Indice

Componenti del gruppo

Nome	Matricola	Email
Alberto Nicoletti	819697	alberto.nicoletti@studio.unibo.it
Devid Farinelli	819683	devid.farinelli@studio.unibo.it
Filippo Morselli	819508	filippo.morselli@studio.unibo.it

Componenti del gruppo

Nome	Matricola	Email
Alberto Nicoletti	819697	alberto.nicoletti@studio.unibo.it
Devid Farinelli	819683	devid.farinelli@studio.unibo.it
Filippo Morselli	819508	filippo.morselli@studio.unibo.it

Introduzione

Nel progetto sono state implementate tutte le richieste obbligatorie della consegna e la coreografia BPMN opzionale.

Nella realizzazione del progetto sono stati utilizzati i seguenti linguaggi e tool:

- **Camunda modeler** per la modellazione del diagramma di collaborazione BPMN;
- **Lucidchart** per la modellazione del diagramma di coreografia BPMN;
- **Camunda Wildfly** server;
- **Jolie** e **Java** per i servizi SOAP;
- **Maven** per la gestione dei progetti Java;
- **Node.js** per i servizi REST;
- **Eclipse** o **IntelliJ Idea** per compilare i progetti Java.

Scripts

Dentro la cartella `scripts` sono presenti anche gli script da eseguire prima di compilare alcuni progetti Java:

- `ext_wsdl2java`, per generare le classi Java a partire dai wsdl dei servizi esterni;
- `ws_wsdl2java`, per generare le classi Java a partire dai wsdl esposti dal web service attivo su Wildfly (il web service deve essere attivo).

Esecuzione

Per eseguire il progetto:

1. Aprire il progetto `acme-model` con un IDE e compilarlo, eseguire il seguente comando per aggiungere il .jar generato nella repository Maven locale:

   ```
   $ mvn install:install-file
       -Dfile=<path to acme-model.jar>
       -DgroupId=org.loopingdoge.acme.model
       -DartifactId=acme-model
       -Dversion=1.0.0
       -Dpackaging=jar
       -DgeneratePom=true
   ```

2. Avviare il server Wildfly;

3. Da terminale andare nella cartella `wildfly-10.1.0.Final/bin` del server Wildfly, eseguire lo script `jboss-cli.sh`, fare `connect` ed eseguire il seguente comando `module add –name=org.loopingdoge.acme.model –resources=<path to acme-model.jar>` per aggiungere `acme-model` come modulo del server;
4. Aprire con un IDE il progetto `acme-agency-ws`, compilare il progetto e mettere il file .war generato nella cartella `wildfly-10.1.0.Final\standalone\deployments` del server Wildfly;
5. Eseguire gli script `scripts/ws_wsdl2java` e `scripts/ext_wsdl2java`;
6. Installare le dipendenze dei progetti **Node.js** tramite `npm install` nella directory dei progetti (`src/distance` e `src/mail`);
7. Definire una variabile d'ambiente di nome `MAPS_API_KEY` dando come valore una chiave da generare a questo indirizzo;
8. Eseguire `scripts/start_services` per avviare i servizi esterni;
9. Aprire con un IDE il progetto `acme-agency`, compilare il progetto e mettere il file .war generato nella cartella `wildfly-10.1.0.Final\standalone\deployments` del server Wildfly;
10. Aprire con Eclipse i progetti `src/java-buyer` e `src/java-seller`, ed eseguire le istruzioni a riga di comando;
11. Aprire con un browser questo indirizzo, loggare usando `john` come *username* e *password*, ed aprire la tasklist, effettuando gli user task quando necessario.

Coreografia

```
(
  (
    request: a → acme;                                          // Richiesta dall'acquirente

    (
      ( askDist: acme → dist; replyDist: dist → acme )* ;       // Ciclo di proposta case
       proposal: acme → a; proposalReply: a → acme
    )*;

    1
      +
    (
      (
        askAvail: acme → v;                                     // Accordo sulla disponibilita'
        availReply: v → acme;
        sendMeetingProposal: acme → a;
        meetingProposalReply: a → acme;

        1
          +
        (
          sendMeetingProposal: acme → v;
          meetingProposalReply: v → acme
        )
      );

      (
        meetDenies: acme → v | meetDenies: acme → a             // No accordo, no incontro
      )
        +
      (
        (
          meetAgree: acme → v | meetAgree → acme → a            // Incontro fatto
        );

        (
          (
            1                                                   // Richiesta di prestito opzionale
              +
            (
              loanRequest: a → bank;
              loanReply: bank → a
            )
          );

          (                                                     // No offerta dopo l'incontro
            noOffer: a → acme;                                  // o dopo prestito rifiutato
            noOffer: acme → v
          )
            +
          (
            offer: a → acme;                                    // Offerta
```

```
(
  (
    (
      askCada: acme → cada;                    // Richiesta al catasto
      replyCada: cada → acme
    );

    1                                          // Ok
      +                                        // oppure
    (
      askCada: acme → cada;                    // Risistemo indirizzo
      replyCada: cada → acme;

      1                                        // Ok
        +                                      // oppure
      sendAgent: acme → cada                   // Invio personale
    )
  )
    |                                          // parallelamente
  (
    askDist: acme → dist;                      // Richiesta distanza notaio
    replyDist: dist → acme
  )* ;                                         // Per ogni notaio in lista
);

offer: acme → v;                               // Proposta al venditore

(
  (
    denyOffer: v → acme                        // Venditore rifiuta
    vendorDenied: acme → a
  )
    +
  (
    agreeOffer: v → acme;                      // Venditore accetta
    vendorAgreed: acme → a;
    (
        pay: a → bank;                         // Invia la caparra
        confirmPayment: bank → a
    )*;
    notifyPayment: bank → v;

    (
      signContract: a → nota | signContract: v → nota
    );

    (
        contractDone: nota → a
        |
        contractDone: nota → acme
        |
        contractDone: nota → v
    );
```

```
                (
                    pay: a → bank;                          // Pagamento ad acme
                    confirmPayment: bank → a
                )*;
                notifyPayment: bank → acme;

                (
                    pay: a -> bank;                         // Pagamento al venditore
                    confirmPayment: bank → a
                )*;
                notifyPayment: bank → v
              )
            )
          )
        )
      )
    )
  )
    |
  (
    offer: v → acme                                         // Offerta di casa dal venidtore
  )
)
```

Coreografia in BPMN [(sources)]

Le coreografie BPMN delle istanze di processo avviate in risposta alle richieste di acquirente e venditore sono state realizzate usando Lucidachart come editor online.

Analisi di Connectedness

La coreografia riportata sopra gode della proprietà di connectedness. Di seguito si effettua una analisi dettagliata per dimostrarlo:

- Le due funzioni principali di Acme, ossia la vendita di una casa e l'inserimento di una nuova casa, sono in parallelo e non richiedono quindi particolari condizioni
- L'iterazione della proposta di case a un acquirente non presenta problemi dato che le sequenze al suo interno sono del tipo $a \rightarrow b$; $b \rightarrow a$ e la condizione di terminazione è data dal contenuto di *proposalReply*
- La choice seguente dipende anch'essa dal contenuto di *proposalReply* e la decisione è presa da Acme
- La fase di accordo sulla data dell'incontro è corretta, dato che tutte le sequenze sono del tipo $a \rightarrow b$; $b \rightarrow c$
- La choice per l'esito dell'accordo sull'incontro è ok, in quanto la decisione è presa da Acme e entrambi i rami coinvolgono gli stessi ruoli (che saranno in attesa su *meetAgree* e *meetDenies*)
- La choice seguente è corretta, dato che la decisione coinvolge solamente l'acquirente e la banca è sempre in attesa per una *loanRequest*
- La choice per una eventuale offerta è ok, in quanto la decisione è presa dall'acquirente e entrambi i rami coinvolgono gli stessi ruoli (che saranno in attesa su *offer* e *noOffer*) e anche il venditore viene informato sull'esito

- Le choice riguardanti il catasto sono corrette, dato che le decisioni sono prese da Acme in base al contenuto di *replyCada*
- La choice riguardo l'esito dell'offerta coinvolge gli stessi attori come primo messaggio di entrambi i rami, e in seguito entrambi informano l'acquirente
- La fase di invio della caparra è corretta, dato che tutte le sequenze sono del tipo $a \to b \ ; \ b \to c$
- Potenziale problema di connectedness tra la firma del contratto e il successivo pagamento, anche se in un contesto concreto l'azione in parallelo (ossia la firma del contratto da parte di acquirente e venditore) avviene contemporaneamente. Si aggiunge una *contractDone* per evitare ciò.

Coreografia proiettata sui ruoli

Indice

Notazione

Si usa il simbolo @ per l'invio di messaggi e # per la ricezione.

Acme

```
(
  (
    request#a;                      // Richiesta da un acquirente

    (
      askDist@dist;                 // Richiesta al servizio delle distanza
      replyDist#dist;
      proposal@a;                   // e propone possibili case d'interesse
      proposalReply#a
    )* ;

    1                               // Skip --- la reply era una interruzione
      +
    (
      (
        askAvail@v;                 // Richiesta disponibilità
        availReply#v;
        sendMeetingProposal@a;
        meetingProposalReply#a;

        1
          +
        (
          sendMeetingProposal@v;
          meetingProposalReply#v
        )
      );

      (
        meetDenies@a                // Non incontro
          |
        meetDenies@v
      )
        +
      (
        (
          meetAgree@a               // Incontro
            |
          meetAgree@v
        );

        (
          noOffer#a;                // Dopo l'incontro nessuna offerta da a
          noOffer@v                 // Lo comunica anche a v
        )
          +
        (
          offer#a;                  // Offerta da a

          (
            (
              (
                askCada@cada;       // Richiesta coordinate
```

```
                replyCada#cada
              );

              1                    // Skip --- era nel formato corretto
                +
              (
                askCada@cada;
                replyCada#cada;

                1                  // Skip --- correzione accettata
                  +
                sendAgent@cada     // Invia dipendente
              )
            )
              |
            (
              askDist@dist;        // Richiede distanza notaio
              replyDist#dist
            )
          );

          offer@v;                 // Proposta al venditore

          (
            (
              denyOffer#v;         // Venditore rifiuta
              vendorDenied@a       // Si informa l'acquirente
            )
              +
            (
              agreeOffer#v;        // Venditore accetta
              vendorAgreed@a;
              contractDone#nota;
              notifyPayment#bank
            )
          )

        )
      )

    )

  )
    |
  (
    offer#v                        // Offerta da un venditore
  )
)*
```

Acquirente

```
(
  request@acme;                       // Effettua richiesta

  (
    (
      proposal#acme;                  // Riceve proposte case
      proposalReply@acme              // Risponde alla richiesta
    )*
  );

  (
    1                                 // Skip --- ha interrotto la richiesta di case
      +                               // oppure

    (
      sendMeetingProposal#acme;       // Risposta alle date di meeting del venditore
      meetingProposalReply@acme
    );

    (
      meetDenied#acme                 // Skip --- incontro riufiutato
    )
      +
    (
      meetAgree#acme;                 // Incontro accettato

      (
        1                             // Skip --- no richiesta prestito
          +
        (
          loanRequest@bank;           // Richiesta prestito
          loanReply#bank
        )
      );

      (
        (
          noOffer@acme                // Rifiuta o nessuna offerta
        )
          +
        (
          offer@acme;                 // Offerta

          (
            vendorDeny#acme           // Ricevo un rifiuto del vendor
          )
            +
          (
            vendorAgreed#acme;        // Il vendor ha accettato, finalizzo
            (
                pay@bank;             // Pagamento della caparra
                confirmPayment#bank
            )*;
```

```
            signContract@nota;
            contractDone#nota;
            (
                pay@bank;                 // Pagamento ad acme
                confirmPayment#bank
            )*;

            (
                pay@bank;                 // Pagamento al venditore
                confirmPayment#bank
            )*
          )
        )
      )
    )
  )
)
```

Venditore

```
(
  (
    (
      askAvail#acme;                  // Riceve da Acme una richiesta di quando e' disponibile
      availReply@acme;                // dato che un acquirente e' interessato a una casa

      1                               // L'acquirente accetta
        +
      (
        sendMeetingProposal#acme;     // L'aquirente ha proposto nuove date
        meetingProposalReply@acme
      )
    );

    (
      meetDenied#acme                 // Skip --- incontro riufiutato
    )
      +
    (
      meetAgree#acme;                 // Incontro accettato

      (
        noOffer#acme                  // Dopo l'incontro non riceve offerte
      )
        +
      (
        offer#acme;                   // Riceve offerta

        (
          denyOffer@acme              // Rifiuta offerta
        )
          +
        (
          agreeOffer@acme;            // Accetta offerta
          sendDeposit#bank;
          signContract@nota;
          contractDone#nota;
          notifyPayment#bank
        )
      )
    )
  )
    |
  (
    offer@acme                        // Propone casa all'agenzia
  )
)
```

Banca

```
(
  (
    loanRequest#a;            // Richiesta di mutuo
    loanReply@a
  )
    |
  (
    pay#a;                    // Richiesta di effettuare un pagamento
    confirmPayment@a;
    (
        1                     // Skip -- pagamento non andato a buon fine
        +
        notifyPayment@dest
    )
  )
)*
```

Catasto

```
(
  askCada#acme;               // Riceve richiesta per coordinate
  replyCada#acme;             // Risponde alla richiesta, anche richiedendo correzioni

  (
    1                         // Skip --- la richiesta era corretta
      +
    (
      askCada#acme;           // Riceve la richiesta aggiustata
      replyCada#acme;         // Risponde, anche richiedendo intervento dipendente

      (
        1                     // Skip --- la correzione era valida
          +
        sendAgent#acme        // Riceve il dipendente dell'agenzia
      )
    )
  )
)*
```

Distanze

```
(
  askDist#acme;           // Riceve richiesta di distanza
  replyDist@acme
)*
```

Notaio

```
(
    (
        signContract#a
        |
        signContract#v
    );
    (
        contractDone@a
        |
        contractDone@v
        |
        contractDone@acme
    )
)*
```

BPMN (sources)

Il BPMN è composto da una pool eseguibile per l'agenzia ACME, una pool per ogni client (acquirente e venditore) ed una per ciascuno dei servizi esterni (Catasto, Distanza, Banca)

Servizi esterni

Vengono in seguito presentate le peculiarità dei servizi esterni ad acme che sono stati realizzati per implementare la coreografia. Per maggiori informazioni su ognuno rimandiamo alle relative pagine Github (raggiungibili dai link presenti sui titoli) in cui sono descritte più approfonditamente le richieste possibili, i loro tipi, e le modalità di esecuzione.

Banca (sources)

Come da specifiche, la banca è stata implementata in Jolie e comunica attraverso il protocollo SOAP.

Sono state implementate le operazioni di:

- `login`
- `logout`
- `deposito di denaro`
- `ritiro di denaro`
- `pagamento`
- `richiesta di mutuo`
- `report`

Non tutte queste operazioni sono usate nella coreografia.

Prima di effettuare una qualsiasi operazione, il servizio della banca richiede in primo luogo di effettuare una operazione di `login`, che restituirà un `sid` da inviare tra i parametri delle chiamate successive. Per implementare questa sessione sono state usati **correlation set** nativi di Jolie.

Per rispettare la coreografia in seguito ad un'operazione di pagamento, la banca invia una mail al destinatario del pagamento attraverso il servizio delle mail.

Catasto (sources)

Il catasto è un semplice servizio Jolie che dispone di una singola operazione `cadastrialCoordinates`, che, data una stringa in formato CSV così composta: `road;civic;city;cap;province;state`, restituisce le coordinate catastali (per semplicità calcolate come interi random tra 1 e 10000).

Un esempio del formato della risposta è il seguente:

```
<coordinates>
    <nord xsi:type="xsd:string">0</nord>
    <east xsi:type="xsd:string">0</east>
</coordinates>
<error xsi:type="xsd:string">Incorrect address format</error>
```

Infatti, essendo il catasto molto esigente sul formato dell'indirizzo, qualora questo non rispetti il formato richiesto, le coordinate verranno settate a 0 e verrà specificato un messaggio di errore.

Distanze (sources)

Il servizio delle distanze è stato realizzato come servizio REST in Node.js ed opera con indirizzi reali utilizzando le API di Google Distance Matrix.

Il servizio prevede due parametri GET, l'indirizzo di partenza e quello di arrivo, e restituisce una risposta così formata con un HTTP status code 200:

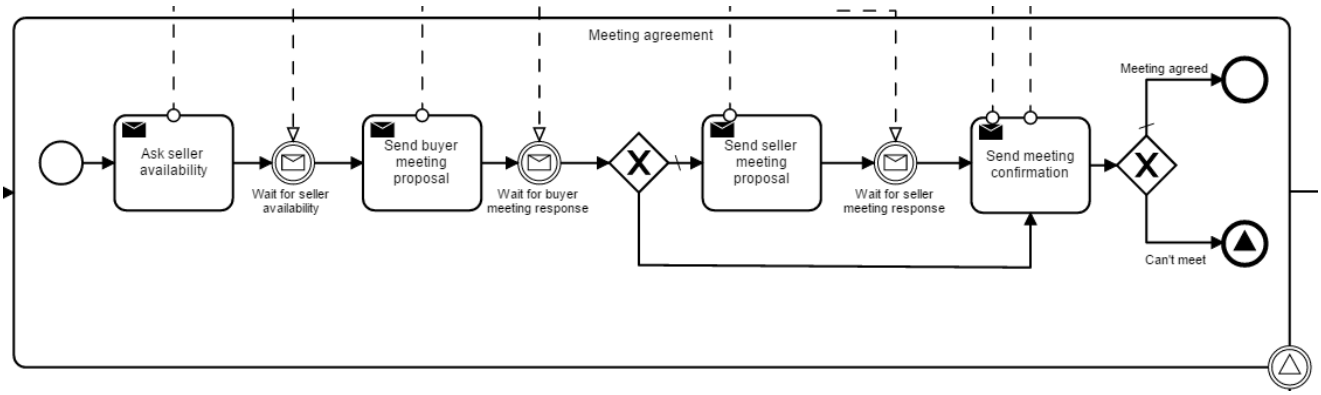
```
{
  "message": "A message describing the result",
  "distance": "The distance from origin to destination in meters"
}
```

Nel caso ci fossero dei problemi con gli indirizzi, viene resituito un HTTP status code 400 ed una risposta vuota.

Mail (sources)

Questo servizio non era richiesto dalle specifiche, ma lo abbiamo realizzato per utilizzarlo come metodo di comunicazione generico nei casi in cui un'alternativa più specifica, come un web service, non ci sembrasse necessaria.

Ad esempio, nel subprocess di meeting agreement, non ci sembrava realistico che i messaggi verso buyer e seller avvenissero tramite SOAP (che avviene invece per le loro risposte), quindi abbiamo scelto di inviare i messaggi attraverso questo servizio.



Il servizio delle mail è realizzato in REST tramite Node.js ed espone una risorsa `/{username}`, verso la quale è possibile fare richieste:

- **GET**, per ottenere la lista delle mail ricevute da tale utente
- **POST**, per inviare una mail a tale utente

Per ulteriori informazioni sull'interfaccia e le operazioni consigliamo di consultare il file swagger.json attraverso l'editor di swagger, con il quale è possibile anche provare le richieste.

Sessioni (sources)

Servizio che tiene traccia di tutte le istanze di processo attive. Esso permette ai client (acquirente e venditore) di interrogarlo allo scopo di recuperare le informazioni riguardo le proprie sessioni in corso. Ad essi vengono infatti restituiti gli ID di istanze di processo in cui ci si aspetta una loro azione per poter completare il successivo task. Il client, dopo aver scelto quale istanza continuare, utilizza il *processId* corrispondente come parametro per le comunicazioni al web service di Acme.

I dati sulle sessioni sono invece inseriti e rimossi dall'agenzia Acme.

Il servizio espone pertanto due interfacce distinte, ed è realizzato in Jolie.

Fanno da eccezione a questa separazione le operazioni di *informDepositDone* e *informPaymentDone*, che permettono a
un client di modificare il proprio stato nel processo notificando direttamente questo servizio.
Il motivo di questa scelta è dato dal fatto che Acme non è coinvolta direttamente nella gestione delle caparre e del
pagamento al venditore, pertanto non sarebbe in grado di aggiornare lo stato dell'acquirente in modo opportuno,
se non a fase di pagamento terminata.

Agenzia ACME

L'agenzia è stata implementata tramite un file BPMN eseguibile e due web service usati come interfaccia per i client.

Model (sources)

Le classi condivise utilizzate come modello per i dati sono state aggregate in un file `.jar` ed incluse come dipendenze Maven di entrambi i progetti.

Web services (sources)

I web services sono stati realizzati utilizzando JAX-WS ed espongono le interfacce utilizzate dai client di acquirente e venditore per scambiare informazioni con il BPMS. Di seguito sono descritti gli enpoint per ciascuno:

Acquirente (sources)

- `requestHouses`
  avvia una istanza di processo per l'acquirente e restituisce una lista di case disponibili in linea con i criteri di ricerca
- `houseProposalReply`
  permette di accettare una casa, richiederne altre o terminare la ricerca
- `getSellerMeetingDateList`
  restituisce l'elenco delle date in cui il venditore è disponibile per mostrare la casa
- `replyToMeetingProposal`
  permette di accettare o rifiutare le date di incontro proposte dal venditore
- `makeOffer`
  permette di avanzare un'offerta di acquisto per una casa
- `getChosenHouse`
  restituisce i dati della casa scelta dall'acquirente

Venditore (sources)

- `proposeHouse`
  permette di mettere in vendita una casa ed avvia un'istanza di processo per il venditore
- `sendAvailability`
  permette di comunicare ad ACME in quali date il venditore è disponibile per mostrare la casa
- `getBuyerMeetingDateList`
  restituisce l'elenco delle date di incontro proposte dall'acquirente
- `confirmMeeting`
  permette di confermare o rifiutare le date di incontro proposte dall'acquirente
- `getOffer`
  permette di visionare un'offerta di acquisto
- `offerReply`
  permette di accettare o rifiutare un'offerta di acquisto

BPMS (sources)

Il BPMS è semplicemente un progetto Maven che implementa i task specificati dal BPM ed è organizzato come segue:

- Services, contiene le classi Java che implementano la logica dei diversi task
- Utils, contiene le interfacce per i servizi REST ed i database delle case e dei notai
- Forms, contiene i form utilizzati per gli user task in camunda

Per semplicità i database sono stati implementati come classi java statiche.

Clients

Entrambi i client sono realizzati con programmi Java e, seppur con interfaccia utente elementare, permettono di realizzare tutte le operazioni ad essi richieste.

Si interfacciano con ACME tramite i web service esposti dall'agenzia e, nel caso dell'acquirente, anche con quello della banca.
Il codice che effettua le richieste ai servizi è generato automaticamente tramite `wsimport`, a partire dai WSDL dei servizi necesssari (si vedano gli script `ext_wsdl2java` e `ws_wsdl2java.bat` nella apposita directory del progetto).

I client sono stati realizzati in modo tale da non esporre servizi, quindi si utilizza il servizio di Mail per inviare ad essi notifiche.

Inoltre utilizzano il servizio delle Sessioni per continuare istanze di processo anche in esecuzioni differenti.

Buyer (sources)

Implementa le seguenti operazioni:

- **House lookup**: richiesta di una casa da poter acquistare, specificando un profilo di interesse
- **House proposal reply**: azione che effettua in seguito ad aver ricevuto una possibile lista di case, fra chiederne altre, accettarne una di quelle proposte o interrompere la ricerca
- **Reply to meeting proposal**: accettare una data proposta dal venditore per incontrarsi, oppure proporne di nuove
- **Make offer**: effettuare un'offerta per una casa di interesse
- **Loan request**: richiesta di prestito alla banca
- **Pay**: pagamento a un venditore o ad Acme, tramite la banca

Seller (sources)

Implementa le seguenti operazioni:

- **House proposal**: richiesta di aggiunta di una nuova casa a quelle in vendita da Acme
- **Send availability**: invio delle date possibili per un incontro con l'acquirente
- **Confirm meeting**: accettazione o rifiuto di una delle date per l'incontro proposte dall'acquirente
- **Get offer**: richiesta delle informazioni riguardo un'offerta ricevuta per una casa
- **Offer reply**: accettazione o rifiuto di un'offerta ricevuta per una casa