**Lezione 4 modulo 3**

In questo modulo approfondiremo una tipologia di diagrammi che è possibile realizzare con BPMN che va sotto il nome di BPMN collaboration diagram . L'obiettivo di questo diagramma è di descrivere l'interazione che può esistere fra diversi processi gestiti da entità differenti. Queste entità vengono anche dette partecipanti. Le entità possono riferirsi a diverse organizzazioni oppure a diverse divisioni all'interno della medesima organizzazione. Vedremo anche che dal punto di vista poi dell'esecuzione del processo, il modo migliore per definire un partecipante è quello dell'orchestratore, cioè quel sistema che è in grado di eseguire un modello di processo. Anche all'interno della stessa organizzazione ce ne possono essere diversi, pertanto a ognuno di essi può essere assegnato il ruolo di partecipante. Al momento la cosa importante da ricordare è che l'interazione fra questi diversi processi è governata e definita da un semplice scambio di messaggi. L'elemento principale che compone un diagramma di tipo collaborativo è rappresentato da un Pool. Un Pool non fa altro che rappresentare, appunto, un partecipante. Come detto prima può essere un'organizzazione, a sua volta il Pool può anche essere suddiviso in diverse Lane, come è già stato visto in alcuni dei moduli precedenti, in questo caso non approfondiremo questa suddivisione perché a noi interessa come granuralità minima il concetto appunto di Pool. E, la cosa importante è che all'interno di un Pool io ho un processo che do per scontato essere orchestrato interamente all'interno appunto dell'organizzazione che è definita, o della divisione all'interno dell'organizzazione che è definita appunto dal Pool. Un Pool può essere rappresentato in due modalità che vengono dette Black box e White box. nel caso Black box Il pool non definisce nulla al suo interno perché non viene dato in alcun modo alcun dettaglio rispetto a quelle che sono le attività svolte da quell'organizzazione quindi un Pool Black box è inserito all'interno di un collaboration diagram solo per dire che c'è un dialogo con una certa organizzazione, è necessario scambiarsi dei messaggi ma non viene dato altro tipo di informazione su questa organizzazione, su questa interazione all'interno della collaborazione. Nel caso invece del White box sono anche presenti i dettagli del processo orchestrato, o almeno un sottoinsieme dei dettagli del processo orchestrato, dove per sottoinsieme intendo dire tutte quelle attività che almeno rappresentano i Task relativi allo scambio di messaggi, alle dipendenze che esistono nel momento in cui io devo scambiare appunto questi messaggi. Dal punto di vista grafico, in questa slide è rappresentato appunto un Pool in modalità Black box, è semplicemente il costrutto grafico dove abbiamo appunto questo rettangolo che rappresenta il Pool con una banda a sinistra, in cui viene indicato il nome dell'organizzazione , o comunque dell'elemento che è in carico all'esecuzione del processo, null'altro è definito all'interno di questo di questo elemento grafico. Passando invece alla modalità White box, noi qui abbiamo l'esempio di un Pool che si riferisce a un'organizzazione detta Supplier, che in questo caso è anche definita in due Lane, Sales and Distribution e all'interno di questo Pool sono state definite alcune attività. Quindi io do una vista, do un dettaglio rispetto a quello che accade all'interno della mia organizzazione, dell'organizzazione Supplier che potrebbe essere la mia nel momento in cui sto raccontando il mio processo, o l'organizzazione di cui sono a conoscenza, ad esempio, di quelle attività che vengono svolte al suo interno in determinati momenti. Fatta questa premessa su quelli che sono gli elementi principali, i building block di un collaboration diagram, definiamo cos'è un collaboration diagram. Un collaboration diagram è un diagramma, che utilizza la notazione BPMN, dove sono presenti uno o più Pool e il protocollo che governa l'interazione fra questi Pool, fra questi processi che sono definiti all'interno dei Pool o, anche se non sono definiti do per scontato che siano in qualche modo gestiti all'interno di ognuno dei Pool, questa iterazione è definita in termini di messaggi scambiati. Va ricordato che, sebbene poi nella stragrande maggioranza dei casi identificheremo come messaggio uno scambio informativo, quindi uno scambio che molto probabilmente è uno scambio digitale, il concetto di messaggio in BPMN è qualcosa di più ampio. Infatti, sebbene l'elemento grafico che rappresenta il messaggio è la busta, può anche avvenire uno scambio fisico fra le diverse entità. E comunque anche lo scambio fisico viene rappresentato con un messaggio. Sarà poi compito del modellista indicare con un nome o con dei commenti il fatto che eventualmente un messaggio si riferisce a uno scambio fisico di oggetti. La collaboration diagram può essere sviluppata, può

descrivere la collaborazione fra due o più organizzazioni secondo il punto di vista di uno dei partecipanti, oppure un punto di vista neutro rispetto ai partecipanti. Cosa si intende dire? Facciamo appunto riferimento ad alcuni esempi che sono presenti anche sulle specifiche BPMN; nello specifico, quando io prendo un determinato punto di vista può accadere che io rappresenti, come avviene per esempio nella parte in basso di questo diagramma, rappresenti un processo non all'interno di un Pool: è per questo motivo che la definizione di un collaboration diagram non richiede necessariamente almeno due Pool ma anche solo un Pool. L'importante è che però sia presente la descrizione di un processo oltre al singolo Pool che poi viene raccontato in questo modo. Io cosa sto dicendo? Sto dicendo che sto prendendo il punto di vista di chi ha in esecuzione questo processo, quindi molto probabilmente la mia organizzazione e voglio descrivere, in questo caso, l'interazione che ha con un'altra organizzazione detta Financial Institution; quindi prendo un punto di vista specifico, definisco lo scambio informativo in termini di messaggi e non definisco però quello che accade all'interno di Financial Institution: perché non lo definisco? Può anche succedere che io non lo conosca, non la conosca la struttura del processo all'interno, oppure non sia importante rappresentarla in un determinato momento. Ne approfitto anche per far notare come il Pool venga, in questo diagramma, rappresentato con una notazione alternativa rispetto a quella vista in precedenza e anche quella solitamente mostrata, che vede la rappresentazione del nome del Pool sul lato sinistro e lo sviluppo poi del Pool da sinistra verso destra destra. Esiste nella realtà anche la possibilità di rappresentarlo in maniera alternativa indicando il nome in alto e sviluppando il Pool dall'alto verso il basso. Se invece vogliamo rappresentare il collaboration diagram rispetto a un punto di vista neutro, allora esisteranno almeno due Pool rappresentanti le organizzazioni di cui voglio descrivere lo scambio informativo, la loro interazione e all'interno di questi Pool io posso avere una descrizione del processo interno, quindi posso averli in modalità white box o posso anche averli tutti in modalità black box. Oppure, posso avere un misto white box/black box, non c'è nessun limite rispetto a questo, dipende dal caso, dipende dal contesto. In questo caso tutti i miei Pool sono rappresentati attraverso un white box. Abbiamo detto che il collaboration diagram definisce l'interazione che esiste fra un insieme di organizzazioni, almeno due e questa interazione viene definita in termini di messaggi. Ecco, esistono principalmente due tipologie di pattern: due pattern di scambio di messaggi fra organizzazioni che è possibile modellare all'interno di un collaboration diagram; e sono i classici pattern che abbiamo incontrato in diverse discipline all'interno della computer science, che sono il One-Way, che definisce uno scambio informativo dove io ho un partecipante che semplicemente manda un messaggio a un altro partecipante oppure Request- response che prevede l'invio del messaggio da un partecipante all'altro partecipante e l'altro partecipante dovrà rispondere, inviare un messaggio di risposta al partecipante che ha effettuato la prima chiamata. In questo caso, molto importante, nel caso Request-response noi abbiamo a che fare con la tipica comunicazione di tipo sincrono e quindi bloccante. Quindi, anche se in questa rappresentazione abbiamo volutamente inserito due pool di tipo Black box, quindi non sappiamo cosa succede all'interno, la cosa che però siamo sicuri che succede è che nel momento in cui il partecipante A invia un messaggio al partecipante B rimane sempre bloccato nella sua esecuzione finché non riceve una risposta da parte del partecipante B. Ecco, questo nel momento in cui io dico esplicitamente che questa è una coppia di messaggi che fa capo a Request-response. Se io non avessi detto nulla rispetto al fatto che questo rappresenti un Request-response, avrei potuto anche pensare che questo diagramma rappresenta una somma di due One-Way: uno nel senso da A verso B e uno nel senso da B verso A; questo mi permette di rappresentare la tipica situazione asincrona e quindi non bloccante, in cui io ho il partecipante A che invia un messaggio, può tranquillamente eseguire altre attività nel frattempo, B eseguirà le attività che dovrà eseguire sulla base dell'input ricevuto da A, risponderà attraverso, potremmo dire una funzione di call back e a un certo punto il partecipante A catturerà attraverso la funzione di call back l'invio del messaggio e quindi opererà poi di conseguenza nel proseguo. Ecco se non viene esplicitamente indicato, la scrittura appena fatta, la scrittura quindi che prevede due scambi, uno da A a B e uno da B a A, può essere interpretata sia come Request-response sia come una coppia di messaggi in One-Way per implementare quella che è una logica asincrona. Una cosa importante

da sottolineare all'interno dei collaboration diagram è che un Pool può anche essere di tipo multi-instance, indicato attraverso la classica decorazione composta dalle tre linee verticali che è stata vista anche per le attività. Inserire questa decorazione all'interno di un Pool indica che il partecipante B non deve essere essere inteso come uno specifico partecipante ma io sto definendo con partecipante B, chiamato partecipante B, questo esempio o comunque il nome che poi viene associato al Pool, io sto rappresentando con questo participante B una classe di partecipanti, dove tutti questi partecipanti hanno lo stesso comportamento. Io sto quindi dicendo che il partecipante A sta dialogando non con un altro partecipante, ma con una classe di partecipanti, dove dal punto di vista di A sono tutti identici in termini di scambio di messaggi di comportamento che poi avranno rispetto all'interazione che hanno con il partecipante, con se stesso con il partecipante A. Quindi la molteplicità, il multi-instance, indicato in questo modo, presuppone che cosa? Presuppone che questo scambio non indica un messaggio che da A va a B, ma indica un insieme di messaggi, mandati in broadcast, a tutte quelle che sono le istanze di partecipante B al momento attive. Quindi potrebbero essere anche diversi messaggi. La stessa cosa funziona anche per il Request-response: io posso avere anche un mult-instance da queste parti. La cosa importante che adesso sottolineerò è la coerenza che esiste fra il messaggio, diciamo così, di ingresso e il messaggio di uscita. Lo stesso discorso in realtà vale anche nel momento in cui questa coppia di messaggi non rappresenta tanto una Request-response ma una coppia di One-Way. Perché è necessario garantire la coerenza? Perché se noi abbiamo detto che adesso supponiamo di avere ad esempio 10 istanze attive, per qualche motivo, io ho 10 partecipanti e so di avere questi 10 partecipanti, questo scambio informativo indica che A sta inviando 10 messaggi identici ad ognuno, quindi uno di questi 10 messaggi va al partecipante B1, poi al partecipante B2 poi a B3 e così via quindi a ognuna delle istanze viene inviata un'unica richiesta un unico messaggio. Di contro, il partecipante A a un certo punto cosa si deve aspettare? Si deve aspettare 10 messaggi in ingresso, 10 messaggi di risposta. La cosa importante è che A deve riuscire sempre a correlare quella che è la richiesta inviata a una determinata istanza e la risposta proveniente da quella istanza. Vediamo che questo richiede un piccolo accorgimento nella definizione delle attività che saranno definite qui all'interno. Ovviamente nel caso Black box questo non è un problema perché non è rappresentato in alcun modo. Vediamo quindi cosa succede nel momento in cui io voglio specificare cosa viene eseguito all'interno dei due Pool. Rimaniamo ancora nel caso in cui non abbiamo nessuna delle due come multi instance quindi un classico esempio di One-Way tradizionale normalissimo. In questo caso io ho, adesso supponiamo che qui ho delle attività, anche dopo delle attività, così come per il partecipante B ci concentriamo solo su una sezione del processo. In questo momento io sto dicendo che dopo aver eseguito una porzione del processo A invia un messaggio a B; l'invio del messaggio presuppone l'utilizzo di un message task di tipo Send e di un message task di tipo Receive dall'altro lato. In questo modo questo invia, una volta inviato passa all'azione successiva, al task successivo questo rimane in attesa del messaggio, quando questo messaggio è arrivato allora può passare all'azione successiva. Due note importanti; la prima: avremmo potuto tranquillamente rappresentare queste due attività anche utilizzando gli eventi di tipo intermediate relativi a questi tipi di eventi, quindi avrei potuto mettere un intermediate con la busta, l'envelope nero per indicare il Send message, che è collegato, dall'altra parte, a un evento di tipo intermediate con la busta, in questo caso bianca, di tipo catch, quindi di tipo receive. Non cambia assolutamente nulla, il consiglio è cercate di essere coerenti; se lo fate, se scegliete un certo tipo di scrittura tenetelo durante tutta la descrizione del vostro processo, in maniera tale da rendere più leggibile il vostro diagramma perché è meglio uno o è meglio l'altro. In realtà in assoluto non è meglio uno o meglio l'altro. In generale quello che possiamo dire è che sicuramente l'utilizzo degli eventi rende il diagramma un po' più compatto. Però ha un difetto: il difetto è legato al fatto che se io volessi gestire ad esempio delle eccezioni durante l'invio o la ricezione del messaggio, quindi appiccicare degli even boundary per esempio sul Send message, non lo posso fare se utilizzo questo tipo di notazione, perché non non è possibile avere un even boundary su un altro evento, direttamente; quindi, in alcune situazioni, siamo obbligati ad utilizzare l'attività, il Task, meglio, e non l'evento. L'altro aspetto importante da sottolineare quando si tratta di messaggi è che un'attività di Send

o di receive Task sono attività che in maniera specifica inviano o ricevono, fanno solo quello. Quindi è concettualmente sbagliato, oltre che sintatticamente sbagliato, avere dei Send message del tipo elaboro la richiesta e poi invio il messaggio: no, perché in questo caso io sto dichiarando in qualche modo che quell'attività fa qualcosa e invia il messaggio o uno o l'altro. Quindi in questi casi è sempre meglio splittare, dividere in due Task distinti, uno legato all'elaborazione, quello che è, e l'altro legato all'invio del messaggio; quindi l'invio o la ricezione, vale la stessa cosa anche per la ricezione, devono indicare unicamente attività riferite al mero invio e la mera ricezione dei messaggi. Quindi abbiamo visto, nel caso One-Way, come può essere modellato l'interno dei miei processi. Cosa succede invece nel caso Request-response? Sempre considerando che può esserci qualunque cosa prima e dopo, qui non vengono rappresentati, qui vediamo che il partecipante A invia un messaggio a B, B lo riceve e dopo averlo ricevuto fa qualcosa e dopo aver fatto una certa cosa invia una risposta. Invia una risposta che viene catturata dal partecipante A. Come vedete, qui in mezzo non c'è assolutamente nulla, non deve esserci nulla altrimenti perdiamo il concetto di bloccante di cui abbiamo parlato in precedenza; perché dopo aver inviato il messaggio, il nostro partecipante A non può far altro che rimanere in attesa del messaggio successivo. Ovviamente, qui non c'è più il dubbio di prima: vediamo, guardando il processo, che questa è effettivamente una Request-response. Se io avessi avuto un'attività qui in mezzo che faceva qualcosa, allora anche in questo caso non avrei più avuto dubbi e avrei interpretato questo scambio informativo come due One-Way speculari, prima uno poi l'altro, perché? Perché nessuno dei due rimaneva bloccato nell'esecuzione del processo in attesa di messaggi che provenivano dall'altro lato. Ma cosa succede quando io ho a che fare con i multi-instance? Vediamo questo esempio: in questo caso abbiamo che il partecipante B è di tipo multi-instance, vediamo che il partecipante B riceve una richiesta, fa qualcosa e invia una risposta. Il fatto di essere multi-instance indica, come abbiamo detto, il fatto che non sto necessariamente parlando di un partecipante ben preciso, ma di un partecipante, o una serie di partecipanti, un insieme di partecipanti che hanno in comune un certo comportamento, un certo processo, tutti reagiscono in un determinato modo. Pertanto il partecipante A non vede dall'altro lato un unico partecipante ma ne vede diversi, tutti che però fanno esattamente la stessa cosa. L'istanza di questo processo è definito da una struttura di questo tipo: ricevo una richiesta, faccio qualcosa, invio una risposta. Per questo motivo l'invio e la ricezione della risposta deve essere gestito in maniera opportuna, perché? Perché in questo caso io cosa ho? Devo dire: non devo inviare un'unica richiesta, ma ne devo inviare diverse; non devo aspettarmi un'unica richiesta, ma mi devo aspettare n richieste. Quindi l'attività di invio della richiesta, così come l'attività della ricezione alla risposta, dovranno essere attività di tipo multi-instance. Quindi, dietro le quinte, come ben sapete, è come se noi avessimo un insieme n istanze, in parallelo, che in questo caso inviano tutte il messaggio, una volta inviate tutte, restano tutte in attesa; c'è un'altra attività, scusate: c'è un'altra attività che rimane in attesa, cioè n istanze di quelle attività rimangono in attesa del messaggio proveniente dalle diverse istanze del partecipante B in risposta alla richiesta fatta prima. Questo tipo di di scrittura però, vedremo fra poco che non è propriamente corretta, perché? Perché, come abbiamo detto prima, da queste parti io invio 10 richieste ma non ho nessuna correlazione fra le 10 richieste e le 10 risposte. Vedremo poi come possiamo in realtà gestire questa coerenza, ci siamo concentrati finora sui mattoni principali nell' interazione fra due partecipanti all'interno del nostro collaboration diagram che si riferiscono al pattern One-way o Request-response. In realtà un collaboration diagram è un insieme, è una combinazione di tutti questi mattoni. Nel caso specifico che abbiamo qui, vediamo come abbiamo un insieme di tre interazioni di tipo One-Way: A che invia un messaggio a B, non è bloccante in nessun modo, né qui né qui, B che invia un messaggio ad A e poi sempre B che invia un altro messaggio ad A. Quindi costruire collaboration diagram significa andare a costruire questa serie di interazioni tra due o più partecipanti che appunto sono rappresentati all'interno del nostro collaboration diagram secondo quella che è la logica di scambio informativo, appunto, fra i nostri partecipanti. Bisogna però fare attenzione, in questa composizione, ad alcune situazioni abbastanza particolari, abbastanza critiche che portano errori che però sono abbastanza comuni. Innanzitutto bisogna prestare attenzione ai punti di decisione; il diagramma qui mostrato, che è un diagramma abbastanza

semplice, porta con sé un problema, un errore abbastanza rilevante: cosa abbiamo qui? Noi abbiamo il partecipante B che a un certo punto invia una domanda al partecipante A che la riceve. ILpartecipante A fa il check di questa domanda e poi può decidere se accettare la domanda o rigettare la domanda e in questi due casi non fa altro che inviare un messaggio. Pertanto il partecipante B, a seconda del risultato, resta in attesa della rejection piuttosto che dell'acceptance. In realtà qui c'è un errore abbastanza grossolano dovuto all'utilizzo di questo gateway per decidere come andare a comportarsi a seconda del risultato. Vi ricordo che questo è un gateway che viene detto databased xor, cioè indica il fatto che la decisione se seguire questo ramo oppure seguire quest'altro ramo, dipende unicamente da conoscenza, da informazioni che si conoscono a priori rispetto a questo processo. Cioè questo processo cosa sa prima, in base ad alcuni dati che conosce e alle condizioni che dovrebbero essere descritte sui rami che dipartono dal gateway, decidere se seguire un ramo piuttosto che un altro. Ecco, in realtà, in questo caso il partecipante B non ha questa informazione perché l'informazione deriva appunto dal messaggio che arriva dopo. Infatti siamo in una situazione che viene detta di defert choice to, di una scelta differita, cioè andare su questo ramo o andare su quest'altro ramo dipende da un qualcosa che non è conosciuto a priori, ma che è figlio di eventi che devono accadere; nello specifico, o questo evento oppure quest'altro evento. Come si modella in questo caso? In questo caso si modella utilizzando il gateway opportuno, che è quello di tipo Event Based. Infatti, questo, appunto, l'Event Based xor, definisce ancora dei comportamenti mutuamente esclusivi, ma in questo caso questo gateway indica il fatto che, associato ad ognuno dei rami ci deve essere un evento, nella fattispecie abbiamo la ricezione di due messaggi, di un messaggio per ognuno dei rami e, se riceviamo l'acceptance, allora si attiva questo ramo e poi andremo avanti. Se invece riceviamo la rejection, si attiverà quest'altro ramo e andremo avanti secondo il processo che è indicato in questa struttura, in questo ramo. Un altro elemento a cui fare attenzione è la mancata sincronizzazione fra i due Pool. Dal punto di vista sintattico non c'è nessun problema in questo diagramma perché abbiamo un One-Way da questa parte, abbiamo una risposta, un'altra risposta, due One-Way, anche qui, dall'altra parte, potremmo vederla, questa qua, se notate non è altro che la sezione che abbiamo visto in precedenza; potremmo vederlo come due Request- response che in qualche modo sono intersecate fra di loro, perché data questa domanda, questa richiesta, io posso avere o questa risposta o quest'altra risposta; ad ogni modo ho una struttura di questo tipo e poi a un certo punto, se ricevo la rejection termino il mio processo, se ricevo l'acceptance allora posso inviare il pagamento.Il mio partecipante A una volta che ha ricevuto il pagamento va avanti. Ecco, in questo caso in realtà per come è stato costruito, c'è la possibilità di avere un deadlock, perché? Perché siccome dopo aver ricevuto la rejection il partecipante B termina il suo processo, dopo aver informato rispetto alla rejection, io il partecipante A l'ho modellato in maniera tale da rimanere in attesa di un pagamento, cosa che non avverrà mai in questo momento, in questo modo, perché? Perché siccome io l'ho rigettato, il mio partecipante non invierà mai nessun tipo di nessun tipo di pagamento e quindi, anche se dal punto di vista sintattico è corretto, dal punto di vista semantico questo potrebbe portarmi a un deadlock nell'esecuzione. Quindi bisogna sempre prestare attenzione a come dialogano i vari attori, quindi devo stare attento e essere consistente nel fatto che quando io informo la rejection, questo termina il messaggio e non mi devo aspettare nient'altro dall'altra parte. Sì, potrei fare delle altre attività, ma non mi posso aspettare nessun messaggio dall'altra parte, così come non mi potrei nemmeno aspettare di inviare un messaggio all'altro capo, perché? Perché quel processo è terminato. Questo processo ha seguito questa execution trace e quindi è da considerarsi terminato. Infatti, noi abbiamo che come receive payment, adesso la ricezione del pagamento segue un branch che è indipendente da quello di prima e che porta alla terminazione del processo assieme alla terminazione del processo poi del partecipante B, una volta che ha inviato il pagamento. Torniamo al caso di prima. Ecco, nel caso di prima, io che più o meno quello che abbiamo visto prima, quia è rappresentato un caso di tipo asincrono perché ho messo un'attività in mezzo fra la richiesta della rispost,a ma non cambia assolutamente nulla. Qui cos'è che ho? Ho il partecipante A che manda, supponiamo, 10 richieste. Quando analizzo il partecipante B, leggo il partecipante B, mi devo mettere nei panni di un'istanza, di un possibile partecipante B. Io ricevo il

messaggio, faccio qualcosa e invio la risposta. Questo in maniera isolata rispetto agli altri partecipanti, perché un'istanza del partecipante B non deve essere a conoscenza necessariamente dell'esistenza di altri partecipanti che hanno il suo stesso comportamento; il partecipante B agisce in maniera isolata e invia il messaggio di risposta. Cosa succede qui? Succede che quando io invio 10 messaggi, poi faccio azioni successive e ho messo un multi-instance per indicare che a seconda del messaggio che ho inviato poi faccio qualcosa, vado avanti e quindi ho 10 azioni in parallelo e poi ho 10 attese nella ricezione dei messaggi sempre in parallelo. Però capite che quando invio il messaggio, faccio qualcosa e ricevo il messaggio, io mi aspetto un certo allineamento fra le varie richieste e le varie risposte e le varie azioni. Quindi io mi aspetto che in un certo modo, quando io invio un messaggio riceva in qualche modo, ci sia una correlazione, riesca a scoprire qual è l'istanza che ha gestito l'invio, con quale istanza per esempio ha gestito la ricezione così come quale istanza ha gestito quell'azione di mezzo; perché, supponendo di aver mandato, numerando da 1 a 10 i miei partecipanti B e supponendo di aver mandato in ordine 10 messaggi da B1 a B10, magari la prima risposta che ottengo è quella di B10. Quindi se io vado e e la risposta di B10 è gestita dalla prima istanza della ricezione del messaggio; quindi c'è un disallineamento fra le varie istanze. Questo potrebbe portare a problemi. Quando soprattutto noi abbiamo un Request-response oppure è necessario mantenere un certo allineamento fra le varie istanze, il modo migliore è quello di raggruppare queste azioni che devono essere allineate. Questo raggruppamento è mostrato in questo diagramma. Cosa è stato fatto? E' stata introdotta un'attività composta, in questo caso nella sua versione estesa e non collassata, che è di tipo multi-instance. E questa attività che è composta da invia messaggio, fai qualcosa, ricevi il messaggio, si rapporta con un'istanza del mio Pool; in questo modo io ho una correlazione fra quello che succede nella particolare istanza del partecipante B, rispetto a quello che succede nella particolare istanza del partecipante A. Più che l'istanza del partecipante A, il partecipante A è unico, però al suo interno ha generato n istanze di questo sottoprocesso e ogni istanza di questo sottoprocesso governa la relazione che esiste, l'interazione che esiste con un'istanza dell'altro Pool. Concludiamo questa presentazione dei BPMN collaboration diagram utilizzando un esempio preso dalle specifiche BPMN, che riguarda il rapporto che esiste tra un venditore di pizza, un pizzaiolo e il consumatore della pizza, un cliente di questa pizzeria. Questi due attori sono rappresentati appunto da due Pool. Il primo è un semplice Pool composto da un'unica Lane, mentre il secondo è un Pool composto da tre Lane. Infatti all'interno della pizzeria è definito il commesso, il pizzaiolo e la persona che dovrà spedire, poi, consegnare la pizza; il processo ha inizio dal lato del consumatore della pizza, che quando decide di essere affamato e dopo aver selezionato la propria pizza, non fa altro che ordinare la pizza. Questo è il primo momento in cui c'è una connessione fra ilcliente e la pizzeria. Infatti viene inviato un messaggio alla pizzeria; in particolare in questo caso vediamo come il messaggio la particolarità di triggerare, quindi di far scattare l'esecuzione del processo che governa la pizzeria. Ovviamente la pizzeria avrà in esecuzione n di questi processi contemporaneamente; in questo caso noi stiamo modellando il rapporto che esiste fra un'istanza di questo processo lato pizzeria e l'istanza del processo legato al singolo cliente. Siccome è noto che il pizzaiolo è sempre abbastanza in ritardo, almeno questo pizzaiolo è sempre abbastanza in ritardo, una particolarità di questo processo riguarda la presenza di questo gateway parallelo, dove da un lato parte il classico processo di produzione della pizza, di cottura della pizza, di spedizione della pizza e poi di pagamento, che viene distribuito sui diversi Lane. Ma in particolare abbiamo che il nostro commesso dovrà anche gestire eventuali lamentele da parte del nostro cliente. Infatti, al tempo stesso, cioè mentre sta producendo la pizza ed eventualmente la sta spedendo, il commesso è pronto anche a ricevere, rimane in attesa anche di un eventuale messaggio proveniente dal cliente, dove probabilmente ci sarà una lamentela. Quindi l'obiettivo del commesso è quello di calmarlo e poi quello che succede è che questo Ramo non viene terminato ma torna indietro in attesa ancora di un'eventuale nuova lamentela. Se guardiamo invece il processo dal lato del customer, quindi del cliente, dopo aver ordinato la pizza possono accadere due cose: come vedete qui viene utilizzato il l'event based xor. Infatti, cosa possono accadere? Due eventi: Il primo è che la pizza arrivi, quindi una volta ricevuta la pizza, pizza che viene inviata appunto dal mio delivery boy, in questo caso notate come sempre con il

concetto di messaggio viene mappata una consegna fisica di un oggetto, quindi il messaggio non necessariamente deve essere un elemento informativo e poi c'è un request-response che riguarda il pagamento della pizza; in questo caso tutti e due sia il request sia il response sono collassate in un'unica attività perché molto probabilmente quello che è stato modellato qui e qui è un'attività composta, quindi in qualche modo anche se non è esplicitamente rappresentata, noi possiamo pensare di avere una decorazione con un più per indicare che questa attività appunto è un'attività composta, non è un'attività atomica, quindi non è un Task, che governa appunto lo scambio fra soldi e l'eventuale ricevuta che poi viene viene affidata al mio cliente. dopo aver ricevuto la pizza, pagata, quindi posso mangiare la pizza e il processo lato customer, viene terminato. Al tempo stesso però, come abbiamo detto potrebbe accadere che dopo 60 minuti e questo è il secondo ramo che può essere essere attivato da quest'altro evento, che in questo caso è un timeout, se entro 60 minuti la pizza non arriva, allora viene fatta partire quell'attività che comunica appunto con il mio commesso in cui io chiedo la pizza, le informazioni sulla pizza e mi vengono restituite le informazioni appunto sulla pizza; in questo caso, a differenza di prima che portava la chiusura del processo, dopo aver ricevuto rassicurazioni appunto sulla pizza, quello che succede è che io rimango, ritorno, o meglio il customer ritorna in attesa dell'accadimento di uno di questi due eventi, della ricezione della pizza piuttosto che della scadenza del timeout. Cosa che può ancora accadere, ma questo viene comunque gestito in maniera sincronizzata, perché anche in questo caso, dopo avermi rassicurato, il commesso ritorna in attesa, appunto, del dell'evento di ricezione appunto della lamentela. Un altro aspetto interessante di questo diagramma è che, se notiamo, l'unico ramo nel lato pizzeria che termina, è identificato non con un end event normale come nel caso del cliente, ma con un terminate, questo perché? Perché in questo caso abbiamo due rami che sono in parallelo, quindi mentre io eseguo questo ramo anche questo ramo è attivo, quindi una volta eseguito questo ramo e arrivati alla fine, non ha senso tenere attivo anche questo ramo, perché ormai non riceverò più le lamentele. Per questo motivo utilizzo il terminate per, diciamo così, disattivare anche questo e concludere in maniera corretta l'intero processo. Questa situazione invece non accade sul lato customer, perché il lato customer non è governato da un end, quindi da due attività in parallelo ma è governato da uno xor, quindi, quando questo avviene, quindi quando la pizza è ricevuta, automaticamente questo ramo viene disabilitato, perché, come detto solo uno dei due eventi può accadere quando io sono in attesa appunto, nell'event based xor.