

ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

LAUREA MAGISTRALE IN INFORMATICA

ACMEat

*Relazione del progetto del corso di
Ingegneria del Software Orientata ai Servizi*

Studenti:

Lorenzo BALUGANI
Alberto PAPARELLA
Mae SOSTO

Docenti:

Prof. Ivan LANESE
Prof. Davide ROSSI

ANNO ACCADEMICO 2021 – 2022

Indice

	Page
1 Introduzione	2
2 Descrizione del dominio e del problema	3
3 Modellazione delle comunicazioni	4
3.1 Coreografia	4
3.2 Proprietà di correctedness	6
3.3 Sistema di Ruoli	6
3.3.1 ACMEat	6
3.3.2 Cliente	9
3.3.3 Banca	11
3.3.4 Ristorante	13
3.3.5 Società di consegna	15
3.3.6 Fattorino	17
3.4 Diagramma di coreografia BPMN	19
4 Documentazione	21
4.1 Diagramma di collaborazione BPMN	21
4.2 ACMEat e cliente	22
4.3 Banca	28
4.4 Ristorante	30
4.5 Società di consegna e fattorino	31
5 Progettazione	33
5.1 Diagramma UML	33
5.2 Diagrammi ER	34
6 Sviluppo	37
6.1 Diagramma di processo BPMN di ACMEat	38
6.2 Banca	41
6.2.1 Parametri di configurazione	41
6.3 Backends Python	42
6.3.1 Dettagli sui backends	42
6.3.2 Parametri di configurazione	45
7 Esecuzione	48
7.1 Istruzioni per l'avvio in ambiente di testing	48
7.2 Istruzioni per il deployment in produzione	48
7.3 Post-Installazione	50
8 Conclusioni	51

1 Introduzione

Scopo di questa relazione è quello di descrivere il lavoro fatto nelle varie fasi di modellazione e sviluppo del progetto di Ingegneria del Software Orientata ai Servizi, la cui descrizione del dominio e del problema è riportata nella Sez. 2. Il codice realizzato è consultabile al seguente [repository GitHub](#). E' stata inoltre resa disponibile una [demo](#) del sistema.

Le fasi di modellazione vengono descritte a partire dalla Sez. 3, in cui viene esposta la modellazione delle comunicazioni avvenuta mediante un diagramma di coreografia dell'intero scenario, raffinata iterativamente allo scopo di migliorare quanto possibile le sue proprietà di correctedness, altresì riportate, e la successiva proiezione in un sistema di ruoli; viene inoltre riportato il corrispondente diagramma di coreografia BPMN.

Tale lavoro di modellazione prosegue nella Sez. 4 con la realizzazione di un diagramma di collaborazione BPMN rappresentante l'intera realtà descritta, compresi i dettagli di ciascun partecipante. Questo diagramma è pensato a scopo documentativo e per tale motivo è stato scelto di riportarlo suddiviso in varie parti, ciascuna con un focus differente relativamente alle varie attività descritte e brevemente commentata.

La Sez. 5 descrive poi la fase di progettazione della SOA per la realizzazione del sistema, documentata per mezzo di diagrammi UML e usando TinySOA come profiling. Vengono inoltre riportati i vari diagrammi ER delle basi di dati che verranno poi utilizzate dalle applicazioni.

La Sez. 6 descrive invece il sistema sviluppato, analizzando brevemente il funzionamento di ciascun componente e come configurare i parametri richiesti.

Infine, la Sez. 7 descrive come utilizzare l'applicazione in ambiente di testing, nonché alcune istruzioni illustrate su come effettuare un eventuale deployment in produzione.

2 Descrizione del dominio e del problema

La società ACMEat propone ai propri clienti un servizio che permette di selezionare un menu da uno fra un insieme di locali convenzionati e farselo recapitare a domicilio.

Per poter usufruire del servizio il cliente deve inizialmente selezionare un comune fra quelli nei quali il servizio è attivo. A fronte di questa selezione ACMEat presenta la lista dei locali convenzionati che operano in quel comune e dei menù che offrono. Il cliente può quindi specificare locale e menù di suo interesse e una fascia oraria per la consegna (si tratta di fasce di 15 minuti tra le 12 e le 14 e tra le 19 e le 21).

Segue quindi una fase di pagamento che viene gestita attraverso un istituto bancario terzo al quale il cliente viene indirizzato. A fronte del pagamento l'istituto rilascia un token al cliente il quale lo comunica ad ACMEat, che a sua volta lo usa per verificare con la banca che il pagamento sia stato effettivamente completato. A questo punto l'ordine diventa operativo. I clienti possono comunque ancora annullare l'ordine ma non più tardi di un'ora prima rispetto all'orario di consegna. In tal caso ACMEat chiede alla banca l'annullamento del pagamento.

ACMEat conosce tutti i locali convenzionati nei vari comuni nei quali opera e i loro giorni e orari di operatività. Nel caso in cui un locale non sia disponibile in un giorno in cui dovrebbe normalmente essere aperto è responsabilità del locale stesso contattare ACMEat entro le 10 del mattino comunicando tale indisponibilità. Entro tale orario vanno anche comunicati cambiamenti dei menu proposti (in mancanza di tale comunicazione si assume che siano disponibili gli stessi del giorno precedente). I locali vengono anche contattati ad ogni ordine per verificare che siano effettivamente in grado di far fronte alla richiesta del cliente. In caso negativo l'accettazione dell'ordine si interrompe prima che si passi alla fase di pagamento.

Per la consegna ACMEat si appoggia a più società esterne: per ogni consegna vengono contattate tutte le società che abbiano sede entro 10 chilometri dal comune interessato specificando: indirizzo del locale dove ritirare il pasto, indirizzo del cliente cui recapitarlo e orario previsto di consegna. A fronte di questa richiesta le società devono rispondere entro 15 secondi specificando la loro disponibilità e il prezzo richiesto; ACMEat sceglierà fra le disponibili che avranno risposto nel tempo richiesto quella che propone il prezzo più basso. Nel caso in cui nessuna società di consegna sia disponibile l'ordine viene annullato prima che si passi alla fase di pagamento.

3 Modellazione delle comunicazioni

La prima fase di lavoro ha visto la realizzazione di una *coreografia* (riportata in Lst. 1) con l'obiettivo di modellare le comunicazioni dello scenario descritto nella Sez. 2. Tale coreografia è stata iterativamente raffinata in modo da migliorare quanto possibile le sue proprietà di *correctedness*; per motivi di spazio, viene riportata solo l'ultima coreografia frutto di questo lavoro di raffinamento. La coreografia è stata poi proiettata in un *sistema di ruoli*, riportato nella Sez. 3.3, in cui vengono distinti i seguenti ruoli: *ACMEat*, la *banca*, il *cliente*, il *fattorino*, il *ristorante* e la *società di consegna*. Infine, viene riportata una modellazione della coreografia anche attraverso un diagramma di coreografia BPMN.

3.1 Coreografia

```
1 Coreografia ::= (
2   ModificaInformazioniLocali |
3   CoreografiaOrdine
4 )
5
6 ModificaInformazioniLocali ::= (
7   RichiestaAggiornamento : Ristorante -> ACMEat ;
8   // Sono passate le 10 ? (Si/No)
9   RichiestaRifiutata : ACMEat -> Ristorante +
10  RichiestaAccettata : ACMEat -> Ristorante
11 )
12
13 CoreografiaOrdine ::= (
14   SelezioneComune : Cliente -> ACMEat ;
15   InvioListaLocali : ACMEat -> Cliente ;
16   Ordine : Cliente -> ACMEat ;
17   RichiestaDisponibilitaRistorante : ACMEat -> Ristorante ;
18   ConfermaDisponibilitaRistorante : Ristorante -> ACMEat ;
19   (
20     RifiutoOrdine : ACMEat -> Cliente // Ristorante non disponibile
21   ) +
22   (
23     RichiestaDisponibilitaSDC : ACMEat -> SDC ;
24     PreventivoDisponibilita: SDC -> ACMEat + 1 // 1: timeout
25   )* ; // Per ogni SDC entro 10 km
26   ( // Nessun SDC disponibile
27     AnnullamentoRistorante : ACMEat -> Ristorante ;
28     RicevutoAnnullamentoRistorante: Ristorante -> ACMEat ;
29     RifiutoOrdine : ACMEat -> Cliente
30   ) +
31   (
32     ContattaSDC : ACMEat -> SDC ; // SDC costo minore
33     PresaInCarico : SDC -> ACMEat ;
34     RedirezionePagamento: ACMEat -> Cliente ;
35   (
36     Login : Cliente -> Banca ;
37     ConfermaLogin : Banca -> Cliente ;
38     Pagamento : Cliente -> Banca ;
39     InvioTokenCliente : Banca -> Cliente ;
```

```

39     Logout : Cliente -> Banca
40 ) + 1 ; // 1: timeout
41 ( // Timeout
42     CoreografiaAnnullamentoOrdine ;
43     RifiutoOrdine : ACMEat -> Cliente
44 ) + (
45     InvioTokenACMEat : Cliente -> ACMEat ;
46     Login : ACMEat -> Banca ;
47     ConfermaLogin : Banca -> ACMEat ;
48     RichiestaValidita : ACMEat -> Banca ;
49     ValiditaToken : Banca -> ACMEat ;
50     Logout : ACMEat -> Banca |
51 ( // Token non valido
52     CoreografiaAnnullamentoOrdine ;
53     RifiutoOrdine : ACMEat -> Cliente
54 ) + (
55 (
56     AttivazioneOrdineRistorante : ACMEat -> Ristorante ;
57     RicevutaAttivazioneOrdineRistorante: Ristorante -> ACMEat
58 ) | (
59     AttivazioneOrdineSDC : ACMEat -> SDC ;
60     RicevutaAttivazioneOrdineSDC: SDC -> ACMEat
61 ) ;
62     ConfermaOrdine: ACMEat -> Cliente ;
63 ( // Manca piu' di un'ora, annullare ?
64 ( // Si
65     CancellazioneOrdine: Cliente -> ACMEat ;
66     CoreografiaAnnullamentoOrdine |
67 (
68     Login : ACMEat -> Banca ;
69     ConfermaLogin : Banca -> ACMEat ;
70     AnnullamentoPagamento : ACMEat -> Banca ;
71     NotificaAnnullamentoPagamento : Banca -> ACMEat ;
72     Logout : ACMEat -> Banca
73 )
74 ) + 1 // 1: no
75 ) + ( // Manca meno di un'ora
76 (
77     Login : ACMEat -> Banca ;
78     ConfermaLogin : Banca -> ACMEat ;
79     PagamentoRistorante : ACMEat -> Banca ;
80     ConfermaPagamentoRistorante: Banca -> ACMEat ;
81     Logout : ACMEat -> Banca
82 ) | (
83     Login : ACMEat -> Banca ;
84     ConfermaLogin : Banca -> ACMEat ;
85     PagamentoSDC : ACMEat -> Banca ;
86     ConfermaPagamentoSDC: Banca -> ACMEat ;
87     Logout : ACMEat -> Banca
88 );
89     ConsegnMerceFattorino : Ristorante -> Fattorino ;
90     ConsegnMerceCliente : Fattorino -> Cliente ;
91     ConfermaRicevutaSpedizione: Fattorino -> ACMEat

```

```

92         )
93     ...
94 )
95
96 CoreografiaAnnullamentoOrdine ::= ((  

97     AnnullamentoRistorante : ACMEat -> Ristorante ;  

98     RicevutoAnnullamentoRistorante: Ristorante -> ACMEat  

99 ) | (  

100    AnnullamentoSDC : ACMEat -> SDC ;  

101    RicevutoAnnullamentoSDC : SDC -> ACMEat  

102 ))

```

Listing 1: Coreografia dello scenario di utilizzo di ACMEat

3.2 Proprietà di correctedness

Durante questa prima fase di lavoro, la coreografia è stata raffinata più volte allo scopo di migliorare tutte e tre le sue proprietà di correctedness, ovvero di composizione sequenziale, scelta, ed usi multipli della stessa operazione; ciò al fine di assicurare che la coreografia funzioni come atteso. In particolare, la correttezza della composizione sequenziale è stata migliorata aggiungendo ovunque possibile degli *ACK*, mentre la correttezza dei punti di scelta è stata assicurata verificando che lo stesso ruolo compaia in ogni transizione iniziale per ogni punto di scelta e che i ruoli dei branches relativi siano gli stessi. La correttezza delle iterazioni è ottenuta rispettando i punti precedenti, in quanto possono essere viste come una combinazione di composizione sequenziali e punti di scelta. A tal proposito, evidenziamo come il blocco alle righe da 22 a 25 sia stato modellato come blocco iterativo ma pensato come blocco parallelo (più società di consegna sono interrogate contemporaneamente), per una limitazione espressiva delle coreografie. Altra nota, è dato per scontato che ogni partecipante sia loggato nel sistema durante tutte le operazioni, fatta eccezione delle comunicazioni con la banca, per cui per motivi di sicurezza è previsto un login esplicito solo al momento della necessità ed un logout non appena l'operazione relativa è terminata.

3.3 Sistema di Ruoli

Di seguito è riportata la proiezione della coreografia in un *sistema di ruoli*, caratterizzato dai seguenti ruoli: *ACMEat*, la *banca*, il *cliente*, il *fattorino*, il *ristorante* e la *società di consegna*. È stato scelto di mantenere la stessa struttura in Lst. 1 in modo che i numeri di riga combaciassero per la stessa operazione, favorendo la visibilità e agevolando la leggibilità.

3.3.1 ACMEat

```

1 Coreografia ::= (
2     ModificaInformazioniLocali |
3     CoreografiaOrdine
4 )

```

```

5 ModificaInformazioniLocali ===
6   RichiestaAggiornamento@Ristorante ;
7   // Sono passate le 10 ? (Si/No)
8   RichiestaRifiutata@Ristorante +
9   RichiestaAccettata@Ristorante
10 )
11 )
12
13 CoreografiaOrdine ===
14   SelezioneComune@Cliente ;
15   InvioListaLocali@Cliente ;
16   Ordine@Cliente ;
17   RichiestaDisponibilitaRistorante@Ristorante ;
18   ConfermaDisponibilitaRistorante@Ristorante ;
19 (
20     RifiutoOrdine@Cliente // Ristorante non disponibile
21 ) + (
22   (
23     RichiestaDisponibilitaSDC@SDC ;
24     PreventivoDisponibilita@SDC + 1 // 1: timeout
25   ) * ; // Per ogni SDC entro 10 km
26   ( // Nessun SDC disponibile
27     AnnullamentoRistorante@Ristorante ;
28     RicevutoAnnullamentoRistorante@Ristorante ;
29     RifiutoOrdine@Cliente
30   ) + (
31     ContattaSDC@SDC ; // SDC costo minore
32     PresaInCarico@SDC ;
33     RedirezionePagamento@Cliente ;
34   (
35     1 ;
36     1 ;
37     1 ;
38     1 ;
39     1
40   ) + 1 ; // 1: timeout
41   ( // Timeout
42     CoreografiaAnnullamentoOrdine ;
43     RifiutoOrdine@Cliente
44   ) + (
45     InvioTokenACMEat@Cliente ;
46     Login@Banca ;
47     ConfermaLogin@Banca ;
48     RichiestaValidita@Banca ;
49     ValiditaToken@Banca ;
50     Logout@Banca ;
51     ( // Token non valido
52       CoreografiaAnnullamentoOrdine ;
53       RifiutoOrdine@Cliente
54     ) + (
55       (

```

```

56         AttivazioneOrdineRistoranteRistorante ;
57         RicevutaAttivazioneOrdineRistorante@Ristorante
58     ) | (
59         AttivazioneOrdineSDCSDC ;
60         RicevutaAttivazioneOrdineSDC@SDC
61     ) ;
62         ConfermaOrdine:Cliente ;
63         ( // Manca piu' di un'ora, annullare ?
64             ( // Si
65                 CancellazioneOrdine@ACMEat ;
66                 CoreografiaAnnullamentoOrdine |
67                 (
68                     Login@Banca ;
69                     ConfermaLogin@Banca ;
70                     AnnullamentoPagamentoBanca ;
71                     NotificaAnnullamentoPagamento@Banca ;
72                     Logout@Banca
73                 )
74             ) + 1 // 1: no
75         ) + ( // Manca meno di un'ora
76         (
77             Login@Banca ;
78             ConfermaLogin@Banca ;
79             PagamentoRistoranteBanca ;
80             ConfermaPagamentoRistorante@Banca ;
81             Logout@Banca
82         ) | (
83             Login@Banca ;
84             ConfermaLogin@Banca ;
85             PagamentoSDCBanca ;
86             ConfermaPagamentoSDC@Banca ;
87             Logout@Banca
88         );
89         1 ;
90         1 ;
91         ConfermaRicevutaSpedizione@ACMEat
92     )
93     ...
94 )
95
96 CoreografiaAnnullamentoOrdine ::= (
97     AnnullamentoRistoranteRistorante ;
98     RicevutoAnnullamentoRistorante@Ristorante
99 ) | (
100     AnnullamentoSDCSDC ;
101     RicevutoAnnullamentoSDC@SDC
102 ))

```

Listing 2: Proiezione della coreografia relativamente ad ACMEat

3.3.2 Cliente

```
1 Coreografia ::= (
2   ModificaInformazioniLocali |
3   CoreografiaOrdine
4 )
5
6 ModificaInformazioniLocali ::= (
7   1 ;
8   // Sono passate le 10 ? (Si/No)
9   1 +
10  1
11 )
12
13 CoreografiaOrdine ::= (
14   SelezioneComune@ACMEat ;
15   InvioListaLocali@ACMEat ;
16   Ordine@ACMEat ;
17   1 ;
18   1 ;
19   (
20     RifiutoOrdine@ACMEat // Ristorante non disponibile
21   ) + (
22     (
23       1 ;
24       1 + 1 // 1: timeout
25     ) * ; // Per ogni SDC entro 10 km
26     ( // Nessun SDC disponibile
27       1 ;
28       1 ;
29       RifiutoOrdine@ACMEat
30     ) + (
31       1 ; // SDC costo minore
32       1 ;
33       RedirezionePagamento@ACMEat ;
34     (
35       Login@Banca ;
36       ConfermaLogin@Banca ;
37       Pagamento@Banca ;
38       InvioTokenCliente@Banca ;
39       Logout@Banca
40     ) + 1 ; // 1: timeout
41     ( // Timeout
42       CoreografiaAnnullamentoOrdine ;
43       RifiutoOrdine@ACMEat
44     ) + (
45       InvioTokenACMEat@ACMEat ;
46       1 ;
47       1 ;
48       1 ;
49       1 ;
50       1 |
51       ( // Token non valido
```

```

52     CoreografiaAnnullamentoOrdine ;
53     RifiutoOrdine@ACMEat
54 ) + (
55 (
56     1 ;
57     1
58 ) | (
59     1 ;
60     1
61 )
62     ConfermaOrdine@ACMEat ;
63 ( // Manca piu' di un'ora, annullare ?
64 ( // Si
65     CancellazioneOrdine@ACMEat ;
66     CoreografiaAnnullamentoOrdine |
67 (
68     1 ;
69     1 ;
70     1 ;
71     1 ;
72     1
73     )
74 ) + 1 // 1: no
75 ) + ( // Manca meno di un'ora
76 (
77     1 ;
78     1 ;
79     1 ;
80     1 ;
81     1
82 ) | (
83     1 ;
84     1 ;
85     1 ;
86     1 ;
87     1
88 );
89     1 ;
90     ConsegnaMerceCliente@Cliente ;
91     1
92 )
93 ...
94 )
95 CoreografiaAnnullamentoOrdine ::= (((
96     1 ;
97     1
98 ) | (
99     1 ;
100    1
101   ))

```

Listing 3: Proiezione della coreografia relativamente al cliente

3.3.3 Banca

```
1 Coreografia ===
2   ModificaInformazioniLocali |
3   CoreografiaOrdine
4 )
5
6 ModificaInformazioniLocali ===
7   1 ;
8   // Sono passate le 10 ? (Si/No)
9   1 +
10  1
11 )
12
13 CoreografiaOrdine ==
14   1 ;
15   1 ;
16   1 ;
17   1 ;
18   1 ;
19   (
20     1 // Ristorante non disponibile
21   ) + (
22     (
23       1 ;
24       1 + 1 // 1: timeout
25     ) * ; // Per ogni SDC entro 10 km
26     ( // Nessun SDC disponibile
27       1 ;
28       1 ;
29       1
30     ) + (
31       1 ; // SDC costo minore
32       1 ;
33       1 ;
34     (
35       Login@Cliente ;
36       ConfermaLogin@Cliente ;
37       Pagamento@Cliente ;
38       InvioTokenCliente@Cliente ;
39       Logout@Cliente ;
40     ) + 1 ; // 1: timeout
41     ( // Timeout
42       CoreografiaAnnullamentoOrdine ;
43       1
44     ) + (
45       1 ;
46       Login@ACMEat ;
47       ConfermaLogin@ACMEat ;
48       RichiestaValidita@ACMEat ;
49       ValiditaToken : Banca -> ACMEat ;
50       Logout@ACMEat ;
51     ( // Token non valido
```

```

52     CoreografiaAnnullamentoOrdine ;
53     1
54 ) + (
55 (
56     1 ;
57     1
58 ) | (
59     1 ;
60     1
61 ) ;
62 1 ;
63 ( // Manca piu' di un'ora, annullare ?
64     ( // Si
65         1 ;
66     CoreografiaAnnullamentoOrdine |
67     (
68         Login@ACMEat ;
69     ConfermaLogin@ACMEat ;
70     AnnullamentoPagamento@ACMEat ;
71     NotificaAnnullamentoPagamento@ACMEat ;
72     Logout@ACMEat
73         )
74     ) + 1 // 1: no
75 ) + ( // Manca meno di un'ora
76 (
77     Login@ACMEat ;
78     ConfermaLogin@ACMEat ;
79     PagamentoRistorante@ACMEat ;
80     ConfermaPagamentoRistorante@ACMEat ;
81     Logout@ACMEat
82 ) | (
83     Login@ACMEat ;
84     ConfermaLogin@ACMEat ;
85     PagamentoSDC@ACMEat ;
86     ConfermaPagamentoSDC@ACMEat ;
87     Logout@ACMEat
88 );
89     1 ;
90     1 ;
91     1
92 )
93 ...
94 )
95 CoreografiaAnnullamentoOrdine ::= (((
96     1 ;
97     1
98 ) | (
99     1 ;
100    1
101   1
102 ))
```

Listing 4: Proiezione della coerografia relativamente alla banca

3.3.4 Ristorante

```

1 Coreografia == (
2   ModificaInformazioniLocali |
3   CoreografiaOrdine
4 )
5
6 ModificaInformazioniLocali == (
7   RichiestaAggiornamento@ACMEat ;
8   // Sono passate le 10 ? (Si/No)
9   RichiestaRifiutata@ACMEat +
10  RichiestaAccettata@ACMEat
11 )
12
13 CoreografiaOrdine == (
14   1 ;
15   1 ;
16   1 ;
17   RichiestaDisponibilitaRistorante@ACMEat ;
18   ConfermaDisponibilitaRistorante@ACMEat ;
19   (
20     1 // Ristorante non disponibile
21   ) + (
22     (
23       1 ;
24       1 + 1 // 1: timeout
25     )* ; // Per ogni SDC entro 10 km
26     ( // Nessun SDC disponibile
27       AnnullamentoRistorante@Ristorante ;
28       RicevutoAnnullamentoRistorante@ACMEat ;
29       1
30     ) + (
31       1 ; // SDC costo minore
32       1 ;
33       1 ;
34     (
35       1 ;
36       1 ;
37       1 ;
38       1 ;
39       1
40     ) + 1 ; // 1: timeout
41     ( // Timeout
42       CoreografiaAnnullamentoOrdine ;
43       1
44     ) + (
45       1 ;
46       1 ;
47       1 ;
48       1 ;
49       1 ;
50       1 |
51       ( // Token non valido

```

```

52     CoreografiaAnnullamentoOrdine ;
53     1
54 ) + (
55 (
56     AttivazioneOrdineRistorante@ACMEat ;
57     RicevutaAttivazioneOrdineRistorante@ACMEat
58 ) | (
59     1 ;
60     1
61 ) ;
62 1 ;
63 ( // Manca piu' di un'ora, annullare ?
64   ( // Si
65     1 ;
66     CoreografiaAnnullamentoOrdine |
67   (
68     1 ;
69     1 ;
70     1 ;
71     1 ;
72     1
73   )
74 ) + 1 // 1: no
75 ) + ( // Manca meno di un'ora
76   (
77     1 ;
78     1 ;
79     1 ;
80     1 ;
81     1
82   ) | (
83     1 ;
84     1 ;
85     1 ;
86     1 ;
87     1
88 );
89     ConsegnaMerceFattorino@Fattorino ;
90     1 ;
91     1
92   )
93 ...
94 )
95
96 CoreografiaAnnullamentoOrdine ::= (((
97     AnnullamentoRistorante@ACMEat ;
98     RicevutoAnnullamentoRistorante@ACMEat
99 ) | (
100    1 ;
101    1
102 ))
```

Listing 5: Proiezione della coreografia relativamente al ristorante

3.3.5 Società di consegna

```

1 Coreografia ::= (
2   ModificaInformazioniLocali |
3   CoreografiaOrdine
4 )
5
6 ModificaInformazioniLocali ::= (
7   1 ;
8   // Sono passate le 10 ? (Si/No)
9   1 +
10  1
11 )
12
13 CoreografiaOrdine ::= (
14   1 ;
15   1 ;
16   1 ;
17   1 ;
18   1 ;
19   (
20     1 // Ristorante non disponibile
21   ) + (
22     (
23       RichiestaDisponibilitaSDC@ACMEat ;
24       PreventivoDisponibilita@ACMEat + 1 // 1: timeout
25     )* ; // Per ogni SDC entro 10 km
26     ( // Nessun SDC disponibile
27       1 ;
28       1 ;
29       1
30     ) + (
31       ContattaSDC@ACMEat ; // SDC costo minore
32       PresaInCarico@ACMEat ;
33       1 ;
34     (
35       1 ;
36       1 ;
37       1 ;
38       1 ;
39       1
40     ) + 1 ; // 1: timeout
41     ( // Timeout
42       CoreografiaAnnullamentoOrdine ;
43       1
44     ) + (
45       1 ;
46       1 ;
47       1 ;
48       1 ;
49       1 ;
50       1 |
51       ( // Token non valido

```

```

52     CoreografiaAnnullamentoOrdine ;
53     1
54 ) + (
55 (
56     1 ;
57     1
58 ) | (
59     AttivazioneOrdineSDC@ACMEat ;
60     RicevutaAttivazioneOrdineSDC@ACMEat
61 ) ;
62 1 ;
63 ( // Manca piu' di un'ora, annullare ?
64 ( // Si
65     1 ;
66     CoreografiaAnnullamentoOrdine |
67 (
68     1 ;
69     1 ;
70     1 ;
71     1 ;
72     1
73     )
74 ) + 1 // 1: no
75 ) + ( // Manca meno di un'ora
76 (
77     1 ;
78     1 ;
79     1 ;
80     1 ;
81     1
82 ) | (
83     1 ;
84     1 ;
85     1 ;
86     1 ;
87     1
88 );
89     1 ;
90     1 ;
91     1
92 )
93 ...
94 )
95 CoreografiaAnnullamentoOrdine ::= (((
96     1 ;
97     1
98 ) | (
99     AnnullamentoSDC@ACMEat ;
100    RicevutoAnnullamentoSDC@ACMEat
101 ))

```

Listing 6: Proiezione della coreografia relativamente alla società di consegna

3.3.6 Fattorino

```
1 Coreografia ::= (
2   ModificaInformazioniLocali |
3   CoreografiaOrdine
4 )
5
6 ModificaInformazioniLocali ::= (
7   1 ;
8   // Sono passate le 10 ? (Si/No)
9   1 +
10  1
11 )
12
13 CoreografiaOrdine ::= (
14   1 ;
15   1 ;
16   1 ;
17   1 ;
18   1 ;
19   (
20     1 // Ristorante non disponibile
21   ) + (
22     (
23       1 ;
24       1 + 1 // 1: timeout
25     ) * ; // Per ogni SDC entro 10 km
26     ( // Nessun SDC disponibile
27       1 ;
28       1 ;
29       1
30     ) + (
31       1 ; // SDC costo minore
32       1 ;
33       1 ;
34       (
35         1 ;
36         1 ;
37         1 ;
38         1 ;
39         1
40       ) + 1 ; // 1: timeout
41     ( // Timeout
42       CoreografiaAnnullamentoOrdine ;
43       1
44     ) + (
45       1 ;
46       1 ;
47       1 ;
48       1 ;
49       1 ;
50       1 |
51       ( // Token non valido
```

```

52     CoreografiaAnnullamentoOrdine ;
53     1
54 ) + (
55 (
56     1 ;
57     1
58 ) | (
59     1 ;
60     1
61 ) ;
62 1 ;
63 ( // Manca piu' di un'ora, annullare ?
64   ( // Si
65     1 ;
66   CoreografiaAnnullamentoOrdine |
67   (
68     1 ;
69     1 ;
70     1 ;
71     1 ;
72     1
73   )
74 ) + 1 // 1: no
75 ) + ( // Manca meno di un'ora
76 (
77     1 ;
78     1 ;
79     1 ;
80     1 ;
81     1
82 ) | (
83     1 ;
84     1 ;
85     1 ;
86     1 ;
87     1
88 );
89 ConsegnaMerceFattorino@Fattorino ;
90 ConsegnaMerceClienteCliente ;
91 ConfermaRicevutaSpedizione@ACMEat
92 )
93 ...
94 )
95
96 CoreografiaAnnullamentoOrdine ::= (((
97     1 ;
98     1
99 ) | (
100     1 ;
101     1
102 ))
```

Listing 7: Proiezione della coreografia relativamente al fattorino

3.4 Diagramma di coreografia BPMN

Al termine di questa prima fase è stato realizzato un diagramma di coreografia BPMN raffigurante la medesima coreografia in Lst. 1. Questo diagramma si è rivelato utile nelle fasi successive in quanto più visivo e semplice da leggere.

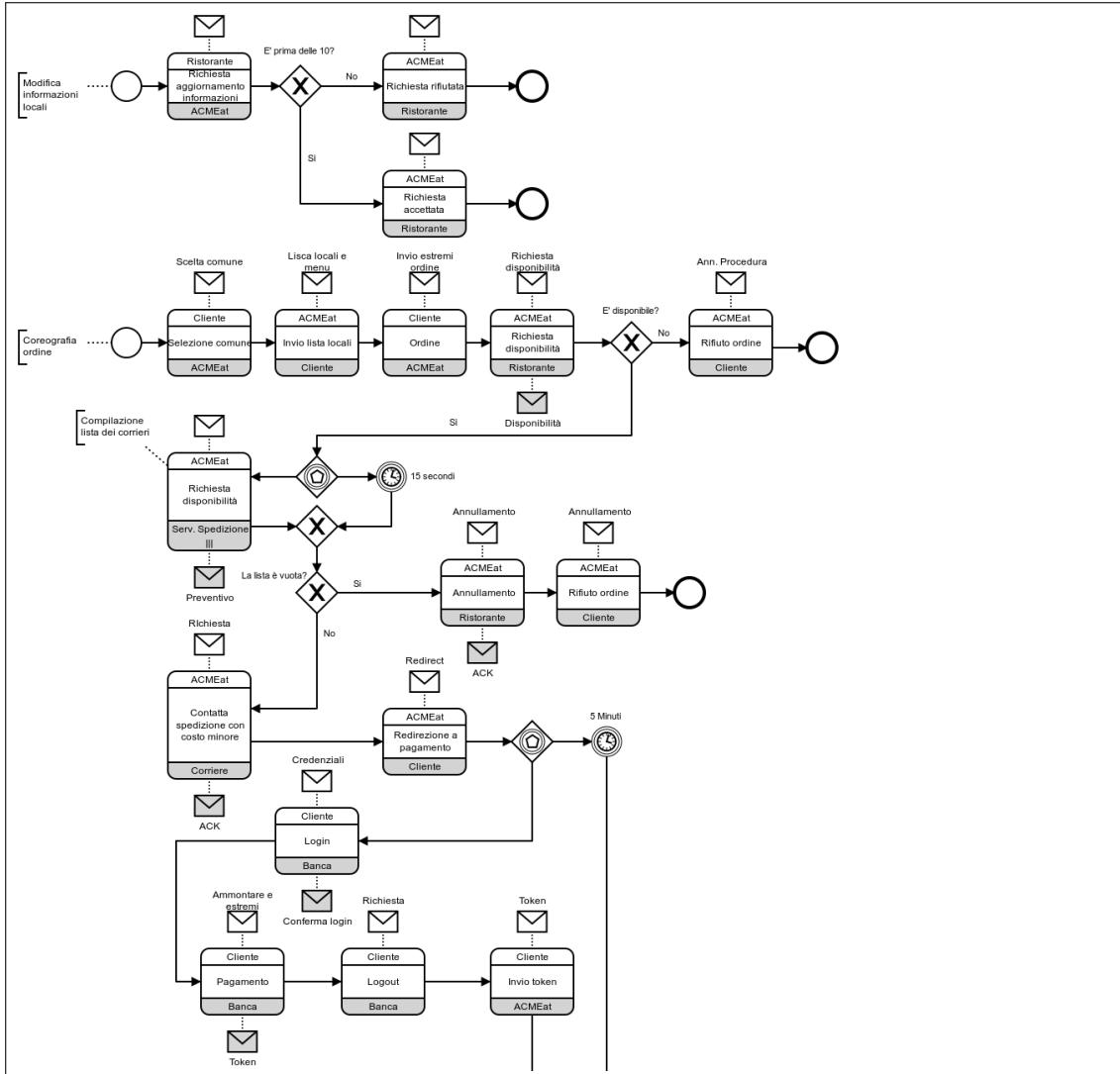


Figura 1: Diagramma di coreografia BPMN (1)

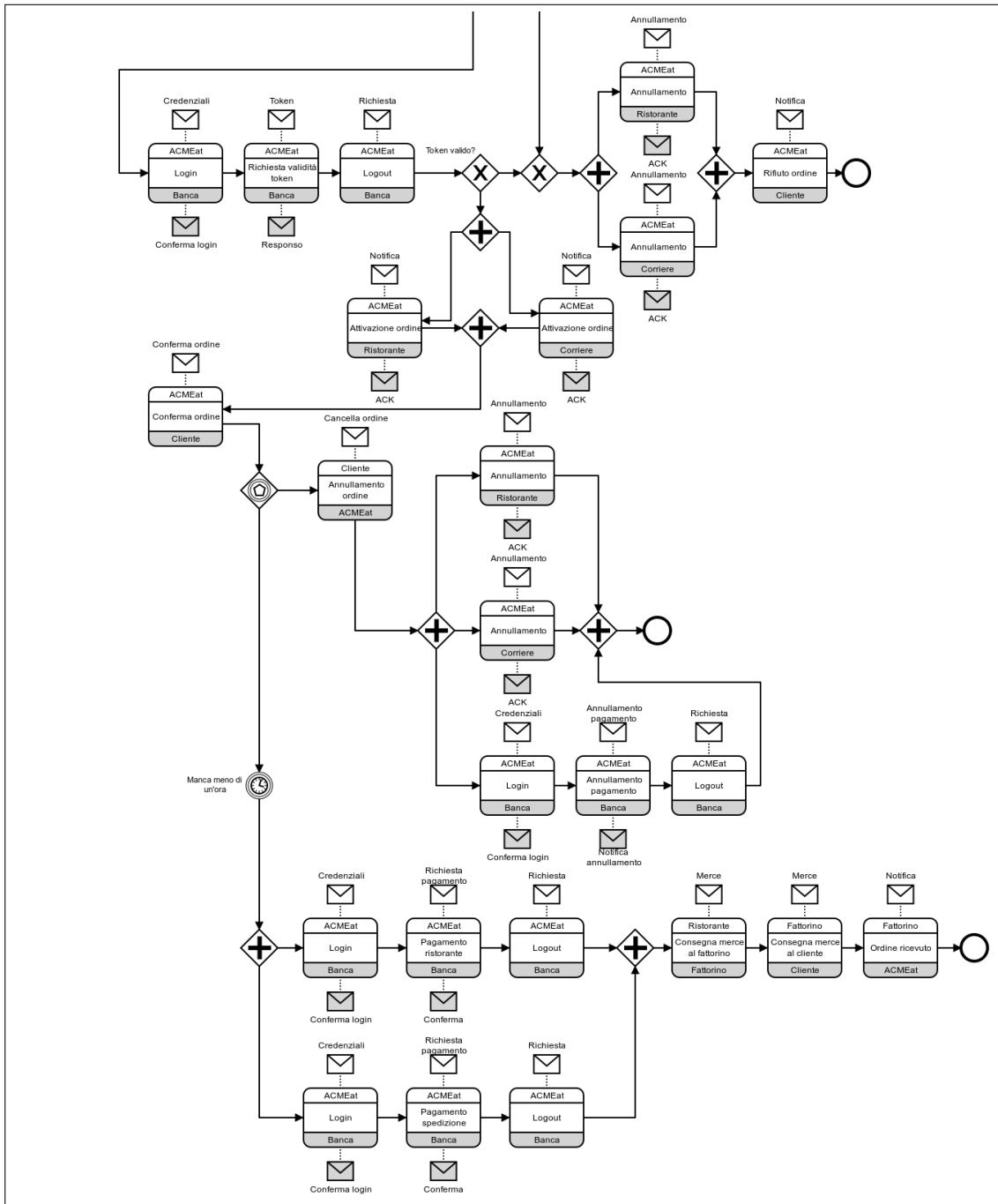


Figura 2: Diagramma di coreografia BPMN (2)

4 Documentazione

Durante la seconda fase di lavoro è stato realizzato un diagramma di collaborazione BPMN (Fig. 3) con l'obiettivo di modellare l'intera realtà descritta a scopo documentativo, compresi i dettagli di ogni partecipante. Questo diagramma è stato suddiviso in diverse parti per facilitarne la leggibilità e la consultazione - riportate nelle Sez. 4.2, 4.3, 4.4 e 4.5. Per ogni figura, le pool coinvolte sono specificate nella caption (tranne nel caso del cliente, il cui comportamento è mostrato sempre in relazione ad ACMEat), così come una breve descrizione di quale parte del processo è modellata. Tale diagramma è stato reso il più possibile compliant con la coreografia in Lst. 1 precedentemente realizzata. E' stato scelto di documentarla il più possibile ma non rendere eseguibili le pool, lasciando la raffinazione a un diverso diagramma BPMN che offrisse capabilities attraverso il BPMS Camunda da sviluppare in una fase successiva.

4.1 Diagramma di collaborazione BPMN

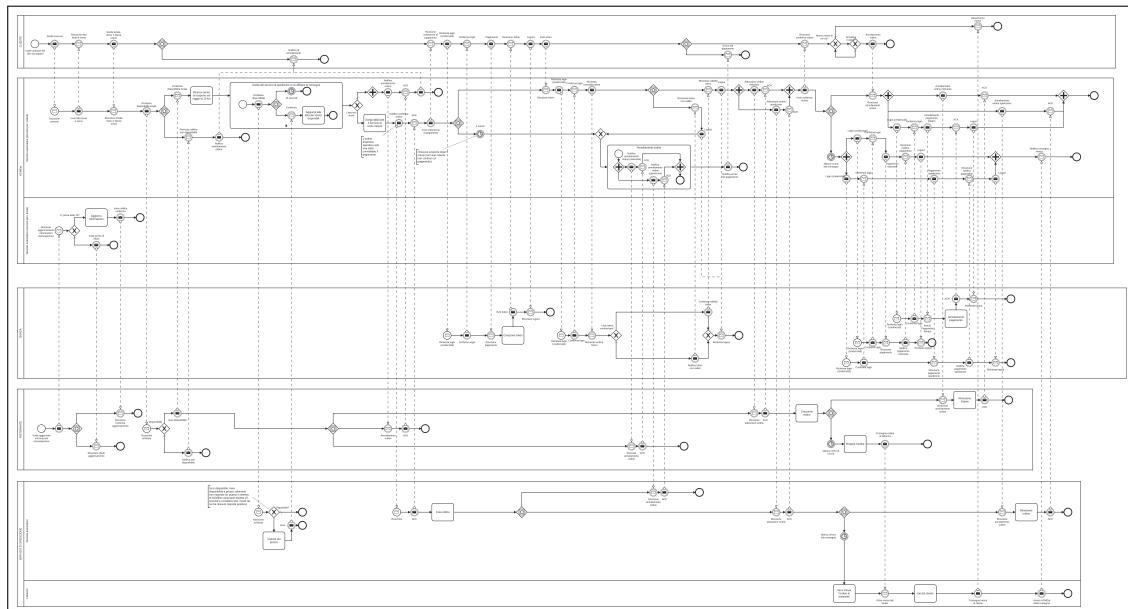


Figura 3: Diagramma di collaborazione BPMN dello scenario

4.2 ACMEat e cliente

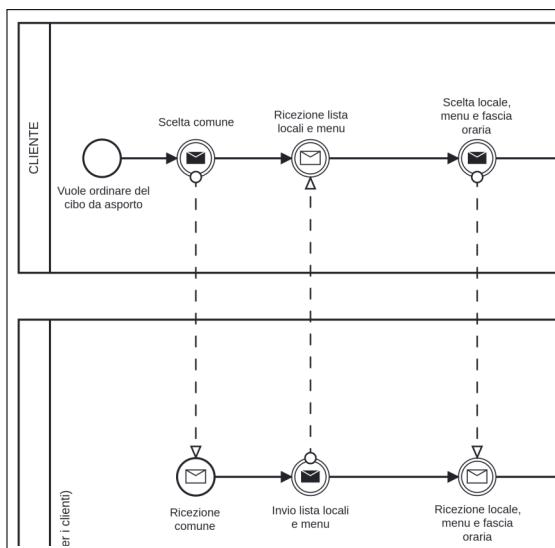


Figura 4: Scambio di messaggi iniziale fra un cliente e ACMEat

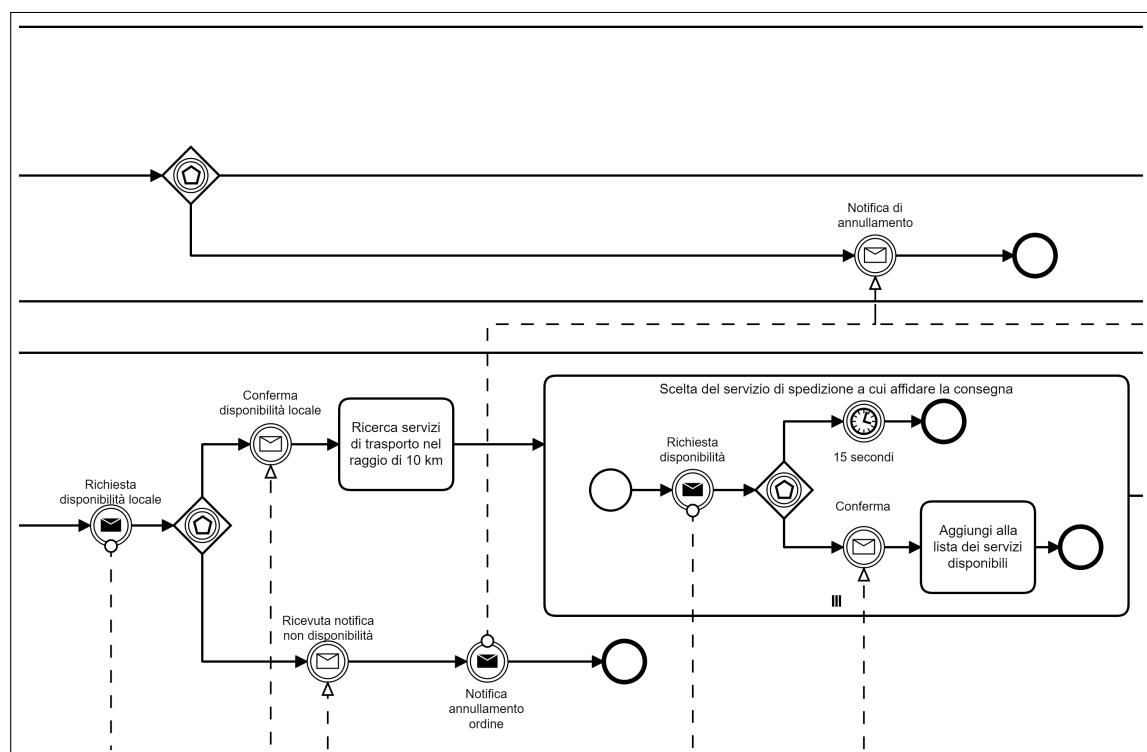


Figura 5: Richiesta disponibilità locale e servizi di consegna - lato ACMEat

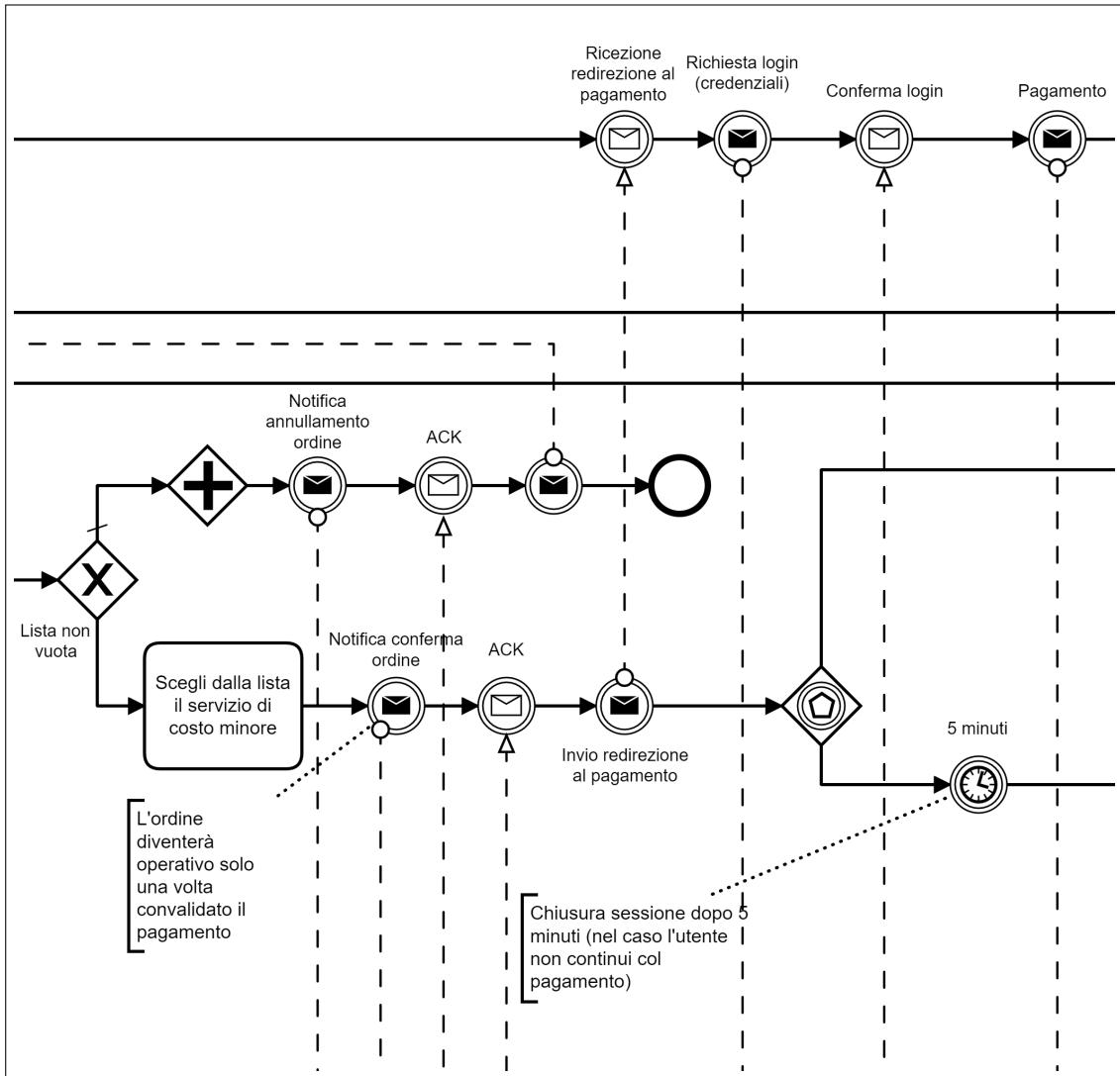


Figura 6: Scelta servizio di consegna e redirezione pagamento - lato ACMEat

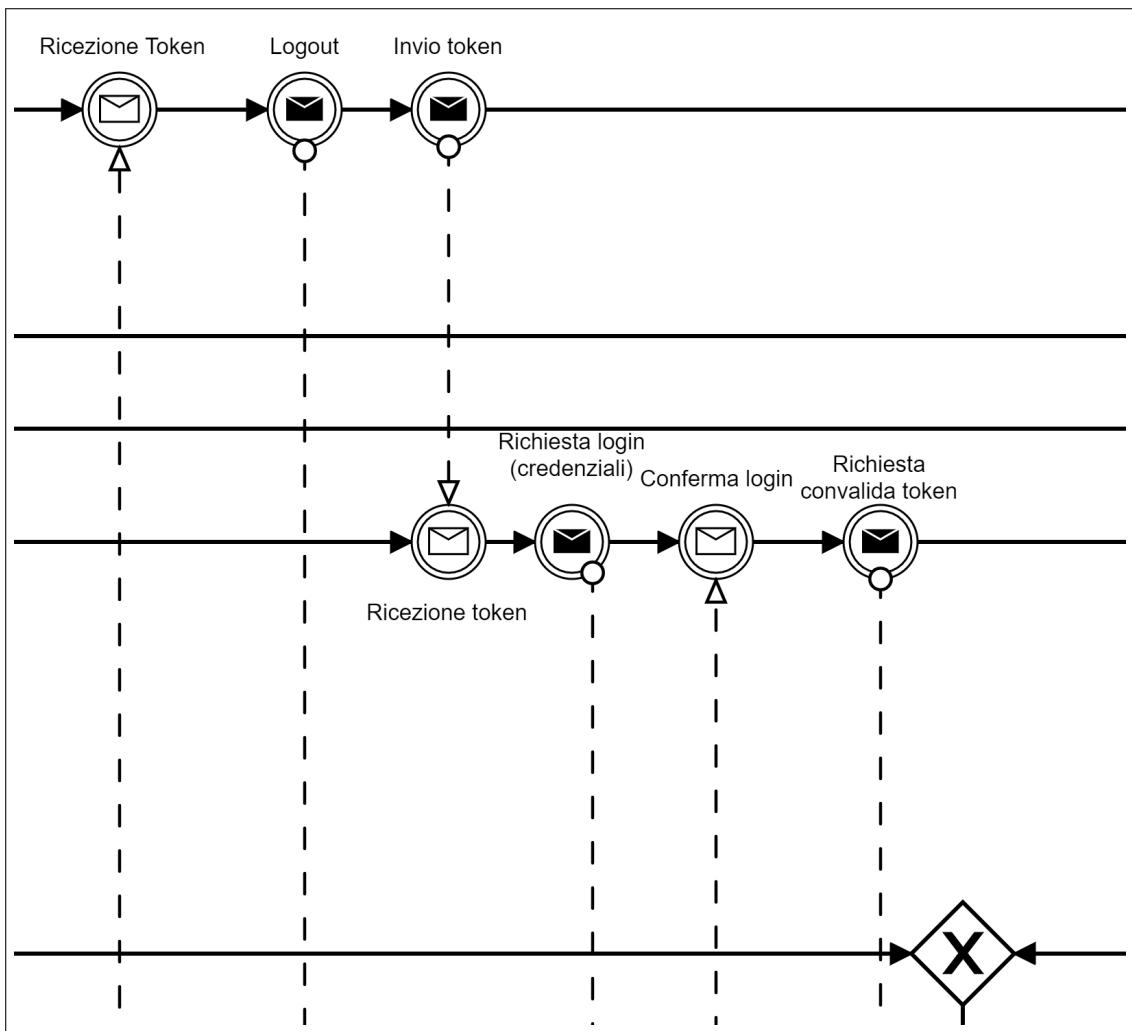


Figura 7: Rimbalzo del token per convalida - lato ACMEat

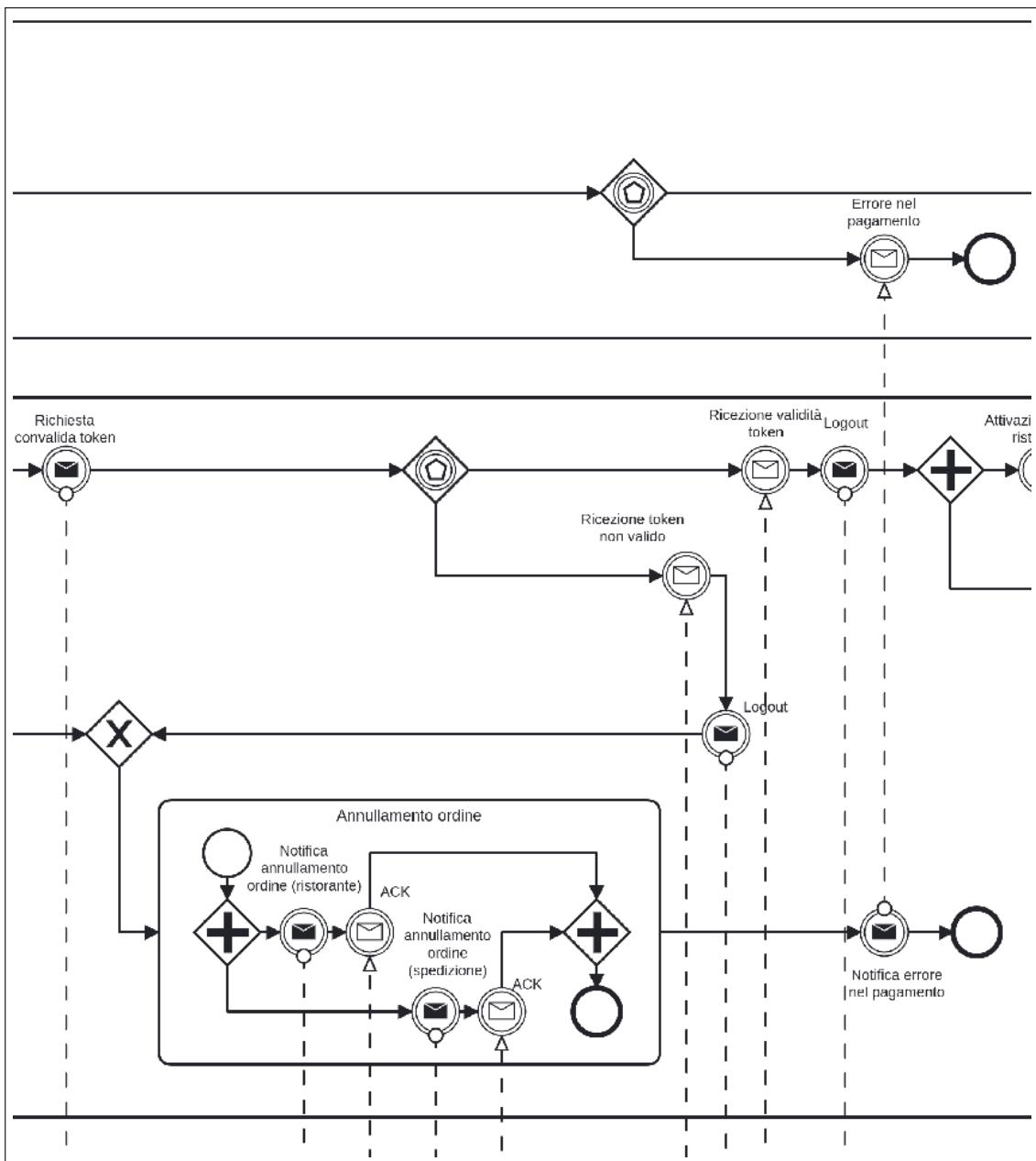


Figura 8: Gestione annullamento ordine - lato ACMEat

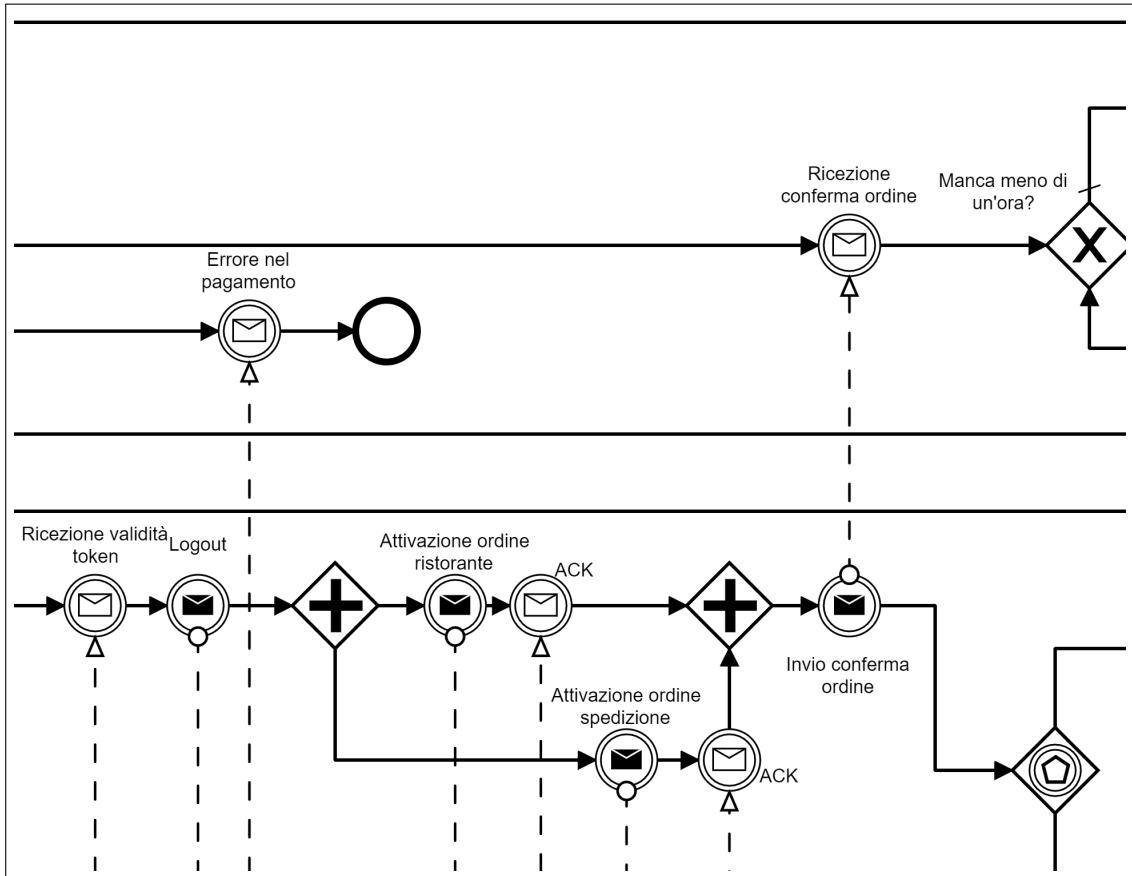


Figura 9: Gestione attivazione ordine - lato ACMEat

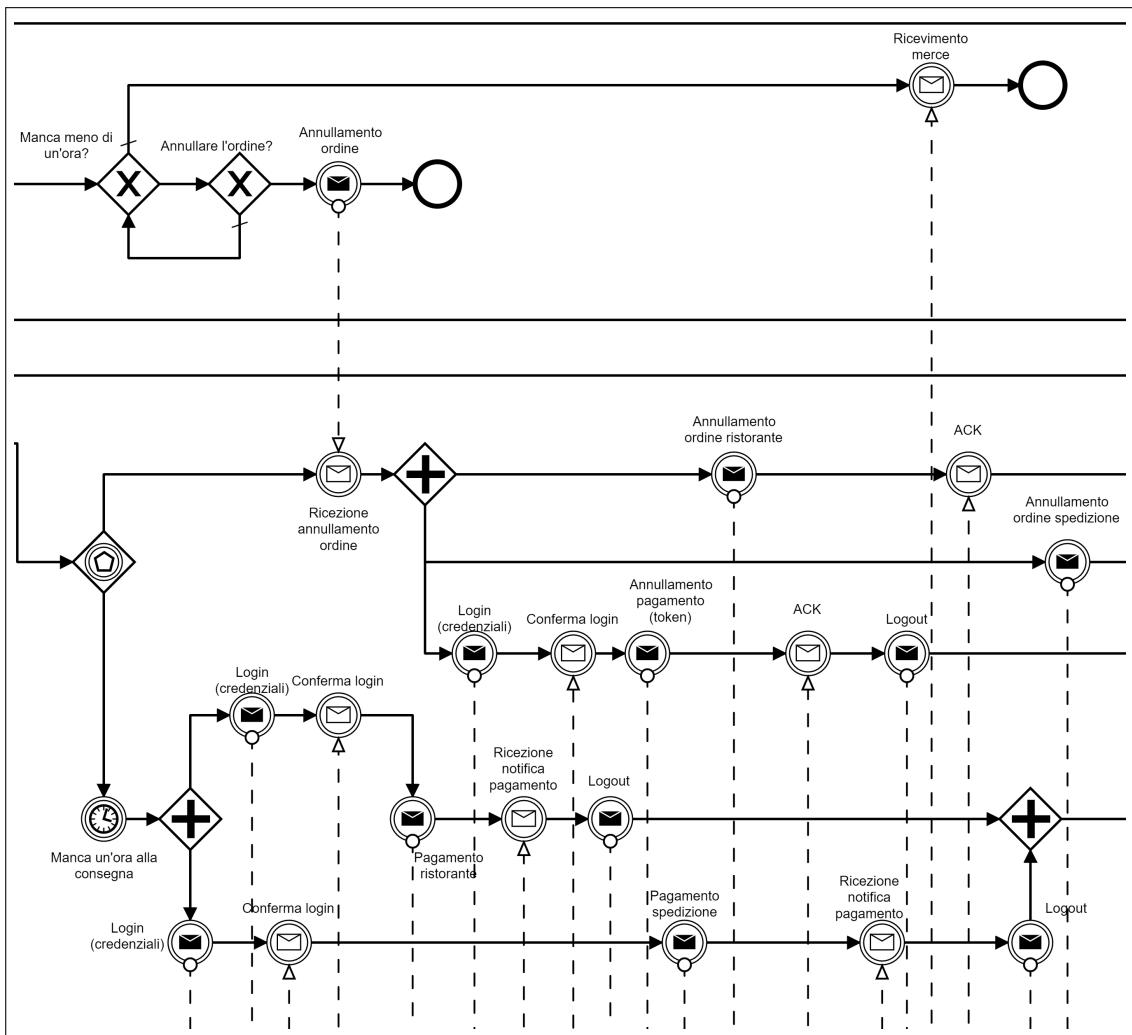


Figura 10: Gestione annullamento ordine da parte del cliente/gestione pagamenti - lato ACMEat

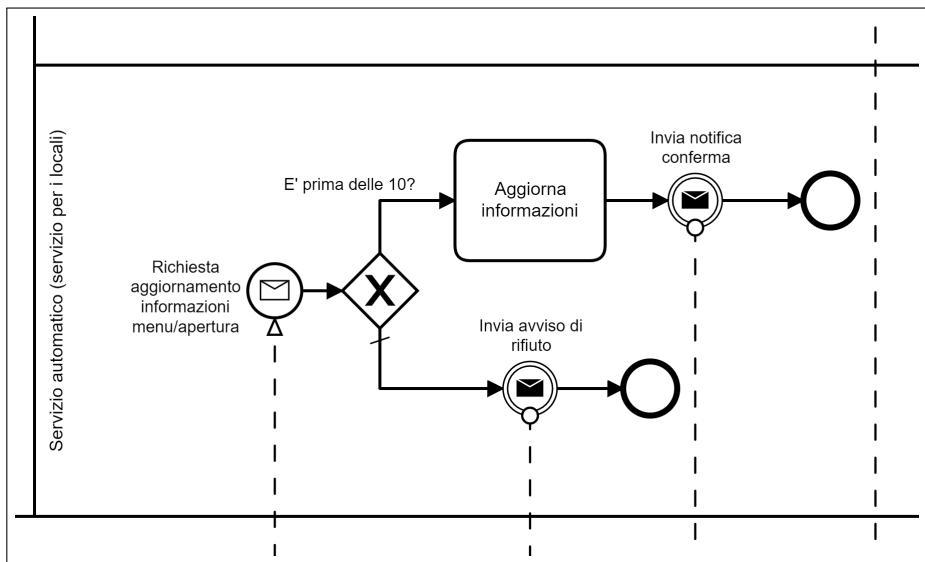


Figura 11: Gestione richiesta modifica informazioni locali - lato ACMEat

4.3 Banca

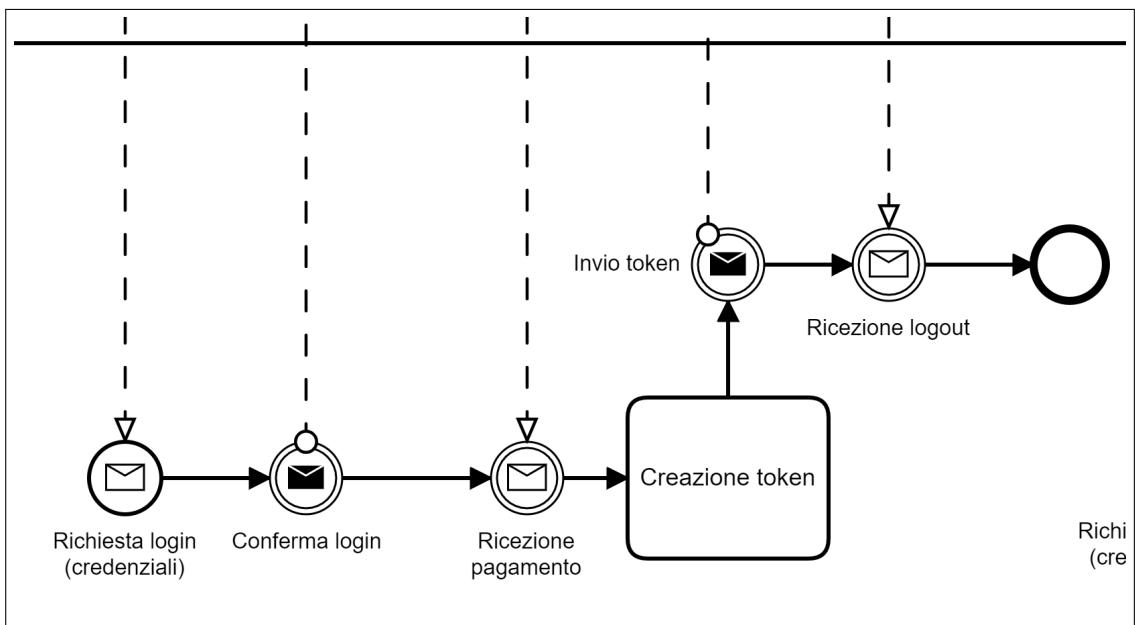


Figura 12: Gestione pagamento cliente - lato banca

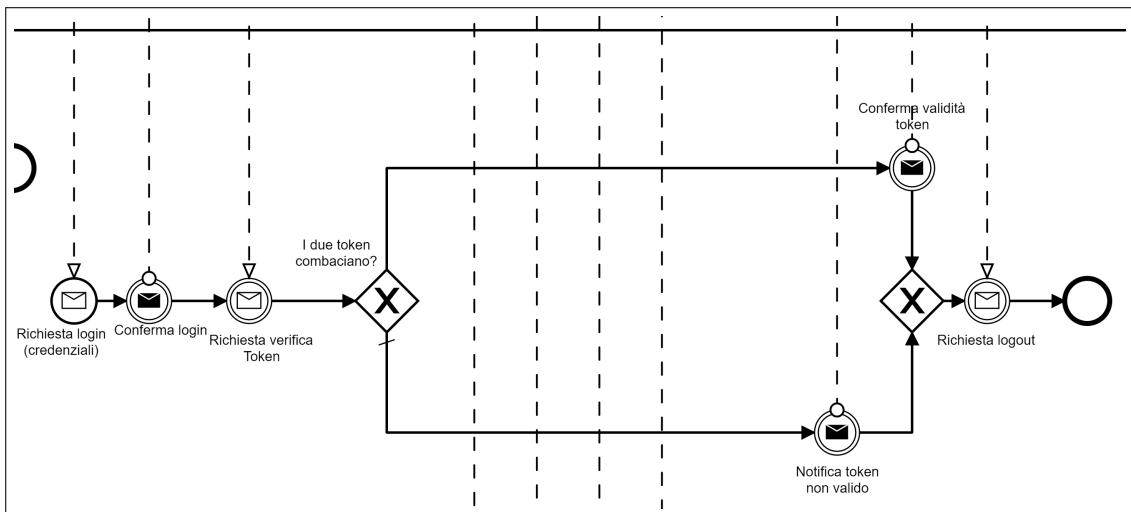


Figura 13: Gestione verifica token - lato banca

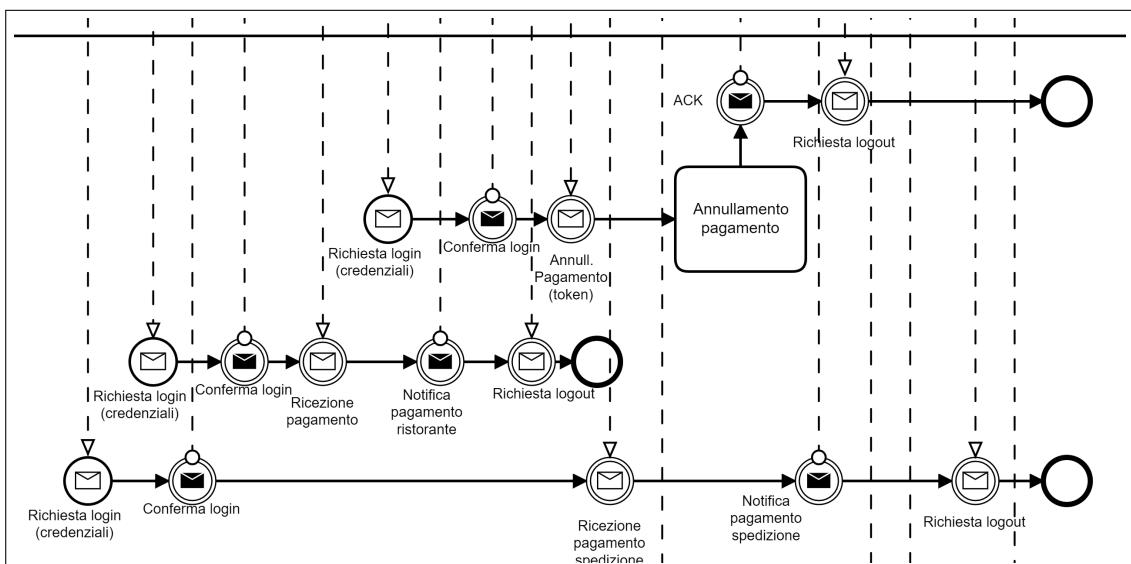


Figura 14: Gestione rimborso/pagamento ristorante e servizio di consegna - lato banca

4.4 Ristorante

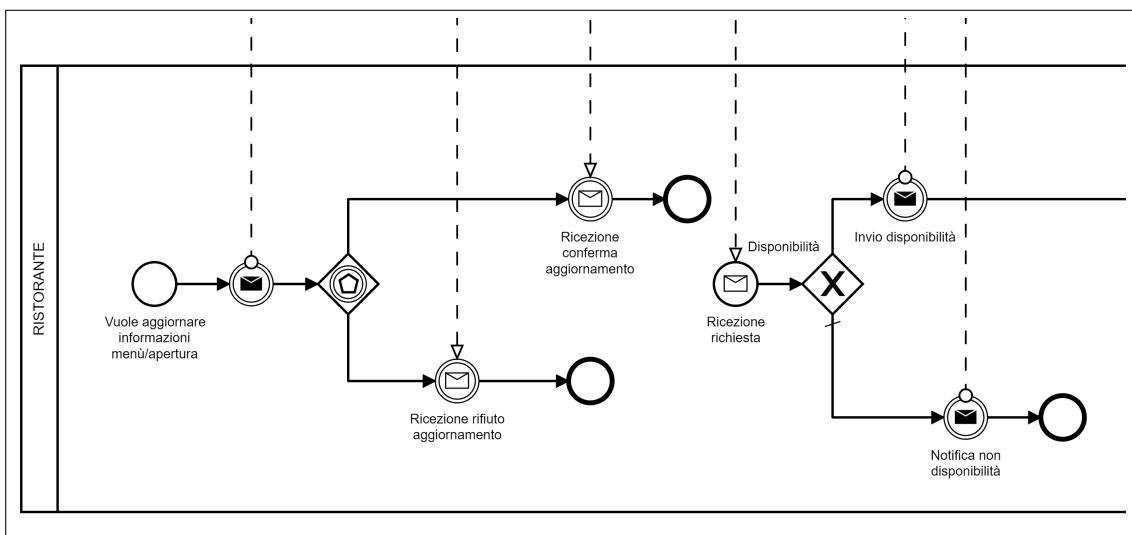


Figura 15: Gestione richiesta modifica informazioni locali/notifica (non) disponibilità - lato ristorante

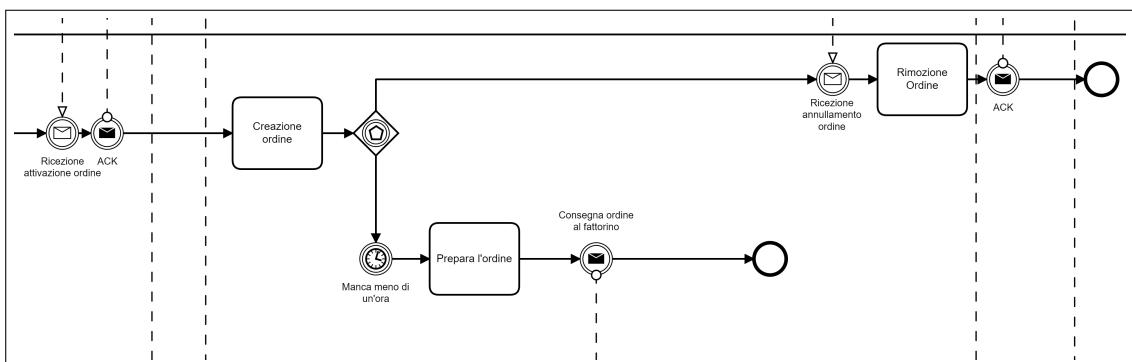


Figura 16: Gestione ordine - lato ristorante

4.5 Società di consegna e fattorino

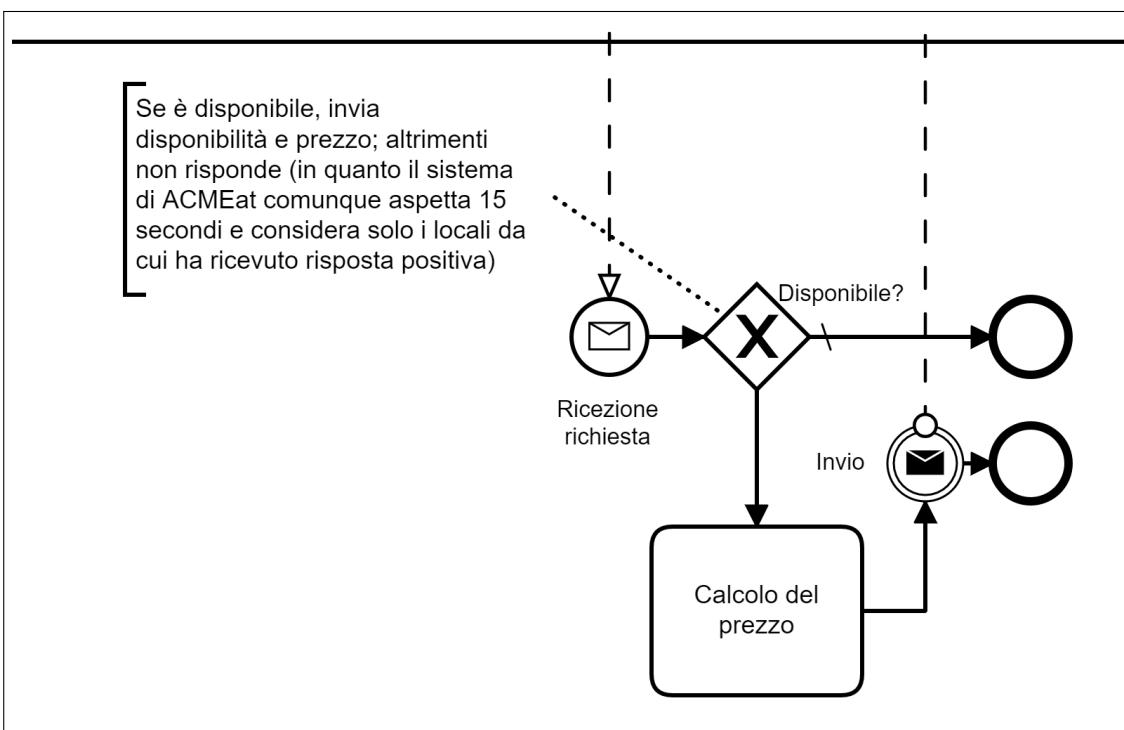


Figura 17: Gestione notifica disponibilità società di consegna - lato società di consegna

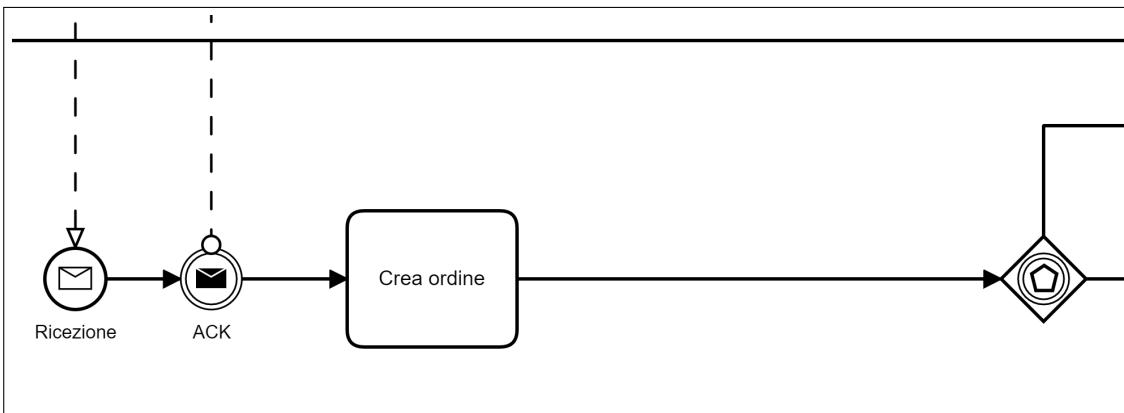


Figura 18: Gestione creazione ordine - lato società di consegna

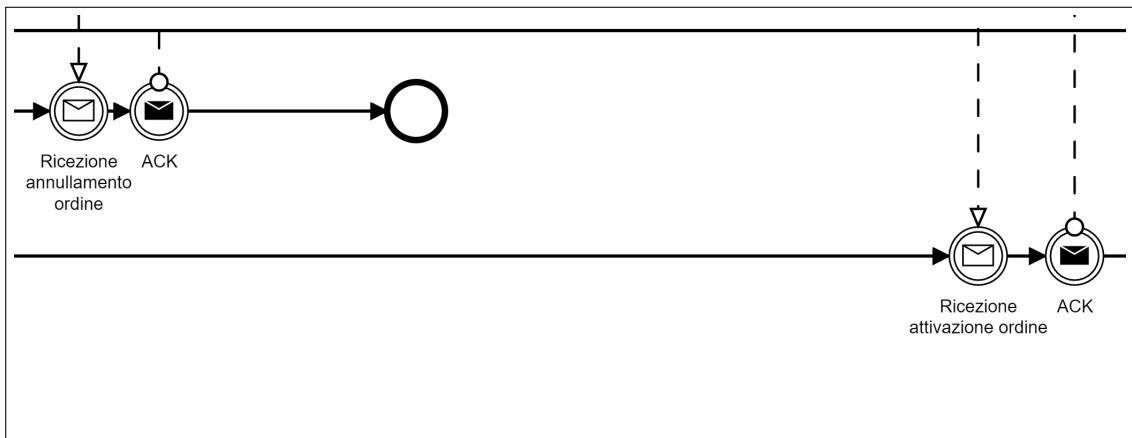


Figura 19: Gestione annullamento/attivazione ordine - società di consegna

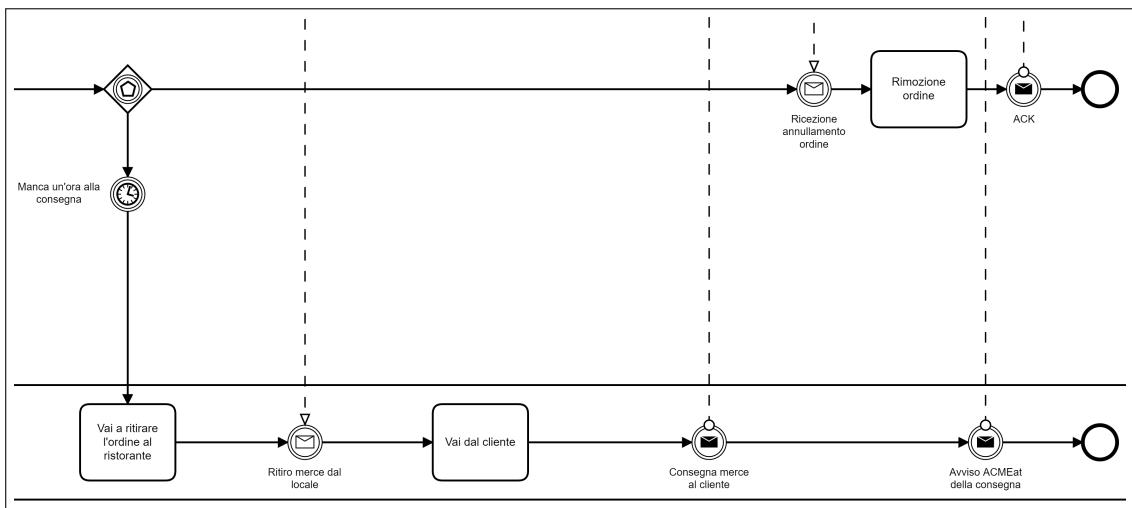


Figura 20: Gestione consegna da parte di società di consegna e fattorino

5 Progettazione

La terza fase di lavoro ha visto la progettazione di una SOA per la realizzazione del sistema, documentata utilizzando UML e utilizzando per il profiling TinySOA. Sono state anche progettate le basi di dati utilizzate dall'applicazione, di cui riportiamo i modelli ER.

5.1 Diagramma UML

Seguendo il profiling TinySOA, sono stati individuati i vari servizi necessari e suddivisi fra *tasks*, *entities* e *utilities*, elencandoli all'interno della dovuta capability e relative interfacce nell'eventuale disponibilità verso l'esterno in base all'attore interessato nel suo utilizzo. Il diagramma è stato realizzato utilizzando **Eclipse Papyrus**.

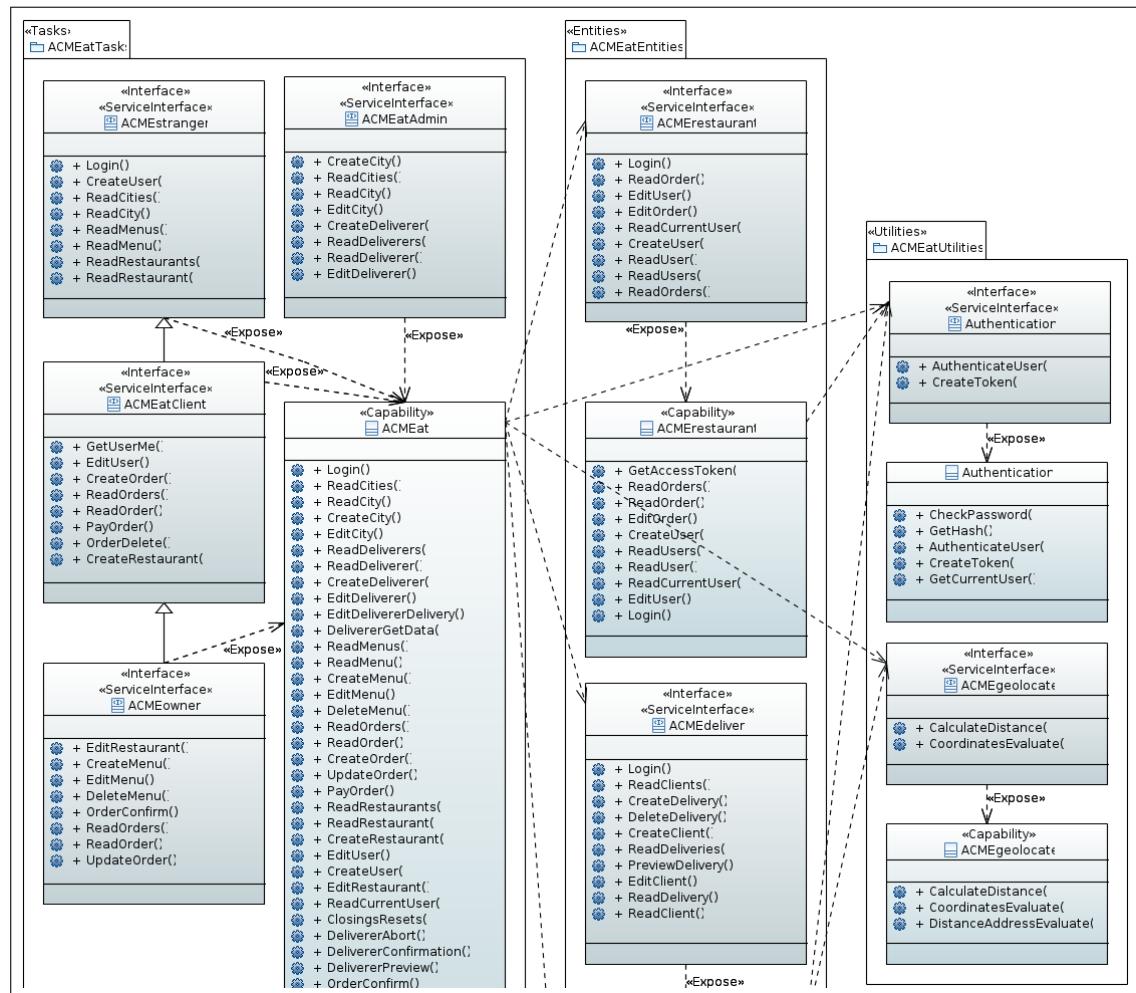


Figura 21: Diagramma UML del sistema (1)

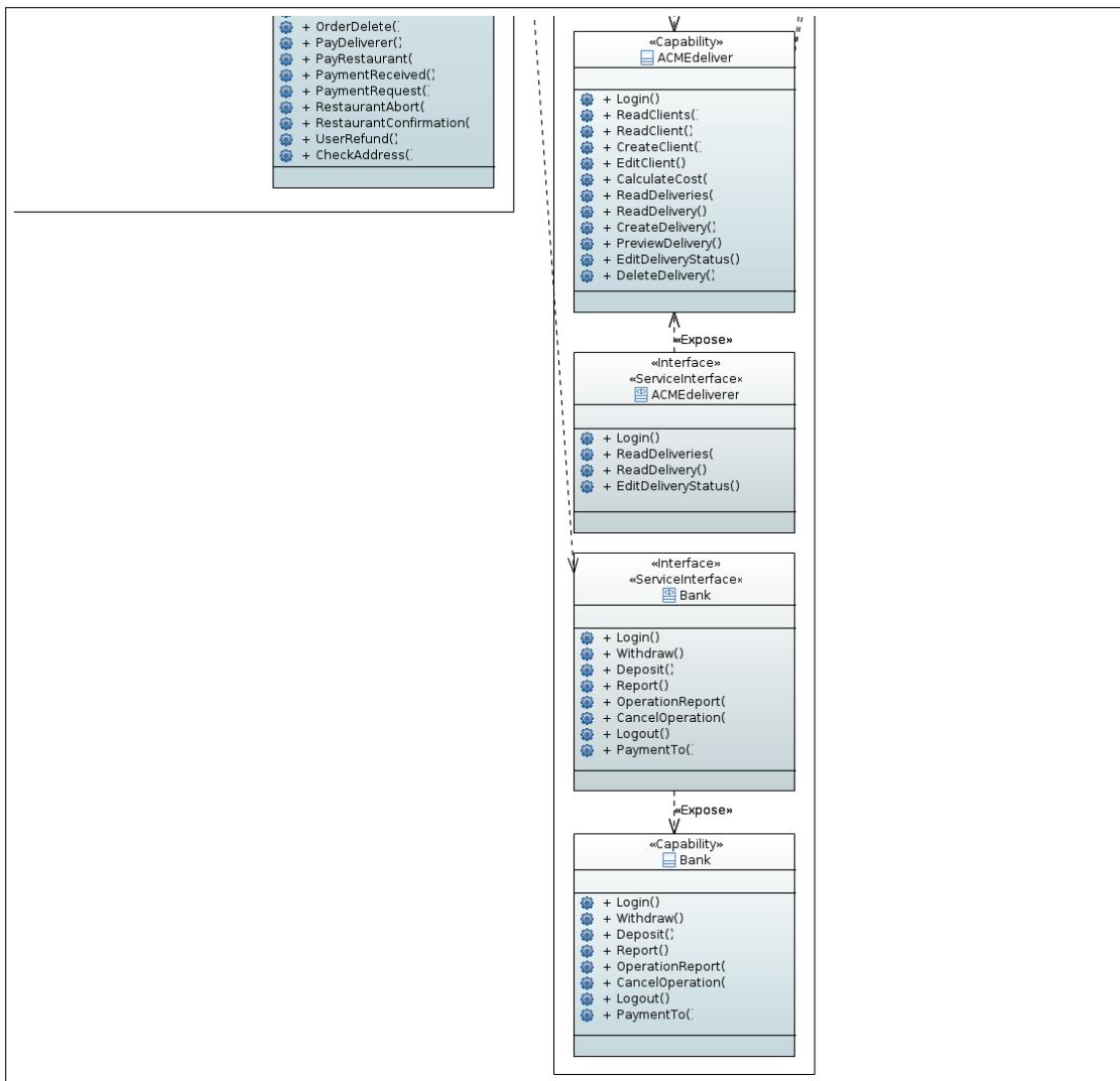


Figura 22: Diagramma UML del sistema (2)

5.2 Diagrammi ER

Di seguito sono riportati i diagrammi ER delle basi di dati utilizzati dal sistema.
 Edit: i modelli collegati ai servizi sviluppati nella fase successiva con API REST sono stati sostituiti con quelli ottenibili direttamente dall'ORM sfruttando l'editor **IntelliJ**, mentre quello della banca è stato creato utilizzando lo strumento online [FigJam](#). Tutte le basi di dati sono state poi realizzate con [PostgreSQL](#).

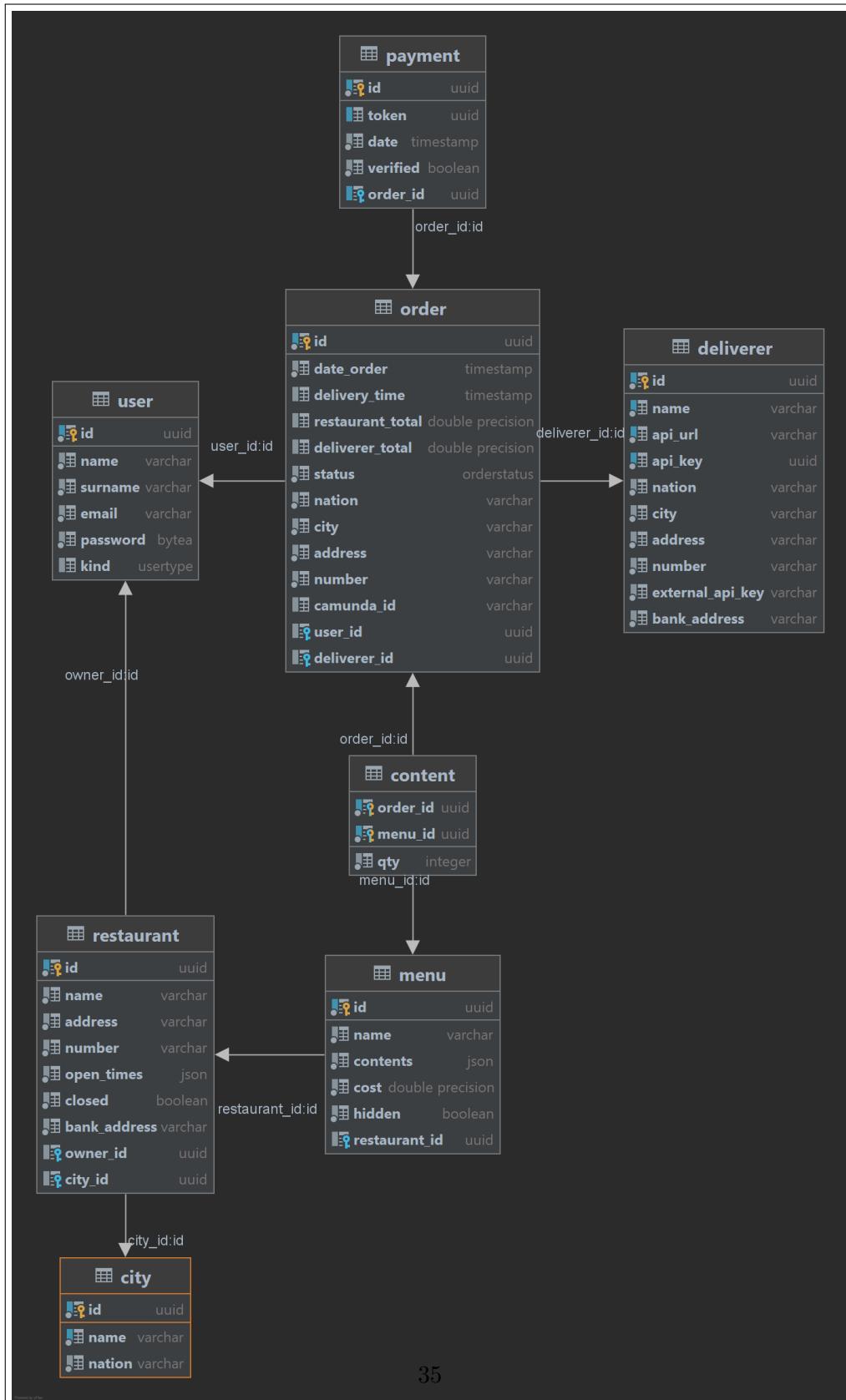


Figura 23: Diagramma ER del database di ACMEat



Figura 24: Diagramma ER del database di ACMErestaurant (a destra) e ACMEDeliver (a sinistra)

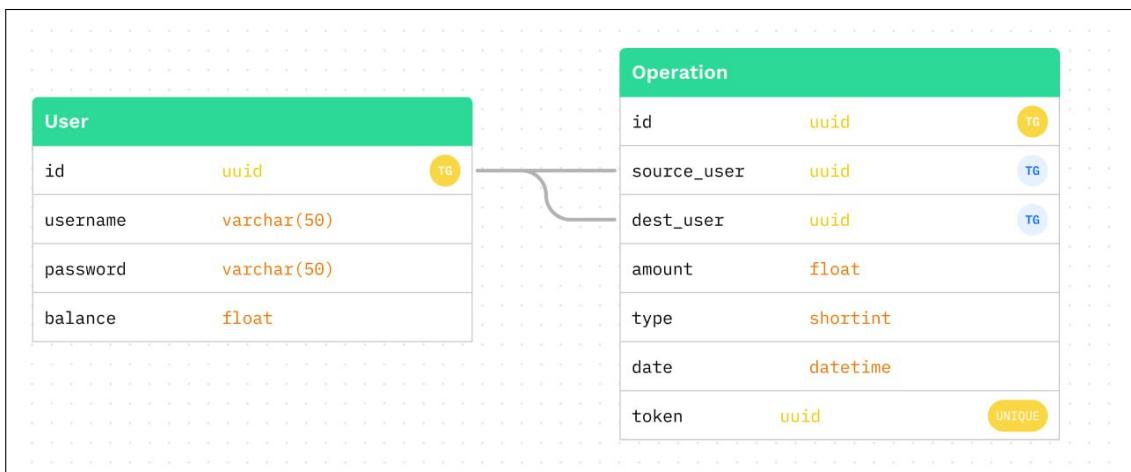


Figura 25: Diagramma ER del database della banca

6 Sviluppo

La quarta fase di lavoro ha visto la realizzazione del sistema. Come da specifica, sono stati realizzati i seguenti servizi:

- Il servizio centrale ACMEat, il quale rende accessibili capabilities realizzate attraverso il BPMS [Camunda](#);
- Il servizio bancario, realizzato in [Jolie](#);
- Il servizio delle società di consegna, denominato ACMEDeliver;
- Il servizio delle società di ristorazione, denominato ACMERestaurant;
- Il servizio di geo-localizzazione, denominato ACMEgeolocate.

Sono stati quindi realizzati i seguenti backends:

- **acmeat**, backend REST per interagire con il database di ACMEat e con il BPMS;
- **acmedeliver**, backend REST che rappresenta una società di consegna e ne consente la gestione e l'utilizzo;
- **acmegeolocate**, backend REST per la geo-localizzazione (implementata appoggiandosi ad [OpenStreetMap](#));
- **acmerestaurant**, backend REST per la gestione degli operatori di un ristorante e che consente l'autenticazione presso ACMEat per la gestione degli ordini;
- **bank**, backend Jolie per la gestione dei servizi bancari, usati per la gestione dei fondi dei clienti, fattorini, ristoranti e di acmeat;
- **bank_intermediary**, backend REST per superare le restrizioni CORS di Jolie, usato solo ed esclusivamente dal frontend della banca.

Tutti i backend (fatta eccezione per bank) sono stati realizzati con le seguenti tecnologie:

- Python 3.8+;
 - fastapi - Framework per la creazione di REST API;
 - bcrypt - Modulo crittografico;
 - beautifulsoup4 - Parsing risposte SOAP;
 - psycopg2-binary - Driver per Postgres;
 - pycamunda - Framework per la comunicazione con Camunda;
 - requests - Framework per la gestione di richieste HTTP;
 - SQLAlchemy - ORM
 - Eventuali dipendenze dei moduli sopra indicati.

- Poetry (Package Manager Python);
- Postgres.

Tutti i dialoghi fra i backends, incluso quello fra Jolie ed il BPMS, avvengono tramite l'uso del protocollo SOAP.

6.1 Diagramma di processo BPMN di ACMEat

Parte delle capabilities del servizio centrale ACMEat sono state rese accessibili attraverso il BPMS Camunda. Come già riportato nella Sez. 4, si è scelto di non rendere eseguibili le pool del modello BPMN in Fig. 3 ma di riferirsi a questo solo a scopo documentativo e sviluppare invece un nuovo modello BPMN eseguibile attraverso BPMS, il cui export è riportato in Fig. 26. Questo modello è tuttavia del tutto consistente con la modellazione a scopo documentativo precedentemente realizzata.

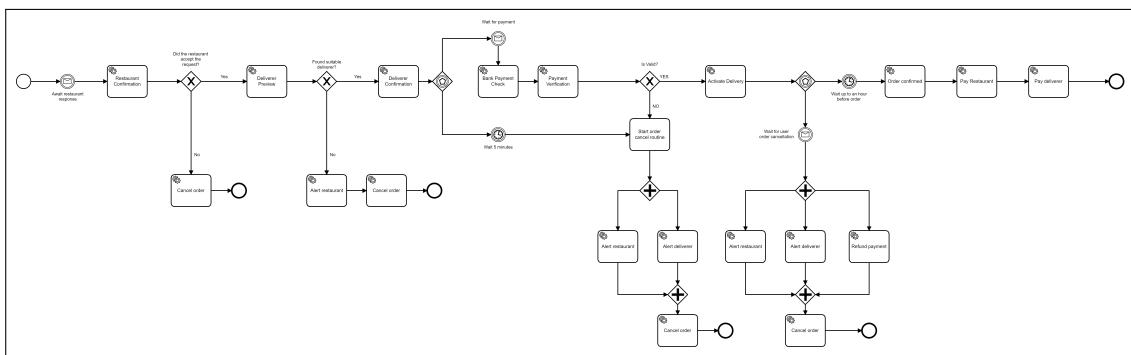


Figura 26: Diagramma di processo BPMN di ACMEat

Come per il modello documentativo, di seguito riportiamo il modello in fig. 26 mettendo in risalto le singole parti.

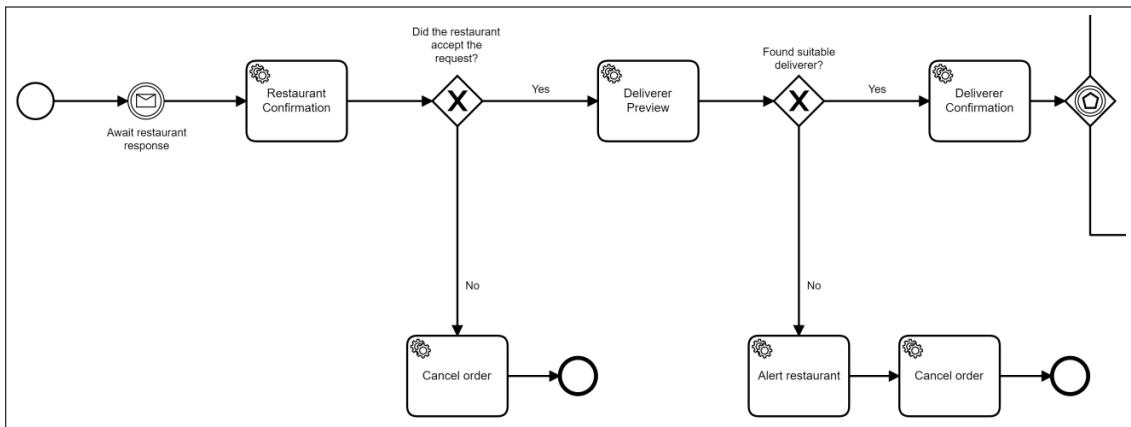


Figura 27: Gestione richieste disponibilità a ristorante e società di consegna

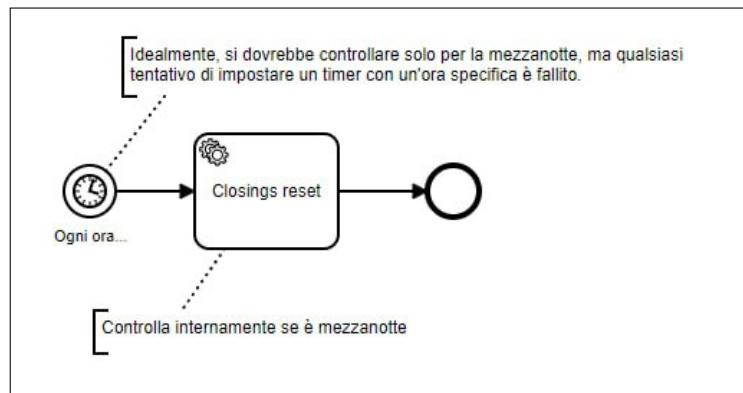


Figura 28: Gestione reset chiusure dei ristoranti a fine giornata

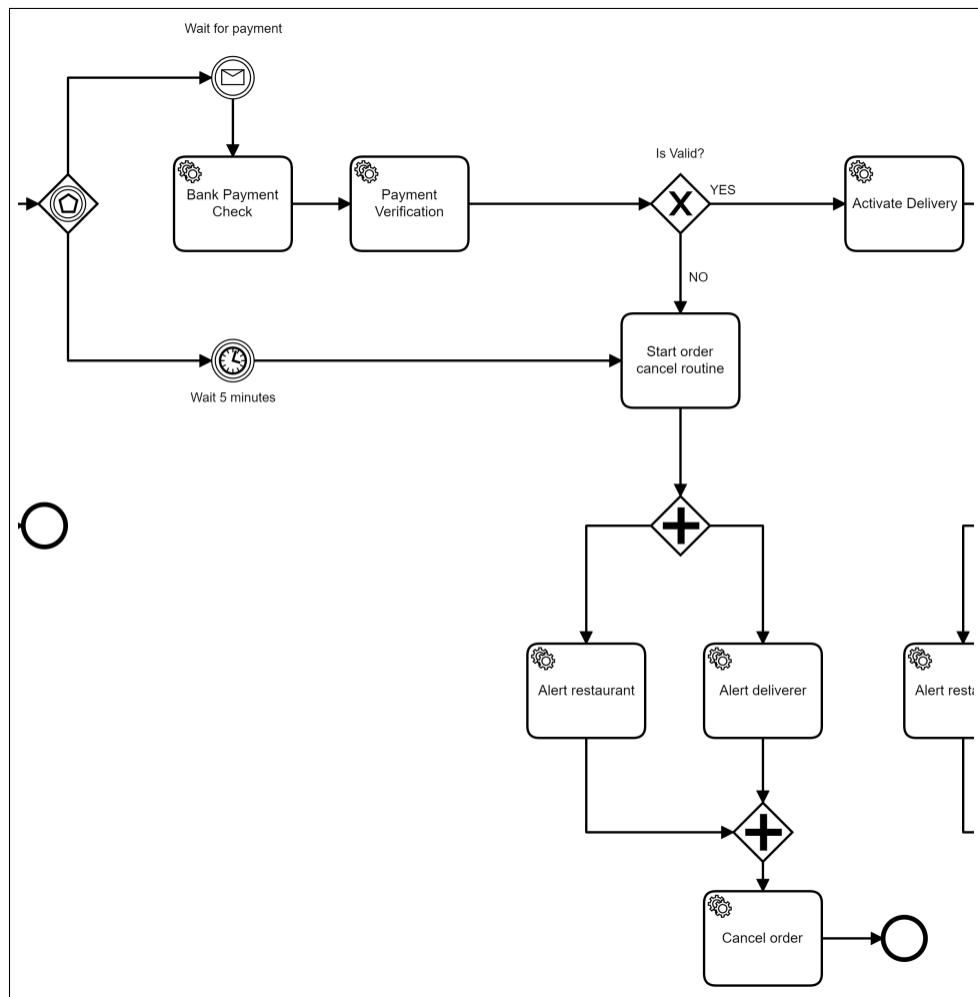


Figura 29: Gestione e verifica pagamento dell'ordine da parte dell'utente

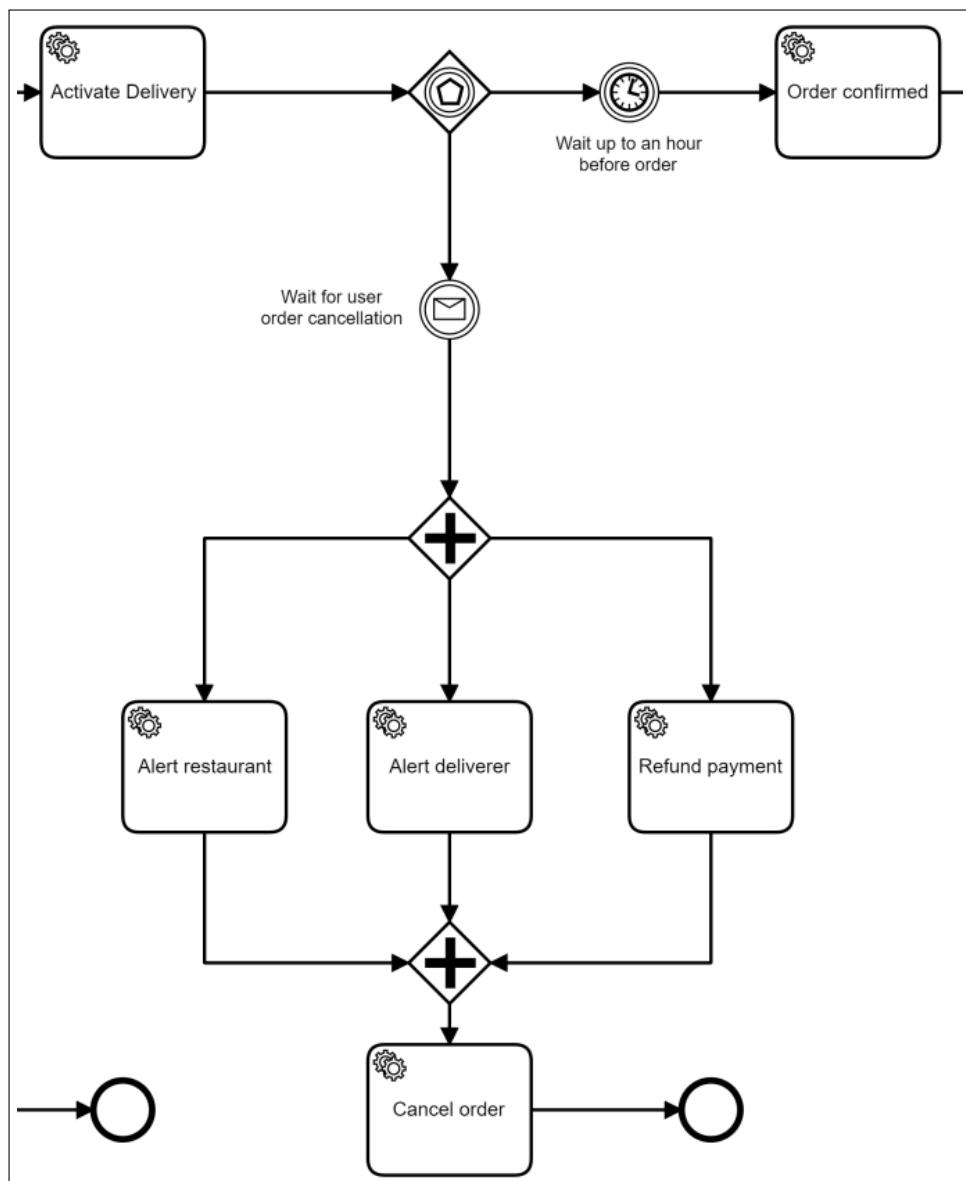


Figura 30: Gestione rimborso e annullamento ordine a ristorante e società di consegna

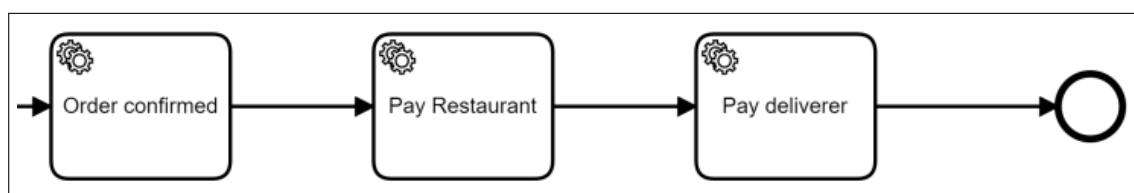


Figura 31: Gestione pagamenti a ristorante e società di consegna dopo conferma ordine

6.2 Banca

I servizi bancari sono stati sviluppati col linguaggio di programmazione Jolie, raffinando la proiezione in Lst. 4 ed implementando alcune funzionalità aggiuntive allo scopo di generalizzarla il più possibile. Come riportato nel diagramma UML in Fig. 22, essa offre infatti verso l'esterno gli stessi servizi in maniera anacronistica. Questi servizi sono:

- Effettuare il login nel sistema, permettendo di avviare una sessione;
- Effettuare il versamento di una somma di denaro;
- Effettuare il ritiro di una somma di denaro;
- Richiedere il report delle operazioni passate;
- Cancellare un'operazione precedente frondendone il relativo token;
- Effettuare un pagamento specificando l'iban (id) del destinatario e la somma da scambiare;
- Effettuare il logout dal sistema, terminandone la sessione.

Tutte queste operazioni possono essere effettuate solo una volta effettuato il login; una volta avviata una sessione, essa rimarrà attiva sino a che non verrà effettuato il logout. Questo nonostante nel sistema sviluppato ogni operazione venga svolta singolarmente, effettuando di volta in volta il login e relativo logout non appena questa è terminata, soprattutto a scopo di sicurezza, come modellato nella coreografia in Lst. 1 e successivamente documentato nel BPMN in Fig. 3. Inoltre, l'applicativo è stato svolto in maniera da accettare invocazioni concorrenti, gestendo le varie sessioni con opportuni sid.

6.2.1 Parametri di configurazione

Per mettere in esecuzione il processo è necessario specificare i seguenti parametri:

- **Location**: "socket://localhost:[porta]" - indirizzo e porta per la comunicazione tramite soap;
- **.username** = "[username]" - username dell'utente postgres per accedere al database della banca;
- **.password** = "[password]" - password dell'utente postgres per accedere al database della banca;
- **.database** = "[nome database]" - nome del database postgres della banca.

Si ricorda inoltre di aggiungere il file del driver **jdbc-postgresql.jar** nell'opportuna cartella **/usr/lib/jolie/lib/**, come da documentazione del linguaggio.

6.3 Backends Python

In questa sezione verranno approfonditi i backends sviluppati in Python, ovvero tutti meno **bank**. La struttura utilizzata all'interno dei vari backend è la medesima (in alcuni, determinate cartelle potrebbero mancare, in quanto un certo componente potrebbe non venire utilizzato), ed è così costituita:

• /

- **database**: contiene la definizione delle tabelle che vengono create all'interno del DB, la definizione di `enum` e l'oggetto `Session` tramite il quale si interroga la base di dati;
- **deps**: contiene le dipendenze di Fastapi, un meccanismo che consente di ottenere informazioni prima di entrare nel corpo della funzione che gestisce un certo endpoint - ad esempio, sapere in anticipo se un utente è autenticato o meno.
- **errors**: contiene la definizione degli errori personalizzati;
- **routers**: contiene le funzioni che gestiscono le richieste ai vari endpoint del backend seguendo la seguente struttura gerarchica di cartelle:
`/api/[soggetto]/v1/[soggetto].py;`
- **schemas**: contiene gli schemi di richiesta e risposta accettati dal backend;
- **services**: contiene il server, il worker e le funzioni necessarie per eseguire le task della coreografia di ACMEat;
- `__main__.py`: runner del server;
- `authentication.py`: modulo per l'autenticazione tramite JWT;
- `configuration.py`: modulo per la configurazione dell'applicazione;
- `crud.py`: modulo che contiene funzioni di utility per la creazione, l'aggiornamento e la ricerca di dati all'interno del database;
- `dependencies.py`: modulo che contiene funzioni da cui dipendono altri moduli dell'applicazione;
- `handlers.py`: gestori di eccezioni specifici per l'applicazione;
- `responses.py`: risposte personalizzate non-json.

6.3.1 Dettagli sui backends

In questa sezione, verranno specificate le caratteristiche dei vari backends. Per informazioni sulle route disponibili per ogni backend, visitare la pagina `/docs` dei backend una volta avviati. Per la documentazione del codice, visionare il sorgente e relativi commenti.

acmeat

acmeat è il backend principale, e consente di:

- Permettere a nuovi utenti di iscriversi al servizio, in veste di cliente o ristoratore;
- Registrare un ristorante e gestirne le caratteristiche, menu inclusi;
- Permettere agli utenti di eseguire ordinazioni presso un locale, e tramite il diagramma di processo BPMN di Camunda gestirne il ciclo di vita;
- Permettere agli amministratori di registrare città in cui il servizio è attivo e gestire la lista dei servizi di spedizione affiliati ad ACMEat.

Tutte le informazioni necessarie vengono immagazzinate all'interno di una base di dati, la cui struttura è riportata in Fig. 23. Il ciclo di vita di un ordine segue le seguenti fasi:

- **Created:** l'ordine è stato appena creato;
- **w_restaurant_ok:** l'ordine è in attesa di conferma da parte del ristorante;
- **w_deliverer_ok:** l'ordine è in attesa di conferma da parte di un servizio di spedizione;
- **confirmed_by_thirds:** l'ordine è stato confermato dalle terze parti (ristorante e servizio di spedizione);
- **cancelled:** l'ordine è stato cancellato dall'utente oppure dal processo Camunda per problemi riscontrati;
- **w_payment:** l'ordine è in attesa di essere pagato. Se non pagato entro 5 minuti, o pagato in modo errato, verrà cancellato;
- **w_cancellation:** l'ordine può venire cancellato dall'utente fino ad un'ora prima dall'orario indicato;
- **w_kitchen:** l'ordine sta venendo preparato in cucina;
- **w_transport:** l'ordine è in attesa del fattorino;
- **delivering:** l'ordine è in consegna;
- **delivered:** l'ordine è stato consegnato.

Gli utenti di **acmeat** possono appartenere ad una fra le seguenti tre categorie:

- **Cliente:** possono solo creare ordini e gestirne i propri, può creare un ristorante (a quel punto il tipo di utente verrà modificato);
- **Ristoratore:** può gestire il proprio locale e crearne di nuovi, oltre ai privilegi del cliente;
- **Amministratore:** può gestire la lista delle città e dei servizi di spedizione.

Le società di consegna accedono ai sistemi di **acmeat** tramite un token, e questo consente loro di aggiornare lo stato dell'ordine (da "in consegna" a "consegnato").

ACMEmanager

ACMEmanager è il server a cui il diagramma di processo BPMN di Camunda delega l'esecuzione dei jobs. Le tasks vengono svolte da un worker, il quale è in ascolto per i seguenti topic:

- **restaurant_confirmation**: ricezione di una conferma (o meno) da parte del ristorante;
- **deliverer_preview**: ottenimento dei preventivi dei servizi di consegna nel raggio di 10km dal locale;
- **deliverer_confirmation**: conferma con la società di consegna di prezzo minore dell'ordine;
- **payment_request**: attesa della ricezione di un pagamento da parte dell'utente;
- **payment_received**: verifica del pagamento ricevuto con la banca;
- **confirm_order**: conferma dell'ordine;
- **restaurant_abort**: notifica annullamento ordine dal lato del ristoratore;
- **deliverer_abort**: notifica annullamento ordine dal lato del fattorino;
- **user_refund**: se pagato, l'ordine viene rimborsato all'utente;
- **order_delete**: l'ordine viene indicato come cancellato;
- **pay_restaurant**: acmeat paga il ristorante;
- **pay_deliverer**: acmeat paga il fattorino.

A questi topic corrispondono funzioni omonime, le quali utilizzano le seguenti variabili di processo:

- **order_id**: l'id dell'ordine interno ad acmeat;
- **success**: flag che indica se l'ultima operazione ha avuto successo o meno;
- **paid**: flag che indica se l'ordine è stato pagato;
- **payment_success**: flag che indica se l'ordine è stato pagato correttamente;
- **TTW**: TimeToWait, durata in secondi alla fine del periodo di cancellazione;
- **found_deliverer**: flag che indica se è stato trovato un fattorino;
- **restaurant_accepted**: flag che indica se il ristorante ha accettato la richiesta.

Il worker è basato su Pycamunda, è multithreaded ed è in grado di interagire con il database tramite SQLAlchemy. Le richieste fatte alla banca vengono trasmesse tramite SOAP.

acmedeliver

acmedeliver è il backend che rappresenta un'azienda di consegne. Permette di:

- Gestire il personale;
- Gestire e ricevere richieste di consegna;
- Gestire la lista dei propri clienti, a cui viene dato accesso;
- Aggiornare il cliente sullo stato della consegna.

Tutte le informazioni necessarie vengono immagazzinate all'interno di una base di dati strutturata come riportato in Fig. 24. Gli utenti di acmedeliver possono essere fattorini oppure amministratori, dove gli amministratori sono in grado di aggiungere fattorini all'azienda.

acmegeolocate

acmegeolocate è il backend che fornisce il servizio di geo-localizzazione. Dato lo stato, la città, la via e il numero civico utilizza OpenStreetMap per ricavarne le coordinate, e in base alla richiesta fornire la distanza tra i due punti in km.

acmerestaurant

acmerestaurant è il backend che fornisce autenticazione alle richieste del ristorante, che comunica poi con acmeat. Gli utenti contenuti all'interno del database non sono utenti di acmeat, ma del ristorante (ad esempio, ogni cameriere può avere un account all'interno del ristorante), e le richieste vengono fatte a nome del titolare del ristorante (che ha un account su acmeat). Il backend consente di eseguire un numero limitato di operazioni, ovvero la lettura degli ordini e accettare/rifiutare/consegnare (ad un fattorino) un ordine. La gestione dei menu, così come degli orari di apertura e delle altre caratteristiche del ristorante è da eseguire su acmeat. La struttura del database è riportata in Fig. 24.

bank_intermediary

Intermediario per superare il blocco CORS di Jolie, si limita a inoltrare le richieste che gli arrivano in *simil-soap* (soap all'interno di un oggetto json) al backend della banca. Viene usato solo ed esclusivamente dal frontend di **bank**.

6.3.2 Parametri di configurazione

Al fine di poter operare, è necessario che ai backend vengano fornite le corrette variabili d'ambiente. Queste sono riportate di seguito.

acmeat

- **JWT_KEY=pippo** - la password con cui i JWT vengono cifrati;

- DB_URI=postgresql://postgres:password@localhost/acmeat - l'uri del database di acmeat;
- BIND_IP=127.0.0.1 - l'indirizzo ip su cui eseguire il binding del socket;
- BIND_PORT=8004 - la porta su cui eseguire il binding del socket;
- BANK_URI=http://127.0.0.1:2000 - l'indirizzo a cui contattare la banca;
- BANK_USERNAME=acmeat - l'username per l'accesso alla banca;
- BANK_PASSWORD=password - la password per l'accesso alla banca.
- GEOFIND_URL=http://127.0.0.1:8001 - l'indirizzo a cui contattare il servizio di geo-localizzazione.

acmedeliver

- JWT_KEY=piotto - la password con cui i JWT vengono cifrati;
- DB_URI=postgresql://postgres:password@localhost/acmedeliver - l'uri del database di acmedeliver;
- BIND_IP=127.0.0.1 - l'indirizzo ip su cui eseguire il binding del socket;
- BIND_PORT=8003 - la porta su cui eseguire il binding del socket;
- PRICE_PER_KM=2 - il costo per chilometro della società di spedizioni;
- GEOFIND_URL=http://127.0.0.1:8001 - l'indirizzo a cui contattare il servizio di geo-localizzazione;
- MAX_DISTANCE=10 - la distanza massima in km del cliente dal locale.

acmegeolocate

- BIND_IP=127.0.0.1 - l'indirizzo ip su cui eseguire il binding del socket;
- BIND_PORT=8001 - la porta su cui eseguire il binding del socket.

acmerestaurant

- JWT_KEY=piotto - la password con cui i JWT vengono cifrati;
- DB_URI=postgresql://postgres:password@localhost/acmerestaurant - l'uri del database di acmerestaurant;
- BIND_IP=127.0.0.1 - l'indirizzo ip su cui eseguire il binding del socket;
- BIND_PORT=8007 - la porta su cui eseguire il binding del socket;

- **ACME_EMAIL=owner1@gmail.com** - l'email del proprietario dell'attività tramite la quale accede ad ACMEat;
- **ACME_PASSWORD=password** - la password del proprietario dell'attività tramite la quale accede ad ACMEat;
- **ACME_RESTAURANT_ID=59294bd6-f61b-48a2-9f53-f76d378b95d9** - l'id del ristorante su ACMEat;
- **ACME_URL=http://127.0.0.1:8004** - l'indirizzo a cui contattare ACMEat.

bank_intermediary

- **BIND_IP=127.0.0.1** - l'indirizzo ip su cui eseguire il binding del socket;
- **BIND_PORT=8006** - la porta su cui eseguire il binding del socket;
- **BANK_URI=http://127.0.0.1:2000** - l'indirizzo a cui contattare la banca.

7 Esecuzione

Scopo di questa sezione è illustrare brevemente come utilizzare l'applicazione in ambiente di testing e su come effettuare un eventuale deployment in produzione.

7.1 Istruzioni per l'avvio in ambiente di testing

1. Installare postgresql, python3 e poetry;
2. Creare un database per l'applicazione che si vuole avviare;
3. Clonare il repository github;
4. Entrare nella cartella **Applications** del repository, eseguire il comando
`poetry install;`
5. Eseguire il comando `poetry shell`;
6. Impostare le variabili d'ambiente richieste dal servizio desiderato;
7. Eseguire il comando `python -m ${service_name}`.

7.2 Istruzioni per il deployment in produzione

In questa sezione, vengono indicati i passaggi necessari per il deployment dell'applicazione in produzione a scopo illustrativo. Si suppone l'utilizzo del sistema operativo Ubuntu, e vengono omessi i passaggi per la realizzazione del reverse proxy e dei certificati per l'https.

Setup iniziale

1. Da root, inserire il comando `useradd ${nome_servizio}`;
2. Da root, creare inserire il comando `adduser user` per creare un utente con cui proseguire la configurazione;
3. Da root, inserire il comando `usermod -aG sudo user` per inserire l'utente user nel gruppo sudoers;
4. Eseguire l'accesso con l'utente user.

Installazione dipendenze software

1. Inserire il comando `sudo apt-get update`;
2. Inserire il comando `sudo apt-get install postgresql python3`;
3. Eseguire il comando `curl -sSL https://install.python-poetry.org | python3` per installare poetry.

Setup del singolo backend

1. Spostarsi nella cartella `/srv` e scaricare il repository git;
2. Spostarsi nella sottocartella del repository **Applications**;
3. Eseguire l'accesso come l'utente `${nome_servizio}`
4. Installare le dipendenze tramite poetry install. Sarà necessario capire quale sia il percorso dell'ambiente creato, il quale dovrebbe essere sotto la cartella `/home/${nome_servizio}/.cache/pypoetry/virtualenvs/` e a cui ci si riferirà come `${poetry_path}`;
5. Eseguire l'accesso con l'utente postgres, eseguire il comando `psql`;
6. Creare il database `${nome_database}`;
7. Creare l'utente `${nome_servizio}` con
`CREATEUSER '${nome_servizio}' WITH ENCRYPTED PASSWORD '${password}';`
8. Fornire all'utente appena creato i privilegi sul database con
`GRANT ALL PRIVILEGES ON DATABASE ${nome_database} TO "${nome_servizio}";`
9. Inserire il comando `exit` 2 volte;
10. Trasferire il possesso della cartella del servizio interessato all'utente `${nome_servizio}` con il comando `sudo chown ${nome_servizio} ${cartella_servizio}`.

Configurazione del server come servizio systemd

1. Creare il file `${nome_servizio}.service` nella cartella `/etc/systemd/system` tramite il comando `sudo touch /etc/systemd/system/${nome_servizio}.service`;
2. Creare la cartella `${nome_servizio}.service.d` nella cartella `/etc/systemd/system` tramite il comando `sudo mkdir /etc/systemd/system/${nome_servizio}.service.d`;
3. Inserire nel file `${nome_servizio}.service` le seguenti righe:

```
1 [Unit]
2 Name=${nome_servizio}
3 Description=${nome_servizio} fastapi server
4 Wants=network-online.target
5 After=network-online.target nss-lookup.target
6
7 [Service]
8 Type=exec
9 User=${nome_servizio}
10 Group=${nome_servizio}
11 # Replace with the directory where you cloned the repository
```

```
12 WorkingDirectory=/srv/ACMEat/Applications/${nome_servizio}/
13 # Replace with the directory where you cloned the repository and the
14     poetry path
14 ExecStart=/home/${nome_servizio}/.cache/pypoetry/virtualenvs/${
15         poetry_path}/bin/python3 __main__.py
15
16 [Install]
17 WantedBy=multi-user.target
```

4. Creare un file nella cartella appena creata chiamato `override.conf` e popolarlo in questo modo, tenendo conto delle variabili d'ambiente necessarie per quel particolare servizio:

```
1 [Service]
2 Environment=KEY=value
```

5. Ricaricare i file di configurazione dei servizi con `sudo systemctl daemon-reload`, per poi avviarlo con il comando `sudo systemctl start ${nome_servizio}`.

7.3 Post-Installazione

Per poter testare l'applicazione senza dover riempire a mano il database, eseguire lo script `post_install.py` nella cartella **Applications**.

8 Conclusioni

In questa relazione è stato descritto il lavoro fatto nelle varie fasi di modellazione e sviluppo del progetto di Ingegneria del Software Orientata ai Servizi, includendo i vari diagrammi prodotti nel processo quali: la coreografia dello scenario ed il relativo sistema di ruoli progettato, nonché il corrispondente diagramma di coreografia BPMN, il diagramma documentativo di processo BPMN ed i diagrammi di progettazione UML. Sono stati inoltre commentati i sorgenti di tutti i servizi, incluso il servizio Jolie relativo ai servizi bancari; tali sorgenti sono stati allegati alla relazione assieme all'export del progetto del BPMS. Per concludere, è stata presentata una breve demo del sistema, comprensiva di istruzioni per l'installazione dell'ambiente di testing, per l'esecuzione, e per un eventuale deployment, a scopo illustrativo.

Scopo del progetto è stato prendere dimestichezza con il workflow, le good practices e le tecnologie usate nell'ambito dell'Ingegneria del Software Orientata ai Servizi affrontate a lezione. Talvolta, si è presentato il bisogno di tornare sui propri passi in modo da rendere coerente la documentazione precedentemente creata con necessità dell'applicazione o dei singoli servizi difficilmente prevedibili; questo allo scopo di utilizzare questi strumenti non solo per la modellazione e progettazione, ma anche controllare formalmente che eventuali modifiche non andassero ad intaccare la correttezza del sistema, e con l'obiettivo che i diagrammi prodotti avessero valore documentativo in futuro.

Per quanto riguarda il sistema ACMEat, numerosi miglioramenti e servizi possono essere implementati in futuro. Siamo tuttavia confidenti nel fatto che la documentazione prodotta possa fornire un ottimo ausilio in tal senso, sia da un punto di vista della sicurezza delle comunicazioni, sia nella semplicità di sviluppo e di successiva integrazione con l'applicazione esistente.