Approfondimenti su jES

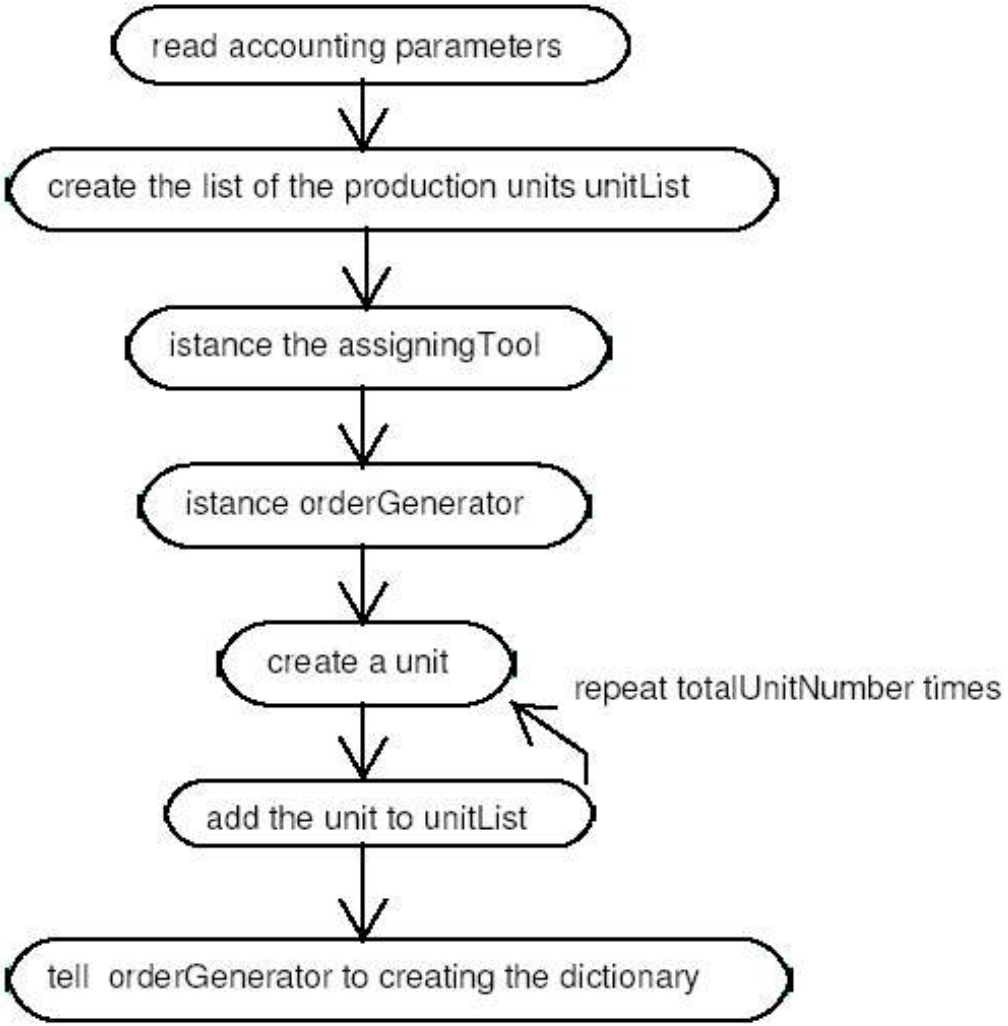
Università degli Studi di Bologna
Facoltà di Scienze MM. FF. NN.
Corso di Laurea in Scienze di Internet
Anno Accademico 2004-2005

Laboratorio di Sistemi e Processi Organizzativi

Come funziona il modello

buildObjects() Activity diagram

Diagramma che descrive il metodo *buildObjects()* in **jESLet**

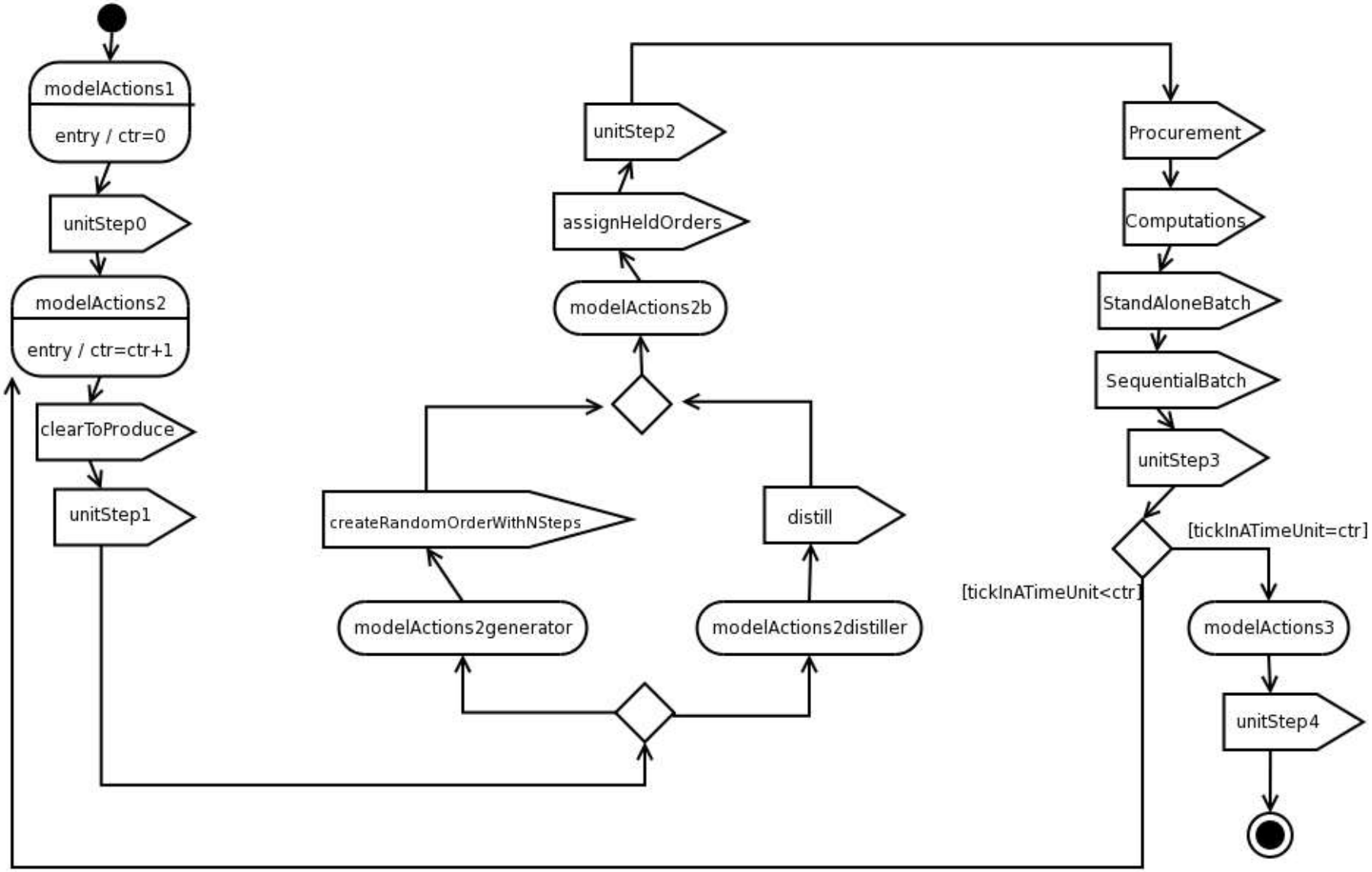


Metodo buildActions()

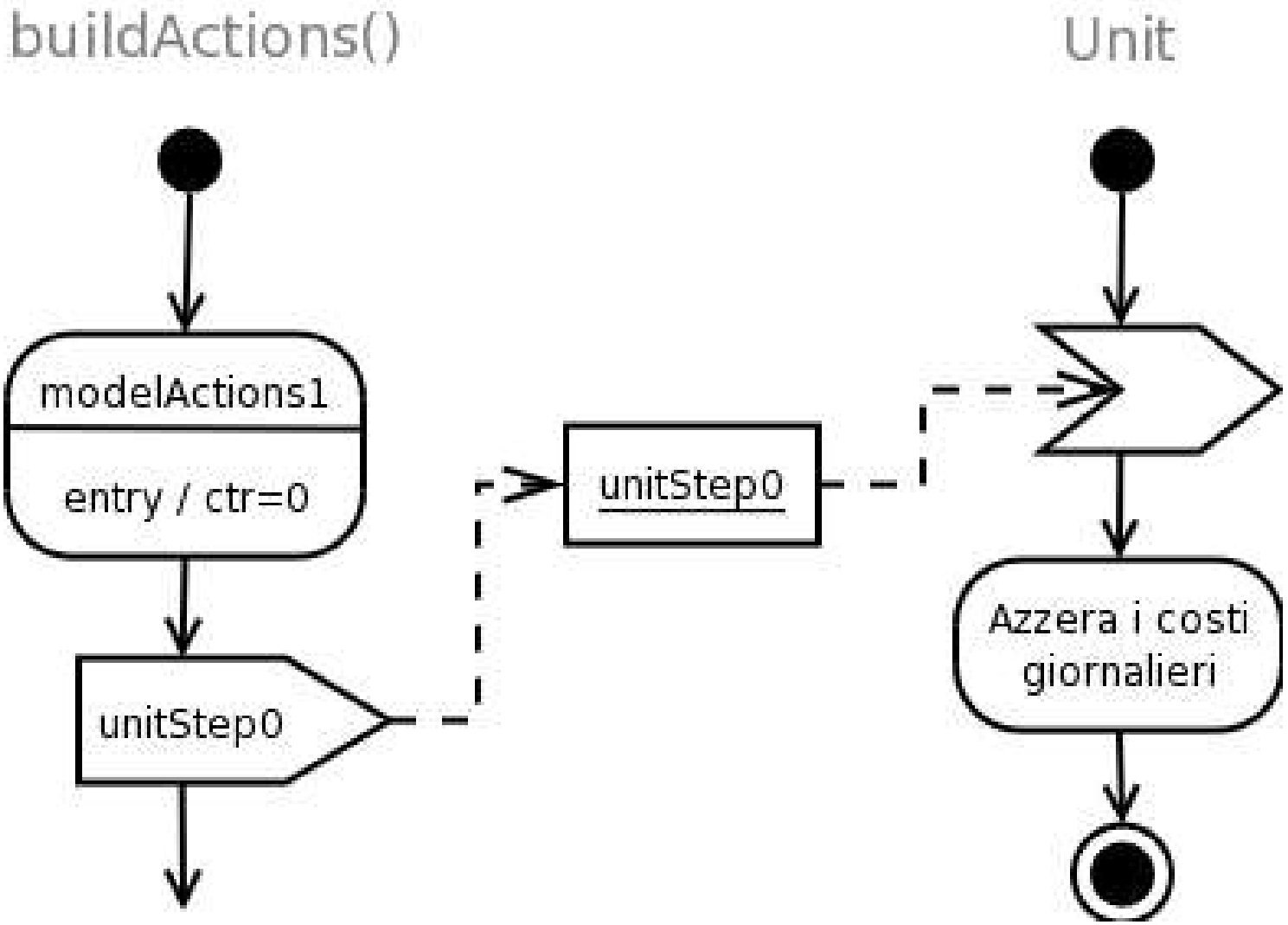
- La simulazione nasce grazie al metodo **buildActions()**
- Questo invia messaggi alle varie *entità* della simulazione grazie agli oggetti *ActionGroup: modelActions*
- L'ordine cronologico con cui vengono spediti i messaggi segue il seguente schedule, ove ***tickInATimeUnit***=*n*:

  (0, modelActions1)
  (0, modelActions2)
  (0, modelActions2generator *oppure* modelActions2distiller)
  (0, modelActions2b)
  ...
  (n-1, modelActions2)
  (n-1, modelActions2generator *oppure* modelActions2distiller)
  (n-1, modelActions2b)
  (n-1, modelActions3)

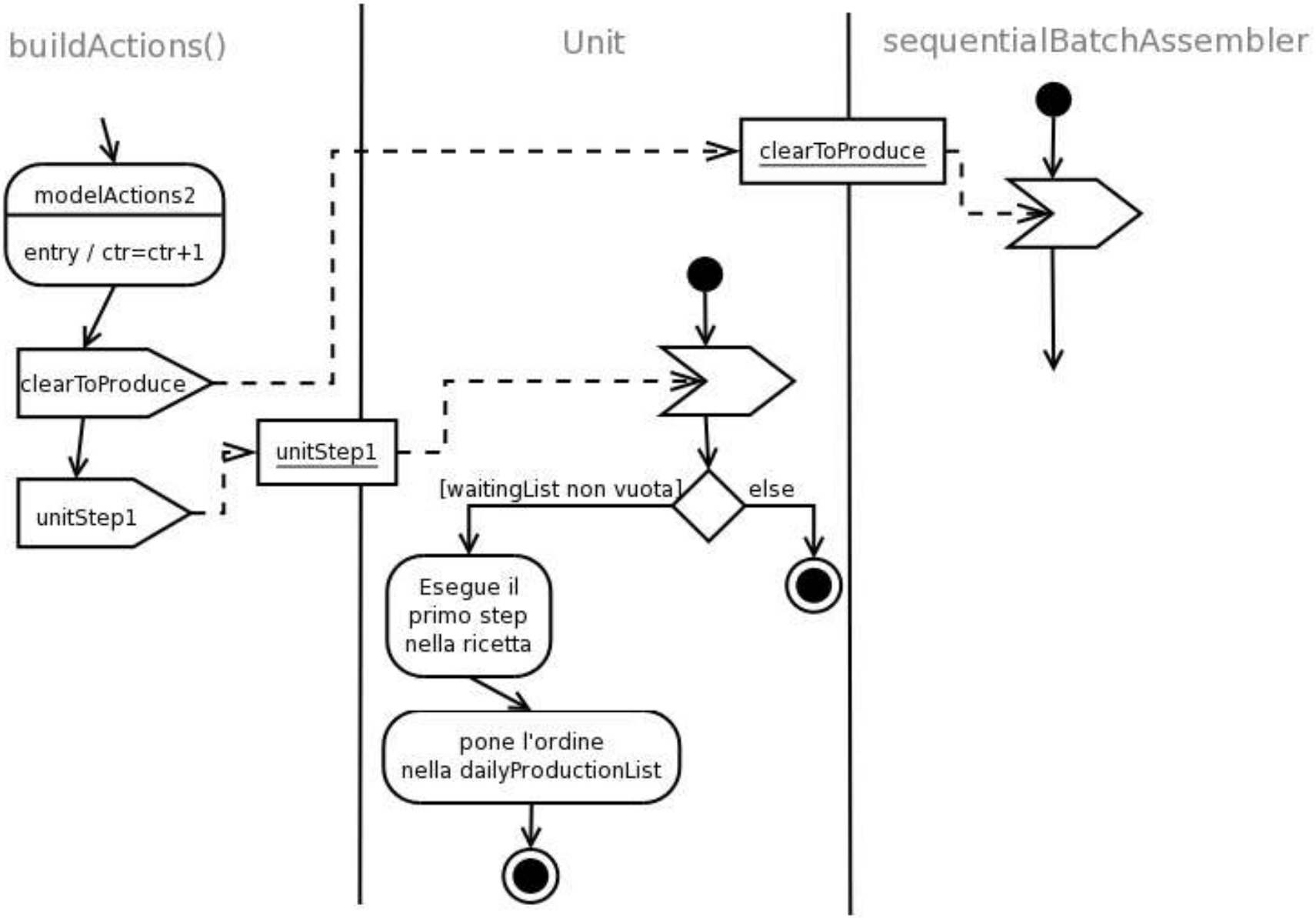
buildActions() Action Diagram



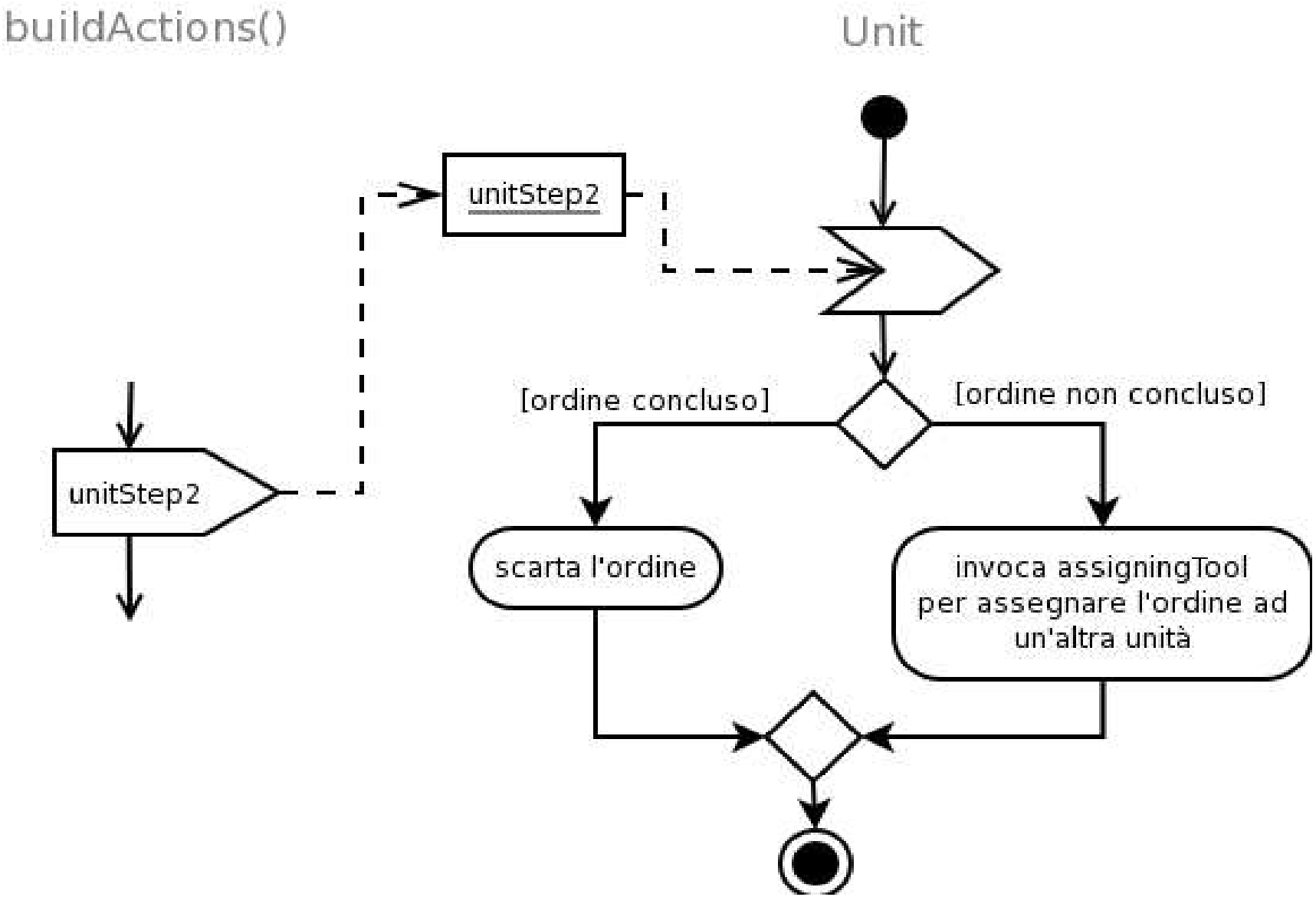
Action Diagram particolare unitStep0



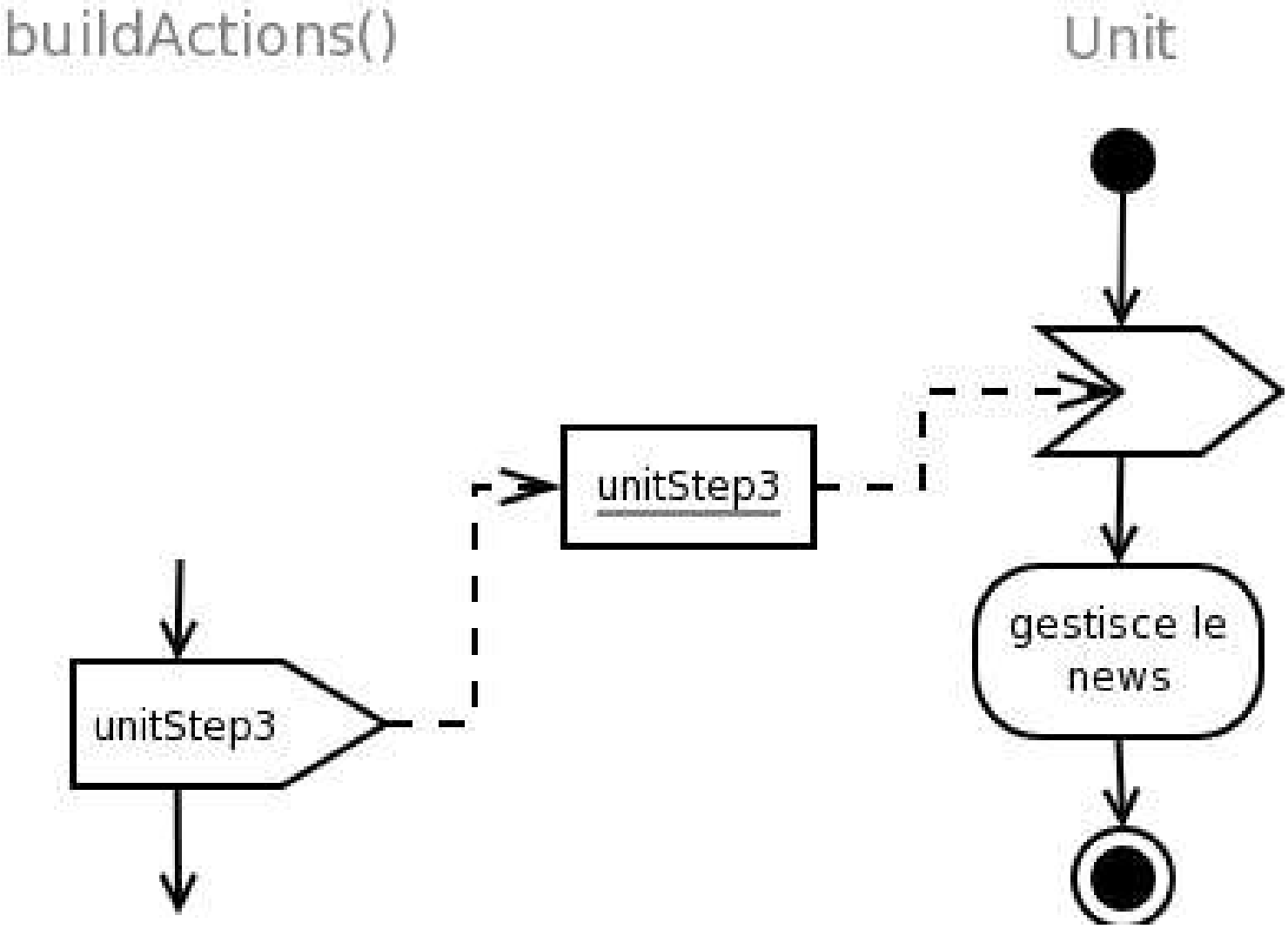
Action Diagram particolare unitStep1



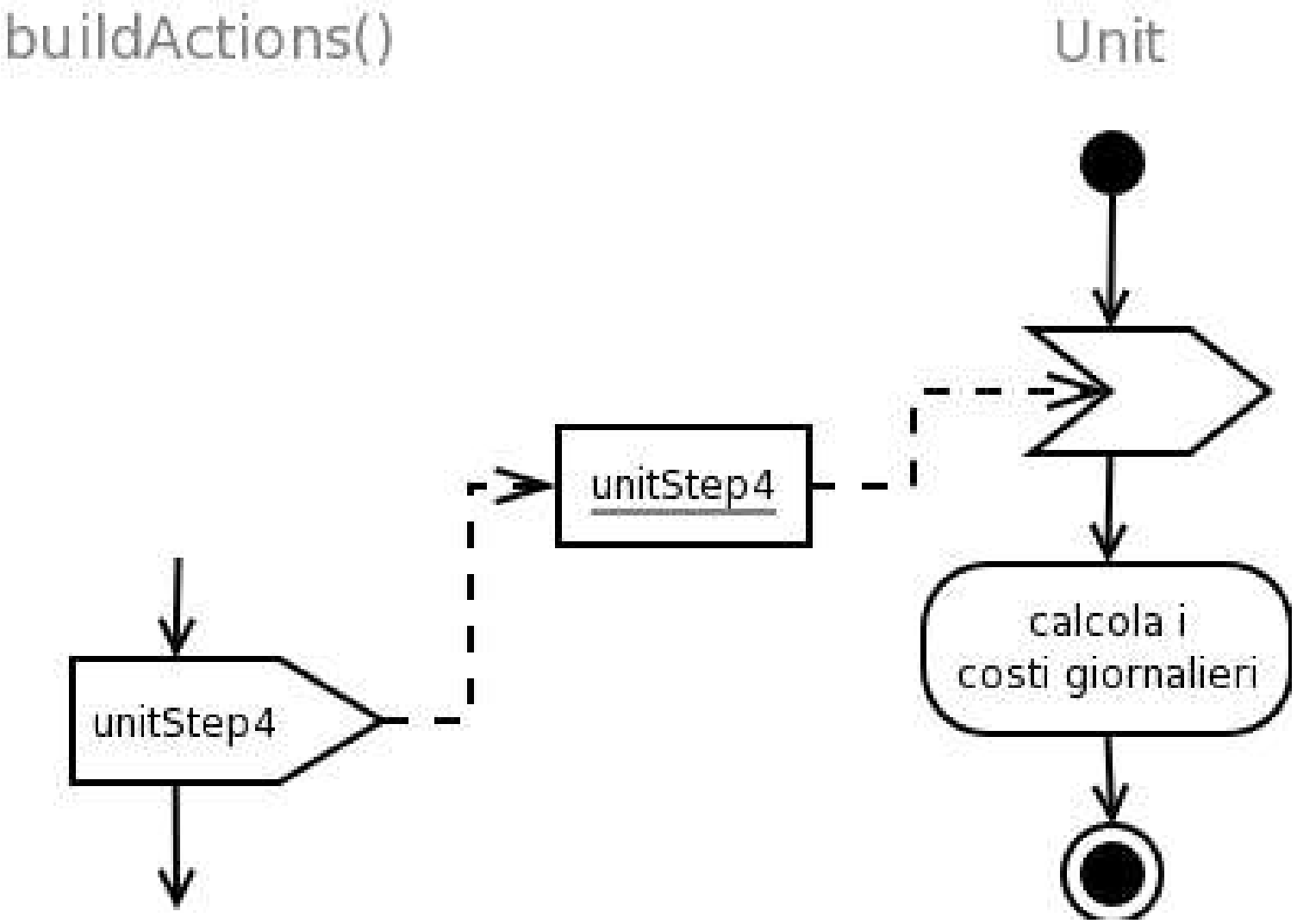
Action Diagram particolare unitStep2



Action Diagram particolare unitStep3

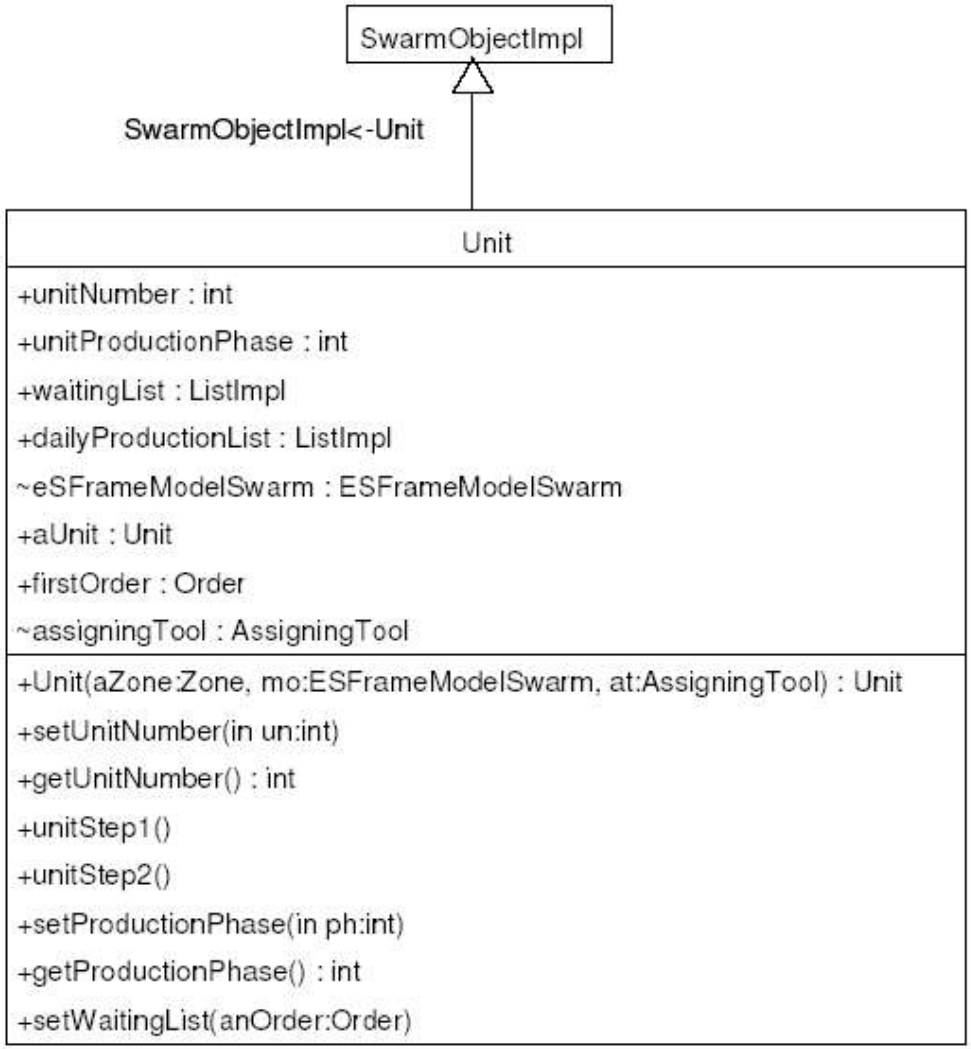


Action Diagram particolare unitStep4



Chi fa cosa?
Affrontiamo l'analisi di jES dal lato DW

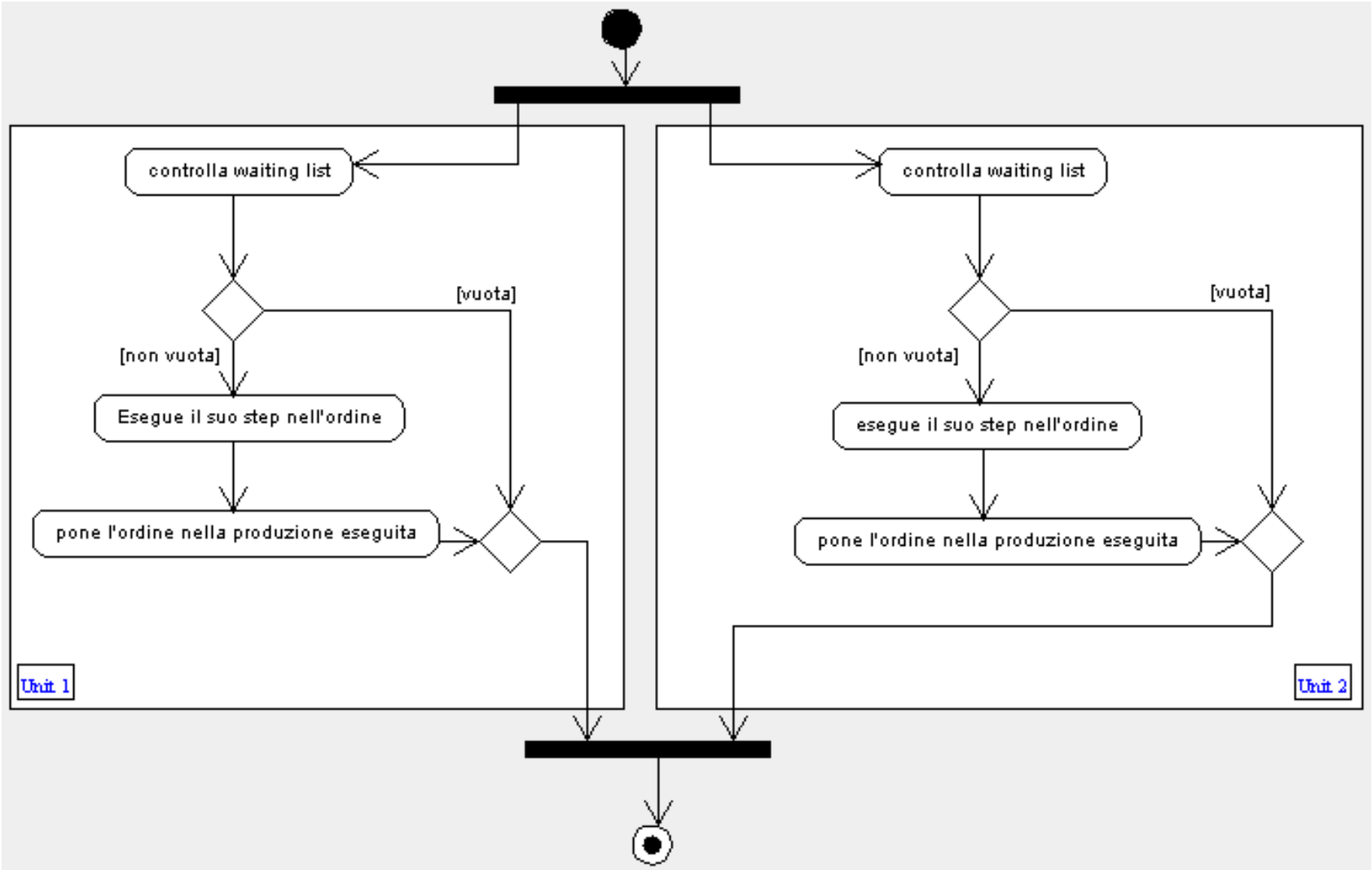
Unit Class diagram



Le Unità

- Le istanze della classe Unit rappresentano le unità produttive dell'impresa.
- Un’unità è identificata dal numero univoco
  - unitNumber
- Ed è caratterizzata dalla fase produttiva che compie
  - unitProductionPhase
- Il vettore *waitingList* contiene gli ordini che devono essere elaborati
- Il vettore *dailyProductionList* contiene gli ordini eseguiti in un giorno

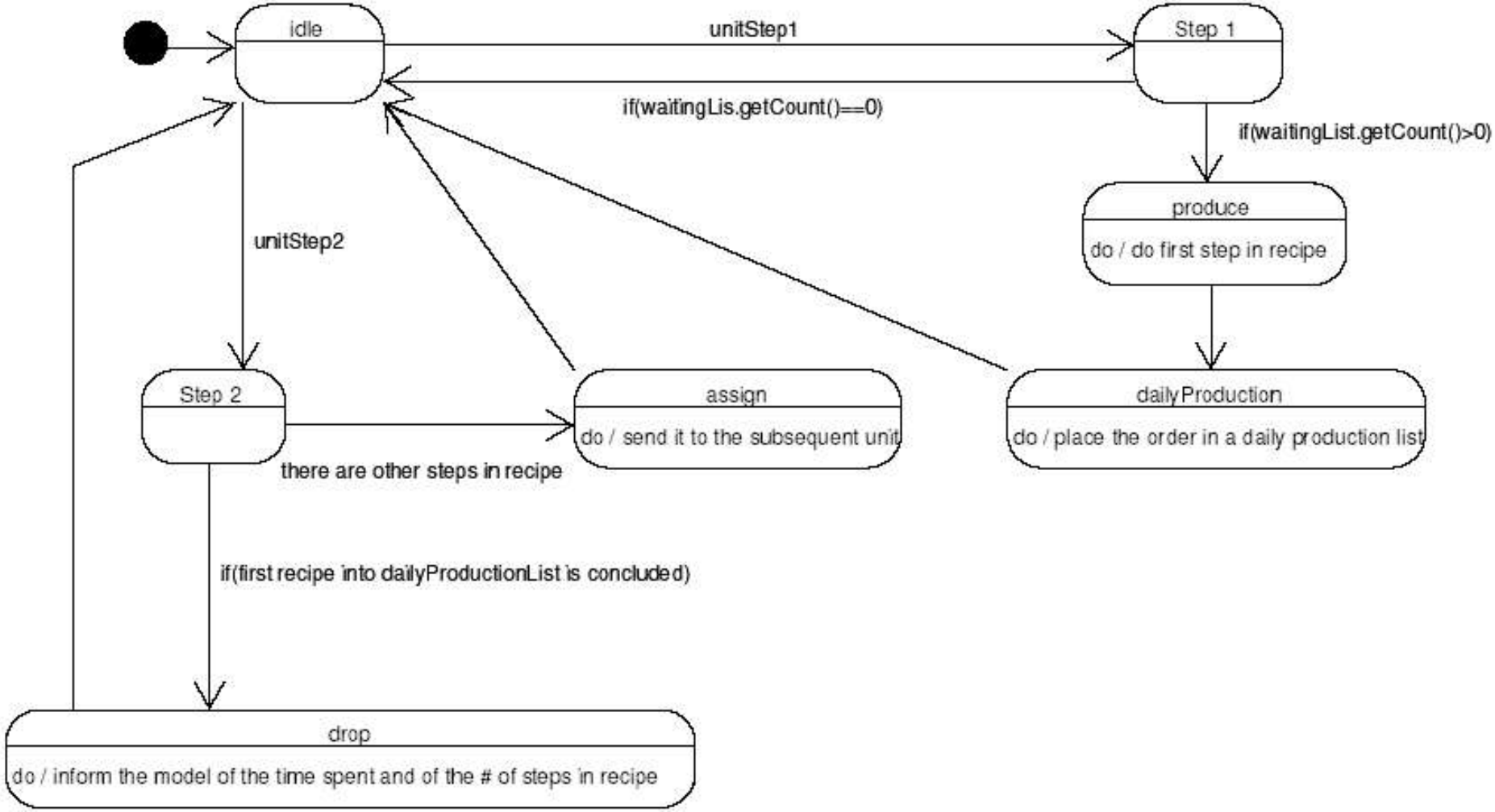
Activity diagram unitStep1



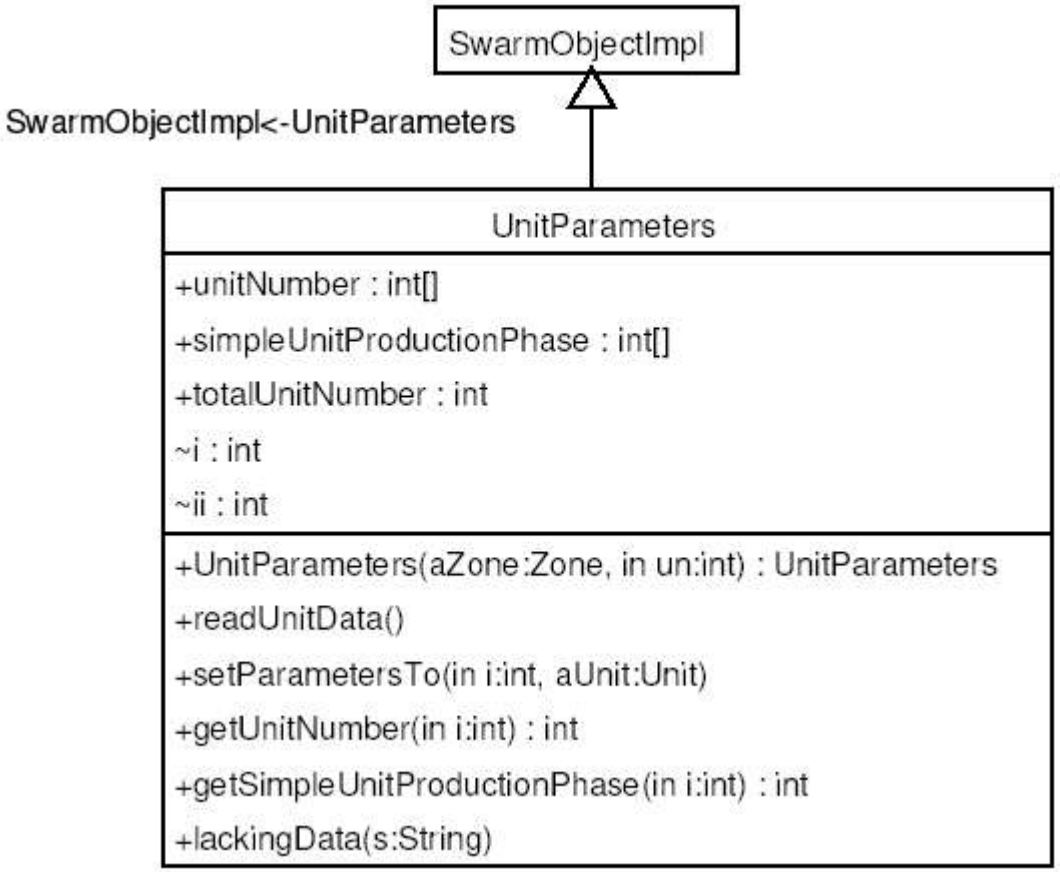
Le Unità

- unitNumber e ProductionPhase sono incapsulati nei rispettivi metodi set e get;
- il metodo setWaitingList ha come parametro un oggetto anOrder di tipo Order
  - inserisce l'oggetto anOrder nella lista waitingList
  - segue un criterio FIFO
- I metodi unitStep1() e unitStep2() sono i metodi principali che caratterizzano la classe Unit

Statechart per le Unità



UnitParameters Class diagram



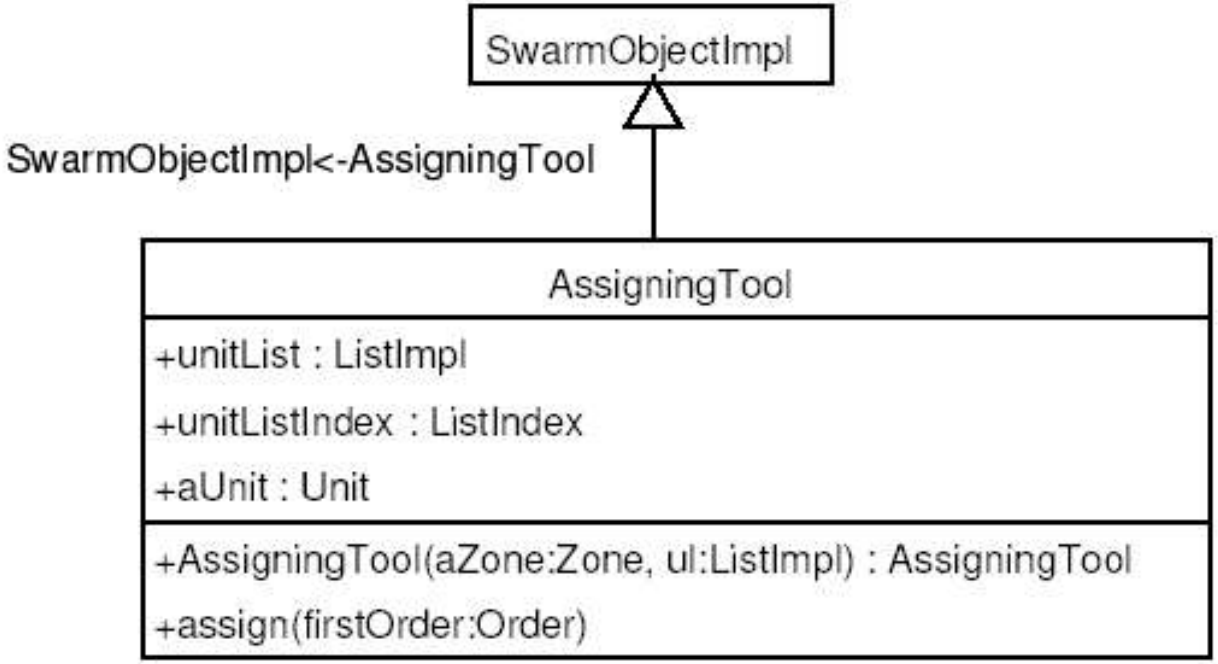
UnitParameters

- Il metodo readUnitData() legge dal file *unitData/unitBasicData.txt* e creadue vettori
  - int[0..totalUnitNumber-1] unitNumber
  - int[] simpleUnitProductionPhase
- Il metodo *getUnitNumber(int i)* ritorna il numero dell'unita che ha posizione i-esima nell'enumerazione delle unità produttive
- Il metodo *setParametersTo(int i, Unit aUnit)* imposta i parametri dell'agente unità aUnit presente all'i-esima posizione nel vettore unitNumber

unitData.txt

- In ogni riga contine
  - il numero identificativo dell'Unità di produzione
  - il numero che indica la sua capacità produttiva
- In jESLet
  - per semplicità, il numero identificativo dell'Unità e quello che indica la sua produzione sono gli stessi
  - sono riportate 10 unità, se è necessario simulare più di 10 unità si devono aggiungere altre righe di codice
  - se più di una unità ha la stessa capacità produttiva, solo la prima viene usata

AssigningTool Class diagram



AssigningTool

- Il vettore *unitList* rappresenta la lista delle unità di produzione normali esistenti nel modello
- Il metodo *assign(Order order)* cerca tra le unità quella che ha fase di produzione uguale al prossimo passo nella ricetta di o*rder*
- Trovata l'unità *aUnit* adatta pone *order* nella lista d'attesa (waitingList) di *aUnit*.

Che cosa fare?
Affrontiamo l'analisi di jES dal lato WD

La classe Recipe

- La classe **Recipe** è usata per leggere le ricette ed il loro codice da un foglio di lavoro *excel*
- Le ricette sono convertite nel formato intermedio di *jES* e sono pronte per essere assegnate ad una lista di ricette per formare un ordine
- Il costruttore della classe imposta il vettore di interi *recipeSteps* di lunghezza 0.

```
//CONSTRUCTOR
public Recipe(Zone aZone)
{
    super(aZone);
    recipeSteps = new int[0];
}
```

La classe Recipe: setRecipeFrom

- Il metodo *setRecipeFrom(ExcelReader e)* carica una ricetta in formato intermedio leggendo in ingresso dall'oggetto di tipo *ExcelReader*
- Tale oggetto indica il file excel in cui sono impostati i dati della ricetta
- Per ogni passo letto si gestiscono i rispettivi casi chiamando opportuni metodi:
  - procurement()
  - or()
  - end()
  - computation()
  - number()

La classe Recipe: number()

- Il metodo number() legge dal file le variabili
  - stepNumber
  - timeUnit
  - timeNumber
- Quindi gestisce l'unità temporale per normalizzarla in secondi e ottenere un corretto timeNumber
- Se timeNumber=0 crea un elemento nullo di tempo 0 codificandolo in formato intermedio con il valore 1000000000+stepNumber
- Altrimenti crea il vettore

  steps={stepNumber,stepNumber,...,stepNumber}

  composto da timeNumber elementi

La classe Recipe: procurement

- La funzione privata *procurement()*
  - legge il numero **n** di componenti necessari all'approvigionamento
  - crea l'array **steps** che contiene la formulazione intermedia della ricetta
- Sia **n**=*numberOfStepsForProcurement* il formalismo intermedio diviene:

  steps={-1,**n**,e**1**,...,e**n**}

  ove -1 è il codice associato ai passi procurement
- Ad esempio: la ricetta in formato esterno {p,2,10,20} viene espressa nel formato intermedio {-1,2,10,20}

La classe Recipe: standAlone batch

- I lotti Stand Alone vengono rappresentati in formato intermedio dal codice -2
- Data la ricetta in formato esterno

  {stepNumber, timeUnit, timeNumber, /, items, e, endUnit}

  si ottiene il formato intermedio

  {-2, timeNumber, items, stepNumber, endUnit}
- Ad esempio, data la ricetta in formato esterno

  {7, s, 1, /, 2000, e, 10}

  si ottiene il formato intermedio

  {-2, 1, 2000, 7, 10}

La classe Recipe: sequential batch

- I lotti sequential vengono rappresentati in formato intermedio dal codice -3
- Data la ricetta in formato esterno

  {stepNumber, timeUnit, timeNumber, \, items}

  si ottiene il formato intermedio

  {-3, timeNumber, items, stepNumber}
- Ad esempio, data la ricetta in formato esterno

  {8, s, 1, \, 100}

  si ottiene il formato intermedio

  {-3, 1, 100, 8}

Order Class diagram

Order Class

- Le istanze della classe Order sono gli oggetti in cui vengono indicati i passi per completare degli ordini
- Il vettore di interi *orderRecipe* contiene i passi della ricetta rappresentata dall'oggetto **order**
- Il vettore *orderState* indica i passi compiuti nella ricetta
- Il metodo *getNextStep()* restituisce il passo successivo da eseguire nella ricetta

Order Class

- Il costruttore della classe order prende i parametri
  - Numero dell'ordine
  - Tempo corrente
  - Lunghezza della ricetta
  - Vettore con i passi della ricetta
  - Modello
  - Lista delle endUnit presenti
- Procede scorrendo il vettore dei passi della ricetta creando il vettore *orderRecipe*

Order Class

- Se l'i-esimo passo dalla ricetta è un numero positivo allora orderRecipe[i] viene impostato a tale numero
- Altrimenti si gestiscono i casi particolari:
  - -1: procurement
  - -2: stand alone batch
  - -3: sequential batch
  - -10: ramificazione or
  - >=-1999 & <= -1001: computazioni
- Imposta a 0 le entrate del vettore *orderState*

Order Class: sequential Batch

- Analizziamo la gestione del caso -3:
  - *orderRecipe[i]* diviene uguale a *stepNumber*
  - Viene istanziata la classe a **SequentialBatchSpecificationSet** con parametri:
    - Numero dell'ordine
    - Numero identificativo del passo i
  - Pone tale oggetto in fondo alla lista *sequentialBatchSpecificationSetList*
  - Esegue il metodo *setSpecifications* dell'oggetto passando i parametri:
    - timeNumber
    - items

Order Class

- Il metodo *getDoneStep()* ritorna l'indice relativo all'ultimo passo di produzione compiuto:
  - se l'ordine è finito ritorna l'ultimo passo della ricetta
  - se non è stato compiuto alcun passo ritorna il valore -999999
- Il metodo *setDoneStep(int u)* cerca il primo passo non compiuto e lo imposta al valore u dell'unità.
- Il metodo *getNumberOfStepsDone()* ritorna il numero di passi compiuti

Classe SequentialBatchSpecificationSet

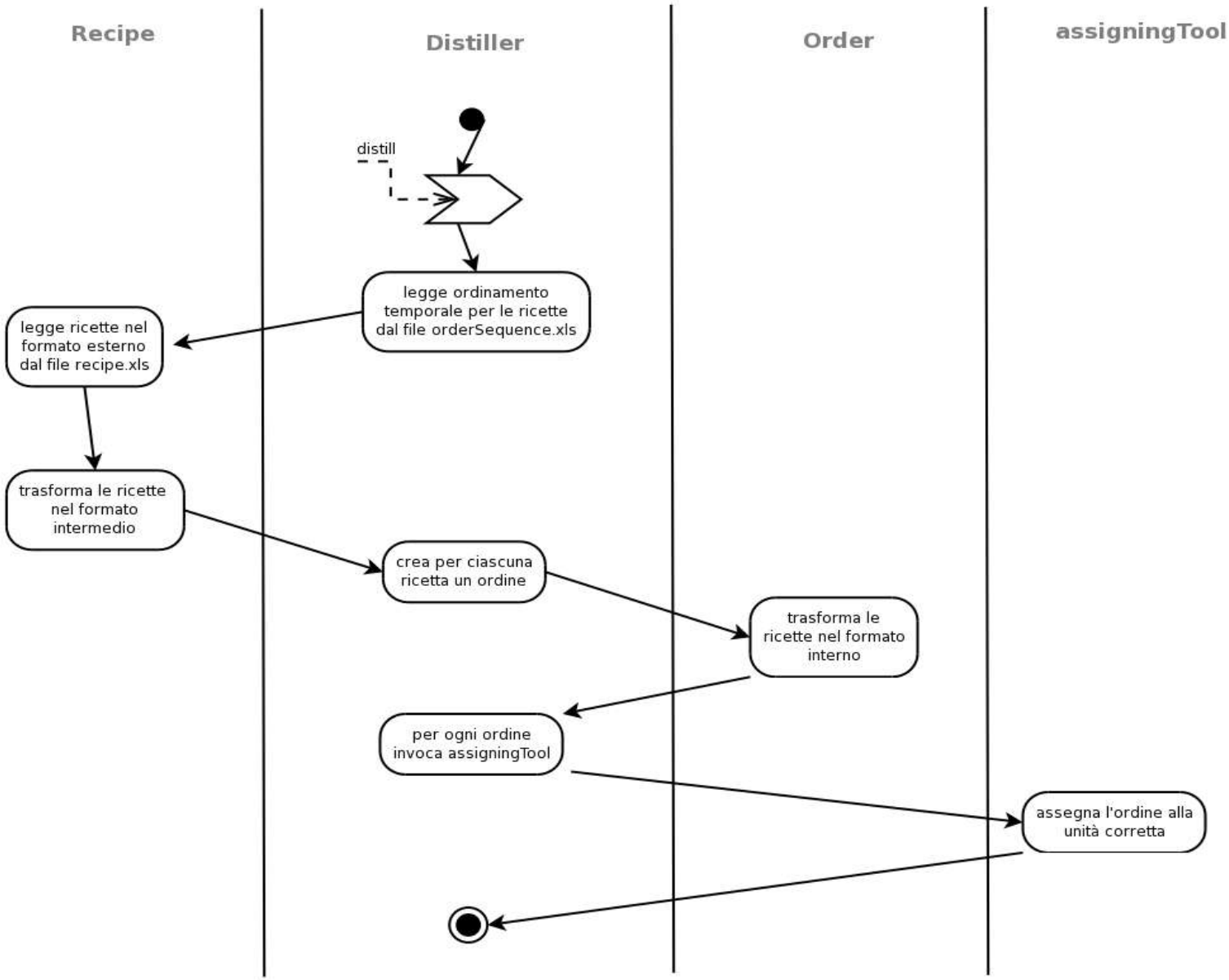
- La classe SequentialBatchSpecificationSet è il contenitore generato da un ordine che mantiene le informazioni su un process sequential batch

- Viene mantenuto il riferimento alla posizione dello step sequential batch nella ricetta

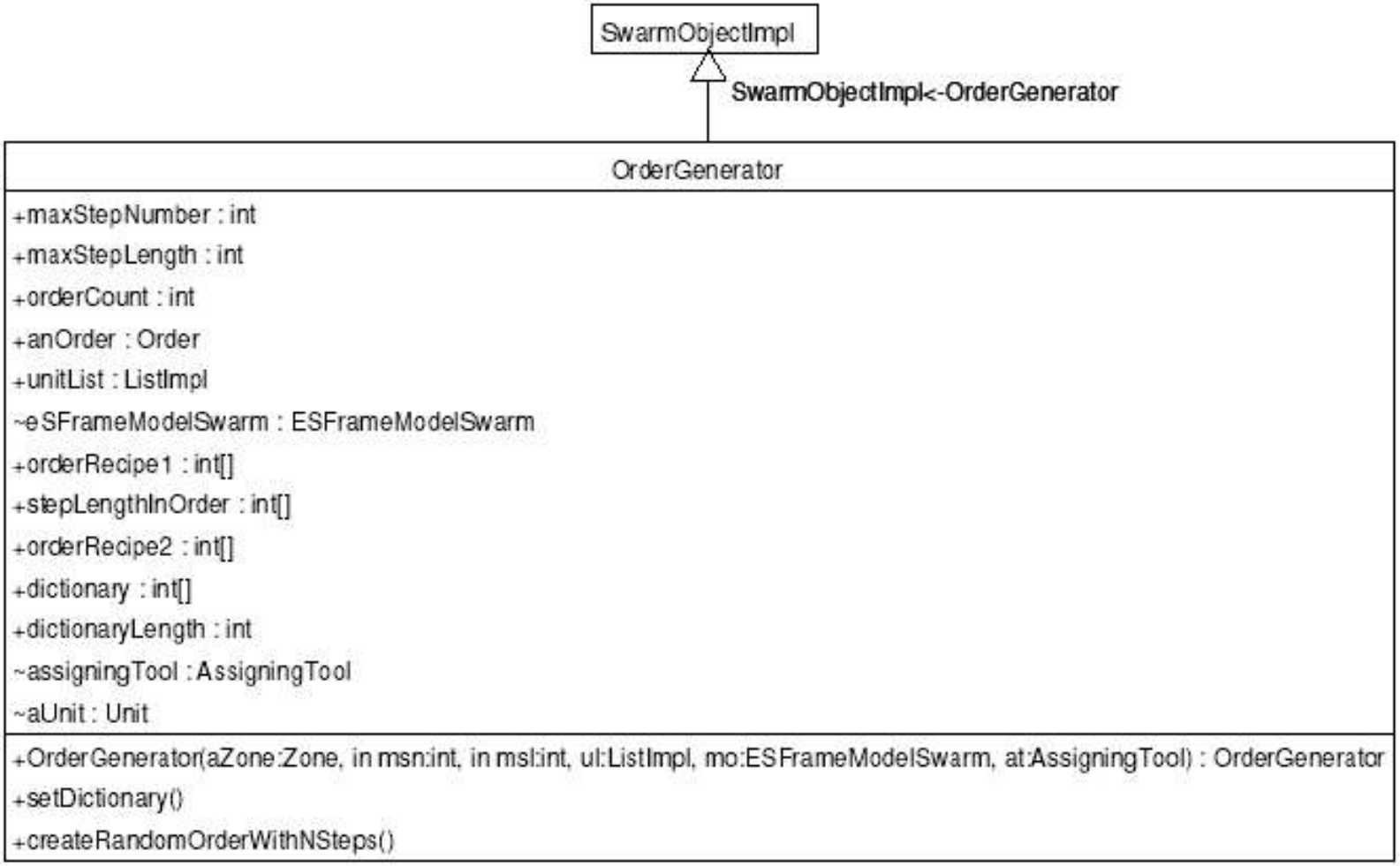
OrderDistiller Class

- Questa classe è usata per leggere i dati da due fogli di lavoro xls:
  - Il primo contenente la lista delle ricette (recipe.xls), sfruttando la classe **Recipe**
  - Il secondo contiene la sequenza temporale degli ordini (orderSequence.xls)
- Il metodo distill() esegue la produzione giornaliera delle ricette
  - Per ciascuna ricetta crea l'ordine
  - Per ciascun ordine invoca **assigningTool** perché venga gestito dalla opportuna **Unit**

OrderDistiller Class



OrderGenerator Class diagram



OrderGenerator Class

- orderRecipe1:
  - vettore di interi che contiene una specifica ricetta di produzione in una forma che non comprende i tempi necessari per la produzione di ciascun passo
- stepLengthInOrder:
  - indica i tempi di produzione per ciascun passo
- orderRecipe2:
  - rappresenta la ricetta di produzione in cui sono esplicitati i tempi di produzione di ciascun passo

orderRecipe example

- Ad esempio:
  - orderRecipe1 = {1, 12, 7, 3}
  - stepLengthInOrder = {1, 3, 2, 2}
  - orderRecipe2 = {1, 12, 12, 12, 7, 7, 3, 3}

OrderGenerator Class

- setDictionary()
  - crea il vettore dictionary
  - per ogni unità listata in unitList aggiunge a dictionary un'entrata contenente un intero che rappresenta la fase di produzione di quella unità
- createRandomOrderWithNSteps()
  - sceglie casualmente la lunghezza della ricetta
  - riempie ogni step della ricetta con un elemento di dictionary e imposta casualmente la durata degli step.
  - Crea l'ordine e lo assegna all'unita di produzione tramite assigningTool

OrderGenerator Class

- Quanto visto è la descrizione della classe **OrderGenerator** per ***jESLet***, in cui si creano casualmente solo ricette semplici.
- La classe **OrderGenerator** in ***jES*** gestisce anche la creazione casuale di ricette di tipo
  - SequentialBatch
  - Stand alone batch
  - Procurement
  - ...

La classe SequentialBatchAssembler

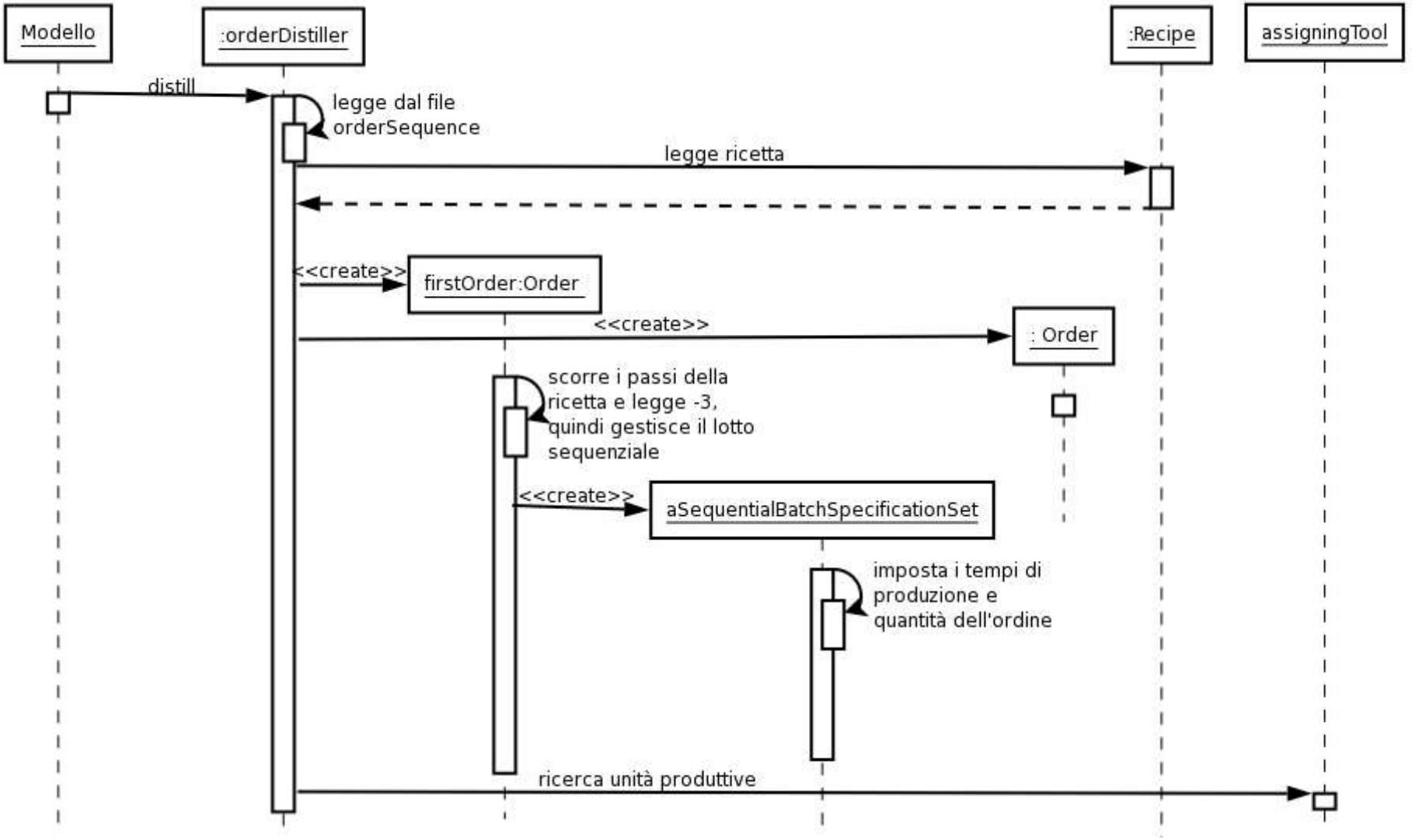
- La classe **SequentialBatchAssembler** assembla i processi per i lotti sequenziali
- Il metodo *setUnit* mantiene riferimenti
  - alla Unit a cui si riferisce (che sta eseguendo l'ordine)
  - alla waitingList dell'unità Unit
- Il metodo clearToProduce() controlla se è stato raggiunto il numero adeguato di ordini per far partire il lotto sequenziale

Esempio del ciclo di vita di un lotto sequenziale

- Consideriamo un'unità con fase 8
- e la ricetta di codice 101 {8, s, 1, \, 2}
- L'ordinamento temporale scritto in *orderSequence.xls* è {101, *, 2}
- All'inizio della simulazione il modello, nel metodo *buildObjects()* crea le unità, e comunica ad **orderDistiller** di leggere le ricette.

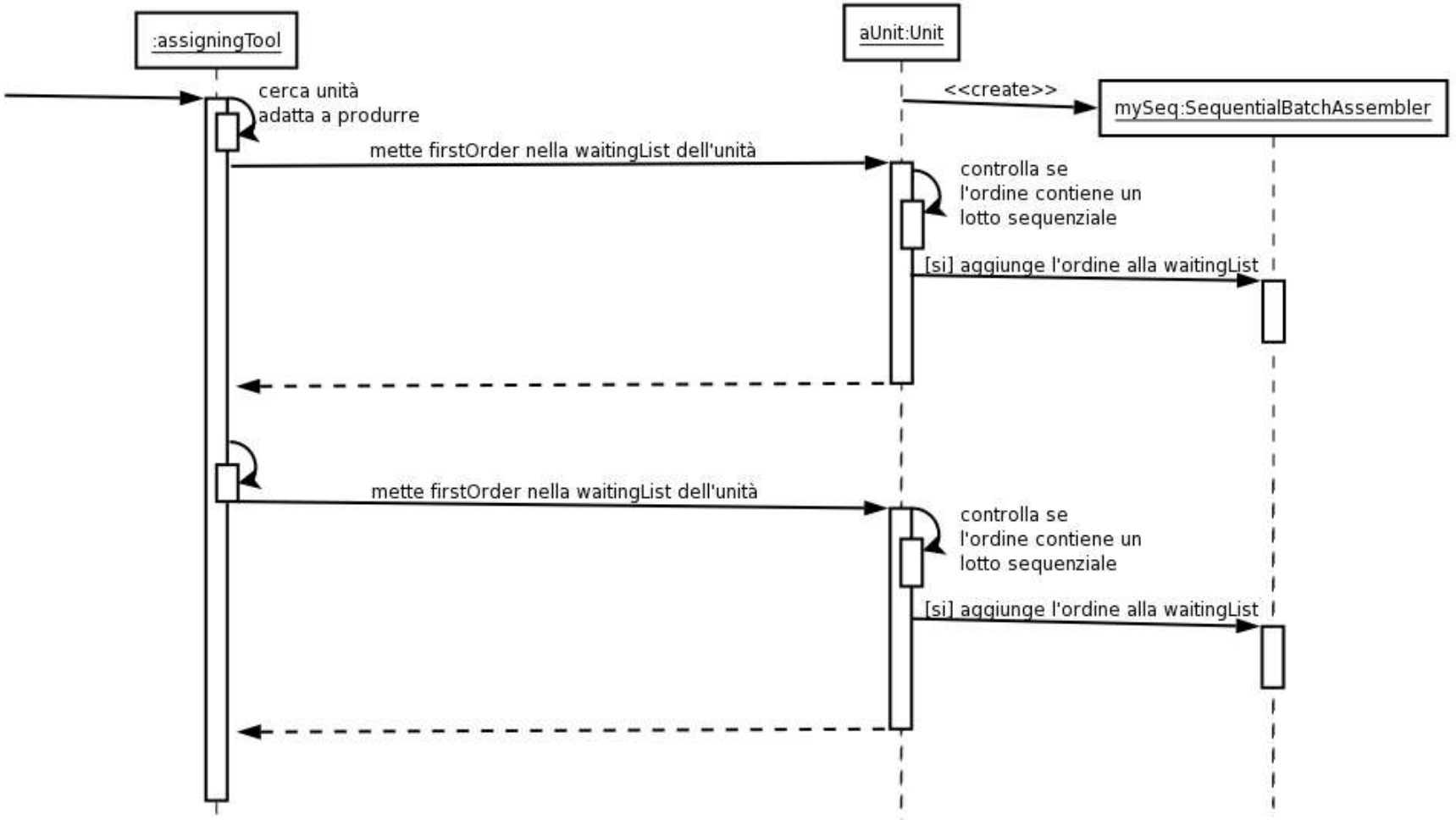
Esempio del ciclo di vita di un lotto sequenziale

Tick 1



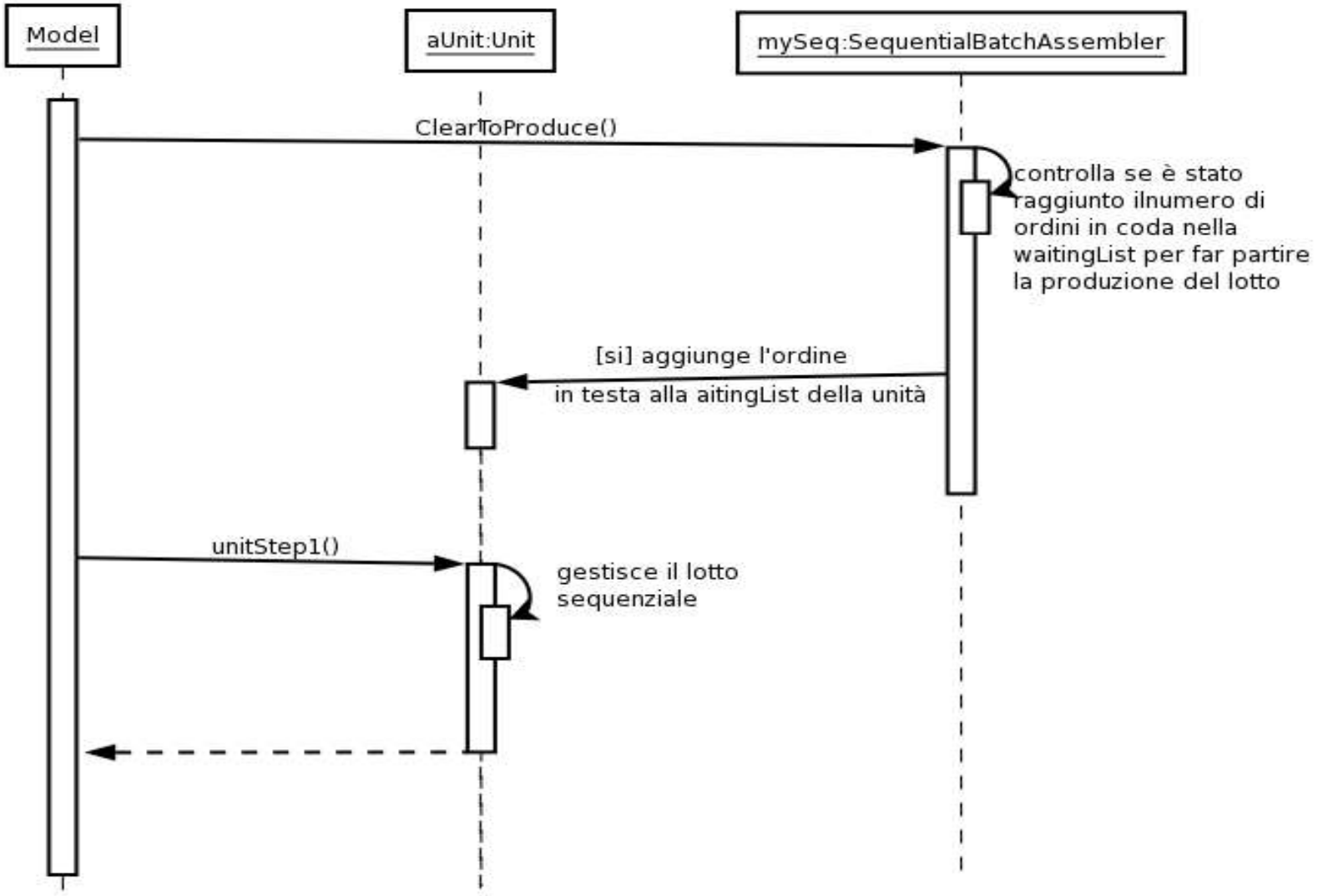
Esempio del ciclo di vita di un lotto sequenziale

Tick 1



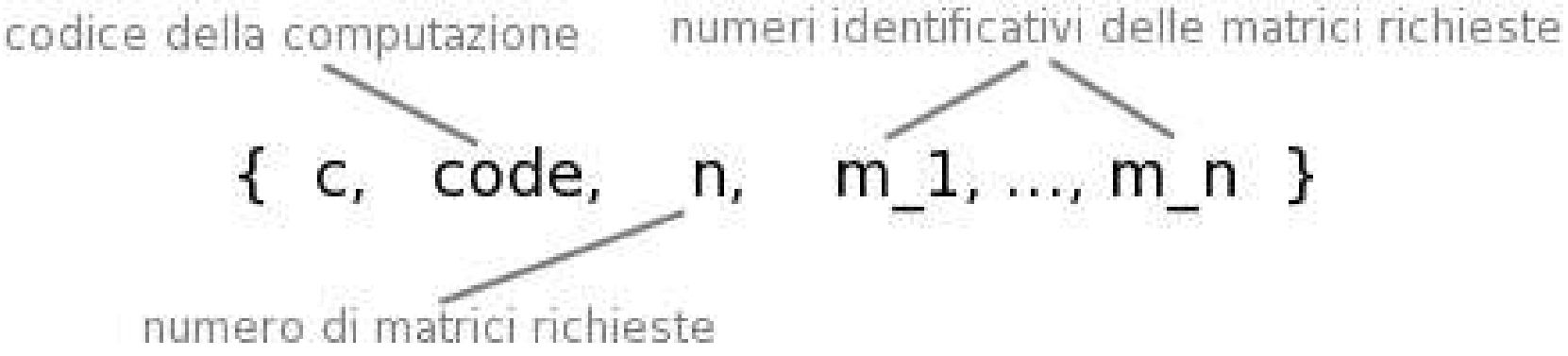
Esempio del ciclo di vita di un lotto sequenziale

Tick 2



Le capacità computazionali (1/7)

- Per lavorare sulle capacità computazionali di jES si deve modificare il file ***ComputationalAssembler.java***
- Nel file ***unitData/memoryMatrixes.txt*** sono contenute le informazioni sulle matrici di memoria (nella forma: n, righe, colonne)
- Le ricette che chiedono una computazione hanno la forma:



Le capacità computazionali (2/7)

- Si consideri la ricetta (*ove è stato sottolineato il passo con computazione*)

  A={1, s, 1, c, 1999, 3, 0, 1 ,3, 2, s, 2, 3, s, 2}

- Il passo 2, che richiede 2 secondi, richiede la computazione di codice 1999 che sfrutta 3 *matrici di meomoria* (0,1,3).
- Nel formato intermedio tale ricetta diviene

  {1, 2 , 2, -1999, 3, 0, 1, 3, 1000000002, 3, 3}

(dove ancora è stato sottolineato il passo con computazione)

Le capacità computazionali (3/7)

- La capacità computazionale con codice 1999 ha la forma seguente:

```
public void 1999() {
        mm0 = memorymatrix(0);
        mm1 = memorymatrix(1);
        mm3 = memorymatrix(2);
        layer = getOrderLayer();
        if( !( mm0.getEmpty(layer,0,0)    || mm1.getEmpty(layer,0,0)
                                          || mm2.getEmpty(layer,0,0) ) )
                mm0.setEmpty(layer,0,0);
                mm1.setEmpty(layer,0,0);
                mm2.setEmpty(layer,0,0);
                done=true;
        }
}
```

Le capacità computazionali (4/7)

- Quando un ordine con ricetta A è eseguito, dopo le 2 unità richieste dal passo 2, viene fatto un controllo sulle matrici 0, 1 e 3 per verificare se nella posizione (0,0) i campi sono vuoti (nel layer corretto).
- Se tutti questi non sono vuoti, vengono svuotati e il passo computazionale è impostato a ***fatto*** (*done=true*).

Le capacità computazionali (5/7)

- Se la variabile done è falsa,
  - La ricetta non vede compiuto il passo computazionale
  - Non si può proseguire con il passo successivo
- Nell'esempio:
  - Se done fosse false
    - La ricetta A non potrebbe procedere con il passo 2
  - La ricetta A non può procedere con il passo 2 se non sono stati eseguiti passi che hanno precedentemente riempito i campi (0,0) nelle matrici 0, 1, e 3.

Le capacità computazionali (6/7)

- La capacità computazionale con codice 1998 ha la forma seguente:

```
public void 1998() {
        mm0 = memorymatrix(0);
        layer = getOrderLayer();
        mm0.setValue(layer,0,0,1.0);
        done=true;
}
```

- Consideriamo le ricette seguenti (mostrate in formato esterno e intermedio):

B={1, s, 1, c, 1998, 1, 0, 5, s, 2} = {1, 5, 5, -1998, 1, 0, 1000000005}

C={1, s, 1, c, 1998, 1, 1, 6, s, 2} = {1, 6, 6, -1998, 1, 1, 1000000006}

D={1, s, 1, c, 1998, 1, 3, 7, s, 2} = {1, 7, 7, -1998, 1, 3, 1000000007}

Le capacità computazionali (7/7)

- Quando l'ordine con ricetta B viene esguito, dopo le 2 unità temporali richieste dallo step 5, la matrice 0 è interessata da un'operazione di scrittura nella posizione (0,0) in cui viene scritto il valore 1.0
- Se l'ordine è il C, la matrice interessata è la 1
- Se l'ordine è il D, la matrice interessata è la 3
- Nell'esempio:
  - La ricetta A non può procedere con il passo 2 se non sono stati eseguiti i passi c1998 in ordini con ricetta B, C, D.

Riferimenti

- http://www.eclipse.org
- http://www.eclipseuml.com
- http://www.swarm.org
- http://web.econ.unibo.it/terna/jes/