TAMPERE UNIVERSITY OF TECHNOLOGY

Factory Information Systems

Assignment 03

Group member:

Farid Khosravi 267964

Mehdi Mahmoodpour 267958

Palash Halder 267962

ASE – 9476 Winter 2017

Contents

Objective:

The objective of this assignment is to implement the **CAMX** framework using standards such as **IPC-2501** and **IPC-2541** with the aid of Node.JS.

Fastory simulator was utilized in this assignment to simulate events of actual production line. Work-Station 8 (WS-8) was considered as reference station. Machine Application (MA) is designed to respond/gather events from conveyor system and robot which are further used in Message Broker (MSB). The communication between MA to MSB was implemented using CAMX framework, MIME messages are used to exchange data between clients. Archiver Application (AA) was used to store received data to the database.

The customer will use the saved data of database, to create trends for production system and its machines. This will allow factory to see the operations of their line in a longer term and search for optimization option.

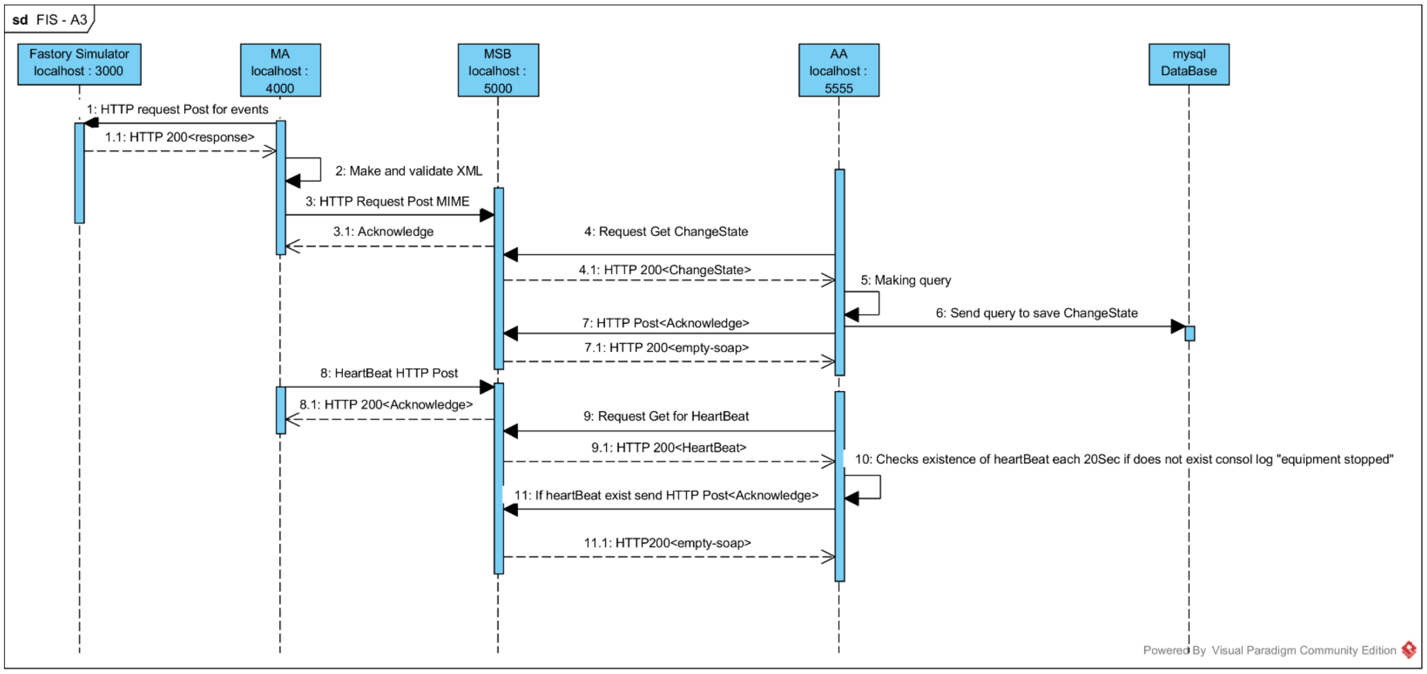
Design:

The design of application is based on the following:

- Fastory line simulator at **localhost:3000** is used to make events.
- **MA.js** is designed to represent machine application at **localhost:4000**. Two types of message are being implemented by MA.js:
    - **EquipmentChangeState**: **MA.js** has subscribed to the events of work station number 8 of **Fastory** Simulator by making a POST request at **localhost:3000**. Each event is sent as JSON to MA.js, the received data will be placed into xml format by **MA.js** based on IPC2501 and IPC2541 standards. Then the xml variables will be validated by **'xsd-schema-validator'** module against defined **xsd** defined by IPC standard.
      The **'mimemessage'** module is used to form MIME Multipart message composed of IPC-2501 messageInfo and IPC-2541 CAMX message as an attachment. The MIME message is being sent to MSB at **localhost:5000** by POST method.
    - **EquipmentHeartbeat**: Hearbeat message is being sent to MSB every 20 seconds by POST method. This message also is in MIME format and includes IPC2501 and IPC2541 messages which are being validated by '**xsd-schema-validator'** module against defined **xsd** by IPC standard before sending to MSB.

- **MSB.js** is designed to act as a broker. It receives MIME message from **MA.js**. It also receives periodic GET request from **AA.js** which informs **MSB.js** that the **AA.js** is alive and wants the heartbeat and equipment change states to be sent once they are available. Here, we have not implemented any memory with **MSB.js** as there is no Domain configuration defined. In case, there is domain configuration and multiple clients calling MSB, then the MSB should be able to store events in memory and respond periodically to AA whenever it receives request.
- **AA.js** is implemented to store data into MYSQL database for future analysis. It subscribed to MSB by GET method to receive **EquipmentChangeState** and **EquipmentHeartbeat** messages. Once any state changes in Fastory simulator occurs, such as 'zone changed' for conveyor and "DrawStart and DrawEnd" for robot , needed data must be stored into database. The useful data is made available by using **string.substring** method and then save it to database by making connection and sending appropriate query. Moreover, Heartbeat of equipment is being checked every 30 seconds and in case that heartbeat message is missing then the "Equipment Stopped" will be printed out on the console.

The following sequence diagram depicts the design of application:


sd FIS - A3
Fastory Simulator localhost : 3000
MA localhost : 4000
MSB localhost : 5000
AA localhost : 5555
mysql DataBase
1: HTTP request Post for events
1.1: HTTP 200<response>
2: Make and validate XML
3: HTTP Request Post MIME
3.1: Acknowledge
4: Request Get ChangeState
4.1: HTTP 200<ChangeState>
5: Making query
6: Send query to save ChangeState
7: HTTP Post<Acknowledge>
7.1: HTTP 200<empty-soap>
8: HeartBeat HTTP Post
8.1: HTTP 200<Acknowledge>
9: Request Get for HeartBeat
9.1: HTTP 200<HeartBeat>
10: Checks existence of heartBeat each 20Sec if does not exist consol log "equipment stopped"
11: If heartBeat exist send HTTP Post<Acknowledge>
11.1: HTTP200<empty-soap>
Powered By Visual Paradigm Community Edition

Database:

MYSQL is utilized in this assignment to store data. Description of query which is used to store data and table schema is explained in the following:

- **Table Schema:** “**CAMX**” Database consists of “**camx_info”** table. The table includes the following data:

  Data from IPC-2501 (messageInfo):

  - **sender**: URI which identifies the sender of message.
  - **destination**: it defines the destination of message.
  - **dateTime**: date and time stamp for the transaction.
  - **messageSchema**: the type of message being transported.
  - **messageId:** a unique identifier of the message being transported.

  Data of CAMX message as an attachment for **EquipmentChangeState**:

  - **currentState**: Defines the current state of equipment.
  - **previousState**: Defines the previous state of equipment.
  - **eventId**: Defines the ID of occurred event.

- Query: The data being stored into database is extracted from MIME message by **string.substring** method. Then data will be saved in database using following query:

```
connection.query('insert into camx_info
(sender,destination,datetime,messageSchema,messageId,currentSta
te,previousState,eventID) values ("' + sender + '", "' +
destination + '", "' +  dateTime + '", "' + messageSchema + '",
"' + messageId + '",' + ' "' + currentState + '", "' +
previousState + '", "' + eventID + '")'
```

XSD schemas:

Following schemas were used to make xml within **MA.js**:

MessageInfo.xsd:

```
<?xml version = "1.0" encoding = "UTF-8"?>
 <xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
            elementFormDefault = "qualified">
    <xsd:element name = "MessageInfo">
        <xsd:complexType>
            <xsd:attribute name = "dateTime" use = "required" type = "xsd:dateTime"/>
            <xsd:attribute name = "sender" use = "required" type = "xsd:anyURI"/>
            <xsd:attribute name = "destination" use = "required" type = "xsd:anyURI"/>
            <xsd:attribute name = "messageId" use = "required" type = "xsd:string"/>
            <xsd:attribute name = "messageSchema" use = "required" type =
"xsd:anyURI"/>
        </xsd:complexType>
    </xsd:element>
 </xsd:schema>
```

Acknowledge.xsd:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
   <xsd:element name = "Acknowledge">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref = "MessageId"/>
            <xsd:element ref = "Extensions" minOccurs = "0"/>
         </xsd:sequence>
         <xsd:attribute name = "dateTime" use = "required" type = "xsd:dateTime"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name = "Extensions">
      <xsd:complexType/>
   </xsd:element>
   <xsd:element name = "MessageId" type = "xsd:string"/>
</xsd:schema>
```

EquipmentChangeState.xsd:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="EquipmentChangeState">
    <xsd:complexType>
      <xsd:attribute name="dateTime" use="required" type="xsd:dateTime"/>
      <xsd:attribute name="previousState" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Z1"/>
            <xsd:enumeration value="Z2"/>
            <xsd:enumeration value="Z3"/>
            <xsd:enumeration value="Z4"/>
            <xsd:enumeration value="Z5"/>
            <xsd:enumeration value="idle"/>
            <xsd:enumeration value="processing"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="currentState" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Z1"/>
            <xsd:enumeration value="Z2"/>
            <xsd:enumeration value="Z3"/>
            <xsd:enumeration value="Z4"/>
            <xsd:enumeration value="Z5"/>
            <xsd:enumeration value="idle"/>
            <xsd:enumeration value="processing"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="eventId" use="required" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

EquipmentHeartbeat.xsd:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="EquipmentHeartbeat">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Extensions" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="dateTime" use="required" type="xsd:dateTime"/>
         <xsd:attribute name="interval" use="required" type="xsd:nonNegativeInteger"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Extensions"/>
</xsd:schema>
```

SKILLS ACQUIRED:

- Acquaintance with Node.JS, MYSQL and standards of CAMX.
- Utilizing appropriated modules for:
  - Making XML within Node.JS
  - Making MIME message.

GROUP MEMBERS' INVOLVEMENT:

- Palash Halder : Writing Node.JS application and implementing MYSQL database .
- Farid Khosravi: Writing Node.JS application and implementing MYSQL database.
- Mehdi Mahmoodpour: Writing Node.JS application and implementing MYSQL database.

*The correspondence between different applications is given as a separate attachment. The console results are stored there.