Relazione Progetto

Ingeneria del software
orientata ai servizi

May 8, 2016

Studenti:

Passaretti Simone
simone.passaretti@studio.unibo.it
Cappella Matteo
matteo.cappella@studio.unibo.it
Cimmino Martin
martin.cimmino@studio.unibo.it
Traini Stefano
stefano.traini5@studio.unibo.it

1 Obiettivi

L'obiettivo è progettare e realizzare un architettura orientata ai servizi a supporto delle attività di ACME. L'azienda si occupa di fornire biciclette assemblate su richiesta ed accessori a rivendite che operano direttamente con il pubblico. ACME si interfaccia con servizi esterni come servizio bancario, fornitore esterno e controllo coordinate geografiche: è percui necessario creare un architettura SOA che sia in grado di supportare le sue capability.

2 Tecnologie

Le tecnologie utilizzate nello sviluppo del progetto sono le seguenti:

- **Camunda**: Camunda è una piattaforma open source per il workflow e la gestione dei processi aziendali. Camunda è stato sfruttato per simulare l'esecuzione di human workflow e per l'orchestrazione dei servizi.
- **Camunda Modeler**: tool sfruttato per la modellazione del diagramma BPMN.
- **Jolie**: Linguaggio di programmazione per lo sviluppo di microservizi.

- **DB SQL**: DataBase di tipo sql, sfruttato per la memorizzazione dei dati dei magazzini.
- **JAX-RS**: è un API sviluppata in java che fornisce supporto nella creazione di servizi web sulla base del pattern architetturale REST.

3 Interpretazione delle specifiche e assunzioni effettuate

Come già brevemente introdotto nella sezione 1, ACME è un'azienda che si occupa di fornire biciclette assemblate su richiesta ed accessori a rivendite che operano direttamente con il pubblico. I componenti sono stati considerati come "customizzazioni" assemblabili sul ciclo, gli accessori come oggetto di una spedizione distinta. L'azienda riceve dalle Rivendite (considerate clienti dell'azienda) ordini relativi a un ciclo, di cui viene specificato il modello base e il colore, il quale puo essere personalizzato attraverso diverse tipologie di componenti come gruppo frenante, gruppo sterzo, trasmissione o ammortizzatori. L'azienda quindi una volta preso in carico l'ordine si occupa di assemblare il ciclo e della fornitura degli accessori. Prima di passare alla fase di assemblaggio viene tuttavia inviato alla Rivendita un preventivo eventualmente scontato a discrezione del personale ACME dedicato. La rivendita può quindi rifiutare il preventivo e terminare l'interazione, oppure accettarlo e pagare un anticipo pari ad almeno un decimo del totale. ACME a questo punto una volta verificato tramite un sistema bancario il pagamento dell'anticipo procede con la spedizione degli eventuali accessori e con l'assemblaggio del ciclo. Prima di effettuarlo, ACME verifica la presenza dei componenti necessari all'interno dei propri magazzini. L'azienda dispone di un magazzino principale che si trova nello stesso stabile dell'officina nella quale avviene l'assemblaggio e di più magazzini secondari. La presenza dei componenti viene innanzitutto verificata all'interno del magazzino principale, quindi, nel caso i componenti non siano presenti al suo interno, la si verifica in quelli secondari. Nel caso in cui i componenti si trovino nel magazzino principale questi vengono trasferiti all'officina; in caso contrario viene effettuata una spedizione per trasferirli. Nel caso in cui invece, nemmeno i magazzini secondari dispongano delle componenti necessarie, ACME richiede le mancanti a un fornitore terzo (si assume che il fornitore disponga in qualsiasi momento di ogni customizzazione) che si preoccuperà di spedirle. Lo stesso vale per gli accessori, per i quali si verifica la presenza nei magazzini e nel caso non siano presenti vengono richiesti al fornitore. Inoltre gli accessori, essendo parte di una spedizione distinta, vengono spediti alla rivendita direttamente dal magazzino piu vicino. La distanza, viene calcolata in linea d'aria attraverso un servizio di calcolo di distanza esterno. Una volta spediti sia il ciclo assemblato che gli accessori, ACME si mette in attesa che il cliente abbia versato il saldo. ACME quindi verifica tramite il sistema bancario l'avvenuto pagamento e il processo termina. Da sottolineare la possibilità che ACME possa spostare il magazzino principale in un'altra sede.

4 Progettazione dell'architettura

Il primo passo nella progettazione dell'architettura è stato la modellazione del diagramma BPMN; al suo interno sono state modellate 6 diverse **pool**, dove in particolare quella corrispondente ad ACME è divisa in due **lane** :

- **Rivendita**
- **ACME**
  - Amministrazione
  - Gestione magazzini
- **Officina**
- **Sistema bancario**
- **Fornitore**
- **Servizio di calcolo delle coordinate geografiche**

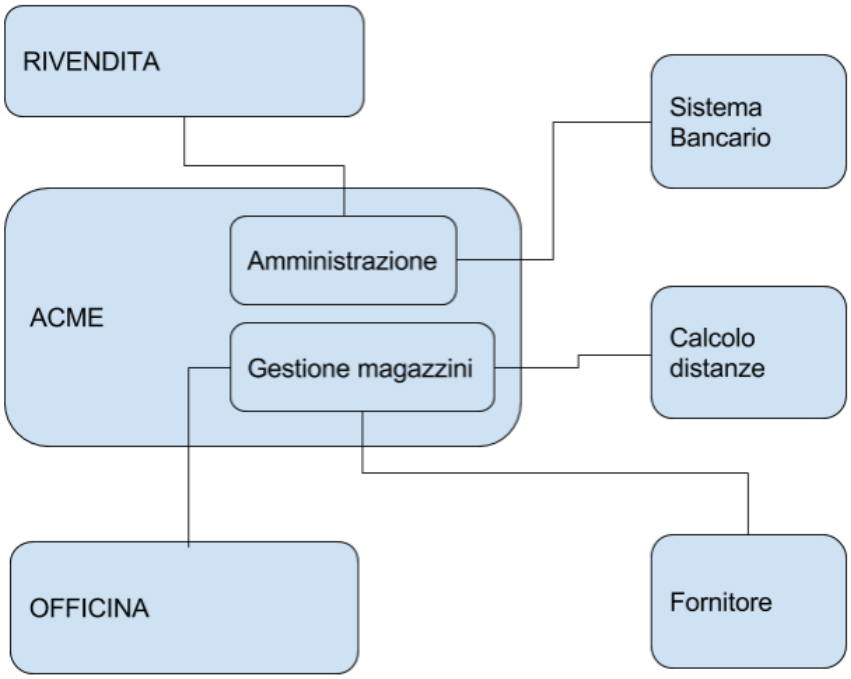


Figure 1: Mappa Concettuale

L'officina è stata considerata come una pool esterna ad ACME, al fine di renderla indipendente dal magazzino principale, per far si che in caso di un suo spostamento il diagramma non debba subire modifiche consistenti. Le pool riguardanti

il sistema bancario, il fornitore esterno e il servizio di calcolo delle coordinate geografiche sono state modellate come collassate (Black Box), mentre la Rivendita solo relativamente all'interazione con ACME; eventuali capability interne non sono state modellate in quanto non rilevanti ai fini della modellazione del processo dell'azienda ed in quanto ACME non ne è a conoscenza. Per quanto concerne ACME invece, sono state modellate tutte le sue capability.

4.1 Amministrazione

Amministrazione è una delle due lane di cui si compone la pool ACME. Le capability che mette a disposizione sono:

- Interazione con la rivendita
- Controllo fattibilità delle customizzazioni
- Calcolo del preventivo
- Valutazione di un eventuale sconto
- Interazione col sistema bancario

4.2 Gestione Magazzini

Gestione Magazzini è l'altra lane che compone la pool ACME, e si occupa di tutti gli aspetti legati ai magazzini:

- Verifica della presenza dei componenti nei magazzini principale e secondario
- Verifica della presenza degli accessori nei magazzini principale e secondari
- Trasferimento dei componenti necessari per l'ordine all'officina
- Spedizione degli accessori
- Invio dell'ordine completo del ciclo all'officina
- Interazione con il fornitore esterno
- Interazione con il sistema di calcolo delle distanze geografiche

4.3 Officina

La pool Officina, che come è stato già detto è separata dalla pool ACME, si occupa di:

- Ricevere l'ordine dai magazzini relativo al ciclo
- Effettuare l'assemblaggio
- Spedire il ciclo

5 Implementazione

5.1 Rivendita

Rivendita interagisce esclusivamente con la pool ACME, in particolare con la lane Amministrazione. Come previsto dalla specifica, il processo di ACME viene istanziato ogni qualvolta viene ricevuto un ordine da una rivendita. La creazione dell'ordine è stata modellata attraverso l'utilizzo di una form html creata all'interno della tecnologia Camunda, associata allo "start event" della pool Rivendita.

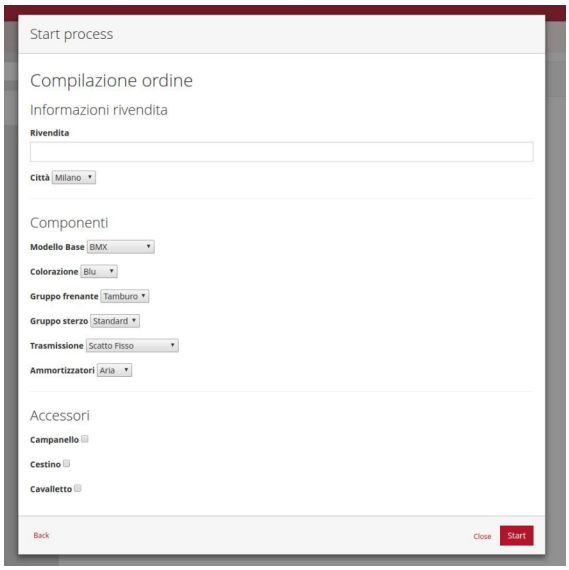


Figure 2: Form della compilazione dell'ordine

INVIO ORDINE

Di seguito viene mostrata la classe "InvioOrdine" responsabile dell'invio dell'ordine, creato attraverso la form html, verso la lane Amministrazione di ACME. Questa classe preleva i valori immessi nella form dell'immagine precedente, li immette in un file JSON "ordine" e li invia attraverso un correlate message.

```
String rivendita=(String) execution.getVariable("rivendita");
String colore=(String) execution.getVariable("colorazione");
String modellobase=(String) execution.getVariable("modellobase");
String freno=(String) execution.getVariable("freno");
String sterzo=(String) execution.getVariable("sterzo");
String ammortizzatori=(String) execution.getVariable("ammortizzatori");
String trasmissione=(String) execution.getVariable("trasmissione");
String citta=(String) execution.getVariable("citta");

execution.getProcessEngineServices().getRuntimeService()
```

```
.createMessageCorrelation("MsgOrdine")
.setVariable("id_rivendita", id_rivendita)
.setVariable("ordine", ordine.toString())
.correlate();
```

INVIO RISPOSTA

La classe java "InvioRisposta" si preoccupa di notificare all'amministrazione l'esito riguradante la valutazione del preventivo proposto. Analogamente alla classe precedentemente mostrata, viene prima prelevato il valore deciso dallo user attraverso la form html dello user task e salvato in una variabile. Quest'ultima viene poi, attraverso un correlate message, inviata all'amministrazione.

```
String appoggio= (String) execution.getVariable("ordine");
String id_acme= (String) execution.getVariable("id_acme");
JSONObject ordine = new JSONObject(appoggio);

boolean esitoprev=(boolean)execution.getVariable("esitoprev");
execution.setVariable("esitoprev", esitoprev);

execution.getProcessEngineServices().getRuntimeService()
.createMessageCorrelation("MsgInvioRisp")
.processInstanceId(id_acme)
.setVariable("esitoprev", esitoprev)
.correlate();
```

5.2 ACME

5.2.1 Amministrazione

L'amministrazione all'interno del processo ACME si occupa di interagire con la Rivendita e il sistema bancario. Sono state dunque implementate classi java per gestire lo scambio di messaggi tra Rivendita ed Amministrazione e una classe per la verifica del pagamento tramite sistema bancario.

NOTIFICA

La classe java "Notifca" si occuppa di notificare alla rivendita l'accettazione o meno dell'ordine richiesto. Terminato lo user task "controllo customizzazioni" occorre soltanto prelevare il valore della varibile "esito" ed inoltrarlo tramite un correlate message.

```
boolean accettato = (Boolean)execution.getVariable("esito");
execution.setVariable("accettato", accettato);
String id_rivendita=(String)execution.getVariable("id_rivendita");
String id_acme= execution.getId();

if (accettato == false)
LOGGER.info("L'ordine è stato rifiutato");
else
LOGGER.info("L'ordine è stato accettato");
execution.getProcessEngineServices().getRuntimeService()
.createMessageCorrelation("MsgNotifica")
```

```
.processInstanceId(id_rivendita)
.setVariable("id_acme", id_acme)
.setVariable("accettato", accettato)
.correlate();
```

VERIFICA PAGAMENTO

La classe java "VerificaPagamento" è responsabile dell'interfacciamento verso il sistema bancario esterno al fine di verificare l'avvenuto pagamento dell'anticipo o del saldo. Attraverso un chiamata http è infatti sufficiente parsare in un file JSON il contenuto della risposta del server.

```
URL url = new URL("http://localhost:8000/ServiziAcmeREST/servizi/banca");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");
conn.setRequestProperty("Accept", "application/json");
BufferedReader br = new BufferedReader
   (new InputStreamReader((conn.getInputStream())));
String prova=br.readLine();
br.close();

JSONObject jsonObj = new JSONObject(prova);
boolean pagamento= (boolean) jsonObj.get("controllo_pagamento");
execution.setVariable("pagamento", pagamento );

if(pagamento){
LOGGER.info("Il pagamento e' stato effettuato");
}
else
{
LOGGER.info("Il pagamento non e' stato effettuato");
}
```

5.3 Gestione Magazzini

Gestione Magazzini all'interno del processo ACME si occupa di determinare il controllo di disponibilità di accessori e componenti. Sulla base di questo si interfaccerà con il fornitore esterno o con il servizio distanze geografiche.

CONTROLLOCOMPONENTE

La classe "ControlloComponente" attraverso una chiamata http ai servizi jolie, ricava in un file JSON tutti i record dei componenti contenuti all'interno del magazzino principale. Successivamente controlla per ogni componente specificato nell'ordine inziale, la corrispettiva presenza o meno nel file JSON precedentemente ottenuto. In caso il componente si trovi nel magazzino principale verrà subito immesso in un file JSON rappresentate l'ordine delle componenti. Al contrario viene effetuata un'ulteriore ricerca nei magazzi secondari, tramite una chiamata http dove viene specificato il tipo di compenente ricercato. Se il componente risulatasse presente in un magazzino secondario verrà immesso nel file JSON rappresentate l'ordine delle componenti. Al contrario viene inserito un in file JSON rappresentate l'ordine delle componenti da richiedere al fornitore esterno.

```
URL url = new URL("http://localhost:8004/retrieveCompMP");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");
conn.setRequestProperty("Accept", "application/json");
BufferedReader br = new BufferedReader
   (new InputStreamReader((conn.getInputStream())));
String appoggio=br.readLine();
br.close();

for (int i = 0; i < 6; i++) {
  for(int j=0; j< rispostaMP.length(); j++){
    JSONObject riga = rispostaMP.getJSONObject(j);
     if(componenti[i].equals(riga.get("Valore_Tipo"))){
       if(riga.get("Presente").equals(true)){
          JSONObject appoggio1=new JSONObject();
          appoggio1.put("tipo", riga.get("Tipo_componente"));
          appoggio1.put("componente", riga.get("Valore_Tipo"));
          appoggio1.put("id_magazzino", riga.get("id_magazzino"));
          ordineComp.put(appoggio1);
       }
       else
       {
       URL url2 = new URL("http://localhost:8004/retrieveCompMS?valore="+
            riga.get("Valore_Tipo"));
       HttpURLConnection conn2 = (HttpURLConnection) url2.openConnection();
       conn2.setRequestMethod("GET");
       conn2.setRequestProperty("Accept", "application/json");
       BufferedReader br2 = new BufferedReader
            (new InputStreamReader((conn2.getInputStream())));
       String appoggiox =br2.readLine();
       br2.close();
       JSONObject riga11= new JSONObject(appoggiox);
       JSONArray riga12= riga11.getJSONArray("values");
       JSONObject riga13= riga12.getJSONObject(1);
       if(componenti[i].equals(riga13.get("Valore_Tipo"))){
         if(riga13.get("Presente").equals(true)){
            JSONObject appoggio2=new JSONObject();
            appoggio2.put("tipo", riga.get("Tipo_componente"));
            appoggio2.put("componente", componenti[i]);
            appoggio2.put("id_magazzino", riga.get("id_magazzino"));
            ordineComp.put(appoggio2);
         }
         else{
            JSONObject appoggio3=new JSONObject();
            appoggio3.put("componente",componenti[i]);
            appoggio3.put("tipo", riga.get("Tipo_componente"));
            ordineForn.put(appoggio3);
         }
       }

     }
  }
}
```

CONTROLLO ACCESSORIO

La classe "ControlloAccessorio" attraverso una chiamata http ai servizi jolie ricava in un file JSON tutti i record degli accessori contenuti all'interno dei magazzini. Dunque controlla per ogni accessorio specificato nell'ordine iniziale

la corrispetiva presenza o meno nel file JSON precedentemente ottenuto. In caso l'accessorio sia presente in uno dei magazzini viene subito immesso in un file JSON rappresentate l'ordine degli accessori presenti nei magazzini, al contrario viene incrementata la variabile "trovato". Se la variabile diviene uguale a tre (numero magazzini) vuol dire che l'accessorio non è presente in nessuno dei magazzini. Di conseguenza l'accessorio viene subito immesso in un file JSON rappresentate l'ordine degli accessori da richiedere al fornitore esterno.

```
URL url = new URL("http://localhost:8004/retrieveA");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");
conn.setRequestProperty("Accept", "application/json");
BufferedReader br = new BufferedReader
        (new InputStreamReader((conn.getInputStream())));
String appoggio=br.readLine();
br.close();

for(int i=0; i<3; i++){
   int trovato=0;
   if(!accessori[i].equals("")){
     for(int j=0; j<ordineAccArr.length(); j++){
       JSONObject riga = ordineAccArr.getJSONObject(j);
       if(accessori[i].equals(riga.get("Nome_Accessorio"))){
         if( riga.get("Presente").equals(true) ){
           JSONObject appoggio1= new JSONObject();
           appoggio1.put("nome_accessorio", accessori[i]);
           appoggio1.put("id_magazzino", riga.get("Id_Magazzino"));
           appoggio1.put("lat", riga.get("Latitudine"));
           appoggio1.put("lon", riga.get("Longitudine"));
           appoggio1.put("citta", riga.get("Citta"));
           ordineAccArrFinale.put(appoggio1);
         }
         else{
         trovato++;
         }
      if(trovato==3){
         JSONObject appoggio2=new JSONObject();
         appoggio2.put("nome_accessorio", accessori[i]);
         ordineAccArrForn.put(appoggio2);
      }
    }
  }
}
```

CALCOLO DISTANZE

La classe "CalcoloDistanze" ha come obiettivo quello di capire, tra i magazzini che dispongono di un determinato accessorio, qual'è il magazzino più vicino alla Rivendita. Per prima cosa, attraverso una chiamata http ai servizi Google Maps, si ottengo longitudine e latitudine della Rivendita indicando la città della rivendita. Dopodichè per ogni magazzino viene calcolata la distanza in linea d'aria e salvata in un file JSON, attraverso una chiamata al servizo esterno calcolo distanze geografiche. Infine per ognuna delle distanze immesse, viene trovata quella minore dalla Rivendita.

```
URL url = new URL
```

```
  ("http://maps.googleapis.com/maps/api/geocode/json?address=" + cittaR);
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");
conn.setRequestProperty("Accept", "application/json");
BufferedReader br = new BufferedReader
      (new InputStreamReader((conn.getInputStream())));
StringBuilder everything = new StringBuilder();
String line;
while((line = br.readLine()) != null) {
  everything.append(line);
}
String appoggio=everything.toString();
br.close();

for(int i=0; i< arrAcc.length(); i++){
   JSONObject elemento= arrAcc.getJSONObject(i);
   double latA=(double) elemento.get("lat");
   double lonA=(double) elemento.get("lon");
   URL url1 = new URL("http://localhost:8000/ServiziAcmeREST/servizi/
____calcolodistanzegeo?lat1="+ latA +"&lon1="+ lonA + "&lat2="+ latR +"&lon2="+ lonR);
   HttpURLConnection conn1 = (HttpURLConnection) url1.openConnection();
   conn1.setRequestMethod("GET");
   conn1.setRequestProperty("Accept", "application/json");
   BufferedReader br1 = new
      BufferedReader(new InputStreamReader((conn1.getInputStream())));
   String appoggio1=br1.readLine();
   br1.close();
   JSONObject rispostaCalcGeo= new JSONObject(appoggio1);
   rispostaCalcGeo.put("id_magazzino", elemento.get("id_magazzino"));
   rispostaCalcGeo.put("nome_accessorio", elemento.get("nome_accessorio"));
   rispostaCalcGeo.put("citta", elemento.get("citta"));
   distanze.put(rispostaCalcGeo);
}

for(int i=0; i<3; i++){
  double dist_min=-1;
  int id_mag=0;
  String citta="";
    for(int j=0; j< distanze.length(); j++){
      JSONObject elemDistanze= distanze.getJSONObject(j);
      if(elemDistanze.get("nome_accessorio").equals(accessori[i])){
        double distanza= (double) elemDistanze.get("distanza");
        if(dist_min==-1 || dist_min > distanza){
          dist_min=distanza;
          id_mag=(int) elemDistanze.get("id_magazzino");
          citta= (String) elemDistanze.get("citta");
        }
      }
    }
  if (dist_min!= -1){
    JSONObject accessorio = new JSONObject();
    accessorio.put("distanza_min", dist_min);
    accessorio.put("citta", citta);
    accessorio.put("id_magazzino", id_mag);
    risultato.put(accessori[i], accessorio);
  }
}
```

5.4 Servizi esterni

I servizi terzi che hanno rapporti con ACME che sono stati modellati sono:

- Servizio Bancario
- Calcolo delle distanze geografiche
- Gestione magazzini
- Fornitore esterno

5.4.1 Servizio Bancario

Il servizio bancario è stato modellato sfruttando le API java JAX-RS quindi con un'architettura di tipo REST. Come descritto nella specifica del progetto, la banca come altri servizi, sono stati modellati con logica elementare. ACME interagisce con il servizio bancario al fine di verificare il pagamento dell'anticipo e del saldo. A tal fine il servizio, contattabile tramite una chiamata http, risponde che il pagamento è stato effetuato o meno in base a una certa percentuale.

```
@Path("/banca")
public class Banca {

  public static boolean getRandom() {
    return Math.random() < 0.90;
  }

  @GET
  @Produces("application/json")
  public Response paytobank() throws JSONException {
    JSONObject jsonObject = new JSONObject();

    if(getRandom()){
      jsonObject.put("controllo_pagamento", true);
    }
    else{
      jsonObject.put("controllo_pagamento", false);
    }
  return response.status(200).entity(jsonObject.toString()).build();
  }
}
```

Naturalmente questo servizio come anche gli atri sono rappresentazioni molto semplificate ed elementari rispetto al reale funzionamento.

5.4.2 Calcolo delle distanze geografiche

Il servizio di calcolo delle distanze geografiche, anche questo implementato come un API REST, si occupa di calcolare la distanza in linea d'aria di due coordinate geografiche che nel nostro caso rappresentano le coordinate della rivendita e di uno dei magazzini.

```
@Path("/calcolodistanzegeo")
public class CalcoloDistanzeGeografiche {
  @GET
```

```
  @Produces("application/json")
  public Response getUsers(
    @QueryParam("lat1") double lat1,
    @QueryParam("lon1") double lon1,
    @QueryParam("lat2") double lat2,
    @QueryParam("lon2") double lon2) throws JSONException{

    JSONObject distanza= new JSONObject();
    lat1 = lat1*Math.PI / 180;
    lon1 = lon1*Math.PI / 180;
    lat2 = lat2*Math.PI / 180;
    lon2 = lon2*Math.PI / 180;

    double dist_long = lon2 - lon1;
    double dist_lat = lat2 - lat1;
    double pezzo1 = Math.cos(lat2)*Math.sin(dist_long);
    double pezzo11 = pezzo1*pezzo1;
    double pezzo2 = Math.cos(lat1)*Math.sin(lat2)-
      Math.sin(lat1)*Math.cos(lat2)*Math.cos(dist_long);
    double pezzo22 = pezzo2*pezzo2;
    double pezzo3 = Math.sin(lat1)*Math.sin(lat2)+
      Math.cos(lat1)*Math.cos(lat2)*Math.cos(dist_long);
    double pezzo4 = Math.atan((Math.sqrt(pezzo11+pezzo22))/pezzo3);

    distanza.put("distanza", pezzo4*6372);

    return Response
    .status(200)
    .entity(distanza.toString()).build();
  }
}
```

Come si evince dal codice la chiamata http al servizio prevede quattro parametri relativi alle coordinate geografiche per le quali calcolare la distanza in linea d'aria. L'algoritmo utilizzato per il calcolo della distanza è stato trovato in rete e utilizzato all'interno del servizio che ritorna come risultato la distanza.

5.4.3 Gestione Magazzini

La gestione dei magazzini è stata implementata utilizzando il linguaggio Jolie. Il servizio StorageService.ol, si occupa di interagire con il database di ACME al fine di verificare la presenza dei componenti. Il servizio viene chiamato attraverso chiamate http che richiamano operazioni Jolie di query al database.

Servizio Jolie

Il servizio Jolie, che si occupa di interagire con il database, viene richiamato attraverso una chiamata http, corrispondente a una specifica operazione di query sul database, con eventuali parametri necessari per effettuarla.

```
inputPort Server {
  Location: "socket://localhost:8004/"
  Protocol: http { .format = "json" }
  Interfaces: StorageInterface
}

init{
```

```
  with (connectionInfo) {
    .username = "root";
    .password = "";
    .host = "127.0.0.1";
    .port = 3306;
    .database = "acmedb";
    .driver = "mysql"
};

  connect@Database(connectionInfo)();
  println@Console("connected")()
}

main{
  [ retrieveCompMP(request)(response) {
    query@Database(
      "select magazzini.Id as id_magazzino,
              componenti.Id as Id_Componente,
              componenti.Tipo as Tipo_componente,
              componenti.Valore as Valore_Tipo,
              magazzini.Nome as Nome_Magazzino,
              cm.Presente
       from magazzini, cm, componenti
       where magazzini.Id = cm.Magazzini_Id and
             cm.Componenti_id = componenti.Id and
             magazzini.Tipo ='P'"
    )(sqlResponse);
    response.values -> sqlResponse.row
} ]

  [ retrieveCompMS(request)(response) {
    query@Database(
      "select magazzini.Id as id_magazzino,
              componenti.Id as Id_Componente,
              componenti.Tipo as Tipo_componente,
              componenti.Valore as Valore_Tipo,
              cm.Presente
       from magazzini, cm, componenti
      where magazzini.Id = cm.Magazzini_Id and
            cm.Componenti_id = componenti.Id and
            magazzini.Tipo ='S' and
            componenti.Valore =:valore"{
       .valore = request.valore
      }
   )(sqlResponse);
   response.values -> sqlResponse.row
} ]

  [ retrieveA(request)(response) {
    query@Database(
      "select magazzini.Id as Id_Magazzino,
              accessori.Id as Id_Accessori,
              accessori.Nome as Nome_Accessorio,
              am.Presente,
              magazzini.Latitudine,
              magazzini.Longitudine,
              magazzini.Nome as Citta
       from magazzini, am, accessori
       where magazzini.Id = am.Magazzini_Id and
             am.Accessori_id = accessori.Id"
     )(sqlResponse);
     response.values -> sqlResponse.row
   } ]
}
```

Le operazioni che mette a disposizione il servizio sono tre:

- **RetrieveCompMP**: operazione che recupera dal database tutti i componenti presenti all'interno del magazzino principale
- **RetrieveCompMS**: operazione che recupera dal database i dati di un componente, specificato come parametro della richiesta, all'interno di tutti i magazzini secondari
- **RetrieveA**: operazione che recupera le informazioni riguardanti gli accessori all'interno di tutti i magazzini.

Database

Il database è composto da cinque tabelle.

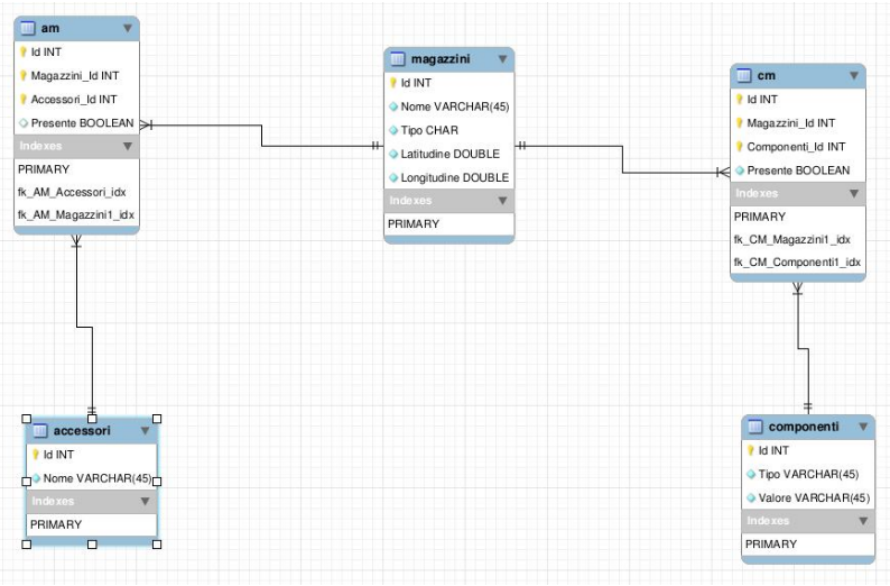


Figure 3: Modello E-R

Tre di queste sono

- **Accessori**
- **Componenti**
- **Magazzini**

Le altre due risolvono le relazioni N-N tra queste. Le tabelle Accessori e Componenti contengono al loro interno dati sulle tipologie di accessori e componenti. La tabella Magazzini contiene invece informazioni su questi ultimi come la città

nella quale il magazzino si trova e le sue coordinate geografiche o la tipologia del magazzino (primario o secondario).
Le tabelle AM e CM risolvono rispettivamente le relazioni N-N tra le tre tabelle sopra elencate. Al loro interno si ha per cui la corrispondenza tra accessori e componenti e il relativo magazzino all'interno del quale questi ultimi sono contenuti.

5.4.4 Fornitore esterno

Il fornitore esterno di accessori e componenti è stato realizzato con logica elementare come un API REST utilizzando come per gli altri JAX-RS. Si assume infatti che il fornitore disponga in ogni momento di componenti e accessori necessari per l'ordine da essere spediti ad ACME.

6 Guida al deployment

In questa sezione verranno illustrate le operazioni per la messa in opera di tutto il sistema. È richiesto:

- **Java Development Kit 7**
- **Un application server JBoss AS 7.4.0**, in particolare nel progetto si è adoperato quello distribuito da Camunda
- **MySqlServer**
- **Jolie Language**, nella ultima versione rilasciata, ovvero la 1.5.0
- **Apache Maven**, nella versione 2.3 del plugin maven-war-plugin e nella versione 3.2 del maven-compiler
- **Apache Tomcat**, nella versione 7.0
- Un accesso ad internet per accedere ai web services di Google

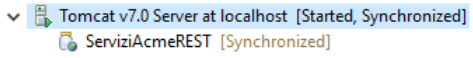
Come primo step, si avvia il servizio mySql, attraverso il comando:

```
$ sudo service mysql start
```

Successivamente, si lancia il servizio di gestione dei magazzini realizzato con Jolie:

```
$ jolie StorageService.ol
```

In seguito, è necessario effettuare il deployment dei servizi REST e avviare il server Tomcat che, nel nostro caso, rimane in ascolto nella porta 8000.



Infine, si effettua il deployment del processo ACME mediante l'esportazione dell'applicazione in un archivio **war**, posto all'interno della directory *deployments* di JBOSS che viene avviato tramite il comando:

```
$ sh start-camunda.sh
```

Una volta entrati all'interno dell'applicazione, si effettua l'accesso su **Tasklist** con l'utente **demo**. Per testare il tutto, si deve avviare la pool **Rivendita**.

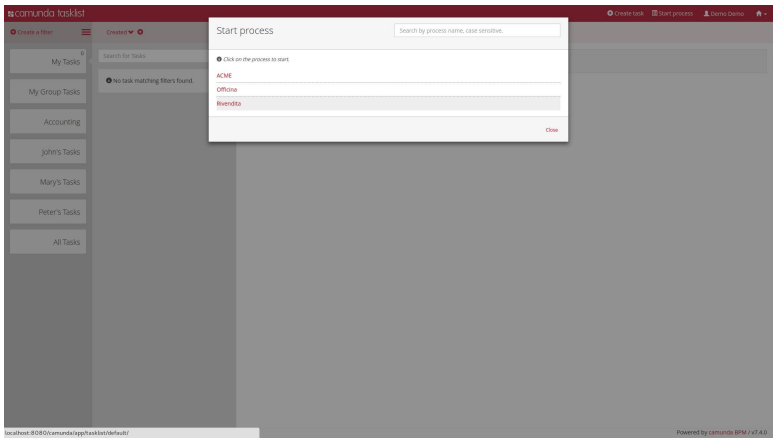


Figure 4: Schermata di Tasklist

7 Conclusioni

In conclusione, questo progetto ci ha permesso di entrare pienamente nell'ottica del paradigma di progettazione service-oriented.
Nel suo sviluppo è stato interessante l'apprendimento e l'interazione di tecnologie non banali quali il linguaggio Jolie, le API JAX-RS e la piattaforma Camunda.
Non a caso, le maggiori problematiche sono state riscontrate nell'utilizzo di quest'ultime. In modo particolare:

- l'installazione delle varie tecnologie sulle nostre macchine e la loro coesistenza
- l'automatizzazione dello scambio di messaggi fra le pool

Nonostante ciò, tali problematiche sono state superate e siamo riusciti a portare a compimento una SOA funzioante.
Naturalmente ciò che è stato realizzato rappresenta solamente lo scheletro di un possibile sistema costruibile su di esso.