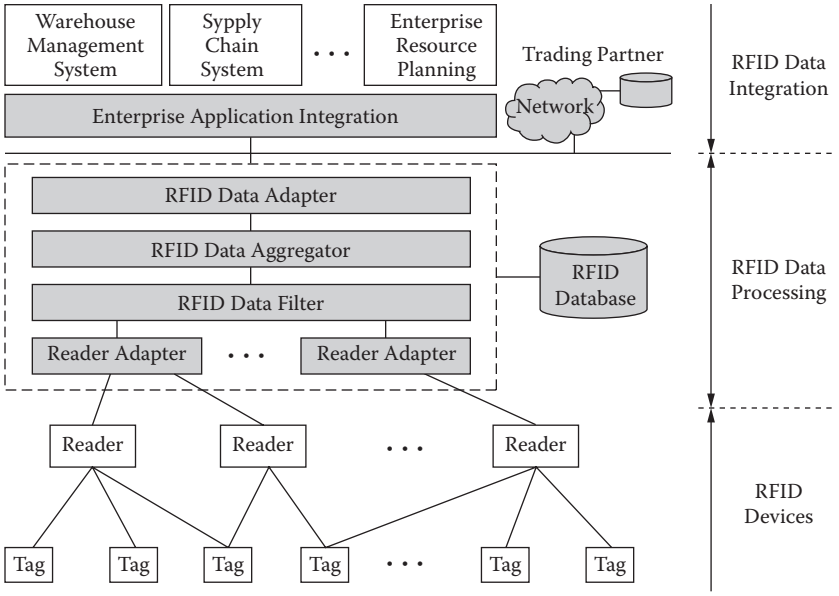


Figure 5.4 RFID architecture. (From Quan Z. Sheng, Kerry L. Taylor, Zakaria Maamar, and Paul Brebner, "RFID Data Management: Issues, Solutions, and Directions," in Lu Yan, Yan Zhang, Laurence T. Yang, and Huansheng Ning (Eds.), *The Internet of Things: From RFID to the Next-Generation Pervasive Networked Systems*, New York: Auerbach Publications, 2008.)

- A format for the data called physical markup language (PML), based on XML (Figure 5.5 is an example)
- An interface to the servers containing PML records
- A directory service called ONS (object naming service), analogous to the DNS. Given a tag's EPC, the ONS will provide pointers to the PML servers containing records related to that tag.

However, since 2004, the unified PML schema has been dropped [51] due to, most likely, practical reasons because most RFID systems are still in the "Intranet of Things" scope. Using the generic PML/ONS approach would involve overhead and sacrifice efficiency. Instead, the PML-like schema was left to the vertical applications to define their own XML

```
<pmlcore:Sensor>
   <pmluid:ID>urn:epc:1:4.16.36</pmluid:ID>
   <pmlcore:Observation>
      <pmlcore:DateTime>2002-11-06T13:04:34-06:00</pmlcore:DateTime>
      <pmlcore:Tag>
         <pmluid:ID>urn:epc:1:2.24.400</pmluid:ID>
         <pmlcore:Sensor>
            <pmluid:ID>urn:epc:1:12.8.128</pmluid:ID>
            <pmlcore:Observation>
               <pmlcore:DateTime>2002-11-06T11:00:00-
06:00</pmlcore:DateTime>
               <pmlcore:Data>
                  <pmlcore:XML>
                     <TemperatureReading xmlns="http://sensor.example.org/">
                        <Unit>Celsius</Unit>
                        <Value>5.3</Value>
                     </TemperatureReading>
                  </pmlcore:XML>
               </pmlcore:Data>
            </pmlcore:Observation>
            <pmlcore:Observation>
               <pmlcore:DateTime>2002-11-06T12:00:00-
06:00</pmlcore:DateTime>
               .....
               .....
```

Figure 5.5 Physical markup language sample.

scheme. Consequently, the overall system architecture of RFID has evolved from a dedicated structure to a more generic, open architecture.

However, the PML approach is believed to be a good IoT data representation method that should be used when the day of the full-blown IoT system comes. Other efforts such as M2MXML (from BiTX) and oBIX (an OASIS standard) are under way that are trying to build a generic IoT data schema, which is discussed in the next chapter.

An example of commercial RFID middleware product is IBM's WebSphere Sensor Events. WebSphere Sensor Events delivers new and enhanced capabilities to create a robust, flexible, and scalable platform for capturing new business value from sensor data. WebSphere Sensor Events is the platform for integrating new sensor data, identifying the relevant business events from that data using situational event processing, and then integrating and acting upon those events with SOA business processes.

The blending or convergence of different pillar IoT applications to build cross-segment IoT systems is a trend that has been demonstrated [228], in which unified data representation and associated communication middleware became more and more important.

5.2.4 WSN Middleware

Middleware also can refer to software and tools that can help hide the complexity and heterogeneity of the underlying hardware and network platforms, ease the management of system resources, and increase the stableness of application executions. WSN middleware is a kind of middleware providing the desired services for sensor-based pervasive computing applications that make use of a WSN and the related embedded operating system or firmware of the sensor nodes [57]. In most cases, WSN middleware is implemented as embedded middleware on the node [82].

It should be noted that while most existing distributed system middleware techniques aim at providing transparency abstractions by hiding the context information, WSN-based applications are usually required to be context aware, as mentioned in Chapter 1 [18].

A complete WSN middleware solution should include four major components: programming abstractions, system services, runtime support, and quality of service (QoS) mechanisms. Programming abstractions define the interface of the middleware to the application programmer. System services provide implementations to achieve the abstractions. Runtime support serves as an extension of the embedded operating system to support the middleware services. QoS mechanisms define the QoS constraints of the system. The system architecture of WSN middleware is shown in Figure 5.6.

Middleware for WSN should also facilitate development, maintenance, deployment, and execution of sensing-based applications. Many challenges arise in designing middleware for WSN due to the following reasons and more:

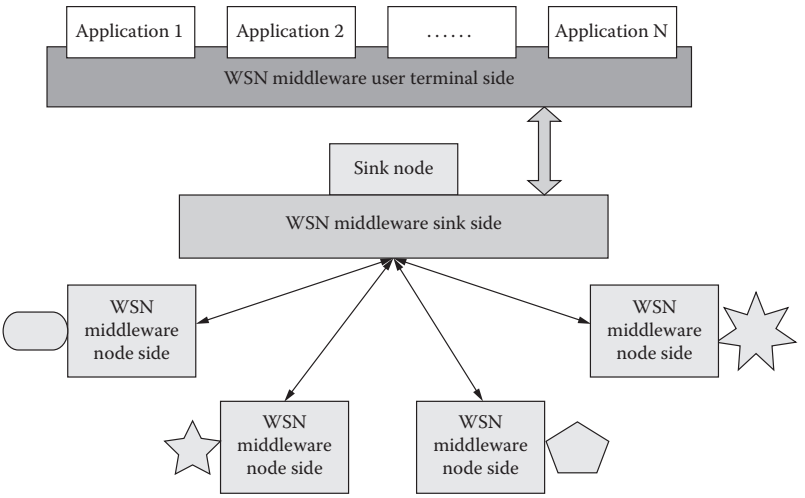


Figure 5.6 WSN middleware architecture.

- Limited power and resources, e.g., battery issues
- Mobile and dynamic network topology
- Heterogeneity, various kinds of hardware and network protocols
- Dynamic network organization, ad-hoc capability

WSN middleware is designed using a number of approaches such as virtual machine, mobile agents, database based, message-oriented, and more. Example middleware are as follows [83]:

- MagnetOS (Cornell University): power-aware, adaptive; the whole network appears as a single JVM, standard Java programs are rewritten by MAGNET as network components, and components may then be "injected" into the network using a power-optimized scheme.
- IMPALA: modular; efficiency of updates and support dynamic applications; application adaption with different profiles possible; energy efficient; used in the ZebraNet project for wildlife monitoring.

- Cougar: represents all sensors and sensor data in a relational database; control of sensors and extracting data occurs through special SQL-like queries; decentralized implementation; message passing based on controlled flooding.
- SINA (system information networking architecture): based on a spreadsheet database wherein the network is a collection of data sheets and cells are attributes; attribute-based naming; queries performed in an SQL-like language; decentralized implementation based on clustering.
- MIRES: publish/subscribe; multihop routing; additional service (e.g., data aggregation); sense—advertise over P/S and route to sink.
- MQTT-S (Message Queue Telemetry Transport for Sensors, IBM): a publish/subscribe messaging protocol for WSN, with the aim of extending the MQTT protocol beyond the reach of TCP/IP infrastructures (non-TCP/IP networks, such as Zigbee) for sensor and actuator solutions; a commercial product.
- MiLAN: provides a mechanism that allows for the adaptation of different routing protocols; sits on top of multiple physical networks; acts as a layer that allows network-specific plug-ins to convert MiLAN commands to protocol-specific ones that are passed through the usual network protocol stack; can continuously adapt to the specific features of whichever network is being used in the communication.

The WSN middleware is considered to be "proactive" middleware in the middleware family. A more comprehensive list of existing WSN middleware platforms, software/OS, and programming languages is shown in Table 5.3. A comparison of some of the WSN middleware is available [84].

As an example, the Agilla middleware is examined here in more detail (Figure 5.7). The Agilla [229] runs on top of TinyOS and allows multiple agents to execute on each node. The number of agents is variable and is determined primarily by the

Table 5.3 Sample WSN Middleware and WSN Languages

WSN Middleware			
Agilla	eCos	MagnetOS	SINA
AutoSec	EMW	MANTIS	SOS
Bertha	Enviro-Track	Mate	TinyDB
BTnut Nut/OS	EYESOS	MiLAN	TinyGALS
COMiS	FACTS	Mire	TinyOS
Contiki	Global Sensor Networks (GSN)	Netwiser	t-Kernel
CORMOS	Impala	OCTAVEX	VIP Bridge
COUGAR	jWebDust	SenOS	
DSWare	LiteOS	SensorWare	
WSN Languages			
c@t	DCL (Distributed Compositional Language)	galsC	nesC
Protothreads	SNACK	SQTL	

amount of memory available. Each agent is autonomous but shares middleware resources with other agents in the system.

Agilla provides two fundamental resources on each node: a neighbor list and a tuple space. The neighbor list contains the addresses of neighboring nodes. This is necessary for agents to decide where they want to move or clone to next. The tuple space provides an elegant decoupled-style of communication between agents. It is a shared memory architecture that is addressed by field-matching rather than memory addresses. A tuple is a sequence of typed data objects that is inserted into the tuple space. The tuple remains in the tuple space even if the agent that inserted it dies or moves away. Later, another agent may retrieve the tuple by issuing a query for a tuple with the same sequence

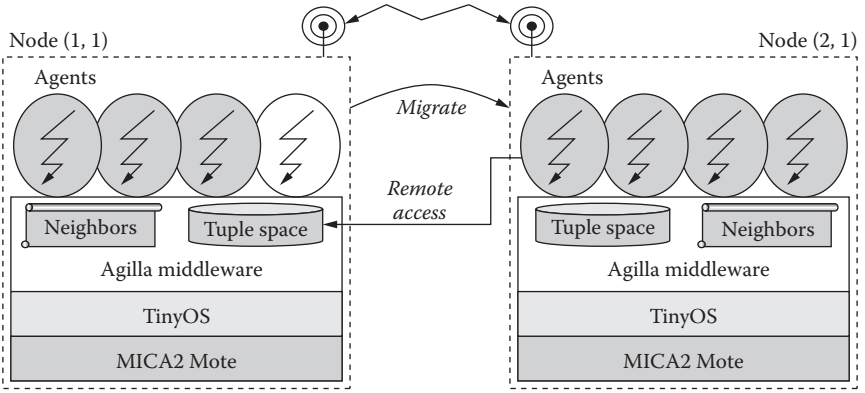


Figure 5.7 The Agilla middleware model. (From Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu, "Software Support for Application Development in Wireless Sensor Networks," in Paolo Bellavista and Antonio Corradi (Eds.), *The Handbook of Mobile Middleware*, New York: Auerbach Publications, 2006.)

of fields. Note that tuple spaces decouple the sending agent from the receiving agent: they do not have to be co-located, or even aware of each other's existence, for them to communicate. This is basically a fault-tolerant distributed computing technology.

All of the above WSN middleware are at the device level up to the gateways (equivalent to the MAN of MTC). Most of them are research projects conducted at universities and research institutions with a few experimental uses and of limited commercial value. This situation is very much like the research on parallel computing architecture one or two decades ago. There was a proliferation of parallel architectures [85] such as hypercube, wavefront arrays, pyramids, systolic arrays, and others, which the author has gone through [86–95]. Many research papers have been produced, but none of these architectures exist in the real world now. Nowadays, 99 percent of the world's fastest high performance computing (HPC) supercomputers use the simple massive parallel processing (MPP) architecture [96]. David Culler, the inventor of TinyOS and Mote, professor at University of California–Berkeley, was one of the

prominent researchers on parallel architecture at that time. He has been doing research on WSN since the wane of parallel architecture research. In fact, some of the WSN architecture and middleware ideas are inherited from parallel computer architectures, which will most likely diminish the same way as time passes by, especially the ad hoc wireless networks (they may have greater value in military uses).

Nevertheless, once the data from the ad hoc mesh WSN reaches the gateways, or if the wireless sensors are directly connected to the higher-tier networks, the remaining process and route to reach the Internet of Things will be the same as the other pillar segments of IoT. The WSN middleware at the system level may be the same as SCADA or M2M or RFID systems, which share the same three-tiered architecture discussed in the last three sections.

5.3 LBS and Surveillance Middleware

Other than the communication middleware and the platform middleware (which will be covered in Chapter 7) for IoT applications, other middleware are related IoT or are part of IoT. Location-based service (LBS) and surveillance middleware are two of the examples we choose to cover in this chapter.

LBS is a service that integrates a mobile device's location or position with other information so as to provide added value to a user [97]. There are several uses of LBS, and some of them are direct IoT applications:

- News: information dissemination based on the location of a user, such as weather information
- Point of interest (POI): shows points of interest near the user or vehicles
- Directions: shows directions from the current location of a user
- Yellow pages: finds services near the user

- Fleet management: tracks positions of a transportation fleet
- Local advertisement: user receives advertisements according to his or her position
- Emergency: tracks current position of a user in an emergency
- Location-based games: player interacts with another player according to his or her position

LBS scenarios involve collecting, analyzing, and matching different types of information including user profiles (e.g., personal information and interests) and information dissemination profiles. For each piece of information, LBS systems have to handle different aspects:

- Spatial: LBS middleware must be able to collect information about mobile position and fixed elements, associate them with physical/logical maps, and efficiently match locations and regions.
- Temporality: Location information has a temporal dimension that must be included in query capability.
- Inaccuracy, imprecision, and uncertainty: LBS must deal with inaccuracy and imprecision associated with location positioning technologies.
- Large volumes: In real scenarios, LBS must handle large volumes of data; scalability is a very important issue.
- Continuous queries: In an LBS scenario, query executions are continuous, so the query engine of an LBS middleware must be efficient.

An example middleware architecture for LBS systems can be found at locationet.com (http://www.locationet.com/LBSmiddleware.php). Most LBS middleware can be categorized as event based (publish/subscribe), tuple space based, context aware, and data sharing based:

- Publish/subscribe: one of the most prominent middleware models, in which communication is defined in terms of exchanging asynchronous messages based on subscription.
- Tuple space: originally proposed to coordinate concurrent activities in parallel programming systems such as Linda, in which a process communicates with another process in a global collection of tuples. A tuple is a data element that contains values of a specified data type.
- DBMS-based: comprises the use of database interaction to implement a communication and coordination; many geographic information systems (GISs) operate according to this scheme. LBS architecture naturally fits the DBMS-model, such as user management systems and accounting information systems.

As an example, LocatioNet middleware is a product that meets mobile operators' needs for in-house location-privacy management, location billing functionality, provisioning interfaces, and links to various content databases. LocatioNet comprises a set of modules offered in any required combination:

- Comprehensive location privacy management: allows users to decide who can see their location, when, and how precisely, application by application
- Billing for location: gives operators a flexible set of billing options for their location and GPS services
- Provisioning: enables operators to provision user-to-location and GPS applications
- Content interfaces: enables operators to take advantage of content properties they have access to (such as local news, the weather, points of interest, traffic) by linking them to the location and GPS infrastructure

A Location API for J2ME has been specified as JSR-179 that enables mobile location-based applications for resource-limited

devices. Java middleware and applications can be developed based on the Location API standard. The Open GIS Consortium (OGC) also produced a specification about location services called OpenLS™ in 2003.

Automated video surveillance networks are a class of sensor networks (people argued that a video surveillance network without automatic image recognition and event detection or alert generation is not a sensor network but instead simply a video or image capture and transmission system) with the potential to enhance the protection of facilities such as airports and power stations from a wide range of threats. However, current systems are limited to networks of tens of cameras, not the thousands required to protect major facilities. Realizing thousand-camera automated surveillance networks demands sophisticated middleware and architectural support as well as replacing the ad hoc approaches used in current systems with robust and scalable methods.

The IBM Smart Surveillance Solution [100] is based on the MILS (middleware for large-scale surveillance) surveillance middleware and designed to work with a number of video management systems from partner companies. The MILS provides the data management services needed to build a large-scale smart surveillance application. While MILS builds on the extensive capabilities of IBM's Content Manager and DB2 systems, it is essentially independent of these products and can be implemented on top of third-party relational databases. The MILS take the automatically detected events from the SSE (smart surveillance engine) as inputs. An SSE is a class of surveillance algorithms such as the HMM (Hidden Markov Model) [99].

The IBM SSS system provides two distinct functionalities:

- Real-time user-defined alerts: The user defines the criteria for alerting with reference to a specific camera view, for example, parked car detection, tripwire, and so forth.

- Indexed event search: The system automatically generates descriptions of events that occur in the scene and stores them in an indexed database to allow the user to perform a rapid search.

Another middleware approach for video surveillance networks is proposed [98]. This surveillance middleware approach partitions systems based on an activity topology—a graph describing activity observed by the surveillance camera network. Processing within topological partitions uses well-known architectural styles such as blackboards, and pipes and filters. Communication between partitions uses a service-oriented architecture. This middleware enables building intelligent video surveillance systems at a far larger scale than was previously possible. Communication on the surveillance network follows the service-oriented model with publish/subscribe messaging, providing scalability, availability, and the ability to integrate separately developed surveillance services.

5.4 Summary

Middleware is a piece of reusable software that communicates to other processes, most of the time over a network connection. This is essential for IoT applications. In this chapter, a comprehensive overview and definition of middleware and their application/relevance to IoT is presented including general purpose, horizontal multitiered platform middleware, communication middleware, embedded middleware, LBS middleware, and others. A number of vertical application-oriented middleware and their relation to the four IoT pillars are also discussed.

Middleware makes more sense if based on standards, is vertical application agnostic, and can be used as a horizontal

platform serving many vertical applications. Two kinds of standardizations are important for middleware: standardized data representations and standardized architecture or frameworks. These two kinds of standards for IoT will be discussed in the next two chapters.

In the next chapter, data representations and protocols for the four IoT pillars and their unified standardization possibilities for a unified IoT middleware framework/architecture are discussed.