Service oriented Middleware for interoperability and open data

Cláudio Nicolau  
Escola Superior de Tecnologia e Gestão  
Instituto Politécnico de LeiriaLeiria, Portugal  
2212476@my.ipleiria.pt

André Filho   
Escola Superior de Tecnologia e Gestão  
Instituto Politécnico de LeiriaLeiria, Portugal  
2212948@my.ipleiria.pt

Prof. Nuno Costa  
Escola Superior de Tecnologia e Gestão  
Instituto Politécnico de LeiriaLeiria, Portugal  
nuno.costa@ipleiria.pt João Tavares  
Escola Superior de Tecnologia e Gestão  
Instituto Politécnico de LeiriaLeiria, Portugal  
2212468@my.ipleiria.pt

Prof. Humberto Ferreira  
Escola Superior de Tecnologia e Gestão  
Instituto Politécnico de LeiriaLeiria, Portugal  
humberto.ferreira@ipleiria.pt

*Abstract*

This paper reflects on the development of our service-oriented middleware for interoperability and open data (SOMIOD) and individual applications that will interoperate with each other it will also reflect on the potential scalability and applications of this technology utilizing a mosquito message broker (MQTT) allowing MQTT endpoints and Hypertext Transfer protocol endpoints (HTTP).

In this project, we successfully developed and implemented a robust SOMIOD to foster seamless interoperability among multiple applications. The primary goal was to establish a versatile framework, ensuring the effortless integration of two or more applications. By doing so, our initiative not only facilitates streamlined interactions but also offers scalable solutions, ultimately minimizing operational costs on a significant scale.

CRUD – Create, read, update , delete

Keywords—component, formatting, style, styling, insert

# Introduction

This project aims to conceptualize solutions for a myriad of societal problems by providing, through a web-based resource structure, service-oriented middleware to uniformize the way data is accessed, written and notified, independent of the application domain.

# System Architecture

The architecture consists of two independent applications, a SOMIOD and a message broker (implemented with MQTT), The applications must be able to communicate with the created API.

Depending on the request our applications perform, it is also possible to fire notifications via MQTT or HTTP endpoints.

A diagram of a software

Description automatically generated

1. System Architecture.

## Message broker

Our system uses Mosquitto as the Message Broker, central to handling notifications. It publishes messages to channels formatted as 'api/somiod/{appName}/{containerName}'. Applications subscribe to these channels to receive real-time updates. When a Data component in SOMIOD is created or deleted, an MQTT notification is sent out, ensuring subscribed applications are promptly informed of these changes.

## Somiod

This service-oriented middleware is able to define an uniformization of the way data is accessed, written and notified, independent of application domain.

* For the application component, the basic CRUD is available through POST, GET, PUT and DELETE operations respectively, the DISCOVERY request will be made with a HTTP GET containing the header “somiod-discover: application”. Deleting an application will recursively delete all its Components, Subscription and Data components (refer to section II C)
* For the container component, the basic CRUD is available through POST, GET, PUT and DELETE operations respectively, the DISCOVERY request will be made with a HTTP GET containing the header “somiod-discover: container”. Deleting an application will recursively delete all its Subscription and Data components (refer to section II C).
* For the Data component, the basic Create Read and Delete operations are available through POST, GET and DELETE requests respectively, the DISCOVERY request will be made with a HTTP GET containing the header “somiod-discover: data”. Creating or Deleting a Data component will fire a MQTT notification, to be received by the corresponding Subscription Components that contain the same father Container. No update is allowed for this component, as requested in the original specification.
* For the Subscription component, the basic Create Read and Delete operations are available through POST, GET and DELETE requests respectively, the DISCOVERY request will be made with a HTTP GET containing the header “somiod-discover: subscription.”. Creating or Deleting a Data component will fire a MQTT notification, to be received by the corresponding Subscription Components that contain the same father. No update is allowed for this component, as requested in the original specification.

## Database

This project includes a SQL Server database where all data will be read from and persisted to.

The following is the representation of our database structure.

* **Entities:**
* 1. Application:
* **Attributes:**
  + ID (Primary Key)
  + Name
  + Creation Date (creation\_dt)
* **Relationships:**
  + Has N Container (Container.Parent references Application ID)
* 2. Container:
* **Attributes:**
  + ID (Primary Key)
  + Name
  + Creation Date (creation\_dt)
  + Parent (Foreign Key)
* **Relationships:**
  + Belongs to one Application (Foreign Key “Parent” referencing Application ID)
  + Has N Data (Data.Parent references Container ID)
  + Has N Subscription (Subscription.Parent references Container ID)
* 3. Data:
* **Attributes:**
  + ID (Primary Key)
  + Name
  + Creation Date (creation\_dt)
  + Content
* **Relationships:**
  + Belongs to one Container (Foreign Key referencing Container ID)
* 4. Subscription:
* **Attributes:**
  + ID (Primary Key)
  + Name
  + Creation Date (creation\_dt)
  + Event
  + Endpoint
* **Relationships:**
  + Belongs to one Container (Foreign Key referencing Container ID)
* **Notes:**
* "N" indicates a one-to-many relationship.
* Each entity has a unique identifier (ID), a name attribute, and a creation date attribute.
* The relationships are established using foreign keys, connecting the primary key of one table to a foreign key in another to represent associations between entities.

# Integration/App Development

These applications not only enable remote control of your residence but also significantly enhances safety and various other key aspects of household management.

## Application A

This is an external application that represents a smart door, integrated with our SOMIOD and Message Broker. It performs three key actions: it creates an Application entity in SOMIOD, establishes a container, and sets up a subscription. The application is designed to respond to messages from the Broker, altering the state of the door to 'open' or 'closed' as instructed, ensuring efficient and responsive door control.

## Application B

This is an external application that will interact with our SOMIOD. This app serves as a remote controller for a door. This simple application when launched, accesses our API to create an Application entity (refer to section C), it also includes two buttons, Firstly the open button will create a Data component with the respective “open” content, the close button will do the same but with a content of “close”.

# Conclusions and Future Work

In this project, we successfully developed and implemented a robust service-oriented middleware to foster seamless interoperability among multiple applications. The primary goal was to establish a versatile framework, ensuring the effortless integration of two or more applications. By doing so, our initiative not only facilitates streamlined interactions but also offers scalable solutions, ultimately minimizing operational costs on a significant scale.

As we reflect on the achievements of this project, it becomes evident that our SOMIOD stands as a testament to the possibilities of innovative solutions in the realm of application integration. Looking forward, there are exciting prospects for expanding upon this foundation, exploring new avenues for enhancement, and adapting the middleware to meet evolving technological challenges.

# Appendix

*Appendix A*

* The curls utilized are also available separately as .txt files

=== GET ===

curl --location 'http://localhost:57696/api/somiod/teste' \

--header 'Accept: application/xml' \

--header 'Content-Type: application/xml'

Discover All Containers

curl --location 'http://localhost:57696/api/somiod' \

--header 'Content-Type: application/xml' \

--header 'somiod-discover: container'

Discover All Apps

curl --location 'http://localhost:57696/api/somiod' \

--header 'Content-Type: application/xml' \

--header 'somiod-discover: application'

Discover All application Containers

curl --location 'http://localhost:57696/api/somiod/teste' \

--header 'somiod-discover: container'

====

== POST ===

Create new Application

curl --location 'http://localhost:57696/api/somiod' \

--header 'Content-Type: application/xml' \

--header 'res\_type: application' \

--data

'<Application xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<name>teste</name>

<creation\_dt>2023-01-05</creation\_dt>

</Application>

'

== PUT ===

Update existing Application

curl --location --request

PUT 'http://localhost:57696/api/somiod/teste' \

--header 'Content-Type: application/xml' \

--header 'res\_type: application' \

--data '<Application xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<name>testeNew</name>

<creation\_dt>2023-01-25</creation\_dt>

</Application>

'

== DELETE ===

Delete Application with connected Container, Data and subscription

curl --location --request DELETE 'http://localhost:57696/api/somiod/teste' \

--header 'Content-Type: application/xml' \

--header 'res\_type: application' \

--data

'<Application xmlns:xsd=<http://www.w3.org/2001/XMLSchema> xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<name>teste3New</name>

<creation\_dt>2023-01-25</creation\_dt>

</Application>

'

* Container

Container Curl

=== GET ===

Get Current Container

curl

--location 'http://localhost:57696/api/somiod/teste/teste'

Discover All application Containers

curl --location 'http://localhost:57696/api/somiod/teste' \

--header 'somiod-discover: container'

Discover All Container subscriptions

curl

--location 'http://localhost:57696/api/somiod/casa/teste' \

--header 'somiod-discover: subscription'

====

== POST ===

Create new Container

curl --location 'http://localhost:57696/api/somiod/teste' \

--header 'Content-Type: application/xml' \

--header 'res\_type: container' \

--data

'<Container xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<name>teste4</name>

<creation\_dt>2023-01-23</creation\_dt>

</Container>'

== PUT ===

Update existing Container

curl --location --request

PUT 'http://localhost:57696/api/somiod/teste/teste' \

--header 'Content-Type: application/xml' \

--header 'res\_type: container' \

--data '<Container xmlns:xsd="

http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<name>teste9ContainerNewV2</name>

<creation\_dt>2023-01-23</creation\_dt>

</Container>'

== DELETE ===

Delete Application with connected Data and subscription

curl --location --request DELETE 'http://localhost:57696/api/somiod/teste/teste' \

--header 'Content-Type: application/xml' \

--header 'res\_type: container' \

--data

'<Container xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<name>teste9ContainerNewV2</name>

<creation\_dt>2023-01-23</creation\_dt>

</Container>'

* Subscription

Container Curl

=== GET ===

Get existing Subscription

curl --location 'http://l

ocalhost:57696/api/somiod/teste/teste/sub/teste'

Discover All Container subscriptions

Curl

--location 'http://localhost:57696/api/somiod/teste/teste' \

--header 'somiod-discover: subscription'

====

== POST ===

Create new Subscription for MQTT endpoit

Curl

--location 'http://localhost:57696/api/somiod/teste/teste/sub/' \

--header 'Content-Type: application/xml' \

--header 'res\_type: subscription' \

--data

'<Subscription xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<name>testData2</name>

<creation\_dt>2023-01-23</creation\_dt>

<event\_>1</event\_>

<endpoint>mqtt://127.0.0.1:1883</endpoint>

</Subscription>'

Create new Subscription for HTTP endpoint

curl

--location 'http://localhost:57696/api/somiod/teste/teste/sub/' \

--header 'Content-Type: application/xml' \

--data

'<Subscription xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<name>testData2</name>

<creation\_dt>2023-01-23</creation\_dt>

<event\_>1</event\_>

<endpoint>

http://localhost:57696/api/somiod/</endpoint>

</Subscription>'

== DELETE ===

Delete Subscription

curl --location --request DELETE 'http://localhost:57696/api/somiod/teste/teste/sub/teste'

* Data

Container Curl

=== GET ===

Get existing Data

curl

--location 'http://localhost:57696/api/somiod/teste/teste/data/testData2'

Discover All

#TODO

====

== POST ===

Create new Data for existing container and app

Curl

--location 'http://localhost:57696/api/somiod/teste/teste/data/' \

--header 'Content-Type: application/xml' \

--header 'res\_type: data' \

--data

'<Data xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<name>testData2</name>

<creation\_dt>2023-01-23</creation\_dt>

<content>ContentTest</content>

</Data>'

== DELETE ===

Delete Data

curl --location --request DELETE 'http://localhost:57696/api/somiod/teste/teste/data/testData2'

*Appendix B*

Application A was made by João Tavares.

Application B was made by André Filho.

The SOMIOD was made between João Tavares, André Filho and Cláudio Nicolau.

To correctly run the project, make sure to have mosquitto.exe running in advance. Setup the project to have multiple starting applications, these should be in the following order: SOMIOD – Application A – Application B.

The database files should be moved from the Data folder to the App\_Data folder.