Distributed Device Management for IoT

In action with Eclipse Leshan, Eclipse Wakaama and OMA-LWM2M

Orange Labs
IoT Research Domain

19/02/2019 Eclipse IoT Days, Grenoble

Speakers:
- BERLEMONT Samuel
- MICHE Arnaud
Outline

1. Device Management?

2. Standards and best practices

3. DM for IoT, a new paradigm

4. The Future of DM: a multi-server architecture

5. Next Steps
Device Management... ?
4 categories of operations

- Remote service activation and configuration
- Error and log collection for analysis
- Configuration visualization
- Diagnostics
- Remote actions (reboot...)
- Firmware update
Standards and Best Practices
CPE WAN Management Protocol v1.4, Issue: 1 amendement 6

Centralized Auto-Configuration Server

Data models

RPC Methods device / server

(HTTP/SOAP)
User Services Platform
V1.0

Multi-controller architecture

Data models w/ methods
RESTful
Full discovery

Service elements / Proxies

Start of Open Source activities
OMA Lightweight M2M

Multi-server architecture

Data models w/ methods
  RESTful
  Shared models

Service objects (OMNA, IPSO) / Proxies

Dynamic ecosystem
  • Open Source implementations (Wakaama, Leshan, …)
OMA Lightweight M2M: Device Management & Service Enablement Standard for IoT

Device Management

Bootstrapping

Device Configuration

Firmware Update

Fault Management

Service Enablement (Application)

Configuration & Control

Reporting
DM for IoT, a new paradigm
IoT DM challenges

• Heterogeneity
  ➔ life-cycle profiles
  ➔ DM features

• Security

• New architectures
  ➔ DM/service convergence
  ➔ Multi-protocol management
  ➔ Softwarization : NFV, SDN

• Scalability
The impact on DM solutions

Device profiles

Requirements

Corresponding DM/SM protocol

Multiple relevant Standards & Proprietary protocols
The impact on DM solutions

Multi-protocol DM solution

Integration of additional DM servers

Integration of non-DM-enabled devices
Distributed Device Management for IoT

In action with Eclipse Leshan, Eclipse Wakaama and OMA-LWM2M

Imagined by the research team working for Orange Labs IoT Research Domain

« Under the hood »
Who am I?

Arnaud MICHE

Software developer at OBS SA (subsidiary of Orange SA)

My job on this project:
Implement prototypes in order to evaluate the ideas and theories envisioned by the research team.
Contents

• Why Leshan?
• How we used it
• Proxies
  • How a proxy works
  • Changes in leshan-client-demo
• Servers
  • How a server works
  • Changes in leshan-server-cluster
• Wakaama on ESP32
• How it is integrated in the demonstration
• Near future work
Why?

1. Major goals
   - Decentralized Multi server
   - LwM2M seems to be a good candidate for this job

2. Need of an implementation for our experiments

3. Leshan is mature, in active development and provides good examples ...
   - We can start our prototype with confidence
Contents

- Why Leshan?
- How we used it
- Proxies
- Servers
- Wakaama on ESP32
- How it is integrated in the demonstration
- Near future work
How we use

Disclaimer:
The code base used for our prototype has been cloned in September 2017 and did not follow the changes of upstream developments since this date.

1. As a server of Device Management (DM) ready to be clustered in our network.

2. As a proxy for connecting constrained devices to our DM network.
How we use

3. For now, only Firmware Upgrade is implemented in the Proof-of-Concept.
Contents

- Why Leshan?
- How we used it
- **Proxies**
- Servers
- Wakaama on ESP32
- How it is integrated in the demonstration
- Near future work
Proxies

- Based on **leshan-client-demo** package provided with Leshan sources
- Addition of an **Avahi** service for device over TCP (here, via WiFi)
- Addition of a service polling serial connections for Bluetooth device connected via an USB dongle
- Process wrapper for launching tools for flashing device
How a proxy manager works

New LWM2M client: g-proxy-<id>/g-esp32-<id>

New LWM2M client: g-proxy-<id>/g-airboard-<id>
Changes on **leshan-client-demo**

**From the package leshan.client.resource**

- **BaseInstanceEnabler**
  - **MyDevice**
    - **SerialDevice**
    - **AvahiDevice**
  - **AbstractDevice**

*In order to address several types of devices. Here, BT and WiFi devices*
Changes on `leshan-client-demo`

**In order to connect WiFi and BT devices**

- AvahiMonitor
  - libavahi4j
  - TCP connection monitoring service
- BlunoMonitor
  - Serial connection monitoring service

- WiFi connection
- USB dongle connection

LWM2M client creation
LWM2M registration
Changes on *leshan-client-demo*

**In order to perform the firmware update (1/2)**

- **Executor**
  - A kind of Process wrapper

- **JavaExecutor**
  - Execute a JAR

- **LoaderExecutor**
  - Launch the tool for flashing a binary file to the esp32

- **ArduinoExecutor**
  - Launch an Arduino build for a given INO file (available for a zipped INO project, also)

- **AvrLoaderExecutor**
  - Launch avrdude for flashing a binary file on an Avr device
Changes on leshan-client-demo

In order to perform the firmware update (2/2)

- The LWM2M object « Firmware »
- It is also in charge of downloading the firmware and launch the execution of the flash tool (by intermediate of the Executors)

- It holds informations necessary to the firmware update:
  - Version of software
  - url where to download the file
  - format of the file and the type of the board enabling the choice of the right executor for flashing the device.
Changes on leshan-client-demo

**In order to allow multiserver (1/2)**

Declaration of two server URLs inside LeshanClientDemo.java:

1. 
   ```java
   private static String serverURI_1 = "coap://192.168.0.100:5683";
   private static String serverURI_2 = "coap://192.168.0.101:5683";
   ```

2. 
   ```java
   Security sec1 = Security.noSec(serverURI_1, 123);
   Security sec2 = Security.noSec(serverURI_2, 234);
   Server serv1 = new Server(123, 30, BindingMode.U, false);
   Server serv2 = new Server(234, 30, BindingMode.U, false);
   initializer.setInstancesForObject(SECURITY, sec1, sec2);
   initializer.setInstancesForObject(SERVER, serv1, serv2);
   ```
Changes on leshan-client-demo

In order to allow multiserver (2/2)

Some changes in LWM2M Registration related classes of Leshan core:

1. Inside leshan-master_bluno/leshan-client-core/src/main/java/org/eclipse/leshan/client/servers/RegistrationEngine.java

   Changed data structure which holds the registration ID for one client from a variable storing the reg_id to a hash map which holds several couples (server_uri, reg_id) as a client can be registered to more than one server.

2. Inside leshan-client-cf/src/main/java/org/eclipse/leshan/client/californium/LeshanClient.java

   Added a function getRegistrationId which calls the getRegistrationId of the RegistrationEngine with the server informations. Server informations passed in parameters enable to retrieve the registration Id of a device giving its server Id.
Contents

- Why Leshan?
- How we used it
- Proxies
- Servers
- Wakaama on ESP32
- How it is integrated in the demonstration
- Near future work
Servers

- Based on `leshan-server-cluster` package provided with Leshan sources
- Manage registered devices
- Relying on Redis PubSub
- Reg IDs stored in Redis Key/Value data base
How a server works

Server start!

Client registered!

Store server Id

Redis Pub/Sub

Same path for update and deregistration of the device.

Store regId and registration info

Same path for the server stop event.
Note: a hook has been added for notifying the stop of the server.
How a server works

Launch Update with `<firmware_url>`!

Get the IP address of the device from its `regId`!

Server can use the Pub/Sub bus to notify the progress of operations

FirmwareUpdate on device `<Id>`!

Redis Pub/Sub

Redis DB
Changes on `leshan-server-cluster`

In order to allow the detection of arrivals and exits of servers in the cluster.

Creation of the class `leshan-server-cluster/src/main/java/org/eclipse/leshan/server/cluster/RedisServerEventPublisher.java`:

- Publish start/stop event of a server
- Store Server ID inside database

Creation of a new key/value in the store:

{ `LESHAN_SERVER_ID:<server_id>` ; `<server_addr>, <secured_server_addr>` }
Changes on leshan-server-cluster

In order to allow multiserver

Modification of the data structure stored in Redis Database:

```
Redis DB
{ REG:EP:<endpoint> ; <serialized_registration_data> }
```

And token handlers hold now, the regID in addition of endpoint name:

Before token handlers were `EP#UID#endpoint` and now it is `EP#UID#regId#endpoint`

Finally following classes have been impacted:
- `leshan-server-cluster/src/main/java/org/eclipse/leshan/server/cluster/LeshanClusterServer.java`
- `leshan-server-cluster/src/main/java/org/eclipse/leshan/server/cluster/RedisRegistrationStore.java`
- `leshan-server-cluster/src/main/java/org/eclipse/leshan/server/cluster/RedisRequestResponseHandler.java`
- `leshan-server-cluster/src/main/java/org/eclipse/leshan/server/cluster/RedisTokenHandler.java`
Contents

- Why Leshan?
- How we used it
- Proxies
- Servers
- **Wakaama on ESP32**
- How it is integrated in the demonstration
- Near future work
Wakaama on ESP32

In order to allow firmware update on ESP32 without the help of the proxy

Based on wakaama client example distributed with Wakaama sources

Using the ESP32 SDK (with FreeRTOS)

RGB LED controller
To light on/off an RGB LED

HTTP Downloader
To download the firmware on the board

OTA
To process the firmware update
Contents

- Why Leshan?
- How we used it
- Proxies
- Servers
- Wakaama on ESP32
- How it is integrated in the demonstration
- Near future work
How it is integrated in the demonstration

- Device LWM2M compatible
- DM Server
- Proxy
- Device
- Bus PUB/SUB
  - Wifi (detected using Avahi)
  - Ou BT (via serial stack of Linux)
- Redis connector
- DM Network
- Socket.io (/hmi)
- Socket.io (/leshan)
- dm_network.json
- firmwares
- Flask
- NetworkX
- DM Network

- DB
Contents

- Why Leshan?
- How we used it
- Proxies
  - How a proxy work
  - Changes in leshan-client-demo
- Servers
  - How a server works
  - Changes in leshan-server-cluster
- Wakaama on ESP32
- How it is integrated in the demonstration
- Near future work
Near future work

- Using the acquired experience to clean the architecture (micro-services?)
- Implement a generic layer to address other protocols (not only LWM2M)
- Synchronize with upstream Leshan code.
Thanks
Multi-server demo architecture
Externe Orange

ESP32 DM API

Message bus

DM Dashboard

Airboard

ESP32

Extensive Orange
Time for the demo!
Conclusion

New solutions
- Multi-server, multi-protocol architecture
- Integration of multiple and new DM servers
- A need for abstraction

Next Steps with Eclipse IoT
- Contributions
  - Leshan
  - Wakaama + ESP32
- Study of integration
  - Hono
  - hawkBit
Thank you