Middleware for Internet of Things: A Survey

Mohammad Abdur Razzaque, Member, IEEE, Marija Milojevic-Jevric, Andrei Palade, and Siobhán Clarke

Abhinav Prashant, Rohan Gupta
Contents

❖ IOT: Definition
❖ IOT: Characteristics
  ➢ Infrastructure perspective
  ➢ Application perspective
❖ Middleware: Introduction
❖ Middleware service requirement
  ➢ Functional
  ➢ Non-functional
❖ Middleware design approaches
  ➢ Event-based Middleware
  ➢ Service-oriented Middleware
  ➢ VM-based Middleware
  ➢ Agent-based Middleware
  ➢ Tuple-space Middleware
  ➢ Database-oriented Middleware
  ➢ Application Specific Middleware

❖ Challenges
  ➢ Functional
  ➢ Non-functional
  ➢ Architectural
❖ Future work
❖ Conclusion
Internet Of Things (IoT): Definition

RFID Community defines IoT as, “The worldwide network of interconnected objects uniquely addressable based on standard communication protocols”
Internet of Things: Characteristics

- Infrastructure perspective
- Application perspective
IOT characteristics: Infrastructure

- **Heterogeneous devices:**
  - Embedded and sensor computing nature of IOT devices; low cost computing platform are used
  - Higher-order computing devices to perform heavy duty task

- **Resource-constrained:**
  - Small device form factor for embedded computing and sensor devices
  - RFID have no processing capacity

- **Spontaneous interaction:**
  - Generation of events when object comes into communication range
  - Generated event is pushed to the system without human attention
IOT characteristics: Infrastructure

- **Ultra large scale network and large number of events**
- **Context aware:**
  - Large amount of data generated, will not have value unless analyzed
  - Context information related to the sensor data is stored
- **Intelligence:**
  - SOA components, virtual objects should be interoperable and independently act
- **Location aware:**
  - Interactions are highly dependent on the location, presence of other entities
- **IoT is inherently distributed**
IOT characteristics: Applications

- **Diverse application and real time**
  - Large number of applications in numerous domains
  - Transportation, healthcare, smart homes, industrial and personal

- **Everything-as-a-service(XaaS):**
  - Mode is scalable, efficient and easy to use
  - Services become accessible online and available to use and reuse

- **Increased security attack-surface and privacy leakage:**
  - Global connectivity means anyone can access services anytime
  - Tremendously increases the attack surfaces
  - Application may collect information about people’s daily activities
Middleware

- A middleware provides a layer between application software and system software
- Middleware abstracts the complexities of the system or hardware, allowing the application developer to focus all his effort on the task to be solved
- Requirements for a middleware to support the IoT, grouped as:
  - Services a middleware should provide
  - System architecture should support
Middleware Service Requirements

Categorised as:

- **Functional**: captures the services and functions (abstraction resource management)
- **Non-functional**: captures QoS support or performances issues
Functional requirements

❖ **Resource discovery:**
  ➢ Should be without human intervention, automated
  ➢ Every device should announce its presence and resources it offers

❖ **Resources management:**
  ➢ Resources usage should be monitored
  ➢ Resources allocated in fair manner and conflicts should be resolved

❖ **Data management:**
  ➢ Middleware needs to provide data acquisition and processing
  ➢ Data preprocessing which may include filtering, compression and aggregation of data
Functional requirements

- **Event management:**
  - Transforms simple observed events into meaningful events
  - Provide real time analysis of high velocity data

- **Code management:**
  - Code allocation and migration services are required
  - Code allocation selects set of devices/nodes to accomplish user/application level task
  - Code migration allows reprogramming nodes in the network
Non functional requirement

❖ **Scalability:**
   ➢ Needs to be scalable to accommodate increasing number of devices
   ➢ Loose coupling/virtualization is useful in improving scalability, by hiding the complexity of underlying hardware or service logic

❖ **Real time or timeliness:**
   ➢ On time delivery of services is critical in real time application

❖ **Reliability:**
   ➢ All components/service of middleware needs to be reliable in order to achieve overall reliability

❖ **Availability:**
   ➢ Services must be available or appear available at all times.
Non functional requirement

➢ If failure in system exists, recovery time and frequency must be small

❖ Security and Privacy:
➢ Security need to be considered in all functional and non-functional blocks
➢ Context awareness may disclose personal information

❖ Ease of Deployment:
➢ Complicated installation and setup procedure must be avoided

❖ Popularity:
➢ Should be continuously supported and extended
Architectural Requirements

❖ **Programming abstraction:**
  ➢ Provide an API for application developers

❖ **Interoperable:**
  ➢ Should work with heterogeneous devices/technologies/applications
  ➢ Can be viewed from network, syntactic and semantic perspectives

❖ **Service-based:**
  ➢ Provides abstraction for underlying complex hardware through a set of services
  ➢ Example includes data management, reliability, security

❖ **Adaptive:**
  ➢ Network, application level demands or contexts are likely to change frequently
Architectural Requirements

➢ Middleware needs to dynamically adapt to fit these variations

❖ **Context aware:**
  ➢ Middleware needs to be aware of contexts of users, devices and environment and use this for effective service delivery

❖ **Autonomous:**
  ➢ Devices/application should communicate without intervention
  ➢ Autonomous agents, embedded intelligence, predictive approaches in middleware can fulfill these requirements

❖ **Distributed:**
  ➢ Need to support functions that are distributed across the physical infrastructure. Centralized view will not be sufficient
Middleware: Existing work

Existing middleware solutions are grouped based on their design approaches:

1. Event-based
2. Service-oriented
3. VM-based
4. Agent-based
5. Tuple-spaces
6. Database-oriented
7. Application-specific
Event based Middleware

- Uses publish/subscribe pattern
- An event system (event service) may consist of a potentially large number of application components (entities) that produce and consume events.
- Message Oriented Middleware (MOM) is event-based middleware where communication relies on messages
- Events are broadcasted to all participants while messages are delivered by a particular subset of participants
Event Based: Examples

**Hermes**
- Created for large scale distributed applications
- Scalable routing algorithm and fault-tolerance mechanisms
- Also addresses interoperability and reliability requirements
- Middleware layer provides an API that programmers use to implement applications
Event Based: Examples

**SensorBus**

- A MOM for Wireless Sensor Networks (WSNs).
- Allows free exchange of more than one communication mechanism among sensor nodes.
- Architecture has three layers:
  - Application service layer: provides API for application development
  - Message service layer: responsible for communication
  - Context service layer: manages heterogeneous sensors
Event Based: Disadvantages

- Appropriate in systems where mobility and failures are common
- Interoperability, adaptability, timeliness, and context-awareness are not adequately addressed
- Event-based middlewares are also rarely autonomous
- Programming paradigm not sufficiently flexible
Service Oriented Middleware

- Design paradigm builds software or applications in the form of services
- Provides support for adaptive service compositions in absence of services
- SOM overcomes IoT ecosystem challenges by provisioning appropriate functionalities and accessing services at runtime
SOM: Examples

❖ **Hydra**
  ➢ Built on a SoA and model driven architecture
  ➢ Architecture consists of service, device, storage, context and security management components
  ➢ Supports dynamic reconfiguration and self-configuration
SOM: Examples

❖ Xively

➢ A PaaS that provides middleware services to create products and solutions for IoT
➢ Supports large-scale and real-time deployments in IoT
➢ Does not homogenise incoming data, so data preprocessing needs to be done individually for each sources
SOM: Disadvantages

- SOMs do not explicitly consider abstraction and code management.
- Most existing SOMs are WSNs-centric typically in the range of thousands, much less than the ultra large-scale (billions) of IoT.
- Resource discovery and management, and its composition mechanism will not scale well in dynamic IoT environment.
- SOMs offer only limited security through authentication.
VM-Based Middleware

- Provides safe execution environment for user applications by virtualizing the infrastructure.
- Applications divided into small modules, which are injected and distributed throughout the network.
- Each node in the network holds a VM, which interprets the modules.
VM-Based Middleware: Examples

❖ **Maté**
   ➢ Consists of resource-constrained sensor nodes
   ➢ Effectively handles resource management for sensor network and provides support for adaptability
   ➢ Not suitable for event based WSN applications, which require a non-blocking approach
   ➢ No support for reprogrammability after deployment
   ➢ Maté cannot run multiple applications concurrently in one node
VM-Based Middleware: Examples

❖ SwissQM
   ➢ Application Specific Virtual Machine (ASVM)
   ➢ Simplifies WSN programming by increasing the programming abstraction level
   ➢ Main design concern is to offer better support for data management
   ➢ Designed to support adaptability, resource management and code management
VM-Based Middleware: Disadvantages

- It is not a viable solution for supporting the heterogeneity of IoT infrastructure because it is heavyweight.

- Trading portability for performance reduces flexibility and the possibility of retasking.
Agent-Based Middleware

- Applications are divided into modular programs to facilitate injection and distribution through the network using mobile agents.
- While migrating from one node to another, agents maintain (as shown in Fig.) their execution state.
- Facilitates the design of decentralized systems capable of tolerating partial failures.
Agent-Based Middleware: Examples

❖ **Impala**
  ➢ Enables application modularity, adaptivity, and repairability in WSNs
  ➢ Requirements are supported by switching between different protocols and modes of operation depending on the applications and network conditions
  ➢ Does not support data preprocessing, which is an important component of data management
Agent-Based Middleware: Examples

❖ Ubiware

➢ Agent is distributed over three layers:
  ■ A behavior engine layer implemented in Java
  ■ A declarative middle-layer
  ■ Third layer contains shared and reusable resources interpreted as Java components

➢ Adds security policies to support the security requirement
➢ It assign a proactive agent to each of the resources
Agent-Based Middleware: Disadvantages

- Do not address the heterogeneity of an IoT infrastructure; only designed for WSNs or mobile devices.
- Hard real-time guarantees is not addressed
- Security and privacy are generally not considered
- The autonomous characteristic of agents can lead to unpredictability in the system at runtime
- Mobile agents are susceptible to message loss, especially in resource-constrained environments
Tuple-Space Middleware

- Each member of the infrastructure holds a local tuple space structure.
- A tuple space is a data repository that can be accessed concurrently.
- All the tuple spaces form a federated tuple space on a gateway.
- Applications communicate by writing tuples in a federated tuple space, and reading them by specifying the pattern of the data they are interested in.
Tuple-Space Middleware: Examples

❖ **LIME**
  ➢ Developed to address mobile devices’ energy limitations
  ➢ Centralized tuple space is broken down and permanently attached to a mobile component
  ➢ Access to the tuple space is carried out using an extended set of tuple space operations
  ➢ Supports good programming abstractions for exploiting a dynamically changing context
Tuple-Space Middleware: Examples

❖ TS-Mid
   ➢ Deploys an asynchronous and decoupled communication style in both time and space.
   ➢ Broken into:
     ■ Leader node: responsible for data aggregation and forwarding to sink
     ■ Sink node: Queried by clients.
   ➢ It supports data management through data aggregation and storage.
Tuple-Space Middleware: Disadvantages

- Currently tuple-spaces middleware solutions have been designed only for WSNs or mobile devices
- Cross-layer design overhead is prohibitive in the IoT
- Not reprogrammable and provide limited support for adaptability and scalability
Network is viewed as a virtual relational database system. An application can query the database using an SQL-like query language, which enables the formulation of complex queries. Focused on developing a distributed database approach to interoperating systems.
Database-Oriented Middlewares: Examples

❖ **SINA**
  ➢ Handles events and can also cope with the mobility of the querying (sink) node.
  ➢ Allows sensor applications to issue queries and command tasks
  ➢ SINA modules, running on each sensor node, provide adaptive organization of sensor information, and facilitate query
  ➢ SINA is not secured or private.
IrisNet

- Deploys heterogeneous services on WSNs
- Supports the control of a global, wide-area sensor network by performing queries on this infrastructure
- Each query operates over data collected from the global sensor network and supports simple and more complex queries
Database-Oriented: Disadvantages

- Only approximate results are returned for query in database approach to middleware
- Database middleware approach uses a centralized model, which makes it difficult to handle large-scale sensor networks dynamics
- Hard real-time applications are not supported
- Not suitable for safety critical systems
Application-Specific Middlewares

- It focuses on resource management support for a specific application or application domain.
- It fine-tunes the network or infrastructure based on the application or application domain requirements.
MiLAN

- Proactive adaptation to respond to application needs
- Allows applications to specify their QoS requirements and adjust the network configuration at runtime
- Adjustments are made based on information collected from the application, the user, the network, and the overall system
Application-Specific: Disadvantages

- Do not address the heterogeneity of an IoT infrastructure as there is tight coupling between applications and middleware layer
- Does not provide a general purpose solution
- It uses a centralized resource discovery mechanism, which is not a viable approach for a distributed fault-tolerant IoT solution
Challenges

- Functional Requirements
- Non Functional Requirements
- Architectural Requirements
Challenges: Functional requirements

❖ **Resource discovery:**
  ➢ Ultra-large-scale of IoT invalidates centralized resource registries and discovery approaches

❖ **Resource management:**
  ➢ Frequent resource conflicts occurs which are not addressed by most existing middleware solution

❖ **Data management:**
  ➢ Most middleware offer support for data aggregation but not for data filtering and compression for vast amount of data generated

❖ **Event Management:**
  ➢ Event management components may become bottleneck in the system
Challenges: Non Functional requirements

- **Real time:**
  - Some middleware are by nature not real time (e.g., database or tuple-space), while the rest provide real-time services.

- **Reliability and Availability:**
  - Hardware devices fail periodically and the service they provide will be unavailable and reduce reliability of system.

- **Security and privacy:**
  - Most middleware solutions offer authentication based partial security.
Challenges: Architectural requirements

❖ **Programming abstraction:**
  ➢ Steep learning curve for developers related to new languages and tools

❖ **Service-based:**
  ➢ Service description needs to be comprehensive but energy efficient to become suitable for resource constrained devices

❖ **Adaptive:**
  ➢ Rules, policies and QoS definitions are hard-coded and not context aware
  ➢ Only application-specific middleware adapt to QoS requirement
Current research and future work

- Dynamic resource discovery, Scalability, reliability, context-awareness, security and privacy with IoT middleware

- Most current middlewares address WSNs, while other perspectives (e.g., M2M, RFID, and SCADA) are rarely addressed
Conclusion

- Middleware is necessary to ease development of diverse applications and services in IoT
- Proposals discussed are diverse and involve various middleware design approach and support different requirements
- Existing middleware solutions have not explored some requirements such as dynamic resource discovery, scalability, reliability, security, privacy and context-awareness
- Significant scope for future work
Thank you!