Sistemi Informativi Aziendali

Appunti per il corso

Fulvio Corno Marco Torchiano

Politecnico di Torino – Dipartimento di Automatica e Informatica

Versione 1.01

16 novembre 2015

INDICE

CAPITOLO 5

MODELLAZIONE DI PROCESSO

Making the simple complicated is commonplace;
making the complicated simple, awesomely simple, that's creativity
Charles Mingus

5.1 Introduzione

La modellazione di processo permette di comprendere e descrivere le operazioni e le azioni che vengono svolte sui dati da parte del sistema informativo, in seguito anche alle interazioni con gli utenti. Si tratta di un punto di vista non più statico ma dinamico. Dal diagramma di processo è possibile ricavare le relazioni logiche e temporali che il sistema informativo deve rispettare (talvolta dette anche *regole di business*), ad esempio che cosa deve essere fatto prima e cosa deve essere fatto dopo.

La descrizione dei *processi* può avvenire a diversi livelli; ad esempio ai livelli più alti si possono utilizzare diagrammi di tipo *CRASO* (Client Request Activity Organization Output), mentre ad un livello di dettaglio più specifico si può utilizzare il formalismo UML degli *Activity Diagram*.

verificare se viene descritto nel capitolo 2

In questo corso si utilizzeranno gli Activity Diagram (diagrammi di attività): essi rappresentano la sequenza di eventi che accadono nel sistema informativo dal punto di vista generale, considerando anche le azioni dell'uomo che interagisce dall'esterno con il sistema stesso.

Ogni processo avrà quindi un inizio, una fase di sviluppo intermedia e una fine. Nella modellazione, è importante è mettersi dal punto di vista di un'attività o una lavorazione, osservando come viaggia lungo l'arco dell'intero processo, costituendo quindi la descrizione di un flusso di lavorazione (Workflow).

I processi modellati molto spesso coinvolgono le persone, e talvolta la progettazione di un processo ha lo scopo di semplificare e migliorare le attività degli utenti. Affinché l'ottimizzazione dei processi sia possibile, bisogna comprendere a fondo, tramite un processo di documentazione e analisi, come queste persone lavorano e quali impatti la reingegnerizzazione di un processo possa avere sul piano organizzativo.

In sintesi, la modellazione di processo ha come obiettivo quello di descrivere il più precisamente possibile un processo (o workflow). Attraverso un modello di processo sarà possibile comunicare, documentare, analizzare, convalidare il workflow delle operazioni. In aggiunta, qualora modello sia rappresentato attraverso una notazione formale, sarà possibile attraverso opportuni strumenti anche implementare (ossia eseguire, o in fase di simulazione o durante il reale funzionamento del sistema informativo) il processo stesso: in tali casi, la modellazione di processo funge anche da fase di sviluppo effettivo dei SI. Nello specifico, le notazioni adottate per la modellazione di processo possono essere *formali*, ed in tal caso permettere l'eseguibilità del modello stesso, oppure *semi-formali*, che come tali non saranno immediatamente eseguibili, ma nondimeno potranno essere utili come punto di partenza per l'analisi ad alto livello.

Nel caso specifico degli Activity Diagram UML, la notazione permette di essere usata sia in modo formale (ma in questo caso occorre lavorare ad un livello di dettaglio molto elevato), sia in modo

semi-formale, qualora l'obiettivo sia principalmente l'analisi e la specifica, e non ancora l'implementazione. In questo corso seguiremo quest'ultimo approccio, adottando quindi gli Activity Diagram in modo semi-formale.

5.2 Activity Diagram UML

Attraverso il formalismo degli Activity Diagram UML, un processo (in UML detto *activity*)viene modellato attraverso i seguenti elementi:

azioni che rappresentano le singole fasi attraverso cui il processo si sviluppa

regole che descrivono le interdipendente logico-temporali tra le varie azioni (esecuzione in serie, esecuzione in parallelo, ripetizione di azioni precedenti, scelta tra più percorsi alternativi, ...)

responsabilità con le quali si specifica, per ciascuna azione, chi sia la persona, l'ente, il reparto, il servizio, ... responsabile di portare a termine tale azione.

Alcune informazioni aggiuntive sui diagrammi di attività UML, oltre a quelle esposte nel seguito, si possono trovare sul sito *UML Diagrams* [6].

Un activity diagram racchiude in sé due diverse chiavi di lettura, complementari e coesistenti:

descrittivo Come strumento descrittivo, il diagramma ci dice come sono articolati i processi del sistema e l'organizzazione delle varie fasi di processo. In altre parole, *capire quali sono i processi in atto*.

prescrittivo Come strumento prescrittivo, il diagramma può essere utilizzato come allegato tecnico di un contratto che specifichi come deve funzionare il sistema. In altre parole, *descrivere quali processi vorrò mettere in atto*.

Soprattutto nell'ottica di processo descrittivo è importante evitare di introdurre, a livello di modello, dei vincoli che non siano necessari per lo svolgimento effettivo del processo. Ciò significa che se più attività possono essere svolte parallelamente, non si devono modellare attraverso una sequenza. Così facendo, infatti, si forzerebbe il sistema informativo a vincolare un ben preciso ordinamento ordinamento. Rappresentando le azioni in parallelo, invece, si forniscono al SI dei gradi di libertà ulteriori sull'ordine di esecuzione delle operazioni.

5.2.1 Azioni

Le azioni, che come detto rappresentano le singole fasi del processo, possono dividersi in:

1. azioni manuali, compiute esclusivamente da uno o più esseri umani utenti del sistema
2. azioni svolte e gestite dal sistema informativo, con l'ausilio degli utenti o attraverso l'interazione con gli utenti
3. azioni svolte dal sistema informativo in completa autonomia, senza l'intervento umano ed attraverso procedure automatiche.

Occorre prestare attenzione alle azioni del primo tipo (completamente manuali), ed in particolare occorre verificare se tale azioni *lascino traccia* nel sistema informativo. Nel caso in cui delle azioni manuali non lascino traccia nel sistema informativo, e l'utente non interagisca con il sistema informativo, allora tali azioni non devono essere modellate, in quanto il sistema informativo non avrà alcun modo per dedurre se tali azioni siano avvenute o meno, e di conseguenza passare ad eseguire le azioni successive. Vale quindi la seguente regola: una azione sarà rappresentabile in un activity diagram solamente se essa lascia traccia nel sistema informativo. Tutte le azioni che coinvolgono il sistema informativo (quelle di tipo 2. e 3.) evidentemente vi lasciano traccia, e quindi sono sempre modellabili.

Conviene fare chiarezza sul concetto di "lasciare traccia nel sistema informativo," visto che è un concetto che sarà utilizzato molto spesso nell'analisi dei processi. La conoscenza di un sistema informativo è rappresentata dalle informazioni che esso gestisce, ossia dalle informazioni rappresentate

nel modello concettuale (diagramma delle classi). Pertanto, lasciare traccia nel sistema informativo significa che le informazioni presenti nel modello concettuale, dopo l'esecuzione dell'azione, saranno necessariamente diverse rispetto alle informazioni che vi erano presenti prima dell'esecuzione dell'azione. Ad esempio, l'azione avrà creato nuovi oggetti (istanze di classi), avrà aggiornato il valore di alcuni attributi, avrà aggiunto nuove occorrenze di associazioni, oppure avrà cancellato qualche oggetto od occorrenza. In assenza di una qualsiasi modifica, l'effetto dell'esecuzione dell'azione sul sistema informativo sarà nullo (potrà invece avere effetto sugli umani, o su altri sistemi informativi diversi da quello considerato, o sull'ambiente esterno, ecc.) e pertanto non deve essere presente in un activity diagram. È opportuno precisare che, nel compiere questi ragionamenti, occorre considerare il sistema informativo *in senso lato*, dove le informazioni considerate non siano solamente quelle informatiche, ma siano comprese anche le altre forme di codifica dell'informazione (ad esempio, quelle di tipo cartaceo). Un'azione che lascia una traccia su un documento cartaceo può essere modellata (anche perché le informazioni presenti sul modulo cartaceo, che fa parte del sistema informativo ma non del sistema informatico, potrebbero successivamente essere trascritte nel sistema informatico, oppure potrebbero essere riprese come input ad azioni successive). Un'ultima puntualizzazione: le informazioni verbali (le cose dette tra persone) e le conoscenze dei singoli utenti (i loro pensieri) non fanno parte del sistema informativo, in quanto tale informazione non è disponibile.

Riassumendo, quindi: l'activity diagram rappresenta le azioni che lasciano traccia nel sistema informativo (inteso in senso ampio, ossia informatico + cartaceo + tangibile) o lo coinvolgono esplicitamente. In ogni caso, il modello concettuale è unico e condiviso tra tutti gli activity diagram.

La notazione, per tutte le tipologie di azioni, è sempre la stessa (figura 5.1): ogni azione è rappresentata da un rettangolo con gli angoli arrotondati, contenente al proprio interno la descrizione sintetica dell'azione (ossia ciò che avviene all'esecuzione di tale azione).

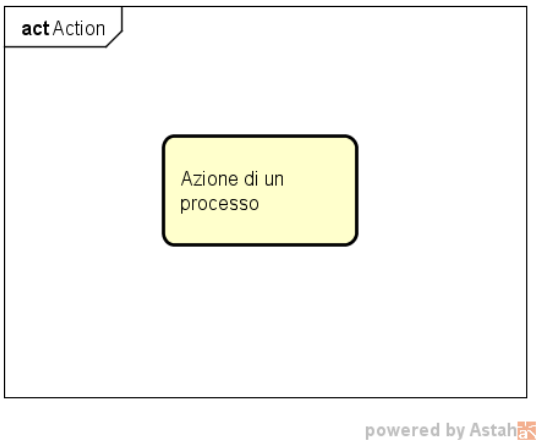

Figura 5.1: Azione in un Activity Diagram

Nella scrittura del testo contenuto nell'azione, occorre accertarsi che sia chiara la *responsabilità* di chi dovrà realizzare tale azione (o comunque attivarsi e/o collaborare affinché venga realizzata). Pertanto, ogni azione dovrà *sempre* avere un soggetto esplicito, con un verbo normalmente in forma attiva e mai impersonale: occorre scrivere "Lo studente prenota l'esame", non "Prenotazione dell'esame" né "L'esame viene prenotato". Nei (rari) casi in cui l'azione non coinvolga alcun utente, perché viene svolta in automatico dal sistema informativo, si userà come soggetto "Il sistema informativo ..." o per brevità "Il SI ...". A tal proposito, si ricorda che tutte le azioni (anche quelle che hanno un utente come soggetto) coinvolgono sempre anche il sistema informativo, e pertanto non è necessario indicarlo: in effetti, l'esempio precedente si potrebbe scrivere come "Lo studente, interagendo con il sistema informativo (o: con il supporto del sistema informativo), prenota l'esame." Ovviamente, per brevità e chiarezza si ometterà sempre la specificazione del ruolo del sistema informativo, ma occorre ricordare che esso è presente, in forma sottintesa.

5.2.2 Nodi iniziale e finale, Token

Le azioni all'interno di un diagramma delle attività sono collegate fra loro e si organizzano in processi caratterizzati da un inizio (nodo di inizio, o di avvio) e una fine (nodo di fine, o di terminazione) con la rappresentazione grafica di Figura 5.2.

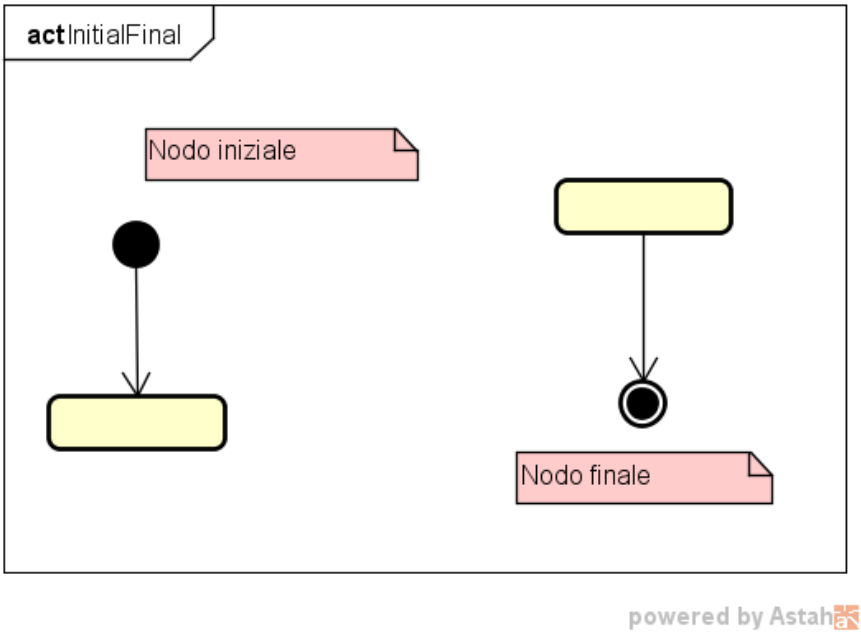

Figura 5.2: Nodi iniziale e finale

Possiamo immaginare che il nodo di partenza generi un "token" (gettone) che attraverserà tutte le fasi intermedie tenendo in memoria le attività già svolte secondo delle regole ben definite, fino a giungere al nodo finale dove il token verrà distrutto.

I diversi costrutti sintattici degli activity diagram specificano delle diverse *regole di viaggio* del token nel suo percorso dal nodo iniziale al nodo finale.

Conviene anche puntualizzare che il diagramma delle attività rappresenta la regola secondo cui il processo si deve svolgere. Il processo, però potrà nella pratica essere svolto molte volte; ad esempio, se il processo rappresenta la procedura per il cambio di residenza di un cittadino, allora il processo sarà attraversato dal token ad ogni nuovo cittadino che vuole cambiare la residenza. Ogni singola attivazione di un processo (o istanza del processo), relativo ad un ben preciso cittadino, è rappresentata da un singolo token. Di conseguenza, è possibile che un processo sia attraversato, allo stesso tempo, da molti diversi token, ciascuno ad un diverso stato di avanzamento, e ciascuno rappresentante un diverso cittadino (o meglio, una diversa pratica di cambio di residenza). Ciascun token procede indipendentemente dagli altri, secondo le regole imposte dall'activity diagram, e porta con sé tutte le informazioni necessarie ad identificare la specifica istanza e non confonderla con le altre.

Ogni activity diagram associa un significato ai token che lo attraversano. Token di natura diversa devono viaggiare in diagrammi diversi. Per questo motivo, per ogni activity diagram, conviene apporre un'annotazione (indicativamente a fianco del nodo iniziale), che ricordi che cosa rappresenta il token di quel processo. Ovviamente processi diversi saranno attraversati da token con significato diverso.

5.2.3 Sequenza

Il concetto di sequenza rappresenta il vincolo per cui un'azione può iniziare solo se un'altra azione è già terminata. La sequenza costituisce lo strumento essenziale per rappresentare l'ordinamento *logico* e *temporale* delle azioni. Temporale in quanto le azioni seguenti nella sequenza inizieranno

certamente ad un istante di tempo successivo alle azioni che le precedono; logico, perché una sequenza implica che l'azione successiva non può iniziare, per qualche motivo legato a ciò che le azioni precedenti devono avere svolto (es. raccolto dei dati, effettuato delle elaborazioni).

La sequenza viene rappresentata, come illustrato in figura 5.3, attraverso una freccia che collega le due attività da compiere in sequenza.

Facendo riferimento alla figura, il token viene creato nel nodo iniziale. Appena creato, immediatamente si sposta all'ingresso della prima azione. Il token potrà rimanere in attesa per tempo indefinito (perché l'utente non è pronto, o il sistema informativo è impegnato in altre cose), finché l'Azione A non verrà iniziata; tuttavia, il fatto che il token abbia raggiunto l'Azione A significa che sono disponibili tutti i dati necessari, e che sono soddisfatte tutte le condizioni richieste, per poter svolgere l'azione.

Ad un certo punto, l'azione verrà iniziata, immaginiamo che il token "entri" nell'azione. Ricordando che nel diagramma possono rappresentare sia azioni che il SI svolge con un operatore umano, sia operazioni che vengono svolte autonomamente dal SI, possiamo aspettarci una diversa velocità di movimento del token: generalmente le operazioni umane richiedono più tempo di quelle automatizzate. Al completamento dell'azione, il token viene "rilasciato" in uscita, e segue il percorso indicato dalle frecce. Ad esempio, appena terminata l'Azione A, immediatamente il token si porta all'ingresso dell'Azione B, ossia abilita all'esecuzione l'Azione B.

Riassumendo, la semantica della connessione in sequenza, si può riassumere come: "L'azione B viene abilitata all'esecuzione solamente dopo che l'Azione A avrà dichiarato di essere completata."

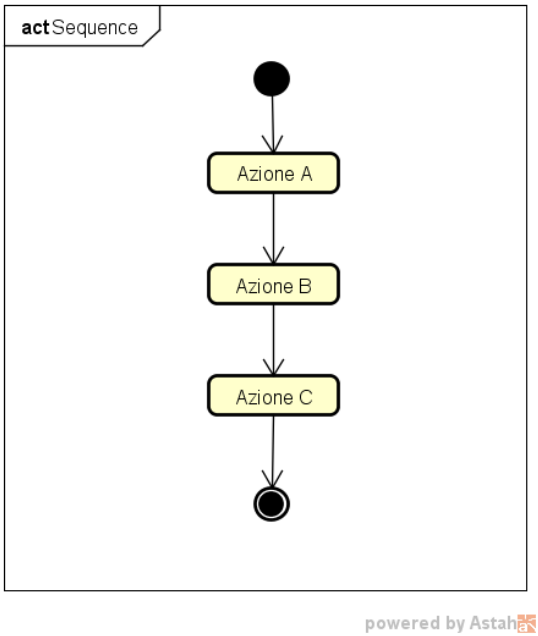

Figura 5.3: Sequenza

5.2.4 Parallelo

L'esecuzione parallela, detta anche *parallel split* oppure *fork* (figura 5.4), rappresenta una condizione in cui alcune azioni sono indipendenti temporalmente tra loro. Non è infatti importante l'ordine con il quale vengono eseguite Azione B ed Azione C: possono essere svolte insieme, una dopo l'altra o iniziare e finire in momenti diversi, l'importante è che siano svolte tutte. La sequenza in parallelo viene rappresentata attraverso un nodo di fork, che rappresenta il punto dove il flusso (rappresentato dal token) si divide in diverse attività indipendenti dal punto di vista temporale. Il nodo fork ha una sola freccia entrante, che riceve il token del processo, ed in uscita ha due o più frecce uscenti, su ciascuna delle quali viene posto un "clone" del token (non si creano quindi nuovi token, ma si divide

quello esistente). Il passaggio dal nodo fork è da intendersi come azione istantanea (non introduce ridardi e non "ferma" il token).

Quali sono le condizioni affinché due (o più) azioni possano essere messe in parallelo? Poiché il costrutto del parallel split specifica esplicitamente l'assenza di una qualunque ipotesi sull'ordinamento delle azioni, esso può essere usato quando non esiste alcun tipo di vincolo che richieda un ordinamento nell'esecuzione. I rami paralleli sono assolutamente indipendenti, ed in particolare deve essere possibile portare a termine ciascuna azione senza sapere se le altre azioni siano già iniziate o se si siano già concluse. Per la precisione, ciascuna azione parallela deve avere a disposizione i dati su cui deve operare già dal momento in cui il token raggiunge il nodo fork, e di norma modificherà delle informazioni disgiunte da quelle modificate dalle altre azioni (altrimenti si potrebbero creare dei conflitti o delle situazioni di non determinismo, in cui il risultato finale potrebbe dipendere dall'ordine di esecuzione).

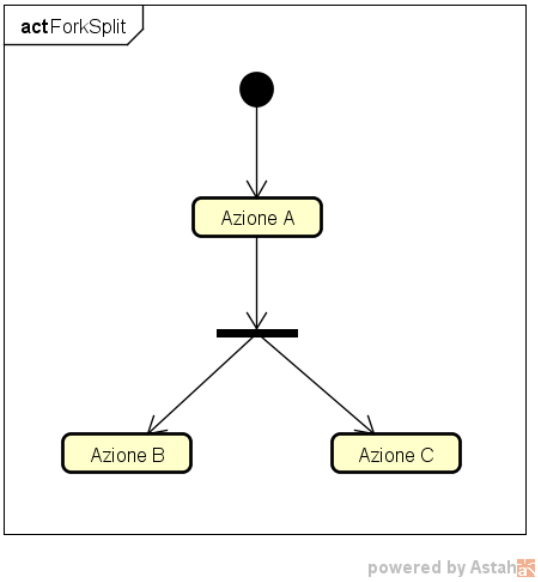

Figura 5.4: Esecuzione parallela (Fork)

Poiché il nodo fork crea più flussi di esecuzione indipendenti (e possibilmente paralleli), emerge spesso la necessità di "riunire i vari flussi, in modo da continuare l'esecuzione dell'attività solamente quando tutte le azioni parallele sono definitivamente concluse. A questo scopo si può utilizzare il costrutto del *punto di sincronizzazione* (detto anche *join*), il quale (figura 5.5) riceve in ingresso più frecce, che porteranno in tempi diversi più token, e presenta in uscita una sola freccia.

I vari cloni del token che hanno attraversato le varie attività parallele si riuniscono nel punto di sincronizzazione, e solamente quando tutti i token hanno raggiunto le varie frecce in ingresso, viene ricostruito in uscita il token iniziale. Qualora un'azione termini prima di un'altra, il relativo token rimarrà "in attesa" all'ingresso del nodo join, finché anche tutti gli altri token non saranno giunti; a questo punto, i token in ingresso vengono fusi tra loro, ed immediatamente viene liberato il token in uscita.

Come è facile intuire, i nodi join e fork molto spesso vengono utilizzati in coppia. Come si vedrà più avanti, negli activity diagram strutturati è necessario che ad ogni nodo fork corrisponda un nodo join che ne raccolga (tutti e soli) i flussi di esecuzione paralleli. Nella sezione 5.3 si vedranno alcune eccezioni a questa regola generale, utilizzate per modellare situazioni particolari.

Esempio La figura 5.6 riporta un esempio completo di applicazione dell'esecuzione parallela. Nel diagramma, si vede come la consegna della merce può avvenire solamente dopo che la merce è stata preparata e che la fattura sia pronta. L'ordine di queste due operazioni, l'una svolta dal magazzino e l'altra dall'amministrazione, non è importante: possono avvenire contemporaneamente, oppure in tempi diversi.

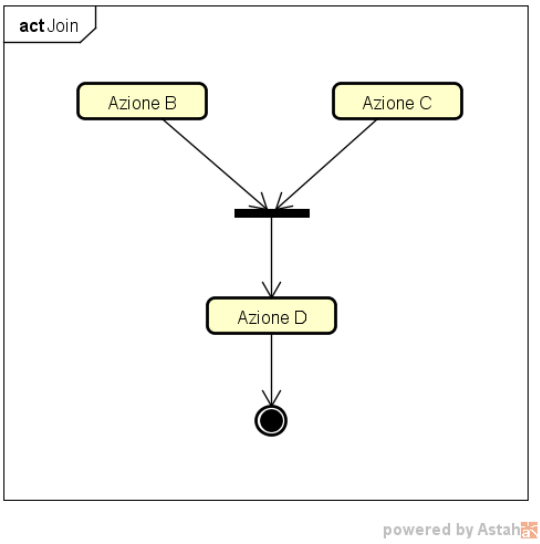

Figura 5.5: Punto di sincronizzazione (Join)

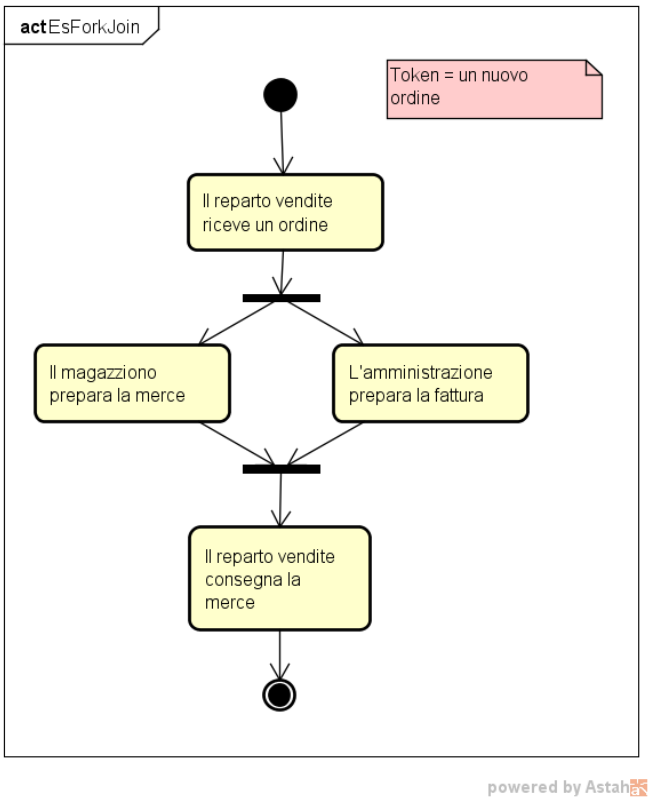

Figura 5.6: Esempio di esecuzione parallela

5.2.5 Scelta esclusiva

La scelta esclusiva (detta anche Choice, o If) permette di discernere fra due (o più) attività in una situazione dove è possibile eseguire l'una o l'altra, ma non entrambe. In tal caso si modella l'esistenza di diversi cammini che il token può seguire instradando il processo in una direzione piuttosto che in un'altra. Per rappresentare la scelta esclusiva si utilizza un simbolo a forma di rombo (figura 5.7). Nel nodo di scelta vi è una sola freccia entrante, e due o più frecce uscenti.

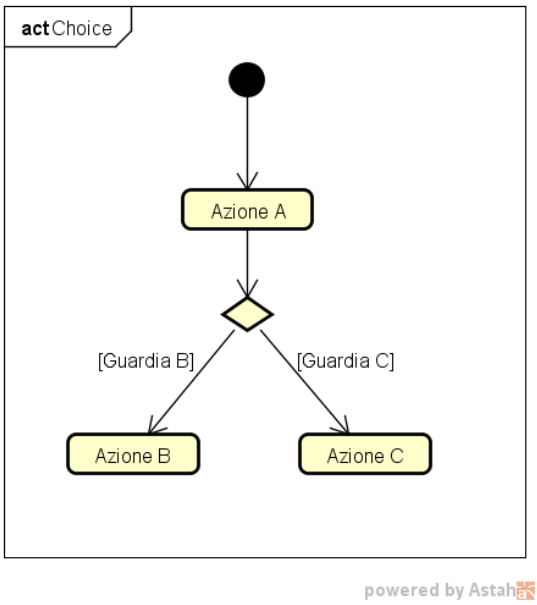

Figura 5.7: Punto di scelta (Choice)

Per decidere quale percorso dovrà intraprendere il token si consultano le informazioni presenti nel modello concettuale, che quindi dovranno già essere disponibili nel momento della scelta. Ogni ramo che esce dal punto di scelta deve essere annotato con un'espressione booleana, detta *guardia*, che rappresenta la condizione sotto cui il token prosegue lungo quel ramo. L'espressione rappresentata dalla guardia dovrà essere calcolabile utilizzando informazioni già presenti nel sistema informativo (in particolare, nel modello contettuale) nel momento in cui il token raggiunge il nodo: di conseguenza, il passaggio del token attraverso il punto di scelta avviene istantaneamente, e non vi è alcuna attesa.

Con questo costrutto il token non verrà sdoppiato, ma dovrà proseguire lungo una ed una sola via d'uscita. Ciò è possibile solamente se le condizioni di guardia sono esaustive e mutuamente esclusive (ossia, non è permesso che siano tutte false, e non è permesso che due di esse siano vere contemporaneamente).

Molto spesso, una volta terminata l'esecuzione su uno dei rami alternativi, il processo deve proseguire in modo comune, indipendentemente dalla scelta fatta: è necessario quindi un costrutto che permetta di "ricongiungere" i diversi flussi di esecuzione, e proseguire con un flusso unico. A questo scopo esiste un nodo di *unione* (o ricongiungimento, o *merge*), la cui rappresentazione è ancora a forma di rombo (figura 5.8) in cui però convergono due o più frecce in ingresso, mentre vi è una sola freccia in uscita. Il nodo di ricongiungimento non compie alcuna elaborazione né scelta, si limita a fare scendere lungo l'ultima uscita il token, non appena esso si presenta in uno qualsiasi degli ingressi. Il nodo di unione non comporta alcun ritardo di esecuzione né può fermare il token. Non vi sono condizioni di guardia associate al nodo merge.

Nella modellazione dei processi, anche nel caso dei nodi choice e merge, si cercherà di lavorare in maniera strutturata, in cui a ciascuna scelta corrisponda un ben preciso punto di ricongiungimento, in modo da ottenere processi in cui ciascuna porzione possegga un solo punto di ingresso ed un solo punto di uscita.

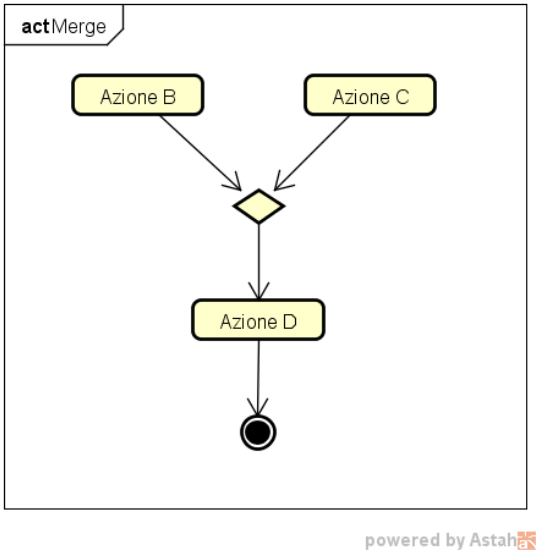

Figura 5.8: Punto di ricongiungimento (Merge)

Esempio L'esempio riportato in figura 5.9 illustra un possibile utilizzo del costrutto di scelta e ricongiungimento. In tale esempio, dopo che l'ordine è stato preparato, vi sono due modalità di spedizione possibili: per corriere espresso oppure per posta ordinaria. Il nodo si scelta discrimia tra queste due alternative valutando le espressioni di guardia "Spedizione rapida" e "Spedizione normale": tali espressioni devono essere calcolabili facendo riferimento ad informazioni che siano già presenti nel modello concettuale (ad esempio, sono state acquisite durante l'azione di ricezione dell'ordine, e la classe che rappresenta l'ordine conterrà un attributo di tipo booleano che ricorda tale scelta).

5.2.6 Attesa

Nei costrutti visti finora, il processo può essere eseguito in modo autonomo, attraverso l'interazione tra il sistema informativo e gli utenti. Fatti salvi i tempi di elaborazione richiesti dal sistema (quasi sempre trascurabili) e le disponibilità degli utenti (che dipendono da quando potranno occuparsi dell'azione da compiere), l'esecuzione del processo può proseguire senza altre pause o interruzioni.

Vi sono però situazioni nelle quali il processo deve attendere il verificarsi di qualche condizione esterna, senza la quale non deve procedere (anche se avesse già tutti i dati a disposizione). Vi sono diverse tipologie di "eventi esterni" che possono influenzare l'esecuzione del processo, i più semplici dei quali sono legati allo scorrere del *tempo*.

Per modellare il fatto che il processo deve rimanere fermo in attesa di un determinato istante di tempo si può usare un *nodo di attesa* (Time Event), che viene rappresentata attraverso un simbolo che ricorda la forma di una clessidra (figura 5.10).

Nel momento in cui il token raggiunge il nodo di attesa, esso viene "fermato" e trattenuto nel nodo stesso, fino al raggiungimento dell'istante di tempo indicato sul nodo; raggiunto tale istante, il token viene rilasciato e prosegue il cammino. Il tempo nel quale il token rimane nel nodo di attesa è da intendersi come tempo non produttivo, ossia nessuno (né utenti né sistema informativo) compie alcuna azione.

Il nodo di attesa può specificare il tempo in due modi diversi:

1. in modo *assoluto*, specificando direttamente l'istante (giorno, ora, minuti, a seconda del livello di precisione richiesto) in cui il token dovrà essere liberato (es. "ore 20:00", "lunedì alle 9:00", "il 25 dicembre 2016");

2. in modo *relativo*, specificando la quantità di tempo per cui il token dovrà rimanere bloccato: tale quantità di tempo sarà misurata a partire dall'istante di ingresso del token nel nodo (es. "5 minuti", "2 giorni lavorativi", "4 ore e 30 minuti").

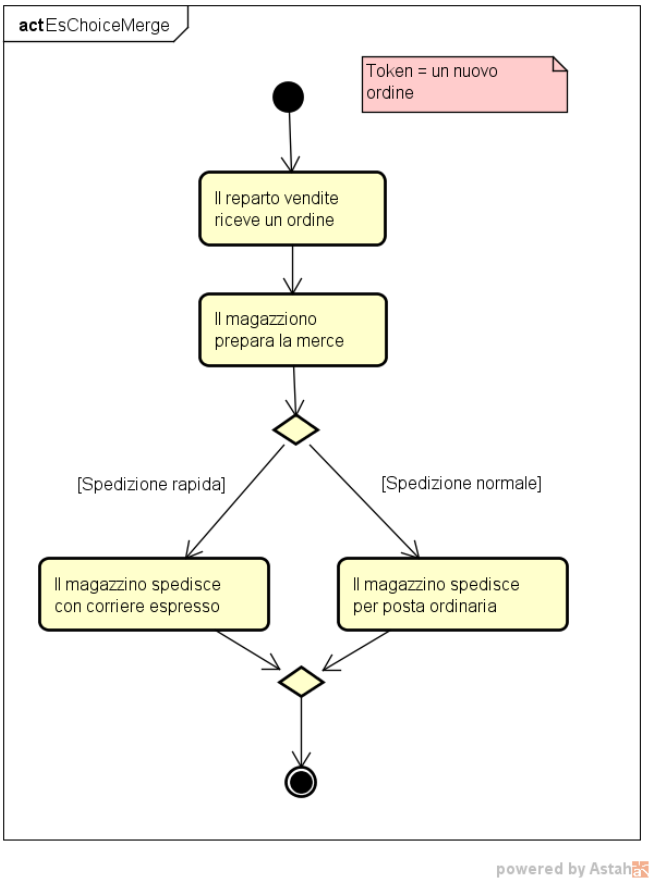

Figura 5.9: Esempio di scelta esclusiva

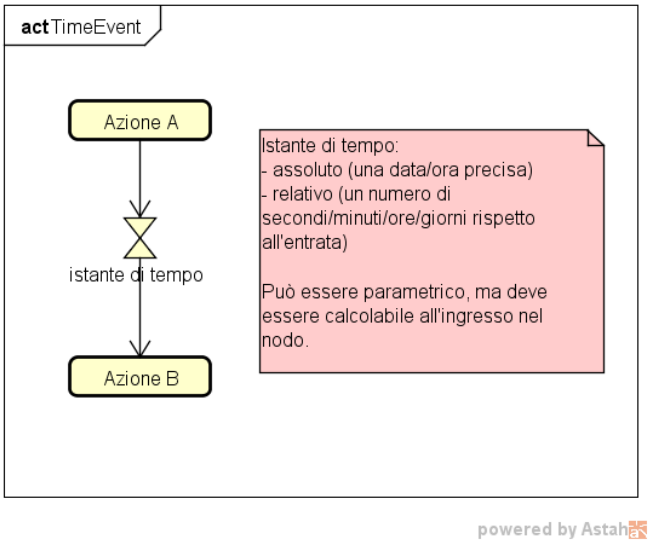

Figura 5.10: Nodo di attesa (Time Event)

Nel primo caso, il token rimarrà fermo una quantità di tempo variabile, poiché è fisso l'istante di uscita, ma può variare l'istante di ingresso. Nel secondo caso, il token rimane fermo una quantità di tempo fissa, ma l'istante di uscita varia in funzione dell'instante di ingresso.

Solitamente, nel primo caso (istante assoluto di tempo di uscita) viene utilizzata una formulazione che indica un istante di tempo *ricorrente* (ossia che si ripete ogni giorno, ogni settimana, ogni anno, ...), poiché specificare un tempo *irripetibile* è poco utile. Ad esempio, il 31/12/2015 alle 9:00:00 si verifica un sola volta nella storia dell'universo, ed è improbabile che il sistema informativo debba svolgere delle operazioni una sola volta nella storia. Ovviamente vi sono delle eccezioni: in un sistema informativo nato per gestire Expo 2015, le date di apertura e di chiusura dell'evento dovranno essere trattati come istanti unici ed irripetibili.

La specifica del tempo può avvenire anche in modo parametrico ("dopo X giorni lavorativi," "aspetta Y minuti"), purché il valore numero di tali parametri sia determinabile (partendo dalle informazioni in possesso al sistema informativo, quindi nel modello concettuale) nell'istante in cui il token raggiunge il nodo. In altre parole, nel momento stesso in cui il token raggiunge il nodo di attesa, è possibile determinare con precisione e senza eccezioni l'istante di tempo assoluto il cui il token verrà rilasciato (eventualmente facendo gli opportuni calcoli). Nessun altro avvenimento potrà liberare il token, se non il raggiungimento del tempo così determinato.

5.2.7 Segnali ed eventi

Oltre alla variabile temporale, possono esistere altre tipologie di eventi esterni che bloccano il token all'interno del processo finché non sia disponibile una certa informazione. In questo caso l'evento che permette al processo di proseguire è la ricezione di un'informazione proveniente dall'esterno. In questo contesto, con "esterno" si intendono altri sistemi informativi (della stessa azienda o di aziende diverse), oppure altri processi dello stesso sistema informativo. Il tempo di attesa dipenderà quindi da quando l'informazione esterna diventerà disponibile.

Queste situazioni sono modellate all'interno degli Activity Diagram attraverso il concetto di segnale, che viene scambiato tra due attività, una delle quali ha il ruolo di mittente (sender) e l'altra di destinatario (receiver). La figura 5.11 illustra la notazione utilizzata: l'invio di segnale è rappresentato tramite un rettangolo con una freccia uscente (a sinistra nella figura), mentre l'attesa e la ricezione di un segnale sono indicate con un rettangolo con una freccia entrante (a destra). Si noti che la figura non rappresenta un singolo processo, ma due processi distinti (ciascuno con i propri noti di inizio e di fine, in quanto di norma i segnali sono scambiati tra processi diversi).

I paragrafi successivi puntualizzano il comportamento dei vari elementi coinvolti nel meccanismo di invio e ricezione.

Signal Sending

Il nodo di invio di segnale rappresenta il fatto che il processo corrente invia un messaggio destinato ad un processo diverso dello stesso sistema informativo, oppure ad un sistema informativo diverso. Le eventuali informazioni associate al segnale devono essere già presenti nel modello concettuale nel momento in cui il nodo viene raggiunto dal token. L'azione di invio del segnale deve considerarsi atomica (non divisibile), istantanea (non rallenta il token) ed asincrona (non si attende il ricevimento del messaggio né tantomeno una conferma di ricezione). Non è normalmente necessario specificare il destinatario del messaggio (processo ricevente o sistema informativo ricevente), in quanto si assume che il segnale sia disponibile a tutti gli interessati (trasmissione in broadcast). L'invio di un segnale non prevede interazioni con gli utenti del sistema, è un'azione da intendersi come automatica, non appena il token raggiunge il nodo.

Signal Receipt

Il nodo di ricezione di un segnale rappresenta un punto bloccante per il token: l'esecuzione del processo rimane ferma (senza fare nulla) fino al momento in cui viene ricevuto il segnale desiderato. Una volta ricevuto il segnale, l'informazione associata allo stesso (se presente) viene incorporata dal sistema informativo (aggiornando il modello concettuale), ed il token può proseguire con l'azione

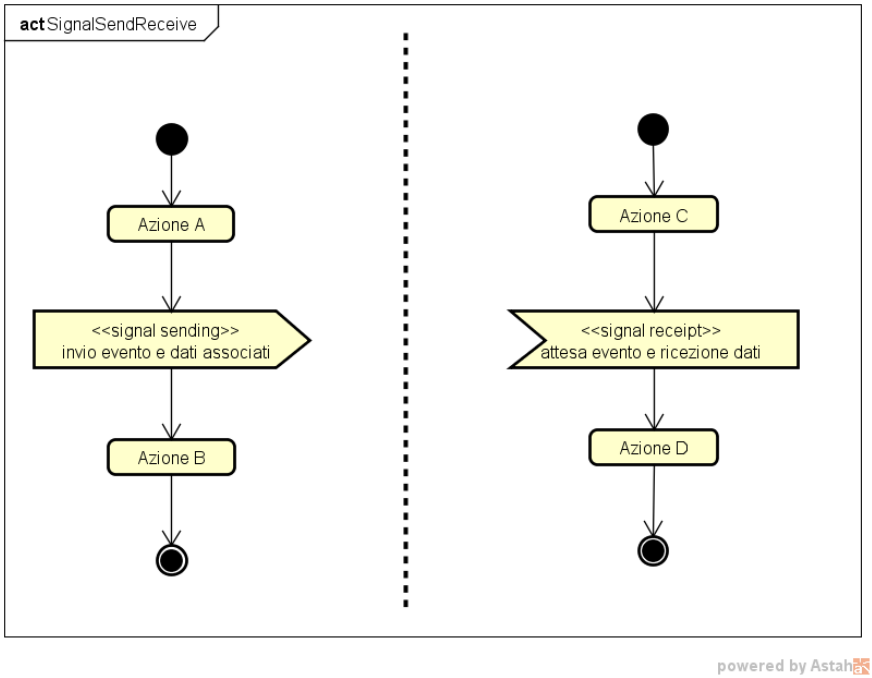

Figura 5.11: Invio e ricezione di segnali

seguente. Normalmente si immagina che i segnali ricevuti vengano accodati, per cui se un segnale dovesse giungere nel momento in cui il processo non è fermo in attesa, allora tale segnale viene salvato e verrà consegnato nel momento in cui il processo arriverà alla nodo di ricezione.

Segnali

Il segnale è un messaggio che viene trasferito dal nodo di Signal Sending ad uno o più nodi di Signal Receipt. La ricezione del segnale non è contemporanea alla sua trasmissione, in quanto non è detto che il processo ricevente sia pronto nel momento in cui il segnale viene inviato. Lo scambio di segnali può svolgere le due seguenti funzioni:

Sincronizzazione quando l'informazione importante da trasmettere è il fatto che si è raggiunto un determinato punto nell'elaborazione. Un segnale di sincronizzazione non porta con sé dati o informazioni aggiuntive, ma solamente l'informazione che il punto di invio segnale è stato raggiunto. Questa informazione può essere utilizzata per "sbloccare" altri processi che erano in attesa e che, per poter procedere, dovevano avere la certezza che alcune operazioni fossero completate. È analogo ad un messaggio "vai pure" (e, sottinteso, tu sai già dove e perché).

Scambio dati nel caso in cui il segnale, oltre a trasportare delle informazioni di sincronizzazione, trasporta anche delle informazioni specifiche (ad esempio, quelle corrispondenti all'istanza di una classe del modello concettuale). Il tal caso, al ricevimento del segnale, oltre all'informazione temporale (sul raggiungimento del punto di invio del segnale), il processo ricevente può incorporare anche una copia dei dati inviati. Questo tipo di segnali è essenziale nel caso di scambio di informazioni tra sistemi informativi diversi, in quanto essi non condividono l'accesso allo stesso sistema informativo.

Combinazioni ammesse

Come si è detto, i segnali vengono sempre scambiati con qualche entità "esterna" al processo che li genera (o li riceve). Per maggior chiarezza, conviene esplicitare alcune regole che permetteranno di comprendere quando sia lecito modellare uno scambio di informazioni attraverso un invio di segnali. Si possono evidenziare le seguenti regole di modellazione:

1. i segnali sono scambiati tra sistemi informativi. L'utente (umano) non potrà **mai** essere la sorgente di un segnale o la destinazione di un segnale. Nelle attività che coinvolgono l'interazione con l'utente, si utilizzi sempre il nodo Azione.

2. i segnali possono essere scambiati tra *diversi processi* (diversi activity diagram) dello stesso sistema informativo. In tal caso lo scambio è relativo ad informazioni di sincronizzazione, in quanto i processi comunicanti hanno accesso allo stesso modello concettuale come mezzo per la condivisione di informazioni.

3. nel caso di segnali scambiati tra diversi processi dello stesso sistema informativo, occorre *mantenere la corrispondenza* tra segnali inviati e segnali ricevuti: per ogni invio di segnale deve esserci (almeno) un nodo di ricezione, ed ogni nodo di ricezione deve attendere segnali generati da (almeno) un nodo di invio.

4. i segnali possono essere scambiati tra *sistemi informativi diversi*. Abbiamo quindi due casi: un segnale inviato dal sistema informativo che stiamo descrivendo (nodo signal sending) sarà ricevuto da un qualche sistema informativo esterno, oppure un qualche sistema informativo esterno invierà segnali al nostro sistema informativo, che li riceverà con un nodo di signal receipt. Pertanto, in questi casi, i nodi di invio e ricezione non sono bilanciati. In tutti questi casi, la comunicazione avviene in modo automatico, e senza coinvolgere gli utenti (né dell'uno, ne dell'altro sistema informativo). In questi tipi di comunicazione, solitamente vengono anche scambiati dei dati (delle informazioni associate all'evento), che è opportuno specificare nel testo del nodo (ad es: "invia le informazioni sul pagamento") e che dovranno essere rappresentabili nel modello concettuale.

5. salvo rari casi dovuti a particolari esigenze di modellazione, non di dovrà mai usare la coppia invio/ricezione di segnale all'interno dello stesso activity diagram: cercare di utilizzare il più possibile i costrutti per il controllo di flusso (sequenza, parallelo, scelta) per i fini di sincronizzazione all'interno dello stesso processo. In altre parole: la sincronizzazione tra le attività dello stesso diagramma avviene attraverso i costrutti base (sequenza, fork/join, choice/merge), la sincronizzazione tra activity diagram diversi avviene attraverso invio/ricezione di segnali (messaggi).

6. i segnali devono sempre rappresentare un'informazione nota al sistema informativo (o ad altri sistemi informativi), non è possibile rimanere in attesa di un segnale per un evento esterno che non sia gestito da un sistema informativo (ad es: "ha smesso di piovere" non è un segnale ricevibile, a meno che non abbiamo una centralina meteo connessa al nostro sistema, o ad altro sistema che ci invii questa informazione).

5.2.8 Nodi oggetto

Esiste la possibilità di specificare all'interno del processo anche i dati che vengono scambiati tra le azioni, fornendo così un livello di documentazione aggiuntiva. È infatti vero che tutte le azioni descritte in ogni activity diagram possono accedere, nella loro esecuzione, a qualsiasi informazione presente nel modello concettuale. Nella pratica, però, ciascuna azione dovrà lavorare su un ben preciso insieme di istanze classi, leggendone le informazioni o modificandole o creandone di nuove. Per chiarezza, quindi, può risultare utile rendere espliciti quali siano i dati creati/modificati da un'azione, oppure i dati utilizzati da un'azione. Ad esempio, la valutazione di un esame può portare a un oggetto di tipo intero che poi verrà utilizzato durante la registrazione.

In questi casi si possono utilizzare i *Nodi Oggetto* (Object Nodes), che si rappresentano mediante un rettangolo, utilizzando la stessa notazione già vista nel Capitolo **??** nella sezione **??** ed illustrata nella figura **??**. Ricordiamo che un nodo oggetto (o istanza) contiene al suo interno il nome della istanza stessa, e la specifica del tipo di dato (o classe) che tale istanza rappresenza, separati dal simbolo " **:** ".

Nel diagramma delle attività, graficamente (figura 5.12) si inserisce l'istanza come parte di un arco che congiunge due attività. Nel caso della figura, stiamo dicendo che l'*Azione A* sta producendo una istanza, denominata *informazione*, che rappresenta dati del tipo *ClasseTipo*. Allo stesso tempo, stiamo dicendo che l'*Azione B* riceverà ed utilizzerà tale informazione.

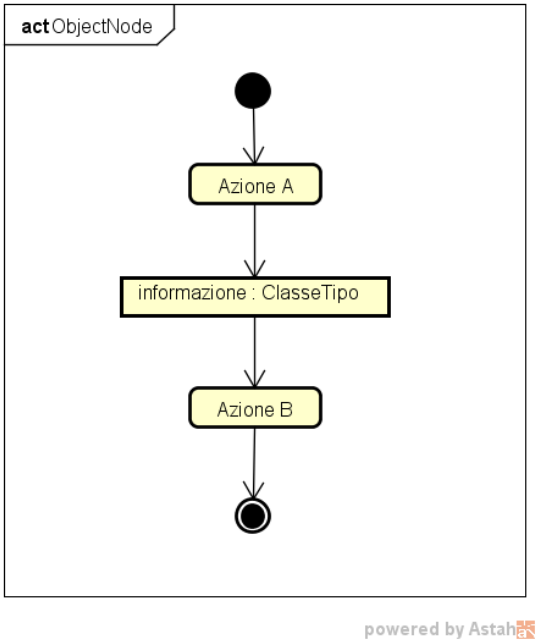

Figura 5.12: Nodo oggetto

Si può visualizzare ciò che succede immaginando che il nostro token, quando attraversa un'azione, venga caricato con un "pacchetto" che contiene l'istanza specificata. Tale istanza viene poi trasportata, dal token stesso, verso l'azione successiva. Gli oggetti possono essere di tipo primitivo, oppure derivanti da classi già definite nel modello concettuale.

Una notazione alternativa, del tutto equivalente, per rappresentare la stessa informazione è illustrata in figura 5.13. In questo caso si utilizzano dei piccoli quadratini, detti "piedini" (Input Pin o Output Pin) che rappresentano le interfacce di ingresso e di uscita dell'azione. I Pin sono annotati con il nome ed il tipo dell'istanza, e devono ovviamente coincidere sui due lati della freccia. L'utilizzo di questa notazione in luogo della precedente è unicamente determinato da ragioni estetiche o grafiche.

5.2.9 Connettori

Per ragioni grafiche, e per semplificare il layout di activity diagram particolarmente estesi, è possibile inserire dei "rimandi" da una parte all'altra del foglio. Tali rimandi si rappresentano attraverso dei nodi connettori, che sono piccoli circoli contenenti una lettera od un numero (figura 5.14). Quando il flusso di controllo (e quindi il token) entra in un connettore (mediante una freccia entrante), esso viene immediatamente "teletrasportato" sul connettore di ugual nome nello stesso activity diagram, e da lì prosegue un suo percorso seguendo la freccia uscente.

5.2.10 Swimlane

Come si ricorderà dalla discussione sul nodo azione (sezione 5.2.1), è fondamentale che a ciascuna azione sia attribuita chiaramente la responsabilità della persona, gruppo o unità organizzativa che ne deve curare l'esecuzione (ad esempio, interagendo con in sistema per portarla a termine).

Per semplificare la notazione, si può pensare di raggruppare le azioni in funzione del soggetto responsabile: tutte le azioni attribuite allo stesso responsabile, vengono aggregate in un'unica *partizione* (partition). Il concetto di partizione è di tipo astratto, e come tale può essere rappresentato, graficamente, in modi diversi. Il metodo più utilizzato prevede di rappresentare le partizioni attraverso delle "corsie" parallele (orizzontali o verticali). Ogni corsia (in inglese "swim lane," facendo riferimento alle corsie di nuoto in una piscina) sarà relativa ad una determinata responsabilità (unità organizzativa o utente).

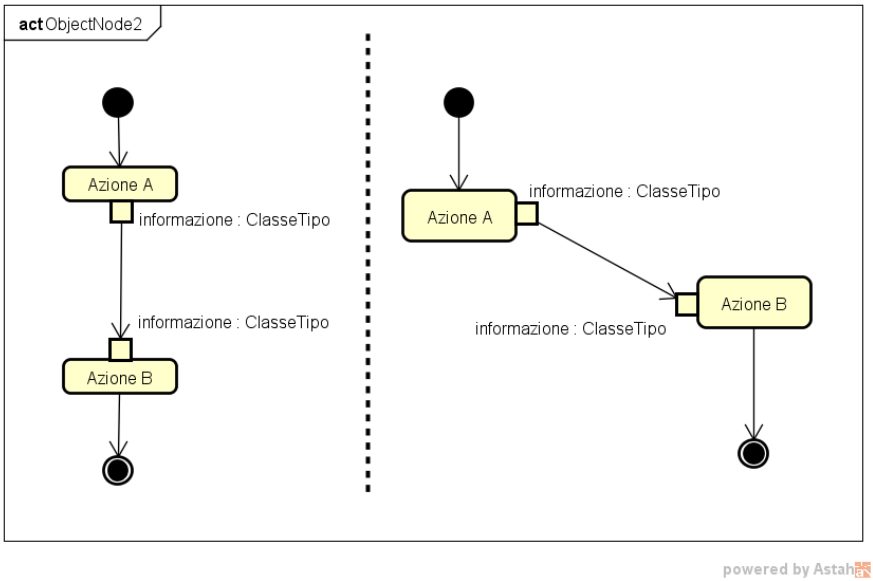

Figura 5.13: Nodo oggetto (notazione alternativa)

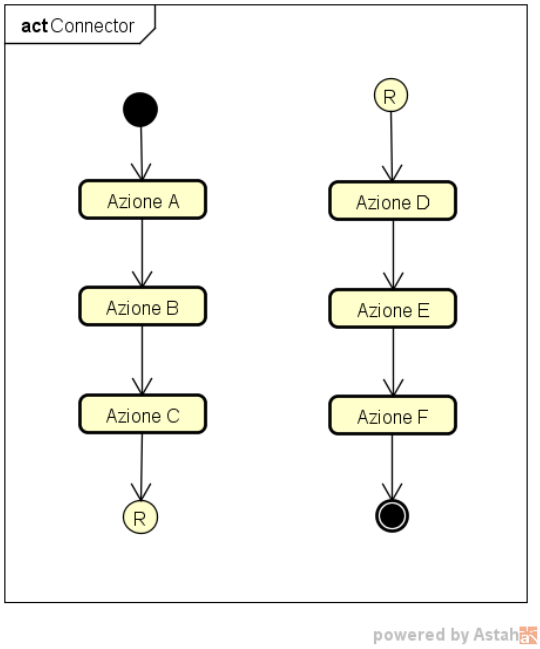

Figura 5.14: Connettore

Le Swimlane sono quindi un costrutto puramente grafico per rappresentare le responsabilità delle varie azioni. Il disegno viene diviso in corsie e ogni corsia viene associata ad un attore. Con questa rappresentazione grafica, tutte le azioni di competenza i tale soggetto vengono inserite nella sua corsia, rendendo superflua l'indicazione del soggetto all'interno dell'azione stessa.

Ad esempio, si consideri il semplice activity diagram di figura 5.15, che rappresenta una sequenza di quattro azioni. Secondo le regole espresse nella sezione 5.2.1, ciascuna azione è descritta esplicitando il soggetto responsabile (Cliente o Fornitore, in questo caso).

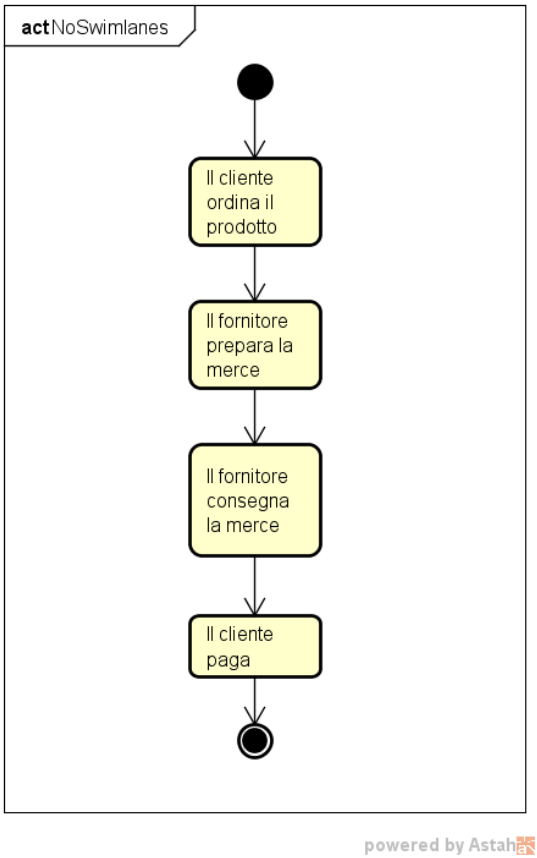

Figura 5.15: Esempio di attività sequenziale

Lo stesso diagramma, se si partizionano le azioni in funzione del responsabile, sarà composto da due partizioni, una per il cliente ed una per il fornitore. Tutte le azioni del cliente saranno raggruppate sotto la prima partizione, e potremo sottindendere il soggetto nel testo descrittivo dell'azione; analogamente si procede per il fornitore. Se si adotta una notazione con partizioni verticali, si ottiene il risultato illustrato in figura 5.16. È del tutto equivalente lavorare con partizioni orizzontali, la cui rappresentazione è in figura 5.17.

Si può notare come la notazione delle swimlane permetta anche di identificare, in modo visualmente immediato i passaggi di responsabilità tra diversi attori, ed in in certo modo le comunicazioni tra tali attori, semplicemente identificando le frecce che attraversano il confine tra due corsie.

In accordo con quanto già discusso nella sezione 5.2.1, le partizioni potranno corrispondere ai diversi utenti del sistema, ed eventualmente potrà esserci un'ulteriore partizione denominata "Sistema Informativo" o "Sistema," riservata alle azioni svolte autonomamente dal sistema informativo, senza l'intervento dell'utente.

Infine, conviene evidenziare che il concetto di partizionamento si applica *esclusivamente alle azioni*, in quanto sono l'unico costrutto a cui si attribuisce una responsabilità. Tutti i restanti nodi di controllo del flusso (nodo iniziale, nodo finale, fork, join, scelta, ricongiungimento, attesa, invio e ricezione di segnale) non appartengono ad alcuna partizione. Dal punto di vista grafico, potranno

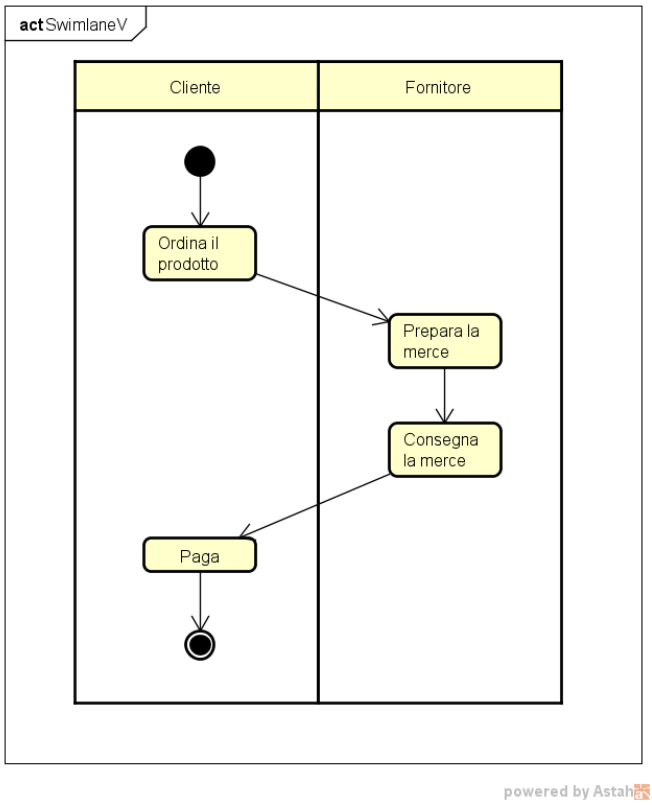

Figura 5.16: Esempio di partizione con swimlane verticali

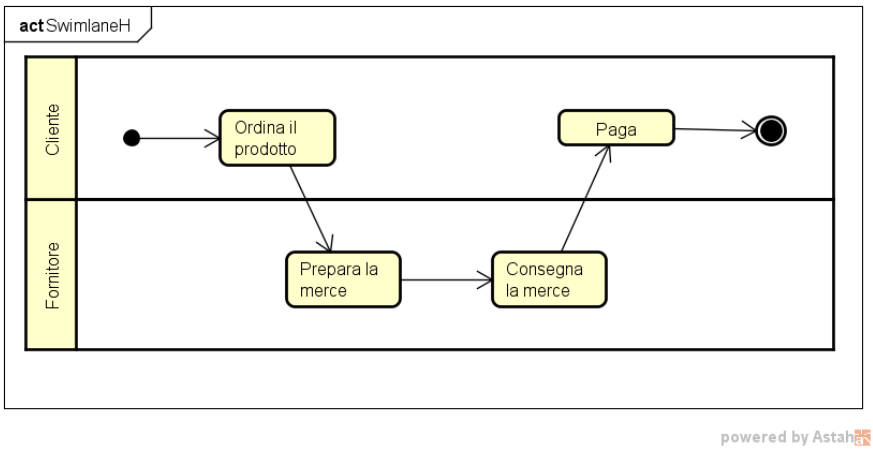

Figura 5.17: Esempio di partizione con swimlane orizzontali

essere disegnati in una qualsiasi delle partizioni, facendosi guidare esclusivamente da criteri di tipo estetico e di chiarezza del disegno. Ad esempio, solitamente il nodo iniziale viene posto nella stessa corsia della prima azione del processo, anche se ciò non ha alcun significato particolare.

5.2.11 Azione complessa

Al crescere della complessità degli processi, risulta difficile descrivere l'intero comportamento in un unico activity diagram. A tale scopo, si possono rappresentare delle azioni complesse, in cui la descrizione del comportamento viene delegato ad un secondo activity diagram, subordinato al primo. Tecnicamente si parla del nodo *Call Behavior Action*, ossia la cui azione corrisponde a "richiamare" (call) un diverso "comportamento" (behavior), ossia un diverso diagramma di attività. Per semplicità, parleremo anche di nodo *azione complessa*, intendendo che l'azione rappresentata dal singolo nodo non è più semplice (o atomica), ma è più articolata, e pertando descritta altrove. Non si tratta infatti di un nodo atomico, ma di un nodo che a sua volta verrà espanso in più rami in un nuovo foglio, senza complicare la rappresentazione grafica del processo. La notazione per un nodo di azione complessa è simile a quella di un normale nodo di azione, ma con un simbolo simile ad un rastrello (⫛) posizionato in basso a destra.

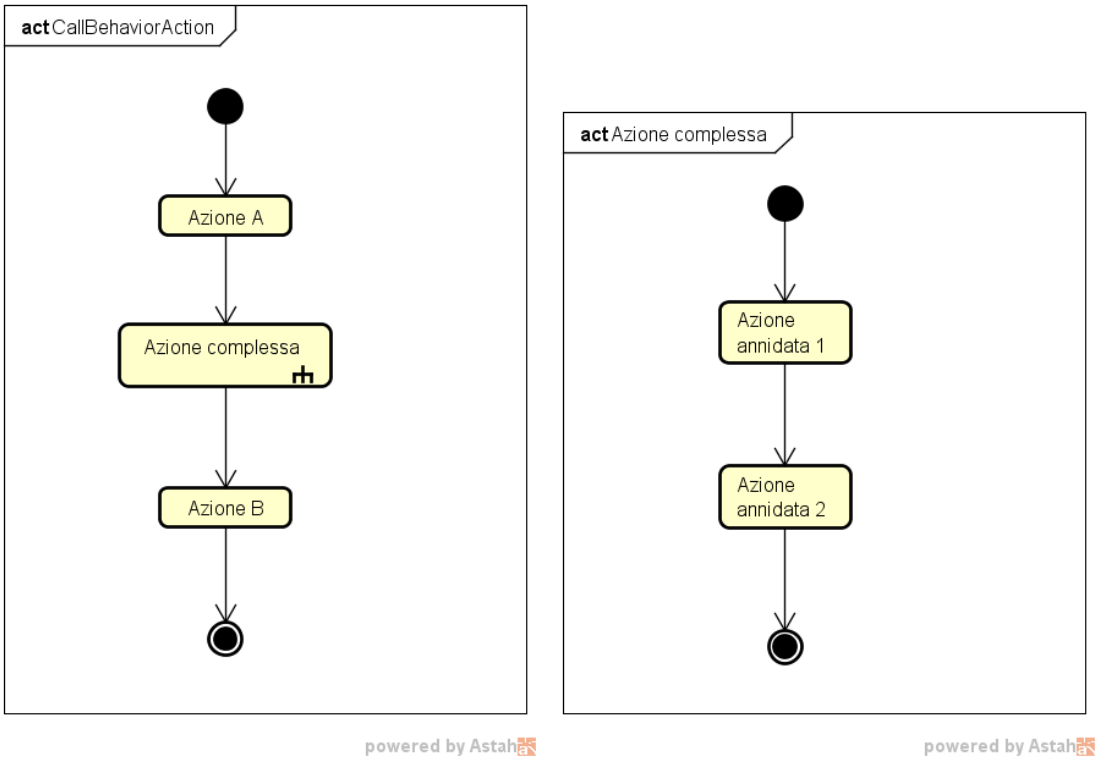

Figura 5.18: Esempio di azione complessa (a sinistra) e relativo diagramma di attività richiamato (a destra)

Il significato di questa notazione è analogo alla chiamata di una procedura, o funzione, in un linguaggio di programmazione. Nel momento in cui il token raggiunge un'azione complessa, allora viene avviato un nuovo activity diagram, il cui nome coincide con il nome dell'azione complessa. Ad esempio nella figura 5.18, nel diagramma di sinistra è presente un nodo denominato "Azione complessa", che corrisponde al diagramma di attività di destra, il cui nome è appunto "Azione complessa". Il diagramma così richiamato verrà eseguito con le normali regole di propagazione del token, e può a sua volta contenere qualsivoglia costrutto di controllo. Solo quando l'attività chiamata sarà terminata, avendo il relativo token raggiunto il nodo terminale, allora l'attività chiamata viene chiusa, ed il token viene liberato in uscita nell'azione complessa del processo principale.

Oltre alla possibilità di disegnare diagrammi più ordinati, i nodi di azione complessa possono essere utilizzati anche qualora si intenda "isolare" il token: infatti nell'attività richiamata potranno esserci operazione di creazione o distruzione di token, che però saranno limitati alla copia locale e non influenzano il token del processo primario (che rimane in attesa all'interno dell'azione complessa, fintantoché il processo chiamato non termini, per un qualsiasi motivo). Nei pattern di modellazione presentati nella sezione 5.3 si troveranno parecchi esempi di questa tecnica.

5.2.12 Eliminazione del token

Nella sezione 5.2.2 si era introdotto il nodo finale di un activity diagram, rappresentato da due cerchi concentrici (⊙). Quando il token raggiunge il nodo finale (formalmente detto *Activity Final Node*), l'intera attività viene interrotta, *distruggendo quindi tutti i cloni* del token eventualmente ancora attivi (su percorsi di elaborazione paralleli generati da nodi fork).

Qualora si volesse, invece, solamente terminare il flusso di esecuzione corrente, senza interrompere l'intera attività e senza interrompere gli altri flussi di esecuzione paralleli, si può utilizzare il nodo di terminazione di flusso (formalmente: *Flow Final Node*), rappresentato con un simbolo di croce inscritta in un cerchio (⊗).

Un esempio di utilizzo di tali simboli è riportato in figura 5.19, in cui esistono due flussi di esecuzione paralleli. Qualora venga eseguita l'azione B prima dell'azione C, allora il processo termina immediatamente, e l'azione C potrebbe non essere eseguita mai. Se invece termina prima l'azione C, allora il token relativo viene distrutto, ma il processo rimane attivo, ed il token che attiva l'azione B non viene distrutto: in questo caso, l'Azione B verrà sempre eseguita. Quindi le sequenze di azioni possibili sono: (A, B) ed (A, C, B).

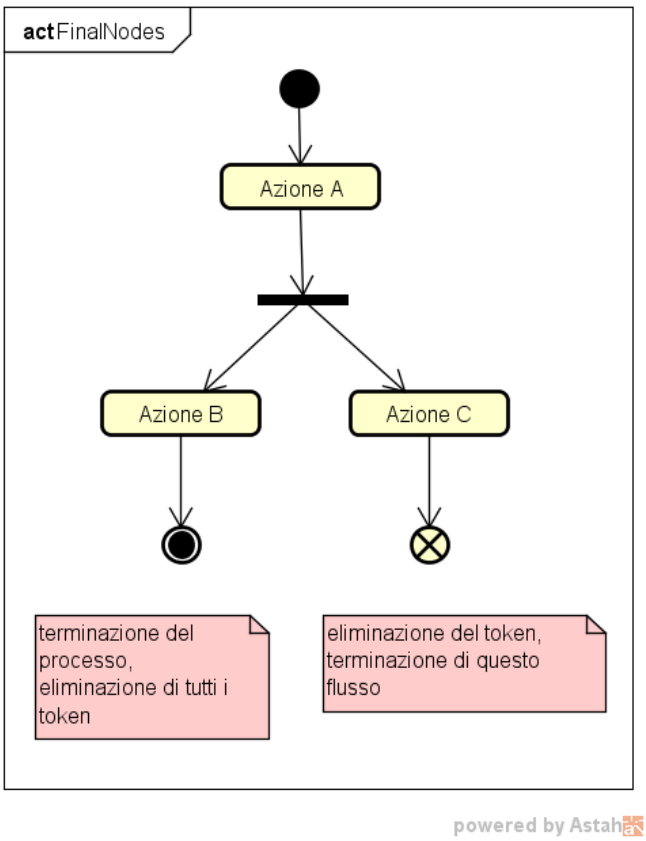

Figura 5.19: Activity Final Node e Flow Final Node

Nella sezione 5.3.3 si tratteranno i casi d'uso pratici di questi nodi di controllo.

5.3 Pattern di modellazione

In questa sezione vengono raccolti alcuni pattern di modellazione, che rappresentano soluzioni collaudate ad alcuni frequenti problemi di modellazione, che non sono rappresentabili in modo diretto né intuitivo con i costrutti base. Molti di questi pattern sono tratti, con notevoli semplificazioni, da [13].

5.3.1 Processi strutturati

Una delle principali regole di modellazione è quella di mirare, il più possibile, a realizzare diagrammi di attività di tipo *strutturato*. Questo è un requisito comune a tutto il mondo dell'informatica, e si applica ai flow chart, ai linguaggi di programmazione, alle pagine web, e così via: l'obiettivo è mantenere la complessità di un artefatto (modello, programma, diagramma, ...) ad un livello comprensibile dai progetti umani. Poiché la complessità dei prodotti informatici supera di gran lunga la capacità di comprensione umana, vi è la necessità di riuscire a descrivere il sistema a diversi livelli di astrazione, e soprattutto potendo nascondere alcuni dettagli se in un determinato momento questi sono irrilevanti. In particolare, ciò richiede che sia sempre possibile selezionare una determinata porzione di un processo, e sostituirla con una singola azione complessa (v. sezione 5.2.11), in modo da poter ignorare temporaneamente la sua implementazione.

Per poter definire un processo come strutturato, è necessario che siano rispettate alcune regole:

- ogni nodo azione (azione semplice o azione complessa) ha esattamente una freccia entrante ed esattamente una freccia uscente;
- esiste un solo nodo di inizio, con una sola freccia uscente;
- esiste un solo nodo di terminazione, con una sola freccia entrante;
- ogni nodo fork ha un nodo join corrispondente, in cui i flussi di esecuzione che si sono separati vengono tutti riuniti;
- ogni nodo choice ha un nodo merge corrispondente, che ricongiunge i flussi di elaborazione alternativi;
- quando vi siano più costrutti di tipo fork-join o choice-merge, ciascun costrutto è completamente annidato all'interno di uno dei rami di uno degli altri costrutti, o viceversa lo comprende completamente. In altre parole, non è possibile "entrare a metà" o "uscire prima" da uno di questi costrutti: si può solamente entrare dal nodo iniziare ed uscire da quello finale.

Nella modellazione, ci si atterrà il più possibile all'utilizzo di processi strutturati. Qualora, per esigenze particolare (si vedranno alcuni esempi tra breve) fosse necessario fare delle eccezioni, avremo cura di inserire la parte di processo non strutturata all'interno di un'azione complessa, in modo da limitare la complessità ad una porzione ben documentabile del processo, isolata dal resto.

5.3.2 Ciclo strutturato

All'interno di un processo è possibile creare un ciclo stutturato mediante l'uso di un nodo di choice e di un nodo di merge, nel quale il nodo di choice venga incontrato *dopo* quello di merge, ed almeno una delle uscite riporti indietro al nodo di merge stesso.

Nell' esempio di figura 5.20 il token rappresenta un generico ordine che è composto da più elementi. Ciascun elemento viene selezionato e prelevato dal magazziono, e queste due operazioni vengono ripetute finché vi sono ancora elementi. In tal caso, i due blocchi centrali (seleziona e preleva) vengono ripetuti più volte, finché una condizione (nessun elemento rimanente) diventa vera. L'intero processo è strutturato, un quanto il ciclo, nel suo complesso, è dotato di un solo punto di ingresso ed un solo punto di uscita, ed i nodi choice e merge sono correttamente accoppiati. Ovviamente, come in tutti i nodi di tipo choice, anche qui le informazioni in base alle quali il processo decide iterare od uscire dall'iterazione sono definite dall'espressione di guardia, che dovrà essere calcolabile sulla base delle informazioni presenti nel modello concettuale.

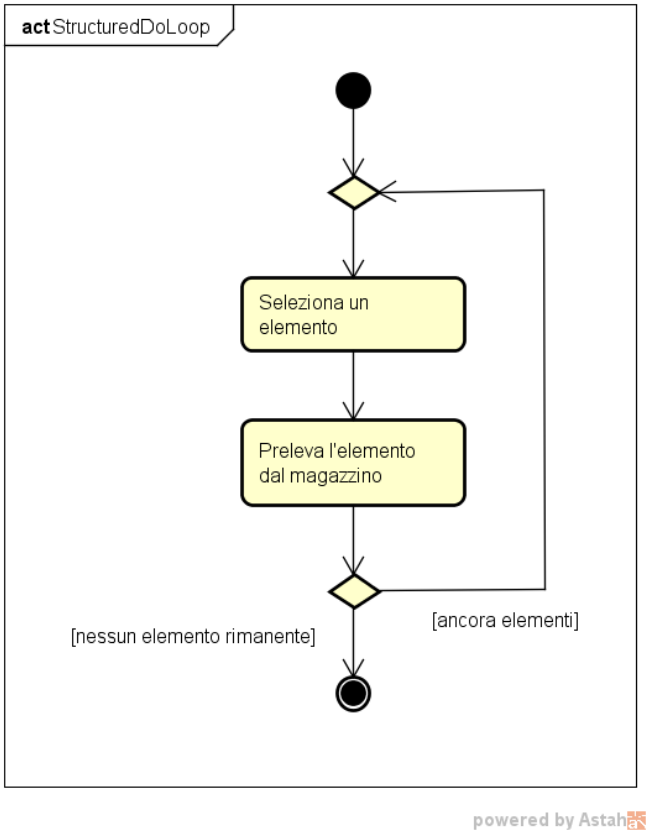

Figura 5.20: Esempio ciclo “Do-Repeat” strutturato

L’esempio di figura 5.20 rappresenta un ciclo di tipo “Do-Repeat” o “Do-While” (dal nome dei costrutti di alcuni linguaggi di programmazione), nel quale la condizione di ripetizione (il nodo choice) viene verificata al termine dell’iterazione (prima eseguo le azioni, poi mi chiedo se devo ripeterle). In alternativa, si può usare una rappresentazione di tipo “While”, come in figura 5.21, in cui il test viene svolto prima dell’esecuzione delle azioni. I due costrutti sono molto simili, e la principale differenza risiede nel fatto che le azioni vengono eseguite almeno una volta nel ciclo Do-Repeat, mentre potrebbero non essere mai eseguite nel ciclo While, qualora la condizione fosse già falsa in partenza.

I costrutti di tipo ciclico possono essere anche combinati in forme più complesse, come si vede in figura 5.22[1], ad esempio annidando due cicli uno dentro l’altro, oppure costruendo sequenze di cicli. Occorre tuttavia evitare di costruire dei cicli non strutturati, come quello rappresentato a destra nella figura.

5.3.3 Terminazione implicita ed esplicita

Come già visto nella sezione 5.2.12, esistono due nodi di controllo capaci di eliminare un token: l’Activity Final Node ed il Flow Final Node. L’utilizzo di questi nodi permette di specificare in due modi diversi la condizione di terminazione dell’attività.

In particolare, un processo può terminare per:

Terminazione esplicita il processo termina quando viene raggiunto un determinato stato, solitamente identificato dal nodo Activity Final, che sarà unico per tutto il processo (figura 5.23, sinistra). Quando il nodo finale viene raggiunto da un qualsiasi token, ogni eventuale attività in corso viene annullata, e l’attività nel suo complesso viene considerata come terminata

[1] immagine tratta da “Using UML, Patterns, and Java Object-Oriented Software Engineering” [4] in Chapter 11, “Testing.”. La pagina ufficiale del libro permette di scaricare delle risorse aggiuntive, tra cui le slide di lezione.

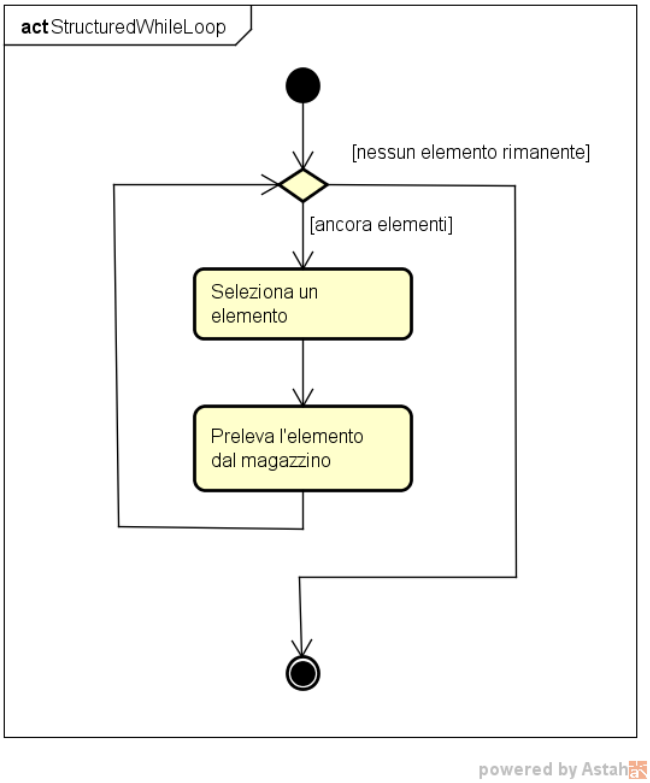

Figura 5.21: Esempio ciclo “While” strutturato

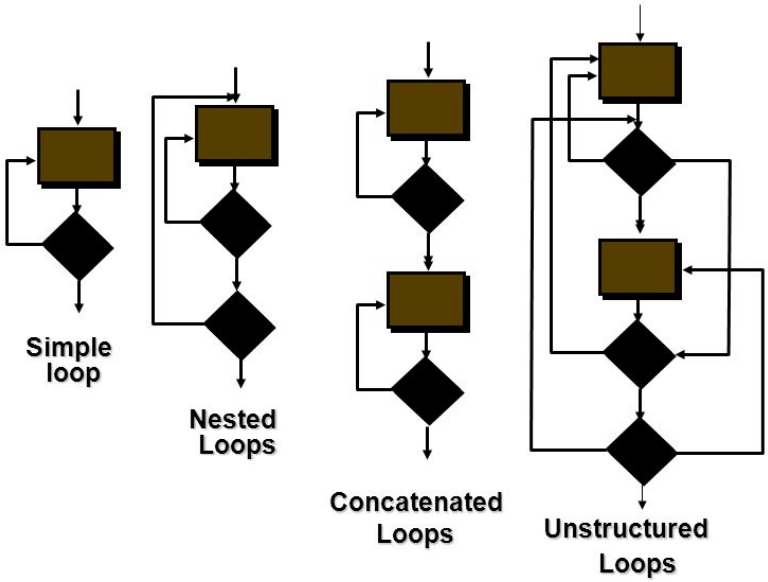

Figura 5.22: Esempio di cicli strutturati (semplici, annidati e concatenati) o non strutturati

con successo, anche se vi fossero ancora dei task in corso o in attesa di esecuzione. Questa è la modalità normale di terminazione, e prevede che tutti i flussi di esecuzione vengano fatti convergere (attraverso nodi join o nodi merge) sul nodo Activity Final. La terminazione è detta esplicita in quanto vi è un ben preciso nodo che identifica il termine dell'attività.

Terminazione implicita il processo termina quando tutte le azioni previste sono completate, per cui ciascun flusso parallelo di elaborazione deve giungere alla propria condizione di terminazione (figura 5.23, destra). Questo si può rappresentare utilizzando un nodo Flow Final al termine di ciascun flusso parallelo, in modo che quando tutti i token sono arrivati a destinazione (e solo allora) il processo può considerarsi terminato con successo. Se invece alcuni token, su alcuni flussi, sono rimasti indietro, allora il processo non si considera ancora terminato e si attende che giungano a conclusione anche questi. In questo caso la terminazione è detta implicita in quanto non esiste un punto unico di controllo in cui si evidenzia la fine delle attività, ma questa avviene implicitamente quando tutti i flussi di elaborazione (cioè i token attivi) si sono esauriti.

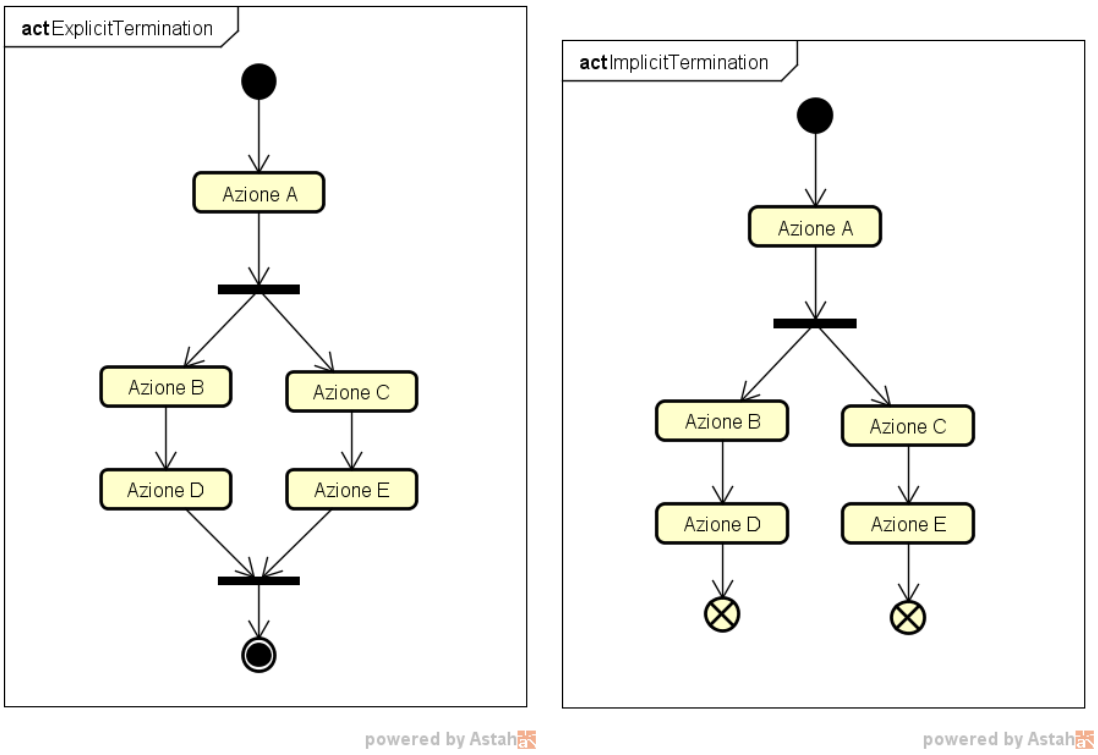

Figura 5.23: Esempio di terminazione esplicita (a sinistra) e terminazione implicita (a destra)

5.3.4 Discriminatore strutturato

Può sorgere il problema di dover scegliere fra due attività che si possono svolgere in alternativa, ma di cui non sia noto a priori, né sia possibile acquisire informazioni per determinare quale si svolgerà. In queste situazioni non è possibile usare un nodo choice (che richiederebbe di conoscere quale azione attivare prima di raggiungerla), ed occorrerà "abilitare" entrambe le alternative, per poi eseguire una sola delle due.

A tal fine, si può utilizzare un costrutto simile a quello riportato in figura 5.23: dopo l'esecuzione dell'Azione A, il token arriva in parallelo sia alla Alternativa 1 che alla Alternativa 2. A priori non sappiamo quale alternativa sarà eseguita, ma quando una delle due verrà eseguita, l'azione complessa giunge al proprio nodo di terminazione, e di conseguenza l'altra alternativa viene annullata. Dopodiché, il processo prosegue con l'Azione B e termina.

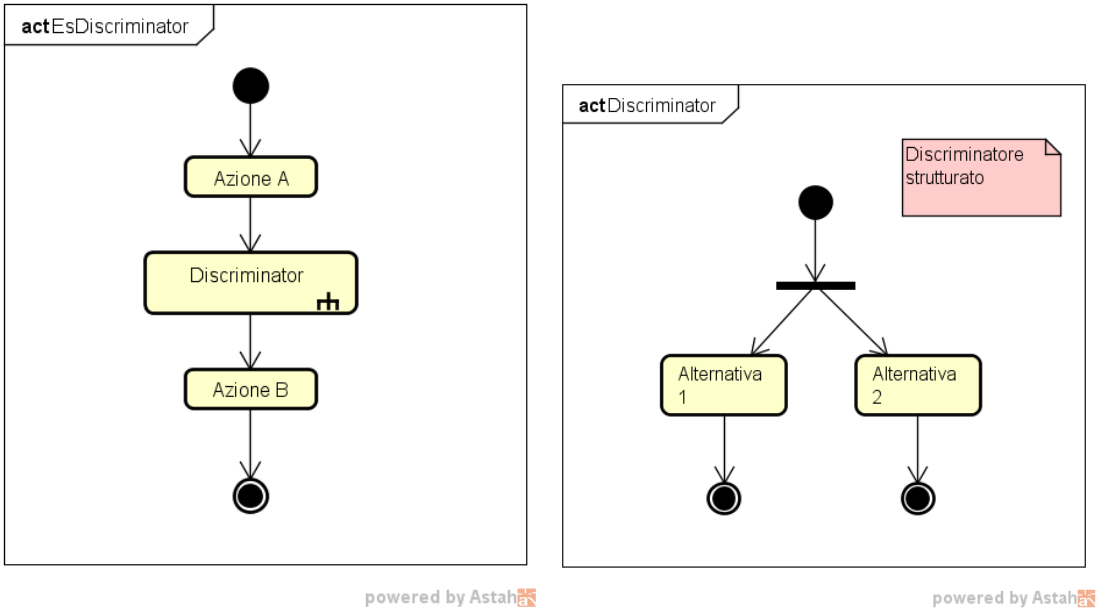

Figura 5.24: Discriminatore strutturato

Come si vede, la costruzione di un discriminatore richiede l'utilizzo di diagrammi non strutturati (un fork senza join, e due nodi di terminazione anziché uno). Anche per questo motivo, il discriminatore viene sempre implementato in un'attività complessa: in questo modo, il processo principale rimane strutturato.

Si noti che questo costrutto è diverso rispetto alla semplice scelta esclusiva (sezione 5.2.5, figura 5.7), in quanto la scelta viene effettuata, in tempo zero, dal sistema informativo sulla base delle informazioni disponibili. Nel discriminatore, invece, il sistema informativo non sa a priori quale alternativa verrà eseguita, e si dovrà predisporre per eseguirle entrambe. Una volta che una delle alternative verrà iniziata, allora l'altra sarà abbandonata. In un certo senso, la scelta viene fatta dall'utente che interagisce con il sistema, e non dal sistema informativo stesso.

5.3.5 Time-out

Un caso particolare di discriminatore è quello in cui una delle due alternative è semplicemente rappresentato dal passare del tempo. Il tal caso, si utilizzerà un'azione di attesa (simbolo della clessidra): se l'azione verrà compiuta prima dello scadere del tempo, allora il processo potrà procedere e l'attesa sarà annullata. Se invece l'azione non viene compiuta, allo scadere del tempo il processo annullerà l'azione (che non è avvenuta in tempo) e proseguirà.

Vediamo un esempio (figura 5.25): immaginiamo di voler prenotare un biglietto su un sito di vendita online. Prenoto il biglietto, il SI invia la richiesta alla compagnia aerea la quale, con i suoi tempi e secondo le sue regole confermerà la prenotazione. Questo passaggio nella maggior parte dei casi è immediato, mentre altre volte invece può accadere che il SI non invii conferme istantanee ma ci metta più tempo. Gli esiti che possiamo ricevere sono quello in cui il SI manda la richiesta, la compagnia lo elabora e conferma la prenotazione, dopodiché possiamo effettivamente pagare il biglietto e il processo è concluso. Normalmente per concludere questo processo ho 48 ore di tempo. Se entro questo termine non mi arriva conferma dalla compagnia aerea la prenotazione viene cancellata. L'altra possibilità è quella di ricevere la conferma ma di non portare a termine l'acquisto del biglietto entro le 48 ore prestabilite, quindi non pago. In entrambi i casi sia che io abbia cancellato la prenotazione sia che io abbia acquistato il biglietto il processo termina.

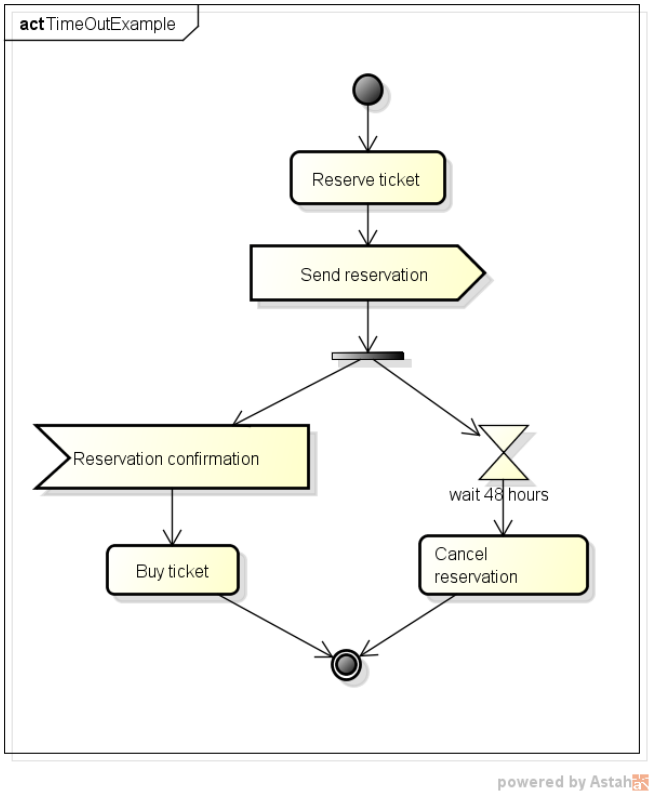

Figura 5.25: Esempio di time-out

5.3.6 Scelta differita

Un'altra variante dello schema del discriminatore (figura 5.26) è quello in cui si deve scegliere sulla base di due alternative, in funzione di un evento esterno (ad esempio un segnale). In questo caso, la ricezione del segnale deve avvenire in un'azione complessa che implementa il pattern del discriminatore (per poter ricevere l'uno o l'altro segnale). Invece, le attività da compiere in conseguenza del segnale ricevute possono essere modellate nel processo principale: in questo caso si può usare un normale nodo di scelta, che ovviamente agirà sulla base delle informazioni raccolte dal discriminatore.

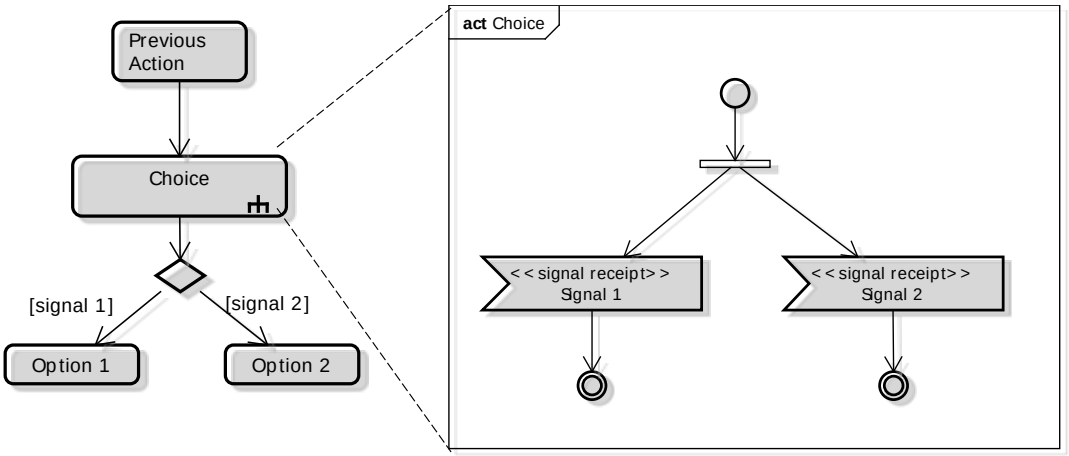

Figura 5.26: Esempio di scelta differita

5.3.7 Segnale o Attesa?

Un dubbio che spesso si incontra, quando occorre modellare il fatto che il processo deve rimanere in attesa (senza che venga eseguita alcuna azione), è se si debba utilizzare il nodo di attesa oppure un nodo di ricezione segnale.

Esiste una semplice regola per evitare errori: mettersi dal punto di vista del token, e chiedersi se, nel momento in cui il token arriva al punto in cui deve fermarsi in attesa, vi siano le condizioni (e le informazioni disponibili) per poter determinare ore, minuti e secondi dell'istante di tempo in cui il token sarà liberato. Se sì, allora sarà corretto utilizzare un nodo di tipo attesa (clessidra).

In caso contrario, si è dimostrato che la condizione di uscita non dipende esclusivamente dal passare del tempo, ma anche da altri fattori. Tali fattori sono sicuramente di tipo esterno (perché il processo corrente è "fermo"), e quindi potranno essere comunicati attraverso l'invio di un segnale. Pertanto, se l'attesa implica la ricezione di informazioni dall'esterno (da altri processi paralleli, o da altri sistemi informativi), è corretto utilizzare un nodo di ricezione segnale.

5.3.8 Notifica all'utente

Le comunicazioni tra utente e sistema, o meglio le azioni che richiedono la collaborazione tra utente e sistema, sono di norma realizzate con dei nodi di tipo azione. Il fatto che un utente sia responsabile di un'azione e che tale azione divenga abilitata (riceva il token), significa che *da quel momento in avanti* l'utente può compiere l'azione stessa: possono passare poche frazioni di secondo, minuti, ore o anche giorni prima che l'utente voglia (o possa) eseguire l'azione. In generale, si assume che l'utente sia parte attiva nel sistema, e sappia esattamente (perché, ad esempio, è collegato all'interfaccia) quali azioni può (e deve) compiere.

Esistono però dei casi in cui la risposta del sistema può essere anche molto lenta, oppure dipendere da fattori esterni, per cui non di può presupporre che l'utente sia parte attiva nel controllare la azioni da svolgere. In tali casi, è opportuno modellare un'azione di *notifica* nei confronti dell'utente, ad esempio inviando una mail, un SMS, un instant message o una notifica vera e propria su un dispositivo mobile. In questo caso, il SI si serve di un altro dispositivo (un altro SI), il quale a sua volta avrà la capacità di stimolare l'utente.

L'invio dei segnali si può utilizzare quindi per modellare l'azione di *notifica* verso l'utente. In questo caso, si potrebbe erroneamente pensare che stiamo inviando il segnale all'utente. In realtà il segnale viene inviato al sistema informativo (es: gmail per la posta, oppure una App installata per la notifica), il quale la presenta all'utente (sotto forma di nodo azione).

5.3.9 Accorpamento di azioni

Un dubbio che può sorgere in fase di modellazione è relativo alle azioni semplici concatenate in sequenza: supponendo di avere un certo numero di nodi azione, collegati in cascata mediante una seplice sequenza, è più corretto lasciarli come nodi separati, oppure conviene creare un solo nodo azione, che riassuma in sé tutte le azioni?

In generale, conviene evitare di entrare troppo nel dettaglio sullo svolgimento delle operazioni, per cui si dovrebbe puntare ad avere un diagramma più semplice e con un numero minore di nodi.

Tuttavia, esistono alcune condizioni che possono impedire la fusione di più azioni in sequenza in una unica azione. Le più importanti sono:

- se due azioni consecutive sono svolte da due *attori diversi* (diverse responsabilità, diverse swimlane), allora **non** possono essere unite;
- se un'azione non la impatto sull'andamento globale del processo (non lascia traccia nel sistema informativo, oppure non influenza in alcun modo le scelte che verrano fatte dai successivi nodi choice), allora è una buona candidata ad essere accorpata con l'azione successiva;
- se l'esecuzione di due azioni consecutive è di norma contestuale (nell'ambito di una singola sessione di interazione, in un arco di tempo limitato, ad esempio dell'ordine della decina di minuti), allora potrebbero essere accorpate. Se invece tra il termine della prima azione e l'inizio della seconda dovessero passare ore o giorni, allora è necessario mantenere separate le azioni.

5.4 Riepilogo

I diagrammi di attività sono una delle tecniche utilizzate per descrivere il flusso di un processo. L'obiettivo è di catturare 3 informazioni fondamentali:

1. Chi è coinvolto (responsabilità)
2. Cosa fa, quali attività svolge (azioni)
3. Come le attività si succedono una con l'altra (nodi di controllo)

Le responsabilità sono indicate come soggetto nel testo dell'azione, oppure sono definite dalle swimlanes (partizioni).

È importantissimo sapere, quando si inizia un activity diagram, **a chi/a che cosa** è associato un token. Questo ci permette di assumere una prospettiva chiara e univoca delle attività svolte.

Le attività sono solo quelle azioni che in un modo o nell'altro lasciano traccia nel SI.

5.5 Esempio di modellazione di processo

Si consideri il seguente scenario[2]:

> *In un sistema di trasporti urbani vengono utilizzati dei biglietti Chip-on-Paper che utilizzano la tecnologia RFID per effettuare la convalida. In tali biglietti, un piccolo chip è integrato nel biglietto cartaceo, e può essere letto appoggiandolo agli opportuni lettori installati a bordo dei mezzi.*
>
> *I biglietti contengono un numero predefinito di corse (5, 10 o 15) e possono essere convalidati a bordo degli autobus. Ad ogni convalida il numero di viaggi residui del biglietto viene decrementato ed il sistema conosce in ogni istante lo stato dei biglietti.*
>
> *Ad ogni convalida di un biglietto, il sistema informativo registra anche le informazione relative: al dispositivo di validazione, all'autobus su cui è installato, all'ora di partenza, alla fermata di partenza, ed alla linea percorsa dall'autobus.*
>
> *Il ciclo di vita dei biglietti prevede che essi vengano emessi dall'azienda di trasporti, successivamente i rivenditori (edicole, tabaccai, etc.) li attivano prima di venderli al pubblico caricando il numero di corse previsto. Gli utenti, in possesso di un biglietto attivato, possono convalidarlo un numero di volte pari alle corse concesse.*
>
> *A fine mese, ogni venditore conferma l'elenco dei biglietti venduti (ovvero attivati) e lo invia all'azienda di trasporti urbana che emette, dopo aver fatto una verifica interna ed eventualmente richiesto una rettifica, una richiesta di pagamento per il venditore (il quale dovrà versare l'importo dei biglietti venduti meno una provvigione fissa). La procedura si conclude quando il sistema riceve dalla banca l'informazione che il pagamento richiesto è stato effettuato.*

5.5.1 Modello concettuale

L'analisi del testo[3] ci porta ad identificare i due concetti principali:

Rivenditore rappresenta le informazioni sui rivenditori che attivano i biglietti e li vendono ai clienti, e gestiscono i pagamenti nei confronti dell'azienda di trasporti;

Biglietto rappresenta un singolo biglietto emesso, che verrà attivato e subirà una serie di convalide.

Un concetto che, invece, **non** è presente nel modello è quello di **Utente**, in quanto il sistema informativo non ha alcuna informazione su di esso (a parte il fatto che possiede un biglietto, le cui informazioni sono già nella classe Biglietto, l'utente è del tutto anonimo). Pertanto le informazioni sull'utente non sono rappresentate nel modello concettuale.

[2] tratto dalla prova scritta d'esame del 2/09/2015

[3] questo capitolo è dedicato soprattutto ai modelli di processo, per cui l'illustrazione del modello concettuale sarà trattata in maniera abbastanza rapida. Il modello concettuale è comunque indispensabile per comprendere il funzionamento del processo, pertanto è stato qui riportato

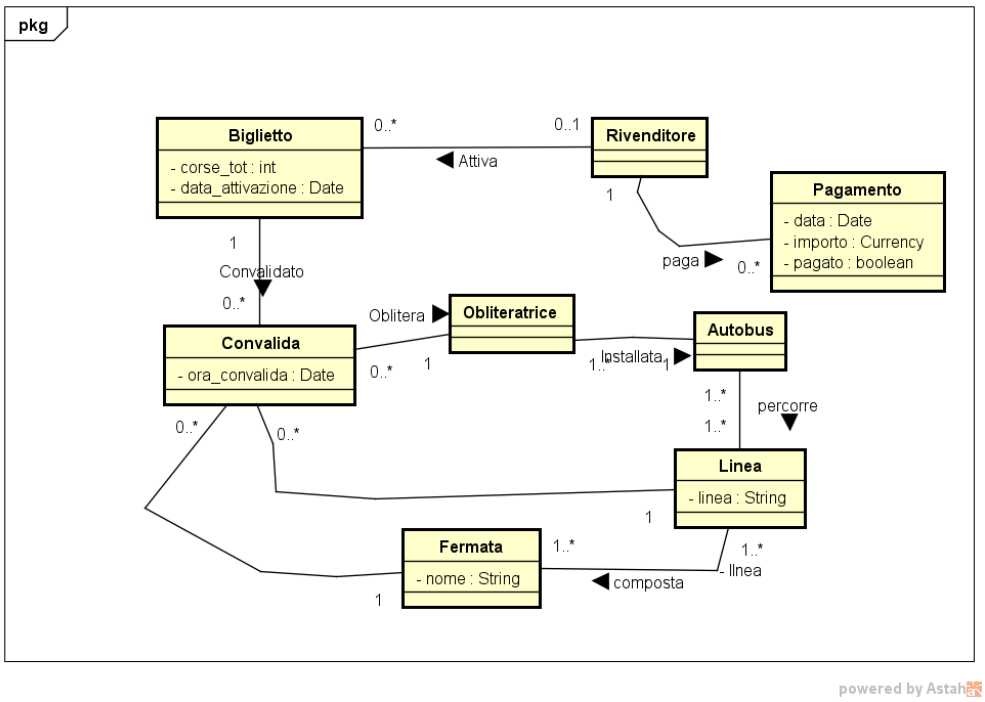

Figura 5.27: Diagramma delle classi

Per quanto riguarda i pagamenti a fine mese, si può definire il concetto **Pagamento**, ovviamente legato a Rivenditore, che permetta di rappresentare l'importo e l'informazione sullo stato di avanzamento del pagamento (l'attributo 'pagato' di tipo booleano).

Più complessa è invece la situazione legata alla convalida dei biglietti a bordo mezzo. A tal fine, si possono utilizzare i seguenti concetti:

Convalida che rappresenta una singola convalida di un determinato biglietto. Questa classe contiene una serie di relazioni 1-*N* che rappresentano le informazioni raccolte durante l'azione di convalida;

Obliteratrice indica quale sia la specifica "macchinetta" in cui è stato convalidato il biglietto;

Autobus indica su quale mezzo è stata installata l'Obliteratrice (non è necessaria una relazione tra Convalida ed Autobus, in quanto la conoscenza della Obliteratrice permette di determinare univocamente l'Autobus coinvolto);

Linea rappresenta la linea percorsa dall'Autobus al momento della Convalida (si noti che un autobus può percorrere più linee, pertanto la conoscenza dell'Autobus non è sufficiente a determinare la linea, ed è quindi necessaria una relazione diretta tra Convalida e Linea);

Fermata rappresenta la fermata nella quale è stato convalidato il Biglietto.

Il concetto Convalida è dunque un pundo di "snodo" (collegato a mo' di centro-stella) rispetto a tutte le informazioni collaterali.

Un'informazione importante, la cui modellazione potrebbe non essere immediata, è il numero di *corse residue* su un determinato biglietto. Ragionandoci, possiamo notare che è sempre valida la seguente relazione:

$$\text{Corse totali} = \text{Corse residue} + \text{Numero di convalide}$$

Per evitare ridondanza, di queste 3 quantità ne dovremo modellare solamente due, poiché la terza si può ricavare dalle due scelte. L'unico valore costante (che non cambia con le varie convalide) è il numero di corse totali, e quindi conviene modellarlo, direttamente come attributo di Biglietto. Per

quanto riguarda gli altri due valori, conviene notare che il numero di convalide si può già ottenere, semplicemente contanto il numero di volte in cui il biglietto partecipa in una relazione Convalidato con la classe Convalida. Quindi questo dato è già disponibile, e non occorre rappresentarlo esplicitamente. Di conseguenza, il numero di corse residue non dovrà essere rappresentato.

Il diagramma delle classi corrispondente a questo modello concettuale è rappresentato in figura 5.27.

5.5.2 Processo per il ciclo di vita del biglietto

Una parte delle attività del sistema informativo deve seguire il ciclo di vita del biglietto: dalla sua emissione, all'attivazione, ad una serie di convalide, fino all'esaurimento delle corse disponibili.

Questo processo, decisamente semplice, può essere modellato come rappresentato nel diagramma di attività di figura 5.28. In tale diagramma, il *Token* rappresenta un singolo biglietto (quindi un'istanza della classe Biglietto) dal momento della sua emissione al momento del suo esaurimento.

Il processo è di natura prettamente sequenziale, con un ciclo strutturato che permette di ripetere l'operazione di convalida un numero di volte pari alle corse totali attivate. La condizione che governa la ripetizione del ciclo strutturato è:

$$\sum \text{Convalida} = \text{corse\_tot}$$

utilizzando esattamente le scelte fatte nel modello concettuale, per cui il numero di corse totali è rappresentato esplicitamente, mentre il numero di convalide effettuate viene calcolato contando il numero di partecipazioni alla relazione tra Biglietto e Convalida.

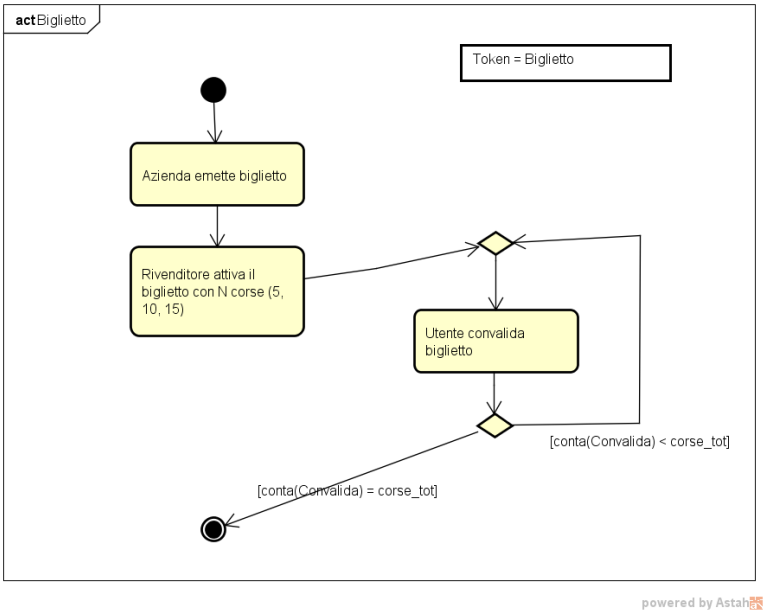

Figura 5.28: Diagramma di attività per il ciclo di vita del biglietto

In questo processo si può notare che, pur essendo menzionato nel testo, non vi è alcuna azione che faccia riferimento all'operazione di vendita. Non è stata modellata alcuna azione "Rivenditore vende biglietto", in quanto tale azione non lascia traccia nel sistema informativo del sistema ChipOnPaper. Sicuramente la vendita lascerà traccia nel registratore di cassa, ma a meno che questo non sia collegato al sistema di bigliettazione, quest'ultimo non potrà mai sapere se il biglietto è stato venduto, oppure regalato, od attivato in anticipo per poter essere venduto successivamente, magari in un momento di calca.

5.5.3 Processo per la gestione dei pagamenti

Mentre il processo precedente (Biglietto) tratta il ciclo di vita di singoli biglietti, occorre modellare anche le operazioni che si svolgono al momento del pagamento, ossia ad ogni fine mese. Queste operazioni vanno necessariamente descritte in un processo diverso, in quanto la natura del token è del tutto diversa: in questo caso il token rappresenterà una singola operazione di pagamento, ed il processo verrà attivato una volta per ogni riventitore, ad ogni fine mese. Il fatto che il numero di attivazioni (token) di questo processo (numero rivenditori per 12 mesi) sia completamente diverso rispetto al numero di attivazioni del processo precedente (pari al numero di biglietti attivati nell'anno) è ulteriore conferma del fatto che i processi devono essere separati.

Il pagamento si può modellare come illustrato in figura 5.29.

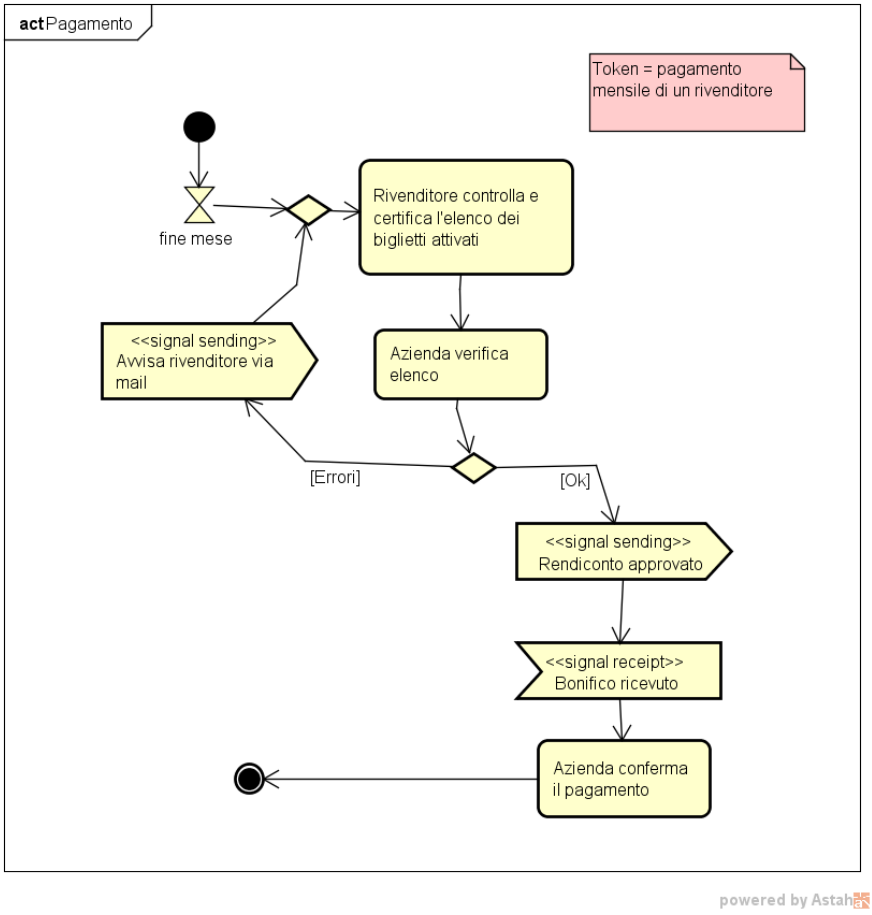

Figura 5.29: Diagramma di attività per il processo di pagamento

Innanzitutto il processo può essere eseguito solo una volta al mese, per cui il primo nodo sarà un nodo di attesa[4].

La fase successiva è rappresentata come un ciclo, nel quale il rivenditore controlal e certifica l'evento e l'azienda verifica l'elenco inviato: qualora vi siano errori, il ciclo viene ripetuto. Nel caso di rilevamento di errori, il sistema invia anche un messaggio di notifica al rivenditore: questo è necessario in quanto il rivenditore aveva confermato l'invio del proprio elenco, e quindi si aspettava l'approvazione del rendiconto. Invece il rivenditore deve essere informato che deve rivedere e controllare il proprio elenco. In questo caso il segnale rappresenta probabilmente un messaggio di e-mail. Non è da considerare obbligatorio, ma piuttosto una cortesia nei confronti del rivenditore (il quale, altri-

[4] in questi casi, alcuni autori preferiscono *non* usare il nodo start, ma iniziare il processo partendo direttamente con il nodo di attesa

menti, dovrebbe periodicamente consultare il sistema informativo per sapere se il proprio rendiconto contiene errori).

Quando il rendiconto (al primo tentativo oppure a seguito di una o più operazioni di revisione) viene approvato, allora il rivenditore viene informato attraverso un messaggio ("Rendiconto approvato"). A questo punto il rivenditore procederà al pagamento, ma questa azione non è rappresentata nel processo in quanto avviene sul sistema informativo della banca (attraverso il sistema di home banking oppure attraverso un'operazione a sportello), e non sul sistema ChipOnPaper (che non ha modo di sapere se e quando il bonifico sia stato fatto). Il processo rimane quindi in attesa di un segnale "Bonifico ricevuto": in questo caso stiamo assumendo che il sistema informativo della banca dell'azienda di trasporti (che può ovviamente essere diversa da quella del rivenditore) invii un segnale ad ogni bonifico ricevuto, associandolo anche al rivenditore che ha emesso tale bonifico. Un operatore dovrà quindi verificare i dati ricevuti (ad esempio, che l'importo effettivamente pagato sia quello inizialmente pattuito) e marcare il pagamento come confermato (attributo 'pagato' della classe Pagamento).

Lo stesso processo, che contiene esattamente le stesse azioni con gli stessi vincoli, si può anche rappresentare utilizzando la notazione delle swimlane. In questo caso vi sono due attori responsabili: il rivenditore e l'azienda dei trasporti, come illustrato in figura 5.30. La descrizione delle azioni può essere semplificata, in quanto il soggetto di ciascuna azione è implicitamente determinato dalla corsia in cui si trova. Ricordiamo che gli unici nodi per cui abbia senso verificare la corsia sono i nodi di tipo azione. Per quando riguarda i nodi di start, stop, signal sending, signal receipt, attesa, choice e merge, fork e join, la loro collocazione è assolutamente irrilevante, in quanto rappresentano operazioni che non coinvolgono direttamente l'utente.

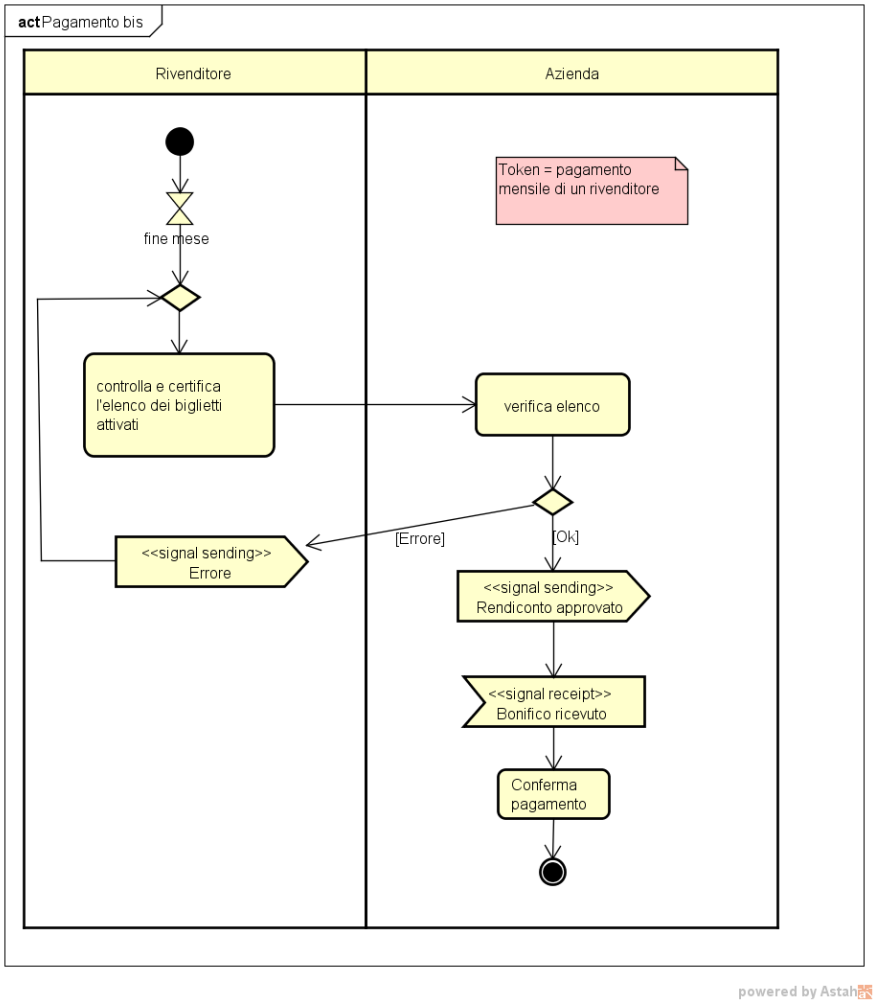

Figura 5.30: Diagramma di attività per il processo di pagamento, realizzato con swimlane

BIBLIOGRAFIA

[1] R. N. Anthony, *Planning and control systems: a framework for analysis*. Division of Research, Harvard Business School, 1965.

[2] N. Bolloju and F. Leung, "Assisting novice analysts in developing quality conceptual models with uml," *Communications of the ACM*, vol. 49, no. 7, p. 108, 112 2006.

[3] Bracchi, Francalanci, and Motta, *Sistemi informativi d'impresa*. McGraw Hill, 2010.

[4] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering Using UML, Patterns, and Java (3rd edition)*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. [Online]. Available: http://www.pearsonhighered.com/educator/product/Object-Oriented-Software-Engineering-Using-UML-Patterns-and-Java-3E/9780136061250.page

[5] P. Chen, "The entity-relationship model: toward a unified view of data," *ACM Transactions on Database Systems*, vol. 1, pp. 9–36, 1976.

[6] K. Fakhroutdinov, "Uml diagrams." [Online]. Available: http://www.uml-diagrams.org/

[7] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd edition*. Addison-Wesley Professional, 2003.

[8] ——, *UML Distilled: Guida rapida al lingugaggio di modellazione standard, 4a edizione*. Addison-Wesley, 2010.

[9] Laudon and Laudon, *Management dei Sistemi Informativi*. Prentice Hall, 2010.

[10] O. Lindland, G. Sindre, and A. Solvberg, "Understanding quality in conceptual modeling," *IEEE Software*, vol. 11, no. 2, pp. 42–49, 1994.

[11] OMG, *Business Process Modeling Notation (BPMN) Version 1.0. OMG Final Adopted Specification*. Object Management Group, 2006.

[12] OMG, *OMG Unified Modeling Language (OMG UML) Version 2.5*. Object Management Group, 2015. [Online]. Available: http://www.omg.org/spec/UML/2.5/

[13] N. Russell, A. H. M. T. Hofstede, and N. Mulyar, "Workflow controlflow patterns: A revised view," BPM Center, Tech. Rep., 2006. [Online]. Available: http://www.workflowpatterns.com/documentation/documents/BPM-06-22.pdf

[14] N. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, and P. Wohed, "On the suitability of uml 2.0 activity diagrams for business process modelling," in *Proceedings of the 3rd Asia-Pacific Conference on Conceptual Modelling - Volume 53*, ser. APCCM '06. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 95–104. [Online]. Available: http://www.workflowpatterns.com/documentation/documents/UMLEvalAPCCM.pdf

[15] S. S. Stevens, "On the theory of scales of measurement," *Science*, vol. 103, no. 2684, pp. 677–680, June 1946.

LICENZA E COLOPHON

Questo volume è stato redatto con il sistema di composizione LaTeX[5] utilizzando il modello di stile `memoir`[6].

Il contenuto del testo è rilasciato con la licenza Creative Commons Attribuzione - Non commerciale - Condividi allo stesso modo 2.5 Italia (CC BY-NC-SA 2.5)[7].

[5] `http://www.latex-project.org/`
[6] `http://www.ctan.org/tex-archive/macros/latex/contrib/memoir/`
[7] `http://creativecommons.org/licenses/by-nc-sa/2.5/it/`