POLITÉCNICO DE LEIRIA
ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
www.dei.estg.ipleiria.pt

Integração de Sistemas

3º ano • 1° semestre • Ano Letivo 2022/2023

Project

SOMIOD: Service Oriented Middleware for Interoperability and Open Data

What is the project about?

The Internet of Things (IoT) concept is pointed as a solution (or at least as a special contribution) for a myriad of societal problems including the most recent ones dealing with climate changes, energy production and consumption and sustainability in general, just to name a few. From our (smart) homes and offices to the domain of (smart) cities the data seamlessly collected will benefit society in general, the business community as well as local and state governments where new applications and innovative services can be developed towards social cohesion, a better environment and economy and a smarter society.
However, nowadays, a major part of IoT proposed and already deployed IoT solutions rely on private protocols and private cloud services which have a huge negative impact on interoperability and data sharing across different devices, applications, and platforms what several researchers call "SILO OF THINGS" such as in [1].

The aim of this project is to create a service-oriented middleware able to define an uniformization into the way data is accessed, written, and notified, independent of the application domain, to promote interoperability and open data (data is always accessed and written in the same way and applications are always created in the same way. Hence, anyone can access data and extend or create new applications and services always following the same format). To achieve that, it is proposed the adoption of Web Services and a Web-based resource structure, always based on the Web open standards.

More concretely, it is proposed the following resources, features, and capabilities:

- The Web service URL must always start by the following structure:

http://<domain:port>/api/somiod/...

- The middleware must support the following resources following the presented hierarchy (with a maximum depth of 3 levels):

```
application (A)
   module (1)
      data (1..N)
      subscriptions (0..N)
   module (N)
     data (1..N)
     subscriptions (0..N)
application (B)
   module (1)
      data (1..N)
      subscriptions (0..N)
   module (N)
      data (1..N)
      Subscriptions (0..N)
```

- An application resource represents a specific application. Middleware should support multiple applications.

- A module resource represents one application module. An application can have one or more modules.
- Data represents each data record created on a specific module.
- A subscription resource represents the mechanism able to fire notifications (e.g., a new data record was added to the module, or a data record was deleted from the module).
- Each resource above should have a unique ID, a name, a creation date, and parent if applicable. More information about this later on this document.
- Through the REST API, it must be possible to create, modify, list, and delete each available resource. Data resources (records) and subscription resources only allow creation and deletion.
- Subscriptions, besides ID, name, creation data and parent, should support two types of events: creation or deletion or both, and an endpoint where the notification should be fired. For each notification fired, the data resource (created or deleted) must be part of the notification as well as the type of fired event (creation, deletion). For now, subscriptions only accept MQTT endpoints. The channel name used to fire the notification should have the same name of the source module resource.
- Transferred and stored data should adopt the XML format.
- The GET REST operations return the resource properties in the XML format.
- The GET REST operation for modules return the module properties and all existing data resources.
- Middleware should persist resources and resource's data on a database.

List of properties, per resource type (res_type):

application:
id <int>; name <string>; creation_dt <string/ISO DateTime>

module:
id <int>; name <string>; creation_dt <string/ISO DateTime>; parent <int>

data:
id <int>; content <string>; creation_dt <string/ISO DateTime>; parent <int>

subscription:
id <int>; name <string>; creation_dt <string/ISO DateTime>; parent <int>; event <string>; endpoint <string>

More details about properties:

- The parent property should store the unique id of the parent resource.
- The id property of each resource should be unique.
- The datetime should be stored in a simplified ISO date format, e.g., 2002-09-28 12:34:23.
- The event property can only contain values: creation or deletion.
- The endpoint property simply stores the endpoint of the messaging broker. The channel is guessed during runtime.
- Property res_type, usually referred to in HTTP body could be application, module, data and subscription.

The testing application

The second part of the project deals with a sample application to test the middleware. Next an example of the proposed scenario it is presented (but each group could choose it's one).

In the context of Internet of Things, imagine we want to create or implement a light bulb to be controlled through desktop applications (web applications, or mobile app can also be available in future versions).
The first step is to create a new application resource with the name (for example) **Lighting.** This could be done the first time the light bulb is attached. Besides the application resource, the light bulb also creates the related module resource called **light_bulb**.
On the other hand, the mobile app creates the module resource called **light_command**.

The light bulb should create a subscription on its module for event **creation**. Hence, to turn on or off the light bulb the mobile app has just to create (write) a data resource on the light_bulb module. The simple application is now represented in the next block diagram.

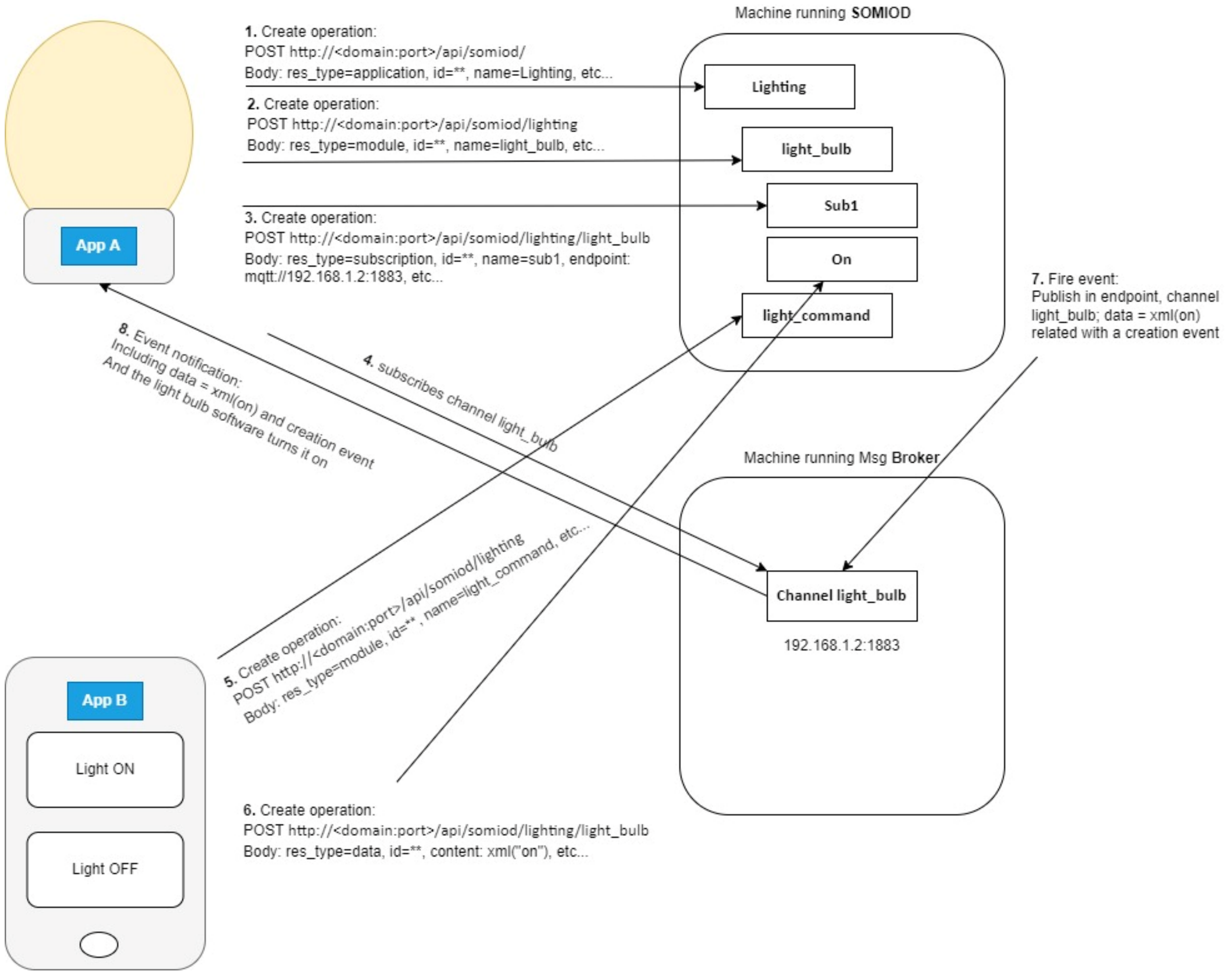
Machine running SOMIOD
1. Create operation:
POST http://<domain:port>/api/somiod/
Body: res_type=application, id=**, name=Lighting, etc...
2. Create operation:
POST http://<domain:port>/api/somiod/lighting
Body: res_type=module, id=**, name=light_bulb, etc...
3. Create operation:
POST http://<domain:port>/api/somiod/lighting/light_bulb
Body: res_type=subscription, id=**, name=sub1, endpoint: mqtt://192.168.1.2:1883, etc...
Lighting
light_bulb
Sub1
On
light_command
App A
7. Fire event:
Publish in endpoint, channel light_bulb; data = xml(on) related with a creation event
8. Event notification:
Including data = xml(on) and creation event
And the light bulb software turns it on
4. subscribes channel light_bulb
Machine running Msg Broker
Channel light_bulb
192.168.1.2:1883
5. Create operation:
POST http://<domain:port>/api/somiod/lighting
Body: res_type=module, id=** , name=light_command, etc...
App B
Light ON
Light OFF
6. Create operation:
POST http://<domain:port>/api/somiod/lighting/light_bulb
Body: res_type=data, id=**, content: xml("on"), etc...

Figure 1 - Generic architecture of the sample testing application

Project assessment

Project assessment will rely in the following criteria:

Criterion	% [0-100]
Middleware	**60%**
CRUD operations for Application resource	15%
CRUD operations for Module resource	15%
CRUD operations for Data resource	15%
CRUD operations for Subscription resource	15%
Testing applications (able to test middleware operations (final hardware and final control software, if needed, may be emulated with desktop applications)	**25%**
Project report (among other information, should include a section with cURL commands for CRUD operations for all supported resources)	**15%**

Guidelines

Groups must be formed by 4 students, which can be from different practical classes. The name and number of the group members must be submitted in the courses page (http://ead.ipleiria.pt - Moodle) until the end of November. The project delivery must be performed until the date published in the official assessment calendar. It has a mandatory project discussion that will occur in a later date, also published in the course assessment calendar.

Besides the source code of the entire solution, it is also mandatory to deliver a **written report** following the published template, which could also include appendixes. The template for the report will be available in the course web page. Remember to tag your source code (using text headers) because, as this project is an evolutionary project, your source code could be elected to be used as a starting point in the next school year.

Delivery

The project delivery must be done in the course web page (http://ead.ipleiria.pt) where students must submit a link to an external ZIP file (or .7z file) with all the project material. The link to the project delivery should use an external cloud service, e.g., Dropbox (shared file), WeTransfer, etc.
Therefore, students must guaranty that all the following material is included in a ZIP file (.7z) when delivering the project, and with the following organization structure:

- Text file (**identification.txt**) with all the information about the team members (number, name, and e-mail) and course name, etc.
- Folder **Project**: all the source code must be included in this folder. The source code of the project must include source files and other resource (files, executables, dll's, etc.)
- Folder **Report**: the written report in .docx format. Don't forget that in the end of the report (appendix) it is mandatory to identify which features has each team member implemented.
- Folder **Other**: other files required by the project that were used by the team members but were not included in the previous folders.
- Folder **Data**: the SQL data files (.sql) and the database files (.mdf and .ldf). Therefore, the database must have data to allow the testing of the project. Also, it is mandatory to deliver a database script (.sql file) including the database structure and data/database records. If using a database in the cloud, the necessary credentials to connect to the database should be also provided.

References

[1] Mohapatra, S. K., Bhuyan, J. N., Asundi, P., & Singh, A. (2016). A Solution Framework for Managing Internet of Things (IOT). *International journal of Computer Networks & Communications.—2016.—8.—P*, 73-87.