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 2024/2025

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 approach). To achieve that, it is proposed the adoption of Web Services and a Web-based resource structure, always based on the open Web 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)
                container(1)
                        record (0..N)
                        notification (0..N)
                container(N)
                        record (0..N)
                        notification (0..N)
        application (B)
                container(1)
                        record (0..N)
                        notification (0..N)
                container(N)
                        record (0..N)
                        notification (0..N)
```

Next, is presented the list of properties, per resource type **(res_type**). It is mandatory to implement the middleware using the exact property names presented below, otherwise existing applications will not be able to use the middleware:

res_type	**Resource type properties**
application	`id` <int>; `name` <string>; `creation_datetime` <ISO DateTime>
container	`id` <int>; `name` <string>; `creation_datetime` <ISO DateTime>; `parent` <int>
record	`id` <int>; `name` <string>; `content` <string>; `creation_datetime` <ISO DateTime>; `parent` <int>
notification	`id` <int>; `name` <string>; `creation_datetime` <ISO DateTime>; `parent` <int>; `event` <string>; `endpoint` <string>; `enabled` <bool>

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

A **container** resource represents a way to group other resources (records and notifications). Each application can have one or more containers.

A **record** represents each data record created on a specific application container.

A **notification** resource represents the mechanism able to trigger notifications when a **new** data record was added to the container, or when a data record was **deleted** from the container where the notification was created.

Each resource above should have a unique ID, a unique name, a creation datetime, and a parent if applicable. Below there is more information about this.

Through the Web Service RESTful API, it must be possible to **create**, **modify**, **list**, **delete** and **discover** each available resource. **Record** resources and **notification** resources do not support/allow the update operation.

For creation and discovery operations, the used URL should point to the parent resource[1], while the delete and get operations should use the virtual (non-existent) **record** or **notif** reference in the URL to highlight the type of the target inner resource (e.g.: `DELETE http://<domain:port>/api/somiod/appplicationXX/containerYY /`**`record`**`/{name}` --> the "record" in the URL identifies that it represents a stored data record and not a stored notification).

For notifications, besides the ID, name, creation datetime and parent, they should support two types of events: **creation** or **deletion**, or both, and an endpoint where the notification should be fired. For each triggered notification, the record resource (created or deleted) must be part of the notification information, as well as the type of triggered event (creation, deletion). It must be possible to trigger a notification via **MQTT** or **HTTP** (e.g., notifications should support MQTT and HTTP endpoints). If required, the channel name used to fire the notification should have the same name of the path to the source container resource (e.g.: `api/somiod/applicationXX/containerYY`).

The SOMIOD middleware should persist resources and resource's data on a database.

Transferred data should always adopt the XML format.

The HTTP action used in the RESTful request should identify the target resource using the resource unique name (**and not the resource id**).

[1] e.g.: `POST` `http://<domain:port>/api/somiod/applicationXX` -> this URL allows to identify the existing parent where the container resource should be created.

The **locate** operation is represented by the **GET** HTTP verb and by the presence of an **HTTP header** called **"somiod-locate:** <res_type>**"**, where <res_type> could be **application**, **container**, **record,** and **notification**. The return is always a **list** of resource names. To clarify the difference between the standard GET operation and the LOCATE operation take the following examples:

```
GET -H "content-type: application/xml" http://<domain:9876>/api/somiod/app1
```
➔ This returns app1 data properties.

```
GET -H "content-type: application/xml" http://<domain:9876>/api/somiod/app1/cont1/record/data1
```
➔ Returns data1 record data properties.

```
GET -H "content-type: application/xml" -H "somiod-locate: application" http://<domain:9876>/api/somiod
```
➔ Returns the list of all applications names.

```
GET -H "content-type: application/xml" -H "somiod-locate: container" http://<domain:9876>/api/somiod/app1
```
➔ returns the list of all containers (names) that are child of app1.

```
GET -H "content-type: application/xml" -H "somiod-locate: record" http://<domain:9876>/api/somiod/app1
```
➔ returns the list of all the records (names) that are child of app1.

etc.

More details about properties:

- The parent property should store the unique id of the parent resource.
- The id and name property of each resource should be unique.
- The datetime should be stored in a simplified ISO date format, e.g., 2024-09-25T12:34:23...
- The event property can only contain number values: 1 for creation or 2 for deletion.
- The MQTT endpoint property simply stores the endpoint of the messaging broker. The channel is guessed during run-time. The HTTP endpoint points to a URL that supports the POST action.
- Every time a resource is requested to be created in the absence of a unique name, the software should create a unique name for it.
- Property **res_type**, usually referred to in the HTTP body could be application, container, record, and notification.

The developed RESTfull API also requires to be well-documented.

The applications

The second part of the project deals with the application "layer" to test the SOMIOD middleware.

a) **Application Scenario 1** - In the context of Internet of Things, imagine we want to create or implement a light bulb (App A) to be controlled through a mobile application (App B). Consider that the light bulb includes a RPI zero inside or any other form of single board computer.
Hence, when the light bulb is attached to the support for the first time, it creates its representing application resource with the name, e.g. (for example) **Lighting.** Besides the application resource, the light bulb also creates a container resource called **light_bulb**.
On the other hand, (by definition) the first time the mobile app is executed, it creates its representing application called **Switch**.
The light bulb should create a notification on its container for event **creation**. Hence, to turn "on" or "off" the light bulb the mobile app has just to create (write) a record resource on the light_bulb container, as this action will trigger the creation notification, and the light bulb will be notified about this new state. The simple application is represented in the next block diagram.

All notifications arriving to each application should be serialized into an **XML file** for a specific time range configured into an XML configuration file (defines the date/time to date/time range and the output file name).

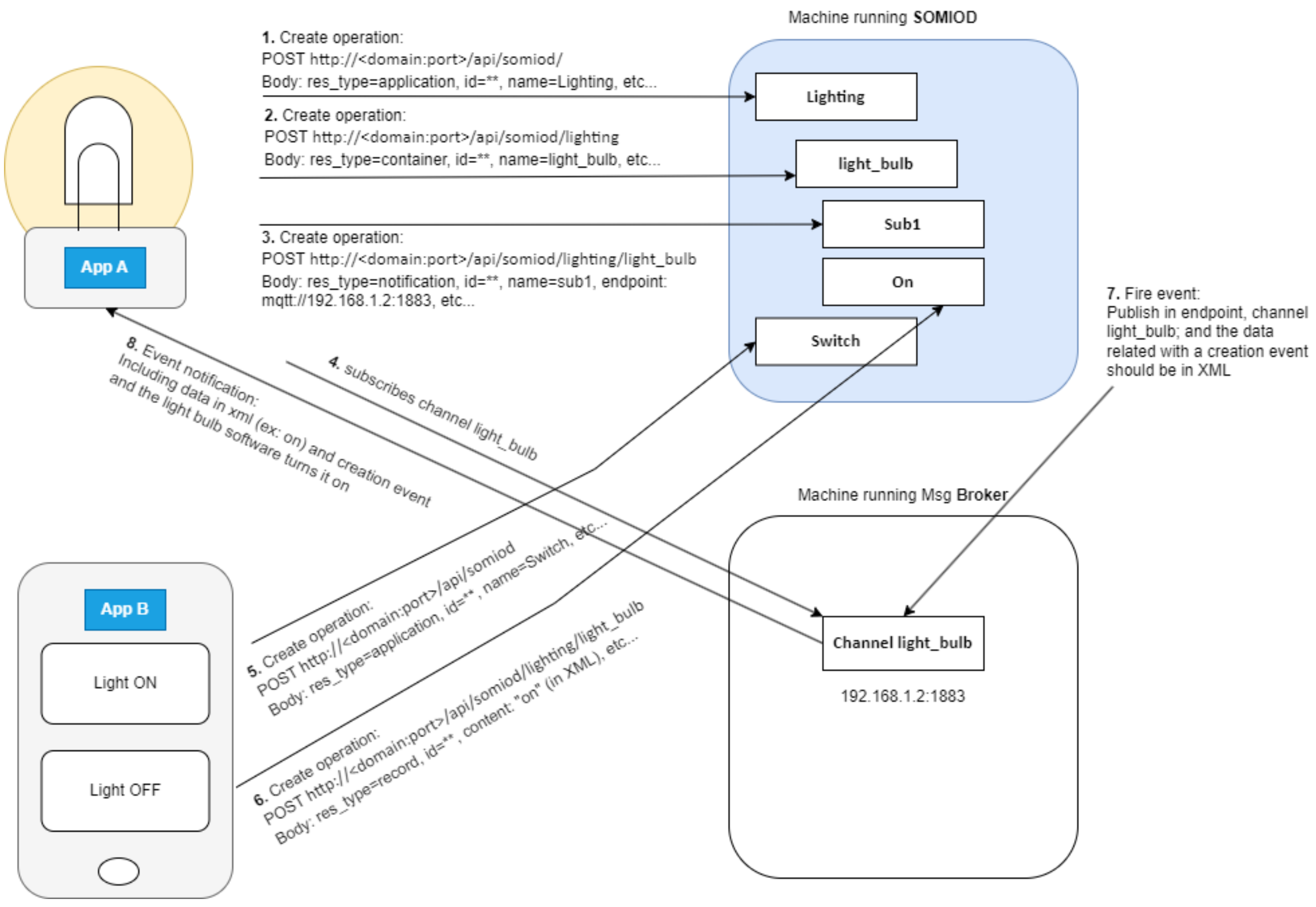
Machine running SOMIOD
1. Create operation:
POST http://<domain:port>/api/somiod/
Body: res_type=application, id=**, name=Lighting, etc...
2. Create operation:
POST http://<domain:port>/api/somiod/lighting
Body: res_type=container, id=**, name=light_bulb, etc...
3. Create operation:
POST http://<domain:port>/api/somiod/lighting/light_bulb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

Figure 1 - Generic architecture of the sample testing application

b) **Application Scenario 2** – The second application scenario deals with the test of the developed middleware. Hence a testing application must be provided and able to call each operation for each supported resource. Data required to execute the calls should be requested from the user using a form with default data.

Both application scenarios should be implemented. All the application addressed in the above scenarios can be developed as Windows Forms based.

Project assessment

Project assessment will rely in the following criteria:

Criterion	% [0-100]
Middleware	**60%**
CRUD+Locate operations for Application resource	15%
CRUD+Locate operations for Container resource	15%
CRUD+Locate operations for Record resource	15%
CRUD+Locate operations for Notification 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+Locate operations for all supported resources)	**15%**

Guidelines

Groups must have 4 students, which can be from different practical classes. The name and number of the group members must be submitted on the course page (http://ead.ipleiria.pt - Moodle) by the end of November. The project delivery must be performed until the date published in the official assessment calendar. There is a mandatory project discussion that will occur later in the semester, also published in the course assessment calendar.
Except for the database, the project must be developed using solely the **technologies used in practical classes**.

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 is available on 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 selected to be used as a starting point in the next school year.

Delivery

The project delivery must be done on 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 resources (files, executables, dll's, etc.)
- Folder **Report**: the written report in .docx format. Don't forget that at the end of the report (appendix) it is mandatory to identify which features each team member has 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 (the .mdf and .ldf file for a SQL Server database). 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 also be 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.