Capitolo 7

Java Virtual Enterprise (jVE)

Introduzione

A confronto dell'analisi svolta da alcune organizzazioni nel campo delle strutture di supporto allo sviluppo di nuovi modelli di business, quali possono essere le imprese virtuali, intendo qui osservare modelli che si pongono come obiettivo l'analisi dei sistemi complessi con la finalità di individuare e, laddove fosse possibile, risolvere problemi di organizzazione nonché formalizzazione della conoscenza e gestione concreta della realtà osservata.

In particolare la mia attenzione si focalizza sulla descrizione di un modello che, tramite la possibilità offerta dall'utilizzo della libreria di funzioni Swarm descritta nei capitoli precedenti, permette di osservare l'interazione tra agenti dei quali viene descritto il comportamento e le attività che sono in grado di compiere.

L'utilità, così come l'interesse scientifico risulta dall'osservazione della simulazione, all'interno della quale i fenomeni *accadono* realmente, avendo perciò la possibilità di interpretare risultati reali, con tutti i vantaggi che ne derivano.

Il modello jVE fonda la propria attività sulla descrizione, in maniera il più possibile astratta e tale da adeguarsi agevolmente ai molteplici casi aziendali che si potrebbero presentare.

L'applicazione del modello consta di più fasi di attività:

1. La prima riguarda l'impostazione e la continua evoluzione del codice informatico, perseguendo l'obiettivo della massima flessibilità, tramite l'utilizzo della tecnologia orientata agli oggetti propria del linguaggio Java, con il pregio fondamentale di permettere una descrizione degli agenti e dei loro comportamenti facilmente evolvibile;

2. La seconda attività, parte preponderante della preparazione di questo lavoro, riguarda l'estrapolazione della conoscenza dalla realtà aziendale che si prende in considerazione. Concettualmente si tratterebbe di osservare la realtà e formalizzare i processi tramite i quali si pone in essere l'attività aziendale. Nella pratica molti ostacoli si interpongono in questa attività, in particolar modo la necessaria razionalizzazione che il management delle imprese fa dei propri processi e delle proprie attività, che è però cosa diversa dalla formalizzazione del proprio business;
3. In una terza fase gli apporti al modello, derivanti dalle singole applicazioni aziendali, devono essere considerati nel loro complesso e da questi si deve astrarre una evoluzione generale del modello;

Un'ultima fase di applicazione vera e propria del modello alla singola impresa permette di osservare, rilevare ed analizzare il risultato dei comportamenti derivanti dall'estrapolazione della conoscenza. Allo stesso tempo è possibile sperimentare modifiche nella gestione, ottenendo risultati realmente verificabili nell'ambito di un sistemo complesso quale è quello aziendale, ma di questo si parlerà nel capitolo seguente nell'ambito dell'indagine sui risultati.

JVE

Con l'applicazione del metodo che di seguito descritto, si intende perciò far funzionare una azienda virtuale, non nel senso considerato nei capitoli precedenti, ma tramite una simulazione vera e propria dell'interrelazione fra agenti dotati di una propria intelligenza, di proprie capacità e, come è tipico dei modelli di simulazione fondati su agenti, della possibilità di osservare e registrare tali relazioni e quelle con l'ambiente.

L'impresa virtuale avrà perciò delle caratteristiche definite a priori, ma che non determineranno i risultati che potrà ottenere e, soprattutto, che non ci daranno mai la possibilità di definire precedentemente i risultati stessi o le scelte che verranno effettuate.

Proprio in quanto detto sino ad ora, si riscontra la verosimiglianza con un *videogame,* in cui gli accadimenti non sono tutti definiti a priori,

bensì emergono da una serie comportamenti che determinano complessità, come definito nel capitolo relativo, delle interazioni tra l'agente virtuale e l'agente umano, o per lo meno da una sua fedele rappresentazione.

La struttura

La modellizzazione del progetto si fonda sullo schema di relazioni fra l'agente, l'ambiente e i gestori di regole, anche se potrà facilmente essere esteso a creatori di regole tipici dell'intelligenza artificiale, così come osservato nella spiegazione della struttura Environment – Rules – Agents.

In particolare il formalismo adottato si basa sulla distinzione fra *what to do* (*WD*) e *which is doing what* (*DW*), ossia la contrapposizione tra quale sia lo scopo di vita dell'azienda e cosa essa deve fare per ottenere il risultato sperato.

Gli agenti WD vengono espressi da sequenze di numeri interi, generalmente positivi, rappresentativi di quali possano essere i passi da eseguire per giungere all'ottenimento del prodotto finito o del servizio, che rappresenta lo scopo dell'impresa virtuale.

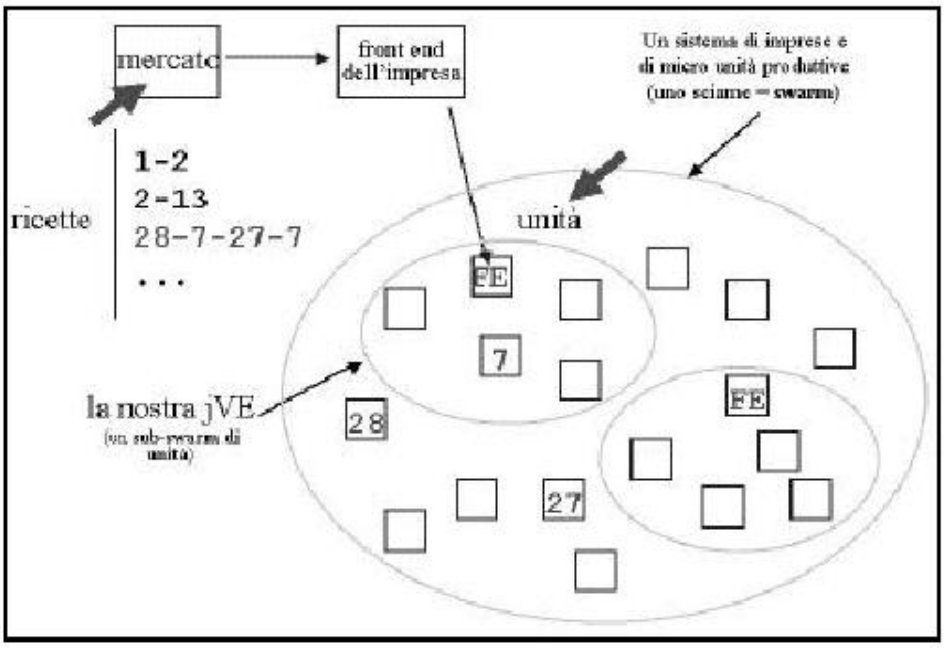


Figura 1: Sequenze di attività e unità in JVE. Fonte Simulazione ad agenti in contesti di impresa (Terna 2000).

In contrapposizione si trovano le unità che dovranno compiere le lavorazioni o le attività necessarie per l'ottenimento del servizio finale e che eseguiranno le attività.

Ad entrambe le parti verrà attribuita una certa dose di *intelligenza*, da un lato la capacità di determinare quali siano le attività da svolgere e in quali tempi, mentre, dall'altro, la capacità di svolgere tali attività e registrare eventuali costi o tempi di transizione tra stati e, in futuro, di riconoscere quindi la possibilità di condividere le risorse.

Per meglio spiegare il significato del distribuire l'intelligenza, si può utilizzare una metafora: si immagini di avere ricevuto una lista di ingredienti necessari per ottenere un piatto di pasta particolare, e di avere a disposizione sia il necessario contenuto nella lista, sia gli strumenti per lavorare tali ingredienti; manca però una cosa fondamentale ossia la *ricetta* che contiene le sequenze delle attività da svolgere per giungere al risultato finale.

Le ricette nel nostro caso non sono altro che sequenze di attività di impresa dotate di intelligenza, le quali, grazie all'interazione con unità dotate delle funzionalità necessarie per lavorarle o per farle transitare da uno stato ad un altro, permettono di raggiungere lo scopo prefissato in maniera non determinata a priori, ma diversa, di volta in volta.

Come si osserva nella figura 1, l'impresa virtuale entra in contatto con il mercato per mezzo di unità cosiddette *Front End* (*FE*), le quali ricevono ordini sotto forma di sequenze di numeri (28 8 65 45 2), successivamente le unità che hanno in carico tali ordini vanno a interrogare le altre unità e se stesse per sapere quale di loro è in grado di eseguire tale passo della ricetta, e così via sino ad aver ultimato la produzione.

Ogni ordine è gestito in maniera decentrata, e così è anche per le informazioni riguardanti la contabilità ed ogni altro evento necessario.

Le unità erano inizialmente in grado di lavorare solo ed esclusivamente un tipo di passo della ricetta, ossia quello corrispondente al proprio numero identificativo; allo stesso tempo le assegnazioni avvenivano in ordine *First In First Out (FIFO).* Nel corso del presente lavoro verranno osservate le modifiche che il modello ha subito per giungere alla sua impostazione attuale.

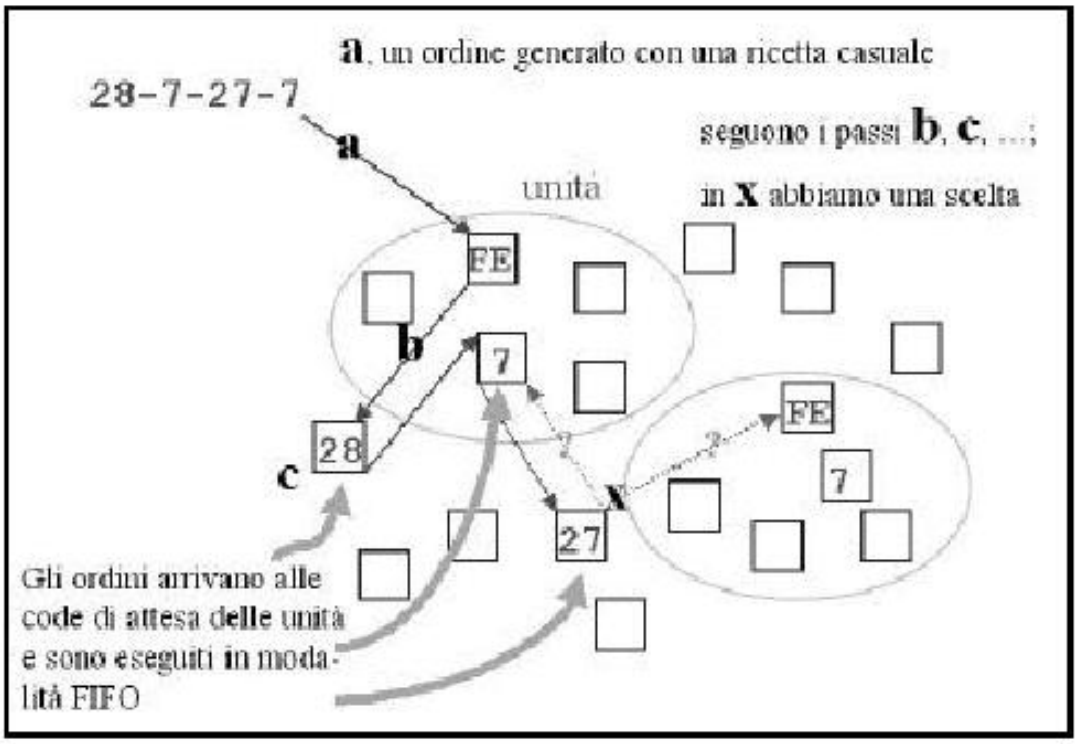


Figura 2: Assegnazione delle lavorazioni alle unità. Fonte Simulazione ad agenti in contesti di impresa (Terna 2000).

Per quanto riguarda la gestione dei magazzini e delle scorte, essa può avvenire attraverso la creazione di *depositi* centri paralleli alle unità in grado di contenere, se necessario, un certo quantitativo di componenti che varia all'interno di limiti determinati nei parametri della simulazione e che è creato in seguito all'interazione con un gestore delle regole del magazzino: la classe *InventoryRuleMaster.java*. A quest'ultimas è affidato il compito di gestire le scorte e di inviare internamente degli ordini nel caso, ad esempio, in cui il limite superiore venisse superato.

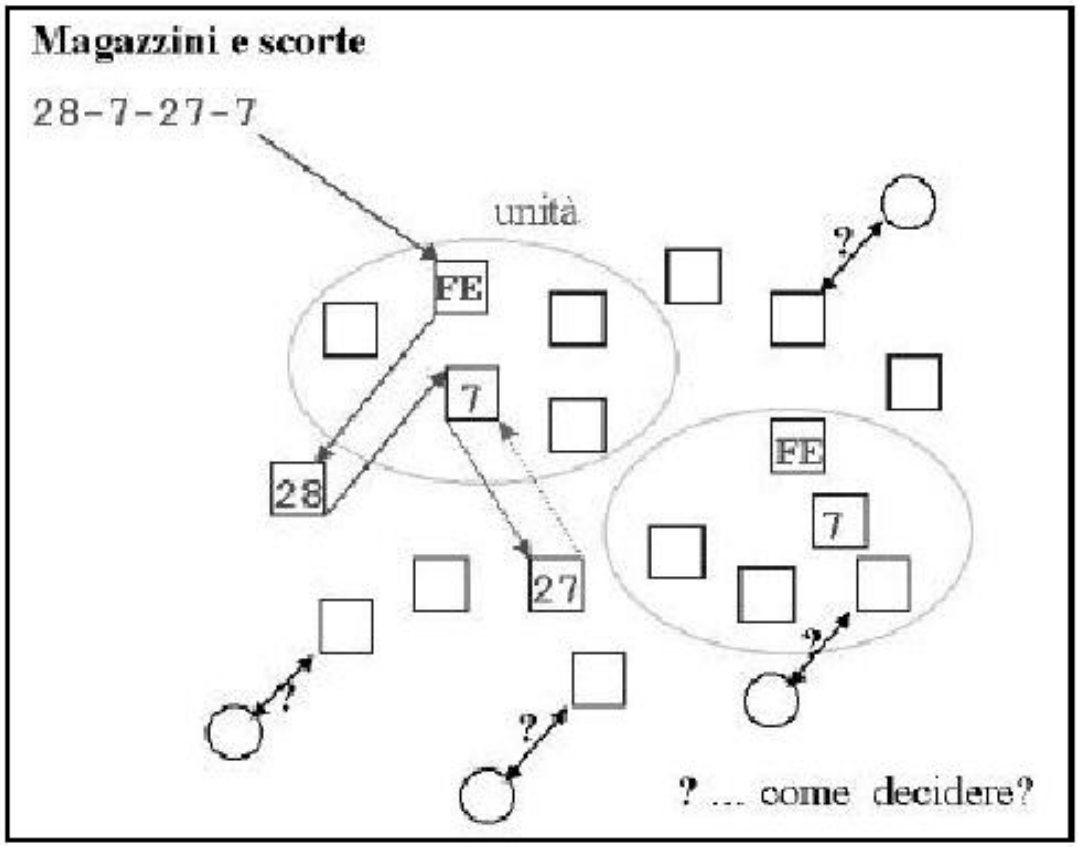


Figura 3: Magazzini e scorte. Fonte Simulazione ad agenti in contesti di impresa (Terna 2000).

Il problema di gestire la trasmissione della conoscenza relativa agli ordini interni è notevole, soprattutto se si pensa alla possibilità che una unità venga occupata da lavoro interno per il magazzino, quando è in arrivo un ordine transitato inizialmente per il Front End, come si è osservato nel capitolo relativo alle supply chain la trasmissione dell'informazione diviene, quindi, un punto cruciale.

La mancanza di uno strumento di indagine sugli effetti della circolazione dell'informazione, avrebbe fatto di JVE uno strumento incompleto, in particolar modo nell'epoca della rivoluzione dell'informazione; ecco allora che la creazione di un gestore di regole per questo aspetto del problema, favorisce lo sviluppo del modello e

permette di compiere valutazioni importanti per una simulazione d'azienda.

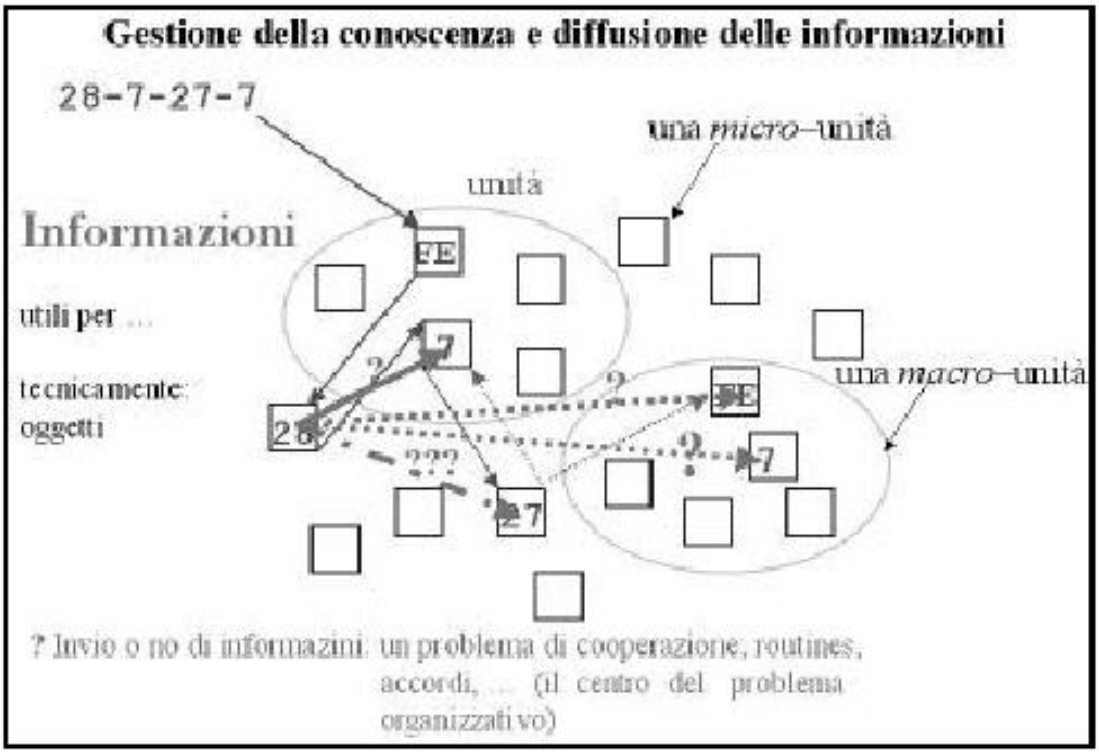


Figura 4: Un problema di circolazione della conoscenza. Fonte Simulazione ad agenti in contesti di impresa (Terna 2000).

La questione della circolazione delle informazioni è tanto più valida se si considera la possibilità della scelta della destinazione delle risorse scarse condivise tra più unità produttive, aspetto molto interessante da sperimentare. Infatti è possibile che, in presenza di risorse condivise e di vincoli di impiego tra più unità produttive, emergano interazioni di rilevante complessità.

Infine, la circolazione delle informazioni è un problema molto realistico nella realtà aziendale ed è in grado di determinare il maggior o il minor successo dei risultati economici simulati dal funzionamento del modello. Esso è però anche fortemente connesso con l'analisi teorica fatta nel capitolo quarto sulla scuola austrica in economia: infatti la

decentralizzazione della conoscenza e l'insorgere di comportamenti di tipo organizzativo non derivanti da una fonte centrale rispecchiano a fondo il pensiero haekyano, soprattutto quando si dichiara in disaccordo con le regole di pianificazione di Lerner e Lange.

L'interazione umana produce mercati attivi ma che, difficilmente, si possono definire a priori a causa di una divisione della conoscenza fra gli agenti che sono gli attori del mercato. Fin troppo semplice è il raccordo fra molti degli argomenti che Hayek porta a sostegno delle proprie tesi e la complessità emergente da simulazioni ad agenti.

Strumenti di indagine come JVE, possono rappresentare un raccordo fra la realtà e la teoria come mai prima era accaduto per le scienze sociali. La concezione di dinamicità dei sistemi, cui Hayek era molto più interessato dei concetti di equilibrio statico, è un punto di raccordo fondamentale fra il suo pensiero e gli studi sulla complessità.

Non solo, ma da un altro punto di vista la possibilità offerta da uno strumento di indagine mediante simulazione, in grado di osservare anche vie non tentate precedentemente dall'imprenditore, o la capacità di adattamento che un tale sistema potrebbe sviluppare, pongono in essere quella figura dell'imprenditore innovatore e ricercatore che sfrutta le opportunità derivanti dagli errori per ottenere un profitto.

Quest'ultimo è eticamente giustificato dalla posizione attiva dell'imprenditore nei confronti sia dell'attività di scoperta di nuovi prodotti sia nella ricerca che JVE permette, grazie alla sua struttura di indagine e a nuovi processi organizzativi e gestionali.

Il modello tiene un aggiornamento continuo della contabilità dei processi attivi sia con una prospettiva tradizionale di contabilità per unità produttive, con costi fissi e costi variabili, sia con una prospettiva

analitica della contabilità secondo i prodotti è così possibile operare anceh scelte di carattere economico e non solo organizzativo.

Un'ultima considerazione in merito alla descrizione del funzionamento del modello jVE, fa fatta con particolare riferimento all'indagine che gli effetti dell'*information techology* possono avere in azienda: la presenza di sub-processi, cioè vere e proprie forniture che devono intervenire all'interno della catena di produzione, con problemi di sincronizzazione dell'inizio delle diverse produzioni e delle relative componenti, sia che si tratti di gestione di reti all'interno di una stessa azienda, sia di un sistema integrato di aziende è osservabile grazie all'inserimento di strutture di questo tipo.

Un campo di prova la V.I.R. S.p.A.

L'applicazione del modello di simulazione sino ad ora introdotto avviene grazie alla possibilità offerta dai titolari della VIR e grazie all'aiuto ed ai consigli ottenuti dal management dell'impresa.

La VIR è un'impresa industriale che opera nel settore delle valvole e delle rubinetterie da più di 30 anni. Situata in Valduggia, polo piemontese che, insieme a quello bresciano, rappresenta la quasi totalità della produzione di valvole e rubinetterie italiane, svolge attività di produzione meccanica tradizionale.

Il prodotto valvola non è ad alto contenuto tecnologico, pertanto l'azienda ha puntato su un'altissima tecnologia nei mezzi e nei metodi di produzione, ossia ad investire sui processi interni e, il lavoro svolto è un esempio di analisi delle opportunità di business ottenuto grazie a strumenti innovativi.

La fonderia, con i moderni forni ad induzione, il reparto lavorazioni meccaniche, dove le operazioni di carico e scarico delle macchine transfer avvengono tramite robot, ed infine, il reparto assemblaggio, con le sofisticate apparecchiature, progettate e realizzate dalla VIR sono le basi del successo ottenuto.

Le materie prime utilizzate dall'azienda sono:

1. l'ottone per l'88%;
2. il bronzo per il 6%;
3. la plastica per il rimanente 6%.

La nostra analisi prenderà in considerazione solo l'ottone, per l'importanza della produzione sul fatturato totale, nella prospettiva di una simulazione.

Analisi interna

La produzione della VIR si articola in tre principali famiglie di prodotti:

- Saracinesche

Figura 5: Una saracinesca

- Valvole a sfera

Figura 6: Una valvola a sfera

- Valvole a globo ed affini

Si svolge inoltre attività di collaudo dei prodotti finiti: ogni valvola è sottoposta alla prova di tenuta idraulica e pneumatica, prima di essere consegnata al cliente.

Con la visita dello stabilimento di Valduggia (VC) il 29 giugno 2001 e la presentazione delle caratteristiche principali del modello di

impresa virtuale, è cominciata la collaborazione con l'azienda al fine di poter simulare i processi aziendali di poter osservare gli effetti di variazioni dell'impostazione dell'azienda o, anche, di singoli parametri.

L'impresa ha così l'opportunità di trarre un'occasione di studio per cogliere aspetti del suo stesso funzionamento, che non aveva preso prima in considerazione, ed una possibilità per osservare l'organizzazione da una nuova prospettiva.

Gli aspetti fondamentali che abbiamo preso in considerazione per impostare la simulazione sono di seguito illustrati.

Gestione ordini

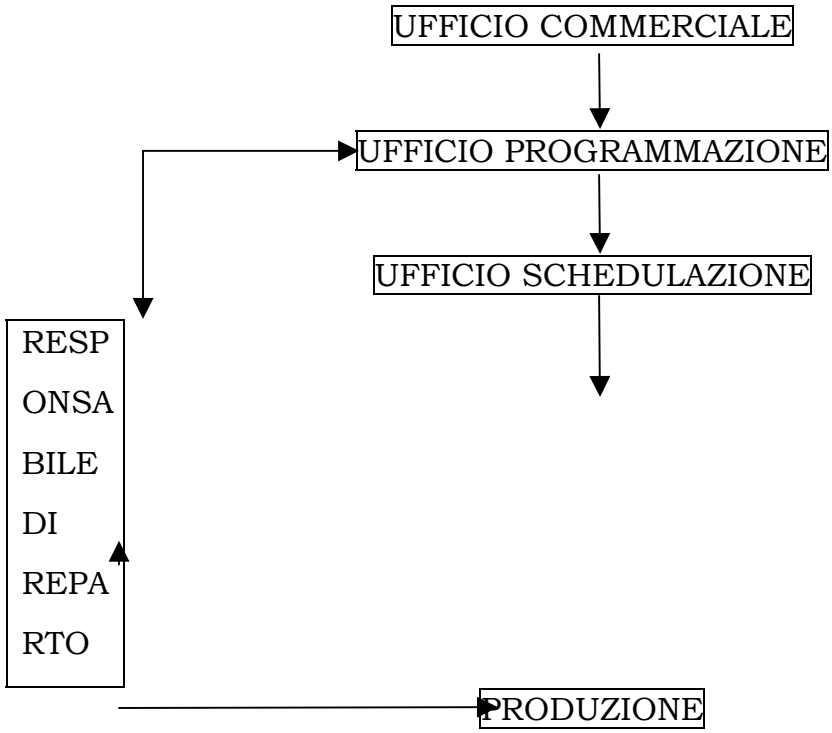
Il punto di partenza è l'ufficio commerciale che si occupa dell'acquisto dei materiali, trasmettendone gli ordini ai fornitori, e della vendita dei prodotti finiti. Sostanzialmente gli ordini che noi prendamo in esame sono basati o su richieste che possono essere evase nei tempi stabiliti con il cliente, anche dovendo avviare del tutto la lavorazione, ordini che possono essere evasi immediatamente poichè i prodotti finiti sono presenti nei magazzini, grazie a previsioni stimate mediante calcoli quali ad esempio la media mobile, meglio nota come storico, oppure ordini derivanti da richieste particolari da parte della clientela, che non potrebbero essere evasi nei tempi stabiliti se non si fosse prevista in anticipo una produzione di componenti che richiedono tempi particolarmente lunghi.

Ricevuti gli ordini, l'addetto alla programmazione stabilisce un programma produttivo giornaliero, partendo da ciò che rappresenta il vero termine della produzione: la spedizione. Si determinano così quali siano le macchine e le risorse umane da impiegare per soddisfare le

richieste provenienti dal mercato. Un programma per la produzione delle componenti è stilato regolarmente ogni mese ed è aggiornato quotidianamente, in base alle effettive risorse disponibili ed alle effettive priorità dei nuovi ordini.

A questo punto, l'ufficio schedulazione si occupa della stampa e della divulgazione delle informazioni inviate alla produzione. In produzione i responsabili di reparto organizzano il lavoro e, se necessario, comunicano con l'ufficio di programmazione.

Ecco uno schema delle relazioni esistenti fra i vari uffici:



Processo produttivo.

Il processo produttivo comincia subito dopo l'arrivo delle materie prime ordinate ai fornitori da parte dell'ufficio acquisti. Le barre di ottone ricevute vengono sistemate nel magazzino materie prime, anche se in realtà il magazzino in Vir non viene considerato esplicitamente, in

quanto si considera unicamente la movimentazione e l'accatastamento di cassoni contenenti i pezzi da lavorare o montare. A questo punto viene effettuata una lavorazione esterna di pressatura delle barre per far sì che prendano la forma desiderata.

I grezzi vengono depositati nell'apposito magazzino. Da qui in avanti i pezzi passano in torneria, dove viene effettuata una lavorazione meccanica per asportare il truciolo dai singoli pezzi.

Qui sono presenti quattordici macchine programmate a seconda del materiale usato, del tipo di valvola da produrre, della dimensione della valvola, degli operai e di altre caratteristiche prettamente tecniche quali ad esempio le tolleranze dei materiali.

I componenti lavorati così ottenuti, escono dalla torneria e confluiscono nel magazzino componenti lavorati, alcuni pezzi prima di arrivare in magazzino possono subire un trattamento superficiale ulteriore effettuato all'esterno.

Successivamente vi è la fase di montaggio per la quale, in caso di lotti numerosi, vengono utilizzati processi totalmente automatizzati, per cui il macchinario necessita unicamente di assistenza e non di intervento diretto per la lavorazione. I problemi sorgono in caso di piccoli lotti, che non giustificano il riassettamento delle macchine o quando si verificano lavorazioni complicate, tanto da non poter essere automatizzate. In questi ultimi due casi l'assemblaggio viene fatto con l'ausilio umano. Le macchine presenti nel reparto di montaggio sono quaranta di cui dodici finalizzate all'ottenimento del prodotto finito, mentre le altre assemblano parti per ottenere semilavorati da utilizzare in altri momenti della produzione.

Sempre in linea vengono effettuati i test di collaudo dei prodotti che sono successivamente sistemati nel magazzino come prodotti finiti

ed infine vengono imballati e caricati nel sistema pronti per la spedizione.

Contabilità.

La VIR utilizza un sistema extracontabile di rilevazione dei costi per centri, ossia essendo presenti centri di produzione e centri ausiliari, vengono effettuate rilevazioni separatamnte per ogni centro. Successivamente i costi dei centri ausiliari vengono ribaltati sui centri di produzione e questi sono imputati, insieme ai suoi costi diretti sul prodotto.

Il costo totale dei centri viene riversato sui prodotti tramite un criterio di ripartizione basato sul tempo (uomo/macchina).

Per ogni pezzo si ha un tempo, uomo o macchina standard, in base alla produzione ogni pezzo si moltiplica per il tempo standard. Successivamente il costo totale del centro si divide per il risultato appena ottenuto e si ottiene un indice di costo. Quest'ultimo, moltiplicato per il tempo standard del prodotto e sommato al valore dei materiali, attribuiti ai prodotti in base all'utilizzo, origina il costo totale del prodotto.

1. *Pezzi * Tempi Standard Uomo/Macchina = α*
2. *Costo totale Centro / α = Indice di Costo*
3. *Indice di Costo * Tempo Standard di Prodotto = Costo di Prodotto*
4. *Costo di Prodotto + Materiali = Totale Costo del Prodotto*

Magazzino

Nell'azienda sono presenti, concettualmente, diversi magazzini ma non si lavora tenendo conto delle valutazioni finanziarie riguardanti le scorte.

Mensilmente si calcola un valore atteso di domanda, che si calcola perverrà loro durante il mese, il cui conteggio viene effettuato sulla base di una media mobile ad ogni periodo, ne consegue che, un surplus imprevisto di domanda provoca lo slittamento dell' evasione dell' ordinativo al mese successivo.

Attrezzaggi

I macchinari sono in grado di effettuare lavorazioni diverse o su pezzi di varie dimensioni semplicemente modificando alcune parti intercambiabili: questa operazione è definita attrezzaggio. L'attrezzaggio ha tempi diversi a seconda del cambio di articolo necessario in seguito alla lavorazione svolta in precedenza.

Focalizzazione sui macchinari

Le fasi di lavorazione dei prodotti sono essenzialmente tre e sono comuni alle due famiglie principali.

Nella prima fase si acquistano le barre e i pani di ottone che vengono prima stampati poi sottoposti a ricottura e quindi sabbiati per ottenere i pezzi grezzi da lavorare. La seconda si compie nel reparto di torneria, per le lavorazioni meccaniche, ed esternamente, per i trattamenti superficiali; il risultato della seconda fase sono le componenti lavorate. La terza fase riguarda il reparto di montaggio in cui si assemblano le componenti lavorate internamente con quelle acquistate all'esterno. Il prodotto finito viene testato poi insacchettato e successivamente imballato.

Di seguito si analizzano nel dettaglio le lavorazioni che subiscono alcuni articoli della famiglia delle valvole a sfera e delle saracinesche, dalla fusione dell'ottone all'imballaggio, con l'indicazione dei macchinari in grado di svolgere le lavorazioni.

Ripercorrendo il processo produttivo a ritroso si verificheranno i seguenti passi:

1. Emissione dell'ordine di lavoro verso il reparto di assemblaggio;
2. Il reparto di assemblaggio richiederà delle componenti da montare;
3. Controllo dei magazzini componenti.

Le valvole a sfera si compongono di dieci parti di cui sono lavorate internamente all'azienda il corpo, il manicotto e la sfera.

Figura 7: Essplosione di una valvola a sfera

Se sono insufficienti si procede:

1. all'acquisto all'esterno di talune parti del prodotto (bullone, maniglia, premistoppa, guarnizione premistoppa, perno, 2 seggi);
2. all'emissione di un ordine di produzione alla torneria (per manicotto, sfera, corpo);
3. La torneria controllerà la disponibilità di pezzi grezzi da lavorare,
4. Acquisto all'esterno dei pani e delle barre.

Il dettaglio di alcuni articoli è riportato per favorire la comprensione della ripartizione effettuata tra le unità del modello, delle capacità di lavorazione di ogni unità:

famiglia 340

1/2 pollice

corpo

1. acquisto dei pani di ottone;
2. stampaggio all'esterno;
3. sabbiatura all'esterno;
4. lavorazione meccanica per asportazione del truciolo:

1. hP A
    1. C5
2. hP B
    1. B1
3. hP C
    1. C1
4. hP D
    1. C2

5. hP E

1. all'esterno

5. cromatura all'esterno;

6. deposito nel magazzino componenti;

manicotto

1. acquisto dei pani di ottone;

2. stampaggio all'esterno;

3. sabbiatura all'esterno;

4. lavorazione meccanica per asportazione del truciolo:

1. hP A

1. A4

2. hP B

1. B5

3. hP C

1. all'esterno

5. cromatura all'esterno;

6. deposito nel magazzino componenti;

sfera

1. acquisto delle barre di ottone;

2. lavorazione meccanica per asportazione del truciolo:

1. hP A

1. D1

3. cromatura all'esterno;

4. deposito nel magazzino componenti;

bullone, maniglia, premistoppa, guarnizione premistoppa, perno, 2 seggi

1. Acquisto all'esterno;

2. Verniciatura all'esterno;

3. deposito nel magazzino componenti;

Assemblaggio:

1. hP A

1. S1 unico percorso di montaggio (dalle componenti alla scatola confezionata);

2. deposito nel magazzino prodotti finiti.

2. hP B

1. P6 dalle componenti al prodotto finito;

2. "STAZIONE DI LAVORO" insachettamento ed inscatolamento manuale;

3. deposito nel magazzino prodotti finiti.

3. hP C

1. P1 seggio nel corpo;

2. L1 seggio nel manicotto;

3. L5 corpo + manicotto + sfera ;

4. L6 assemblaggio finale;

5. "STAZIONE DI LAVORO" insachettamento ed inscatolamento manuale;

6. deposito nel magazzino prodotti finiti.

3/4 pollice

corpo

1. acquisto dei pani di ottone;

2. stampaggio all'esterno;

3. sabbiatura all'esterno;

4. lavorazione meccanica per asportazione del truciolo:

1. hP A

1. C5

2. hP B

1. B1

3. hP C

1. C1

4. hP D

1. C2

5. hP E

1. all'esterno

5. cromatura all'esterno;

6. deposito nel magazzino componenti;

manicotto

1. acquisto dei pani di ottone;

2. stampaggio all'esterno;

3. sabbiatura all'esterno;

4. lavorazione meccanica per asportazione del truciolo:

1. hP A

1. A4

2. hP B

1. B5

3. hP C

1. all'esterno

5. cromatura all'esterno;

6. deposito nel magazzino componenti;

sfera

1. acquisto delle barre di ottone;

2. lavorazione meccanica per asportazione del truciolo:

1. hP A

1. D1

2. hP B

1. C4

3. cromatura all'esterno;

4. deposito nel magazzino componenti;

bullone, maniglia, premistoppa, guarnizione premistoppa, perno, 2 seggi

1. Acquisto all'esterno;

2. Verniciatura all'esterno;

3. deposito nel magazzino componenti;

Assemblaggio:

1. hP A

1. S1 unico percorso di montaggio (dalle componenti alla scatola confezionata);

2. deposito nel magazzino prodotti finiti.

2. hP B

1. P6 dalle componenti al prodotto finito;

2. "STAZIONE DI LAVORO" insachettamento ed inscatolamento manuale;

3. deposito nel magazzino prodotti finiti.

3. hP C

1. P1 seggio nel corpo;

2. L1 seggio nel manicotto;

3. L5 corpo + manicotto + sfera ;

4. L6 assemblaggio finale;

5. "STAZIONE DI LAVORO" insachettamento ed inscatolamento manuale;

6. deposito nel magazzino prodotti finiti.

1 pollice

corpo

1. acquisto dei pani di ottone;

2. stampaggio all'esterno;

3. sabbiatura all'esterno;

4. lavorazione meccanica per asportazione del truciolo:

1. hP A

1. A2

2. hP B

1. C1

5. cromatura all'esterno;

6. deposito nel magazzino componenti;

manicotto

1. acquisto dei pani di ottone;

2. stampaggio all'esterno;

3. sabbiatura all'esterno;

4. lavorazione meccanica per asportazione del truciolo:

1. hP A

1. A4

2. hP B

1. B3

5. cromatura all'esterno;

6. deposito nel magazzino componenti;

sfera

1. acquisto delle barre di ottone;

2. lavorazione meccanica per asportazione del truciolo:

1. hP B

1. C4

3. cromatura all'esterno;

4. deposito nel magazzino componenti;

bullone, maniglia, premistoppa, guarnizione premistoppa, perno, 2 seggi

1. Acquisto all'esterno;

2. Verniciatura all'esterno;

3. deposito nel magazzino componenti;

Assemblaggio:

1. hP A

1. L4b seggio nel corpo,seggio nel manicotto,corpo + manicotto + sfera ;

2. L4 assemblaggio finale;

3. "STAZIONE DI LAVORO" insachettamento ed inscatolamento manuale;

4. deposito nel magazzino prodotti finiti.

famiglia 342

3/4 pollice

corpo

1. acquisto dei pani di ottone;

2. stampaggio all'esterno;

3. sabbiatura all'esterno;

4. lavorazione meccanica per asportazione del truciolo:

1. hP A

1. C1 prima fase;

2. C2 seconda fase;

2. hP B

1. C2 att2 prima fase;

2. C2 seconda fase;

5. cromatura all'esterno;

6. deposito nel magazzino componenti;

manicotto

1. acquisto dei pani di ottone;

2. stampaggio all'esterno;

3. sabbiatura all'esterno;

4. lavorazione meccanica per asportazione del truciolo:

1. hP A

1. A4

2. hP B

1. B5

3. hP C

1. all'esterno

5. cromatura all'esterno;

6. deposito nel magazzino componenti;

sfera

1. acquisto delle barre di ottone;

2. lavorazione meccanica per asportazione del truciolo:

1. hP A

1. D1

2. hP B

1. C4

3. cromatura all'esterno;

4. deposito nel magazzino componenti;

bullone, maniglia, premistoppa, guarnizione premistoppa, perno, 2 seggi

1. Acquisto all'esterno;

2. Verniciatura all'esterno;

3. deposito nel magazzino componenti;

Assemblaggio:

1. hP A

1. P1 seggio nel corpo;

2. L1 seggio nel manicotto;

3. L5 corpo + manicotto + sfera ;

4. L6 assemblaggio finale;

5. "STAZIONE DI LAVORO" insachettamento ed inscatolamento manuale;

6. deposito nel magazzino prodotti finiti.

Analogamente a quanto visto per le valvole a sfera, anche le saracinesche sono composte da dieci parti, ma soltanto il corpo, il cuneo e il vitone sono prodotti internamente.

Figura 8: Esplosione di una saracinesca

Analizziamo nel dettaglio le fasi di produzione di alcuni articoli.

famiglia 42

3/4 pollice

corpo

1. acquisto dei pani di ottone;
2. stampaggio all'esterno;
3. sabbiatura all'esterno;
4. lavorazione meccanica per asportazione del truciolo:
    1. hP A
        1. B1
5. deposito nel magazzino componenti;

vitone

1. acquisto dei pani di ottone;
2. stampaggio all'esterno;
3. sabbiatura all'esterno;
4. lavorazione meccanica per asportazione del truciolo:
    1. hP A
        1. A4
    2. hP A
        1. B3
5. deposito nel magazzino componenti;

cuneo

1. acquisto dei pani di ottone;
2. stampaggio all'esterno;
3. sabbiatura all'esterno;
4. lavorazione meccanica per asportazione del truciolo:
    1. hP A
        1. B4
5. deposito nel magazzino componenti;

dado, volantino, dado premistoppa, 2 ghiere, guarnizione premistoppa, asta

1. Acquisto all'esterno;
2. deposito nel magazzino componenti;

Assemblaggio:

1. hP A
    1. O4
    2. O5
    3. "STAZIONE DI LAVORO" insachettamento ed inscatolamento manuale;
2. spedizione

1 pollice

corpo

1. acquisto dei pani di ottone;
2. stampaggio all'esterno;
3. sabbiatura all'esterno;
4. lavorazione meccanica per asportazione del truciolo:
    1. hP A
        1. B1
5. deposito nel magazzino componenti;

vitone

1. acquisto dei pani di ottone;
2. stampaggio all'esterno;
3. sabbiatura all'esterno;
4. lavorazione meccanica per asportazione del truciolo:
    1. hP A
        1. A4
    2. hP A
        1. B3
5. deposito nel magazzino componenti;

cuneo

1. acquisto dei pani di ottone;

2. stampaggio all'esterno;

3. sabbiatura all'esterno;

4. lavorazione meccanica per asportazione del truciolo:

1. hP A

1. B4

5. deposito nel magazzino componenti;

dado, volantino, dado premistoppa, 2 ghiere, guarnizione premistoppa, asta

1. Acquisto all'esterno;

2. deposito nel magazzino componenti;

Assemblaggio:

1. hP A

1. O4 att2

2. O5 att2

3. "STAZIONE DI LAVORO" insachettamento ed inscatolamento manuale;

2. spedizione

Come emerge dall'analisi del dettaglio degli articoli, le due famiglie principali di prodotti sono segmentate in una molteplicità di dimensioni e sottocaratteristiche in continua evoluzione in armonia con le esigenze di mercato, entrando così in contrasto con le rigidità del sistema determinate dalle risorse finanziarie, dai tempi di evoluzione dei macchinari e dalle regole del mercato del lavoro.

In VIR la gestione delle risorse umane è vista come una necessità di assistenza che le singole unità lavorative hanno sia per svolgere la loro attività di produzione sia per assumere le diverse "vesti" che permettono di eseguire più lavorazioni (attrezzaggi).

Le assunzioni e il numero di turni vengono stabiliti sulla base dei volumi produttivi, previsti ed effettivi, e del grado di flessibilità del mercato del lavoro

Nel reparto torneria ogni addetto è in grado di seguire più di una macchina per volta.

Il rapporto può essere di un addetto per due macchine o di due addetti per tre macchine.

Il personale è generalmente specializzato per lavorare soltanto sulle macchine cui è assegnato e nel caso sia una coppia di addetti molto spesso soltanto uno dei due è in grado di operare gli attrezzaggi.

All'interno del reparto vi sono però persone che conoscono lavorazioni e attrezzaggi di più macchine.

Al contrario della torneria, nel reparto montaggio, non è possibile seguire contemporaneamente le lavorazioni di più macchine perché alcune stazioni di montaggio sono soltanto manuali.

Come per la torneria, ogni dipendente è in grado di seguire soltanto la macchina cui è assegnato e nel caso di una coppia di addetti spesso soltanto uno è in grado di fare gli attrezzaggi

La Vir è un'azienda sottodimensionata e lo sfruttamento delle macchine a pieno regime per la torneria e solo parziale per il montaggio , è condizionato dai contratti con i dipendenti e dalle impostazioni dei turni che non sono modificabili in modo repentino e ripetuto.

Dalla realtà aziendale precedentemente analizzata, organizzata secondo i criteri di un'azienda meccanica che produce su larga scala, emerge la varietà di articoli in produzione, apparentemente ripetitivi, ma differenziabili per dimensioni e sottocaratteristiche e le molteplici unità produttive in grado di svolgere le diverse lavorazioni con tempi e attrezzaggi differenti a seconda dell'articolo trattato. Volendo analizzare nei dettagli le caratteristiche, talvolta minime, che differenziano tra loro articoli e unità produttive, si rischierebbe di perdere la visione d'insieme

della realtà aziendale non riuscendo più ad astrarne le informazioni necessarie ai fini di un'analisi economica.

Guardando invece all'impresa come sistema economico si è in grado di separare due distinti elementi di conoscenza: le fasi di lavorazione per realizzare i prodotti e le unità con le caratteristiche necessarie per compiere le fasi descritte.

L'applicazione del modello jVE ad una realtà così intricata non può prescindere dall'individuazione di queste due componenti.

Astrazione e jVE primo passo

La Java Virtual Enterprise, modello di impresa virtuale in JavaSwarm *(Terna, 2002)*, si pone l'obiettivo di riuscire a simulare le implicazioni dovute a modifiche operate nel comportamento organizzativo delle imprese, grazie all'analisi dell'attività di queste come entità composte di una plurità di agenti indipendenti. Ogni agente è dotato di una propria intelligenza ed è posto in relazione con altri agenti dotati, a loro volta, di conoscenza.

Caratteristica propria delle simulazioni fondate su agenti, è quella di essere in grado di far emergere comportamenti imprevisti ma, ancor meglio, di mettere in evidenza situazioni imprevedibili tipiche della realtà, rappresentando, ad esempio, irregolarità nella gestione del processo produttivo dovute ad errori nella trasmissione delle informazioni, estranee all'impostazione del modello.

Durante l'analisi della realtà aziendale della Vir S.p.A. , sono emersi molti aspetti notevolmente rilevanti ai fini della nostra analisi, con implicazioni per la rappresentazione sia per ciò che riguarda le

singole attività aziendali, sia per ciò che riguarda le funzioni svolte dalle persone che compongono l'azienda.

Senza il ricorso all'astrazione delle attività che presentano caratteri comuni e dei comportamenti degli operatori e dei manager, per ottenere comportamenti il più generalizzati possibile, non si sarebbe potuto costruire un modello rappresentativo della realtà; infatti gli aspetti da prendere in considerazione sarebbero troppo numerosi da includere, anche cercando di sfruttare al meglio le potenzialità di calcolo dei computer, la versatilità della programmazione ad oggetti, nonché la metodologia di simulazione basata sugli agenti e non per processo di cui si è discusso nei capitoli precedenti.

Durante lo svolgimento di questa attività di semplificazione, sovente la difficoltà di estrapolare la conoscenza dall'azienda, in maniera ben definita, ha rappresentato difficoltà non immaginabili. Il problema principale si incontra quando è necessario ottenere dati, non da utilizzare nella prospettiva di chi debba descrivere un modello del quale può fare uso unicamente per confermare che, al verificarsi di determinate condizioni, si avrà una determinata situazione od un determinato comportamento (risultato tra l'altro difficilmente ottenibile poichè sarebbe conseguente ad assunzioni e semplificazioni notevoli).

L'uso delle informazioni è tale da differenziarsi da qualunque modello il quale, seppur molto raffinato, prescinda dal lasciare agli "attori" la libertà di stabilire le proprie regole di comportamento; ciò che si cerca di osservare è l'implicazione che una certa dotazione di *scienza* dell'agente ha nel modello.

Il tentativo da parte del management di razionalizzare i processi sia produttivi, sia di trasmissione delle informazioni è lecito, se si pensa alla quantità innumerevole di variabili da cui la realtà aziendale è

influenzata, ma può rivelarsi di intralcio per l'attività di raccolta dei dati ed astrazione che deve essere condotta. Nello sviluppo dei ragionamenti che si sono rivelati necessari per l'applicazione del modello jVE, il compito più difficile in assoluto è stato proprio quello di non considerare la razionalizzazione che i manager della Vir S.p.A. hanno fatto della propria attività. Occorre cercare di cogliere direttamente ciò che nella azienda è fatto, da chi e come, osservando con occhio critico quali sarebbero potute essere le alternative, volontariamente abbandonate dal management, in seguito a valutazioni più o meno elaborate e di diversa natura (valutazioni tecniche, economiche, personali). Tutto ciò si rivela necessario per poter descrivere più a fondo l'ambito di libertà degli agenti simulati.

L'attività di raccolta dei dati nella prospettiva finora descritta ha reso possibile impostare uno schema di comportamento e di relazione fra gli agenti che permettesse di osservare le conseguenze dell'assegnazione dell'intelligenza in maniera distribuita.

Siamo in grado di osservare l'emergenza di comportamenti indipendenti dei singoli agenti, non conseguenti a scelte che siano il risultato del lavoro di un gruppo centrale, ma che rappresentino scelte operate realmente all'interno del modello da soggetti autonomi in posizione decentralizzata, in grado perciò di rappresentare un frame, cioè una immagine della realtà, che si svolge mentre viene osservata, non la rappresentazione ipotetica di un modello *as if*.

Un grado di astrazione elevato nelle scelte di modifica al modello è perseguito con la esplicita volontà di permettere ulteriori applicazioni anche in settori completamente differenti dalla metalmeccanica tradizionale. Il modello potrà così essere applicato ad altre imprese, al credito oppure a forme molto innovative di azienda senza perdere la

caratteristica di flessibilità, astrattezza ma anche di maggior realismo che lo contraddistingue dagli altri tipi di emulazione della realtà.

Evoluzioni progressive

Rispetto al modello di impresa virtuale sinora considerato, molteplici sono le modifiche da apportare per raggiungere l'obiettivo di rappresentare l'azienda nel modo il più possibile corrispondente al vero; si è stabilito di procedere ad un suo primo sviluppo, inserendo la gestione del tempo in maniera più intricata del semplice susseguirsi di cicli rappresentativi dei giorni, preparando così le basi per una ulteriore evoluzione che consentirà di mostrare il trascorrere tempo come è normalmente conteggiato, con la distinzione che ci è naturale considerare (in secondi, minuti, ore, giorni, ecc…). Quest'ultimo è anche oggetto di considerazioni concernenti la conoscenza intrinseca dei singoli agenti descritti, perciò l'introduzione di nuovi formalismi di gestione del tempo all'interno del modello jVE richiede una attenta analisi delle *classi,* che sono state a loro volta modificate per cogliere le importanti innovazioni sia nella prospettiva del funzionamento della simulazione, sia per quanto riguarda il notevole passo verso una rappresentazione più concreta della realtà.

Inoltre il metodo adottato sottende l'attribuzione di una maggiore informazione direttamente all'agente rappresentativo della sequenza produttiva, va quindi anche osservato, per le implicazioni teoriche nel campo della complessità, che in questo specifico caso sorge interazione tra agenti e dalle scelte su come avviene la distribuzione dell'intelligenza all'interno del modello medesimo.

Sulla struttura formale del programma, la necessità di miglioramento della gestione del tempo ha avuto riflessi diretti su quasi tutte le *classi* che componogono il modello; anzitutto sono state apportate delle modifiche alla classe *OrderGenerator.java* per l'introduzione del diverso periodo di esecuzione di una lavorazione rispetto alle altre: si è stabilito di sostituire il vettore *orderRecipe[],* che conteneva l'indicazione della sequenza di passi necessari per ottenere un prodotto finito, con un nuovo vettore

public int[] orderRecipe1

avente sempre la funzione di raccolta dell'informazione circa le lavorazioni da svolgere. Si è stabilito di introdurre un nuovo vettore

public int[] stepLengthInOrder

contenente le unità di tempo necessarie per consentire la generazione del vettore finale

public int[] orderRecipe2.

Valga come esempio *Terna [2001]*:

…the integer vector containing the unit of time spent for each step in the recipe; il will determine the tranformation of the orderRecipe1 vector into the orderRecipe2 vector containing the repetitions

of the steps, to reproduce the length of each step in time as an examples

orderRecipe1 contains 1 12 7 3

stepLengthInOrder contains 1 3 2 2

orderRecipe2 contains 1 12 12 12 7 7 3 3.

Sia la lunghezza ed il contenuto di *orderRecipe1[]* sia quella di *stepLengthInOrder[]* sono di tipo casuale, come era nella versione precedente del programma, mentre ciò non vale sia per il contenuto sia per la lunghezza del vettore *orderRecipe2[].*

```
orderRecipe1 = new int[maxStepNumber];
        stepLengthInOrder = new int[maxStepNumber];
        orderRecipe2 = new int[maxStepNumber*maxStepLength];
```

Dopo aver determinato casualmente un valore compreso fra uno ed il massimo numero di unità attivabili nella simulazione ed aver inizializzato il vettore dei tempi

```
        randomStepNumber=Globals.env.uniformIntRand.getIntegerW
ithMin$withMax(1,maxStepNumber);
    for (i=0;i<maxStepNumber;i++)stepLengthInOrder[i]=0;
```

si procede alla assegnazione dei valori relativi scegliendo, se la variabile maxStepLength è maggiore di una unità, valori casuali inferiori alla variabile stessa

```
        if (maxStepLength>1){
            for (i=0;i<randomStepNumber;i++)
```

stepLengthInOrder[i]=Globals.env.uniformIntRand.getIntegerWithMin$with Max

(1, maxStepLength);}

else {for (i=0;i<randomStepNumber;i++)stepLengthInOrder[i]=1; }.

Infine si assegnano i valori al vettore *orderRecipe2[]*

```
count=0;
for (i=0;i<randomStepNumber;i++)
   for (ii=0;ii<stepLengthInOrder[i];ii++){
         orderRecipe2[count]=orderRecipe1[i];
         count++;}.
```

Il resto del programma rimane pressochè invariato salva la necessità, quando viene richiamata una istanza della classe *Order.class,* di passare al costruttore della classe il vettore orderRecipe2[] anziché il vettore orderRecipe[]

anOrder = new Order(getZone(), orderCount,Globals.env.getCurrentTime(),

randomStepNumber, orderRecipe2, vEFrameModelSwarm);

L'assegnazione dell'informazione, relativa ai tempi di produzione, alle ricette è il risultato di un duplice ragionamento:

1. da un lato la necessità di astrarre il più possibile il formalismo adottato nella scrittura del codice informatico, poiché il modello deve potersi indistintamente applicare a più realtà aziendali, siano esse pubbliche, private di settori tradizionali od innovativi. In questa prospettiva il modello deve mantenere un elevato grado di flessibilità, che non si otterrebbe con l'aggiunta di questa informazione alle Unit;
2. dall'altro lato il modello è il risultato di una impostazione per la quale l'intelligenza è distribuita agli agenti, in questo caso è l'agente *Recipe* l'unico in grado di contenere ed utilizzare bene l'informazione in oggetto.

Alla classe StartVEFrame viene apportata unicamente una modifica formale del codice per indicare che l'aggiornamento alla versione jVEFrame-0.5.1 è stato eseguito

Globals.env.initSwarm ("jveframe", "v.0.5.1", "pietro.terna@unito.it", args).

La classe VEFrameObserverSwarm ha subito, invece, delle modifiche necessarie per introdurre la gestione del tempo in *tick,* ossia come sottomultipli del ciclo considerato, in particolare per permettere agli oggetti grafici di gestire la nuova suddivisione del tempo e per operare le registrazioni della contabilità solo quando necessarie, visto che alcune hanno bisogno di essere registrate una sola volta a ciclo, mentre altre più volte.

Le due variabili che seguono servono per assegnare gli eventi di tipo GUI che dovranno verificarsi una volta attivato lo schedule, a differenza di prima abbiamo due variabili di tipo ActionGroup

```
public ActionGroup displayActions1, displayActions2;
        displayActions1 = new ActionGroupImpl (getZone());
        displayActions2 = new ActionGroupImpl (getZone());
```

Stabiliamo di assegnare alla prima variabile di tipo *ActionGroup* l'aggiornamento dei grafici generali

```
        displayActions1.createActionTo$message      (waitingListGraph
,SwarmUtils. getSelector (waitingListGraph, "step"));

        if(vEFrameModelSwarm.getUseWarehouses())
           displayActions1.createActionTo$message(
                        warehouseGraph,

SwarmUtils.getSelector(warehouseGraph,"step"));

        displayActions1.createActionTo$message
           (totalTimeLengthGraph,
            SwarmUtils.getSelector(totalTimeLengthGraph, "step"));

        displayActions1.createActionTo$message
           (totalTimeLengthDataFile,
            SwarmUtils.getSelector(totalTimeLengthGraph, "step"));

        displayActions1.createActionTo$message
           (ptHistogram,
            SwarmUtils.getSelector(ptHistogram, "addPoints"));
```

```
displayActions1.createActionTo$message
    (totalDailyCostFile,
     SwarmUtils.getSelector(totalDailyCostFile, "step"));

displayActions1.createActionTo$message
    (totalCostFile,
     SwarmUtils.getSelector(totalCostFile, "step"));

displayActions1.createActionTo$message
    (totalRevenueFile,
     SwarmUtils.getSelector(totalRevenueFile, "step"));

displayActions1.createActionTo$message
    (dailyRevenueFile,
     SwarmUtils.getSelector(dailyRevenueFile, "step"));

displayActions1.createActionTo$message
    (dailySemimanufacturedOrderRevenueFile,

SwarmUtils.getSelector(dailySemimanufacturedOrderRevenueFile, "step"));

if(vEFrameModelSwarm.getUseWarehouses()){
    displayActions1.createActionTo$message
        (totalInventoryFinancialCostFile,

SwarmUtils.getSelector(totalInventoryFinancialCostFile, "step"));
}
```

```
if(vEFrameModelSwarm.getUseWarehouses()){
    displayActions1.createActionTo$message
        (dailyStoredComponentValueFile,

SwarmUtils.getSelector(dailyStoredComponentValueFile, "step"));
}

displayActions1.createActionTo$message
    (dailySemimanufacturedOrderCostFile,

SwarmUtils.getSelector(dailySemimanufacturedOrderCostFile, "step"));

displayActions1.createActionTo$message
    (finishedOrderCostFile,
     SwarmUtils.getSelector(finishedOrderCostFile, "step"));

displayActions1.createActionTo$message
    (benefitFile,
     SwarmUtils.getSelector(benefitFile, "step"));

displayActions1.createActionTo$message
    (benefitGraph,
     SwarmUtils.getSelector(benefitGraph, "step"));
```

mentre alla seconda è assegnato il compito sia di contenere l'azione di aggiornamento del display delle sonde…

displayActions2.createActionTo$message(Globals.env.probeDisplayManager,SwarmUtils.
getSelector(Globals.env.probeDisplayManager,"update"));

…sia di contenere l'azione necessaria ad aggiornare l'intera interfaccia

displayActions2.createActionTo$message(getActionCache(),SwarmUtils.getSelector(get ActionCache(),"doTkEvents")).

Lo schedule, che sinora era attivato in un solo momento, è qui ripetuto con un ciclo iterativo, necessario per introdurre i tick nei quali può essere suddiviso il *"giorno"*. Un ciclo condizionale impone l'inserimento nello *Schedule* dell'evento *displayAction1,* sopra decritto, alla fine di ogni ciclo, garantendo l'aggiornamento dei file relativi alla contabilità e dei grafici una sola volta al giorno

```
for (i=0;i<displayFrequency;i++)
  {
      if(i==displayFrequency-1)
         displaySchedule.at$createAction (i, displayActions1);
      displaySchedule.at$createAction (i, displayActions2); }.
```

La classe VEFrameModelSwarm viene modificata in particolare per l'esigenza di introdurre i singoli tick nel *"giorno"*

public int ticksInADay;

Occorre anche introdurre, per poter assegnare un valore di default e per assegnare un valore di parametro al momento della generazione di una istanza della classe *OrderGenerator.java*, la variabile

public int maxStepLength;

L'ultima variabile che si aggiunge è del tipo *ActionGroup*

public ActionGroup modelActions1, modelActions2, modelActions3;

il motivo della separazione del *modelActions,* della versione precedente, in tre parti è dovuto alla necessità di introdurre una gestione delle azioni non identicamente distribuita, ma che permettesse di separare la gestione contabile, dalla gestione produttiva.

```
modelActions1 = new ActionGroupImpl (getZone ());
modelActions2 = new ActionGroupImpl (getZone ());
modelActions3 = new ActionGroupImpl (getZone ());
```

L'obiettivo è stato di iniziare il ciclo con l'azzeramento dei costi giornalieri,

```
modelActions1.createActionForEach$message(unitList,
                    SwarmUtils.getSelector("Unit","unitStep0"));
```

effettuare tutti i passi di effettuazione di un passo della ricetta, ricevimento ordine, propagazione delle news, contabilizzazione dei singoli costi per ogni *tick,*

```
modelActions2.createActionForEach$message(unitList,
                    SwarmUtils.getSelector("Unit","unitStep1"));
modelActions2.createActionTo$message(orderGenerator,
                SwarmUtils.getSelector(orderGenerator,
                        "createRandomOrderWithNSteps"));
modelActions2.createActionForEach$message(unitList,
                    SwarmUtils.getSelector("Unit","unitStep2"));
modelActions2.createActionForEach$message(unitList,
                SwarmUtils.getSelector("Unit","unitStep3"));
```

contabilizzazione dei costi e dei ricavi giornalieri

```
modelActions3.createActionForEach$message(unitList,
                    SwarmUtils.getSelector("Unit","unitStep4"));

modelSchedule = new ScheduleImpl (getZone (), ticksInADay);

modelSchedule.at$createAction (0, modelActions1);
for (i=0;i<ticksInADay;i++)     modelSchedule.at$createAction
(i, modelActions2);
modelSchedule.at$createAction                (ticksInADay-1,
modelActions3).
```

Le ultime modifiche apportate conseguenti l'introduzione del tempo sono state apportate alla classe *Unit.java*. Anzitutto il richiamo del metodo

```
public void addTickVariableCosts(){
totalTickVariableCosts += variableCosts;}
```

per registrare i costi di ogni *tick*, aggiunto in coda alle procedure del metodo *unitStep1().*

Molto importante è la modifica apportata per assegnare alla unit il passo successivo della ricetta, nel caso si trattasse di una fase di lavorazione ripetuta. In questo modo si utilizza un metodo di gestione del magazzino di tipo *"First in First out" (FiFo),* anziché *"Last in First out" (LiFo),* riuscendo così a riassegnare alla medesima unità il passo di produzione, eventualmente duplicato in conseguenza dell'introduzione del tempo nelle ricette.

```
if(getProductionPhase()==firstOrder.getNextStep())waitingList.
addFirst(firstOrder);
else dailyProductionList.addLast(firstOrder);}
```

L'introduzione del metodo *unitStep4()* serve per impostare i costi ed i ricavi totali giornalieri

```
public void unitStep4 (){
setTotalCosts();
setTotalSaleRevenues();}

public void setTotalDailyCosts(){
totalDailyCosts += fixedCosts + totalTickVariableCosts;}
```

Sviluppi nella formalizzazione del modello di impresa virtuale

Successive astrazioni del comportamento tenuto dagli agenti da simulare ha è reso necessario un aggiornamento utile

L'unica modifica apportata al programma *StartVEFrame.java* è relativa all'indicazione della versione

Globals.env.initSwarm ("jveframe", "v.0.6.0", "pietro.terna@unito.it",args)

Per poter gestire la pluralità di istanti all'interno del ciclo è stato inserito, nella classe VEFrameObserverSwarm un controllo che segnali lo sfasamento del calcolo dei cicli rispetto ai *tick* ogni volta che la variabile displayFrequency sia diversa da 1 oppure non sia un multiplo del numero dei tick presenti nel giorno.

```
if (! (1==displayFrequency || 0==displayFrequency%vEFrame ModelSwarm.getTicks InADay()) ){
        System.out.println("Use displayFrequency == 1 or");
        System.out.println(" displayFrequency == k*ticksInADay, with k integer.");
        System.exit(0);}
```

Si avverte l'utente di correggere il valore impostato per la variabile *displayFrequency* come multiplo intero della variabile *ticksInADay*, evitando così sfasamenti nelle registrazioni contabili.

Invece nel caso della classe *VEFrameModelSwarm.java*, Innanzitutto si definisce una variabile della classe *UnitParameters.java*,

che è la classe che sostituisce *AccountingParameters.java* per la lettura dei dati relativi alle unità

public UnitParameters unitParameters;

treamite il comando *new* si crea una istanza della classe *UnitParameters.java*,

unitParameters = new UnitParameters(getZone(), totalUnitNumber);

si leggono i costi fissi e variabili e li si assegnano ad un vettore tramite il richiamo al metodo

unitParameters.readUnitData();

facente sempre parte della medesima classe.

Infine si crea un nuovo metodo di tipo get per ottenere il numero di tick impostati in ogni ciclo

public int getTicksInADay(){
return ticksInADay;}

Il cambiamento apportato alla classe InformationRuleMAster.java è relativo alla struttura della directory contenente il file con le informazioni circa il flusso delle news

data = new BufferedReader(new FileReader("unitData/informationFlow Matrix.txt")).

La classe ExcelReader apporta un notevole contributo all'integrazione fra più strumenti applicativi. La creazione di questa classe (Sonnessa, 2002) permette di leggere i dati presenti in fogli elettronici, sia sequenzialmente sia specificando la posizione della cella di cui si è interessati.

Il comando *import* richiama il package *xlrd.jar* (Andy Khan, 2002), libreria che fornisce una libreria per la lettura da file tipo worksheet.

```
import xlrd.*;
```

Tramite l'uso di questa classe si facilita la lettura dei fogli di lavoro perché questi possono essere letti in maniera sequenziale, come fossero file di solo testo.

Il seguente costruttore apre il file di tipo *worksheet* trovato nella stessa directory e stabilisce che sia quello che dovrà essere letto.

```
public ExcelReader(String fileName){
        openWorkbook(fileName);
```

In caso vi fossero delle celle vuote le salta senza riportare il valore

```
        emptySkip = true;
        currentSheet = currentWorkbook.getSheet(0);
        updateSheetValues();}
```

Il secondo costruttore apre il foglio di lavoro e lo prepara alla lettura, nel caso in cui la variabile *skipEmptyCells* fosse vera salterebbe le celle vuote, mentre restituirebbe 0 se la variabile fosse falsa.

```
public ExcelReader(String fileName, boolean skipEmptyCells){
openWorkbook(fileName);
emptySkip = skipEmptyCells;
currentSheet = currentWorkbook.getSheet(0);
updateSheetValues();}
```

Il terzo costruttore permette di selezionare il nome del foglio di lavoro, operando come il precedente per ciò che riguarda il restodei parametri. Se il nome del foglio di lavoro è sbagliato restituisce un messaggio di errore.

```
public ExcelReader(String fileName, boolean skipEmptyCells,
String sheetName){
openWorkbook(fileName);
emptySkip = skipEmptyCells;
currentSheet = currentWorkbook.getSheet(sheetName);

if (currentSheet == null){
System.err.println("ExcelReader Error: in '" + fileName + "' the
worksheet '" + sheetName + "' was not found!");
System.exit(0);}
updateSheetValues();}
```

Il seguente metodo ha il compito di leggere il foglio di lavoro. Sfrutta un blocco *try – catch* per segnalare l'errore, in caso non trovi il file

```
private void openWorkbook(String fileName){
try{currentWorkbook = Workbook.getWorkbook(new File(fileName));}
catch (ExcelReadException e){System.err.println("Trying to read '" + fileName + "': " + e.toString());}
catch (Throwable t){
System.out.println(t.toString());
t.printStackTrace();}}
```

Controlla che il foglio di nome *"sheetName"* sia presente nel documento.

```
public boolean existsSheetWithName(String sheetName){
return (currentWorkbook.getSheet(sheetName) != null);}
```

Cambia il foglio corrente sul foglio di nome *"sheetName"*, ma se il foglio di destinazione non è presente il programma termina.

```
public void selectSheet(String sheetName){
currentSheet = currentWorkbook.getSheet(sheetName);
if (currentSheet == null){
System.err.println("ExcelReader Error: the worksheet '" + sheetName + "' was not found!");
System.exit(0);}
```

updateSheetValues();}

Restituisce il foglio corrente

public Sheet getCurrentSheet(){
return currentSheet;}

I metodi seguenti hanno il compito di leggere in una determinata cella del foglio di lavoro che si sta considerando, per ottenere il valore in essa contenuta sotto forma di Stringa, mentre con i due metodi successivi convertono il valore in essa contenuta in un valore *intero* oppure in un valore di tipo *double*.

Stesso comportamento hanno i tre metodi successivi, i quali però non si possono riferire ad una cella utilizzando le coordinate, ma operano una lettura in sequen.za

public String getStrValue(int atRow, int atCol)
public int getIntValue(int atRow, int atCol)
public double getDblValue(int atRow, int atCol)
public String getStrValue()
public int getIntValue()
public double getDblValue()

I due seguenti metodi verificano rispettivamente che la cella indicata alle coordinate, o quella successiva all'attuale, contenga un valore stringa ed in caso affermativo restituisce il valore *true*, altrimenti restituisce il valore *false*.

public boolean checkForLabelCell(int atRow, int atCol)

public boolean checkForLabelCell()

Infine sono presenti due metodi, rivelatisi estremamente utili, i quali segnalano il raggiungimento dell'ultima cella non vuota, permettendo così di terminare la lettura dei fogli o di ricominciarla.

public boolean eof()

public boolean eol()

La classe MyReader.java che è introdotta in questa versione è una classe di tipo statico il cui compito è controllare se vi siano delle nuove righe da leggere nei programmi *VEFrameModelSwarm.java* oppure in *UnitParameters.java*, leggerle e registrare in un buffer momentaneo.

Le variazioni principali introdotte col passaggio dalla classe *AccountingParameters.java* alla *UnitParamenters.java*, sono rappresentate dalla presenza delle unità speciali al fianco delle unità semplici e dall'inserimento di tre livelli di decisione nell'uso dei magazzini uno dei quali può essere definito proprio tramite l'impostazione di valori in file che verranno gestiti dall'interno di questa classe.

Per ciò che riguarda la distinzione fra unità si definiscono semplici le unità in grado effettuare una sola lavorazione, esprimendosi in termini dell'impresa meccanica presa in considerazione, mentre l'unità speciale è in grado di effettuare più lavorazioni, le quali sono definite in un foglio di lavoro di tipo Excel insieme a costi fissi e variabili ed alla decisione riguardante l'utilizzo del magazzino.

All'interno della classe Unit.java la variazione di rilievo riguarda la possibilità che un passo di ricetta con numero identificativo di lavorazione identico al precedente venga messo in lavorazione secondo il metodo Lifo, piuttosto che col metodo Fifo, rendendo così possibile la lavorazione consequenziale di passi di una ricetta che richiedono più tempo della singola unità per essere effettuati.

Le modifiche alle versioni successive sono conseguenza della necessità di evolvere le informazioni che le ricette sono in grado di portare con sé e possono schematizzarsi come segue:

1. ricetta di partenza, consiste in una sequenza di numeri rappresentativi delle attività da svolgere per giungere al prodotto finito;

1	2	3	12	6

2. come detto avviene l'introduzione per considerare tempi diversi di lavorazione dei singoli passi. Cambia anche il metodo di assegnazione delle code agli ordini, infatti, essendo necessario effettuare i passi con ugual numero tutti in fila, occorre impostare un criterio *Last In First Out (LIFO)*;

tempi

3

passi

5	7

ricetta finale

5	s	3	7	ecc…

ricetta interna

5	5	5	7	ecc…

3. Per soddisfare la necessità di avere a disposizione sottoforniture per i singoli passi delle ricette che lo richiedano (situazione molto frequente in VIR ove ogni lavorazione necessita del prodotto della lavorazione precedente) si è introdotto un sistema di procurement. Ogni fase di lavorazione che richieda una sottofornitura dovrà contenere tale informazione che verrà interpretata dall'unità che a sua volta farà richiesta ad un *procurement* che dovrebbe essere in grado di soddisfare tale richiesta. La disponibilità sarà reperibile presso le EndUnit, unità con lo scopo di contenere ricette rappresentative di componenti o semilavorati;

ricette con procurement

p	1	100	5	s

ricette intermedie

-1	1	100	5	ecc…

4. L'ultima evoluzione delle ricette ha consistito nell'introduzione della possibilità di operare una scelta circa quale ricetta utilizzare nel caso in cui vi fosse l'opportunità di giungere allo stesso prodoto finito, seguendo strade diverse , ma non completamente. La medesima ricetta offre più soluzioni per ottenere lo stesso risultato;

ricette con criterio or

||	1	5	S	1
||	2	6	S	1
||	0	E	100	;

L'indicazione riportata significa che per ottenere il semilavorato che

verrà archiviato nella endUnit 100, è possibile selezionare due strade indicate da ||1 e || 2, mentre ||0 indica il termine delle alternative.

Le classi OrderDistiler.java e Recipe.java

Per poter interpretare tutti i dati relativi alle sequenze di lavorazioni e per avviare la produzione nei tempi corretti è stato necessario creare una classe, che in gergo è definita *parser,* in grado di analizzare e interpretare il linguaggio adottato per scrivere le sequenze contenute nelle ricette.

La struttura delprogramma è la seguente::

1. anzitutto si apre il file *recipe.xls* contenente le ricette;
2. si effettua una prima lettura delle singole ricette perdeterminarne nome, codice e lunghezza;
3. si crea un oggetto della classe *Recipe.java* al quale tali parametri sono passati;
4. si effettua nuovamente la lettura del file contenente le ricette per assegnare i contenuti;
5. quando la simulazione è avviata una delle attività dello *Schedule* è quella di richiamare il metodo *distill()* il quale ha il compito di *leggere*, sul file *orderSequences.xls* la struttura della programmazione della produzione così come l'abbiamo desunta dall'attività di formalizzazioe della conoscenza svolta sulla Vir SpA;
6. lo stesso programma provvede a crare un oggetto della classe *Order.java* a cui passare la ricetta che dovrà essere elaborata.

#	LAVORAZIONI GENERICHE																										
	acquisto panettone	000001	100001	s	17200	1	500000	s	1	,																	
	stampaggio	000002	p	1	1	27	s	1	100002	s	172800	,	500000	s	2	:											
	sabbiatura interna	000003	p	1	2	28	s	1	100003	s	1	,	1	s	3	:											
	rifoltura esterna	000004	p	1	3	29	s	1	100004	s	40000	,	20000	s	4	:											
	sabbiatura esterna	000005	p	1	4	30	s	1	100005	s	40000	,	20000	s	5	:											
	pre assemblaggio esterno	000006	100006	s	43200	1	5000	s	6	,																	
	acquisto barattolo	000007	100007	s	17280	1	500000	s	7	:																	
	stampaggio sfera	000008	p	1	7	31	s	1	100001	s	172800	,	500000	s	11	:											
+	COMPONENTI ACQUISTATE ALL'ESTERNO																										
#	per valvola sfera																										
	bullone	000009	100014	s	172800	1	200000	s	14	,																	
	lava	000010	100015	s	172800	1	10000	s	15	:																	
	prima	000011	100016	s	262800	1	80000	s	16	:																	
	premistoppa	000012	100017	s	262800	1	175000	s	17	:																	
	guarnizione premistoppa	000013	100018	s	262800	1	175000	s	18	:																	
	seggio	000014	100019	s	172800	1	115000	s	19	,																	
	rivestimento sfera	000015	100020	s	172800	1	10000	s	20	,																	
+	per sfera finita																										
	asta	000016	100021	s	262800	1	10000	s	21	:																	
	dado per premistoppa	000017	100022	s	172800	1	37500	s	22	:																	
	guarnizione	000018	100023	s	262800	1	25000	s	23	,																	
	ghiera	000019	100024	s	262800	1	22400	s	24	:																	
	volantino	000020	100025	s	172800	1	2000	s	25	:																	
	dado	000021	100026	s	172800	1	37400	s	26	:																	
#	[illegible]																										
+	per valvola sfera grande																										
	manico tappo valvola	000022	p	1	4	100	s	8	100007	s	172800	,	10000	s	100	:											
	manico tappo grande	000023	p	1	4	101	s	16	100008	s	172800	,	10000	s	101	:											
	corpo piccolo	000024	p	1	5	102	s	5	100009	s	172800	,	10000	s	102	:											
	corpo grande	000025	p	1	5	103	s	4	100008	s	172800	,	10000	s	103	:											
	sfera piccola	000026	p	1	7	104	s	5	100000	s	172800	,	10000	s	104	,											
	sfera grande	000027	p	1	7	105	s	16	100009	s	172800	,	10000	s	105	:											
	anello grande	000028	p	1	4	106	s	10						s	106	:											
	anello piccolo	000029	p	1	5	107	s	5						s	107	:											
	vite grande	000030	p	1	4	108	s	7						s	108	:											
	vite piccola	000031	p	1	5	109	s	4						s	109	,											
	corpo macchina sfera grande	000032	p	1	4	110	s	17						s	110	:											
	corpo macchina sfera piccolo	000033	p	1	5	111	s	5						s	111	:											
+	MONTAGGI																										
#	per valvola sfera grande																										
	montaggio valvola grande	000034	ll	1	22	s		s	11	2	15	10	7	10	12	10	20	1101	1103	1105	10001	s		17			
			ll	2	23	s			p	2	15	1001	500	s	2		p	2	19	1102	501	s		4			
											p	4	500	501	1105	18	502	s	5								
											s	6	502	12	15	16	17	20	10001	s	5	l	0	e	11001	;	
	tabellaggio valvola grande	000035	p					1	11001	3307	5	3	:														
	montaggio valvola piccola	000036	ll	1	24	s		s	11	2	15	10	7	10	12	10	20	1101	1103	1104	10002	s		10			
			ll	2	25	s			p	2	15	1002	503	s	2		p	2	19	1102	504	s		3			
											p	4	503	504	1104	18	505	s	7								
											s	6	505	12	15	16	17	20	10002	s	9	l	0	e	11002	;	
	tabellaggio valvola piccola	000037	p					1	11002	3308	5	2	:														
	montaggio saracinesca grande	000038	ll	1	30	s		s	9	2	22	22	24	25	26	1100	1102	1110	10003	s	32						
			ll	2	31	s			p	2	21	22	27	24	1106	1100	1110	506	s	15							
									p	3	25	26	506						10003	s	13	l	0	e	11003	;	
	tabellaggio saracinesca grande	000039	p					1	11003	3309	5	9	:														
	montaggio saracinesca piccola	000040	ll	1	28	s		s	9	2	22	22	24	25	26	1107	1109	1111	10004	s	13						
			ll	2	29	s			p	2	21	22	29	24	1107	1100	1111	507	s	6							
									p	3	25	26	507						10004	s	7	l	0	e	11004	;	
	tabellaggio saracinesca piccola	00004	p					1	11004	10000	5	2	:														

La classe *Recipe.java* è, fondamentalmente una classe di tipo *wrapper* utilizzata per convertire il tipo di dati *array*, che le viene passato in fase di creazione di un oggetto, in un tipo di dati *recipe* che può essere facilmente gestito dalle liste di Swarm.

Conclusioni

Ulteriori evoluzioni potranno seguire, una fra tante è la possibilita di svolgere due passi della ricetta contemporaneamente o la possibilità di introdurre unità in grado di gestire la condivisione delle risorse, siano esse umane o di materiali e altro ancora.

Un problema che però si è riscontrato, ed è fortemente limitante ai fini di un'ottimo utilizzo dello strumento della simulazione è che ogni ulteriore funzionalità crea richieste di risorse interne della macchina su cui si fa eseguire il programma. Problemi che verranno tratta ti nei dettagli nel prossimo capitolo riguardante proprio l'analisi dei risultati.

Bibliografia

Terna, P. (2002), *Simulazione ad agenti in contesti di impresa*, dattiloscritto