International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

Self-configuring, modular, ETSI-compliant M2M service platform

BADAVATH SRIN IVAS NAIK , VUDARU SHIVA , BASAWA SURYA VAMSI KRISHNA
SUPERVISOR , M RAMBABU
Associate Professor
ANURAG ENGINEERING COLLEGE
AUTONOMOUS
(Affiliated to JNTU-Hyderabad,Aprroved by AICTE-New Delhi)
ANANTHAGIRI (V) (M), SURYAPETA (D), TELANGANA-508206

Abstract

The M2M idea, in which machines communicate with one another, is a major part of the IoT. In the not-too-distant future, it will link billions of devices across several fields. However, M2M is hindered by the absence of standards and the extreme vertical fragmentation of existing M2M marketplaces. In order to fill this need, the European Telecommunications The European Telecommunications Standards Institute (ETSI) has published a standard platform for M2M services. In this work, we suggest the OM2M project, an autonomous ETSI-compliant M2M service platform that is available under an open source license. To promote compatibility across different systems, OM2M offers a Restful API. It presents a plug-in-friendly modular architecture deployed on top of an OSGi layer. It allows for the binding of various communication protocols, the re-use of pre-existing remote device management systems, and the integration with older devices. A novel M2M service is designed using the autonomic computing paradigm and semantic models to provide dynamic discovery and reconfiguration processes, thereby solving the M2M complexity problem.

1. Introduction

The phenomena of machine-to-machine (M2M) communication have been developing in the background, and it is now entering the stage when an explosion of use scenarios in enterprises is expected to take place. It is now possible for devices such as sensors, actuators, RFID/NFC tags, automobiles, and smart machines to exchange data with one another. There has been a steady rise in the number of machine-to-machine connections, and it is expected that eventually billions of devices will be networked together. Advantages of M2M applications range from construction and energy management to healthcare and industry to transportation and commerce to retail and safety and environmental services. The M2M industry is experiencing a radical transformation, moving away from a series of vendor-specific, closed vertical markets and toward a unified, worldwide horizontal M2M platform. Large-scale M2M deployment is made possible by this revolution because it allows for smooth interactions between smart apps and heterogeneous devices.

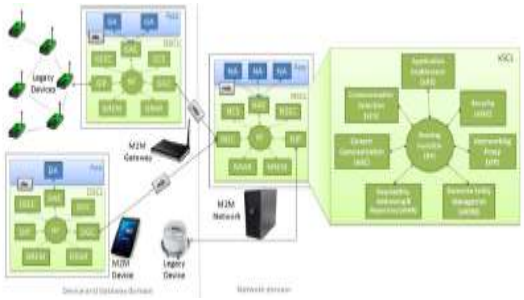
Challenges to the M2M idea include the absence of interoperability, the growing complexity of dispersed systems, and the absence of standards. The problem of M2M interoperability has prompted many attempts at standardization [1]. In terms of M2M protocols, the European Telecommunications Standards Institute (ETSI) provides the most cutting-edge specification. M2M service needs, functional architecture, communication interfaces, and interoperability with current standards and technologies are all topics that have been addressed in recent ETSI specifications [2, 3, and 4]. The ever-increasing complexity of supercomputers was a problem that was tackled by some other group.
IBM introduced the autonomic computing (AC) paradigm, which takes cues from the body's own autonomic nervous system. The goal of AC is to provide systems the ability to self-manage in order to shield administrators and users from the system's inherent complexity. In order to facilitate communication between M2M devices, we offer

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

the OM2M project, which is an ETSI-compliant platform. OM2M suggests an Osage layer with a Restful design. It supports a wide range of protocols and technologies and may be expanded with the help of plugging. To further facilitate M2M application dynamic discovery and self-configuration, a semantic-based autonomic computing-based service has been developed. The remaining sections of the paper are as follows: The ETSI M2M standard is introduced in Section 2. In Section 3, we provide the UML diagrams for the core components and plugging of the OM2M platform.
In Section 4, we will discuss how the autonomic computing service may be used to automatically configure OM2M. The last section offers some final thoughts and observations.

2. ETSI M2M standardization

The Global Standards Collaboration Machine-to-Machine Task Force reports that there are over 140 groups actively working on M2M standards development throughout the globe. By developing a horizontal service platform for M2M, ETSI is working hard to reduce market fragmentation. To facilitate service and application development that is not reliant on the underlying network, the proposed approach offers a Restful Service Capability Layer (SCL) [3] that can be accessed using open APIs. A platform like this makes it easier to roll out industry-specific apps and encourages creative thinking across all sectors to improve interoperability.



ETSI M2M Functional Architecture, Figure 1

The SCL is used to power applications on an M2M Device. Using the Access network, it communicates directly with the Network Domain and may potentially serve undetectable peripherals. Internetwork Space. In addition, a Gateway across a LAN may link it to the Network Domain. In addition to acting as a proxy for local devices connecting to the network domain, a Gateway may also execute SCL-based M2M applications. The access network facilitates interaction between the Core Network and the M2M devices and gateways. Multiple Applications may make use of the same set of features thanks to the SCL. Access and Core Networks may be managed with the use of Network Management Features like Provisioning, Supervision, and Fault Management. The M2M Management Functions provide everything needed to control the SCL in the Network Domain.
ETSI's M2M functional architecture is shown in Figure 1. Any of an M2M network's nodes, a gateway, or a device may host a SCL. To facilitate machine registration, synchronous and asynchronous communication, resource discovery, access rights management, group broadcast, etc., the service offers a number of functions. There are three benchmarks [4] defined by open APIs: mIa, dIa, and mId. Through the mIa point of reference, a Network Application (NA) may connect to the NSCL. Access to the D/GSCL is granted through the dIa to any Device or Gateway Application (D/GA). A D/GSCL may connect to the NSCL using the mId reference point. These interfaces are designed in a broad enough sense to function with a variety of network technologies and protocols, boosting compatibility between them. The ETSI M2M standard has a Restful design. The data is organized in a uniform

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

resource tree that is part of every SCL. A resource is anything that can be located using a specific identifier (a URI) and then moved around and changed using verbs (like "get," "put," "delete," and "execute").

3. OM2M service platform

OM2M building blocks

OM2M is an ETSI-compliant M2M service platform that we propose in this research. In a highly scattered setting, OM2M offers a Restful API for exchanging XML data across shaky connections. As may be seen in [5,] it provides a modular framework atop the OSGi Equinox runtime. OM2M's SCL is versatile and may be used in any M2M network, gateway, or endpoint (see Figure 2). Each plug-in in a SCL is a tiny, closely connected module that provides a unique set of capabilities. There is no need to reboot the computer in order to install, launch, stop, update, or delete a plug-in. In addition, it monitors the service register for changes that may affect the SCL extension and adjusts appropriately.

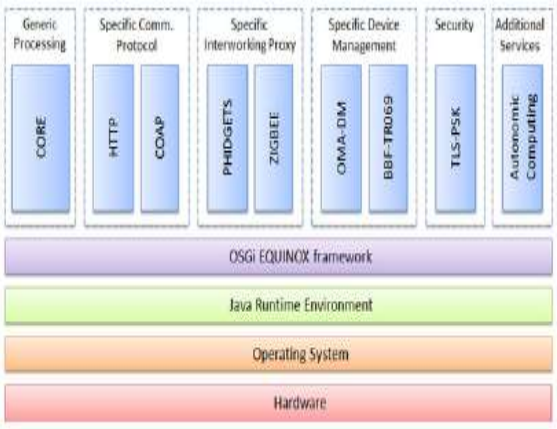


Figure 2: The Backbone of OM2M

The CORE is the primary plug-in that has to be installed in every SCL. It's a protocol-agnostic service for dealing with requests sent over the Restful interface. Custom communication mapping plug-ins may be installed to multi-protocol support with HTTP and CoAP bindings. By leveraging existing protocols like OMA-DM and BBF TR-069, OM2M may be enhanced with specialized device management mapping plugging to update device firmware. It may be enhanced with additional proxy plugging for interoperability, allowing it to communicate without any hitches with older technologies like Zigbee and Phi gets. For secure pre-shared-key M2M communications, the TLS-PSK protocol is utilized. To improve OM2M resource discovery and self-configuration, a new plug-in has been developed based on the autonomic computing paradigm.

OM2M plugging components diagram

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

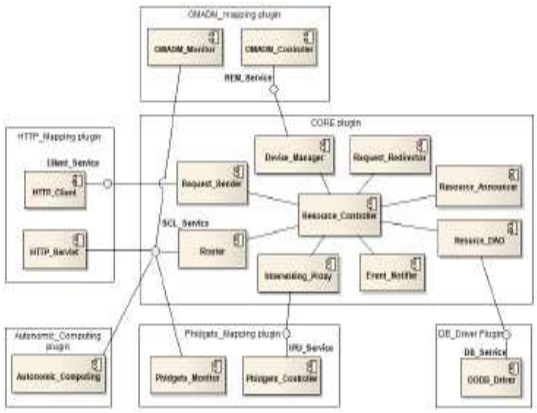


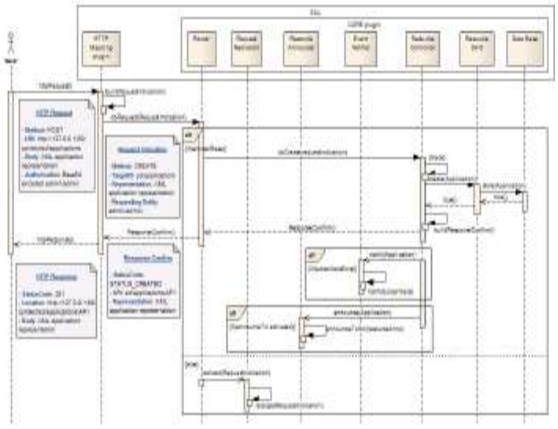
Figure 3: Component schematics for OM2M

The general Restful requests are handled by the CORE plug-in, which implements the SCL_Service interface. It takes in a generic request indication and returns an equally generic confirm. The With only the request URI and method, a router may build a single route to process all requests made to a resource controller. Each resource has its own set of CRUD (Create, Read, Update, and Delete) methods that are implemented by the Resource Controller. It validates things like syntax in resources and ensures that users have the proper permissions to access them. Without disclosing database specifics, the Resource DAO offers an abstract interface to encapsulate all access to resource persistent storage. When a resource is created, changed, or removed, the Event_Notifier will notify any subscribers who have asked to be notified. It filters out irrelevant occurrences so that subscribers only get relevant information. In order to increase the resource's discoverability and accessibility, the Resource Announcer broadcasts it to a distant SCL. It also takes care of announcing resources. Clients for a certain protocol that have been found to implement the Client Service interface are stored in the Request Sender. It proxies requests and ensures that they are sent using the appropriate protocol. To make sure the right IPU controller is contacted, the Interworking Proxy stores all IPUs that have been detected and are using the IPU_Service interface. Remote Entity Managers (REMs) that implement the REM_Service interface are stored in Device Manager, which then proxy's calls to the appropriate device manager controller. The HTTP Mapping plug-in gives you a way to connect to the HTTP protocol in both directions. When an HTTP request is received, the HTTP_Servlet modifies it into a generic request and then calls the CORE plug-in via the SCL_Service interface. The HTTP Client uses the Client Service interface to issue a broad request over the HTTP protocol. To facilitate communication with older Phi gets devices, the Phidgets_Mapping plug-in offers a two-way mapping. After locating a Phi gets device, the Phidgets_Monitor contacts the CORE plugin to generate the necessary resources on the SCL. To make it easy to make a general request using the Phidgets API, the Phidgets_Controller implements the IPU_Service interface. The OMADM_Mapping plug-in allows for the management of devices that are OMA-DM enabled through a bidirectional mapping. In order to establish the necessary resources on the SCL, the OMADM_Monitor listens for OMA-DM enabled devices and then invokes the CORE plugin. When a generic request is received, the OMADM_Controller uses the REM_Service interface to transform it into an OMA-DM management session.

The DB_Driver plugin offers an OODB that may be accessed using the DB_Service plugin. The same method may be used to roll out other plugins like the Autonomic Computing plug-in, which adds support for new protocols and features.

Application resource creation scenario

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86



Case Study on Resource Generation

An issuer creates an Application resource by sending a POST HTTP request to this URI, with the application's XML data in the body and the issuer's username and password in the authorization header. http://ip:port/context/scl/applications.

The HTTP Mapping plug-in converts the incoming request into a Router-friendly Request Indication object. If the target SCL is the same as the sclBase, the request is sent locally to the Application Controller; otherwise, the Router retargets the request to the appropriate remote SCL. The Application Controller verifies the representation of the application and the issuer's permission to issue it. From the incoming XML representation, it constructs an Application object and persists it to the database using the Application DAO. Then, the Resource Announcer broadcasts the application's available resources to the remote SCLs that were specified in the Event_Notifier's input.

Adaptation to a wide range of M2M solutions using compatible proxy plugging

The issuer in the following scenario will solely use REST requests to manage devices of varying types. It keeps an eye on a Phi gets temperature sensor and uses HTTP to command a Zigbee fan actuator. After locating the temperature sensor, the Phidgets_IPU plugin sets up the necessary resources (TEM_APP application resource and TEM_DATA container sub-resource) to log data. The Zigbee_IPU plugin is similar in that it adds a FAN_APP resource to the SCL. The provider first makes a GET HTTP request to find registered apps, and then makes a POST HTTP request to subscribe to TEM_DATA content Instances. Phidgets_IPU initiates a new TEM_DATA content Instance sub-resource whenever the temperature sensor reports an event. An HTTP notice is sent to the issuer. When the inside temperature becomes too high, a POST HTTP request is sent to turn on the fan. When a request is received, the Zigbee_IPU plugin processes it according to the Zigbee protocol and sends the result back.

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

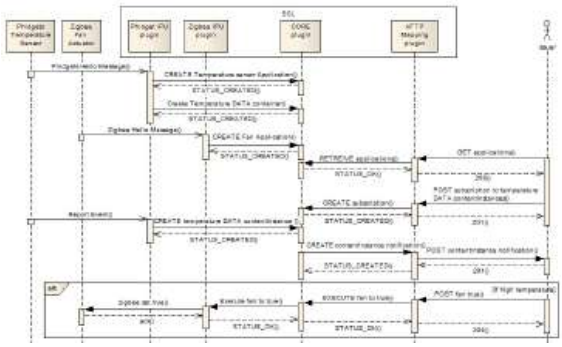
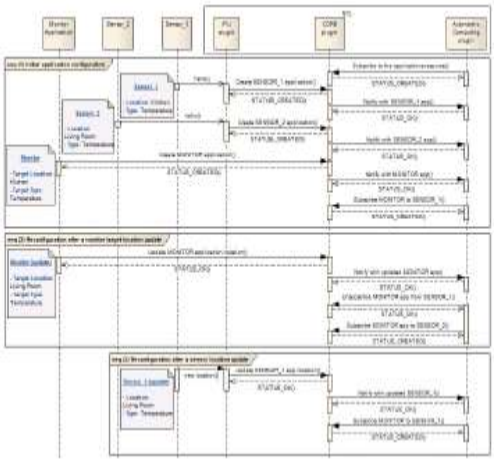


Figure 5: Multiple M2M interoperability proxies in operation. Specific communication mapping plugins may be installed to improve compatibility with different M2M communication protocols. For instance, you might insert a CoAP_Mapping plugin in such a manner that a request from an issuer may traverse numerous HTTP and CoAP equipped computers without interruption.

4. M2M self-configuration via an Autonomic Computing plugin

IBM suggested the concept of "Autonomic Computing" [6, 7] in 2001. The goal is to create self-managing distributed systems that administrators and users never have to see. The four basic components of an autonomous manager are the observer, the analyst, the planner, and the actor. These modules collaborate in an active control loop dubbed MAPE-K, in which they pool their knowledge (details of controlled resources, policies, symptoms, requests for change, plans, etc.) and use policies based on goal and environment awareness. To allow M2M communications to auto-configure them based on device profiles and application responsibilities, an autonomic computing service (ACS) is developed and deployed. The suggested ACS offers an expert-like autonomic manager to mimic human judgment in solving complicated issues via the use of knowledge-based reasoning. We propose integrating the ACS into the SCL as a service capability. To begin, the ACS will sign up for updates from the applications resource so it can keep tabs on any freshly registered apps. When a new application is made, the ACS is alerted and given a representation of the application's resources, such as its kind, category, location, etc. When an application is found by the ACS, it is added to an ontology model in the system's local knowledge base. It uses semantic reasoning to find pre-existing app compatibility [8, 9]. It then modifies the SCL resource tree to link the necessary programs together. Three simple examples of self-reconfiguration are shown in Figure 6. As shown in Figure 7, the ACS takes into account an ontology model in order to carry out the suggested self-configuration scenario, and the JESS engine uses SWRL rules [10] in order to pair up pre-existing monitor and controller managers with suitable sensor and controller devices inside the M2M system.

International journal of basic and applied research
www.pragatipublication.com
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86



Sequence schematic for self-configuring logic gates

One such setup is shown in sequence (1). When the apps resource is subscribed to, the ACS monitoring module gets the following events: Sensor_1_Event (Kitchen) (Event_1) Event_1 (Id=Sensor_1, Location: Kitchen, Target Category=Temperature), Event_2 (Id=Sensor_2, Location: LivingRoom, Category: Temperature), and Event_3 (Id=Monitor_1, TargetLocation: Kitchen, Target Category=Temperature). The M2M ontology model shown in Figure 7 is updated by the ACS monitoring module, which then infers and communicates the following symptoms to the analyzing module: Type: NewSensor, Id: Sensor_1, Type: NewSensor, Id: Sensor_2 and Type: NewMonitor, Id: Monitor_3 are all symptoms. In order to determine whether or not two applications are related, the ACS analyzer uses the two SWRL rules shown in Figure 7. Based on the data provided by the user, it develops and submits the following Request for modification to the planning module after determining that there may be a connection between Monitor_1 and Sensor_1. Goal-Oriented Subscription to RFC_1; Monitor_1 as Subscriber; Sensor_1 as Publisher. The following step is included in the RFC_1-based action plan that is generated by the planning module and sent to the executer: Subscription resource comprising Monitor_1 Contact URI, Action_1 (Method: CREATE, TargetID: Sensor_1). Action_1 is converted into an HTTP request by the executer module, which is then sent to the SCL as follows: HTTP Request (Method: POST, URI: Sensor_1 subscriptions URI, Body: Subscription XML incorporating Monitor_1 contact URI). After the subscription resource has been setup, Sensor_1 will begin sending temperature events for the kitchen to Monitor_1.

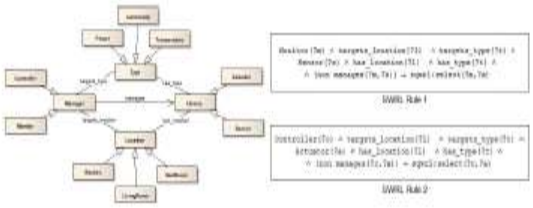


Fig. 7. M2M ontology model and SWRL rules

5. Conclusion

Herein, we introduced OM2M, an open-source M2M service platform compliant with the ETSI M2M standard. The suggested system makes use of a pluggable OSGi architecture and exposes services via a Restful API to facilitate M2M communication. OM2M architecture, parts list, and network diagram A new autonomic service for

dynamically discovering and configuring M2M apps was introduced, and the plugging' primary functions were described. After OM2M was tested in a mock-up building, it was implemented in a real-world smart building and verified using the LAAS ADREAM platform. The Eclipse foundation has approved the OM2M platform as an open source initiative. The Eclipse Internet of Things (IoT) Working Group has adopted it [11]. Future plans include expanding OM2M's functionality to include mapping for resource-constrained internet devices-friendly protocols like CoAP, MQTT, and Lightweight M2M. With the SSN ontology [12], the autonomic computing service will be able to tackle increasingly complicated use cases. To further improve scalability, we are developing a new M2M service that enables information-centric routing [13].

References

[1] Boswarthick D, Elloumi O, Hersent O. M2M Communications: A Systems Approach. Wiley: Chichester, U.K., 2012. Print.
[2] ETSI TS 102.689 v1.1.1. Machine-to-Machine communications (M2M); M2M service requirements. August 2010.
[3] ETSI TS 102.690 v1.1.1. Machine-to-Machine communications (M2M); Functional architecture. October 2011.
[4] ETSI TS 102 921 v1.1.1. Machine-to-Machine communications (M2M); mIa, dIa and mId interfaces. February 2012.
[5] McAffer J, VanderLei P, Archer S. OSGi and Equinox: Creating Highly Modular Java Systems. Addison-Wesley; Upper Saddle River; NJ; 2010.
[6] Parashar M, Hariri S. Autonomic Computing: Concepts, Infrastructure, and Applications. CRC/Taylor and Francis: Boca Raton; 2007. Print.
[7] Kephart JO, Chess DM. The vision of autonomic computing. Computer 2003; 36(1):41–50. DOI: 10.1109/MC. 003.1160055.4. Dobson S, Sterritt R.