Relazione del Progetto di Ingegneria del Software Orientata ai Servizi

Alessio Bortolotti
Università degli Studi di Bologna
Dipartimento di Informatica
alessio.bortolotti@studio.unibo.it

Sebastian Davrieux
Università degli Studi di Bologna
Dipartimento di Informatica
sebastian.davrieux@studio.unibo.it

Francesco Trombi
Università degli Studi di Bologna
Dipartimento di Informatica
francesco.trombi@studio.unibo.it

AA. 2015/2016

Abstract

La seguente relazione espone le soluzioni progettuali ed implementative utilizzate durante la realizzazione del progetto finale del corso di Ingegneria del Software Orientata ai Servizi. Il progetto consiste nell'implementazione di una SOA che simuli il funzionamento di un'azienda (chiamata ACME) che si occupa della vendita e dell'assemblaggio di componenti per cicli. Si sono poi implementati come capability esterne alcuni servizi, integrati con i processi tramite protocolli standard. Per soddisfare le specifiche proposte, sono state utilizzate tre tecnologie principali: Signavio, Camunda e Jolie.

Introduzione

Il progetto consiste nell'implementazione di una *Service Oriented Architecture* (SOA) che rispecchi le necessità di un'azienda fittizia ma con processi verosimili.

L'introduzione di tale relazione descrive brevemente il dominio del problema e i vincoli imposti alla soluzione.

Il primo capitolo si concentra sulla presentazione degli attori coinvolti nei processi implementati.

Il secondo capitolo riguarda la descrizione dello schema BPMN utilizzato per la progettazione dei processi che compongono la soluzione.

Il terzo capitolo mostra i dettagli implementativi della soluzione, soffermandosi dunque su aspetti più tecnici.

Le conclusioni mostrano una panoramica sulle problematiche riscontrate ed un'appendice contenente qualche informazione aggiuntiva sulle scelte progettuali adottate.

Dominio del problema

L'azienda presa come esempio è la *ACME*, che si occupa di fornire cicli assemblati oppure componenti atomiche. La ACME si basa su un sistema di magazzini, così organizzato: esiste un magazzino primario, adibito alla coordinazione di un insieme di magazzini secondari. Il magazzino primario contiene anche l'officina, per l'assemblaggio dei cicli.

La ACME riceve dunque ordini dai clienti, con i quali interagisce durante diverse fasi: l'accertamento delle compatibilità delle eventuali customizzazioni, l'accettazione del preventivo, il pagamento dell'acconto e del saldo. Gli ordini contengono una serie di particolari, quali modello base, la colorazione e le eventuali customizzazioni per gruppo frenante, gruppo sterzo, trasmissione e ammortizzatori.

L'azienda deve interagire con una ditta di spedizioni per consegnare gli oggetti ai clienti. Normalmente le spedizioni partono dal magazzino primario; ma nel caso l'ordine contenga solamente componenti che non richiedono assemblaggio, è possibile spedire direttamente dal magazzino più vicino al cliente, separatamente. Quando si procede all'assemblaggio, si riservano le componenti necessarie e le si spostano, se necessario, nel magazzino primario; se non presenti vengono ordinate presso un fornitore esterno, unico. Il costo dell'ordine viene calcolato tramite costo delle componenti e costo di spedizione.

I pagamenti che il cliente effettua verso ACME vengono verificati tramite un istituto di credito. Diversi particolari non ancora indicati verranno mostrati in seguito insieme all'implementazione adottata.

Vincoli

I vincoli imposti alla soluzione sono stati i seguenti:

- implementazione della gestione dei magazzini tramite SOA di servizi Jolie;

- esternalizzazione di alcune capability, quali il calcolo delle distanze geografiche, l'istituto di credito e il fornitore delle componenti;
- utilizzo di *API REST* per l'implementazione dei servizi di calcolo delle distanze geografiche e dell'istituto bancario;
- utilizzo di logica elementare per l'implementazione dei servizi, in modo da simulare comportamenti verosimili.

Descrizione della soluzione

L'implementazione della soluzione è stata affrontata basandosi sulle tre componenti principali richieste. Si sono dunque progettati ed implementati:

- lo schema BPMN del processo principale dell'azienda ACME, utilizzando il tool di progettazione grafica *Signavio*;
- il processo principale dell'ACME utilizzando un sistema BPMS, *Camunda*;
- la gestione dei magazzini, tramite una SOA di servizi Jolie;
- le capability esterne, sotto forma di servizi, utilizzando Jolie.

1 Attori

Come prima cosa è necessario definire con chiarezza quali sono i vari "attori" che formano il sistema della soluzione. Si procederà dunque con una panoramica generale su di essi.

1.1 ACME

L'attore principale della soluzione è l'azienda ACME: data però la dimensione di tale entità si è ritenuto utile suddividerla in diversi attori, che corrispondono ai diversi dipartimenti dell'azienda.

1.1.1 Ufficio

L'ufficio è l'attore che modella il processo principale dell'azienda ACME, occupandosi dell'interazione con il cliente, con il magazzino primario (e dunque con i magazzini secondari) e con l'istituto di credito. Le modalità di interazione con gli altri attori verrà descritta con precisione nella sezione relativa alla progettazione dello schema BPMN.

1.1.2 Magazzino

I magazzini possono essere di due categorie: primario e secondario. I magazzini vengono identificati tramite un `id` numerico univoco, assegnato nella seguente modalità: il magazzino primario ha `id = 0`, e i magazzini secondari vengono numerati progressivamente a partire da questo. In questo progetto sono stati implementati un magazzino primario (dunque, `id = 0`) e due magazzini secondari (`id = 1, 2`).

Il magazzino primario funge da coordinatore dei magazzini secondari, ponendosi dunque come intermediario tra l'ufficio ed i magazzini secondari. Il magazzino primario si occupa inoltre di tutte le funzioni dell'officina e del reparto che si occupa della preparazione delle spedizioni. Il magazzino primario comunica inoltre con la ditta di spedizione e con il fornitore delle componenti.

1.1.3 Officina

Come detto in precedenza, l'officina è contenuta all'interno della stessa struttura del magazzino primario. È la sede delle operazioni di assemblaggio delle componenti e dei cicli. All'interno della nostra soluzione, è un attore "fittizio", poiché le sue funzionalità vengono completamente implementate all'interno delle operazioni del magazzino primario.

1.2 Cliente

Il cliente è l'attore che inizia il processo principale dell'azienda ACME. Come l'officina, non gode di un'implementazione effettiva, ma viene simulato tramite richieste e risposte nelle interazioni con l'ufficio. Un cliente può essere di due

tipologie diverse: un privato o un rivenditore. Tale distinzione non è però rilevante dal punto di vista dell'implementazione.

1.3 Fornitore

Il fornitore è l'attore che si occupa della fornitura delle componenti non presenti nei magazzini. Se durante la ricerca di una componente nei magazzini essa non viene trovata, viene mandata una richiesta al fornitore, che si occuperà di spedirlo al magazzino principale. Come scelta, si consideri che il fornitore avrà sempre le componenti richieste dal magazzino disponibili. La gestione di altre operazioni non viene considerata.

1.4 Istituto di credito

L'istituto di credito è un attore che deve simulare la verifica della solvibilità di un cliente, in base ad un bonifico, identificato da un numero univoco. Le interazioni avvengono solamente con l'ufficio. L'implementazione simula in maniera semplice la generazione di un esito, tramite un sistema implementato nel seguente modo: viene selezionato un numero casuale, e se esso è minore di `0.3`, l'istituto restituisce `false`; altrimenti restituisce `true`. Si è preferito ridurre le possibilità che uscisse un esito negativo per rendere più verosimile la soluzione. Non sono previste altre operazioni.

1.5 Ufficio legale

L'ufficio legale è un attore fittizio, senza implementazione. Il suo unico ruolo è quello di ricevere le notifiche di mancato pagamento da parte dell'ufficio. Non sono previste operazioni che coinvolgano ulteriormente tale attore.

2 Diagramma BPMN

Il diagramma BPMN è stato realizzato utilizzando il software di modellazione online *Signavio*. Sono state modellate diverse *pool*: la principale, in chiaro, è quella dell'azienda ACME, suddivisa al proprio interno in tre *lane*, corrispondenti all'ufficio, alla gestione del magazzino ed all'officina. Sono state poi modellate alcune *collapsed pool*, che rappresentano processi di organizzazioni esterne alla ACME: il cliente, il fornitore, l'istituto di credito, l'ufficio legale e la ditta di spedizioni.

2.1 Descrizione del processo ACME

Il processo dell'azienda ACME da noi descritto viene iniziato dalla ricezione di un ordine, inviato dalla *collapsed pool* del cliente. L'ordine viene catturato da un *message start event*. Ricevuto l'ordine, viene controllata la presenza di customizzazioni, tramite un *exclusive gateway*: in caso compaiano, viene fatta una valutazione sulla compatibilità di tali customizzazioni. Questa valutazione porta ad un altro *exclusive gateway*, che corrisponde alla decisione di acccettazione o rifiuto delle customizzazioni. Nel caso non vengano accettate come compatibili, viene eseguito un *message end event*, che corrisponde all'invio al cliente di una notifica di annullamento dell'ordine. In caso le customizzazioni vengano accettate, si passa ad un altro *exclusive gateway*, che smista gli ordini nelle due categorie previste: ordini che contengono cicli e ordini che contengono componenti. Questo passaggio serve a determinare il costo di spedizione che verrà addebitato al cliente.

In caso l'ordine contenga un ciclo, il prezzo di spedizione viene calcolato prendendo come parametri l'indirizzo del cliente e quello del magazzino primario: questo perché i cicli vengono assemblati all'interno del magazzino primario.

Nel caso l'ordine contenga delle componenti, si effettuerà una ricerca all'interno di tutti i magazzini (utilizzando la lane della gestione magazzini), per ogni componente. Un *exclusive gateway* mappa i due possibili esiti: se la componente viene trovata in uno o più magazzini, il costo di spedizione si baserà sull'indirizzo del cliente e quello del magazzino più vicino al cliente che contiene la componente; nel caso la componente non venga trovata, il costo di spedizione viene calcolato in base all'indirizzo del cliente e a quello del magazzino primario, dato che il fornitore spedisce solamente a quello. Le componenti vengono riservate, qualora si trovino in un magazzino, ma non vengono ordinate al fornitore nel caso non ci siano.

Finita la fase di definizione dei costi di spedizione, il sistema compone un preventivo, sommando ad esse i prezzi degli oggetti ordinati. Nel caso il preventivo superi una certa cifra x, un membro dell'ufficio decide l'applicazione di uno sconto. Questo meccanismo viene mappato da un exclusive gateway. Il preventivo finale viene poi spedito al cliente e l'ufficio si mette in attesa con un *event based gateway*. Gli *intermediate event* mappati sono due: o un *catching message intermediate event*, che corrisponde alla risposta del cliente, oppure un *timer intermediate event*, che corrisponde alla scadenza di un timer fissato ad

un tempo *y*.

Se scade il timer, si comunica la magazzino di rilasciarele componenti e si arriva ad un *message end event*, che corrisponde all'invio della notifica di annullamento dell'ordine al cliente. Nel caso il cliente invii una risposta, un *exclusive gateway* mappa le due possibili risposte: accettazione o rifiuto del preventivo. Nel caso di rifiuto, si attua la stessa procedura descritta nel caso della scadenza del timer.

L'ufficio si mette poi in attesa dell'acconto dal cliente, con un *timer boundary event*, che avvia la procedura di annullamento dell'ordine vista in precedenza, dopo la scadenza di un tempo *y*. L'acconto viene mandato dal cliente sotto forma di un bonifico: se tale bonifico è inferiore ad un decimo della somma indicata dal preventivo finale, viene avviata la procedura di annullamento vista in precedenza; altrimenti il bonifico viene mandato alla *collapsed pool* dell'istituto di credito.

In base all'esito sulla verifica della solvibilità del bonifico comunicato dall'istituto di credito, l'ufficio decide se annullare l'ordine seguendo la procedura indicata in precedenza oppure procedere. Questo meccanismo viene implementato da un *exclusive gateway*.

Ricevuto l'esito positivo dall'istituto di credito, un *exclusive gateway* consente di smistare gli ordini nelle due categorie già incontrate: cicli e componenti. Il processo viene dunque spostato nella *lane* della gestione del magazzino.

Nel caso l'ordine contenga un ciclo, si controlla la presenza delle componenti nel magazzino primario, una per volta. Un *exclusive gateway* mappa le due possibilità: o la componente viene trovata, oppure non viene trovata. Se non viene trovata si cerca all'interno dei magazzini secondari: anche in questo caso un *exclusive gateway* mappa i due risultati possibili. Se viene trovata la si spedisce al magazzino primario, altrimenti la si ordina. Viene dunque mandato un ordine alla *collapsed pool* del fornitore, il quale provvede a spedire la componente al magazzino primario. Quando tutte le componenti sono arrivate al magazzino primario, si procede alla fase di assemblaggio, nella *lane* dell'officina. L'officina assembla il ciclo, prepara la spedizione e si affida alla ditta di spedizioni.

Nel caso l'ordine contenga delle componenti si verifica tramite un *exclusive gateway* che siano già state riservate; nel caso non sia così, le componenti vengono ordinate al fornitore, con la procedura vista in precedenza. Quando tutte le componenti sono presenti, anche se sparse in diversi magazzini, vengono preparate alla consegna e date alla ditta di spedizioni.

Quando all'officina arriva la notifica di consegna avvenuta, catturata da un *catching message boundary event*, il processo torna all'ufficio.

L'ufficio riparte dall'attesa del saldo, affidandosi ad un *event based gateway*. Gli eventi che l'ufficio attende sono due: un *catching message intermediate event*, che rappresenta l'arrivo del saldo dal cliente, oppure un *timer intermediate event*, che rappresenta la fine di un'attesa fissata ad un tempo *y*. Se scade il timer, viene inviata una notifica all'ufficio legale, per poi terminare il processo. Se invece arriva il saldo dal cliente, lo si invia all'istituto di credito, aspettando l'esito. Un *exclusive gateway* basato sull'esito della verifica del bonifico indirizza il processo a due possibili terminazioni: se l'esito è negativo, si invia una notifica

all'ufficio legale e poi si termina; se l'esito è positivo, il processo termina senza ulteriori passaggi.

3 Implementazione

Questo capitolo tratta della descrizione dell'implementazione adottata per la soluzione in analisi.

3.1 BPMS: Camunda

Il meccanismo di interazione tra il sistema (in altre parole, l'azienda ACME) e l'utente è stata gestita tramite la piattaforma Camunda. Si sono creati due progetti:

1. `CamundaProject`;
2. `UserWebEndPoint`.

`CamundaProject` contiene il diagramma BPMN, creato grazie al software *Camunda Modeler*, l'implementazione Java delle azioni descritte nel diagramma stesso e i form che permettono di dare un formato regolare alle richieste degli utenti. In altre parole, implementa la logica del server. `UserWebEndpoint` contiene invece una *servlet* che permette di lanciare i processi (cioè, in questo caso, gli ordini).

La servlet implementata è la `SendOrderRequest`, che simula l'invio di un ordine da parte di un utente al sistema. Esistono tre tipologie di ordini possibili, mappate attraverso il parametro `value`:

1. ordine contenente n cicli, con $n > 0$;
2. ordine contenente n componenti, con $n > 0$;
3. ordine contenente n cicli e m componenti, con $n > 0 \land m > 0$.

L'ordine viene implementato attraverso un messaggio. Le prime due tipologie di ordine contengono una lista di customizzazioni: per esempio la customizzazione di un componente potrebbe essere il colore del componente stesso.

L'invio del messaggio di avvio a Camunda è possibile perché siamo sullo stesso server: si applica infatti la tecnica *injection*, che permette di prendere una risorsa interna al server ed "iniettarla" all'interno del processo. In questo caso viene iniettata una variabile di tipo `ProcessEngine`, che è una risorsa di Camunda: tale variabile viene utilizzata per inviare il messaggio a tutti i servizi in attesa: se esso contiene l'id `start`, il servizio che lo riceve istanzia un processo.

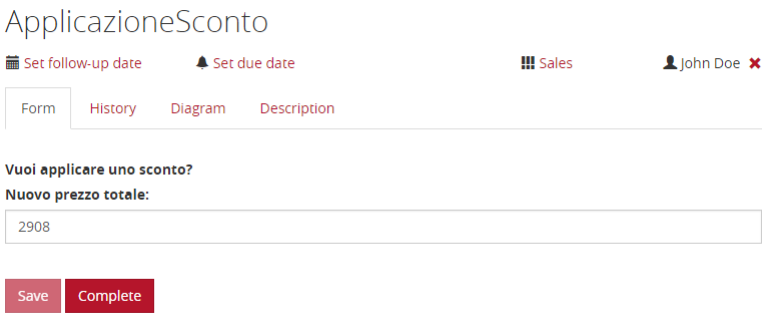
Si esaminano ora i vari passaggi effettuati dal processo, dalla ricezione dell'ordine fino alla sua conclusione.

Per prima cosa viene analizzata la tipologia di ordine: se è della terza tipologia indicata in precedenza, genera tanti messaggi quanti sono gli ordini presenti e lancia un ugual numero di processi. Fatto ciò termina. Se invece si ha a che fare con un ordine singolo, si procede.

Si controlla la presenza di customizzazioni, seguendo i passaggi indicati nel diagramma BPMN. È importante notare che il processo memorizza le informazioni legate all'ordine all'interno di variabili d'ambiente. Il controllo sulla

compatibilità delle customizzazioni viene effettuata tramite una decisione pseudo-randomica (con una probabilità di successo del 90%), presa su tutto l'insieme delle customizzazioni.

Dopo aver controllato la tipologia di ordine, il sistema si comporta come descritto nel diagramma BPMN; successivamente viene calcolato il prezzo di spedizione, basato su un valore arbitrario. Viene composto un preventivo e presentato in formato XML, predisposto dunque per eventuali visualizzazioni all'interno di siti internet o altre piattaforme. Se il costo totale del preventivo supera i 2500€, si forza l'utente `john` alla visione di un form ed egli deciderà se modificare il prezzo. Il preventivo finale viene dunque spedito.



Si attende poi l'attesa della conferma del preventivo da parte dell'utente. Si noti che i messaggi tra utente e sistema non sono stati gestiti, a causa di una serie di problemi elencati nelle conclusioni. L'implementazione dello svolgimento di questi dialoghi è stata affrontata nel seguente modo: viene fissato il timeout dell'attesa a due minuti, mentre all'invio della risposta viene assegnato un timeout di un minuto o di tre minuti. Naturalmente, la risposta verrà ricevuta solo nel primo caso. Tale meccanismo viene utilizzato per tutte le interazioni con l'utente. Tornando alla conferma del preventivo, l'utente (fittizio) può rispondere anche con un rifiuto, tramite una decisione pseudo-randomica come visto in precedenza.

Il resto del processo prevede l'interazione con i servizi Jolie, come descritti nella sezione apposita.

Infine è necessario soffermarsi su due sotto-progetti di servizio, `CommonObjects` e `JolieWrapper`. `CommonObjects` contiene le classi per la serializzazione e la deserializzazione degli ordini, mentre `JolieWrapper` contiene la gestione dell'interazione con i servizi Jolie.

3.2 Gestione magazzini: Jolie

L'implementazione della gestione dei magazzini è stata fatta utilizzando una SOA di servizi Jolie. I file utilizzati sono quelli contenuti nella directory `Magazzino`. La gestione del magazzino viene dunque implementata utilizzando tre componenti principali:

- magazzino primario, suddiviso in:
  - interfaccia (`interfMagazzinoPrimario.iol`);
  - implementazione (`magazzinoPrimario.ol`);
- magazzini secondari, suddivisi in:
  - interfaccia (`interfMagazzinoSecondario.iol`);
  - implementazione (`magazzinoSecondarioX.iol`);
- tipi di dato utili alle operazioni (`dataTypes.iol`).

`intefMagazzinoPrimario.iol`

Interfaccia che contiene le operazioni che possono essere effettuate dal magazzino primario. Le operazioni sono tutte del tipo `RequestResponse`, come mostrato in figura.

```
1  include "dataTypes.iol"
2
3  interface InterfMagazzinoPrimario {
4      RequestResponse:
5      ricercaComponentiMagazzini ( Ordine )( InfoComponenti ),
6      ricercaComponenteMagazzinoPrimario ( ComponenteIndirizzo )( InfoComponente ),
7      rilasciaComponente ( InfoComponente ) ( Stringa ),
8      riservaComponente ( InfoComponente ) ( Stringa ),
9      assolvoOrdineCiclo ( OrdineCiclo ) ( Booleano ),
10     assembloESpedisco ( OrdineCiclo )( Booleano ),
11     assolvoOrdineComponenti ( OrdineComponenti )( Booleano ),
12     ricercaComponenteMagazzinoPrimarioSenzaRiservare ( ComponenteIndirizzo )( InfoComponente )
13 }
```

`magazzinoPrimario.ol`

Questo file contiene l'implementazione delle operazioni del magazzino primario. Contiene una `inputPort`, in ascolto sulla porta `8000` utilizzando un protocollo `soap`, ed una serie di `outputPort`, per gestire le interazioni con:

- se stesso (`soap`, `8000`);
- il primo magazzino secondario (`soap`, `8001`);
- il secondo magazzino secondario (`soap`, `8002`);
- il servizio di calcolo delle distanze (`http`, `8100`);
- il servizio del fornitore (`http`, `8300`);
- il servizio di corriere (`http`, `8400`).

Viene poi creata una procedura di `init`, nel quale vengono inizializzate le informazioni principali: le informazioni del magazzino primario, le informazioni dei magazzini secondari e il listino delle componenti presenti nel magazzino secondario.

I magazzini implementano un sistema elementare per simulare la presenza dei componenti al loro interno, in modo da permettere di effettuare test dei meccanismi di ricerca, di riserva componenti e di calcolo delle distanze. Non viene utilizzato un database, poiché non necessario per mostrare le caratteristiche del sistema.

Vengono poi implementate le operazioni definite nell'interfaccia; alcune di queste operazioni eseguono solamente compiti fittizi, semplificazioni dei meccanismi del mondo reale:

- `rilasciaComponente`: si limita semplicemente a stampare una componente riservata, indicando per esempio in che magazzino si trova, delegando a questo il rilascio effettivo (che consiste nella stampa di una stringa);
- `riservaComponenti`: ha un funzionamento analogo a quello della funzione precedente;
- `assembloESpedisco`: per simulare le attività effettive di un'officina viene generata una richiesta di spedizione con i dati del prodotto (un identificativo intero generato casualmente tramite l'interfaccia `Math` offerta da Jolie) e i dati del cliente. Vengono stampate stringhe per simulare l'assemblaggio del cliclo; si attende poi la risposta della ditta di spedizioni per quanto riguarda l'esito della spedizione al cliente.

Altre operazioni hanno invece un'implementazione più realistica, poiché rappresentavano problemi centrali per la soluzione. Queste sono:

- `ricercaComponentiMagazzini`: questa operazione, presa una lista di componenti, le ricerca all'interno di tutti i magazzini. Restituisce una lista di componenti con la loro ubicazione (scelta prendendo la più vicina al cliente). I componenti trovati vengono riservati. Interagisce sia con il magazzino primario che con quelli secondari;
- `ricercaComponenteMagazzinoPrimario`: presa una componente, restituisce le informazioni ricavate dalla ricerca all'interno del magazzino primario. Utilizza il servizio di calcolo delle distanze e, in caso di successo nella ricerca, l'operazione `riservaComponenti`;
- `assolvoOrdineCiclo`: preso un elenco di componenti, vengono ricercate all'interno del magazzino primario (senza riservarle); se la componente non viene trovata, viene cercata nei magazzini secondari. Se l'esito di questa ricerca è positivo, si simulerà la spedizione al magazzino primario tramite l'operazione `spedisciAlMagazzinoPrimario`. Altrimenti la componente viene ordinata al fornitore tramite l'operazione `richiestaOrdine`. Si occupa inoltre di assemblare il ciclo e di spedirlo;
- `assolvoOrdineComponenti`: preso un elenco di componenti, verifica il campo booleano che determina la necessità di ordinare componenti. Nel caso ci sia, si chiama l'operazione `richiestaOrdine` del fornitore. Finito il

controllo e gli eventuali ordini, si chiama l'operazione `richiestaSpedizione`, dando al corriere come indirizzo di partenza quello del magazzino indicato nelle informazioni della singola componente.

`intefMagazzinoSecondario.iol`

Interfaccia che contiene tutte le operazioni che possono essere effettuate dai magazzini secondari. Questa interfaccia è comune a tutti i magazzini secondari. Le operazioni sono tutte del tipo `RequestResponse`, come mostrato in figura.

```
1  include "dataTypes.iol"
2
3  interface InterfMagazzinoSecondario {
4      RequestResponse:
5      ricercaComponenteMagazzino ( InfoComponenteIndirizzo )( InfoComponente ),
6      rilasciaComponente ( InfoComponente )( Stringa ),
7      riservaComponente ( InfoComponente )( Stringa ),
8      spedisciAlMagazzinoPrimario ( Componente )( Booleano )
9  }
```

`magazzinoSecondario[x].ol`

Mentre il file di interfaccia dei magazzini secondari è unico per tutti, le implementazioni sono tante quanti sono i magazzini secondari.

Come si può intuire, utilizzando questo meccanismo, aggiungere o cancellare un magazzino secondario diventa un'operazione estremamente semplice: basta aggiungere/eliminare il file `magazzinoSecondario[x].ol` corrispondente ed aggiungere/eliminare le informazioni e l'`OutputPort` all'interno del file di implementazione del magazzino primario. Contiene una `inputPort`, in ascolto sulla porta `800[x]` con protocollo `soap`, ed una serie di `outputPort`, per gestire le interazioni con:

- il magazzino primario (`soap`, `8000`);
- il servizio di calcolo delle distanze (`http`, `8100`).

Non sono contemplate interazioni con altri magazzini secondari, poiché la coordinazione viene completamente gestita dal magazzino primario.

Viene poi creata una procedura di `init`, nella quale si definiscono le informazioni del magazzino, come id, indirizzo e listino delle componenti presenti. Vengono poi implementate le operazioni definite nell'interfaccia:

- `ricercaComponenteMagazzino`:
  analoga all'operazione `ricercaComponenteMagazzinoPrimario`,
  con l'unica differenza che, in caso che la componente sia presente e il magazzino sia più vicino al cliente di quanto lo fosse il magazzino precedente (indicato nelle informazioni della componente), viene chiamata l'operazione `rilasciaComponente` sul magazzino precedente;
- `rilasciaComponente`: si limita a restituire una stringa con notifica di successo;

- `riservaComponente`: analoga alla `rilasciaComponente`;
- `spedisciAlMagazzinoPrimario`: simula la spedizione dei componenti al magazzino primario.

`dataTypes.iol`

Questo file contiene i tipi di dato che verranno utilizzati all'interno della soluzione. Gli ultimi due tipi, `Booleano` e `Stringa` sono stati creati per la generazione dei file *wsdl*, che non consentono tipi semplici nei parametri in ingresso ed uscita delle operazioni.

3.3 Capability esterne

Le capability esterne, come l'istituto di credito, la ditta di spedizioni e il fornitore, sono state implementate utilizzando dei servizi Jolie. Tali servizi sono stati semplificati: la loro implementazione è difatto un semplice scambio di oggetti e stringhe con la SOA costruita per la gestione del magazzino.

Istituto di credito

L'istituto di credito ha un'interfaccia semplificata, definita nel file `interfCredito.iol`, come si può vedere in figura.

```
1  include "creditoTypes.iol"
2
3  interface InterfCredito {
4      RequestResponse:
5          richiestaVerifica( Bonifico )( Risultato )
6  }
```

L'unica operazione viene implementata nel file `creditoServer.ol`: dopo aver ricevuto un bonifico, viene deciso l'esito della verifica scegliendo un numero casuale (grazie all'interfaccia `Math` di Jolie). Se il numero scelto è minore o uguale di `0.3`, l'esito viene settato a `false`, altrimenti a `true`. Il server contiene poi una `inputPort`, in ascolto sulla porta `8200` con protocollo `http`. Nel file `creditoTypes.iol` vengono descritti i tipi utilizzati.

Calcolo Distanze

Il servizio di calcolo delle distanze viene definito nel file `interDistanza.iol`, come si può vedere in figura.

```
1  include "distanzaTypes.iol"
2
3  interface InterfDistanza {
4      RequestResponse:
5          calcolaDistanza( PuntiDaCalcolare )( RisultatoDistanza )
6  }
```

Il servizio di calcolo delle distanze maschera una chiamata ai servizi offerti da Google Maps, raggiunti tramite il parametro `location` impostato a

socket://maps.googleapis.com:80/maps/api/distancematrix/. Il servizio prende in input due punti geografici tra i quali calcolare la distanza, richiedendo per entrambi i parametri contenenti la città e la provincia. Una volta che i punti vengono dati al servizio, viene composta una richiesta per Google Maps, impostando come mezzo di trasporto la macchina (tramite il parametro `driving`). Il servizio, ricevuta la risposta, ne estrae le informazioni utili che verranno poi utilizzate dal sistema. Nel particolare, il servizio contiene una `outputPort` verso il servizio di distanze di Google Maps, con protocollo `http` {`.method = ''get''`}.

Ditta di spedizioni

La ditta di spedizioni ha un'intefaccia semplificata, definita nel file `interfCorriere.iol`, come si può vedere in figura.

```
1  include "corriereTypes.iol"
2  interface InterfCorriere {
3      RequestResponse:
4          richiestaSpedizione( OrdineSpedizione )( RisultatoSpedizione )
5  }
```

L'unica operazione viene implementata nel file `corriereServer.ol`, che compone una stringa formattata utilizzando i campi dell'input di tipo `OrdineSpedizione`. Il file contiene inoltre una `inputPort`, in ascolto sulla porta `8400` con protocollo `http`. Nel file `corriereTypes.iol` vengono definiti i tipi utilizzati.

Fornitore

Il fornitore ha un'intefaccia semplificata, definita nel file `interfFornitore.iol`, come si può vedere in figura.

```
1  include "fornitoreTypes.iol"
2  interface InterfFornitore {
3      RequestResponse:
4          richiestaOrdine( OrdineComponente )( RispostaOrdine )
5  }
```

L'unica operazione viene implementata nel file `fornitoreServer.ol`, che compone una stringa formattata utilizzando i campi dell'input di tipo `OrdineComponente`. Il file contiene inoltre una `inputPort`, in ascolto sulla porta `8300` con protocollo `http`. Nel file `fornitoreTypes.iol` vengono definiti i tipi utilizzati.

4 Conclusioni

4.1 Informazioni aggiuntive

Per comprendere meglio alcune scelte che hanno influenzato l'implementazione della soluzione, è necessario fornire alcune informazioni aggiuntive.

4.1.1 Invio dell'ordine

L'invio di un ordine è stato realizzato tramite una pagina HTML che implementa a tutti gli effetti un form. L'utente sceglie la tipologia di ordine da inviare. Selezionata la tipologia, l'ordine viene inviato al sistema, mentre all'"utente" compare una schermata che conferma l'avvenuto invio.

4.1.2 Avvio del sistema

Per implementare con semplicità l'avvio del sistema, si è scelto di scrivere un file `.bat`, che permette tramite il suo avvio di lanciare tutti i servizi Jolie.

4.1.3 Camunda + WildFly

La versione di Camunda utilizzata per l'implementazione della soluzione è quella che comprende WildFly, a causa di una migliore conoscenza del software.

4.2 Problemi riscontrati

Durante l'implementazione della soluzione sono comparsi alcuni problemi.

4.2.1 Messaggi da e per l'utente

Il messaggio di avvio del processo è stato implementato senza alcuna difficoltà: i problemi sono sorti nel momento in cui si deve manipolare un dialogo attivo. In quel caso un evento intermedio deve interagire con un messaggio da parte dell'utente genera un errore, nel quale il sistema non trova nessuna componente "in ascolto" di tale messaggio.

4.2.2 Bug generazione del WSDL

Durante la generazione dei file WSDL, effettuata tramite il tool `jolie2wsdl`, si è riscontrato un problema: i file WSDL cos generati non erano infatti validi per il tool `wsdl2java`, necessario per permettere l'interfacciamento tra il sistema implementato in Camunda ed i servizi implementati in Jolie. Per valutare il problema, si è deciso di effettuare alcuni test tramite il software SoapUI, riscontrando che in questo caso i file WSDL venivano accettati; ma, nel momento in cui venivano utilizzati per eseguire alcune chiamate, Jolie rispondeva segnalando un errore di mismatch tra tipi. Il problema è stato risolto scrivendo i file WSDL a mano.