UNIVERISTÁ DEGLI STUDI DI TORINO

FACOLTÁ DI ECONOMIA

CORSO DI LAUREA IN ECONOMIA E COMMERCIO

TESI DI LAUREA

GENERALIZZAZIONE DEL MODELLO JAVASWARM DI SIMULAZIONE DI IMPRESA PER L'APPLICAZIONE AD UN CASO CONCRETO

Relatore:
Prof. PIETRO TERNA

Correlatore:
Prof. SERGIO MARGARITA

Candidato:
ANTONELLA BORRA

Anno Accademico 2001 - 2002

Desidero ringraziare l'ing. Savino Rizzio, fondatore della Vir S.p.a.,ed il figlio ing. Giovannibattista Rizzio, che hanno consentito l'accesso ai dati aziendali ed alle informazioni che sono state necessarie al completamento di questo lavoro.

Ringrazio inoltre l'ing. Gervasini, che ha fornito gli elementi relativi all'organizzazione dell'impresa da un punto di vista gestionale ed il sig. Camana, che ha illustrato le problematiche tecniche di gestione della produzione e dei macchinari e ha guidato una visita dello stabilimento.

Indice degli argomenti

Introduzione

Parte prima - L'evoluzione della visione di impresa

7 Capitolo 1 L'impresa neoclassica e la visione Austriaca

14 Capitolo 2 La definizione di impresa nell'economia odierna

14 2.1 Le regole dell'economia dell'informazione

18 2.2 Old e new economy si intrecciano: il Business to Business

Parte seconda - L'impresa virtuale: modelli di simulazione

32 Capitolo 3 La simulazione: vantaggi e problemi

39 3.1 La realizzazione di *micromondi*: l'utilizzo congiunto di modelli contabili e *system dynamics*

46 3.2 Simulazione ad agenti e programmazione ad oggetti: il progetto Swarm

53 Capitolo 4 L'azienda virtuale industriale: il modello NIIIP

53 4.1 Il progetto NIIIP

55 4.2 L'architettura NIIIP

61 4.2.1 Il paradigma XML

66 Capitolo 5 Java Virtual Enterprise (jVE): un *frame* in Swarm

66 5.1 La struttura

71 5.2 Il codice

86 Capitolo 6 Il modello di impresa virtuale VIR

86 6.1 Descrizione della realtà Vir

114 6.2 Astrazione per jVE

115 6.2.1 Le ricette (What to do - WD)

130 6.2.2 Le unità (which is Doing What - DW)

135 6.2.3 Le modifiche al codice

159 6.2.4 L'analisi dei risultati

170 6.2.4.1Nuove soluzioni tecnologiche

Parte terza – La notazione UML

182 Capitolo 7 UML e extreme programming (XP)

182 7.1 Extreme programming (XP)

188 7.2 Unified Modelling Language

199 7.3 Tools che supportano la notazione UML

201 7.4 Un esempio di utilizzo: Poseidon for UML

208 Bibliografia

214 APPENDICE A- Il codice jVE-VIR

247 APPENDICE B- Analisi del corretto funzionamento della versione Jveframe-0.7.0.a

Introduzione

Il complesso dei cambiamenti provocati dalla diffusione su larga scala delle tecnologie dell'informazione e della comunicazione (ITC) impone all'impresa la necessità di una riorganizzazione.
L'elemento distintivo del valore di un'azienda diventa la capacità del management di governare catene del valore articolate e dinamiche e di modificare le strategie produttive e distributive per sfruttare le opportunità offerte dal mercato globale.
Esplorare una realtà aziendale attraverso un modello di simulazione consente di migliorare la comprensione dei meccanismi interni all'azienda, rappresentando l'interdipendenza tra le entità che la compongono con diversi livelli di dettaglio, corrispondenti ai diversi livelli di astrazione possibili.
L'impresa virtuale può essere oggetto di indagine e di sperimentazione per valutare, come in un laboratorio, gli effetti che decisioni di innovazioni di processo potrebbero avere sulla sua struttura organizzativa.

Con queste premesse, descriviamo brevemente l'organizzazione, il contenuto e la metodologia utilizzata.

La Parte prima tratta dell'evoluzione della visione di impresa, da un punto di vista teorico, valutando gli strumenti concettuali e metodologici dell'economia neoclassica attraverso la prospettiva della scuola Austriaca. Successivamente da un punto di vista applicativo, prendendo in considerazione le nuove regole dell'economia dell'informazione e gli effetti di questo nuovo paradigma tecnologico sull'organizzazione delle imprese e dei settori.

La Parte seconda tratta dell'impresa virtuale e dei modelli di simulazione.
Il capitolo 3 presenta un'analisi dei vantaggi e delle limitazioni che possono derivare dallo studio della realtà attraverso la simulazione che riproduce nel computer i fenomeni osservati. Si esaminano i modelli dinamici ed il progetto Swarm.
Il capitolo 4 analizza il modello di azienda virtuale industriale NIIIP con l'intento di trarne preziose indicazioni applicabili alla simulazione di impresa oggetto di questo lavoro.
Il capitolo 5 descrive la struttura ed il funzionamento del modello javaSwarm di impresa virtuale utilizzato come base per il modello jVE-Vir di simulazione di un caso concreto.
Il capitolo 6 tratta del processo di generalizzazione del modello jVE per la simulazione del caso aziendale Vir, di cui si descrive la realtà aziendale e la realtà simulata.

La parte terza presenta la notazione UML e valuta il contributo che essa può fornire alla nuova metodologia di sviluppo di software XP.

Parte prima

L’evoluzione della visione di impresa

Capitolo 1 - L'impresa neoclassica e la visione Austriaca

Attraverso la prospettiva della scuola Austriaca ci si propone di valutare gli strumenti concettuali e metodologici dell'economia neoclassica, ponendone in evidenza i limiti positivi e normativi.
Nell'analizzare tale impostazione, si intende valutare se di fronte alle trasformazioni imposte dai recenti cambiamenti tecnologici il paradigma neoclassico sia in grado di interpretare i complessi meccanismi che regolano la vita della moderna impresa capitalistica e che sono oggetto di ricerca e di simulazioni nel campo delle scienze sociali.

Malgrado la visione neoclassica della microeconomia abbia rivestito un ruolo di primo piano nell'analisi economica di tutto il XX secolo, nella storia del pensiero economico si individua come periodo dell'economia neoclassica una fase che inizia intorno al 1871, anno in cui vengono pubblicate le opere di Jevons e Menger, rispettivamente "La Teoria dell'Economia Politica", e "I Principi di Economia Politica", fino agli anni trenta in cui comincia una fase di critica e ripensamento, soprattutto in seguito ai lavori di John Maynard Keynes, e dei suoi allievi.
L'impostazione neoclassica propone una concezione meccanicistica dell'economia secondo la quale l'impresa risolve un problema di ottimizzazione valutando la quantità dei propri input e dei propri output per massimizzare il profitto.

I vincoli di tale problema di ottimizzazione sono rappresentati dalle condizioni del mercato in cui l'impresa opera e dalle tecnologie disponibili per i processi produttivi.

L'impresa neoclassica (Colombatto 2001) si compone di un insieme di fattori organizzati per produrre e commercializzare un bene che possa essere venduto almeno a un prezzo pari alla somma delle remunerazione corrisposte e delle risorse impiegate.

La definizione di una funzione di produzione permette all'impresa di lavorare secondo meccanismi noti e prevedibili e di produrre sempre in modo tecnologicamente efficiente. Fissato un livello di output viene scelta la combinazione di input che minimizza i costi di produzione nel breve o nel lungo periodo.

Un'analisi statica dello schema conduce alla conclusione che l'impresa raggiunge il livello di massimo profitto se produce nel punto in cui i profitti marginali di tutti gli input sono nulli, vale a dire nel punto in cui i ricavi marginali derivanti dalla variazione della quantità di un input sono annullati dai rispettivi costi marginali di acquisto.

Sono annoverabili tra i fattori di produzione la capacità di organizzare le risorse in modo adeguato e l'attività di ricerca e sviluppo.

Una volta definiti i fattori produttivi disponibili, la singola impresa è replicabile più volte in relazione con le caratteristiche della domanda e il tipo di tecnologia produttiva utilizzata.

La perfezione propria della visione neoclassica, di cui l'impresa costituisce un esempio particolarmente calzante, pare potere escludere la presenza di durature asimmetrie informative e di incertezza.

L'imprenditore neoclassico risulta, in realtà, soggetto a una duplice fonte di rischio derivante dalla carenza di informazioni sulle variabili esterne e dal processo di selezione dei fattori produttivi.

I valori assunti dalle variabili esogene condizionano la scelta delle funzioni di produzione, l'attività di investimento, la scelta delle tecniche, la dimensione ottima di impresa.
L'imprenditore è considerato un "funzionario organizzatore" con il compito di selezionare la funzione di produzione più adatta per gli obiettivi assegnati dal modello e verificare che i fattori siano combinati coerentemente.
La figura imprenditoriale, nella concezione neoclassica, non è quindi animata da uno spirito innovativo orientato alla ricerca di nuove opportunità, nuovi bisogni da soddisfare.
Come si è detto, il delineare una funzione di produzione nota a tutti rende l'impresa replicabile; la perfezione del mondo potrebbe rendere impossibile la differenziazione tra le imprese.
Nella prospettiva di breve periodo è possibile ipotizzare la presenza di imprese differenziate compatibile con la visione neoclassica dell'economia a patto che si verifichi una delle due condizioni seguenti:

- gli imprenditori sono tra loro differenziati e hanno percezioni diverse in merito al beneficio atteso generato dall'acquisto dell'informazione marginale;
- gli imprenditori hanno accesso a insiemi di informazioni distinti, a parità di spesa, che determinano la costruzione di una diversa funzione di produzione.

Nel lungo periodo il sistema sarebbe riequilibrato dagli operatori che, per imitazione dei migliori, correggeranno i propri errori.
Secondo questa concezione il lungo periodo è da intendersi non come un intervallo temporale più esteso, ma come una somma di brevi periodi in successione durante i quali, se gli imprenditori sono tra loro differenziati, possono realizzarsi profitti positivi anche in un mercato concorrenziale.

Il concetto di equilibrio neoclassico non prevede la presenza di profitti duraturi, il loro perdurare è segnalato come un fallimento del mercato, un comportamento strategico inefficiente che richiede la necessità di un intervento pubblico per conseguire la configurazione desiderata.
L'equilibrio concorrenziale si realizza in un contesto di lungo periodo stazionario che può essere turbato soltanto da *shock* esterni imprevisti.
L'azione imprenditoriale, nella concezione neoclassica, si esaurisce nella scelta fra le diverse funzioni di produzione, segue la traccia già percorsa da altri, come in un modello predeterminato.
Gli incrementi di benessere, in un mondo con tecnologia costante, possono derivare principalmente da tre fattori:

- lo scambio efficiente;
- l'impiego ottimo dei fattori disponibili;
- il miglioramento dell'assegnazione dei diritti di proprietà fra imprese note.

La modellistica analizzata si rivela, però, scarsamente aderente alla realtà e i suoi suggerimenti normativi risultano solitamente irrilevanti.
La scuola di *public choice* tende a sottolineare come l'intervento pubblico non necessariamente persegua il benessere collettivo, ma al contrario massimizzi il benessere del decisore che opera secondo le modalità neoclassiche, rivelandosi più dannoso dei fallimenti del mercato.
Lo schema neoclassico è stato posto in discussione anche sotto il profilo positivo, la critica più incisiva si è rivelata essere quella Austriaca che ha contestato il concetto neoclassico di economia di mercato, di concorrenza e di equilibrio concorrenziale.
Carl Menger può essere considerato il vero "padre fondatore" della scuola Austriaca dell'economia, che un ruolo tanto rilevante ha avuto nella rinascita novecentesca del pensiero liberale.

Lo scopo della teoria economica, secondo Menger, è quello di ricostruire i sistemi sociali a partire dalle loro singole componenti, primo fra tutte l'individuo, poiché solo questi agisce, desidera e sceglie.
Questo metodo deduttivo e privo di formalizzazioni matematiche è indicato da Joseph Schumpeter con il termine di "individualismo metodologico".
Seguendo il percorso di Menger, Friedrich von Hayek (1948) propose una teoria secondo la quale la civiltà e la società sarebbero il prodotto di azioni individuali non intenzionali, di un ordine spontaneo che deriva dall'interazione di milioni di esseri liberi nel corso della storia.
L'aspetto cruciale del processo economico hayekiano è rappresentato dal cambiamento e non dall'equilibrio.
Motore del cambiamento è la conoscenza dispersa degli individui.
Hayek mostra proprio come, al fine di permettere ad un individuo di acquisire i modelli mentali di un gruppo, a causa della natura tacita della conoscenza, siano fondamentali i processi di imitazione, sovente inconsapevoli, che nascono dall'interazione reciproca.
Von Mises (1949), un altro degli esponenti della scuola Austriaca, interpreta lo studio dell'economia come l'analisi delle possibilità, in condizioni di incertezza, di rallentare i vincoli tipici dell'attività economica, di accrescere la soddisfazione dei consumatori e di creare gli incentivi affinché gli imprenditori si impegnino a superare i vincoli imposti dalle funzioni di produzione esistenti, per sperimentare nuovi processi e nuovi prodotti, per raggiungere nuovi mercati.
Nella visione Austriaca non si esclude a priori la presenza pubblica, ma si valutano con attenzione e cautela quegli incentivi pubblici che non sono giustificati dall'esigenza di garantire le libertà individuali fondamentali e rischierebbero di distorcere l'economia.
Per gli Austriaci non è auspicabile raggiungere l'equilibrio, che congelerebbe il mondo nella sua perfezione, ma è essenziale

individuare i meccanismi attraverso i quali gli squilibri possono correggersi automaticamente e con maggiore rapidità.

Il protagonista di questa economia di mercato è l'imprenditore di Kirzner (1973), lo scopritore per eccellenza, colui che cogliere le opportunità che altri avevano trascurato e rende possibile la descrizione di nuove produzioni, vede realtà già presenti, ma non percepite da altri imprenditori come opportunità di crescita e benessere.

Il profitto diviene la remunerazione dell'abilità dell'imprenditore nel percepire gli errori altrui e le opportunità di miglioramento. La scoperta imprenditoriale precede l'attività aziendale, l'imprenditore kirzneriano è puro, privo di mezzi di produzione.

L'imprenditore è motivato alla scoperta imprenditoriale dall'esistenza di informazione incompleta, distorta o poco costosa che rende possibile la creazione di ricchezza.

Il profitto è la parte appropriabile di questa ricchezza.

La scuola Austriaca riesce così a spiegare esaurientemente la presenza di profitti positivi in assenza di vincoli normativi, a prescindere dai fallimenti del mercato.

La concezione dell'imprenditore di Kirzner si discosta parzialmente da quella della scuola Austriaca in quanto vede l'individuo come fulcro non intenzionale dell'attività economica nel suo insieme.

La casualità delle scoperte attraverso le quali l'imprenditore individua imperfezioni e opportunità di cui non immaginava l'esistenza è il nodo cruciale della questione. Kirzner avanza la teoria dell'"ignoranza inconsapevole", mentre l'ortodossia, anche Austriaca, sostengono la tesi dell'ignoranza razionale.

Se si nega alla funzione imprenditoriale la casualità del processo di scoperta si rischia di considerare l'imprenditore come un fattore

produttivo anomalo: la ricerca sistematica richiede tempo e quindi l'impiego di una risorsa.
Al contrario, accettare l'attributo della casualità significa ammettere che l'imprenditore puro è destinato a non essere remunerato e che la scoperta imprenditoriale non solo è casuale, ma sporadica.
A tale proposito, Kirzner sottolinea come la critica Austriaca alla visione neoclassica riponga le basi nella diversa concezione dell'individuo che esprime la propria vivacità intellettuale attraverso scoperte imprenditoriali.

Obiettivo di questo capitolo è valutare se nell'ambito della teoria economica siano presenti contributi sull'impresa in grado di superare alcuni evidenti limiti della visione tradizionale neoclassica.
In coclusione si può affermare che, nella rappresentazione del mercato, la letteratura economica ha proposto sia una visione dell'impresa come funzione di produzione, sia una visione dell'impresa come individuo/imprenditore in cui lo sviluppo dell'impresa è messo in relazione con quello delle persone che ne fanno parte; motivazioni e capacità dei lavoratori sono influenzati dall'ambiente in cui operano e ne determinano a loro volta le caratteristiche.
Questa seconda rappresentazione, in virtù dell'accento posto sulla dispersione della conoscenza e sull'innovazione, è sicuramente più adeguato a rappresentare alcuni aspetti della nuova economia.
Nell'ambito delle scienze sociali, le simulazioni di imprese virtuali forniscono, a tale proposito, un importante contributo alla teoria economica.

Capitolo 2 – La definizione di impresa nell'economia odierna

Il complesso dei cambiamenti provocati dalla diffusione su larga scala delle tecnologie dell'informazione e della comunicazione (ITC) impone all'impresa la necessità di una riorganizzazione, a prescindere dal settore in cui lavora. Una quota crescente della domanda finale si sposta verso i prodotti legati alla conoscenza e all'informazione e anche i business più tradizionali sono influenzati dai cambiamenti provocati dalle sempre più pervasive nuove tecnologie. L'elemento distintivo del valore di un'azienda diventa la capacità del management di governare catene del valore articolate e dinamiche e di modificare le strategie produttive e distributive per sfruttare le opportunità offerte dal nuovo mercato globale. La diffusione delle ITC pare più rapida tra le grandi imprese e tende a propagarsi nelle piccole-medie imprese soprattutto quando la filiera produttiva è dominata da una grande impresa. Il maggior impatto delle ITC è nei settori *information-intensive* nei quali è importante la capacità di coordinare numerose sub-unità o componenti. Per poter valutare in anticipo gli effetti generati dall'introduzione nella struttura aziendale delle nuove tecnologie può essere utile sperimentarli in una simulazione

2.1 Le regole dell'economia dell'informazione

L'informazione e la conoscenza sono considerate le risorse strategiche di questo nuovo paradigma tecnologico, che affianca alle radicali innovazioni di prodotto e di processo modifiche nell'organizzazione delle imprese e dei settori. L'uso delle ITC appare più facile ed efficace

laddove le attività presentano, già prima dell'informatizzazione, un più elevato grado di formalizzazione in quanto le ITC richiedono la traduzione in informazione strutturata anche di una parte della conoscenza "*tacita*" accumulata nelle organizzazioni. Le nuove conoscenze codificate possono essere facilmente trasferite a lunga distanza e a bassi costi, come suggeriscono Varian e Shapiro (1999), infatti, l'informazione è un prodotto costoso da produrre, ma economico da riprodurre. La codifica delle pratiche organizzative e la standardizzazione determinano la caduta dei costi di coordinamento offrendo importanti opportunità di deverticalizzazione. La disponibilità di informazione in tempi brevi e a costi contenuti influenza la strategia di gestione del prodotto delle imprese ricettrici e produttrici di informazioni, aumenta la trasparenza e la conoscenza comparata dei prezzi e delle condizioni contrattuali, migliora l'efficienza allocativa del mercato, riduce i benefici dell'integrazione verticale e della grande dimensione determinando un più esteso ricorso all'*outsourcing.*

L'incremento di produttività che la "rivoluzione" ITC produce è collegato, in prima approssimazione, alla necessità di elaborare e trasmettere informazione. Come nella tabella 1, nei settori tradizionali (tessile, abbigliamento e cuoio) il guadagno di produttività è probabilmente destinato ad essere inferiore rispetto ad altri comparti in quanto la fase di produzione dei beni non appare essere *information intensive*. Il capitale umano degli occupati in tali settori è, inoltre, caratterizzato da un livello di istruzione relativamente basso e quindi poco adatto alle esigenze delle nuove tecnologie. Un settore che, al contrario, pare poter fortemente beneficiare degli sviluppi delle ITC è quello della meccanica strumentale in cui la diffusione e lo sviluppo di macchinari che incorporino le nuove tecnologie garantisce guadagni di efficienza. In questo caso la forza lavoro, abituata ad operare con

GRADO DI DIFFUSIONE DI ALCUNE TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE NELLE IMPRESE INDUSTRIALI ITALIANE
(anno 2000)

	Spesa per acquisto e manutenzione TIC per 100 addetti (milioni di lire)		PC per 100 addetti		Quota di imprese con computer collegati a Internet		Quota di imprese con sito Internet aziendale		Quota di imprese dotate di una unità organizzativa dedicata alle TIC		Quota di imprese dotate di tecnologia ERP	
	Totale	Distrettuali	Totale	Distrettuali	Totale	Distrettuali	Totale	Distrettuali	Totale	Distrettuali	Totale	Distrettuali
Classe di addetti												
Medio-piccole (50-99)	62	60	31,4	31,2	95,9	96,2	79,7	82,8	25,6	20,4	10,5	13,5
Medie (100-199)	85	98	34,6	30,8	97,3	96,7	83,2	83,8	43,1	40,4	25,3	26,5
Medio-grandi (200-999)	97	101	39,2	35,2	98,4	98,0	84,8	89,2	55,6	47,1	42,7	37,6
Grandi (oltre 1000)	114	99	49,5	41,3	98,9	100,0	92,6	97,3	85,5	82,2	74,9	77,8
Area geografica												
Nord-ovest	95	87	44,4	35,8	97,3	98,0	83,3	86,6	34,7	27,4	22,8	23,0
Nord-est	94	97	32,9	33,5	97,3	96,4	86,7	84,9	40,9	34,2	20,5	22,7
Centro	95	72	41,8	28,5	95,3	94,3	77,9	77,7	29,5	31,7	17,0	16,4
Sud - Isole	57	55	29,3	24,2	94,6	92,2	64,3	79,3	30,5	12,4	12,4	3,8
Settore												
Tessile, abbigliam., pelli e calzature	61	61	25,1	25,2	94,7	94,9	79,0	81,6	36,3	35,7	15,2	15,1
Chimica, gomma e plastica	113	127	57,4	39,1	96,0	94,3	77,3	70,2	45,3	32,0	23,6	20,0
Metalmeccanica	104	108	38,6	36,3	97,2	96,4	84,5	87,3	35,8	29,6	22,8	26,7
Altre manifatturiere	72	60	32,2	30,3	97,8	99,2	81,1	87,0	29,8	25,6	17,1	17,1
Altre industrie in senso stretto	101	182	63,3	105,9	94,4	100,0	72,8	47,6	37,4	47,6	24,8	52,6
Totale	92	88	39,2	33,7	96,7	96,6	81,6	84,2	35,4	30,3	20,1	21,3

Tabella 1: grado di diffusione delle ITC nelle imprese italiane (Trento e Warglier 2001)

macchine automatizzate, lascia supporre che le difficoltà e i costi di adattamento all'innovazione tecnologica siano inferiori a quelli che si potrebbero incontrare nei settori più maturi.

Cambiare la struttura dell'impresa nella prospettiva di poter usufruire dei vantaggi derivanti dalla nuova economia dell'informazione espone, però, l'impresa stessa a rischi e costi di transizione.

Quando i costi associati alla transazione da una tecnologia a un'altra sono rilevanti si dice che gli utenti affrontano un problema di lock-in, ovvero l'impresa si trova bloccata in una situazione di *"vecchia tecnologia"* dalla quale non riesce più ad uscire. Il ciclo di lock-in comincia con la selezione di una marca a cui segue un periodo di valutazione e quindi la fase di consolidamento; nel momento in cui i costi di transizione divengono insostenibili la fase di consolidamento culmina nel lock-in. Il segreto della comprensione della gestione del lock-in, secondo Varian e Shapiro (1999), consisterebbe nella capacità di anticipare il ciclo fino dal suo inizio proiettandosi nel futuro e ragionando a ritroso. Nell'economia dell'informazione i costi di transizione e i lock-in non rappresentano un'eccezione in virtù del continuo cambiamento delle tecnologie che rendono imprevedibile il ciclo di vita del prodotto e difficile la valutazione della lunghezza della sua vita economica.

Un'impresa può scegliere la strategia dell'evoluzione, che offre ai consumatori un approdo graduale alle nuove tecnologie, oppure la strategia della rivoluzione che offre una tecnologia con performance nettamente superiore, ma talvolta a discapito della compatibilità con i sistemi precedenti. In questo senso talvolta è necessario valutare l'opportunità di fare uso di convertitori e di tecnologie di transizione In un passaggio graduale a nuove tecnologie l'impresa deve ragionare in termini di sistema, spesso infatti si produce una sola componente, ma al consumatore è il sistema intero ad interessare.

Nell'economia dell'informazione le dimensioni cruciali sono la posizione di mercato esistente, le capacità tecniche e il controllo sulla proprietà intellettuale. Se lo standard è veramente aperto i consumatori saranno meno preoccupati dall'effetto di lock-in in quanto potranno fare affidamento sul fatto che in futuro il mercato sarà concorrenziale. Uno standard aperto è però un rischio in quanto, se privo di uno sponsor, può essere soggetto al fenomeno della frammentazione. Si dice che uno standard è frammentato quando vengono sviluppate più versioni incompatibili della stessa tecnologia sottoposta allo standard.

Vi sono alcuni aspetti chiave di un mercato di rete che non possono essere trascurati. Il primo luogo l'impresa deve avere il controllo sulla base installata di utenti, deve possedere i diritti sulla proprietà intellettuale, deve avere capacità di innovare e abilità nella produzione, forza nei mercati di prodotti complementari, marchio di fabbrica riconosciuto e reputazione internazionale.

Mentre le vecchie strutture industriali erano caratterizzate dalla presenza di economie di scala, la nuova economia dell'informazione è invece dominata dalla presenza di economie di rete. Nell'economia di rete la combinazione delle economie di scala dal lato della domanda con quelle dal lato dell'offerta genera un forte *feedback* positivo. La popolarità apporta valore aggiunto alle reti, costruire in anticipo sui rivali una posizione di comando permette di sfruttare il *feedback* positivo contro i rivali e consolidare la propria base di utenti, uno fra i beni più preziosi nell'economia dell'informazione, anche attraverso l'offerta di servizi informativi, l'attivazione di programmi di fidelizzazione e lo studio delle aspettative della clientela.

2.2 Old e new economy si intrecciano: il Business to Business

Con l'acrostico B2B (business-to-business) si intendono le transazioni commerciali tra due o più imprese che hanno luogo tramite il supporto di Internet in specifici mercati virtuali.
L'evoluzione del business su Internet può essere articolata in alcune fasi principali.
Intorno alla metà delle anni '90 le imprese hanno cominciato a predisporre siti di presentazione delle caratteristiche dell'attività aziendale principalmente per fini di comunicazione e marketing.
Il 1997 vede l'affermazione delle prime reti Intranet attraverso le quali le aziende tentavano di riorganizzare le attività aziendali in funzione della rete, dell'interazione fra diversi attori dell'organizzazione e della filiera.
La quantità e la qualità delle informazioni messe in rete dalle imprese disponibili per altre imprese e per i clienti finali sono andate crescendo insieme alla maggiore consapevolezza circa le opportunità che Internet può offrire. La fase attuale vede le imprese investire per cercare di utilizzare le opportunità in modo più proficuo. Si sviluppano i settori del commercio elettronico, i sistemi per effettuare transazioni in rete e per offrire servizi, diventano operativi i portali business-to-business e quelli business to consumer (B2C), Internet entra fra le strategie aziendali delle imprese e ne influenza sempre più profondamente i modelli organizzativi.
Come mostrato nella figura 1 il cambiamento nell'organizzazione di impresa diventa via più marcato con l'introduzione di tecnologie più complesse, dal semplice scambio di e-mail all'introduzione di sistemi

ERP (Enterprise Resource Planning), i nuovi sistemi gestionali integrati.

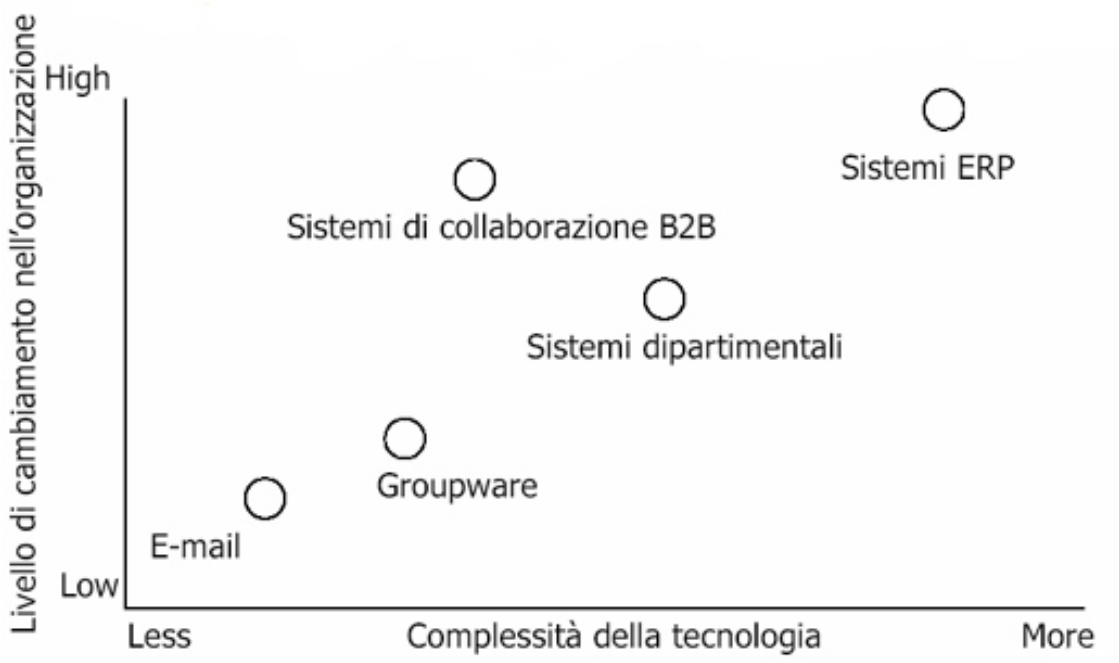


Figura 1: cambiamenti organizzativi e complessità della tecnologia

Il business-to-business, modificando radicalmente il modo con cui si realizzano i rapporti commerciali tra le imprese, rappresenta lo snodo principale attraverso il quale la cosiddetta nuova economia si intreccia con la vecchia.

Sono soprattutto le grandi aziende della cosiddetta old economy, infatti, a conquistare una posizione di rilievo nel panorama di Internet.

Le ragioni di questa tendenza sono molteplici: in primo luogo è necessaria una solida base per poter "spalmare" le perdite, o gli investimenti, che l'avvio dell'e-business comporta; marchi di richiamo e consistenti campagne di marketing e comunicazione permettono di ottenere visibilità e rinsaldare la massa critica a quelle imprese che dispongono già di clienti distribuiti, forti di esperienze di multinazionalità e multisettorialità; orientarsi ad Internet è più facile per un'azienda *"vecchia maniera"* che può destrutturare e riorganizzare

una divisione rispetto ad una piccola azienda che deve agire su tutta la sua realtà.
Il B2B rimane comunque uno strumento privilegiato per le piccole medie imprese che possono competere alla pari con le grandi imprese se, con intelligenza e creatività imprenditoriale, riescono a sfruttare le loro caratteristiche di flessibilità e velocità di interazione con i fornitori nella progettazione e nella produzione di beni.
Pur rimanendo entità separate il grado di cooperazione tra le imprese appartenenti alla filiera può arrivare ad assomigliare a quello che si realizza nell'ambito di un'impresa verticalmente integrata.
L'impatto più significativo del ricorso ai mercati virtuali e quindi il principale effetto dal business-to-business riguarda l'abbattimento dei costi di transazione perché le informazioni vengono gestite in modo più efficiente e standardizzato e le contrattazioni di beni intermedi sono concentrate in mercati geograficamente ampi. Nella definizione di Coase (1960) i costi di transazione derivano dalle seguenti attività:

> *"Al fine di realizzare una transazione di mercato è necessario individuare con chi si vuole trattare, informare la gente che si vuole commerciare e a quali condizioni, condurre negoziazioni per giungere a un accordo, per stendere un contratto e per intraprendere i controlli necessari per assicurarsi che i termini del contratto siano effettivamente rispettati e così via."*

Per meglio comprendere il contributo addizionale del B2B è bene ricordare alcuni sistemi utilizzati dalle imprese prima del suo avvento. Uno tra questi è il cosiddetto *Material Requirement Planning* (MRP) che costituisce un sistema informatizzato di gestione del fabbisogno di input da acquistare in funzione delle necessità produttive. Su tale sistema si innesta il cosiddetto *Electronic Data Interchange* (EDI) che collega direttamente l'acquirente con il singolo fornitore, imponendo la

necessità di predisporre uno specifico software per ciascun singolo collegamento, risultando quindi oneroso e in grado di soddisfare solo una porzione dell'intero ammontare degli acquisti di un'impresa.
Gli scambi tra imprese nell'ambito del B2B non sono più dipendenti dai singoli prodotti e possono venire nell'ambito dello stesso ambiente, accessibile non solo da tutte le aree operative di un'impresa, ma anche da più imprese contemporaneamente. La riduzione del costo di funzionamento dei mercati provoca significativi benefici nell'organizzazione interna e risparmi nella gestione in outsourcing di alcune attività. In figura 2 è stimato il risparmio per ogni settore industriale dovuto all'adesione al business-to-business.

Risparmi stimati garantiti dal passaggio al B2B

Settore industriale	**Risparmio stimato**
Aerospaziale	11%
Chimica	10%
Estrattiva	2%
Communications	5%-15%
Computing	11%-20%
Componenti elettronici	29%-39%
Ingredienti alimentari	3%-5%
Prodotti forestali	15%-25%
Trasporto merci	15%-20%
Salute	5%
Life sciences	12%-19%
Macchinari	22%
Media & advertising	10%-15%
Manteinance, repair e operations	10%
Prodotti combustibili	5%-15%
Carta	10%
Acciaio	11%

Figura 2:Risparmi stimati garantiti dal passaggio al B2B (Trento e Warglier 2001)

È importante ricordare che le riduzioni dei costi che si ottengono con il ricorso al business-to-business non derivano soltanto dalla struttura organizzativa delle imprese, che l'introduzione di sistemi quali MRP o EDI aveva già contribuito ad evolvere, ma dalla standardizzazione nella gestione e nello scambio delle informazioni che il nuovo sistema di comunicazione rende necessaria.

In primo luogo in un B2B sono specificati i beni da negoziare con un grado di precisione nella descrizione dei prodotti assai più elevato di quanto in genere avvenga con l'utilizzo di altri sistemi di scambio; i vari stadi della filiera possono inoltre interagire nell'individuazione delle caratteristiche dei beni intermedi sin dalla fase della loro ideazione rendendo più spedita l'introduzione di prodotti nuovi e accorciando sensibilmente il ciclo produttivo.

Le banche dati del B2B raccolgono le indicazioni dei prodotti concorrenti in modo omogeneo, con un formato unico che riporta le principali caratteristiche semplificando notevolmente l'onere della scelta e quindi l'individuazione delle possibili controparti dello scambio.

Le informazioni sulla vera e propria negoziazione vengono diffuse sia tra i concorrenti sia tra gli operatori posti su differenti stadi della filiera.

L'ultima fase di cui si occupa il B2B è quella relativa al perfezionamento dell'ordine prendendosi cura di garantire il buon esito della transazione e minimizzare il rischio di errori nello scambio. In questa fase possono essere forniti anche servizi accessori per la commercializzazione dei beni intermedi quali servizi di consulenza commerciale, aziendale o finanziaria.

I mercati virtuali generati nell'ambito del B2B si distinguono in mercati orizzontali e verticali. I primi trattano beni indiretti, quei beni cioè che, pur essenziali per il processo produttivo, non vengono inglobati nel

prodotto finale; nei secondi vengono generalmente commercializzati i beni diretti, quei prodotti intermedi che sono incorporati nel bene finale. Ne mercati orizzontali, inoltre, vengono trattati beni utilizzati da tutti i settori industriali mentre i mercati virtuali verticali interessano soltanto le imprese attive in una specifica industria.

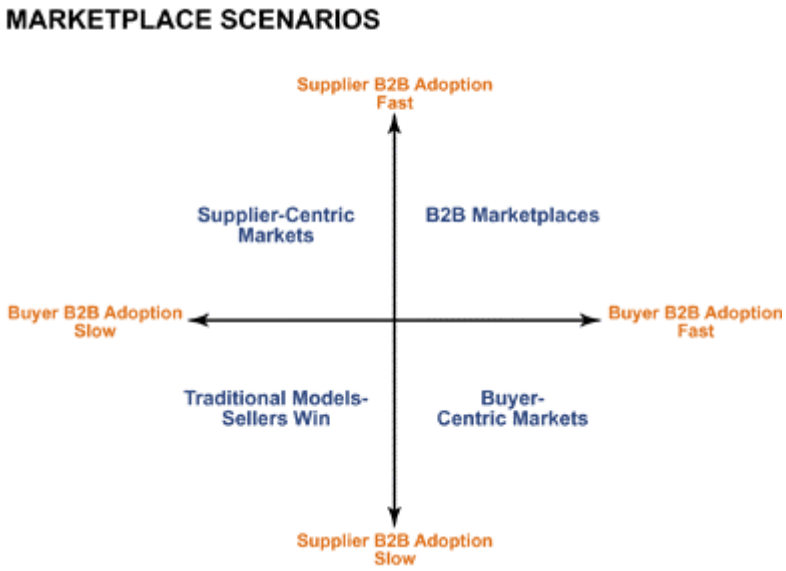


Figura 3: I marketplace

Gli operatori attivi nel B2B si distinguono in relazione agli aspetti di controllo e più in generale al grado di integrazione con le imprese di uno specifico settore industriale. La struttura di controllo ha ad un estremo gli operatori indipendenti, specializzati nell'offrire servizi a molteplici industrie, all'altro estremo la proprietà del B2B è in mano alle imprese del settore per il quale esso opera. Vi sono poi vie intermedie della struttura con una composita partecipazione di operatori dell'industria e di società di *venture capital.*

Le fasi di contrattazione utilizzate sono molto articolate e anche in questo caso vi sono due estremi opposti: da un lato troviamo sistemi con prezzo prefissato (cataloghi in Internet), dall'altro metodi cosiddetti

dinamici per fissare il prezzo in cui le imprese interessate hanno una contrattazione diretta di tipo più tradizionale al fine di definire gli elementi fondamentali del contratto. Rientrano in questa categoria i meccanismi d'asta nei quali si verifica il parziale oscuramento delle informazioni sui prezzi offerti dai partecipanti che, a ciascun round d'asta conoscono unicamente la propria posizione relativa. Le scelte di prodotto possono talvolta essere effettuate non soltanto in base al prezzo, ma anche alle caratteristiche qualitative dei beni offerti.
Si è detto in precedenza dell'effetto dell'adesione al B2B sulla riduzione dei costi di transazione e del vantaggio che questi guadagni arrecano alle imprese, è interessante ora analizzare come questi guadagni si ripartiscano tra i diversi soggetti che fanno parte della filiera e come possano essere un vantaggio netto per l'intera società.

Una catena di fornitura, nella definizione di Christopher (1992) è:

> "the network of organizations that are involved, through upstream and downstream linkages, in the different processes and activities that produce value in the form of products and services in the hands of the ultimate customer"'.

In altre parole, ripercorrendo la successione di processi che trasformano gli input in output e i passaggi che il prodotto deve compiere per acquisire valore, si è in grado di ricostruire la struttura della catena di fornitura di un'impresa o di una successione di imprese e individuare i rapporti dell'azienda con i propri fornitori, come illustrato nella figura 4.

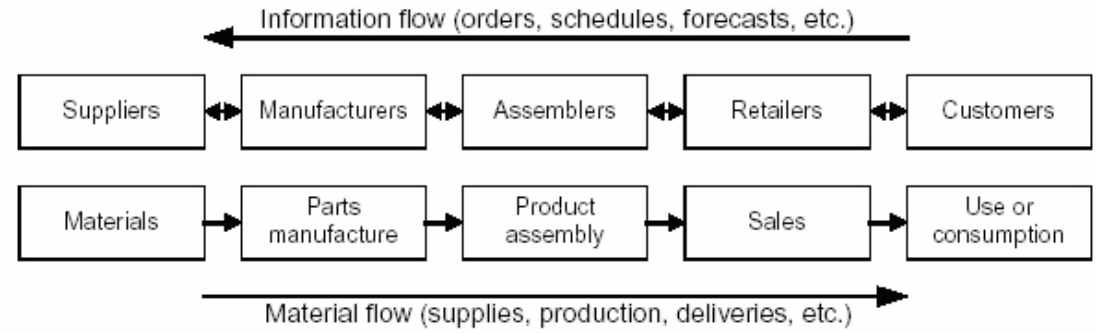


Figura 4: Schema di una supply chain

Gli approvvigionamenti hanno iniziato a diffondersi come unità organizzativa a sé stante intorno gli anni '50 con l'inizio della deverticalizzazione produttiva, della suddivisione per settori merceologici e delle prime indagini sommarie per l'innovazione di prodotto. Lo sviluppo delle tecnologie e della concorrenza dovuta all'internazionalizzazione del commercio hanno ulteriormente incrementato, negli anni successivi, il processo di deverticalizzazione ed il ricorso a mercati specialistici. La struttura per gli approvvigionamenti diventa articolata per mansioni e ha come obiettivo l'ottimizzazione del mix qualità prezzo. La competitività spinta introdotta dal mercato globale, l'innovazione tecnologica che ha modificato processi e dinamica concorrenziale, la delocalizzazione di attività secondarie richiedono maggiore attenzione alle relazioni critiche tra le diverse unità che operano all'interno e all'esterno dell'azienda. L'efficacia di ogni entità dipende strettamente dalla propria tempestività nell'individuare, predisporre e manutenere catene di fornitura innovative, flessibili, competitive e quality oriented.

Il B2B consente alle imprese che riescono a presidiare la fase della compravendita di prodotti intermedi di controllare più strettamente le attività svolte nei diversi stadi della filiera e di incidere sulla concorrenza verticale. Tale concorrenza verticale può risultare a volte ben più intensa di quell'orizzontale che si realizza tra imprese che

operano allo stesso livello della filiera produttiva o nello stesso mercato. Imprese collocate su stadi differenti, infatti, competono aspramente per mutare a proprio favore la redistribuzione dei profitti, questa crescita dei profitti aumenta anche gli utili dei concorrenti presenti sul medesimo mercato. Ne deriva che l'accrescimento dei profitti a un certo stadio della filiera causa una corrispondente diminuzione a un altro livello. In considerazione dei vantaggi che possono essere ottenuti vi è quindi una concorrenza verticale per conquistare il predominio del B2B.

Se siano i fornitori o i compratori a detenere il controllo del B2B dipende essenzialmente dal volume di transazioni e dal grado di concentrazione. Se il mercato a monte è maggiormente concentrato e se il bene intermedio costituisce una porzione assai rilevante del valore finale del prodotto saranno i fornitori ad avere il sopravvento nella gestione del B2B, altrimenti saranno gli acquirenti a prevalere.

Nel funzionamento di un B2B gli scambi di informazioni tra venditori o tra acquirenti e i vincoli di esclusiva a danno di concorrenti o altre imprese della filiera possono alterare le libera concorrenza per questo motivo coloro che controllano un B2B stabiliscono procedure tali da oscurare agli stessi gestori le informazioni per non incorrere in un intervento antitrust.

Se da un lato il B2B rappresenta un mercato potenziale enorme da un altro pare difficile fare valutazioni quantitative.

U.S. B2B E-Commerce Forecasts by Industry

Industry	2000 (billions of dollars)	2004 (billions of dollars)
Computing and electronics	230	593
Motor vehicles	35	412
Petrochemicals	27	299
Utilities	30	266
Paper and office products	14	235
Consumer goods	13	217
Food and agriculture	23	211
Construction	6	141
Pharmaceutical and medical products	4	124
Industrial equipment and supplies	7	70
Shipping and warehousing	5	68
Aerospace and defense	9	33
Heavy industries	3	27
Total	406	2,696

Tabella 2: B2B in USA (Trento e Warglier 2001)

L'adesione a questo nuovo mercato virtuale segna marcate differenze tra gli Stati Uniti e l'area europea. Come si può osservare in tabella XXX gli Stati Uniti, agevolati dall'abitudine e dall'accresciuta esperienza con le tecnologie di rete, dal progressivo consolidamento degli standard industriali, dalla semplificazione delle applicazioni tecnologiche, dalla crescente stabilità delle applicazioni B2B, dalle opportunità di risparmio e di ricavi aggiuntivi, hanno intuito che il commercio on-line non soppianterà quello fisico, ma reagirà e si evolverà con questo attraverso forti sinergie e non solo competizione.

Ricavi dal commercio elettronico in Europa occidentale (miliardi $)

	1998	1999	2000	2001	2002	98/02
Commercio elettronico to	5,6	18,9	49,4	112,5	223,0	151%
B2C	1,9	5,4	12,5	26,1	48,6	126%
B2B per uso finale	0,9	3,1	8,6	19,9	38,3	155%
B2B per uso intermedio	2,9	10,4	28,0	66,4	136,1	163%
Totale B2B	3,8	13,5	37,0	86,4	174,4	161%
Totale usi finali (B2C+B2B)	2,8	8,5	21,2	46,0	86,9	137%
B2C	33%	29%	25%	23%	22%	
B2B per uso finale	16%	17%	17%	18%	17%	
B2B per uso intermedio	51%	55%	57%	59%	61%	
Commercio elettronico pro capite per anno (tota	14,5	48,7	127,0	288,0	571,0	
B2C	4,8	13,9	32,0	67,0	125,0	
B2B per uso finale	2,3	8,1	22,0	51,0	98,0	
B2B per uso intermedio	7,4	26,7	73,0	170,0	348,0	

Tabella 3: B2B nei paesi europei (Trento e Warglier 2001)

Per quanto riguarda i paesi europei ed in particolare l'Italia, la situazione attuale sembra far emergere che le nostre imprese non si stiano muovendo a loro agio nel riconfigurare il funzionamento delle attività sfruttando anche solo una parte dell'enorme potenzialità di Internet.

Le barriere che si oppongono all'adozione del commercio elettronico sono percepite in modo diverso dai manager americani rispetto a quelli europei. L'esperienza americana pone l'accento su alcuni fattori principali. In primo luogo una cultura aziendale talvolta poco incline ad ampliare i propri orizzonti, l' interoperabilità tra le applicazioni di e-commerce e i legacy systems, la carenza di conoscenze o di personale, la carenza di standard, la scarsa sensibilizzazione delle top-management, la difficoltà nell'attrarre personale qualificato, possibili problemi legati alla sicurezza e a difficoltà organizzative sono indicati come barriere e inibitori dell'e-commerce. L'esperienza europea individua nella sicurezza e crittografia, nella mancanza di fiducia, nella cultura aziendale, nelle problematiche relative al riconoscimento dei clienti e alla mancanza di infrastrutture per la certificazione delle chiavi pubbliche, nella carenza di attitudine ai pagamenti on-line, nella mancanza di modelli di riferimento, nell'incertezza sui bilanci, nella carenza di conoscenze o di personale qualificato i principali ostacoli per l'affermazione consistente dell'e-business.

Difficoltà comunemente riscontrate per chi offre prodotti sul web nascono dalla necessità di integrare il sito Web con il sistema informativo gestionale esistente in azienda, dalla mancanza di supporto da parte del management nella definizione delle iniziative di commercio elettronico, dalla struttura aziendale non adeguata al cambiamento richiesto dall'attività in rete e dalla mancanza di fiducia che inducono a ritenere il rischio imprenditoriale eccessivo.

Parte seconda

L’impresa virtuale: modelli di simulazione

Capitolo 3 – La simulazione: vantaggi e problemi

L'obiettivo del capitolo è valutare i vantaggi e le limitazioni che possono derivare dallo studio della realtà attraverso le simulazioni che riproducono nel computer i fenomeni osservati.
Si afferma (Terna 2002) che:

> "(. . .) lo scopo sarebbe quello di stabilire i vincoli, specificare l'ambiente istituzionale o le regole di decisione per gli agenti e poi eseguire la simulazione per vedere che cosa accade. L'idea non è quella di creare un modello matematico che ha le proprie conclusioni implicite nelle premesse. Invece, è quella di eseguire la simulazione come un esperimento mentale, dove ciò che è interessante non sono tanto i risultati finali, ma il modo in cui il processo funziona. E noi, i programmatori, non sapremo come il processo finirà per risultare fino a che non avremo eseguito l'esperimento mentale. L'ordine dovrebbe emergere non dal disegno del programmatore, ma dall'interazione spontanea delle parti componenti."

L'analisi di questo nuovo strumento, che si affianca a quelli tradizionali, con cui la scienza esplora la realtà, ci permette di mettere a fuoco gli obiettivi che ci si pone con la realizzazione del modello di impresa virtuale, che andremo a descrivere nei capitoli successivi.
Esplorare una realtà aziendale attraverso un modello di simulazione consente di migliorare la comprensione dei meccanismi interni all'azienda, rappresentando l'interdipendenza tra le entità che la compongono con diversi livelli di dettaglio, corrispondenti ai diversi livelli di astrazione possibili.

La progressiva diffusione di macchine di calcolo sempre più performanti ha offerto a tutti i settori scientifici la possibilità di ricorrere alla simulazione e aprire nuove strade alla ricerca sperimentale.
Le prime scienze ad utilizzare il nuovo metodo sono state le cosiddette scienze "*hard*" (matematica, fisica, chimica, biologia) in grado di produrre teorie falsificabili basate sul metodo sperimentale.
Negli ultimi anni anche il cosiddetto settore delle scienze sociali si è servito di questo nuovo strumento scientifico per la ricerca.
Accanto ai modelli letterari descrittivi e a quelli matematico statistici tradizionali si sono affiancati i modelli basati su codice informatico.
Si afferma (Parisi 2001) che le simulazioni, a differenza dei modelli tradizionali, non sono soltanto teorie, ma realtà artificiale.
Le simulazioni, infatti, soddisfano i tre criteri che definiscono la realtà:

- la simulazione può essere osservata guardando sullo schermo di un computer, abbiamo quindi accesso ad essa con i nostri sensi;
- attraverso i comandi che abbiamo impostato possiamo agire su di essa e valutare i risultati delle nostre azioni;
- la simulazione costituisce un vincolo, un limite alle nostre azioni, ma nello stesso tempo è il mezzo per svolgerle.

Le simulazioni si pongono come nuovo metodo di ricerca; i risultati della simulazione, cioè i fenomeni virtuali che essa produce, diventano le predizioni empiriche derivate della teoria incorporata nella simulazione. Le simulazioni sono veri e propri laboratori sperimentali virtuali che amplificano le potenzialità dei laboratori sperimentali reali senza ereditarne i difetti e i limiti. In un laboratorio virtuale è infatti possibile sperimentare tutti quei fenomeni che, per ragioni fisiche legate alle dimensioni degli agenti della simulazione o per motivi di durata, non si possono analizzare in un laboratorio reale. Oltre al

vantaggio evidente della ripetibilità dell'esperimento emergere la facilità con cui è possibile modificare le impostazioni della simulazione variando le variabili d'ambiente, la modalità di esecuzione o qualsiasi altro aspetto che si ritenga utile per lo studio.
Le teorie espresse con i modelli tradizionali della scienza fanno predizioni empiriche che possono essere verificate soltanto in riferimento al mondo reale, le simulazioni creano mondi reali o mondi possibili in cui si applicano soltanto alcuni dei principi che governano il mondo reale. Studiare mondi possibili permette di ampliare il campo dei fenomeni mediante i quali mettere alla prova una teoria, oppure studiare fenomeni del passato che non si sarebbe in grado di sperimentare in laboratori reali.
Parisi (2001) afferma che le simulazioni sono mondi, sia che riproducano il mondo reale, sia che riproducano mondi possibili:

> "…una conseguenza del fatto che una simulazione è un mondo è che lo scienziato, anche quello che ha costruito la simulazione, la studierà come se studiasse il mondo reale. Condurrà delle osservazioni sulla simulazione, agirà sulla simulazione per scoprire quali effetti hanno le sue azioni, scoprirà nella simulazione cose che non si aspettava, soluzione ai problemi a cui non aveva pensato, fenomeni e processi che hanno un'aria di novità analoga a quella che hanno i fenomeni e processi del mondo reale."

Nel campo delle scienze sociali si ricercano nelle simulazioni principalmente due contributi:

- vantaggi sostanziali alla comprensione dei fenomeni derivanti dalla necessità di rielaborazione dei concetti per la formalizzazione del modello;
- la possibilità di osservare fenomeni emergenti in sistemi dinamici capaci di auto-organizzazione; la simulazione può mostrare

comportamenti imprevedibili a priori generati dall'interazione di un elevato numero di elementi semplici, riuscendo a realizzare sistemi artificiali intelligenti le cui capacità di cognizione superano quelle delle singole unità e danno luogo ad una vera e propria intelligenza distribuita tra più menti.

Un modello di simulazione (Pistolesi e Paolucci, 2000) comprende due componenti:

- la descrizione dei limiti e delle astrazioni che individuano la zona di applicabilità del modello stesso;
- la descrizione formale della simulazione: i tre stadi (ingresso, elaborazione, uscita) del processo devono avere descrizioni univoche.

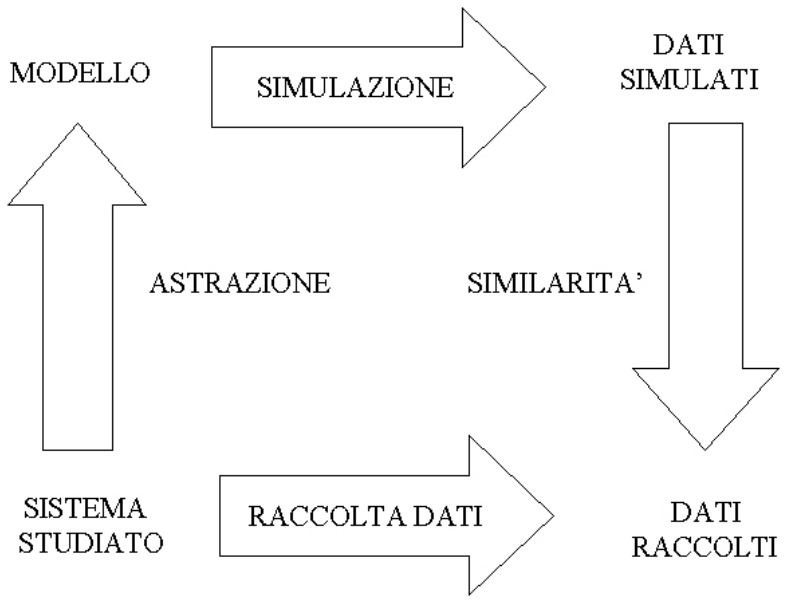


Figura 1:. Rappresentazione schematica della metodologia di studio di sistemi reali attraverso modelli simulativi

I passi della metodologia di studio di sistemi reali attraverso modelli interattivi possono essere illustrati con un semplice schema (figura 1).

Il sistema oggetto di studio è soggetto a due operazioni: una di astrazione per la costruzione del modello e una di raccolta dei dati che serviranno per valutarne la bontà.
Il risultato della simulazione è una previsione del comportamento del sistema modellato, il confronto con la realtà sui parametri di plausibilità, rilevanza e accuratezza danno il valore del modello utilizzato.
Per indagare la complessità del mondo reale si ricorre all'uso del computer per elaborare tecniche algoritmicamente e computazionalmente troppo onerose per la mente umana, ma in grado di supportare la ricerca e l'indagine verso zone altrimenti inaccessibili.
L'abilità del ricercatore diventa allora non quella di calcolo, ma quella di ridurre il problema a qualcosa di algoritmico e di ripetitivo.
La simulazione consente di studiare sistemi complessi in cui la molteplicità di elementi che si influenzano reciprocamente danno vita a proprietà globali non deducibili e non prevedibili.
I sistemi oggetto dell'analisi si possono classificare in:

- semplici: si segue uno schema di causa-effetto semplice e lineare;
- complicati: sono composti da un insieme di elementi interrelati a livello meccanico, il cui funzionamento è prevedibile e organizzabile (es. motore di un'automobile);
- complessi: si contraddistinguono da una dinamica non lineare, non sono né prevedibili né organizzabili e soprattutto non lo è la loro efficienza (es. il tempo, il cervello, il sistema economico);

I sistemi complessi (Kauffman 1993) sono caratterizzati da alcuni isomorfismi:

- stabilità: tendono a mantenersi stabili nonostante i cambiamenti ambientali. Questo viene realizzato con dei circuiti di retroazione

negativa che traggono informazione dall'ambiente. Maggiore la complessità del sistema, e maggiore può essere la sua stabilità;

- finalità: sembrano sempre avere un comportamento finalizzato, nel senso che le loro dinamiche tendono ad ottenere un determinato stato;
- procedura: sequenze di azioni che vengono effettuate per determinare un certo risultato I sistemi più versatili scelgono la sequenza a seconda delle situazioni;
- adeguamento dei comportamenti: sono modelli fondati su agenti artificiali adattivi (AAA) le cui regole di comportamento sono determinate, ad esempio, da reti neurali artificiali (RNA) tramite l'apprendimento;
- anticipazione: è una caratteristica che permette ai sistemi di anticipare i cambiamenti ambientali, rilevando "segni premonitori" e prendendo contromisure prima che il cambiamento vero e proprio si verifichi;
- interazione con l'ambiente: in alternativa al modificare i propri meccanismi interni in risposta all'ambiente, un sistema può agire direttamente sull'ambiente per modificarlo;
- riproduzione: molti sistemi complessi creano altri esemplari di se stessi. Questo è tipico di tutti gli organismi viventi, ma lo stesso principio opera nelle comunità che si riproducono, le aziende che aprono nuove sedi, e così via;
- autoriparazione: una caratteristica tipica dei sistemi biologici e sociali, rara invece nelle macchine;
- riorganizzazione: alcuni sistemi complessi riescono a modificare la loro stessa struttura interna per adeguarsi alle situazioni. È tipico dei sistemi sociali

- autoprogrammazione: è la possibilità di inventare i propri scopi, nonché i metodi per conseguirli. È una cosa che si rileva solo nei sistemi di massima complessità: gli esseri umani.

L'aumentare della complessità in un sistema va in genere a suo vantaggio, perché lo rende più versatile.

Vi sono tuttavia, segnala Kauffman (1993), alcuni problemi tipici legati alla complessità.

Quando un sistema è formato da sottosistemi, questi possono assumere degli obiettivi in conflitto tra loro, che causano danni a tutto il complesso. Questo fenomeno è noto in sociologia come "tragedia dei pascoli comuni". La definizione è stata coniata nel 1968 dal biologo G. Hardin rifacendosi al concetto di capacità di carico delle risorse naturali elaborato dall'economista W.F. Lloyd alla fine dell'800.

"Comuni" significa che questi pascoli potevano essere utilizzati da qualunque membro della comunità per alimentare le proprie greggi. In pratica questa mancanza di vincoli fa sì che ogni pastore sfrutti al massimo i pascoli disponibili per avere più pecore, e innalzare così il proprio standard di vita.

Ma dal momento che ogni pastore ha interesse a ragionare in questo modo, le pecore totali aumenteranno a dismisura, e ciò fa sì che l'erba venga mangiata fino alla radice, desertificando il terreno e, nel lungo periodo, portando la comunità a una miseria ancora maggiore.

Se consideriamo le decisioni singolarmente si nota che queste sono prese al fine di migliorare la situazione, ma a livello aggregato determinano il disastro generale.

L'accentramento riduce i conflitti tra sottosistemi e evita i problemi dei "comuni", ma con pesanti costi di gestione e di raccolta delle informazioni. D'altro canto, decentrare le decisioni aumenta la velocità e la flessibilità, a rischio però di conflitti tra le parti.

Uno dei metodi matematici utilizzati per affrontare la complessità è costituito dai sistemi dinamici.

3.1 - La realizzazione di micromondi: l'utilizzo congiunto di modelli contabili e *system dynamics*

I modelli dinamici (Bianchi 2001) si basano su una prospettiva orientata all'individuazione e allo studio dei circuiti di retroazione (feedback) caratterizzanti i fenomeni gestionali, osservati nell'ambito di sistemi chiusi. Tali modelli tendono ad esplicitare la chiave di lettura implicita della realtà che ciascun decisore detiene nella propria mente.
Un modello dinamico accoglie tre principali categorie di variabili:

1. variabili stock: indicano il livello di risorse chiave del sistema;
2. variabili flusso: rappresentano le variazione in aumento o in diminuzione delle variabili stock;
3. variabili di input: rappresentano vincoli esogeni o leve direzionali sulle quali è possibile agire allo scopo di influenzare le variabili di stock.

Un modello dinamico di simulazione si basa sull'esplicitazione delle politiche che sono alla base del processo decisionale. La formulazione delle decisioni è osservata con un continuo processo di conversione delle informazioni in segnali suscettibili di alimentare delle azioni orientata a modificare gli stock di risorse nella direzione desiderata.
Nel processo di formulazione delle decisioni si possono distinguere due tipi di circuiti di retroazione:

- positivo: descrive un circolo virtuoso o vizioso riguardante un processo di sviluppo o di involuzione;
- negativo: di tipo bilanciante.

La dominanza dei circuiti a retroazione positiva non è illimitata nel tempo, nel lungo periodo possono essere controbilanciati da circuiti a retroazione negativa.
Si distinguono feedback negativi di più gradi e di ordine superiore al primo, la distinzione riguarda il numero di livelli tra loro concatenati nelle relazioni causali che influenzano una determinata risorse chiave.
Quando in un modello dinamico coesistono sia circuiti positivi sia negativi l'andamento delle risorse chiave può variare a seconda della predominanza di un circuito sull'altro; è questo il caso del ciclo di vita di un prodotto.
Un concetto implicito nell'esistenza di un circuito di causa effetto è quello di ritardo temporale. È possibile distinguere due diverse tipologie di ritardo:

- il ritardo fisico materiale: fa riferimento ad un flusso di risolse fisiche che pervengono dopo essere transitate da uno o più stock intermedi;
- il ritardo informativo: può essere importabile sia a congetture soggettive sia alla tardiva disponibilità di informazioni.

Attraverso i modelli dinamici i diversi processi gestionali in relazione ai quali si desidera comprendere le strategie e politiche da adottare in futuro sono rappresentati attraverso sistemi chiusi, ovvero mediante circuiti di causa effetto che inglobano le strategie e politiche costituenti oggetto di valutazione. Un sistema può definirsi chiuso quando è influenzato dai risultati che esso stesso ha concorso a generare in passato.
Gli attori chiave dell'impresa devono comprendere la struttura del sistema in cui operano. La realizzazione del modello diventa parte di un più ampio processo di apprendimento che tende all'acquisizione di un assetto mentale che consenta all'imprenditore proprietario di

cogliere cause da effetti, ritardi temporali, relazioni non lineare tra le variabili rilevanti e intervenire sulla configurazione del sistema aziendale per influenzare la dinamica futura delle risorse chiave verso la direzione desiderata.
Migliorare i modelli mentali per avvicinarli quanto più possibile alla realtà è il presupposto per un consapevole governo dello sviluppo aziendale. Il processo di apprendimento continuo si articola lungo le seguenti fasi:

- osservazione della realtà;
- riflessione, esplicitazione e confronto della percezione di ciascun decisore in merito al sistema;
- diagnosi e condivisione di uno schema interpretativo della realtà;
- formulazione delle decisioni.

Il processo di modellizzazione dinamica viene condotto attraverso un approccio maieutico da un soggetto esterno all'impresa, il cosiddetto facilitatore dell'apprendimento.
Affinché lo studio di un fenomeno possa giustificare l'adozione della metodologia della system dynamics deve presentare determinati requisiti:

- esistenza di relazioni di causa effetto;
- presenza di effetti di breve e lungo periodo contrastanti tra loro per una medesima variabile;
- andamenti contrastanti nel tempo fra diverse variabili;
- esistenza di significative relazioni non lineari fra diverse variabili rilevanti;
- il significativo peso di variabili non monetarie o anche puramente qualitative sulla dinamica delle risorse chiave oggetto dall'analisi.

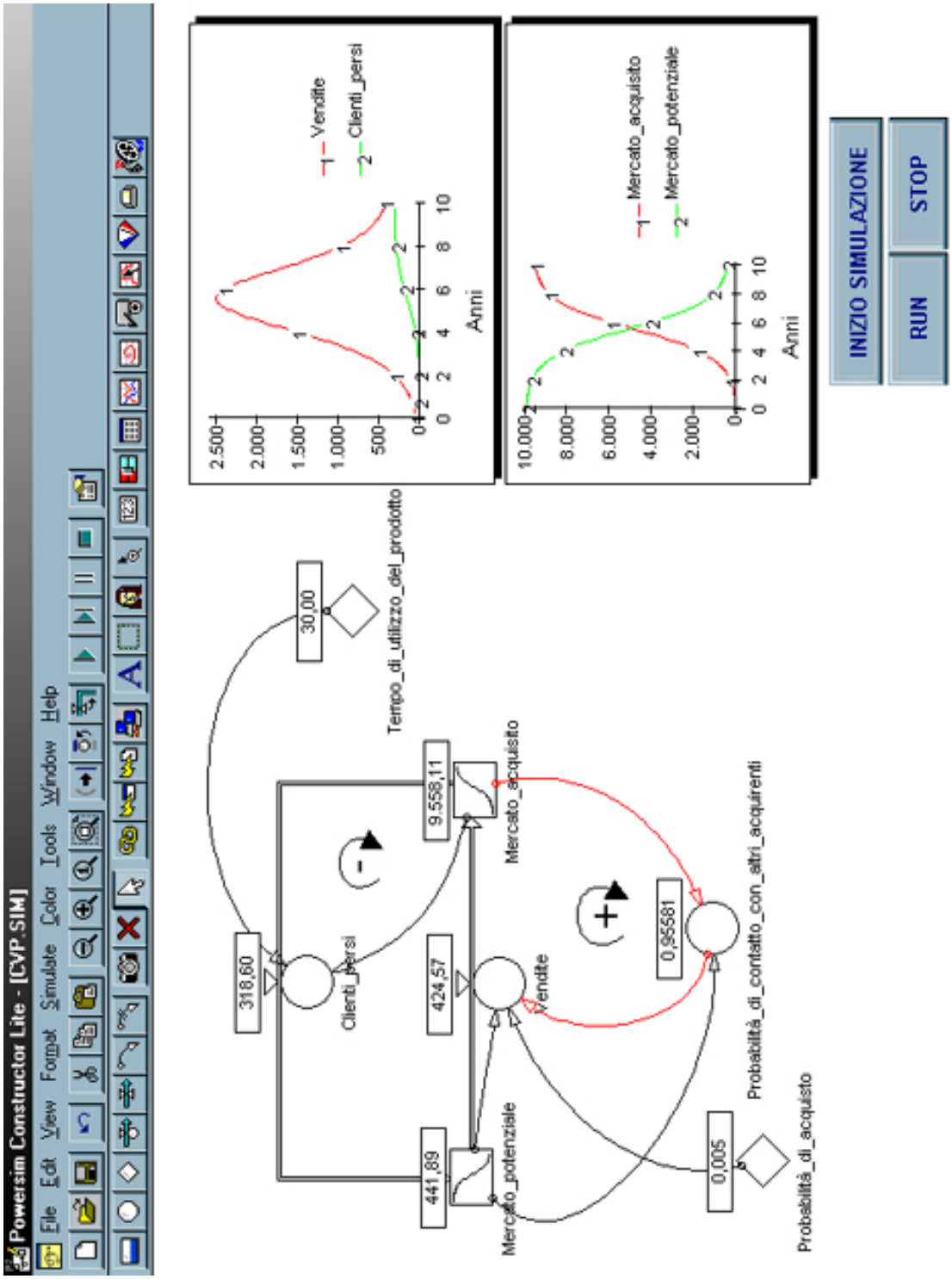


Figura 2: Esempio di formalizzazione di un sistema dinamico tramite il software Powersim Constructor Lite

Quanto più elevata è la complessità gestionale e la discontinuità ambientale, tanto maggiore è il rischio che la realtà oggettiva venga percepita da ciascuno degli attori chiave aziendali in modo diverso. La diversità dei modelli mentali è una ricchezza per l'azienda, perché consente di osservare i fenomeni gestionali sotto differenti prospettive.

Figura 3: La circolarità del processo di modellizzazione dinamica

I modelli dinamici hanno una natura descrittiva relativamente alla realtà rappresentata, le informazioni e i valori in essi accolti sono finalizzati ad offrire una visione di sintesi riguardo alla struttura e alla dinamica del sistema in funzione delle ipotesi adottate, con riferimento ad un determinato arco temporale. L'attitudine a rappresentare

l'andamento delle risorse chiave in funzione delle politiche volte di influire su di esse ed egli scenari volitivi del sistema analizzato sono una misura della qualità del modello come supporto all'apprendimento. Una progressiva focalizzazione dei confini del sistema rilevante consente di migliorare i modelli mentali dei decisori.

Figura 4:L'attività di programmazione e controllo come processo di apprendimento continuo attraverso il supporto di modelli dinamici

L'esplicitazione dei modelli mentali rappresenta un mezzo attraverso il quale i soggetti non individuano semplicemente le soluzioni ai problemi, ma indagano sulle cause al fine di adottare delle scelte che siano in grado di incidere sulle reali disfunzioni piuttosto che sui sintomi esterni.

Il processo di modellizzazione - apprendimento deve rendere visibili a ciascun attore chiave i limiti sottostanti ai propri "filtri informativi" ed

a promuoverli in relazione al progressivo emergere dell'evidenza dei fenomeni concreti, come risultato di un processo di diagnosi condiviso dai diversi soggetti operanti nel sistema aziendale.
Con il termine micromondi (Bianchi 2001) si intendono modelli di simulazione che tendono a riprodurre le principali caratteristiche del sistema in cui opera il decisore, consentendo a quest'ultimo di sperimentare in un ambiente protetto, individualmente o in gruppo, gli effetti derivanti dall'adozione di determinate strategie e politiche.

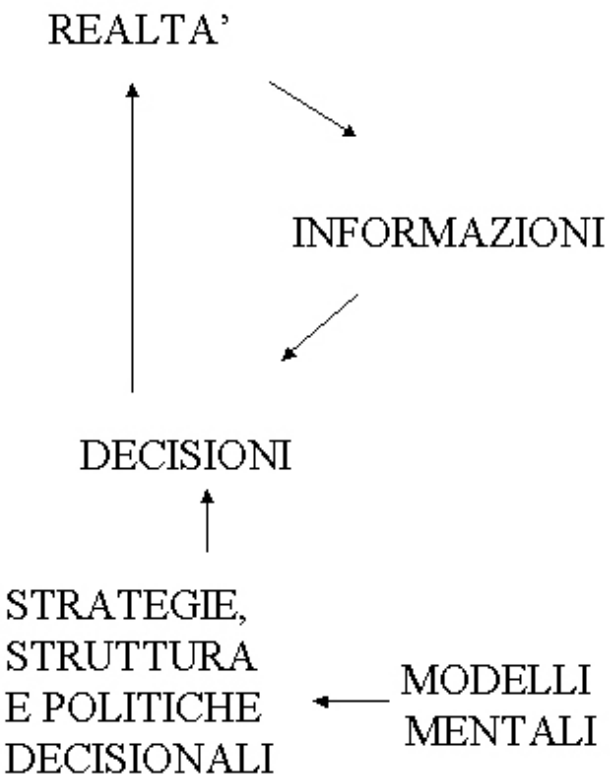


Figura 5: ReLazioni tra decisioni e sistema di controllo

Attraverso i micromondi basati su modelli dinamici il soggetto decisore può scorgere in anticipo i presumibili effetti sulle risorse chiave conseguenti alla manovra delle leve direzionali disponibili e quindi meglio comperati diversi possibili esiti conseguenti all'adozione delle

alternative strategiche perseguibili. I micromondi sono quindi in grado di fornire una chiave di lettura adeguata ai fabbisogni conoscitivi sia per le situazioni di cambiamento incrementale, sia per quelle di cambiamento strutturale. Sono stati distinti micromondi di prima e di seconda generazione, la differenziazione riguarda il modo in cui tali strumenti sono in grado di influire sulle fasi caratterizzanti il processo di apprendimento individuale e di gruppo.
Tali strumenti cognitivi possono essere di utile adozione a supporto dei meccanismi di programmazione e controllo, la loro realizzazione deve essere personale per ogni impresa e può convenientemente basarsi su un opportuno connubio tra modelli contabili e dinamici.

3.2 – Simulazione ad agenti e programmazione ad oggetti: il progetto Swarm

Il progetto *Swarm* (www.swarm.org) nasce nel 1995 dal lavoro di un gruppo di ricercatori, poi trasformatosi in un'organizzazione no-profit, del Santa Fe Institute (New Mexico, USA) al fine di produrre strumenti per il lavoro di ricerca orientato allo sviluppo di simulazioni ad agenti.
Swarm è una libreria di funzioni scritte in Objective C o Java, rilasciata in licenza *open source* (sotto la GNU General Public Licence).
Inizialmente, utilizzando il linguaggio *Objective-C,* era diffuso soltanto nell'ambito della piattaforma Unix.
I primi compilatori *Objective-C* per il sistema operativo Windows e nuove versioni di Swarm in linguaggio java, strettamente legato allo sviluppo di internet e compatibile con sistemi diversi, hanno contribuito alla sua diffusione.

Le librerie svolgono due funzioni principali:

- possono essere utilizzate dai costruttori dei modelli per creare oggetti direttamente dalle classi in esse contenute;
- possono essere utilizzate per creare sottoclassi specializzate in particolari funzioni, in relazione al sistema che si desidera modellizzare.

L'unità fondamentale nel contesto del progetto è l'agente, che costituisce l'oggetto del sistema.
La simulazione consiste di gruppi di agenti, costituiti da un insieme di regole e di capacità di risposta agli stimoli, che interagiscono tra loro attraverso un meccanismo di notifica di messaggi.
Una collezione di oggetti, il cui comportamento è definito dall'interazione degli agenti in essa contenuti, costituisce uno sciame (da cui il nome Swarm).
Ogni sciame contiene un orologio indipendente da quello degli altri sciami, che può scandire in modo diverso il tempo della simulazione definito dall'orologio globale cui tutti gli agenti fanno riferimento.
Se nella realtà il tempo trascorre indipendentemente dalla volontà e dalle azioni degli operatori, in un modello di simulazione è necessario gestire e regolamentare questo aspetto. Comunemente il tempo della simulazione è scandito dagli intervalli che intercorrono tra le azioni degli agenti in ogni ciclo e non da un tempo macchina di calcolo.

I linguaggi utilizzabili per sviluppare il codice ad oggetti si possono suddidividere in:

- *object oriented:* dispone di strumenti per gestire l'ereditarietà dei soggetti;
- *object based:* si serve di interfacce per simulare il concetto di ereditarietà.

Nella tabella 1 è fornita una comparazione tra diversi linguaggi appartenenti alle due categorie sopra indicate.

Proprietà	Object Based (es. Visual Basic)	Object Oriented (es. Java, Objective C)
Astrazione	Si	Si
Classe&Oggetto	Si	Si
Incapsulamento	Si	Si
Ereditarietà	No	Si
Polimorfismo	No	Si

Figura 6: Linguaggi a confronto

Per meglio comprendere le differenze tra le diverse tipologie di linguaggio di programmazione analizziamo di seguito il significato delle proprietà:

- *astrazione;*
- *classe & oggetto;*
- *incapsulamento;*
- *ereditarietà;*
- *polimorfismo.*

Per *astrazione* si intende la capacità di esprimere concetti senza considerarne i dettagli, in programmazione significa scrivere un programma valido per un determinato argomento indipendentemente dai contenuti che dovrebbe rappresentare, suddividendo le funzioni da svolgere nel programma in singoli oggetti in grado di svolgere autonomamente il compito cui è preposto.

La *classe* è una descrizione di *oggetti* simili, che si ottengono con una *instance,* la creazione di un esemplare, della classe stessa.

Una classe si crea in questo modo:

[accesso] [modificatori] class nomeclasse { … }

In [accesso] si può facoltativamente indicare public, protected o private, che determinano la visibilità della classe rispetto alle altre e quindi il suo livello di accesso.
Ogni classe necessita di un costruttore: un metodo speciale che ha lo stesso nome della classe.
Programmare ad oggetti significa definire classi di oggetti tra loro correlati dagli elementi in comune, i metodi.

L'*incapsulamento* è la capacità di racchiudere all'interno dell'oggetto le informazioni relative alle scelte di progetto, i metodi e le variabili.
Definire le variabili e i metodi come pubblici, privati o protetti significa consentire all'utente di entrare nell'oggetto per modificarlo o esclusivamente di utilizzarlo senza poter alterare ciò che in esso è incapsulato. Questo aspetto consente all'utilizzatore di disinteressarsi delle complicazioni interne al metodo qualora egli debba riutilizzarlo in altre applicazioni, con evidenti vantaggi di gestione degli strumenti a sua disposizione.

L'*ereditarietà* è un sistema che consente di derivare delle sottoclassi da una classe precedentemente definita. Le sottoclassi erediteranno variabili e metodi dalla classe *parent* e si potranno aggiungere manualmente altre variabili e metodi, specifici della sola classe derivata.
Sarà dunque possibile utilizzare contemporaneamente istanze della superclasse e delle classi derivate e derivare classi da sottoclassi già derivate, creando dunque una sorta di "albero genealogico".
La sintassi da utilizzare è la seguente:

```
class nome_nuova_subcl extends superclasse_ereditata {...};
```

Il *polimorfismo* è la capacità dell'oggetto di rispondere ad un comportamento generico con un'azione specifica, la programmazione ad oggetti consente, infatti, di riferirsi con lo stesso nome a variabili, metodi o oggetti diversi, che, essendo appunto poliformici, potranno essere utilizzati per finalità differenti.
Uno dei vantaggi considerevoli della programmazione ad oggetti risiede nella possibilità di reimpiegare il software e quindi di strutturare più progetti su una stessa base.

Nella costruzione di modelli di dimensioni rilevanti può divenire problematico monitorare tutte le parti se non si identifica una struttura con cui descrivere il modello stesso.
Uno schema generale che può essere utilizzato per costruire simulazioni basate su agenti è quello rappresentato nella figura 7 (Gilbert e Terna 2000), si tratta dello schema ERA (Environment-Rules-Agents).
L'ambiente contiene le informazioni condivisibili da tutti gli agenti del modello, le cui interazioni si realizzano nell'ambito dello schema Swarm.
Lo strato denominato *Agent* corrisponde alla rappresentazione logica del modello, in cui sono descritti tutti gli agenti, che devono possedere soltanto i metodi relativi alle diverse tipologie di operazioni che sono in grado di svolgere.
Le regole e la capacità adattiva che ne governano il comportamento devono essere collocate nella sezione Rules.
Il RuleMaster è il depositario delle regole, esso è legato nello schema ad un altro oggetto, detto RuleMaker il cui compito è quello di modificare le regole che governano il comportamento dei singoli agenti.

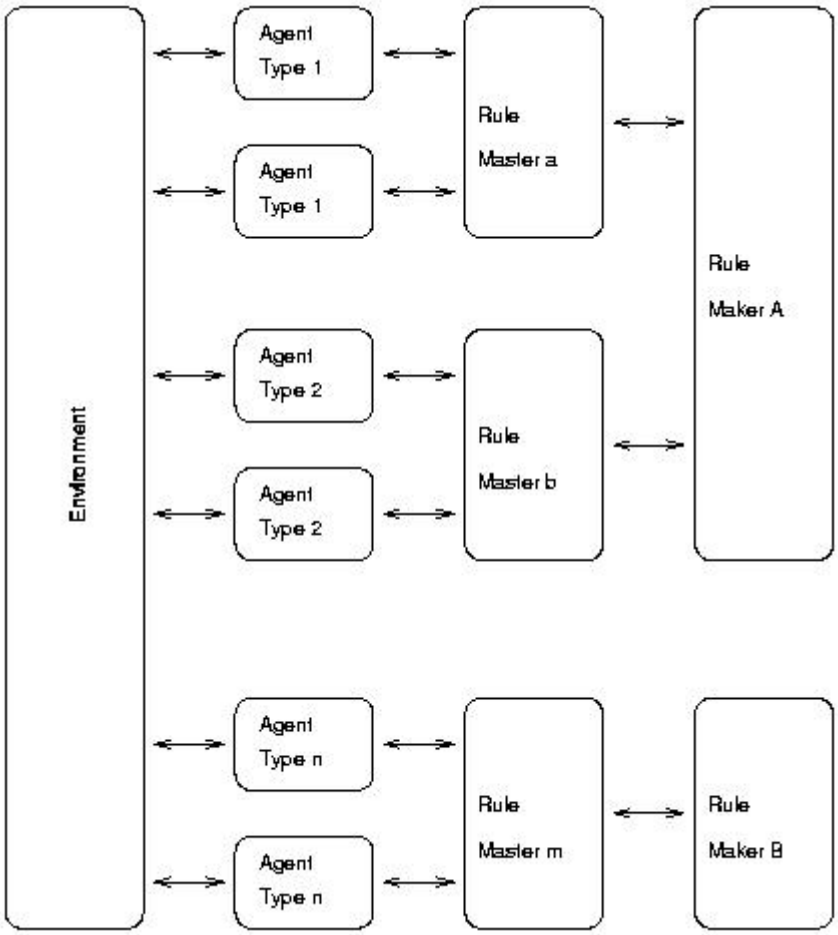


Figura 7: Lo schema ERA

Lo schema di funzionamento di una simulazione costruita con Swarm è rappresentato nella figura 8.

Le classi *Model* ed *Observer,* la cui struttura sarà analizzata nel dettaglio nel corso del capitolo seguente attraverso la descrizione del modello jVE, consentono di perfezionare la simulazione separando l'aspetto grafico da quello strettamente inerente agli eventi generati dagli agenti.

Un elemento molto interessante è costituito dalle *probe*, sonde che consentono all'utilizzatore del modello di visualizzare ed eventualmente

modificare il contenuto dei dati dei singoli oggetti (agenti) in modo interattivo.

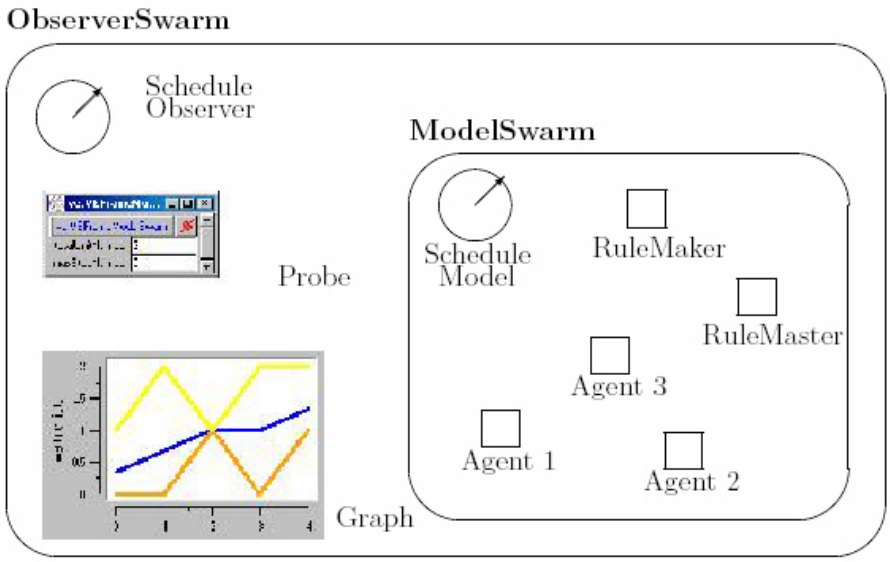


Figura 8: Schema di funzionamento di una simulazione in Swarm

Queste funzioni consentono all'utilizzatore di osservare i parametri critici della simulazione nel corso della sua esecuzione e di trarne importanti informazioni ai fini dell'indagine condotta nella ricerca.

Il progetto di simulazione di impresa per l'applicazione ad un caso concreto, descritto nei capitoli successivi, è sviluppato in Swarm.

Capitolo 4 – L'azienda virtuale industriale: il modello NIIIP

L'analisi del progetto NIIIP di azienda virtuale industriale, inteso come aggregato di aziende reali, può fornire preziose indicazioni applicabili alla simulazione di impresa, organismo individuale, oggetto di questo lavoro.
Estendendo a livello teorico la simulazione, si possono, infatti, intendere le relazioni che intercorrono tra le unità produttive all'interno di un'impresa come relazioni tra imprese appartenenti ad uno stesso settore e quindi trasportare ad un livello di analisi macroeconomica le problematiche relative a questioni interne all'azienda.

4.1 Il progetto NIIIP

Il progetto NIIIP (*National Industrial Information Infrastructure Protocols consortium*) nasce nel 1994 da un accordo tra il Governo americano e alcune industrie volto a sviluppare un'infrastruttura informativa che favorisse la collaborazione tra imprese di uno stesso settore produttivo al fine di incoraggiare la condivisione dei costi infrastrutturali e delle informazioni relative alle tecnologie produttive, ridurre i tempi di attesa nella catena di fornitura e di reazione ai cambiamenti.
Il modello di industria virtuale, che si delinea nel progetto NIIIP, ricalca il paradigma del business-to-business e si propone di essere impiegato sul mercato americano come standard di dialogo tra le imprese.
Pur rimanendo entità separate, il grado di cooperazione tra gli aderenti al progetto, può arrivare ad assomigliare a quello che si realizza nell'ambito di un'impresa verticalmente integrata e consentire anche alle piccole imprese di competere nel mercato globale.

Il modello NIIIP si articola in diversi progetti (figura 1).

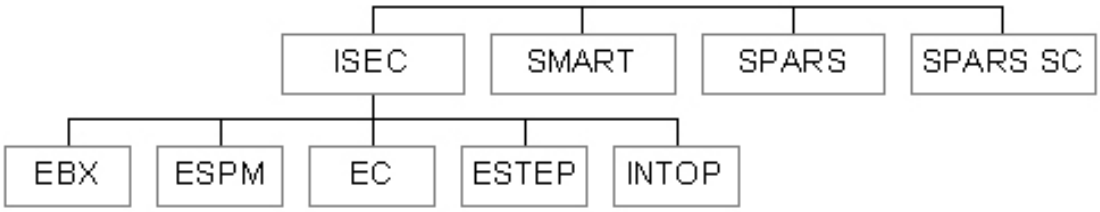


Figura 1: Progetti NIIIP

Integrated Shipbuilding Environment Consortium (ISEC)- Extended Smart Product Model (ESPM) è il progetto cui è legato lo sviluppo di componenti software, corsi di apprendimento e dimostrazioni per verificare la validità della tecnologia proposta.

The Integrated Shipbuilding Environment Consortium- Electronic Commerce (EC) è un progetto che si propone di valutare i vantaggi dell'introduzione del commercio elettronico nelle catene di fornitura di componenti per la progettazione e la costruzione di navi.

MES-Adaptable Replicable Technology (SMART) è un progetto finalizzato a consentire l'integrazione e l'interoperabilità tra i MES (Manufacturing Execution Systems).

Shipbuilding Partners and Suppliers (SPARS) è un progetto per estendere le potenzialità dell'impresa virtuale al mondo dei cantieri navali.

The Shipbuilding Partners and Suppliers (SPARS) Consortium, Supply Chain (SC) si propone di integrare all'interno dell'impresa virtuale le catene di fornitura del settore di produzione di navi per rendere più trasparenti le relazioni tra clienti, soci, subappaltatori e fornitori e ridurre i costi e i tempi di approvvigionamento.

Tutti i progetti sopra citati concorrono a generare un un'organizzazione temporanea di aziende che si coalizzano per realizzare una *Industrial Virtual Enterprise.*

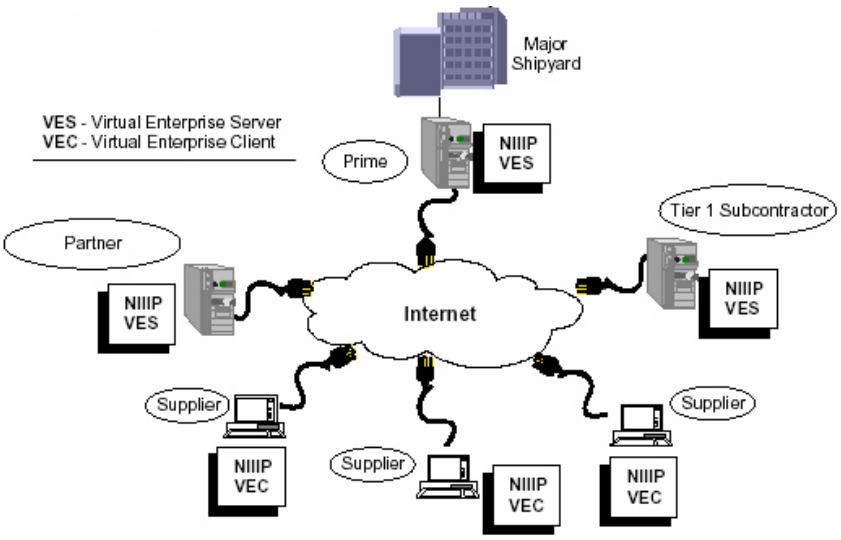


Figura 2: Supply chain in VE

4.2 L'architettura NIIIP

Per perseguire gli obiettivi prefissati il consorzio NIIIP ha sviluppato un'architettura (figura 3) aperta e con tecnologia non proprietaria in cui si identificano quattro elementi necessari per la realizzazione di un'azienda virtuale:

- protocolli di comunicazione comuni;
- tecnologia ad oggetti uniforme per garantire la comunicazione tra i sistemi e tra le applicazioni;
- modello comune di scambio delle informazioni;
- gestione cooperativa dei processi integrati della *virtual enterprise* (VE).

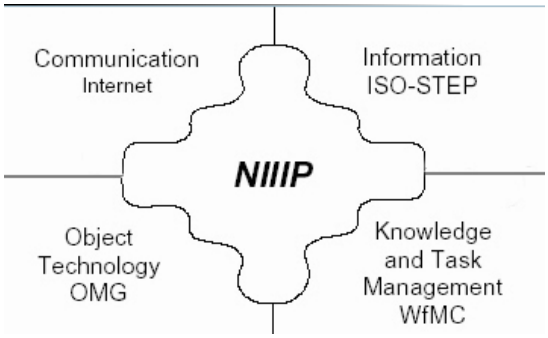


Figura 3: L'architettura NIIIP

Per quanto concerne il trasferimento dei dati il consorzio ha deciso di adottare il protocollo di comunicazione Tcp/Ip, che garantisce stabilità e supporta numerosi applicativi.

Sulla base del paradigma (figura 4) sviluppato dal consorzio, una applicazione della VE seleziona un meccanismo che le consenta di accedere ai dati e, tramite un messaggio inviato attraverso questa infrastruttura, invoca un servizio.

Il messaggio che viene inviato deve essere confidenziale e non può essere divulgato a soggetti terzi non autorizzati. La fonte del messaggio deve essere identificabile in modo attendibile, la sicurezza delle informazioni e la certezza che i dati non vengano alterati è garantita dal protocollo di comunicazione.

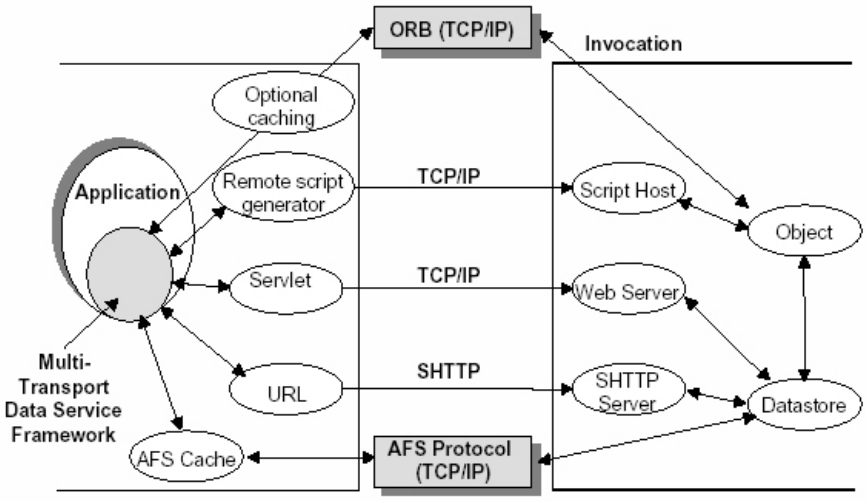


Figura 4: Il trasferimento dei dati

Il secondo livello dell'architettura definisce la tecnologia degli oggetti software. Essi devono rispondere alle specifiche denominate CORBA 2.0. Gli oggetti sono dotati di un'interfaccia che garantisce loro la visibilità attraverso la rete e sono in grado di comunicare all'esterno grazie al linguaggio IDL (*interface definition language*).

Il software adottato dal consorzio si appoggia all' *Object Management Group (OMG)* ed è dotato di caratteristiche di portabilità, interoperabilità e possibilità di riutilizzo.

Il modello comune di scambio delle informazioni cui fa riferimento lo standard del consorzio è ISO STEP (*the international standard for exchange of products*) che richiede il rispetto di alcune specifiche.

Il protocollo stabilisce, in primo luogo, che le informazioni relative ai differenti gruppi di lavoro operino tra loro in modo da costituire un'unica struttura logica relativa al prodotto, che deve essere fornito di una propria documentazione. Tutte le applicazioni, infine, devono sviluppare i differenti aspetti del progetto, compresi i processi di produzione.

L'ultimo livello riguarda la predisposizione delle strutture di descrizione delle informazioni, che garantiscono la possibilità di

collaborazione tra le aziende, attraverso la realizzazione di un modello che unifica i flussi di dati, le relazioni tra i gruppi e le applicazioni, favorendo l'integrazione fra le differenti competenze.

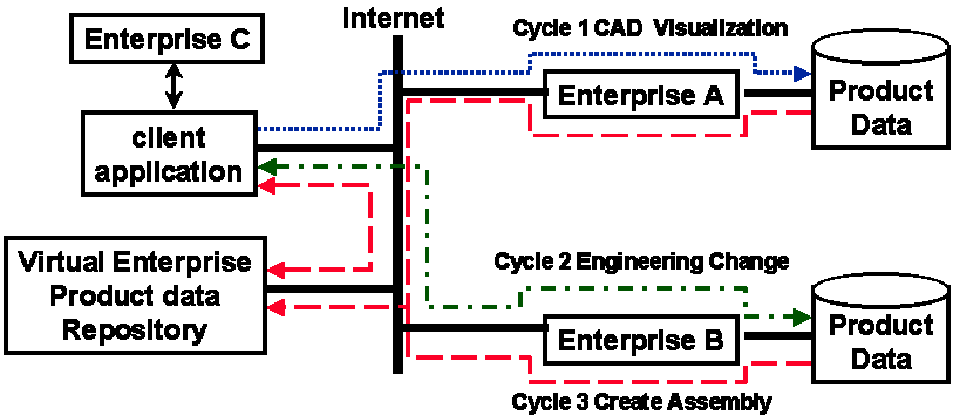


Figura 5: The challenge problem of the NIIIP project from the perspective of the product data team

Il modello è strutturato su tre cicli (esempio figura 5):

- nel primo si descrivono le risorse di ciascun componente, i dati, le associazioni tra essi, le regole;
- nella seconda fase è definito un schema globale che descrive la totalità delle risorse dell'organizzazione virtuale;
- il terzo prevede la descrizione dei meccanismi che permettono di mediare i contenuti dei singoli schemi locali nella metrica dello schema globale.

Il risultato di questo processo è la descrizione unitaria di un meta-modello dell'azienda virtuale.
In figura 6 sono descritti i passi del processo di formazione di una VE.

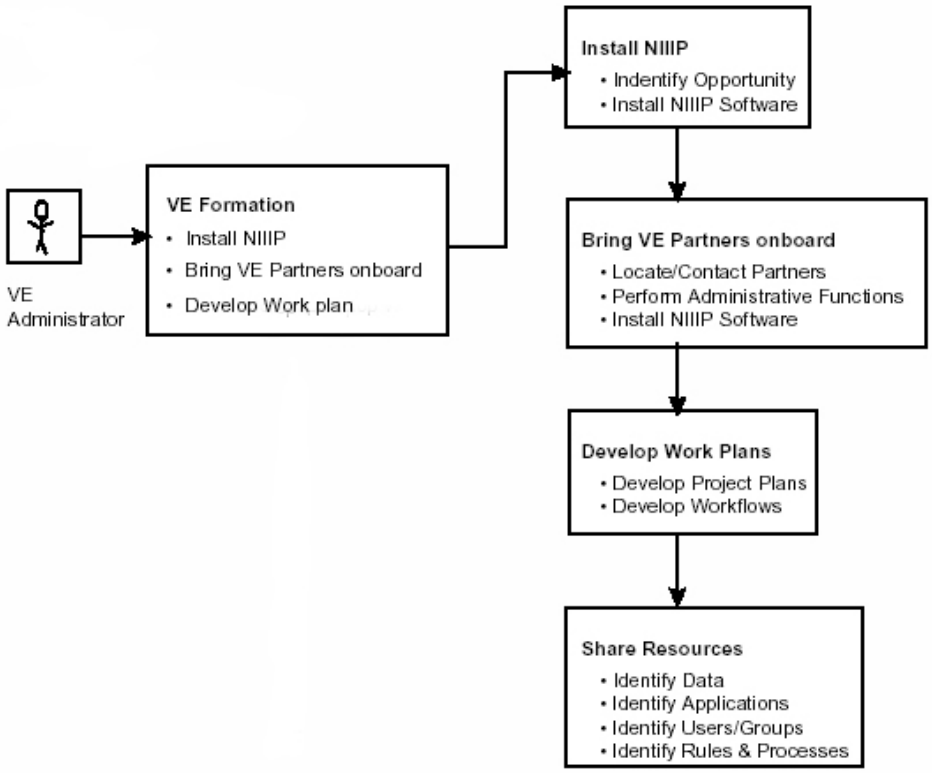


Figura 6: Processo di formazione di una VE

Affinché l'impresa reale prenda forma è necessario predisporre *interfaces protocols,* oggetti con attributi, stati e funzioni che consentono di definire le dotazioni dell'impresa.

Gli utilizzatori finali del progetto e gli agenti che rendono disponibili i servizi si incontrano attraverso il NIIIP *Desktop* (figura 7), il luogo di esecuzione del lavoro dell'impresa virtuale.

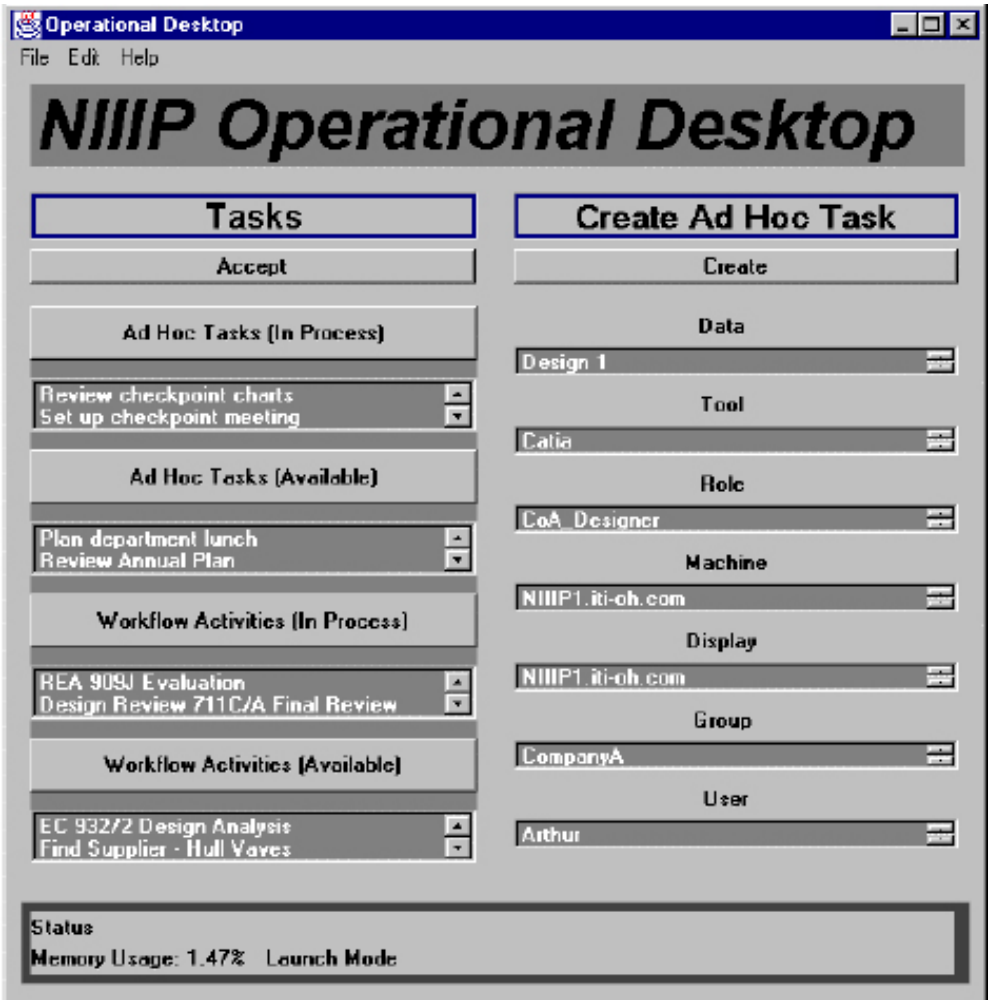


Figura 7: NIIIP Desktop

I flussi di lavoro tra i componenti sono gestiti dal *workflow,* mentre un *mediator* regola il dialogo tra i vari database.

Le risorse della VE sono raccolte dal *VE Member Resources,* una piattaforma comune facilmente accessibile, che le rende visibili a tutti gli utilizzatori affinché possano avere una visione di insieme del sistema.

I vantaggi per le imprese che aderiscono al modello sono rilevabili in termini di competitività, accresciuta dalla collaborazione attraverso l'impresa virtuale, di facilità di accesso ai mercati senza limiti geografici e di specializzazione sul prodotto, data l'interazione tra più imprese in diverse fasi della catena del valore.

Il potenziale difetto di questa metodologia è la scarsa flessibilità del risultato derivante dalla necessità, per i progettisti del paradigma NIIIP, di scegliere alcune tecnologie e definire gli standard per una realtà in continua evoluzione.
Un esempio che avvalora questa tesi è fornito dalla constatazione che nello standard NIIIP non è previsto l'utilizzo del linguaggio XML come formato universale di descrizione e trasporto delle informazioni, che si sta prepotentemente affermando sul mercato.

4.2.1 Il paradigma XML

XML (eXtensible Markup Language) è un nuovo standard che si sta affermando sempre più in quelle realtà in cui l'interscambio di dati tra applicazioni e l'integrazione di sistemi eterogenei sono percepite come problematiche di primaria importanza e complessità.
Ogni applicazione rappresenta, infatti, i propri dati in formati differenti ed anche se consente l'accesso alle informazioni tramite i protocolli di rete o per mezzo di API (Application Programming Interfaces) i meccanismi per farlo cambiano di volta in volta. Questo approccio, sebbene tuttora ampiamente utilizzato, presenta notevoli inefficienze: poca flessibilità, portabilità limitata, scarsa robustezza e costi di sviluppo e manutenzione elevati e obbliga le imprese a concordare di volta in volta schemi rigidi e proprietari per intraprendere con successo una business-communication.
XML sembra proprio essere la soluzione a tutto questo.
Si tratta di una specifica per organizzare le informazioni all'interno di documenti caratterizzata da due aspetti che la rendono estremamente flessibile ed efficiente.

Un documento XML:

- è rappresentato da testo: questa caratteristica rende tale formato universale ed indipendente dalla piattaforma (un file di testo su Windows è compatibile con il sistema operativo UNIX) conferendogli di conseguenza una notevole portabilità;
- contiene i tags che descrivono la struttura dell'informazione nel documento, i dati sono descritti in termini di elementi ai quali possono essere associati degli attributi.

Le informazioni relative ad un cliente potrebbero essere rappresentate in un documento XML nel seguente modo:

```
<CLIENTE>
<SOCIETA> B&B&B </SOCIETA>
      <INDIRIZZO> Corso Tesi</INDIRIZZO>
      <CITTA>Torino</CITTA>
      <CAP>10100</CAP>
      <PAESE>Italia</PAESE>
      <TELEFONO>011 123456</TELEFONO>
</CLIENTE>
```

Gli elementi XML possono essere annidati in maniera più o meno complessa e in modo del tutto arbitrario per rappresentare informazioni strutturate, nell'esempio appena illustrato l'elemento cliente ha cinque sotto-elementi: società, indirizzo, città, cap, paese e telefono; gli attributi consentono di descrivere ed arricchire maggiormente la semantica dell'informazione.

A prescindere dall'informazione che contiene, un documento XML deve attenersi a due semplice regole:

- deve esistere un elemento root che contiene tutti gli altri (l'elemento <cliente> nell'esempio appena mostrato);
- gli elementi devono essere annidati in maniera rigorosamente gerarchica.

Un documento XML è considerato valido se è conforme ad una particolare Document Type Definition (DTD) mediante la quale è possibile specificare un particolare insieme di elementi e regole all'interno di un documento.
Per esempio, la DTD del database dei clienti potrebbe specificare che deve esistere un elemento <CLIENTI> per il quale deve essere precisato l'attributo ID e che tali elementi a loro volta ne conterranno altri come <SOCIETA>, <INDIRIZZO> che possono essere più o meno obbligatori o che devono rispettare un certo ordine.
Grazie a questo meccanismo le applicazioni XML prima di utilizzare un documento saranno in grado di validarne la stuttura verificando che sia conforme ad una specifica DTD.
Le applicazioni di B2B sono implementate attraverso l'uso di server che trasmettono e ricevono documenti XML, usando i protocolli web esistenti come l'HTTP. Questi server, che prendono il nome di B2B integration server, forniscono le interfacce alle risorse aziendali esistenti come database, legacy-system ed applicazioni ERP generando, a fronte di una richiesta, la creazione e la trasmissione di documenti XML.

A conclusione di questo capitolo si può affermare che il progetto NIIIP rappresenta un primo tentativo di creazione di un'infrastruttura che permetta ad ogni azienda di fare e-commerce mantenendo la propria piattaforma informativa.
La difficoltà di realizzazione del modello risiede, però, nell'enorme fase progettuale da realizzare e nei tempi che essa può richiedere.
La dinamicità del mercato dell'informatica ed il continuo sviluppo di nuovi standard renderebbero, infatti, necessario un continuo aggiornamento del modello mettendo a rischio la possibilità di avere il tempo per farne un uso effettivo.
Il futuro dell'e-commerce e dell'integrazione delle informazioni si gioca sull'affidabilità dello scambio di dati, che deve avvenire in maniera sicura, asincrona e standardizzata e deve disporre di una soluzione poco costosa, scalabile e facile da mantenere.
Riportando le analisi effettuate a livello macroeconomico ad una dimensione aziendale, dall'analisi del progetto NIIIP si possono trarre suggerimenti funzionali per la simulazione di impresa in termini di organizzazione della comunicazione e portabilità degli oggetti software.
A livello di gestione della comunicazione, nel modello di simulazione di impresa, che sarà descritto nei capitoli 5 e 6, le entità che tra loro devono entrare in contatto sono da un lato le unità produttive e dall'altro le ricette di produzione.
Affinché l'interazione tra questi due aspetti del sistema sia proficua è necessario definire uno standard di comunicazione delle informazioni che permetta alle unità di operare correttamente secondo le informazioni contenute nelle ricette.
Per quanto concerne la portabilità, che nel progetto NIIIP è riferita al supporto delle interfacce che mettono in comunicazione gli aderenti al progetto, a livello di sviluppo di un modello di simulazione di impresa può essere una indicazione per l'evoluzione del codice.

In questo lavoro si tratta il caso dello sviluppo di un modello di impresa virtuale per l'applicazione ad un caso concreto.
La filosofia secondo la quale sono state apportate le modifiche al codice è stata quella di generalizzare il modello, per renderlo applicabile a imprese tra loro differenti.
Una volta individuato un aspetto della realtà aziendale in esame che richiedesse un'evoluzione del modello, il codice è stato sviluppato in modo da poter trattare non soltanto il caso specifico, che ne ha determinato la necessità di evoluzione, ma svariate situazioni ad esso analoghe che potrebbero verificarsi in altre imprese.
Il progetto NIIIP può essere visto, in questa prospettiva, come un progetto pilota per modelli di simulazione, in cui possano interagire tra loro imprese virtuali costruite secondo lo schema di jVE, che sarà analizzato nel dettaglio nei capitoli che seguono.

Capitolo 5 - Java Virtual Enterprise (jVE): un frame in Swarm

In questo capitolo si descrivono la struttura ed il codice del modello d'impresa virtuale jVE (Terna 2002) avviato nel corso dell'anno 2000 e successivamente sviluppato grazie anche ai contributi di alcuni tesisti della Facoltà di Economia di Torino. Il modello di jVE, descritto in queste pagine, è la base per il modello d'impresa virtuale applicato al caso concreto Vir che andremo ad analizzare nel capitolo che segue.

5.1 La struttura

I prodotti della *java Virtual Enterprise* sono rappresentati da una sequenza di numeri indicativi delle lavorazioni cui il prodotto deve essere sottoposto nel suo percorso di creazione. Ad ogni numero corrisponde l'unità produttiva, interna o esterna all'impresa, che sarà in grado di svolgere tale lavorazione.
Come evidenziato nella figura 1, le sequenze dei numeri sono contenuti nelle *ricette* di produzione, che possono essere di lunghezza variabile. La simulazione inizia con la generazione casuale degli ordini di produzione, le ricette, si ottiene così una serie di vettori di numeri casuali che simboleggiano i prodotti da realizzare e costituiscono l'insieme delle informazioni delle operazioni da compiere, che consentono all'impresa di individuare il percorso da seguire per completare la produzione.

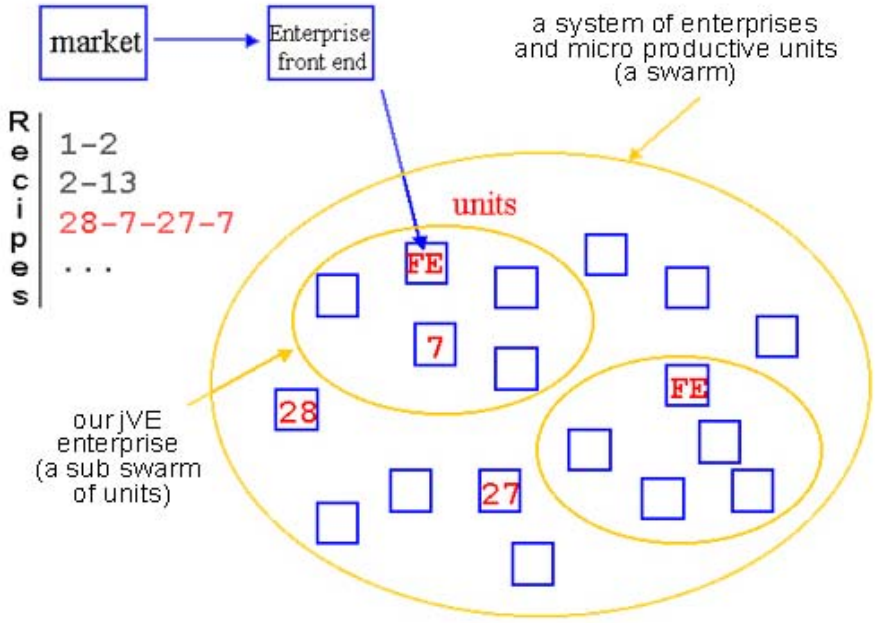


Figura 1: Schema di jVE (Terna 2002)

Gli ordini così generati, rappresentativi di quelli dei clienti della jVE, suono presi in consegna da un'unità interna all'impresa che svolge il compito di Front End e si occupa di smistare tali ordinativi tra le unità dell'impresa.

Le unità si distinguono tra loro in base al numero identificativo del tipo di lavorazione che sono in grado di svolgere, tutte le lavorazioni hanno la stessa durata.

La produzione dell'impresa virtuale jVE svolta dalle unità produttive consiste nell'apprendere e modificare le informazioni contenute nei vettori dell'ordine: le unità devono inserire l'informazione di completamento della proprio lavorazione ed individuare a quale unità spetta il processo produttivo seguente.

Il modello, in questa prima formalizzazione, non prevede la gestione di produzioni asincrone, la possibilità che le fasi di lavorazione abbiano durata diversa o che un'unità produttiva possa essere in grado di svolgere più di una fase di lavorazione.

Ogni unità produttiva è dotata di un proprio magazzino per raccogliere la produzione in eccesso, se attuata.
Qualora il livello delle scorte presenti sia inferiore al quantitativo minimo stabilito l'unità produce per il magazzino fino a raggiungere il livello massimo di scorte consentito.
L'introduzione di scorte di semilavorati è stata necessaria per ridurre le code nelle liste di attesa delle unità produttive, tuttavia un accumulo eccessivo di scorte può determinare problemi di spazio, per un'impresa reale, e un incremento di costi finanziari sia nella realtà che nella simulazione.
Ogni unità, infatti, è dotata di un sistema di contabilità. La unit computa per ogni ciclo produttivo un costo fisso ed un costo variabile nel seguente modo:

- i costi fissi vengono computati giornalmente, indipendentemente dall'attività svolta dall'unità;
- i costi variabili vengono computati quando l'unità produce per l'order o per il magazzino

In questo modo è possibile calcolare i costi giornalieri, fissi e variabili, dell'impresa ed i costi totali come somma di quelli giornalieri.
Ai magazzini viene imputato anche un costo finanziario, che dipende dalle quantità che giornalmente rimangono inutilizzate, sulla base di tre criteri:

1. a costi variabili;
2. a costi fissi più costi variabili;
3. con lo stesso criterio dei prodotti finiti.

La contabilità viene registrata anche dal lato degli ordini in termini di costi fissi e variabili calcolati nel momento del passaggio attraverso le unità. Seguendo questa metodologia i costi delle unità e degli ordini

vengono registrati nel medesimo momento in un'impresa che non faccia uso dei magazzini, altrimenti in due momenti separati.
Gli ordini forniscono anche l'informazione relativa ai ricavi ottenuti dall'impresa, calcolati come prodotto tra la lunghezza della ricetta ed un tasso di *ricarica* predefinito.
La contabilità degli ordini ci fornisce due importanti informazioni:

- quanto costa uno specifico bene;
- quanto si spende in totale per produrre;
- quanto l'impresa riesce a valorizzare in termini di prodotti finiti e di semilavorati.

Il modello, in questa formalizzazione, può funzionare in tre diverse versioni:

- senza i magazzini;
- con i magazzini;
- con i magazzini e le news.

Le news sono state introdotte per simulare la circolazione delle informazioni all'interno dell'impresa, nella versione del modello descritta in queste pagine, sono utilizzate unicamente per fornire indicazioni riguardanti l'order, il messaggio consiste nell'indicare alle unit con chi possono o devono comunicare per anticipare il processo produttivo.
Come mostra la figura 2 a seconda della struttura dell'azienda che si sceglie di simulare vi saranno risultati differenti. Le due situazioni estreme sono rappresentate dall'impresa senza magazzini e da quella con i magazzini, in cui il ritardo di produzione è rispettivamente più elevato e meno elevato. La soluzione con magazzini e news si pone all'interno della forbice formata dai due casi precedenti e costituisce una mediazione tra ritardare la produzione e creare oneri finanziari.

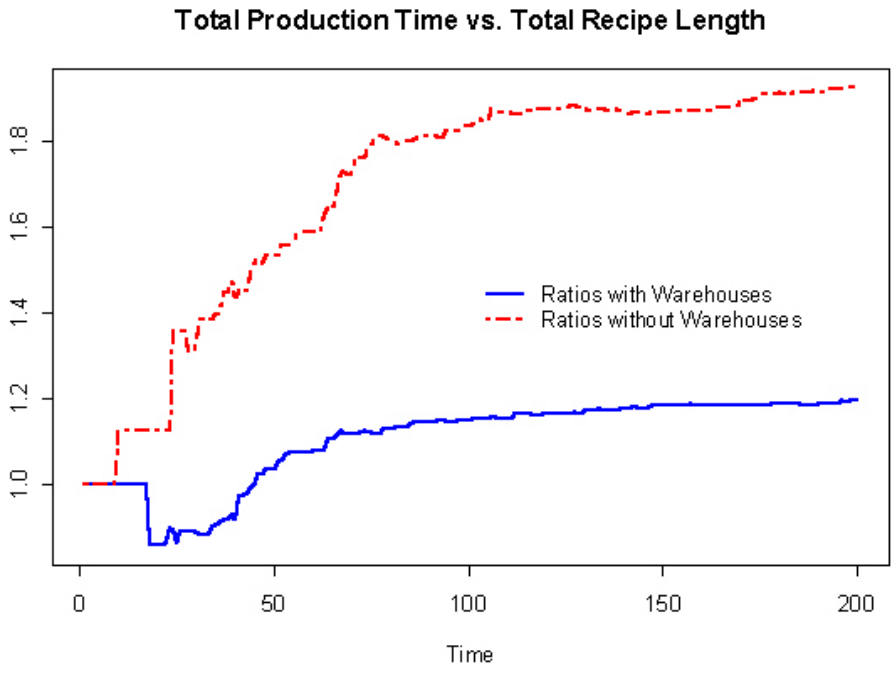


Figura 2: Ritardi nella produzione

Nel sistema aziendale simulato circolano:

- decisioni;
- beni;
- informazioni.

I flussi che identificano questi passaggi possono essere elaborati mediante tre matrici tra loro correlate (figura 3).

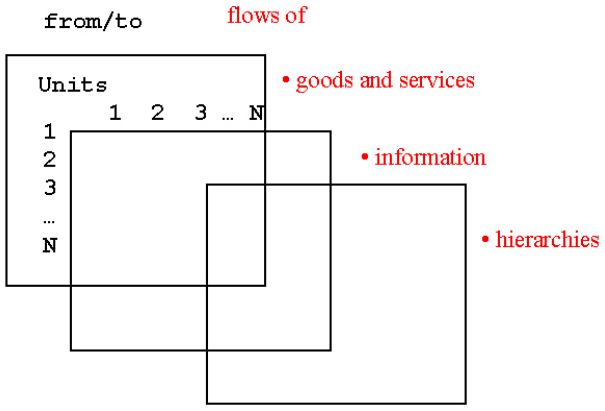


Figura 3: I flussi aziendali

In questa formalizzazione del modello gli spostamenti dell'ordine di produzione racchiudono i flussi dei beni ed in qualche misura quelli delle informazioni, mancano ancora le relazioni gerarchiche tra le unità.

5.2 Il codice

La versione che descriviamo in queste pagine è: jveframe-0.4.6.
Il modello si compone delle seguenti classi:

- StartVEFrame.java;
- VEFrameObserverSwarm.java;
- VEFrameModelSwarm.java;
- Ordergenerator.java;
- Order.java;
- Unit.java;
- InformationRuleMaster.java;
- Warehouse.java;
- InformationFlowMatrix.java;
- News.java;
- AccountingParameters.java;
- SwarmUtils.java;
- PTHistogram.java;

e di un file jveframe.scm.

Tutti gli elementi della simulazione sono rappresentati da oggetti generati dalle classi appena elencate, dalle quali ereditano tutte le caratteristiche e le capacità (metodi) di svolgere azioni.

Figura 4: Class Diagram di jVE

La programmazione ad oggetti, già analizzata nel capitolo precedente, consente, in fase di sviluppo del modello, di operare le modifiche necessarie alle classi ed ottenere che vengano ereditate da tutti gli oggetti che ad esse fanno riferimento.
Ogni oggetto può ereditare da più di una classe, rendendo possibile rappresentare diverse strutture aziendali senza stravolgere il modello.
Per meglio comprendere le relazioni tra le classi elencate può essere utile far riferimento al diagramma UML (figura 4), di cui parleremo nel capitolo 7.
La sequenza di generazione dell'ambiente di simulazione è tipica di ogni applicazione Swarm: si compone di una fase di creazione degli oggetti e di una di costruzione delle azioni.
Il programma ha inizio eseguendo la classe StartVEFrame.java contenente il metodo Main, che ingloba le azioni che il programma dovrà compiere permettendo l'avvio della simulazione.
Si procede, in questa fase, alla creazione di un'istanza dell'oggetto *VEFrameObserverSwarm* non tramite il metodo classico, che utilizza il metodo new, ma richiamando il file *jveframe.scm* che contiene i valori di default della simulazione e che possono essere modificati senza dover ricompilare l'intera classe ogni volta:

```
vEFrameObserverSwarm = (VEFrameObserverSwarm)
Globals.env.lispAppArchiver.getWithZone $key(Globals.env.globalZone,
"vEFrameObserverSwarm");
```

La simulazione inizia in questo punto con:

- il richiamo del metodo *buildobjects()* necessario per costruire tutte le parti di cui è composto l'*Observer*

```
vEFrameObserverSwarm.buildObjects ();
```

- il richiamo del metodo *buildActions()* necessario per determinare le azioni che svolgerà l'*Observer*

  vEFrameObserverSwarm.buildActions();

- il richiamo del metodo *activateIn()* necessario per attivare tutto ciò che è stato creato

  vEFrameObserverSwarm.activateIn(null);

- il richiamo del metodo go(), ereditato dalla classe *swarm.simtoolsgui.GUISwarmImpl*, con il compito di avviare tutte le attività e permettere all'utente di gestirle tramite il pannello di controllo

  vEFrameObserverSwarm.go();

- il richiamo del metodo *drop()* per terminare le attività

  vEFrameObserverSwarm.drop();

- *System.exit(0);* per chiudere i grafici e la simulazione.

L'Observer contiene il Model e gli oggetti grafici per la rappresentazione dell'andamento dell'impresa virtuale:

- *public int displayFrequency* definisce il numero di volte che i grafici devono essere aggiornati nel corso della simulazione;
- *public boolean verboseChoice* se vera consente di mostrare le descrizioni degli avvenimenti che si susseguono;

- *public ActionGroup displayActions* questa variabile di tipo ActionGroup farà da contenitore alla sequenza di eventi grafici;
- *public Schedule displaySchedule* contiene un esemplare della classe Schedule che servirà per gestire le sequenze temporali nell'Observer;
- *public VEFrameModelSwarm vEFrameModelSwarm* è lo Swarm che noi osserveremo, è un esemplare della classe VEFrameModelSwarm;
- *public PTHistogram ptHistogram* è una variabile di tipo PTHistogram che servirà a realizzare l'istogramma delle code presso le unità ed i magazzini, appartiene al progetto Ptolemy sviluppato dal Department of Electrical Engeneering and Computer Science, University of California Berkeley (http://ptolemy.eecs.berkeley.edu).

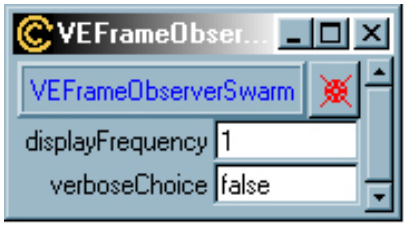


Figura 5: Pannello parametri Observer

Si creano ora gli oggetti necessari per rappresentare a video il Model ed effettuarne le rilevazioni:

```
public Object buildObjects () {
```

quindi si creano in sequenza:

- il vEFrameModelSwarm

```
vEFrameModelSwarm =
(VEFrameModelSwarm)Globals.env.lispAppArchiver.getWithZone$key( getZone(),
"vEFrameModelSwarm");
```

- e le sonde, per l'analisi e la gestione dei parametri della simulazione, sia del vEFrameModelSwarm sia del vEFrameObserverSwarm;

getControlPanel ().setStateStopped (); comunica al pannello di controllo di attivare per la prima volta il comando di Stop: la simulazione sarà avviata dall'utente, una volta valutati i parametri da utilizzare, attraverso i comandi rappresentati nella figura 6.

Figura 6: Comandi per avviare e interrompere la simulazione

StartVEFrame.verbose=verboseChoice; comunica allo StartVEFrame la decisione se mostrare tutte le descrizioni degli avvenimenti o meno
A questo punto il vEFrameModelSwarm costruisce i propri oggetti.
Al termine del metodo VEFrameObserverSwarm.buildObjects() si sono descritti sia gli oggetti necessari per avviare il modello sia quelli necessari ad osservarlo.
Occorre ora creare all'interno del VEFrameObserverSwarm le azioni per avviare la simulazione.
Una volta definite le azioni da svolgere e creato lo Schedule si inserisce un'istanza delle azioni nell'istanza dello Schedule medesimo:

```
displaySchedule.at$createAction (0, displayActions);
return this
```

in tale modo le azioni saranno svolte come un tutt'uno dal gestore d'eventi.

Il VEFrameModelSwarm contiene le unità e tutti i relativi strumenti utilizzabili, come ad esempio i magazzini, necessari per dialogare con gli inventari, o le notizie.
E' fatta la dichiarazione di tipo di variabile per quelli che saranno tutti i parametri della simulazione:

- *public int totalUnitNumber*

il numero totale di unit che utilizziamo nella simulazione;

- *public int maxStepNumber*

il numero massimo di passi da eseguire per completare un ordine

- *public int maxInWarehouses*

la quantità massima di scorte presenti in un magazzino

- *public int minInWarehouses*

la quantità minima di scorte presenti in un magazzino

- *public boolean useWarehouses*

la scelta se usare (true) oppure no i magazzini e le scorte

- *public boolean useNewses*

la scelta se usare (true) oppure no i magazzini e le scorte

- *public int infDeepness*

quanto in profondità l'informazione è propagata (quanti passi esaminiamo dopo la fase di produzione corrente)

- *public float revenuePerEachRecipeStep*

la stima di ricchezza dell'impresa ad ogni passo della ricetta;

- *public float inventoryFinancialRate*

il tasso annuo utilizzato per valutare il costo delle scorte;

- *public float enterpriseBenefit*

il guadagno dell'impresa;

- *public int nOfNewsesToProduce*

il numero di notizie necessarie per decidere di produrre scorte;

- *public int nOfNewsesToBeCleared*

il numero di notizie che devono essere cancellate dopo la decisione di incrementare le scorte;

- *public int nOfOrdersInNewses*

il numero d'ordini per i quali le notizie sono propagate da un'unità

- *public int inventoryEvaluationCriterion*

il criterio di valutazione delle scorte;

- *public int totalProductionTime*

il tempo totale di produzione richiesto dall'ordine terminato ;

- *public int totalRecipeLength*

la lunghezza totale della ricetta dell'ordine terminato;

- *public OrderGenerator orderGenerator*

il creatore d'ordini, l'orderGenerator, che per il momento opera con una selezione casuale sia della lunghezza sia del contenuto delle ricette;

- *public ActionGroup modelActions*

l'oggetto di tipo ActionGroup necessario per mantenere una sequenza ordina degli eventi;

- *public Schedule modelSchedule*

lo Schedule operante nel VEFrameModelSwarm;

- *public InventoryRuleMaster inventoryRuleMaster*

il gestore di regole che controlla le scorte;

- *public InformationRuleMaster informationRuleMaster*

il gestore del flusso d'informazioni (notizie);

- *public AccountingParameters accountingParameters*

la variabile necessaria per il conteggio di costi fissi e variabili;

- *public ListImpl unitList*

la lista indicante le unità produttive attivate

- *public ListImpl warehouseList;*

la lista indicante i magazzini attivati

- *public ListImpl orderList;*

l'intera lista degli ordini

Il costruttore della classe VEFrameModelSwarm ha il compito di definire quali siano i principali parametri da passare all'oggetto quando questo è creato.
Successivamente vengono inizializzate le variabili della simulazione con i valori di default uguali a quelli contenuti nel file jveframe.scm, che verranno visualizzati nel pannello (figura 5)
La costruzione vera e propria del VEFrameModelSwarm ha inizio in questo punto con la definizione di quali siano gli oggetti che determinano il "comportamento" della classe: cosa dovrà fare quando sarà attivata.
Si impostano le seguenti attività:

- creazione di un'istanza della classe AccountingParameters che contiene i metodi relativi alla gestione della contabilità dei costi fissi e variabili;

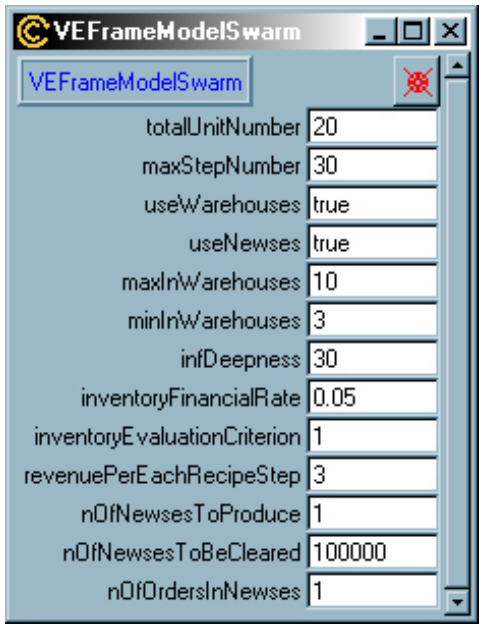


Figura 7: Pannello con i parametri della simulazione

- lettura tramite i metodi propri della classe AccountingParameters dei costi fissi e variabili dai file FixedCosts.xt e VarialeCosts.txt
- controllo del limite massimo di scorte possibili in magazzino, se pari a 0 allora verrà attribuito il massimo valore integer possibile;
- creazione di un'istanza della classe ListImpl che registrerà le unità (il numero assegnato alle unità) attivate;
- creazione di un'istanza della classe ListImpl che registrerà i magazzini assegnati alle unità attivate;
- creazione di un'istanza della classe ListImpl che registrerà gli ordini esistenti ad ogni passo di simulazione;
- creazione di un'istanza della classe OrderGenerator che creerà una ricetta di lavorazioni, scegliendo casualmente sia la lunghezza sia il contenuto;
- creazione di un'istanza della classe InventoryRuleMaster che è il gestore delle regole per le scorte
- creazione di un'istanza della classe InformationRuleMaster che è il gestore delle regole per il flusso d'informazioni;
- con l'utilizzo di un ciclo iterativo sono create tante unit quanto è il valore di totalUnitNumber e se attivati anche i magazzini relativi;

Il passo successivo è quello di creare un ActionGroup per gestire le azioni necessarie, in particolare si tratta delle azioni definite nelle classi Unit ed OrderGenerator:

- UnitStep1;
- UnitStep2.

UnitStep1 rappresenta il momento della produzione giornaliera, la unit controlla la lista degli order in attesa, se è piena effettua la lavorazione sul primo order della lista, utilizzando, se disponibili, le scorte, altrimenti interroga l'Inventory Rule Master circa l'opportunità di produrre per il magazzino.
In UnitStep2 la unit controlla il contenuto della propria waitingList ed invia le informazioni riguardanti il primo order che subirà la lavorazione alle unità successive interrogando l'Information Rule Master per verificare se sia possibile effettuare la comunicazione.
Se invece l'order è completo è possibile dichiarare terminata la sua produzione.
Una volta che abbiamo definito le azioni da svolgere e che abbiamo creato lo Schedule l'avvio del tempo si avrà in zero considerando tutto ciò che compone modelActions, ottenendo così una parvenza di gestione parallela degli eventi in realtà definiti in sequenza.
VEFrameObserverSwarm, Model e schedule saranno attivati all'interno dell'ambiente dell'Observer.
Sono elencati di seguito alcuni dei metodi utilizzati:

- getUnitList()

restituisce il contenuto della lista delle unità;

- getWarehouseList ()

restituisce il contenuto della lista dei magazzini;

- getOrderList()

restituisce il contenuto della lista degli ordini;

- getUseWarehouses()

restituisce vero se i magazzini sono usati e falso nel caso opposto;

- setProductionTimeAndRecipeLength(int pt, int rl)

registra il tempo totale di produzione e la lunghezza della ricetta. Richiede, come argomenti, il valore intero sia del tempo di produzione sia della lunghezza dell'ordine eseguito

- getTimeLengthRatio()

restituisce totalProductionTime/totalRecipeLength se totalRecipeLength è diverso da 0 altrimenti restituisce 1, valore questo considerato un buon compromesso essendo 0 la lunghezza totale della ricetta nel primo giorno di simulazione ed avendo comunque impiegato una unità di tempo per concludere il primo ordine

- getInventoryFinancialRate()

restituisce il tasso finanziario da applicare alle scorte

- getInventoryEvaluationCriterion()

restituisce il valore scelto per la valutazione delle scorte

- getRevenuePerEachRecipeStep()

restituisce il valore della ricchezza d'ogni passo della ricetta

- getEnterpriseBenefit()

restituisce il profitto dell'impresa come differenza tra:

- la sommatoria dei ricavi dovuti ai prodotti finiti e quelli derivanti dai semilavorati e dalle rimanenze, nel caso di attivazione dei magazzini;
- la sommatoria dei costi fissi e variabili e, qualora fossero attivati i magazzini, quelli derivanti dal costo delle scorte

L'OrderGenerator è utilizzato per simulare il mercato, per il momento gli ordini da evadere sono una generazione casuale sia nei contenuti sia nella lunghezza delle ricette, il cui limite massimo è rappresentato dalla quantità di Unit presenti.
Le variabili d'istanza sono:

- public int maxStepNumber;

il numero massimo di passi presenti nella ricetta

- public int orderCount;

utilizzato per registrare il numero d'ordini generati

- public Order anOrder;

un ordine

- public ListImpl unitList;

la lista contenente tutte le unità

- public ListImpl orderList;

la lista contenente tutti gli ordini

- VEFrameModelSwarm vEFrameModelSwarm;

Il Model cui si applica il guadagno derivante dall'ordine

- public int[] orderRecipe;

il vettore contenente una ricetta.

La classe Order contiene i prodotti dell'impresa virutale che sono lavorati dalle unità: gli oggetti della classe Unit.java.
Per poter elaborare il caso con e senza magazzini e poter gestire la presenza o meno delle news la classe Unit è dotata di più costruttori, questo è consentito dal cosiddetto polimorfismo delle classi java, descritto nel capitolo precedente.
La giornata virtuale di produzione si articola in Step 1 e Step 2, già analizzati in queste pagine.

La classe AccountingParameters è utilizzata per la lettura dei valori dei costi. Si serve del metodo:

```
public void readFixedCosts(){
```

che legge i dati presenti nel file *CostParameters\fixedCosts.txt* e li utilizza al fine del calcolo dei costi fissi.
La classe *BufferedReader* velocizza lo streaming dei dati da disco aprendo un contenitore temporaneo in cui i dati vengono depositati a blocchi dopo esser stati letti dal'oggetto della classe *FileReader*

La classe *StringTokenizer* ci permette di assegnare a t tutta la riga letta sapendo già che l'oggetto della classe *StringTokenizer* sarà in grado di leggere tra uno spazio e l'altro. Successivamente potremo effettuare parsing per trasformare tale valore in un valore di tipo float da utilizzare per le nostre rilevazioni.
In particolare il file in questione contiene una colonna di soli numeri rappresentativi dei costi fissi delle singole unità.

```
StringTokenizer t;
        for(i = 0; i < fixedCosts.length; i++){
                String line = fileInputFixedCosts.readLine();
                t = new StringTokenizer(line, " ");
                fixedCosts[i] = Float.parseFloat(t.nextToken());
```

La classe Wharehouse è necessaria al fine dell'utilizzo sia dei magazzini sia delle scorte.
La classe InventoryRuleMaster è qui utilizzata per prendere decisioni riguardanti la gestione delle scorte.
Se il livello delle scorte è inferiore al massimo ed è presente una News che segnala l'arrivo di un passo di produzione, oppure se il livello delle scorte è inferiore al minimo, il metodo seguente risponderà in modo affermativo alla chiamata della Unit, la quale vuole sapere se dovrà produrre per il magazzino.

```
public boolean increaseInventoriesUsingNewses(int un, Warehouse aw, ListImpl nl){
```

La classe News contiene la descrizione di un oggetto necessario per trasmettere le informazioni riguardanti i passi successivi nella sequenza di produzione di un Order.
La classe InformationRuleMaster è utilizzata per gestire il flusso dell'informazione all'interno della impresa virtuale.

Le informazioni contenute nel file *informationFlowMatrix.txt* sono lette ed assegnate ai singoli elementi della matrice *informationFlowMatrix[][]*. I dati sono raccolti come descritto per la classe *AccountingParameters.java* e sono rappresentati da valori 1 (il flusso dell'informazione scorre) oppure da valori 0 (l'informazione si ferma).

La classe SwarmUtils è un estratto da: *jSIMPLEBUG: a Swarm tutorial for Java Charles P. Staelin Smith College Northampton, MA April 2000 based on ObjectiveC code and original text by Chris Langton & Swarm development team Santa Fe Institute, NM.*

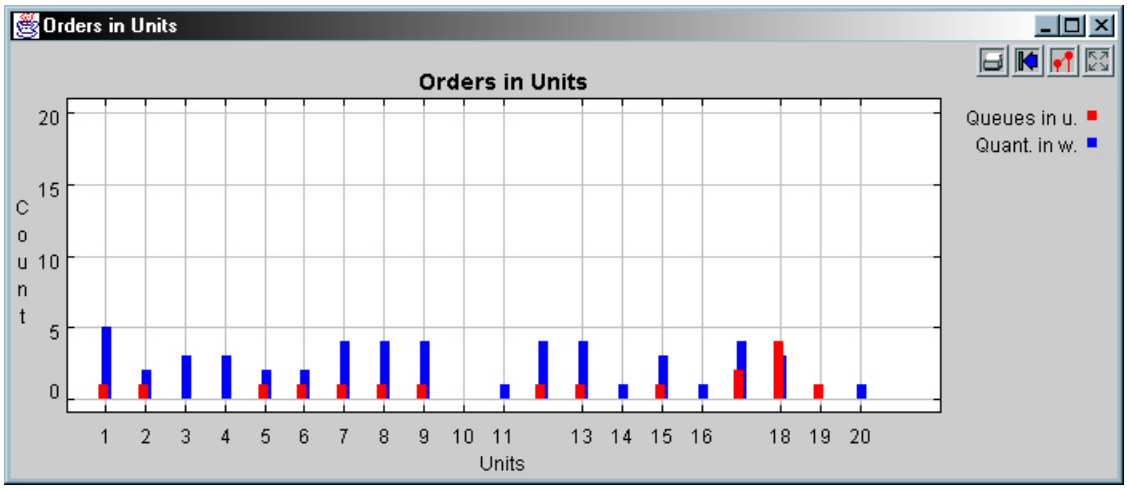


Figura 8: Istogrammi relativi alle code presso le unità e le quantità nei magazzini

L'ultima classe, PTHistogram, contiene tre costruttori per disegnare i grafici degli istogrammi (figura 8) con le code di produzione presso le singole Unit e il contenuto dei magazzini.
Il primo costruttore considera la versione semplice del modello di impresa virtuale senza magazzini, il secondo la versione con i magazzini ed il terzo è stato progettato in previsione di un futuro sviluppo del modello.

Capitolo 6 - Il modello di impresa virtuale VIR

Il presente capitolo ripercorre il processo di generalizzazione per l'applicazione ad un caso concreto del modello *javaSwarm* di simulazione d'impresa, descritto nel capitolo precedente.
Con la cooperazione con un'azienda reale ci si propone di studiare l'impresa come sistema economico, valutare la bontà del modello jVE in corso di sviluppo e la sua capacità di fotografare la realtà aziendale presa in considerazione. L'impresa, da parte sua, può trarne un'occasione di studio per osservare la sua struttura organizzativa da una prospettiva inedita e la possibilità di valutare, attraverso la simulazione, gli effetti che decisioni di innovazioni di processo potrebbero avere sulla sua struttura organizzativa.

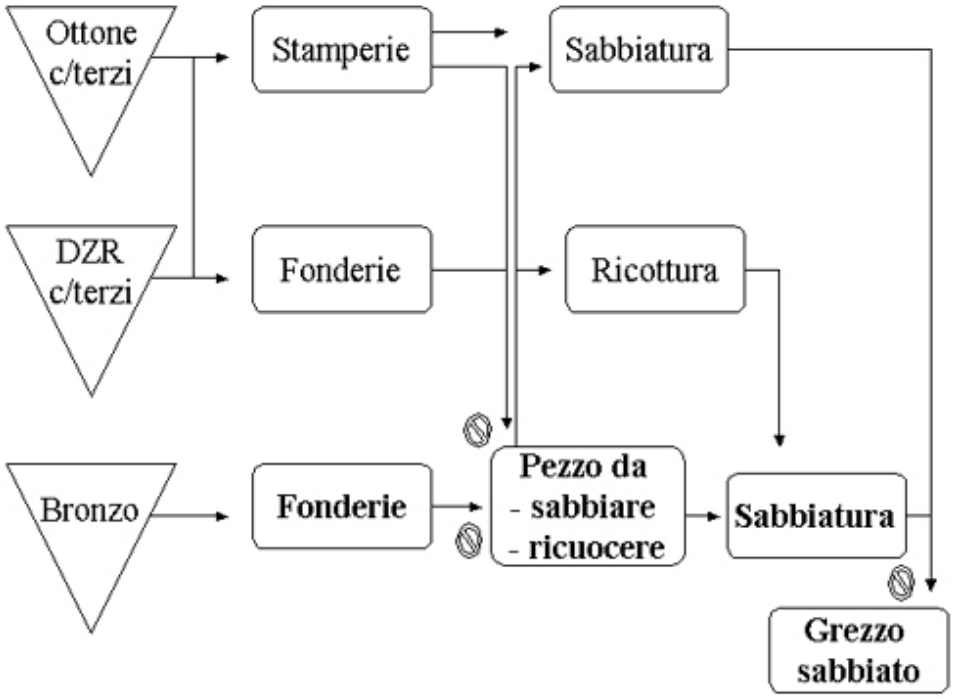
6.1 Descrizione della realtà Vir

La Vir S.p.a. è un'azienda meccanica produttrice di organi di intercettazione e regolazione di fluidi riconducibili a due famiglie principali: valvole a sfera e saracinesche con dimensioni comprese fra otto millimetri e un centimetro. L' 88% del suo fatturato è relativo alla produzione in ottone, il restante 12% è equamente ripartito fra bronzo e plastica. Gli stabilimenti dell'azienda sono situati a Valduggia, polo piemontese che insieme a quello bresciano rappresenta circa la totalità della produzione di valvole e rubinetterie italiane. Il centro è dotato di un reparto di torneria per le lavorazioni meccaniche di asportazione del

truciolo composto da quattordici macchine e di un reparto di montaggio con quaranta macchinari di cui dodici di prodotto finito.

La produzione avviene su larga scala e le macchine sono altamente automatizzate, alcune postazioni rimangono però manuali a seguito delle difficoltà di progettazione di macchinari in grado far fronte alla complessità di assemblaggio di prodotti in continua evoluzione. Se i macchinari sono infatti versatili, in quanto in grado di produrre articoli molto diversi tra loro, sono anche molto poco flessibili in quanto sono lenti i passaggi da una tipologia a un'altra. Le macchine sono caratterizzate da un braccetto universale pronto a ricevere l'attrezzatura e da un supporto con il porta pezzo.

Figura 1: Fase I



A seconda della tipologia e delle dimensioni della componente da lavorare

si modifica la pinza di graffaggio e l'utensile che esegue la lavorazione.
Le fasi di lavorazione dei prodotti sono essenzialmente tre e sono comuni alle due famiglie principali.
Nella prima fase, descritta in figura 1, 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.

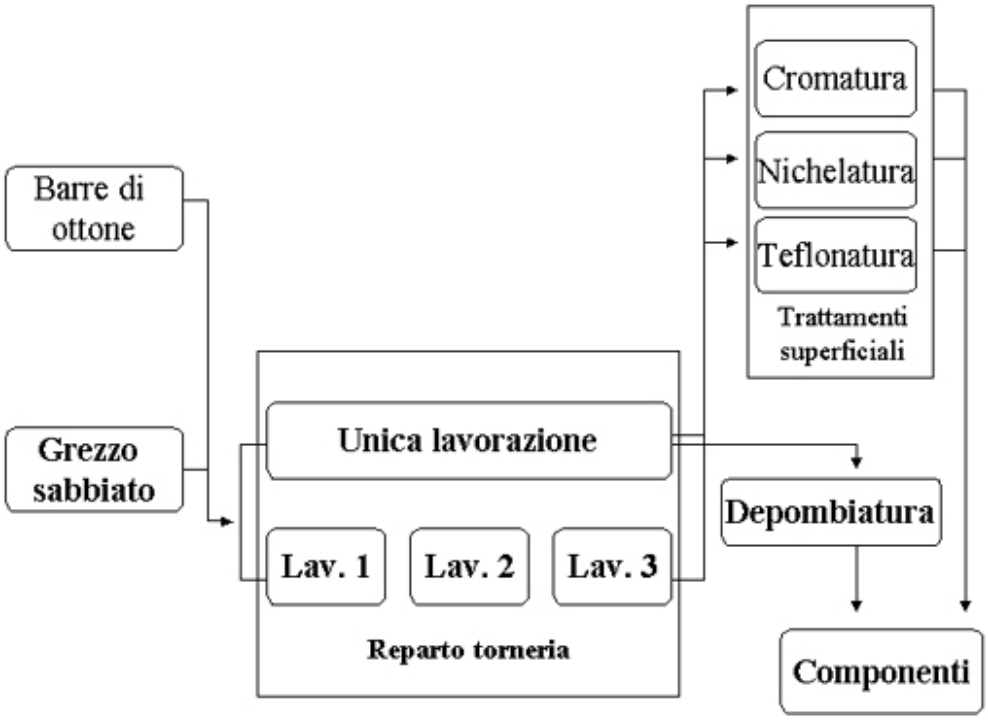


Figura 2: Fase II

Sono indicate in grassetto le lavorazioni che vengono svolte internamente all'impresa. Il simbolo che compare accanto ad alcune fasi di lavorazione indica il processo di controllo e di pesatura che viene effettuato sui pezzi da lavorare.
La seconda fase (figura 2) riguarda le lavorazioni meccaniche sui pezzi

grezzi che interessano il reparto della torneria e i trattamenti superficiali che devono essere svolti esternamente all'azienda. In questa seconda fase il magazzino delle componenti si riempie delle parti lavorate nella fase 2 e di quelle acquistate esternamente che possono richiedere, come descritto in figura 3, la verniciatura o un preassemblaggio, entrambi svolti esternamente all'impresa.

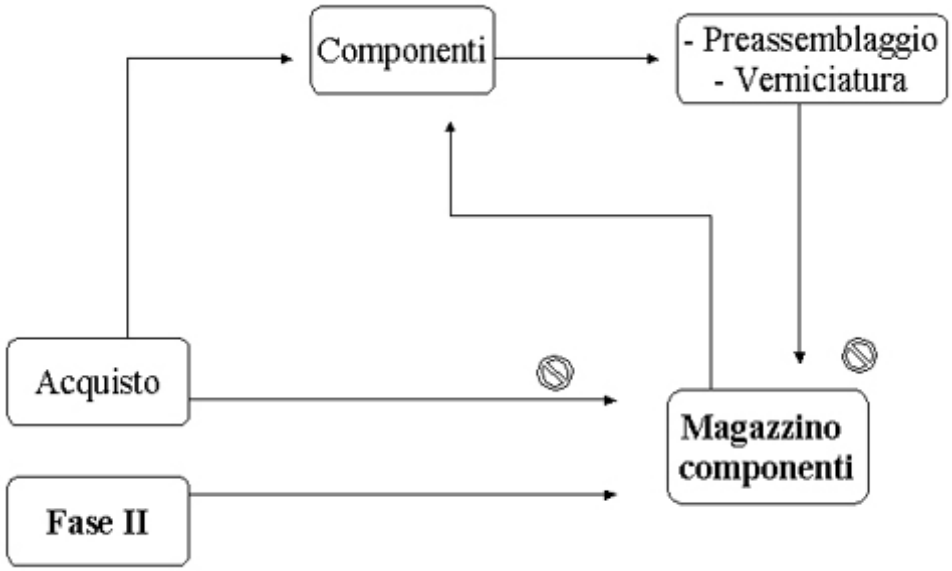


Figura 3. Fase II b

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 ed imballato.

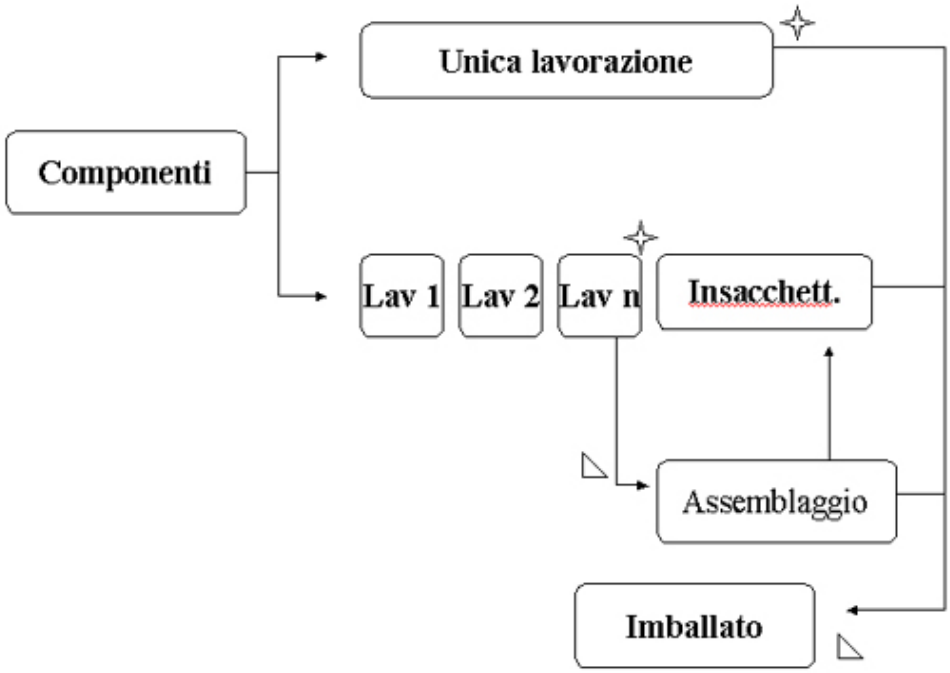


Figura 4: Fase III

Come mostra la figura 4 il montaggio può essere eseguito da un unico macchinario automatizzato oppure da più macchinari posti in sequenza. Nel primo caso il collaudo (indicato in figura 4 con una stella) avviene in linea, cioè all'ultima stazione prima dello scarico. I pezzi assemblati in questa fase vengono quindi pesati (indicato in figura 4 con un triangolo) e poi imballati.

Le sofisticate apparecchiature per l'assemblaggio ed il collaudo sono progettate e realizzate dalla VIR stessa, che in questo modo è riuscita ad ottenere impianti che si adattassero perfettamente alle esigenze di produzione e di spazio proprie dell'azienda, senza dover ricorrere ad esborsi di capitale eccessivamente onerosi.

All'interno dello stabilimento si collocano anche gli uffici dei progettisti e il

centro decisionale che stabilisce le tempistiche e modalità di produzione, il vero "cuore" della VIR, che deve mettere alla prova lo schema generale del modello di impresa virtuale.
Come mostra lo schema in figura 5, la Vir è dotata di tre magazzini, due per le componenti e uno per il prodotto finito.

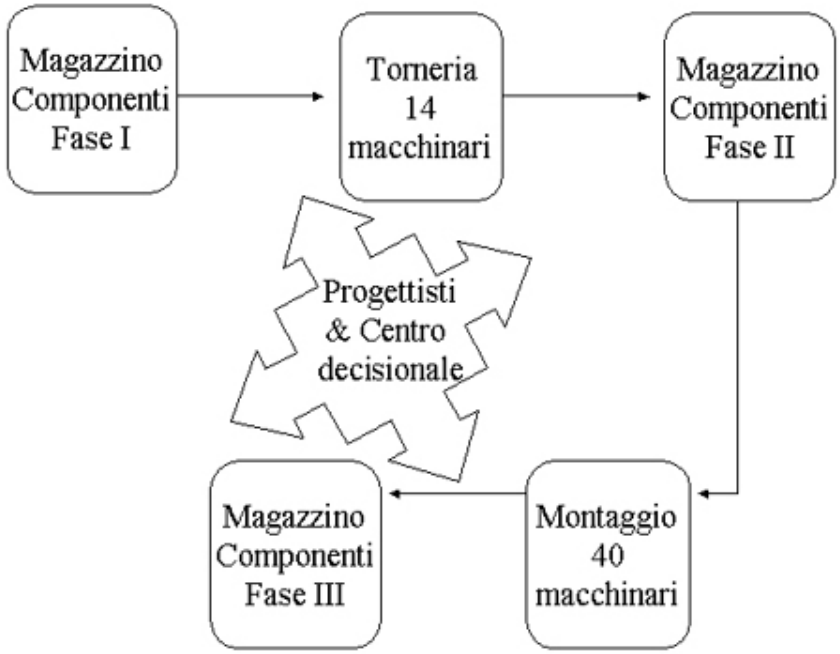


Figura 5: Struttura dello stabilimento della Vir

Il terzo magazzino è destinato ai prodotti imballati in attesa di spedizione, la Vir, per ragioni finanziarie e di gestione degli spazi non mantiene un livello di scorte di prodotti finiti.

Alcune componenti, comuni a più prodotti, hanno un livello di scorte superiore rispetto ad altre e sono prodotte ogni qual volta i macchinari ad esse dedicati siano poco attivati dalle necessità di produzione. In generale, però, i magazzini delle componenti sono alimentati principalmente

all'inizio di ogni mese di un quantitativo stimato sufficiente per poter far fronte agli ordini che saranno evasi nella mensilità in questione. L'andamento del livello delle scorte per le componenti tende ad oscillare, come descritto in figura 6, intorno ad un valore medio fissato mensilmente.

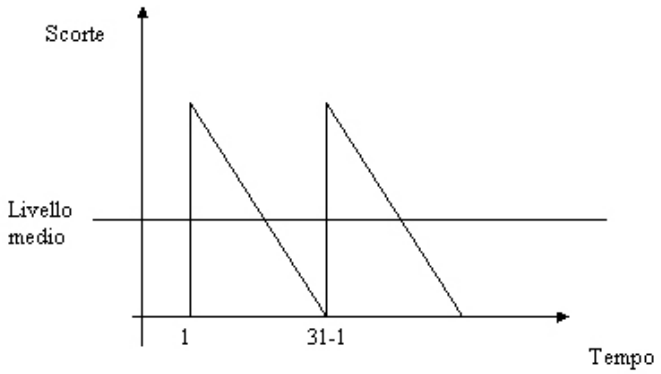


Figura 6: Andamento del livello delle scorte per le componenti

All'inizio di ogni mese viene stilato anche il programma di produzione sulla base degli ordini attesi e di quelli effettivamente ricevuti dall'ufficio commerciale.

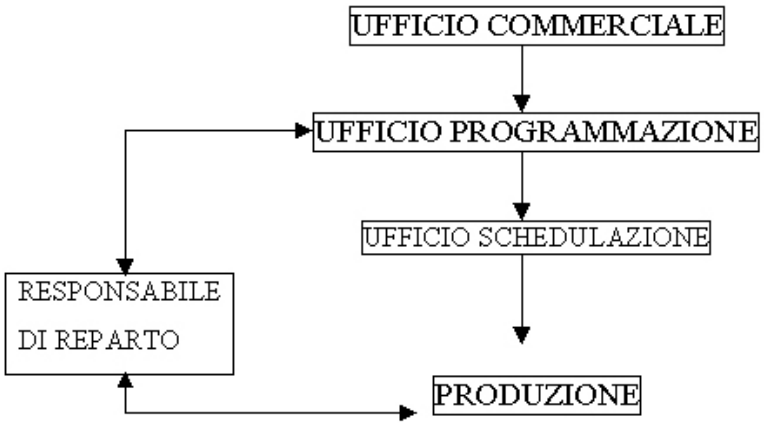


Figura 7: Relazioni tra gli uffici

Il programma di produzione così organizzato viene quotidianamente aggiornato sulla base delle effettive risorse disponibili e degli ordini ricevuti dall'ufficio commerciale che si possono suddividere in tre principali categorie:

1. ordini che possono essere evasi nei tempi stabiliti con il cliente anche iniziando da zero la lavorazione;

2. ordini che si possono evadere immediatamente perché vi è disponibilità di prodotti finiti grazie a previsioni stimate correttamente;
3. ordini basati su richieste che non potrebbero essere evase nei tempi stabiliti con il cliente se non si fosse prevista in anticipo una produzione di componenti che richiedono tempi particolarmente lunghi.

La correttezza delle previsioni e i tempestivi adattamenti della produzione giornaliera dipendono dal grado di comunicazione e cooperazione tra gli uffici dell'azienda, le cui relazioni sono illustrate in figura 7.

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

1. emissione dell'ordine di lavoro verso il reparto di assemblaggio;
2. il reparto di assemblaggio richiede le componenti da montare;
3. controllo dei magazzini componenti.

   qualora il magazzino componenti fosse sprovvisto degli elementi richiesti si procede:

   a. all'acquisto all'esterno di talune parti del prodotto;
   b. all'emissione di un ordine di produzione alla torneria (manicotto, sfera e corpo per le valvole a sfera, corpo cuneo e vitone per le saracinesche);

4. la torneria controlla la disponibilità di pezzi grezzi da lavorare,
5. si acquistano all'esterno pani e barre di ottone.

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.

Figura 8: Valvola a sfera

Nel reparto torneria sono coinvolti tutti i macchinari (indicati con le sigle utilizzate in azienda):
A1, A2, A3, A4, B1, B3, B4, B5, C1, C2, C3, C4, C5, D1.
Gli articoli descritti non coinvolgono, invece, tutte le quaranta macchine del montaggio; quelle interessate sono:
P6, S1, P1, L1, L5, L6, O4, O5, L4b, L4

Le valvole a sfera (figura 8) si compongono di dieci parti: il corpo, il manicotto e la sfera sono lavorati internamente all'azienda.

Il dettaglio di alcuni articoli:

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. ipotesi A
        1. C5
    2. ipotesi B
        1. B1
    3. ipotesi C
        1. C1
    4. ipotesi D
        1. C2
    5. ipotesi 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. ipotesi A
        1. A4
    2. ipotesi B
        1. B5
    3. ipotesi 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. ipotesi 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. ipotesi A
    1. S1 unico percorso di montaggio (dalle componenti alla scatola confezionata);
    2. deposito nel magazzino prodotti finiti.
2. ipotesi B
    1. P6 dalle componenti al prodotto finito;
    2. "STAZIONE DI LAVORO" insachettamento ed inscatolamento manuale;
    3. deposito nel magazzino prodotti finiti.
3. ipotesi 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. ipotesi A
        1. C5
    2. ipotesi B
        1. B1
    3. ipotesi C
        1. C1
    4. ipotesi D
        1. C2
    5. ipotesi 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. ipotesi A
        1. A4
    2. ipotesi B
        1. B5
    3. ipotesi 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. ipotesi A
        1. D1

2. ipotesi 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. ipotesi A
    1. S1 unico percorso di montaggio (dalle componenti alla scatola confezionata);
    2. deposito nel magazzino prodotti finiti.
2. ipotesi B
    1. P6 dalle componenti al prodotto finito;
    2. "STAZIONE DI LAVORO" insacchettamento ed inscatolamento manuale;
    3. deposito nel magazzino prodotti finiti.
3. ipotesi C
    1. P1 seggio nel corpo;

2. L1 seggio nel manicotto;
3. L5 corpo + manicotto + sfera ;
4. L6 assemblaggio finale;
5. "STAZIONE DI LAVORO" insacchettamento 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. ipotesi A
        1. A2
    2. ipotesi 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. ipotesi A
        1. A4
    2. ipotesi B
        1. B3
5. cromatura all'esterno;
6. deposito nel magazzino componenti;

<u>sfera</u>

1. acquisto delle barre di ottone;
2. lavorazione meccanica per asportazione del truciolo:
    1. ipotesi B
        1. C4
3. cromatura all'esterno;
4. deposito nel magazzino componenti;

<u>bullone, maniglia, premistoppa, guarnizione premistoppa, perno, 2 seggi</u>

1. Acquisto all'esterno;
2. Verniciatura all'esterno;

3. deposito nel magazzino componenti;

Assemblaggio:

1. ipotesi A
    1. L4b seggio nel corpo,seggio nel manicotto,corpo + manicotto + sfera ;
    2. L4 assemblaggio finale;
    3. "STAZIONE DI LAVORO" insacchettamento ed inscatolamento manuale;
    4. deposito nel magazzino prodotti finiti.

famiglia 342

3/4 pollice

<u>corpo</u>

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

1. C1 prima fase;
2. C2 seconda fase;

2. ipotesi B
    1. C2 att2 prima fase;
    2. C2 seconda fase;

5. cromatura all'esterno;
6. deposito nel magazzino componenti;

<u>manicotto</u>

1. acquisto dei pani di ottone;
2. stampaggio all'esterno;
3. sabbiatura all'esterno;
4. lavorazione meccanica per asportazione del truciolo:
    1. ipotesi A
        1. A4
    2. ipotesi B
        1. B5
    3. ipotesi 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. ipotesi A
        1. D1
    2. ipotesi 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. ipotesi A
    1. P1 seggio nel corpo;
    2. L1 seggio nel manicotto;
    3. L5 corpo + manicotto + sfera ;

4. L6 assemblaggio finale;
5. "STAZIONE DI LAVORO" insacchettamento ed inscatolamento manuale;
6. deposito nel magazzino prodotti finiti.

Analogamente a quanto visto per le valvole a sfera, anche le saracinesche (figura 9) sono composte da dieci parti, ma soltanto il corpo, il cuneo e il vitone sono prodotti internamente.
Come per le valvole a sfera analizziamo nel dettaglio le fasi di produzione di alcuni articoli.

Figura 9: Saracinesca

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. ipotesi 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. ipotesi A
        1. A4
    2. ipotesi 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. ipotesi 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. ipotesi A
    1. O4
    2. O5
    3. "STAZIONE DI LAVORO" insacchettamento 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. ipotesi 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. ipotesi A
        1. A4
    2. ipotesi 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. ipotesi 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. ipotesi A
    1. O4 att2
    2. O5 att2
    3. "STAZIONE DI LAVORO" insacchettamento 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, ma 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 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), la necessità di essere seguite dal personale.
Sia le assunzioni sia 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 molte macchine.
Al contrario della torneria, nel 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

Nel reparto di torneria le macchine sono sfruttate a pieno regime, mentre nel reparto di montaggio sono utilizzate soltanto parzialmente.

Per quanto concerne la contabilità, in Vir i costi sono calcolati per processo: si determinano i costi per ogni singola fase di produzione ed i prodotti acquistano un costo pari al tempo di transito.

Si utilizza una contabilità per *centri di costo,* sono presenti centri di produzione e centri ausiliari. I costi dei centri ausiliari vengono poi "ribaltati" ai centri di produzione.

Centri di costo:

- Manodopera
- Impiegati, Dirigenti, Tecnici
- Attrezzature e Macchine
- Energia
- Combustibili

Calcolo dei singoli costi:

- Personale: si attribuiscono direttamente le ore lavorate sul centro considerato.

Salari / Ore Totali = €/h * h Centro = Costo del Personale del Centro

- Impiegati: i costi sono valutati in base al valore del centro.
- Ammortamenti: si individua la quota annuale.

Quota Annua / gg Lavorativi = €/g Costo Fisso di giorno lavorativo
€/g * gg del mese = Costo mensile d'ammortamento

- Energia: calcolato in base alla potenza e ad un coefficiente di

utilizzo.

- Combustibili: in base ai metri cubi.

Il costo totale dei centri viene imputato ai 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 determina un indice di costo. Quest'ultimo viene moltiplicato per il tempo standard del prodotto e sommando il valore dei materiali, attribuiti ai prodotti in base all'utilizzo, si arriva al costo totale del prodotto.

Pezzi * Tempi Standard Uomo/Macchina = α

Costo totale Centro / α = Indice di Costo

Indice di Costo * Tempo Standard di Prodotto = Costo di Prodotto

Costo di Prodotto + Materiali = Totale Costo del Prodotto

6.2 Astrazione per jVE

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 scendere ad 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. Le simulazioni sono realtà artificiale (Parisi 2001) che della realtà cercano di cogliere l'essenziale al di sotto della complicatezza e della varietà dei fenomeni osservabili. La complessità nella simulazione dell'impresa analizzata emergerà dal simultaneo operare delle sequenze produttive descritte e delle capacità produttive individuate.

6.2.1 Le ricette (What to Do - WD)

Come descritto nel capitolo precedente, in jVE i prodotti sono identificati da una sequenza di numeri corrispondenti alle fasi di lavorazione per realizzarli, questa sequenza viene generata in modo casuale e costituisce la *ricetta* per la produzione di ogni singolo prodotto.
Per applicare jVE alla realtà Vir, è stato necessario individuare prodotti rappresentativi che permettessero di ragionare sui volumi reali di produzione dell'azienda, limitando, però, la varietà del campionario.
Si è scelto di simulare la produzione delle due famiglie principali: valvole a sfera e saracinesche differenziandole entrambe per due sole dimensioni, grandi e piccole. Come accennato in precedenza, i due prodotti

considerati si compongono ciascuno di dieci componenti, in parte acquistate esternamente e in parte lavorate all'interno dell'impresa. Per simulare la produzione di ciascuno dei componenti si è dovuta ricostruire la sequenza di lavorazioni cui è sottoposto ogni pezzo grezzo nel corso delle tre fasi sopra descritte ed assegnare a ciascuna di esse un numero.
La sequenza dei numeri così ottenuti costituisce la *ricetta* della componente, non più generata in modo casuale, ma coerente con il suo reale processo produttivo.
Nella prima formalizzazione il dizionario delle fasi di lavorazione era strutturato nel modo seguente:
Valvole a sfera

Corpo
1. stampaggio all'esterno;
2. sabbiatura all'esterno;
3. ricottura esterna;
100. sabbiatura interna;
4. cromatura all'esterno;
5. teflonatura all'esterno;
6. nichelatura all'esterno;

Manicotto
1. stampaggio all'esterno;
7. sabbiatura all'esterno;
8. ricottura esterna;
101. sabbiatura interna;
9. cromatura all'esterno
10. teflonatura all'esterno;

11. nichelatura all'esterno;

Sfera

12. sabbiatura all'esterno;
102. sabbiatura interna;
13. cromatura all'esterno

Parti

14. bullone;
15. maniglia;
16. premistoppa;
17. guarnizione premistoppa;
18. perno;
19. seggio (ne servono sempre 2);

20. verniciatura all'esterno;
21. sfera;

Saracinesche

Corpo

1. stampaggio all'esterno;
22. ricottura all'esterno;
23. sabbiatura all'esterno;

Cuneo

1. stampaggio all'esterno;
24. sabbiatura all'esterno;
25. ricottura esterna;
103. sabbiatura interna;

Vitone

1. stampaggio all'esterno;
26. ricottura;
27. sabbiatura all'esterno;
104. sabbiatura interna;

Parti

28. Dado
29. Asta
30. Volantino
31. Guarnizione

Torneria

Lavorazione meccanica per asportazione del truciolo

Componente	Grande	Piccolo
Manicotto	105	106

Sfera	107	108
Corpo (valvole a sfera)	109	110
Corpo (saracinesche)	111	112
Cuneo	113	114
Vitone	115	116

Montaggio

Prodotto finito	Grande	Piccolo
Valvole a sfera	117	118
Saracinesche	119	120

Preassemblato	Grande	Piccolo
Valvole a sfera:		
Seggio + Manicotto (A)	121	122
Seggio + Corpo (B)	123	124
A + B + Sfera + Perno (C)	125	126
C + Maniglia + Bullone + Dado + Premistoppa + Prova	127	128
Saracinesche:		
Corpo + Cuneo + Vitone + Asta + Guarnizione	129	130
Volantino + Dado + Prova	131	132

I numeri rappresentano i possibili passi della *ricetta*, mentre le descrizioni indicano il tipo di lavorazione.
In analogia con il caso generale le ricette delle componenti sono individuate da numeri in sequenza e la durata di ogni fase è segnalata dal numero di ripetizioni della cifra che contrassegna la fase stessa.

Il caso generale non tiene però in conto un problema che emerge dall'applicazione alla realtà Vir: le ricette che costituiscono i prodotti in jVE vengono generate in modo casuale, transitano attraverso le diverse unità che compiono le lavorazioni e quindi si trasformano in prodotti finiti; nel caso Vir le ricette sono relative a componenti che dovranno essere tra loro assemblati per concorrere a formare il prodotto finito. Questa frammentazione della ricetta principale nelle sotto-ricette per le componenti determina la necessità di gestire produzioni asincrone. Inizialmente si è valutata la possibilità di introdurre all'interno della sequenza produttiva un "valore sentinella" (ad esempio -1) seguito da un contatore (±g). Il primo serve per formalizzare la possibilità che una parte della lavorazione debba iniziare in un momento differente da quello che effettivamente le verrebbe assegnato in maniera sequenziale; il contatore viene utilizzato per determinare lo spostamento nel tempo sia in termini quantitativi, cioè il numero di unità di tempo, sia in termini qualitativi, in grado cioè di determinare la direzione sull'asse dei tempi che dovrà assumere: lo spostamento nel tempo dell'inizio della lavorazione avviene a ± giorni dalla generazione della ricetta.

Il valore sentinella dovrebbe far fronte alla necessità di spostare nel tempo un processo produttivo riuscendo così ad ottenere il componente o il semilavorato in un momento utile al fine del rispetto della scadenza.

Tale impostazione delinea quattro diversi scenari:

I. il segno del contatore è negativo: la lavorazione x dovrà cominciare in anticipo rispetto alla generazione della ricetta. Nella realtà VIR ciò è assimilabile al ricevimento di un ordine di prodotti composti da componenti che richiedono un tempo lungo di produzione più lungo rispetto a quello utile per il cliente: si lavorerà su previsione sfruttando l'esperienza dell'addetto alla programmazione e si produrrà in anticipo;

La sequenza produttiva sarebbe...

1 - 2 - 5 - 8 - 3 - 2 - 14 - 18

...con le modifiche avremo...

1 - 2 - 5 - (-1 -4) 8 - 3 - 2 - 14 - 18

...che significa:

8 - 8 - 8 - 8 - 8 - 8 - 8 -

1 - 2 - 5 - 8 - 3 - 2 - 14 - 18

nelle prime quattro unità di tempo si produce su previsione, poi in parallelo fino alla quarta fase

II. il valore di g è pari a zero oppure il segno del contatore è positivo, ma g è minore della somma dei tempi di lavorazione dall'inizio

della sequenza fino al processo considerato: la lavorazione dovrà partire nel momento opportuno creando talora la necessità di avviare più lavorazioni in parallelo;

La sequenza produttiva sarebbe...

1 - 2 - 5 - 8 - 3 - 2 - 14 - 18

...con le modifiche avremo...

1 - 2 - 5 - (-1 +1) 8 - 3 - 2 - 14 - 18

...che significa:

8 - 8 -

1 - 2 - 5 - 8 - 3 - 2 - 14 - 18

la quarta fase di lavorazione comincia già insieme alla seconda.

III. il segno del contatore è positivo, ma g è maggiore della somma dei tempi di lavorazione dall'inizio della sequenza fino al processo considerato: occorre modificare la sequenza produttiva ed avviare più lavorazioni in parallelo. Nel caso VIR questa è la situazione in cui non vi sia l'obbligo che la sequenza produttiva venga rispettata e si preferisca effettuare una produzione di componenti dopo averne avviata un'altra che dura più a lungo. Occorrerebbe introdurre la gestione delle priorità (ad esempio spazio in magazzino);

La sequenza produttiva sarebbe...

1 - 2 - 5 - 8 - 3 - 2 - 14 - 18

...con le modifiche avremo...

1 - 2 - 5 - (-1 +5) 8 - (-1 +3) 3 - 2 - 14 - 18

...che significa:

8

1 - 2 - 5 - 8 - 3 - 2 - 14 - 18

3 - 3 -

la quarta fase di lavorazione viene ritardata di una unità di tempo perché non fondamentale per eseguire la quinta fase.

IV. non ci sono "valori sentinella": nella realtà VIR è interpretabile come il punto I) di cui sopra, mentre nel modello nulla cambierebbe rispetto all'impostazione generale di jVE.

L'impostazione descritta non permetteva di far fronte alla problematica relativa al punto I): il "valore sentinella" presente nella ricetta avvertiva della necessità che una fase di lavorazione cominciasse in anticipo, ma non era chiaro come potesse essere "lanciata" ancora prima che la ricetta stessa entrasse in produzione.

La soluzione migliore per simulare questo tipo di situazione è parsa essere quella di ricorrere alla struttura di una catena di fornitura. La ricetta continua ad essere una sequenza di numeri indicativi delle fasi di produzione, ma i passi della ricetta che richiedono produzioni asincrone sono preceduti da un indicatore di procurement.

La sequenza della ricetta risulta strutturata nel modo seguente:

n1 ts m1 p k c1 c2 . . . ck n2 ts m2 . . .

in cui i simboli sono indicativi di:

- n1 il numero della fase di lavorazione;
- ts il tempo che tale fase richiede (s = secondi, m = minuti, h = ore, d = giorni);
- m1 la quantità di s, m, h oppure d;
- p l'inizio del procurement;
- k il numero di parti intermedie da reperire;
- ck la parte intermedia da reperire.

Le ricette delle parti intermedie c1, c2 … ck sono costruite secondo il modello tradizionale, con l'indicazione del numero della fase di lavorazione e il tempo che tale fase richiede, ma terminano tutte con una "e" ed un numero identificativo per indicare che non sono prodotti finiti, ma componenti intermedi che verranno richiamati da altre fasi.
Il formalismo è il seguente:

…n1 ts m1 e ck

Strutturando le ricette in questo modo il modello continua a funzionare in modo sequenziale: i passi della ricetta vengono eseguiti dalle unità uno dopo l'altro, per i passi che richiedono un procurement saranno interrogate unità speciali per reperire le parti intermedie necessarie alla lavorazione.
Questa evoluzione nel formalismo delle ricette ha permesso di trattare i casi delle produzioni di quelle componenti che sono lavorate all'interno dell'impresa e concorreranno a formare il prodotto finale. Come accennato in precedenza, però, alcune parti delle valvole a sfera e delle saracinesche sono acquistate oppure subiscono parte delle lavorazioni all'esterno.

Nel caso della jVE generale, questo aspetto non solleva alcuna difficoltà, in quanto la lavorazione esterna all'impresa è semplicemente indicata con un numero differente dalle lavorazioni svolte internamente. Questa impostazione non considera, però, i vincoli sui lotti ai quali un'impresa reale è soggetta. Nel momento in cui l'impresa si trova ad ordinare delle componenti o ad inviarne altre all'esterno per essere lavorate non può permettersi di ragionare in termini unitari, ma deve sottostare alle limitazioni sui lotti imposte da fornitori.

Si determina quindi la necessità di una nuova evoluzione nel formalismo delle ricette che tratti il caso dei lotti, si distinguono due situazioni:

- stand alone batch: 11 s 10 / 5 e 101

  per trattare il caso di acquisto di componenti all'esterno;
- sequential batch: ... 10 s 2 11 s 10 \ 5 12 s 3

  per trattare il caso di lavorazioni svolte esternamente, ad esempio i trattamenti superficiali descritti in precedenza nella fase II (figura 2).

Rimangono ancora da descrivere la possibilità di scelta tra lavorazioni diverse che determinano uno stesso prodotto e il caso di lavorazioni che possono essere svolte in parallelo.

Il formalismo che tratta la prima situazione (OR) è così strutturato:

data la ricetta

n1 ts m1 n2 ts m2 n3 ts m3 n22 ts m22 n4 ts m4

se le lavorazioni

n2 ts m2 n3 ts m3 n22 ts m22

sono tra loro alternative il formalismo della ricetta sarà:

n1 ts m1 || 1 n2 ts m2 n3 ts m3 || 2 n22 ts m22 || 0 n4 ts m4

Nel caso di lavorazioni parallele (AND) avremo invece:
data la ricetta

n1 ts m1 n2 ts m2 n3 ts m3 n22 ts m22 n4 ts m4

se le lavorazioni

n2 ts m2 n3 ts m3 n22 ts m22

Possono essere lavorate in parallelo il formalismo della ricetta sarà:
n1 ts m1 && 1 n2 ts m2 n3 ts m3 && 2 n22 ts m22 && 0 n4 ts m4

Le ricette che si ottengono con l'introduzione dei formalismi descritti sono raccolte nel file Recipe.xls che risulta così strutturato:

- nella prima colonna il segno # indica la presenza di una riga di commento;
- nella seconda colonna viene indicato il nome della ricetta contenuta nella riga;
- la terza colonna riporta il codice della ricetta;
- dalla quarta colonna in poi sono indicate le fasi che compongono la ricetta, secondo le strutture precedentemente descritte;

ogni riga si conclude con un ";"

#	LAVORAZIONI GENERICHE	;											
	acquistopaniottone	1000001	100001	d	2	/	500000	e	1	;			
	stampaggio	1000002	p	1	1	100002	d	2	500000	e	2	;	
	sabbiaturainterna	1000003	p	1	2	100003	s	1	\	1	e	3	
	ricotturaesterna	1000004	p	1	3	100004	d	5	\	20000	e	4	
	sabbiaturaesterna	1000005	p	1	2	100005	d	2	\	5000	e	5	
	preassemblaggioesterno	1000006	100006	d	5	/	5000	e	6	;			
	acquistobarreottone	1000007	100007	d	2	/	500000	e	7	;			
#	cromatura		100008	d	2	\	10000	;					
#	teflonatura		100009	d	15	\	30000	;					
#	nichelatura		100010	d	2	\	10000	;					
	stampaggiosfera	1000008	p	1	7	100011	d	2	/	500000	e	11	

Figura 10: File recipes.xls lavorazioni generiche

#	COMPONENTI ACQUISTATE ALL'ESTERNO	;				;				
#	per valvola a sfera	;								
	bullone	1000009	100014	d	2	/	300000	e	14	;
	leva	1000010	100015	d	3	/	15000	e	15	;
	perno	1000011	100016	d	1	/	100000	e	16	;
	premistoppa	1000012	100017	d	2	/	200000	e	17	;
	guarnizionepremistoppa	1000013	100018	d	1	/	200000	e	18	;
	seggio	1000014	100019	d	2	/	150000	e	19	;
	rivestimentoplastico	1000015	100020	d	1	/	15000	e	20	;
	per saracinesca	;								
	asta	1000016	100021	d	1	/	20000	e	21	;
	dadopremistoppa	1000017	100022	d	2	/	50000	e	22	;
	guarnizione	1000018	100023	d	1	/	35000	e	23	;
	ghiera	1000019	100024	d	3	/	40000	e	24	;
	volantino	1000020	100025	d	1	/	15000	e	25	;
	dado	1000021	100026	d	2	/	40000	e	26	;

Figura 11: File recipes.xls componenti

#	TORNERIA	;													
#	per valvola a sfera grande	;													
	manicottopiccolo	1000022	p	1	4	100	s	3	100008	d	2	/	10000	e	1100
	manicottogrande	1000023	p	1	4	101	s	16	100008	d	3	/	10000	e	1101
	corpopiccolo	1000024	p	1	5	102	s	6	100008	d	4	/	10000	e	1102
	corpogrande	1000025	p	1	5	103	s	14	100008	d	5	/	10000	e	1103
	sferapiccola	1000026	p	1	11	104	s	5	100008	d	6	/	10000	e	1104
	sferagrande	1000027	p	1	11	105	s	18	100008	d	7	/	10000	e	1105
	cuneogrande	1000028	p	1	4	106	s	15						e	1106
	cuneopiccolo	1000029	p	1	5	107	s	6						e	1107
	vitonegrande	1000030	p	1	4	108	s	13						e	1108
	vitonepiccolo	1000031	p	1	5	109	s	4						e	1109
	corposaracinescagrande	1000032	p	1	4	110	s	17						e	1110
	corposaracinescapiccolo	1000033	p	1	5	111	s	5						e	1111

Figura 12: File recipes.xls torneria

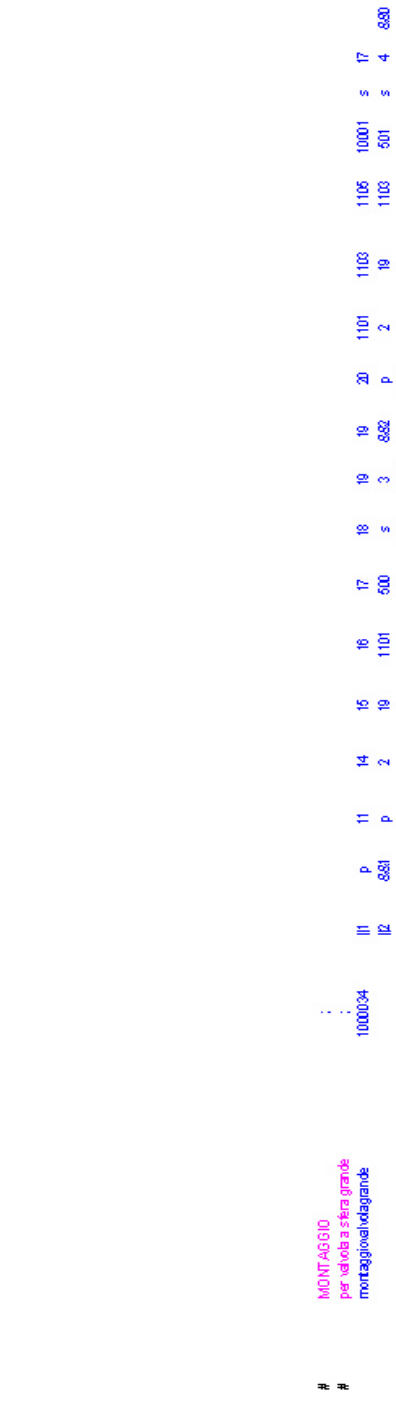


Figura 13: File recipes.xls montaggio

Il generatore di ordini della Vir virtuale seleziona dal file recipes.xls le tipologie di ricette da mandare in produzione. Per simulare il funzionamento reale dell'impresa è stato necessario introdurre una sorta di archivio storico, il file orderSequences.xls, in cui ogni riga indica la produzione giornaliera dell'azienda, il numero a sinistra dell'asterisco è identificativo del tipo di ricetta, quello a destra della quantità.

100001 * 1 100002 * 1 100003 * 49500 100004 * 2 100005 * 10 100007 * 1

Figura 14: Inizio della prima riga del file orderSequences.xls

Nell'organizzare il file orderSequences.xls si è dovuto tenere conto di diversi fattori:

- volumi giornalieri di produzione;
- lotti diversi per le diverse componenti;
- produzioni per il magazzino;
- tempistiche degli ordinativi.

In base alla descrizione dei quantitativi di produzione, emersa durante la visita agli stabilimenti, si è cercato di ripartire il volume reale della produzione totale tra i due prodotti scelti per la simulazione.

La produzione della Vir virtuale è costituita per il 60% da valvole a sfera (di cui 60% piccole e 40% grandi) e per il 40% da saracinesche (di cui 70% piccole e 30% grandi).

Nei primi giorni di ogni mese la produzione delle valvole a sfera è quasi doppia perché si devono alimentare i magazzini per le componenti, le saracinesche non prevedono accumuli di scorte per le componenti.

Nel corso del mese, con tempistiche regolari, devono essere lanciate le ricette per i prodotti intermedi, che saranno richiamati attraverso i

procurement, e devono essere eseguiti gli acquisti delle componenti esterne. Un'errata temporizzazione dei procurement, un eccesso di produzione di prodotti intermedi o di acquisto di componenti possono determinare squilibri nella struttura aziendale della Vir simulata, provocando code presso unità sistematicamente in ritardo o sistematicamente in attesa ed un aumento dei costi finanziari, che nella simulazione si manifestano con andamenti anomali degli istogrammi, ma nella realtà si traducono in una perdita di efficienza dell'azienda.

6.2.2 Le unità (which is Doing What - DW)

Fino ad ora si è data una descrizione delle sequenze di lavorazioni per la produzione; per costruire la simulazione occorre introdurre le unità con le caratteristiche necessarie per espletare le fasi descritte. La predisposizione di tali strutture virtuali consente di riprodurre le regole di funzionamento e di interazione interne all'impresa.

La descrizione delle unità produttive si è rivelato l'aspetto più difficoltoso della collaborazione con l'impresa reale, in quanto, in questa fase, è stato necessario distinguere tra ciò che l'azienda fa e quello che i gestori razionalizzano faccia.

La rappresentazione della conoscenza tacita consente di guardare all'impresa virtuale come a una struttura riprodotta in laboratorio, che consente alla sperimentazione di indirizzare gli sforzi della ricerca verso i problemi di organizzazione delle unità produttive e di gestione della ricerca e di ottenere risultati significativi per il mondo reale.

Se ripercorriamo nel dettaglio la descrizione delle due famiglie di prodotti principali, riportata sopra, è semplice notare come si ricada facilmente nella simulazione di processo assegnando ad ogni fase di lavorazione il macchinario attraverso cui il pezzo transita realmente in azienda.
Affinché la complessità, invece, emerga dall'interazione tra gli agenti, tra le ricette e le unità, è necessario descrivere queste ultime compiutamente.
Nel modello generale di impresa virtuale le unità erano in grado di svolgere un'unica lavorazione, il numero distintivo della fase descritta nella ricetta corrispondeva al numero dell'unità che si occupava di quella fase. Nel caso Vir ogni unità è in grado di riattrezzarsi e occuparsi di più fasi di lavorazione, la corrispondenza tra i numeri, pertanto, non è più possibile.
Nel file unitBasicData.txt (figura 15) si descrivono le unità semplici, indicando per ognuna la lavorazione di cui si occupa ed i costi fissi e variabili ad essa associati.
Per le unità che sono in grado di svolgere più di una lavorazione è indicato uno zero nella colonna relativa al numero della fase di produzione.
La descrizione di queste unità complesse è raccolta nel file Units.xls: ogni unità è associata ad una “linguetta” contrassegnata da un numero corrispondente a quello dell'unità stessa, ogni “linguetta” identifica un foglio di lavoro in cui sono riportate le specifiche organizzate secondo lo schema riportato in figura 16.
In alto a sinistra si indica il numero di lavorazioni che l'unità è in grado di eseguire. Di seguito, per ogni lavorazione, vengono riportati i costi fissi e i costi variabili.

unit_#	useWarehouse	prod.phase_#	fixed_costs	variable_costs
1	1	100001	1	1
2	1	100002	1	1
3	1	100003	1	1
4	1	100004	1	1
5	1	100005	1	1
6	1	100006	1	1
7	1	100007	1	1
8	1	100008	1	1
9	1	100009	1	1
10	1	100010	1	1
11	1	100011	1	1
12	1	100014	1	1
13	1	100015	1	1
14	1	100016	1	1
15	1	100017	1	1
16	1	100018	1	1
17	1	100019	1	1
18	1	100020	1	1
19	1	100021	1	1
20	1	100022	1	1
21	1	100023	1	1
22	1	100024	1	1
23	1	100025	1	1
24	1	100026	1	1
25	1	0	0	0
26	1	0	0	0
27	1	0	0	0
28	1	0	0	0
29	1	0	0	0
30	1	0	0	0
31	1	0	0	0
32	1	0	0	0
33	1	0	0	0
34	1	0	0	0
35	1	0	0	0
36	1	0	0	0
37	1	0	0	0
38	1	0	0	0
39	1	500	1	1
40	1	501	1	1
41	1	502	1	1
42	1	503	1	1
43	1	504	1	1
44	1	505	1	1
46	1	506	1	1
47	1	507	1	1
48	1	10001	1	1
49	1	10002	1	1
50	1	10003	1	1
51	1	10004	1	1
52	1	9997	1	1
53	1	9998	1	1
54	1	9999	1	1
55	1	10000	1	1

Figura 15: File unitBasicData.txt

nOfPhasesToDealWith

phase 1	fixed costs 1	variable costs 1	inventories in production 1
phase 2	fixed costs 2	variable costs 2	inventories in production 2
...			
phase n	fixed costs n	variable costs n	inventories in production n

sc 1 1	sc 1 2	...	sc 1 n
sc 2 1	sc 2 2	...	sc 2 n
...	...	...	...
sc n 1	sc n 2	...	sc n n

st 1 1	st 1 2	...	st 1 n
st 2 1	st 2 2	...	st 2 n
...	...	...	...
st n 1	st n 2	...	st n n

Remarks
hopefully, fixed costs are the same for all phases,
anyway, when the unit state is undefined (no production made) we use the fixed costs of the first row

inventory production can be 0 (no) or 1 (yes); normally, only one row is set to 1
if we find more than one row set to 1, the first one is chosen

sc i j = setup costs from state i to state j st i j = setup time from state i to state j

Figura 16: File Units.xls

Successivamente sono indicate due matrici contenenti rispettivamente i costi e tempi di attrezzaggio attraverso il quale l'unità passa da uno stato ad un altro.

Le unità tradizionali del modello jVE generale si articolano, quindi, in unità semplici ed unità complesse nel modello di simulazione Vir.

Accanto a queste due tipologie di unità è stato, però, necessario inserirne una terza: le unità speciali, presso le quali sono reperibili le parti intermedie di produzione asincrona.

Queste unità sono definite EndUnits e sono identificate con un codice corrispondente a quello identificativo dei prodotti intermedi le cui ricette terminano con una "e".

```
unit_#
10001
10002
10003
10004
10005
10006
10007
10008
10009
10010
10011
10012
10013
10014
10015
10016
10017
10018
10019
10020
10021
10022
10023
10024
10025
10026
10027
10028
10029
10030
10031
10032
10033
```

Figura 17: File endUnitList.txt

Le endUnits sono elencate nel file riportato in figura 17.

La codificazione disgiunta di questi due aspetti dell'azienda virtuale, le unità produttive da un lato e le ricette dall'altro, determina un funzionamento della simulazione in cui le interazione non sono controllate a priori, come avverrebbe in una simulazione di processo, ma attraverso il loro intrecciarsi fanno emergere la realtà aziendale simulata in tutta la sua complessità.
Per meglio comprendere la metodologia della simulazione ad agenti passiamo ora ad analizzare nel dettaglio le modifiche al codice che si sono rese necessarie per adattare il modello Java Swarm di impresa virtuale al caso concreto Vir.

6.2.3 Le modifiche al codice

L'applicazione al caso Vir del modello di impresa virtuale descritto nel capitolo precedente, versione 0.4.6, ha richiesto importanti modifiche al codice, che, senza stravolgerne la struttura di base, hanno permesso di generalizzare il modello e renderlo adattabile alla realtà aziendale presa in considerazione.
Per tappe successive il codice si è sviluppato dalla versione 0.4.6 fino alla versione 0.9.1.
Il primo passo dell'evoluzione è consistito nell'archiviare in un file jar tutti i file con estensione .class della versione 0.4.6 in modo da poter sperimentare l'introduzione di nuove classi, per l'adattamento del modello al caso Vir, senza alterare il modello di base.

L'utilizzo del *Jar* non implica modifiche del codice, quanto piuttosto modifiche inerenti il package nella sua interezza.
La struttura delle cartelle si articola come segue:

- src: contiene i file della versione corrente;
- classes: contiene i file .class relativi alle classi in src e i file usati per generare l'archivio jar:
  - makefile: per richiamare l'archivio jar;
  - jarJve
  - MANIFEST.MF: necessario per dare la possibilità al compilatore di trovare il file *StartVEFrame.class* di avvio della simulazione;
- lib: contenente l'archivio jar generato, gli archivi gui.jar, plot.jar e le sottodirectory di PT utilizzate per creare gli istogrammi.

Questa nuova impostazione determina la possibilità di compilare ed eseguire il programma con diverse opzioni:

a. make run per mandare in esecuzione jVE come è;
b. make compileBasic per compilare e make runFromClasses per eseguire il programma con gli stessi effetti del punto precedente;
c. make jar per rigenerare l'archivio jar;
d. make e make compile per aggiungere o modificare classi;
e. make runJar per usare la versione base o in alternativa al punto a.

Con la versione 0.5.1 si introducono i primi cambiamenti significativi al codice, la prima modifica importante riguarda l'Order Generator.
Si introduce la possibilità di generare ricette con passi di produzione di durate diverse. Le informazioni relative ai tempi di lavorazione sono

contenute nelle ricette e non presso le unità produttive per due ragioni principalmente:

- esplorando la ricetta è possibile determinare il tempo standard di produzione, vale a dire il tempo che occorre per ottenere il prodotto finito nella situazione in cui siano disponibili tutte le componenti;
- inserendo le informazioni presso le unità significherebbe confondere due elementi di conoscenza distinti WD e DW cui si è accennato precedentemente.

A livello di listato la modifica è individuabile nelle seguenti righe:

```
   * orderRecipe1      contains  1 12  7  3
   * stepLengthInOrder contains  1  3  2  2
   *
   * orderRecipe2      contains  1 12 12 12  7  7  3  3
   */

// generating the length of each step

        // putting 0 in the vector is not strictly necessary, but may be
        // useful for future software modifications
        for (i=0;i<maxStepNumber;i++)stepLengthInOrder[i]=0;

        if (maxStepLength>1)
           {
                for (i=0;i<randomStepNumber;i++)
                   stepLengthInOrder[i]=
                        Globals.env.uniformIntRand.getIntegerWithMin$withMax
                        (1, maxStepLength);
           }
        else
           {
                for (i=0;i<randomStepNumber;i++)stepLengthInOrder[i]=1;
           }

        // putting (i) sequence and (ii) length, together
        count=0;
        for (i=0;i<randomStepNumber;i++)
           for (ii=0;ii<stepLengthInOrder[i];ii++)
                {
                   orderRecipe2[count]=orderRecipe1[i];
                   count++;
                }
```

Le fasi di produzione sono ripetute tante volte quanto è indicato nel vettore dei tempi corrispondente: stepLengthInOrder.
Questa ripetizione determina la necessità di gestire fasi di lavorazione consecutive tra loro uguali in modo LIFO e non FIFO.
In questa versione del modello si introduce anche la scansione del tempo in giorni introducendo nello schedule il valore di ticks in a day

```
// Then we create a schedule that executes the
        // modelActions. modelActions is an ActionGroup, by itself it
        // has no notion of time. In order to have it executed in
        // time, we create a Schedule that says to use the
        // modelActions ActionGroup at particular times
        modelSchedule = new ScheduleImpl (getZone (), ticksInATimeUnit);

        modelSchedule.at$createAction (0, modelActions1);

        for (i=0;i<ticksInATimeUnit;i++)
        modelSchedule.at$createAction (i, modelActions2);
        if(useOrderDistiller)modelSchedule.
                                at$createAction (0, modelActions2distiller);
        for (i=0;i<ticksInATimeUnit;i++)
        modelSchedule.at$createAction (i, modelActions2b);

        modelSchedule.at$createAction (ticksInATimeUnit-1, modelActions3);

        return this;
   }


…
/** the number of ticks in a day */
   public int getTicksInATimeUnit(){
        return ticksInATimeUnit;
```

Dal lato delle unità le prime modifiche significative si hanno con la versione 0.5.2. Sono introdotte unità in grado di svolgere più di una lavorazione.
Le descrizioni delle unità semplici sono contenute nel file UnitBasicData.txt (figura 15, paragrafo 6.2.2), quelle delle unità complesse sono raccolte nel file Units.xls (figura 16, paragrafo 6.2.2) le

cui strutture sono state ampliamente analizzate nel corso del paragrafo 6.2.2. Per leggere le informazioni contenute in questi file è stata introdotta la classe ExelReader, che si fonda sulla libreria xlrd.jar (http://www.andykhan.com/excelread/).

L'introduzione di unità complesse altera la corrispondenza tra i numeri delle fasi di lavorazione contenuti nelle ricette e quelli indicativi delle unità in grado di eseguire la lavorazione.

Le modifiche sono state apportate nel file UnitParameters di cui segue la parte di listato:

```
/** read the unit number, production phase, fixed cost and variable cost
    *  parameters from the unitBasicData.txt file*/
   public void readUnitData(){
         try{
            unitNumber = new int[totalUnitNumber];
            simpleUnitProductionPhase = new int[totalUnitNumber];
            fixedCosts = new float[totalUnitNumber];
            variableCosts = new float[totalUnitNumber];
            useWarehouseLocalChoice = new int[totalUnitNumber];

            // read the file
            BufferedReader fileUnitBasicData = null;
            StringTokenizer t;
            fileUnitBasicData = new
                  BufferedReader(new FileReader("unitData/unitBasicData.txt"));

            // reading a control line (titles)
            String line = " ", lineTest =
     "unit_#__useWarehouse____prod.phase_#____fixed_costs_____variable_costs";

            line=fileUnitBasicData.readLine();
            if (! line.equals(lineTest))
                  {
                     System.out.println(
                     "First line in unitData/unitBasicData.txt must contain:");
                     System.out.println(lineTest);
                     System.exit(1);
                  }
            line=" ";

            // starting with a void line; the actual
            // reading is done in MyReader, dealing also
            // with the missing data case
```

```
            t = new StringTokenizer(line, " ");

            for(i = 0; i < totalUnitNumber; i++)
                {
                    t=MyReader.check("unitData/unitBasicData.txt",
                                     fileUnitBasicData, line, t);
                    unitNumber[i]    = Integer.parseInt(t.nextToken());
                    useWarehouseLocalChoice[i] = Integer.
                          parseInt(t.nextToken());
                    simpleUnitProductionPhase[i]
                                    = Integer.parseInt(t.nextToken());
                    fixedCosts[i]    = Float.parseFloat(t.nextToken());
                    variableCosts[i] = Float.parseFloat(t.nextToken());
                }

            fileUnitBasicData.close();
        }
        catch (FileNotFoundException fnfe)
            {
                System.out.println(fnfe);
                System.exit(1);
            }
        catch (IOException e)
            {
                System.out.println("IOException:" + e.toString());
            }
}

/** setting the parameters of units */
public void setParametersTo(int i, Unit aUnit){
        String fileName;
        float fixedCosts, variableCosts;
        int unitProductionPhase, nOfPhasesToDealWith, phaseInventories;

        aUnit.setUnitNumber(getUnitNumber(i));
        aUnit.setPhaseInventories(getUseWarehouseLocalChoice(i));
        aUnit.setProductionPhase(getSimpleUnitProductionPhase(i));
        aUnit.setFixedAndVariableCosts(getFixedCosts(i), getVariableCosts(i));


        // complex unit case, with a sheet in a workshhet file related to
        // the each unit
        if(getSimpleUnitProductionPhase(i) == 0) // i.e. no unique phase
            {
                // open (once) the Excel file containing special unit data */
                if(! worksheetUnitFileOpen)
                    specialUnitWorksheet =
                          new ExcelReader("unitData/units.xls");
                worksheetUnitFileOpen = true;

                specialUnitWorksheet.selectSheet(String.valueOf(i));
```

```
            if(StartVEFrame.verbose)
               System.out.println("Unit " + getUnitNumber(i)
                                  + " is a special one.");

            // the unit is able to deal with nOfPhasesToDealWith
            // different production phases
            nOfPhasesToDealWith=specialUnitWorksheet.getIntValue();
            if(StartVEFrame.verbose)
            System.out.println("special unit " +
          getUnitNumber(i) + " nOfPhasesToDealWith " +
              nOfPhasesToDealWith);

            aUnit.setNOfPhasesToDealWith(nOfPhasesToDealWith);

            for (ii=1; ii<=nOfPhasesToDealWith; ii++)
               {
                     // production phases (one or more)
                     unitProductionPhase=specialUnitWorksheet.getIntValue();
                     aUnit.setProductionPhase(unitProductionPhase);

                     // costs for each phase

                     fixedCosts = (float)
                        specialUnitWorksheet.getDblValue();
                     variableCosts = (float)
                        specialUnitWorksheet.getDblValue();
                        if(StartVEFrame.verbose)
                          System.out.println("special unit " +
                          getUnitNumber(i) + " fixed costs " +
                          fixedCosts + " variable costs " +
                                          variableCosts);
                        aUnit.setFixedAndVariableCosts(fixedCosts,
                                                       variableCosts);
                        phaseInventories =
                              specialUnitWorksheet.getIntValue();
                        aUnit.setPhaseInventories(phaseInventories);
               }
         }
```

Una volta definite le unità secondo le caratteristiche descritte è stato necessario modificare l'Order Generator affinché interrogasse le Units e creasse un dizionario delle fasi di lavorazione da inserire, per il momento ancora in modo casuale, nelle ricette.

```
/** building the dictionary containing the steps to be included in the
      recipes */
public void setDictionary()
{
      int i, ii, iii;
      // how many words in the dictionary?
```

```
        // starting point: the no. of endUnits
        dictionaryLength=endUnitList.getCount();

        for (i=0;i<unitList.getCount();i++)
           {
                aUnit=(Unit) unitList.atOffset(i);
                dictionaryLength += aUnit.getNOfPhasesToDealWith();
           }
        if(StartVEFrame.verbose)
           System.out.println(
         "Length of the dictionary (# of codes identifying steps in recipes): "
         + dictionaryLength);

        //creating the dictionary
        dictionary = new int[dictionaryLength];

        // loading terms into the dictionary (from endUnits and then units)
        i=0;
        for (ii=0;ii<endUnitList.getCount();ii++)
           {
                anEndUnit=(EndUnit) endUnitList.atOffset(ii);
                dictionary[i++]=anEndUnit.getUnitNumber();
           }
        for (ii=0;ii<unitList.getCount();ii++)
           {
                aUnit=(Unit) unitList.atOffset(ii);
                for (iii=0;iii<aUnit.getNOfPhasesToDealWith();iii++)
                   dictionary[i++]=aUnit.getProductionPhase(iii);
           }

        if(StartVEFrame.verbose)
           for (i=0;i<dictionaryLength;i++)
                System.out.println(i + "  " + dictionary[i]);
    }
```

Nel listato si può notare che l'Order Generator interroga, tra le unità, anche le End Unit introdotte con la versione 0.6.2 per simulare il caso di produzioni asincrone. Le End Unit sono unità speciali che contengono i prodotti intermedi necessari per le lavorazioni di altre unit "tradizionali".
Il numero che le identifica corrisponde al codice con cui si concludono le ricette dei prodotti intermedi corrispondenti. L'elenco delle End Unit è contenuto nel file: endUnitList.txt (figura 17 paragrafo 6.2.2).

La classe delle End Unit è costruita sfruttando la proprietà dell'ereditarietà delle classi (Unit è la *parent class*), propria della programmazione ad oggetti, già descritta nel capitolo 5.

```
/**
    * the constructor for EndUnit
    */
   public EndUnit (Zone aZone, VEFrameModelSwarm mo)
   {
         // Call the constructor for the Unit parent class.
         super(aZone);

         // the model (for future use)
         vEFrameModelSwarm = mo;

         // creating the internal list temporary the items temporary in hold
         // hold
         temporaryList = new ListImpl (getZone());
   }
```

Per poter attivare queste unità speciali è stato necessario introdurre nelle ricette un nuovo formalismo. La versione 0.6.3 introduce nelle ricette i passi di procurement (p).

Riportiamo di seguito il listato dell'Order Generator

```
// introducing "p" or "c" sequences

// the "p" strucuture is
// p k a1 a2 .... ak
// where k is the # of items to be procured and
// a1, a2, ..., ak are the codes of those items
// (corresponding to "e" or endUnit codes in sub-recipes
// related to internal or external sub-processes

// the recipe scheme is
// external recipe (with the time specifications and
// a tick corresponding to 1 sec. or to an analogous
// minimum fraction of the time

// 11 s 3 p 3 101 102 103 22 s 2

// internally, in orderGenerator, with p reported as -1

// 11 11 11 -1 3 101 102 103 22 22

// into anOrder

// 11 11 11 -22 22
```

```
            // with a procurementSpecifications object having the same
            // # of the order and a position paramer relative to the
            // recipe sequence (here position = 4)

            // and, when anOrder comes back from procurementAssembler,
            // 11 11 11 22 22
            // always with the procumerementSpecifications object

            if(endUnitList.getCount() > 0)
                {
                   if (ii==0 && 0.05 >= Globals.env.uniformDblRand.
                        getDoubleWithMin$withMax(0,1)){

                        // length of the "p" or "c" step
                        k=Globals.env.uniformIntRand.
                           getIntegerWithMin$withMax
                           (1, endUnitList.getCount());

                        // increase the length of orderRecipe2 vector
                        orderRecipeTmp = new int[orderRecipe2.
                                              length+2+k];
                        System.arraycopy(orderRecipe2,0,
                                         orderRecipeTmp,0,
                                         orderRecipe2.length);
                        orderRecipe2=orderRecipeTmp;

                        //the code of process p
                        orderRecipe2[count++]= -1;

                        // the length
                        orderRecipe2[count++]=k;

                        // k codes, chosen randomly
                        for (iii=0;iii<k;iii++)
                           orderRecipe2[count++]=
                                 ((EndUnit) endUnitList.
                                  atOffset(
                                           Globals.env.uniformIntRand.
                                           getIntegerWithMin$withMax
                                           (0, endUnitList.getCount()-1)
                                           )).getUnitNumber();
                   }
                }

            // copying normal steps
            orderRecipe2[count++]=orderRecipe1[i];

         }

if(StartVEFrame.verbose)
{
   for (iii=0; iii<count;iii++)
```

```
            System.out.print(orderRecipe2[iii]+ " ");
        System.out.println(" internal recipe for order " + orderCount);
    }
```

Al contempo sono create due nuove classi:

- ProcurementSpecificationSet: contiene le sottoricette;
- ProcurementAssembler: cerca le componenti da procurare.

La struttura delle ricette necessita ancora di una modifica per gestire le produzioni batch, il cui formalismo è stato già descritto nel precedente paragrafo.

Sono create per questo scopo due classi per ogni tipologia di bach:

- StandAloneBatchSpecificationSet;
- StandAloneBatchAssembler;
- SequentialBatchSpecificationSet;
- SequentialBatchAssembler;

in analogia con le classi generate per gestire i procurements.

Un ultimo aspetto da considerare in materia di formalismo delle ricette riguarda la possibilità di trattare produzioni alternative.

L'introduzione dei criteri di "OR", esaminati nel paragrafo precedente, ha richiesto l'introduzione della nuova classe Or.java.

All'interno della classe sono definite cinque possibilità:

- Si eseguono in sequenza entrambi i rami;
- Si esegue il primo ramo;
- Si esegue il secondo ramo;
- Si opera una scelta casuale tra i due rami;
- Si sceglie il ramo in cui l'unità che può effettuare il primo passo di produzione ha la lista di attesa più corta.

```
// applying the 'or' criterion
        if(orCriterion == 0)return;
```

```
chosenNode=0;
recipe = (int []) o.getRecipeVector();
state =  (int []) o.getStateVector();

if(orCriterion == 1)chosenNode=1;
if(orCriterion == 2)chosenNode=2;
if(orCriterion == 3)chosenNode=Globals.env.uniformIntRand.
                    getIntegerWithMin$withMax(1,spec.getNodeNumber());

if(orCriterion == 4)
   {
        length=1000000000;
        // looking for the first unit in each branch
        for (node=1; node<=spec.getNodeNumber();node++)
           {
                length0=length;
                unitNotFound=true;
                for (i=0;i<unitList.getCount() && unitNotFound;i++)
                   {
                        aUnit=(Unit) unitList.atOffset(i);
                        for (iii=0;iii<aUnit.getNOfPhasesToDealWith()
                                && unitNotFound;iii++)
                           {
                              if(recipe[spec.getNodePosition(node)]==
                                aUnit.getProductionPhase(iii))
                                {
                                     unitNotFound=false;
                                     length=aUnit.
                                        getWaitingListLength();
                                     if(length<length0)chosenNode=
                                                        node;
                                }
                           }
                   }

           }
   }
```

Il modello di simulazione di impresa virtuale, per essere adattato al caso Vir, richiede una evoluzione non soltanto dal lato del formalismo delle ricette e della complessità delle unità, ma anche nell'aspetto della generazione degli ordini.

Per introdurre la possibilità di gestire un archivio storico (figura 14, paragrafo 6.2.2) da cui estrarre le ricette da mandare in produzione è stato necessario generare due nuove classi:

- Order Distiller;
- Recipe;

che sostituissero la generazione casuale degli ordini operata dall'Order Generator con una produzione aderente a quella della realtà aziendale simulata.

Si riportano di seguito i file javadoc relativi alle due classi in questione.

Class OrderDistiller
OrderGenerator
|
+--**OrderDistiller**

---

public class **OrderDistiller**
extends OrderGenerator
OrderDistiller.java This class is used to read data from two worksheets. The first one contains the list of recipes of our virtual enterprise. The second one contains a sequence of orders to be launched, shift by shift, in order to make the daily production activities Created: Wed May 08 15:29:12 2002
Author:
Cristian Barreca dgbarrec@libero.it Elena Bonessa elena.bonessa@infinito.it Antonella Borra anborra@libero.it

---

Field Summary	
Order	**anOrder** a specific order
java.lang.String	**backslash** Flags to operate checks while reading
java.lang.String	**checkTheCell** Flags to operate checks while reading
java.lang.String	**end** Flags to operate checks while reading
swarm.collections.ListImpl	**endUnitList**

	the list containig the end units
java.lang.String	**gate** Flags to operate checks while reading
java.lang.String	**min** Flags to operate checks while reading
java.lang.String	**or** Flags to operate checks while reading
int	**orderCount** used to record the number of generated orders
swarm.collections.ListImpl	**orderList** the list containig all the orders
java.lang.String	**p** Flags to operate checks while reading
swarm.collections.ListImpl	**recipeList** the list containig all the orders
java.lang.String	**sec** Flags to operate checks while reading
java.lang.String	**semicolon** Flags to operate checks while reading
java.lang.String	**slash** Flags to operate checks while reading
swarm.collections.ListImpl	**unitList** the list containig the operating units
boolean	**unitNotFound**
boolean	**worksheetOrderSequenceFileOpen** A flag to check if the orderSequences worksheet file is open
boolean	**worksheetRecipeFileOpen** A flag to check if the orderSequences worksheet file is open

Fields inherited from class OrderGenerator
dictionary, dictionaryLength, maxStepLength, maxStepNumber, orderRecipe1, orderRecipe2, orderRecipeTmp, stepLengthInOrder

Constructor Summary

OrderDistiller(swarm.defobj.Zone aZone, int msn, int msl, swarm.collections.ListImpl ul, swarm.collections.ListImpl eul, swarm.collections.ListImpl ol, VEFrameModelSwarm mo)

Method Summary

void	**calculateLength**(java.lang.String cTC) This method is used to calculate the length of each row after a strong check of the elements
void	**checkForComments**(java.lang.String cTC) This method is used to check if there are comments in the cells
int	**checkTheExistence**(int c) This method is used to check the corrispondence with the dictionary of production phases
void	**distill**() This is the method containing the iterator needed to launch the daily production of recipes.
void	**end**(ExcelReader e) This method dial with the end choice
int	**errorIsNotAnInteger**(ExcelReader e) This method is used to check if a type error occurs
void	**errorIsNotAString**(ExcelReader e) This method is used to check if a type error occurs
int	**getOrderSequence1**(int j) This method is used to obtain the elements of the orderSequence1 array
int	**getOrderSequence2**(int j) This method is used to obtain the elements of the orderSequence2 array
void	**minute**(ExcelReader e)
void	**number**(ExcelReader e) This method dial with normal or batch choice
void	**oR**(ExcelReader e) This method dial with the or choice
void	**procurement**(ExcelReader e)

	This method dial with the procurement choice
void	**readOrderSequence**() This method reads from the worksheet containing, shift by shift, the sequence of orders to be launched and fills in the orderSequence1 with the ID codes of recipes and the orderSequence2 with the quantities of each recipe.
void	**readRecipes**() This is the method needed to read and store the recipes.
void	**second**(ExcelReader e)
void	**setDictionary**() This method is used to collect the names of the units and of the end units, so that it can operate the check of corrispondency between the production phases required by recipes and the phases of production the units can do.
void	**setRecipeContainers**() This method is used to set the length of each recipe

Methods inherited from class OrderGenerator
createRandomOrderWithNSteps

Field Detail

worksheetOrderSequenceFileOpen
public boolean **worksheetOrderSequenceFileOpen**
A flag to check if the orderSequences worksheet file is open

---

worksheetRecipeFileOpen
public boolean **worksheetRecipeFileOpen**
A flag to check if the orderSequences worksheet file is open

---

semicolon
public java.lang.String **semicolon**
Flags to operate checks while reading

---

checkTheCell
public java.lang.String **checkTheCell**
Flags to operate checks while reading

---

gate
public java.lang.String **gate**

Flags to operate checks while reading

---

p
public java.lang.String **p**
Flags to operate checks while reading

---

sec
public java.lang.String **sec**
Flags to operate checks while reading

---

min
public java.lang.String **min**
Flags to operate checks while reading

---

end
public java.lang.String **end**
Flags to operate checks while reading

---

slash
public java.lang.String **slash**
Flags to operate checks while reading

---

backslash
public java.lang.String **backslash**
Flags to operate checks while reading

---

or
public java.lang.String **or**
Flags to operate checks while reading

---

orderCount
public int **orderCount**
used to record the number of generated orders

---

anOrder
public Order **anOrder**
a specific order

---

unitList
public swarm.collections.ListImpl **unitList**
the list containig the operating units

---

endUnitList
public swarm.collections.ListImpl **endUnitList**
the list containig the end units

orderList
public swarm.collections.ListImpl **orderList**
the list containig all the orders

recipeList
public swarm.collections.ListImpl **recipeList**
the list containig all the orders

unitNotFound
public boolean **unitNotFound**

Constructor Detail

OrderDistiller
public **OrderDistiller**(swarm.defobj.Zone aZone,
int msn,
int msl,
swarm.collections.ListImpl ul,
swarm.collections.ListImpl eul,
swarm.collections.ListImpl ol,
VEFrameModelSwarm mo)

Method Detail

setDictionary
public void **setDictionary**()
This method is used to collect the names of the units and of the end units, so that it can operate the check of corrispondency between the production phases required by recipes and the phases of production the units can do.
Overrides:
setDictionary in class OrderGenerator

setRecipeContainers
public void **setRecipeContainers**()
This method is used to set the length of each recipe

readRecipes
public void **readRecipes**()
This is the method needed to read and store the recipes. The procedure follows this steps:
It opens the worksheet file recipes.xls, in which are stored the sequences of steps of all the recipes;
It makes some check of the routines;
It search the object containig the same code of the recipe we are considering;

Finally it substitutes the values of the array of that object with the new ones.

---

distill

public void **distill**()

This is the method containing the iterator needed to launch the daily production of recipes. It take a look at the orderSequence arrays to determine which recipes must be done and how many times. A request for which unit can do the first production phase of each recipe will be done to units or endUnits.

---

readOrderSequence

public void **readOrderSequence**()

This method reads from the worksheet containing, shift by shift, the sequence of orders to be launched and fills in the orderSequence1 with the ID codes of recipes and the orderSequence2 with the quantities of each recipe.

---

checkForComments

public void **checkForComments**(java.lang.String cTC)

This method is used to check if there are comments in the cells

---

calculateLength

public void **calculateLength**(java.lang.String cTC)

This method is used to calculate the length of each row after a strong check of the elements

---

getOrderSequence1

public int **getOrderSequence1**(int j)

This method is used to obtain the elements of the orderSequence1 array

---

getOrderSequence2

public int **getOrderSequence2**(int j)

This method is used to obtain the elements of the orderSequence2 array

---

checkTheExistence

public int **checkTheExistence**(int c)

This method is used to check the corrispondence with the dictionary of production phases

---

errorIsNotAnInteger

public int **errorIsNotAnInteger**(ExcelReader e)

This method is used to check if a type error occurs

---

errorIsNotAString
public void **errorIsNotAString**(ExcelReader e)
This method is used to check if a type error occurs

---

procurement
public void **procurement**(ExcelReader e)
This method dial with the procurement choice

---

oR
public void **oR**(ExcelReader e)
This method dial with the or choice

---

end
public void **end**(ExcelReader e)
This method dial with the end choice

---

number
public void **number**(ExcelReader e)
This method dial with normal or batch choice

---

second
public void **second**(ExcelReader e)

---

minute
public void **minute**(ExcelReader e)

Class Recipe
Recipe

---

public class **Recipe**
This class is used to record the recipes and their referring number (ID), so we can assign them to a List
Author:
Cristian Barreca dgbarrec@libero.it Elena Bonessa elena.bonessa@infinito.it Antonella Borra anborra@libero.it

---

Field Summary	
java.lang.String	**backslash**

	Flags to operate checks while reading
java.lang.String	**checkTheCell** Flags to operate checks while reading
java.lang.String	**end** Flags to operate checks while reading
java.lang.String	**gate** Flags to operate checks while reading
java.lang.String	**min** Flags to operate checks while reading
java.lang.String	**or** Flags to operate checks while reading
java.lang.String	**p** Flags to operate checks while reading
java.lang.String	**sec** Flags to operate checks while reading
java.lang.String	**semicolon** Flags to operate checks while reading
java.lang.String	**slash** Flags to operate checks while reading

Constructor Summary

Recipe(swarm.defobj.Zone aZone, int c, int l, java.lang.String rN)

Method Summary

void	**end**(ExcelReader e) This method dial with the end choice
int	**errorIsNotAnInteger**(ExcelReader e) This method is used to check if a type error occur
void	**errorIsNotAString**(ExcelReader e) This method is used to check if a type error occur
java.lang.String	**getCheckTheCell**()
int	**getCodeNumber**()
int	**getLength**()

int[]	**getOrderRecipe**()
java.lang.String	**getRecipeName**()
void	**minute**(ExcelReader e, int step)
void	**number**(ExcelReader e) This method dial with normal or batch choice
void	**oR**(ExcelReader e) This method dial with the or choice
void	**procurement**(ExcelReader e) This method dial with the procurement choice
void	**second**(ExcelReader e, int step)
void	**setSteps**(java.lang.String cTC, ExcelReader recipeWorksheet)

Field Detail

semicolon
public java.lang.String **semicolon**
Flags to operate checks while reading

---

checkTheCell
public java.lang.String **checkTheCell**
Flags to operate checks while reading

---

gate
public java.lang.String **gate**
Flags to operate checks while reading

---

p
public java.lang.String **p**
Flags to operate checks while reading

---

end
public java.lang.String **end**
Flags to operate checks while reading

---

sec
public java.lang.String **sec**

Flags to operate checks while reading

---

min
public java.lang.String **min**
Flags to operate checks while reading

---

slash
public java.lang.String **slash**
Flags to operate checks while reading

---

backslash
public java.lang.String **backslash**
Flags to operate checks while reading

---

or
public java.lang.String **or**
Flags to operate checks while reading

Constructor Detail

Recipe
public **Recipe**(swarm.defobj.Zone aZone,
int c,
int l,
java.lang.String rN)

Method Detail

getCodeNumber
public int **getCodeNumber**()

---

getOrderRecipe
public int[] **getOrderRecipe**()

---

getLength
public int **getLength**()

---

getRecipeName
public java.lang.String **getRecipeName**()

---

setSteps
public void **setSteps**(java.lang.String cTC,
ExcelReader recipeWorksheet)

---

getCheckTheCell
public java.lang.String **getCheckTheCell**()

---

errorIsNotAnInteger
public int **errorIsNotAnInteger**(ExcelReader e)
This method is used to check if a type error occur

---

errorIsNotAString
public void **errorIsNotAString**(ExcelReader e)
This method is used to check if a type error occur

---

procurement
public void **procurement**(ExcelReader e)
This method dial with the procurement choice

---

oR
public void **oR**(ExcelReader e)
This method dial with the or choice

---

end
public void **end**(ExcelReader e)
This method dial with the end choice

---

number
public void **number**(ExcelReader e)
This method dial with normal or batch choice

---

second
public void **second**(ExcelReader e,
int step)

---

minute
public void **minute**(ExcelReader e,
int step)

I metodi definiti nelle due classi consentono di effettuare una lettura controllata dei dati da due file:

- OrderSequences.xls per la classe Order Distiller;
- Recipe.xls per la classe Recipe.

Nella fase di raccolta dei dati e di inizializzazione dei vettori si verifica, infatti, che la struttura definita per i due file sia rispettata e non si siano verificati errori nel corso dell'inserimento delle informazioni.

Seguiamo ora idealmente il percorso di formazione di un ordine di produzione:

- all'inizio di ogni ciclo l'Order Distiller legge dal foglio di lavoro orderSequences.xls e registra i numeri identificativi delle ricette nel vettore orderSequences1 e delle quantità nel vettore orderSequences2;
- la classe Recipe legge dal file Recipe.xls e registra i passi che compongono ciascuna delle ricette e il loro codice identificativo nel vettore orderRecipe;
- l'Order Distiller è quindi in grado di generare l'ordine e di inviarlo alla prima unità produttiva, simulando il ruolo del Front End.

In questo momento si avvia la produzione di jVE-Vir, che, in termini di volumi, eguaglia quella dell'impresa reale.

6.2.4 L'analisi dei risultati

Il primo passo che seguiamo prima di giungere all'effettiva simulazione è quello di verificare il funzionamento del modello, prendendo come spunto una semplificazione della realtà che simuleremo con il programma jVEFrame.

Questa semplificazione consiste nello sperimentare il modello prendendo in considerazione schemi che riproducono il più fedelmente possibile la formalizzazione che è stata fatta delle sequenze produttive e degli schemi di approvvigionamento materiali e componenti per la ricerca che sta alla base di questo lavoro.

Di seguito vengono presentate le tabelle riportanti:

1. esemplificazione di ricette

```
#  comment    ;
#  comment    ;
       alfa    100  1  s  2    2   s  3  3  s  4  4   s  3  5  s  1  e  100
       beta    200  6  s  1    2   s  1  7  s  2  8   s  3  4  s  2  e  200
       teta    300  p  1  100  9   s  10
#  comment    ;
      sigma    400  p  1  200  10  s  1
```

2. esemplificazione dello schema di approvvigionamento materiali e componenti

```
1    100  *  500    200  *  500                                    ;
2    100  *  500    200  *  500                                    ;
3    100  *  500    200  *  500                                    ;
4    100  *  500    200  *  500                                    ;
5    100  *  500    200  *  500                                    ;
6                                 300  *  200    400  *  300      ;
7    100  *  500    200  *  500                                    ;
8    100  *  500    200  *  500   300  *  3000   400  *  1000     ;
9                                                                 ;
10   100  *  500    200  *  500                                    ;
11   100  *  500    200  *  500   300  *  1200   400  *  1200     ;
12                                                                ;
13   100  *  500    200  *  500                                    ;
14   100  *  500    200  *  500                                    ;
15                                300  *  1100   400  *  300      ;
```

Impostiamo la simulazione con i seguenti parametri:

useOrderDistiller true
attiviamo l'utilizzo delle classi orderDistiller.java e Recipes.java.;

ticksInATimeUnit 28800
consideriamo un numero di passi che possa essere almeno rappresentativo dell'ordine di grandezza che vorremmo osservare in un turno di lavoro dell'azienda presa in considerazione. Se volessimo osservare la realtà completamente dovremmo avere a disposizione degli strumenti in grado di calcolare 28800 tick in ogni ciclo, rappresentativi dei secondi di un turno di otto ore;

totalUnitNumber 97
il numero di unità lavorative in grado di effettuare le lavorazioni necessarie per completare i passi delle ricette produttive;

totalEndUnitNumber 49
il numero di unità di transito che contengono ciò che, nell'applicazione ad un'azienda del settore meccanico, si considerano semilavorati o componenti;

maxStepNumber 30
non va considerato nella simulazione in questione;

maxStepLength 0
non va considerato nella simulazione in questione;

useWarehouses false
la gestione del magazzino e non è simulata, infatti ogni unità lavora ciò che le viene assegnato da un incisore centrale e non vi è creazione di scorte presso le singole unità, la gestione di componenti semilavorati è fatta con la metodologia dei procurement;

useNewses false
la trasmissione di informazioni fra le unità non avviene, come conseguenza della mancanza di magazzini;

maxInWarehouses 10
non va considerato nella simulazione in questione;

minInWarehouses 3
non va considerato nella simulazione in questione;

infDeepness 30
non va considerato nella simulazione in questione; pt

inventoryFinancialRate 0.05
determina il tasso al quale la permanenza delle scorte magazzino senza che siano utilizzate produce un costo;

inventoryEvaluationCriterion 1
abbiamo stabilito di attribuire unicamente i costi variabili di competenza;

revenuePerEachRecipeStep 3
il profitto da attribuire ad ogni passo della ricetta;

nOfNewsesToProduce 1
non va considerato nella simulazione in questione;

nOfNewsesToBeCleared 100000
non va considerato nella simulazione in questione;

nOfOrdersInNewses 1
non va considerato nella simulazione in questione;

orCriterion 0
la scelta effettuata indica di considerare tutti i percorsi alternativi possibili che si presentano nella ricetta

Breve descrizione dei risultati

Prendendo in considerazione il parametro ticksInTimeUnit pari a 28800 il risultato che si ottiene è quello di occupare tutta la memoria e di bloccare per intero l'unità di CPU del computer, in grado di lavorare su una frequenza di 1,34 GigaHertz.

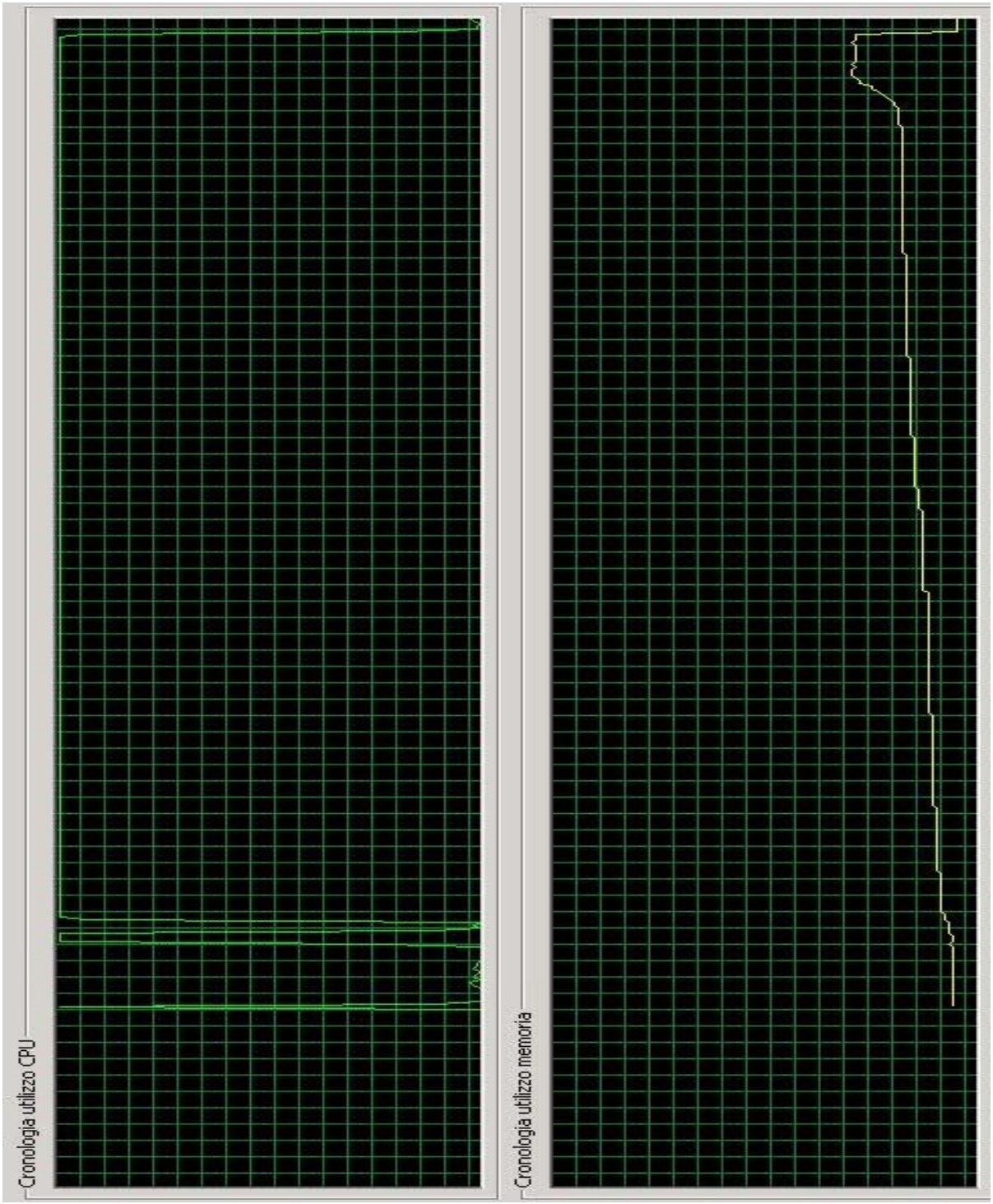


Figura 18: Utilizzo delle risorse del computer con cicli di 28800 tick

Il tempo richiesto da una simulazione esemplificativa, come quella posta in essere sino questo momento, è decisamente elevato e non permetterebbe di effettuare una valutazione reale dell'impresa che interessa in tempi utili, ma soprattutto sarebbe logorante l'attesa per effettuare il confronto fra simulazioni con parametri differenti, che richiederebbero l'esecuzione ripetuta del programma. Il problema principale va ricollegato anzitutto alla notevole quantità di azioni che un agente in una simulazione reale si trova a dover compiere, ma anche ad un più generale problema di gestione delle risorse a disposizione e della scelta delle tecniche di ricerca o di calcolo: ogni agente ed ogni oggetto si mette in relazione con il resto dell'ambiente e con gli altri agenti mediante una tecnica di ricerca di tipo lineare ossia, per esemplificare, quando un oggetto necessita di informazioni o messaggi relativi ad altri oggetti o all'ambiente, invia dei messaggi alle liste di oggetti, scorrendole in maniera sequenziale sino a trovare ciò di cui ha bisogno. Lo sviluppo progressivo delle funzionalità degli agenti e della complessità dell'ambiente pongono in essere problemi di tipo tecnico-tecnologico.

La strategia di ricerca adottata è ben strutturata in quanto è in grado di rispondere al requisito di completezza, ossia è possibile trovare una soluzione quando esiste. Limiti di carattere temporale e spaziale impongono, d'altro canto, di tenere in considerazione altre tre caratteristiche (Russel, Norvig; 1995) che ben si adattano all'impostazione dei problemi di ricerca:

1. Complessità temporale, ossia quanto tempo occorre per trovare una determinata soluzione:
2. Complessità spaziale, ossia quanta memoria occorre per effettuare la ricerca;

3. Ottimalità, la strategia deve trovare la soluzione di qualità massima quando ci sono varie soluzioni differenti;

Nel nostro ambito, per ciò che attiene all'ultima caratteristica, la scelta che viene fatta a riguardo della soluzione è che essa deve semplicemente rispondere ai requisiti richiesti (se mi occorre una lavorazione del tipo 5 andrà bene qualunque unità in grado di effettuare tale lavorazione, indipendentemente dalle code).

Diversa è la questione relativa alle altre due caratteristiche, infatti potrebbe rivelarsi utile in futuro effettuare un confronto fra algoritmi per verificare quale sia più veloce o richieda meno memoria.

Esistono in particolare due modi possibili:

1. il primo consiste nell'usare dei benchmark, ossia dei parametri di confronto in grado di stabilire, dopo aver eseguito i due algoritmi, quale sia il più celere o quale richieda meno memoria. Proprio in questo calcolo si comprendono anche caratteristiche specifiche della macchina e della piattaforma presente; alternativa può essere il conteggio effettivo del numero di operazioni che vengono eseguite;
2. il secondo metodo può essere un'analisi degli algoritmi di tipo matematico, indipendente dalla particolare implementazione. Tale analisi tiene conto principalmente della possibilità di confrontare un numero elevato di volte, tendente all'infinito, i passi compiuti dagli algoritmi ed a quel punto effettuare una analisi asintotica; resta comunque il problema non banale di definire la lunghezza dell'algoritmo.

Altro problema può nascondersi nella crescita esponenziale delle

interrelazioni fra gli agenti e perciò nella complessità sottostante e difficilmente prevedibile.

Per porre in essere una simulazione funzionante ed in grado di fornire dati utili in tempi ragionevoli occorre perciò trovare una soluzione al problema della potenzialità dello strumento. Tale soluzione può seguire due strade opposte:

1. nel primo caso è possibile adottare una convenzione ed effettuare prove che considerino una granulosità del tempo differente da quella dei secondi, consderandone decine o centinaia per ogni tick, valutando però la perdita di dati che si ha in questo modo;
2. la seconda soluzione attiene invece alla possibilità di sviluppare le potenzialità dei calcolatori seguendo la strada dei super calcolatori o cluster.

La prima soluzione consiste nel considerare un numero inferiore di tick all'interno del singolo ciclo di simulazione stabilendo, per convenzione, che ogni istante sia rappresentativo di tanti tick quanto è il denominatore dell'operazione di riduzione effettuata.

Allo stesso tempo occorrerà ridurre proporzionalmente il numero di ordini in programmazione, mentre non sarà necessario effettuare modifiche, salvo che per gli ordini di tipo *stand alone batch,* le cui quantità saranno anch'esse ridotte di conseguenza.

Di seguito sono riportati i risultati di alcune prove:

ticksInATimeUnit 2880;

quantità di secondi occorrenti per ottenere una determinata fornitura di ricette *batch stand alone* 1/10;

numero di lavorazioni interne o richieste di forniture di ricette del tipo *sequential batch* 1/10.

Purtroppo anche in questo caso il risultato non è stato utilizzabile; la situazione è simile alla precedente, infatti, anche se i grafici vengono disegnati, lo scorrere del tempo è troppo lento per giungere a risultati utili.

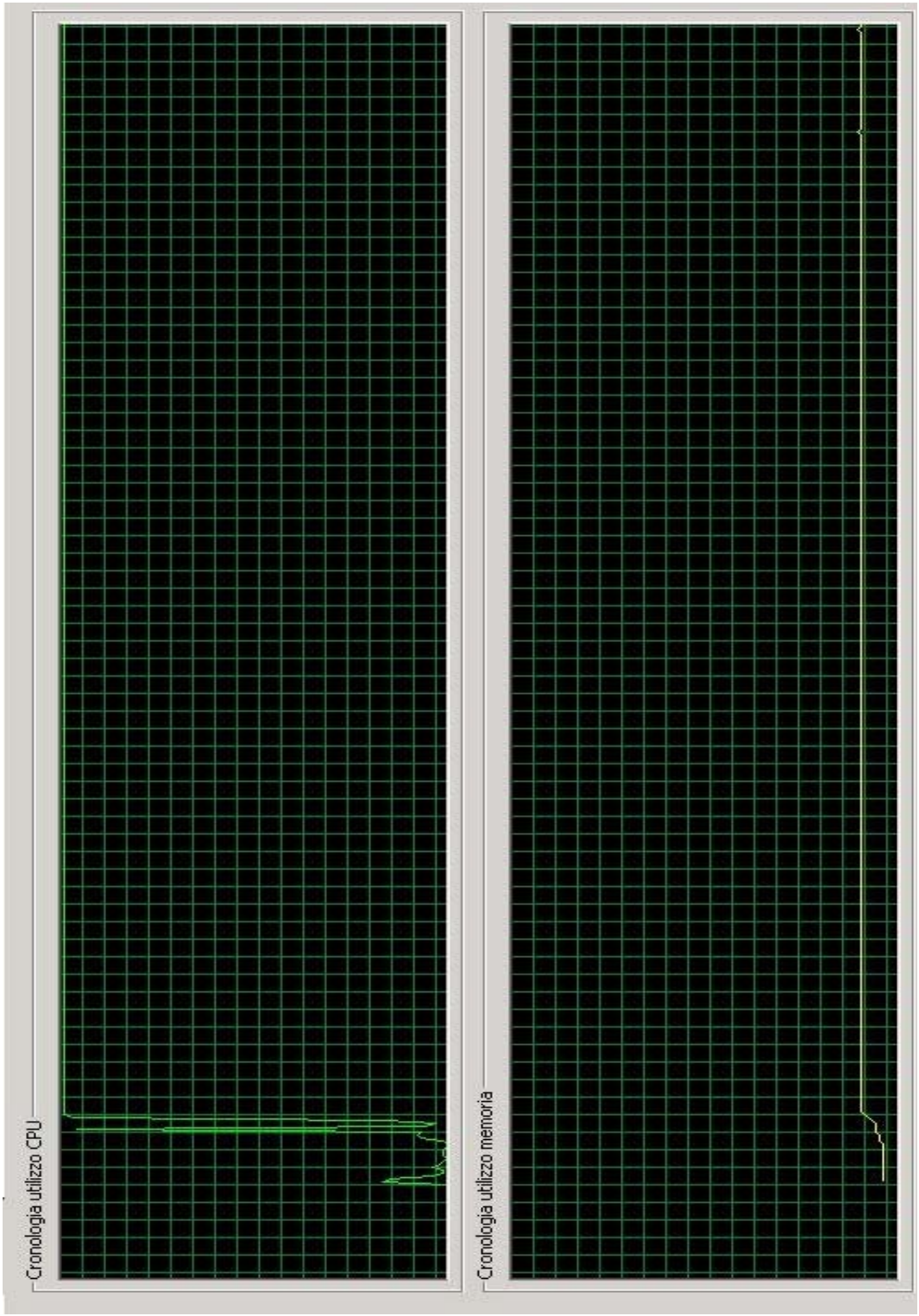


Figura 19: Utilizzo delle risorse del computer con cicli da 2880 tick

Per confronto si può osservare la figura 2, dalla quale si osserva che il problema è dovuto alla limitata capacità di calcolo del computer, problema superabile con l'adozione di computer più potenti o con lo sviluppo di nuove tecnologie.

6.2.4.1 Nuove soluzioni tecnologiche

Nel corso degli ultimi anni si è verificato uno sviluppo costante nelle prestazioni offerte dai personal computer tale da fare sì che un moderno PC possa essere tranquillamente assimilato ad una workstation, una stazione di lavoro dotata di sistemi con buona qualità di grafica, con video di più elevate prestazioni, con CPU di media potenza, spesso attrezzata con unità opzionali orientate alla grafica, quali coprocessori specializzati, scanner e tavolette grafiche e destinata ad attività di studio, professionali o di sviluppo di programmi. L'incremento di prestazioni è senz'altro dovuto allo sviluppo di componenti hardware sempre più sofisticati ed in grado di operare ad altissime velocità. In particolare per quanto riguarda i PC questo incremento di prestazioni è avvenuto senza compromettere la tradizionale economicità di queste macchine, che vantano dunque un sorprendente rapporto prestazioni/costo.
Anche l'hardware di rete si è sviluppato con analoga rapidità ed economicità, permettendo il diffondersi di reti locali veloci ed affidabili. In particolare grande diffusione hanno conosciuto in questi ultimi anni gli switch, ad esempio Fast Ethernet, che permettono di realizzare reti di

interconnessione fra PC e workstation analoghe a quelle dei multiprocessori.

Parallelamente allo sviluppo dell'hardware, c'è stato quello del software: ad esempio, per i moderni PC sono oggi disponibili sistemi operativi dotati di tutte quelle caratteristiche che per anni sono state disponibili esclusivamente per calcolatori di livello superiore (multitasking, multithreading), oltre che dei servizi di rete standard basati sul TCP, UDP e IP. Tali sistemi sono sia commerciali (ad es. Windows 95, Windows NT) che liberamente distribuiti come Linux, una versione di Unix per PC particolarmente interessante perché nella distribuzione sono compresi i sorgenti, modificabili secondo i termini della licenza GNU .

Tutto ciò ha contribuito a determinare l'incredibile diffusione dei PC in ogni ambiente, facendone dunque una realtà di cui non si può non tener conto.

Ben diversa è stata la situazione per quanto riguarda i calcolatori paralleli, siano essi multicalcolatori o multiprocessori . La loro diffusione è stata piuttosto modesta, e i loro costi rimangono tutt'ora elevatissimi, sopportabili solo da un numero piuttosto limitato di enti specializzati. In particolare, ben di rado multicalcolatori sono disponibili nei centri di calcolo universitari, sia per il loro costo che per la necessità di personale specializzato alla loro manutenzione. Inoltre il mercato piuttosto limitato di questi sistemi di calcolo ne rende antieconomico un aggiornamento frequente.

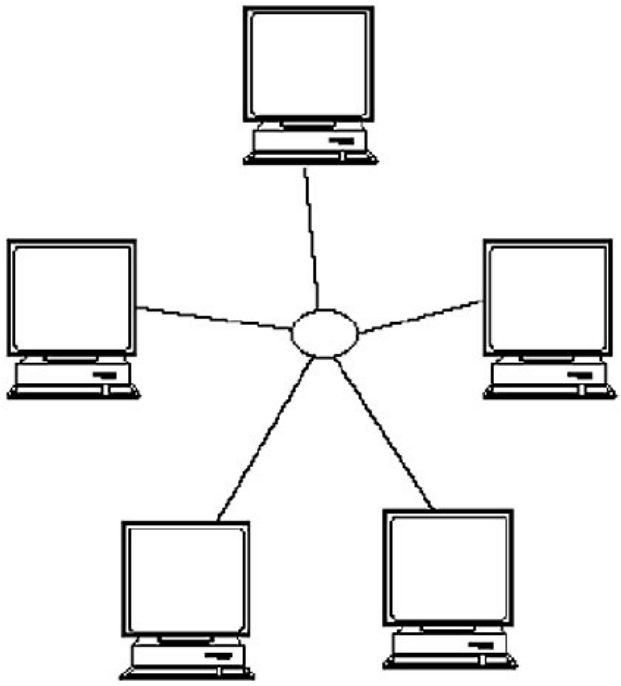


Figura 20: Super cluster

In altri termini spesso essi usufruiscono delle innovazioni nel campo della componentistica con un notevole ritardo.

Proprio per questi motivi è in atto già da alcuni anni in diverse Università e centri di calcolo la tendenza a sviluppare sistemi alternativi ai calcolatori paralleli ``convenzionali'', e particolare sviluppo in questo senso hanno avuto i concetti di rete di workstation (Network Of workstation, NOW) e di cluster di PC (analogo alle NOW ma con PC invece di workstation). L'idea è quella di utilizzare le dotazioni normalmente presenti nei centri di calcolo o comunque acquistabili a prezzi contenuti, e cioè reti locali di PC, dotandole di un sistema operativo che ottimizzi le comunicazioni fra i vari processori, offrendo così una potenza di calcolo il più possibile vicina a quella degli specializzati multicalcolatori e multiprocessori, ma a prezzi decisamente più bassi.

Gli algoritmi che determinano le allocazioni dei task possono essere più o meno efficienti a seconda che operino allocazioni statiche o dinamiche. Le allocazione statiche si rivelano più efficaci nel caso in cui i processori che compongono il cluster siano tra loro omogenei e le attività da distribuire siano tra loro uguali.

Nel caso di allocazione dinamica si distingue tra:

- data-oriented in cui al processore che ha esaurito il proprio task viene assegnato uno dei task in dotazione al processore con la coda maggiore;
- task-oriented in cui al processore che eseguito il proprio task ne viene assegnato uno nuovo.

Un calcolo parallelo consiste di una o più mansioni che si eseguono simultaneamente. Il numero di mansioni può variare durante l'esecuzione di programma; un'operazione incapsula un programma sequenziale e una memoria locale. Come evidenziato in figura 2 un'operazione può realizzare quattro azioni di base oltre che la lettura e scrittura della relativa memoria locale: trasmettere i messaggi sui relativi outports, ricevere i messaggi sui relativi inports, generare le nuove mansioni e terminare.

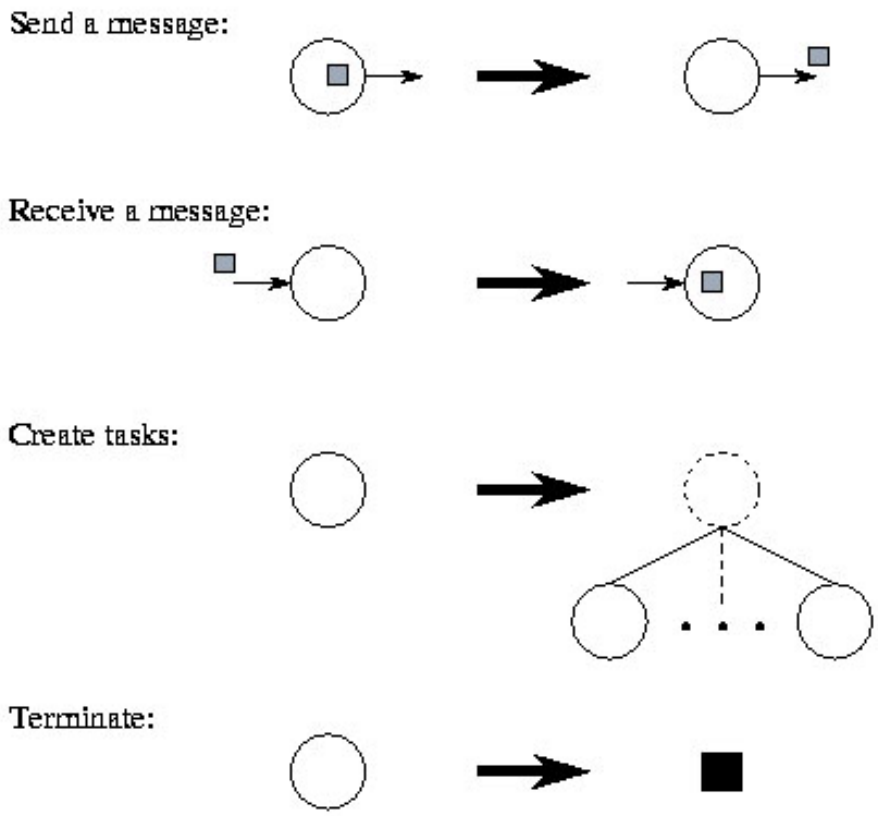


Figura 21: Operazioni generate da un'azione

Un funzionamento di trasmissione è asincrono: termina immediatamente. Un funzionamento di ricezione è sincrono: causa l'esecuzione dell'operazione ostruire fino a che un messaggio non sia disponibile. Gli accoppiamenti di Outport/inport possono essere collegati dalle code di messaggio denominate scanalature.

I primi cluster di PC realizzati per il calcolo parallelo erano poco più che dei “giocattoli”, con prestazioni molto limitate. L'interfaccia verso il programmatore era comunque costituita in genere da MPI (Message Passing Interface) o PVM (Parallel Virtual Machine), le stesse di cui la maggior parte delle piattaforme parallele ``vere'' sono dotate.

Lo scopo di questi cluster era essenzialmente quello di permettere, a costi ridotti, lo sviluppo ed il debugging di applicazioni da eseguire poi su calcolatori paralleli ben più potenti e costosi, nonché l'addestramento, sempre a costi assai contenuti, alla programmazione parallela. Nei cluster di PC le comunicazioni fra i vari host utilizzavano i servizi di rete standard, tipicamente TCP o UDP, su reti locali di vecchia concezione quali Ethernet.

La complessità dei protocolli e la scarsa banda passante della rete fisica rende alta la latenza di comunicazione (tempo occorrente per comunicare un messaggio di lunghezza minima) e limita il throughput (velocità di spedizione dei messaggi vista dall'applicazione) del sistema di comunicazione.

In pratica, i cluster di PC e le NOW erano dei “giocattoli” sia per la limitata potenza di calcolo delle loro CPU sia per la scarsa efficienza delle comunicazioni tra processi, essendo la rete il vero collo di bottiglia di tali sistemi. In pochi anni la situazione è radicalmente cambiata.

Le prestazioni di calcolo delle moderne CPU sono enormemente aumentate. Inoltre, l'aumento della banda passante offerta dalle moderne LAN e il rapido sviluppo degli switch di rete hanno reso disponibili interconnessioni di rete locale a costo contenuto ma con prestazioni confrontabili con quelle delle interconnessioni inter-processore dei calcolatori paralleli commerciali. Vista l'attuale disponibilità di reti con bande di 1 Gigabit/s ed oltre, e la possibilità di dotare ciascun PC di più interfacce di rete, il collo di bottiglia di una NOW o di un cluster di PC è oggi costituito dal bus PCI (banda 133 MByte/s), almeno dal punto di vista hardware. L'atteso sviluppo di bus PCI a 64 bit (banda 266 MByte/s) dovrebbe risolvere tale situazione.

Tuttavia a tale impressionante sviluppo dell'hardware non ha corrisposto un analogo progresso dei protocolli di comunicazione, oggi del tutto inadeguati alle moderne LAN. Di fatto, il collo di bottiglia di una NOW o di un cluster di PC continua ad essere il sistema di comunicazione, ma stavolta più sul piano software che hardware.

Nell'ambito dello studio dell'elaborazione parallela massiccia (*MPP*) svolto alla *NASA* nasce nel 1994 il progetto Beowulf con lo scopo di studiare le potenzialità dei clusters di PC in attività di calcolo proprie delle workstation contemporanee, ma ad un costo non superiore. Nell'ottobre del 1996 è stato annunciato che un sistema Beowulf ha raggiunto una performance di regime di 1 Gigaflops in un'applicazione scientifica, per un costo totale inferiore i $50000, non paragonabile con i costi dei calcolatori paralleli tradizionali, con cui invece la performance è paragonabile.

I sistemi operativi impiegati sono Linux e BSD che supportano un ampio insieme di periferiche e di piattaforme.

Accanto alla necessità di ottimizzare i tempi di calcolo si è detto dell'obiettivo di impiegare tecnologie economiche, per questa ragione molti di questi sistemi di classe Beowulf sono realizzati "riciclando" hardware dai laboratori, o comunque con processori non più attuali e che pertanto non sono utilizzati a causa dell' "evoluzione" del software.

Inoltre le principali librerie per la programmazione parallela basate sullo scambio di messaggi sono facilmente reperibili e permettono di realizzare un sistema "*open source*" basato su questa tecnologia in modo estremamente economico.

Ricorrere all'utilizzo di queste "*pile di pc*" presenta quindi evidenti vantaggi in termini di materiali, tecnologicamente collaudati e facilmente reperibile a costi contenuti, ed in termini di *scalabilità*: se le reti di

comunicazione sono efficienti e, almeno inizialmente, sottosfruttate è possibile incrementare le prestazioni aggiungendo nuove risorse.
Tuttavia queste "pile di pc" non sostituiscono i calcolatori paralleli tradizionali, ma costituiscono un mezzo di calcolo complementare alle workstations di fascia alta, multiprocessori simmetrici e sistemi a memoria distribuita scalabili. Il software per questi sistemi è derivato dal lavoro di collaborazione di un ampio insieme di persone in aree quali i sistemi operativi, linguaggi, compilatori e librerie di calcolo parallele, ed è complessivamente noto con il termine di "Grendel Software Architecture".
La progettazione di software per questa piattaforma non può essere semplicemente basata sui principi della programmazione parallela, ma deve tenere conto anche delle latenze di comunicazione, bilanciando opportunamente l'elaborazione con la comunicazione e del problema che deve essere affrontato. Spesso le comunicazioni di rete sono ottimizzate impiegando più schede di rete su ciascuna macchina, in modo da realizzare un sistema connesso praticamente su più reti, ma che vengono collegate insieme in modo da funzionare come un'unica rete con ampiezza di banda superiore (channel bonding). E' stato provato che per supportare il traffico dati dovuto a trasferimenti di files concorrenti su un sistema di classe Beowulf con sedici processori è sia *necessario che sufficiente* l'uso di due Fast Ethernet a 100 Mbps collegate in *channel bonding*.

Per ogni elemento del cluster possiamo ragionare in termini di *livello e* il modello di riferimento si chiama ISO/OSI.

Figura 22: pila ISO/OSI

Alla base della pila troviamo il *livello fisico* (physical layer) che rappresenta gli aspetti hardware del collegamento tra un computer e l'altro: a questo livello appartengono standard relativi ai mezzi trasmissivi, ai segnali elettrici ed ottici, ai connettori etc.etc.
Sopra il livello fisico troviamo il *livello di collegamento dati* (link layer): è il primo strato software e si occupa di trasferire una sequenza di bit tra due computer adiacenti; fornisce agli strati superiori gli strumenti (*primitive di collegamento*) per inviare blocchi di dati (frame) con una correzione trasparente degli errori di trasmissione.
Sopra il livello di collegamento c'è il *livello di rete* (network layer) che ha il compito di rendere trasparente ai livelli superiori del protocollo la complessità della struttura della rete.
Sopra il livello di rete c'è il *livello di trasporto* (transport layer) il cui compito principale è di offrire, indipendentemente dalla complessità della

rete sottostante, un canale di trasmissione virtuale tra due computer connessi, un canale semplice da usare e con un grado negoziabile di affidabilità nello scambio dei dati.
Sopra il livello di trasporto c'è il *livello di sessione* (session layer) che fornisce le primitive per un dialogo ordinato tra due computer.
Sopra il livello di sessione c'è il *livello di presentazione* (presentation layer) che offre la possibilità di uno scambio di dati indipendententemente dai formati dei dati usati dalla piattaforma sulla quale si opera.
Infine sopra il livello di presentazione c'è il *livello di applicazione* (application layer) che ha il compito di fornire i servizi applicativi, concretizzando l'utilizzo della rete da parte degli utenti.
Attualmente si utilizzano sistemi server ad alte prestazioni o proprio i cluster per gestire gli aspetti di livello più alto delle comunicazioni.

Per osservare i risultati della simulazione, seppur con notevoli differenze dall'impostazione più realistica data all'inizio, proviamo ad effettuare una analisi dei risultati considerando:
ticksInATimeUnit 29;
quantità di secondi occorrenti per ottenere una determinata fornitura di ricette *batch stand alone* 1/1000;
numero di lavorazioni interne o richieste di forniture di ricette del tipo *sequential batch* 1/1000.

Ciò significa essenzialmente che ogni tick della simulazione è rappresentativo di 1000 secondi nella realtà.

I risultati e l'analisi degli stessi è riportata sul sito ospitato dal Dipartimento di scienze economiche e finanziarie "G. Prato" eco83.econ.unito.it/tesive

Parte terza

La notazione UML

Capitolo 7 – Extreme programming (XP) e UML

In questo capitolo ci si propone di valutare i vantaggi e gli aspetti innovativi della nuova metodologia di sviluppo di software (XP) e l'apporto che ad essa può essere fornito dalla notazione UML.
Un aspetto cruciale nello sviluppo di un modello di simulazione di impresa per l'applicazione ad un caso concreto è proprio la collaborazione tra gli sviluppatori del codice e l'impresa cui esso è riferito: la base da cui trae origine la metodologia XP.

7.1 Extreme programming (XP)

(http://www.extremeprogramming.org/) La metodologia di Extreme Programming (XP) nasce nel 1996 da un progetto di Kent Beck presentato alla Daimler Chrysler Research and Technology. Fin dai primi anni novanta Kent Beck, insieme con Ward Cunningham, aveva elaborato e sperimentato una nuova filosofia di sviluppo del software che pareva essere più semplice e più efficiente.
Dalla loro esperienza emergono quattro aspetti cruciali nei progetti di sviluppo di software che possono essere migliorati e che si pongono alla base della filosofia XP:

1. *communication*: tra gli sviluppatori del codice e tra il team che lavora al progetto e i suoi destinatari,
2. *simplicity*: nella struttura del codice e quindi del modello;
3. *feedback*: la collaborazione con i destinatari del progetto per valutare le risposte che si ottengono attraverso il software in corso di sviluppo;

4. *courage*: necessario per affrontare le problematiche di modellizzazione e di astrazione dalla realtà.

Il successo della metodologia XP si fonda sul lavoro di squadra: i responsabili, i clienti e gli sviluppatori sono tutti parte di uno stesso gruppo di lavoro che si pone come unico obiettivo quello di sviluppare un software funzionale e di qualità.
Il confronto tra i programmatori ed i clienti durante la fase di sviluppo del codice permette di valutare la validità del lavoro sviluppato e di adattarla in corso d'opera ai suggerimenti ed alle mutevoli esigenze dei clienti.
La metodologia XP si allontana, in questo senso, in modo significativo dal metodo tradizionale di sviluppo del software.

Figura 1: La filosofia di XP

Secondo questo metodo innovativo, ogni parte del programma è assimilabile ad un pezzo di un puzzle, che assume il suo significato e la sua valenza soltanto nel momento in cui viene combinato insieme agli altri per completare il progetto finale.
XP è una metodologia, con poche regole e un modesto numero di pratiche facili da seguire, maturata osservando ciò che rallenta lo sviluppo di software e analizzando gli aspetti che ne accelerano il processo, è un ambiente in cui i programmatori sono liberi di essere creativi e produttivi, ma organizzati e concentrati su un obiettivo comune.

I passi attraverso i quali si articola lo sviluppo del software sono i seguenti:

1. *planning*;
2. *designing*;
3. *coding*;
4. *testing*.

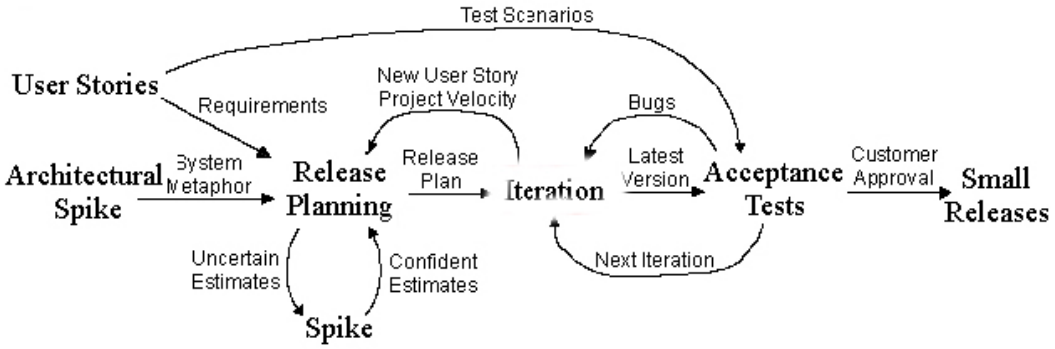


Figura 2: Extreme programming project

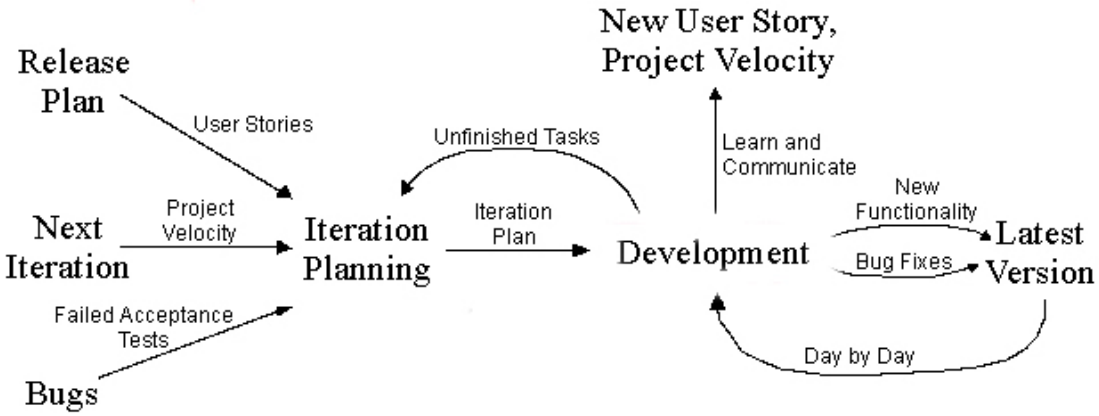


Figura 3. Iteration

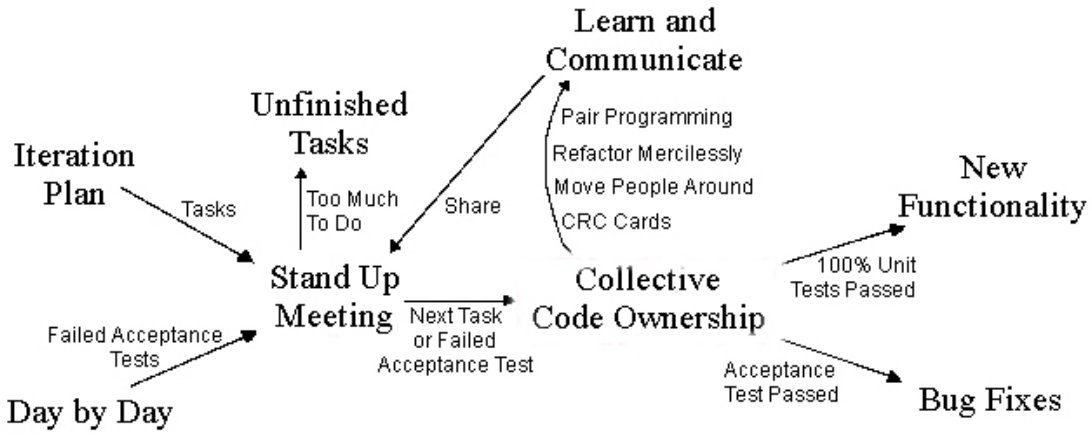


Figura 4: Development

Le fasi indicate non si susseguono nel tempo, come avviene secondo la metodologia tradizionale, ma sono tra loro strettamente interrelate e si sviluppano contemporaneamente, nelle figure 2, 3 e 4 è illustrata l'organizzazione di un progetto di XP.

Appartengono alla fase di *planning*:

- *user stories:* sono scritte dai clienti e contengono l'indicazione delle necessità che il programma dovrà soddisfare;
- *release planning meetings:* consentono al gruppo di lavoro di stimare i tempi necessari per lo sviluppo del codice relativo ad ogni user story. In queste riunioni, in accordo con i clienti, si decide la priorità da assegnare ad ogni user story e quindi si definiscono i "calendari" di realizzazione del codice;
- *small releases*: sono necessarie al team di sviluppo per avere frequenti confronti con il cliente ed introdurre nel progetto le modifiche necessarie a renderlo più funzionale;
- *project velocity*: è un indicatore del lavoro svolto sul progetto in ogni *iteration;*

- *iteration planning meetings*: sono fissate all'inizio di ogni iteration, (figura 3) che dura da una a tre settimane, per definire i *programming tasks*;
- *move people around*: una squadra è molto più flessibile se ogni elemento conosce ogni parte del sistema a cui sta lavorando (*cross training*). Se ogni componente del team è in grado di seguire lo sviluppo del codice di ogni parte del sistema è possibile distribuire in modo efficiente il lavoro di sviluppo del codice tra i componenti del gruppo di lavoro e rendere l'intero team più produttivo;
- *stand up meeting*: si tengono ogni giorno per favorire la comunicazione all'interno del team, servono per comunicare problemi, soluzioni trovate e mettere a fuoco lo stato di avanzamento del lavoro.

Alla fase di *designing* appartengono:

- *semplicity*: la formalizzazione del sistema di cui si scriverà il codice deve essere il più semplice possibile;
- *system metaphor*: indica il processo di astrazione della conoscenza tacita attraverso il quale si delinea la struttura del sistema che si dovrà realizzare e si scelgono i nomi degli oggetti che si inseriranno nel codice;

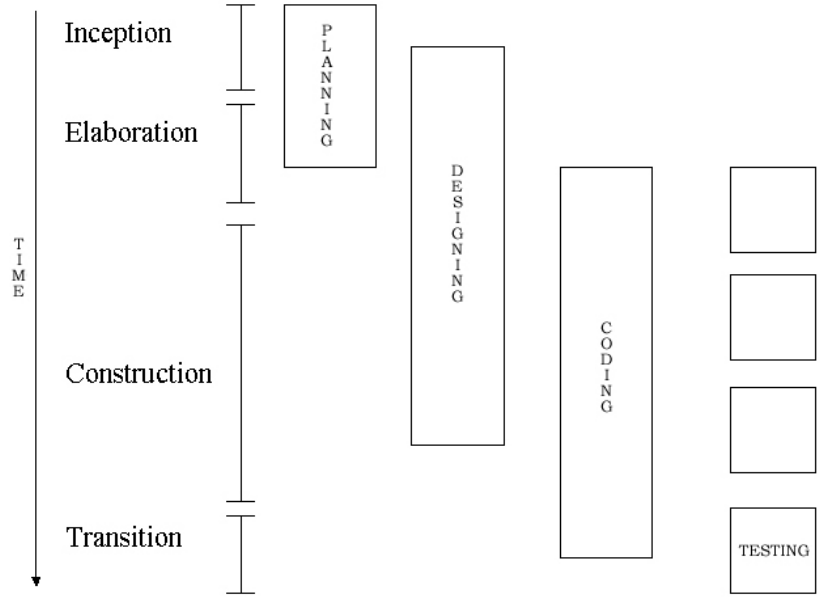


Figura 5: Unified modified process

- *CrC cards:* Class, Responsibilities, and Collaboration (CRC) Cards per presentare il sistema come un team.
-

La fase di *coding* è definita dalle seguenti caratteristiche:

- *consumer always available*: tutte le fasi di un progetto di XP richiedono la comunicazione con il consumatore;
- *coding standards*: il codice deve essere scritto secondo gli standard convenuti;
- *unit test*: consente di controllare i risultati del software in corso di sviluppo;
- *pair programming*: tutto il codice incluso nel progetto è creato da programmatori che lavorano in coppia;
- *integration code problems*: ogni coppia di programmatori verifica il codice che ha sviluppato, al momento dell'integrazione dei diversi moduli si possono tuttavia verificare dei problemi che richiedono un test ulteriore sul progetto integrato con tutte le sue parti;

- *collective code ownership*: ogni componente del team può sviluppare qualsiasi linea del codice;
- *optimization*: è l'ultimo passo nella scrittura del codice.

La fase di *testing* è soggetta ad alcune regole:

- *unit tests*: il funzionamento di ogni classe deve essere sottoposto ad un test per controllare la correttezza del codice; se il test fallisce il modulo non è integrabile con il resto della struttura;
- *bug:* quando si individua un *bug* nel programma si crea un *acceptance test* per individuare dove il problema si colloca nel codice;
- *acceptance test:* rappresenta i risultati che ci si attende dal sistema. Il consumatore che utilizzerà il software è chiamato a confrontare tali risultati con l'ouput del programma e segnalare l'eventuale discrepanza negli *iteration planning meetings*. Lo scopo di questa fase è garantire la soddisfazione delle richieste del consumatore e l'accettabilità del sistema sviluppato.

Durante le fasi di analisi e design di un sistema è possibile utilizzare una notazione grafica object oriented (UML) per rappresentare due diversi punti di vista del sistema: la vista statica, che rappresenti informazioni che non evolvono nel tempo e la vista dinamica, in cui si mostra l'evoluzione delle parti del sistema.

7.2 Unified Modelling Language (UML)

(www.omg.org/uml/) UML (Unified Modelling Language) è l'evoluzione e l'unificazione (da cui il nome) di tre notazioni

precedentemente esistenti: Booch (dell'omonimo autore), OMT (di Rumbaugh) e OOSE (di Jacobson).
UML riunisce aspetti dell'ingegneria del software, delle basi di dati e della progettazione di sistemi, è sufficientemente espressivo e preciso, ma può essere esteso, è indipendente da qualsiasi linguaggio di programmazione ed è usabile con i più diffusi linguaggi ad oggetti, è utilizzabile in domini applicativi diversi e per progetti di diverse dimensioni ed è estensibile per modellare diverse realtà.
Nel 1997 UML è diventato uno standard approvato dall'OMG (Object Management Group).
L'utilizzo di UML non richiede particolare conoscenza teorica, non impone alcun processo di sviluppo predefinito e copre l'intero processo di produzione.
I modelli diventano un veicolo di comunicazione in quanto descrivono in modo "visuale" il sistema da costruire e sono uno strumento per gestire la complessità nella misura in cui consentono di analizzare le caratteristiche particolari del sistema da un punto di vista statico e da uno dinamico (figura 6).
I diagrammi che appartengono alle viste statiche sono:

- *use case diagrams:* catturano le funzionalità del sistema (vista utente)*;*
- *class diagrams:* catturano il vocabolario del sistema*;*
- *object diagrams:* rappresentano le istanze e I loro legami*;*
- *component diagrams:* catturano la struttura fisica dell'implementazione*;*
- *deployment diagrams:* catturano la topologia del sistema.

I diagrammi che appartengono alle viste dinamiche sono:

- *sequenze diagrams:* catturano il comportamento dinamico (rispetto al tempo)*;*

- *collaboration diagrams:* catturano il comportamento dinamico (rispetto ai messaggi)*;*
- *statechart diagrams:* catturano il comportamento dinamico (rispetto agli eventi) *;*
- *activity diagrams:* catturano il comportamento dinamico (rispetto all'attività).

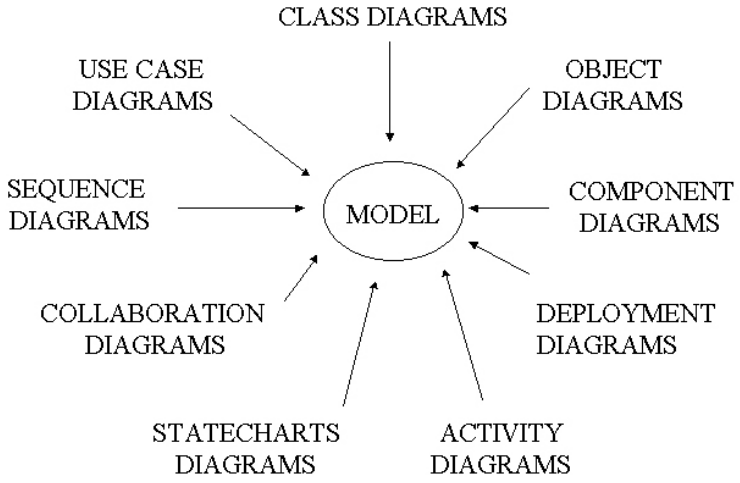


Figura 6: Relazioni tra i diagrammi e il modello

Per meglio comprendere la struttura dei diagrammi analizziamo di seguito un sistema di financial trading.

Per rappresentare la dinamica del sistema si crea un *activity diagram* (figura 7), un flow chart attraverso il quale esplicitare i flussi di controllo da una attività a un'altra. Con le linee orizzontali si indicano azioni che possono essere svolte in contemporanea, il rombo evidenzia la possibilità di una scelta.

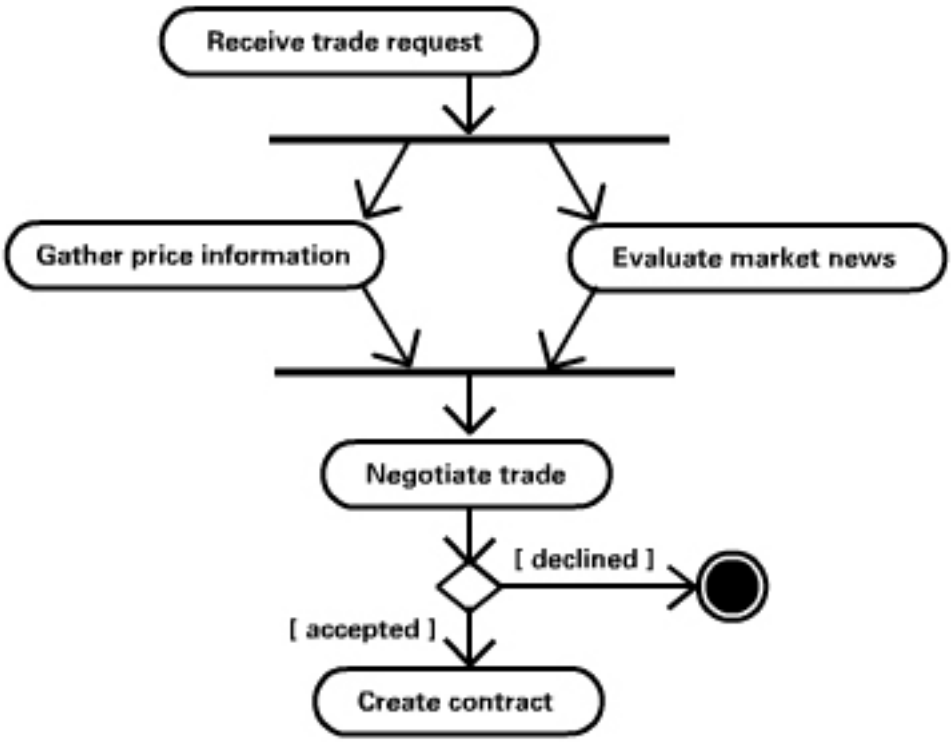


Figura 7: Activity diagram

Il *use case diagram* (figura 8) definisce le funzionalità dei processi principali e come il sistema agisce e reagisce. Descrive il sistema, l'ambiente e le relazioni tra il sistema e l'ambiente a diversi livelli di dettaglio. Nel caso preso in considerazione si possono identificare tre attori:

- trader;
- trader contact
- market data provider:

e cinque *use case:*

- analyze potential trade;
- distribute market news;
- distribute news;
- trade commodities
- generate contract.

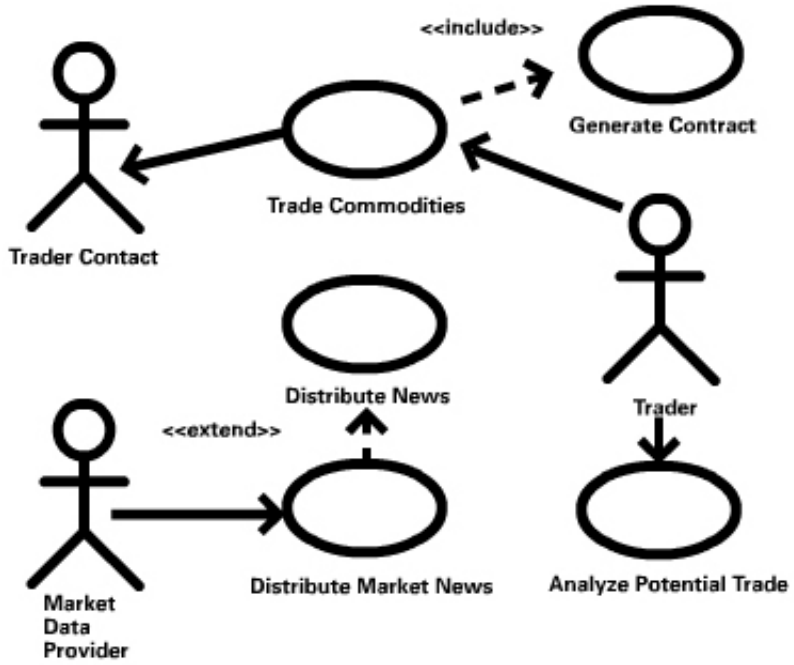


Figura 8: Use-case diagram

Dal diagramma *use case* è possibile ricavare l'*interacion diagram* (figura 9) in cui le relazioni tra il sistema e l'ambiente sono descritte secondo una sequenza grafica (*collaboration diagram*) e una temporale (*sequence diagram*):

1. new;
2. new;
3. register (trade);
4. notift (trade);
5. update (trade);
6. distribuite news (trade);

Le entità rappresentano gli oggetti, mentre i messaggi scambiati i metodi.

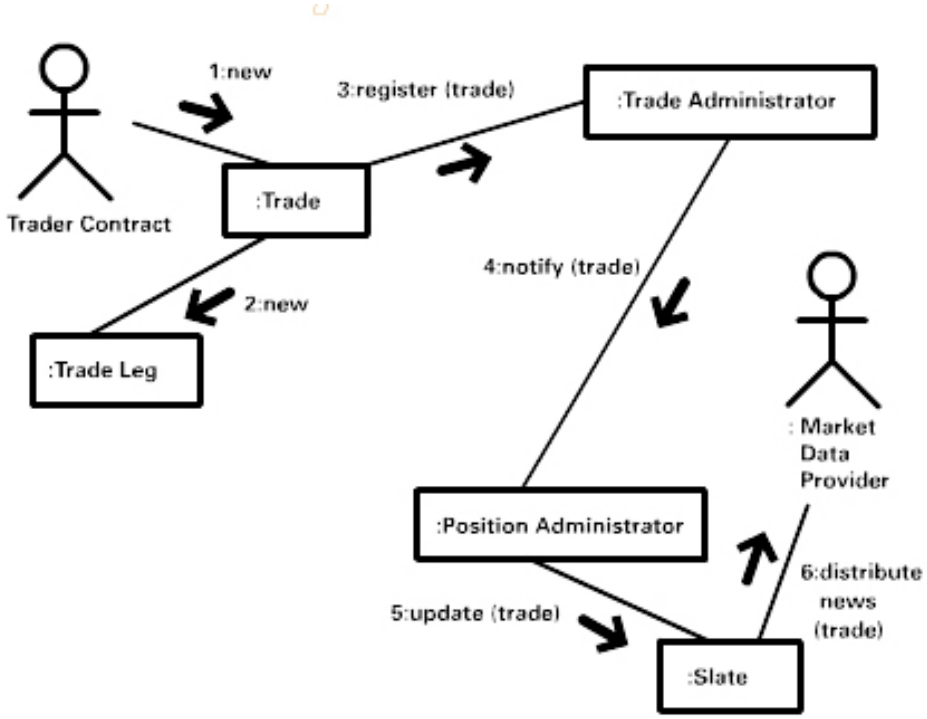


Figura 9: Interaction diagram

Il *class diagram* (figura 10) definisce gli elementi base del sistema e la sua visione statica con l'individuazione delle classi e delle relazioni tra di esse:

- associazione (uso): ha un nome e una direzione di lettura;
- generalizzazione (ereditarietà): esplicita eventuali comportamenti comuni;
- aggregazione (contenimento).

In UML una classe è composta da tre parti:

1. nome
2. attributi (lo stato) sono caratteristiche delle classi che devono essere definite in modo preciso;
3. metodi (il comportamento).

Nel caso del nostro esempio sono identificabili cinque classi:

- trade;

- trade leg
- trade administrator;
- position administrator;
- slate.

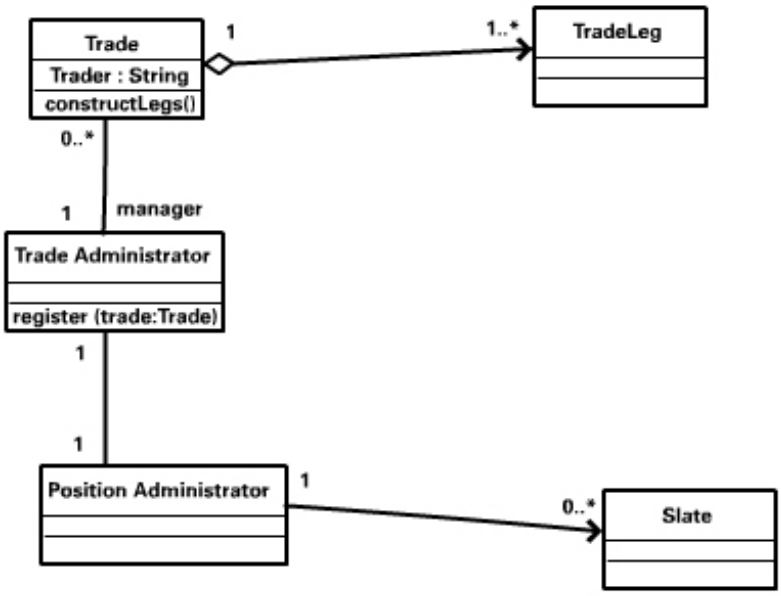


Figura 10: Class diagram

Il numero accanto ad ogni classe è indicativo della molteplicità ovvero il numero minimo e massimo di oggetti che possono essere relazionati ad un altro oggetto e segnala se l'associazione è obbligatoria oppure no. Le relazioni possono essere descritte tramite la cardinalità, il nome e un particolare elemento grafico che ne chiarisca il tipo (figura 11).
Le istanze delle classi e i loro legami sono rappresentate negli *object diagrams;* questi diagrammi sono utilizzati durante l'analisi e il progetto per comprendere la struttura di oggetti complessi, esplicitarla e presentare un'immagine del sistema.

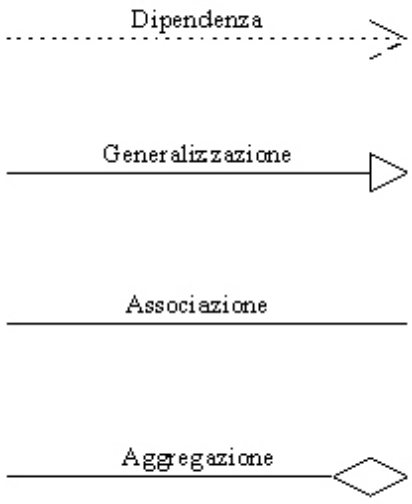


Figura 11: Adornments

Lo *statechart diagram* (figura 12) rappresenta il comportamento dei singoli oggetti di una classe in termini di:

- eventi a cui gli oggetti (la classe) sono sensibili;
- azioni prodotte;
- transizioni di stato (identificazione degli stati interni degli oggetti).

Lo stato è una situazione, durante la vita di un oggetto, che soddisfa alcune condizioni: svolge alcune attività o attende alcuni eventi.
Le operazioni che sono illustrate in questo diagramma si possono dividere tra:

- azioni: hanno durata istantanea e sono associate alle transizioni di stato oppure all'ingresso o all'uscita da uno stato;
- attività: hanno durata significativa e sono associate ad uno stato, possono essere continue o sequenziali.

Nel caso dell'esempio che stiamo analizzando si individuano cinque stati:

- initialization;
- estimation;
- actualization;

- executed.

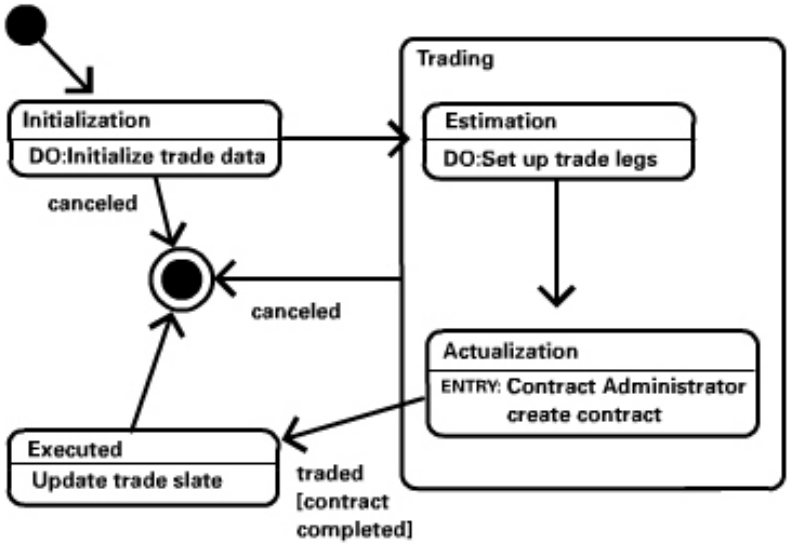


Figura 12: Statechart diagram

In uno *statechart diagram* si possono individuare due tipi di special state:

1. start state:  che è sempre singolo;
2. stop states:  che possono essere più di uno.

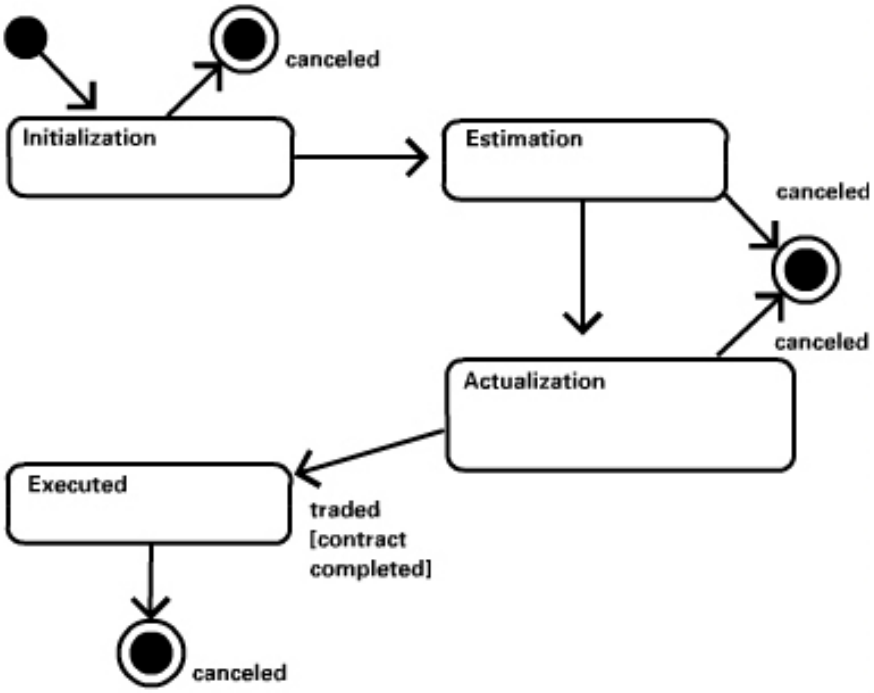


Figura 13: Special state

I diagrammi di *special state* possono essere semplificati introducendo nodi di *stop state* nidificati:  la semplificazione genera i diagrammi di *nested state* (figura 14).

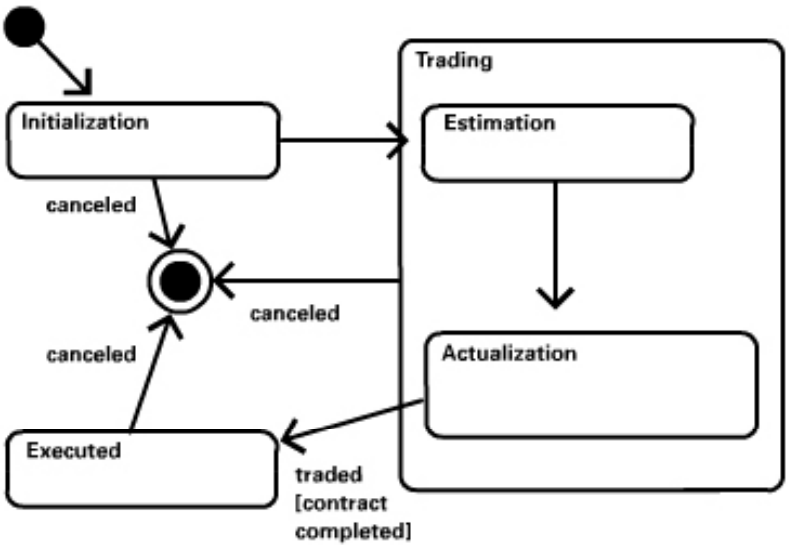


Figura 14: Nested state

Il *component diagram* (figura 15) è utilizzato per illustrare la struttura fisica del codice.

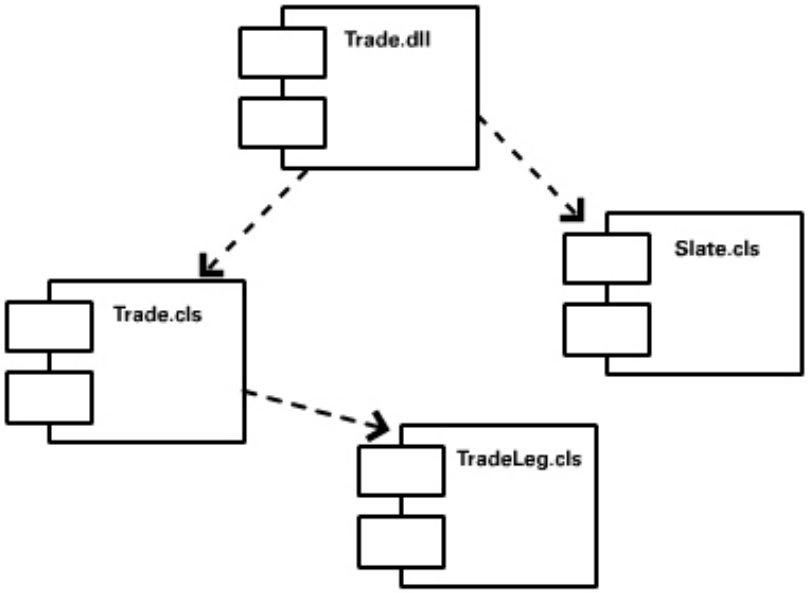


Figura 15: Component diagram

Il *deployment diagram* (figura 16) definisce la configurazione al momento del run time dei componenti individuati al passo precedente e identifica processi e processori (nodi) dell'applicazione.

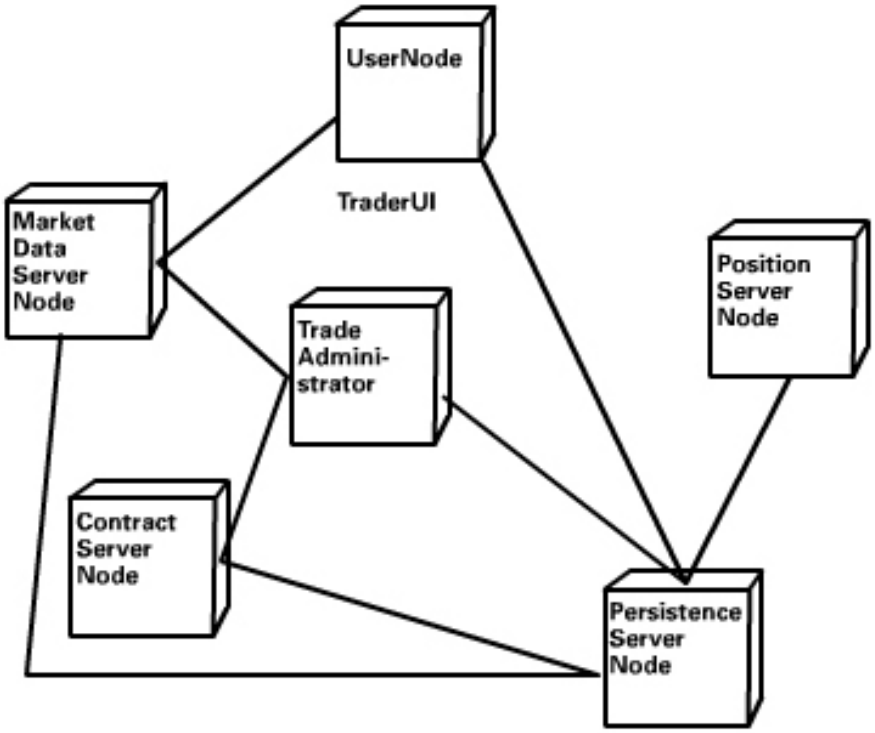


Figura 16: Deployment diagram

7.3 Tools che supportano la notazione UML

Alcuni fra i tools che supportano, almeno in parte, la notazione UML e di cui è disponibile una versione freeware, shareware o dimostrativa sono:

- Aonix - Software Through Pictures & Select Enterprise: http://www.aonix.com/;
- ArgoUML (open source): http://argouml.tigris.org/
- Computer Associates - Paradigm Plus:
- http://www.cai.com/products/alm/paradigm_plus.htm
- Embarcadero Technologies - Describe: http://www.embarcadero.com/products/describe/

- Gentleware - Poseidon for UML: http://www.gentleware.com/ svilupato sul progetto di ArgoUML;
- I-Logix - Rhapsody: http://www.ilogix.com/
- Object International - Together Products: http://www.togethersoft.com/
- Oracle - Oracle Designer : http://www.oracle.com/
- OTW Software - OTW Workbench: http://www.otwsoftware.com/
- Popkin - System Architect http://www.popkin.com/
- Rational Software - UML Resource Center http://www.rational.com/uml/
- SOFTEAM - Objecteering http://www.objecteering.com/
- Telelogic - UML Suite http://www.telelogic.com/
- Visio - Software Design and Modeling http://microsoft.com/office/visio/default.htm

Quasi tutti i tool indicati contengono editor dotati di funzioni di *code-completion* che consentono il passaggio immediato dal codice ai diagrammi UML e viceversa.

Per mettere a confronto tra loro gli strumenti di visual modeling UML indicati è necessario valutare le caratteristiche del prodotto e quelle del produttore e quindi selezionare quello che più si avvicina alle proprie esigenze.

Il primo aspetto da valutare tra le caratteristiche del prodotto riguarda l'adeguatezza dei diagrammi e la loro aderenza a UML, che determineranno l'adeguatezza del metamodello che verrà elaborato con l'utilizzazione del software in questione, le tipologie di linguaggi supportati e le possibilità di generazione codice e reverse Engineering. Secondariamente bisogna ponderare le possibilità di gestione dei progetti che il software offre nella prospettiva di un lavoro di gruppo e di necessità di strumenti per la condivisione. Indicativa della qualità del software è poi la documentazione prodotta, i reports predefiniti, le

possibilità di integrazione fra testo e diagrammi. Un altro aspetto da prendere in considerazione è l'usabilità, in termini di presenza di tutorial, funzioni help, assistenza all'uso e intuitività.
Rimangono da valutare gli aspetti grafici, possibilità di visualizzazione selettiva, zoom, stampe e le possibilità di estendibilità.
Restano infine da considerare i requisiti di sistema minimi per un buon utilizzo del software.
Tra le caratteristiche del fornitore è necessario considerare l'affidabilità del produttore, il suo fatturato e gli investimenti dedicati allo sviluppo del software, i costi di licenza fissati dal distributore, i corsi disponibili e il supporto offerto, per individuare i costi di transazione futuri e anticipare i rischi di lock-in.

7.4 Un esempio di utilizzo: Poseidon for UML

Si propone di seguito un esempio di utilizzo di Poseidon for UML Community Edition 1.2.1.
L'esempio ripercorre i passi necessari per creare il *class diagram* relativo al codice di jVE per l'applicazione al caso Vir descritto nel capitolo 6.
Il primo step consiste nell'importazione (figura 17) di tutti i file .java relativi al modello di cui si vuole creare il diagramma.
Le classi sono inserite nella parte sinistra della finestra (figura 18).

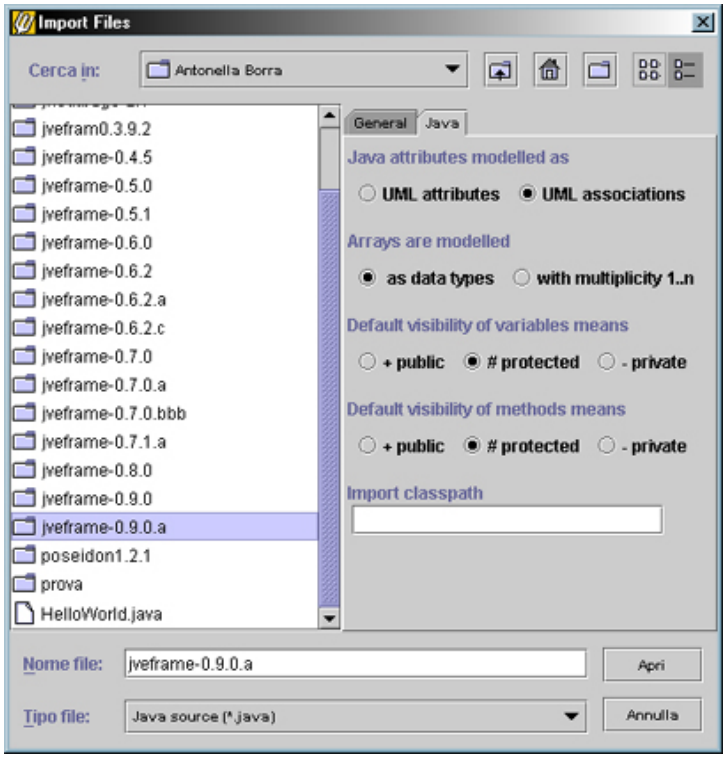


Figura 17: Importazione delle classi

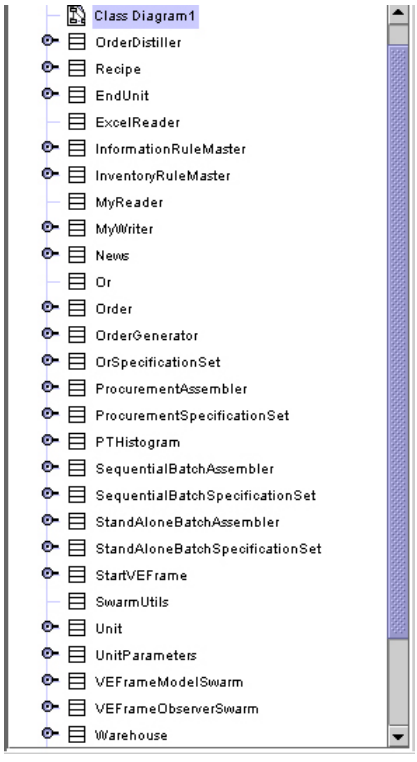


Figura 18: Elenco delle classi

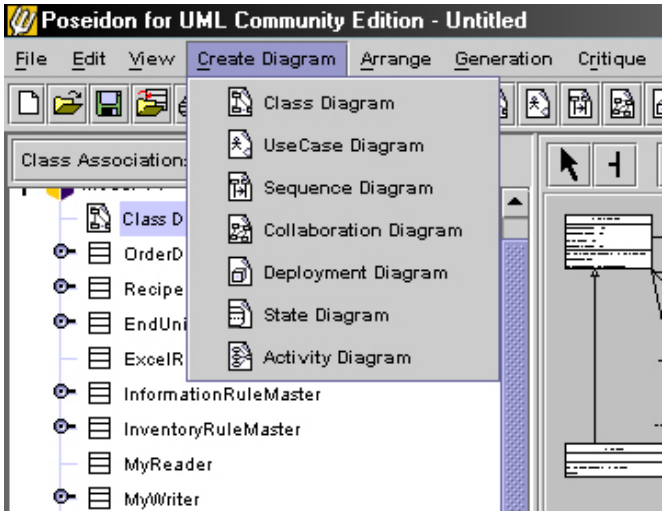


Figura 19: Creazione dei diagrammi

Una volta importate le classi si procede alla scelta del diagramma (figura 19), nel nostro esempio è il *class diagram,* che compare istantaneamente nella parte destra della finestra.

Il diagramma è dinamico da due punti di vista:

- grafico: è possibile spostare all'interno della finestra le classi senza alterare la struttura dei legami che si aggiorna automaticamente;
- strutturale: è possibile modificare le relazioni o le classi esistenti secondo le regole della notazione UML e determinare corrispondenti modifiche al codice.

Il diagramma così generato può essere salvato in un'immagine con estensione gif.

La generazione di codice java attraverso Poseidon comincia con la creazione del diagramma secondo lo standard UML.

Il software permette di disegnare tutti gli elementi che comporranno il codice, di generare le classi e quindi di ottenere il file .java desiderato.

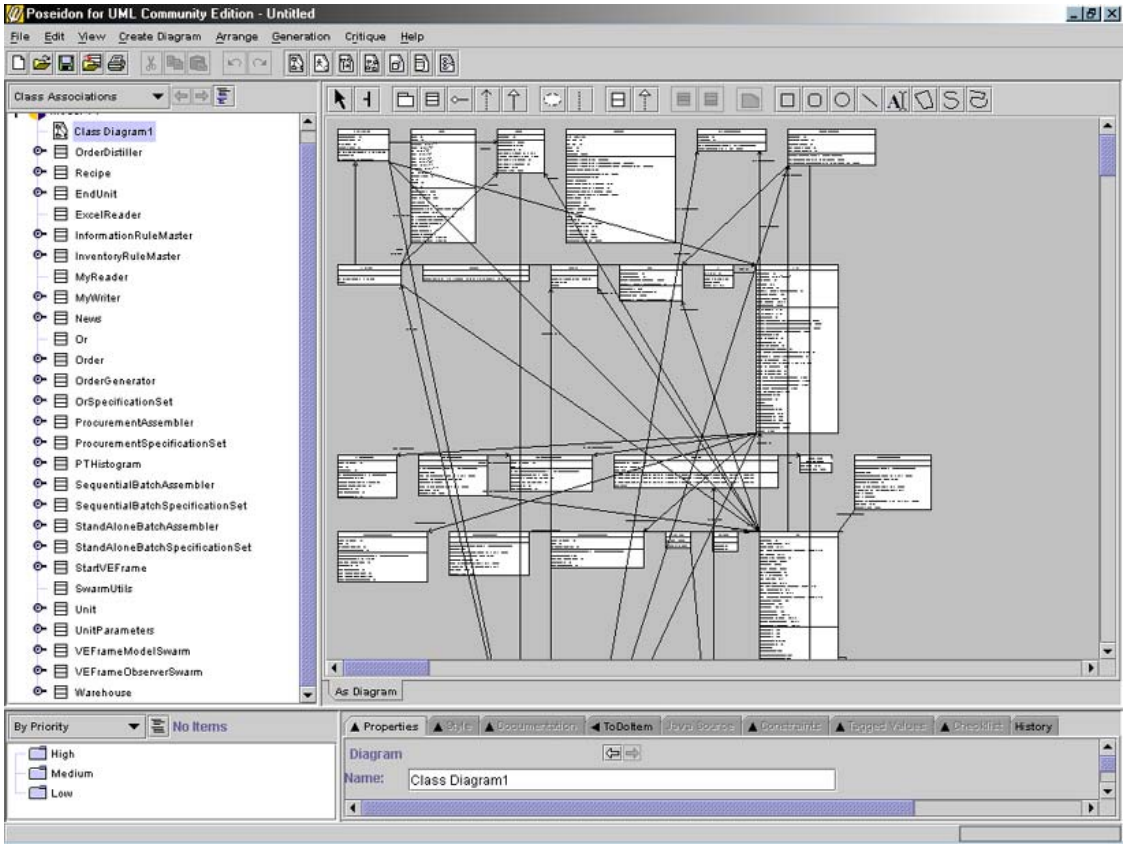


Figura 20: Class diagram di jVE-Vir

Per seguire i passi di generazione del codice si descrive di seguito un esempio estremamente semplificato: un “Hello World”.

In primo luogo si disegna la classe utilizzando gli strumenti messi a disposizione dal software.

Figura 21: Elementi grafici

Nella parte inferiore della finestra (figura 22) si vede comparire il codice generato dagli elementi grafici introdotti.

Per generare il file .java e compilarlo occorre selezionare “Generation” (F7).

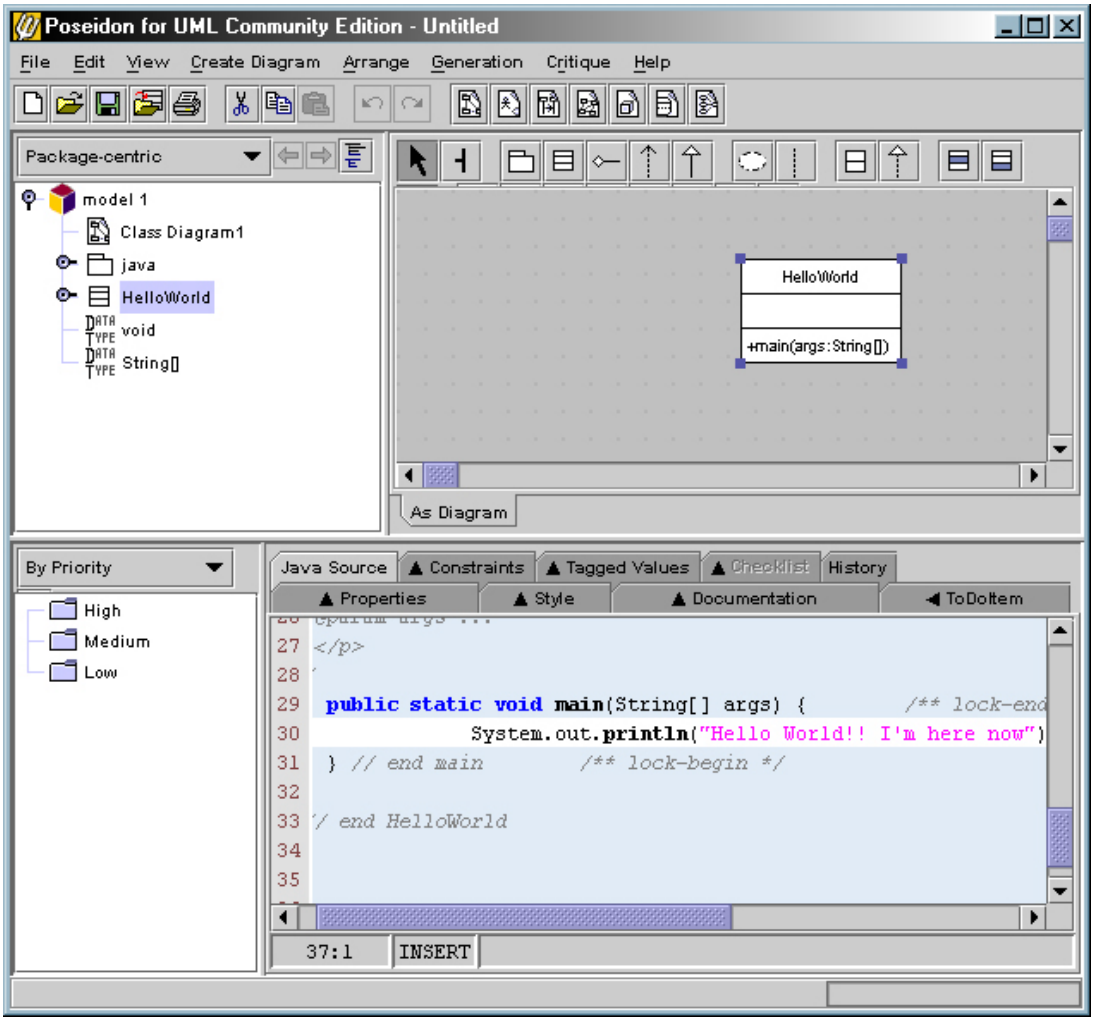


Figura 22: Il class diagram di Hello World

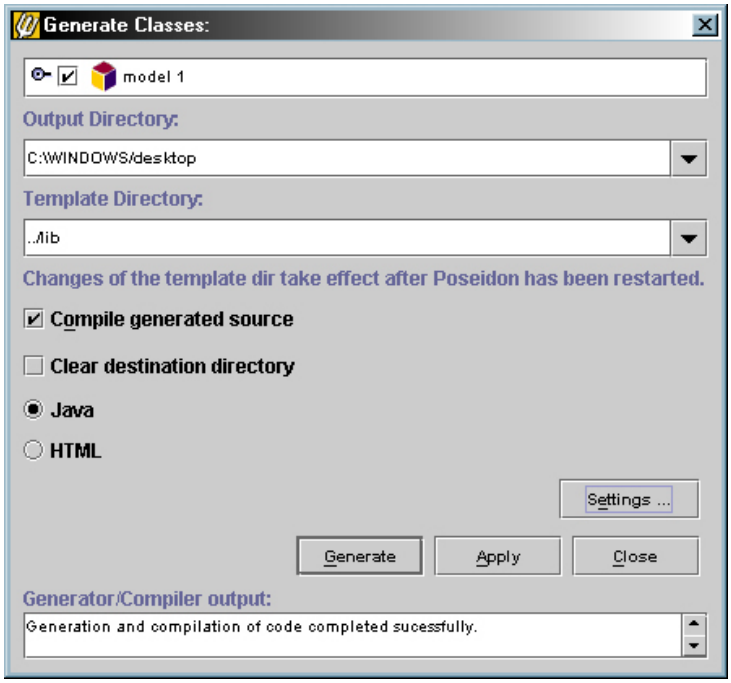


Figura 23: Generate classes

Se nella finestra (figura 23) "Generator/Compiler output" compare il messaggio: "*Generation and compilation of code completed sucessfully*" significa che il codice generato non contiene errori.

Il file HelloWorld.java generato in questo esempio è il seguente:

```
/** Java class "HelloWorld.java" generated from Poseidon for UML.
 *  Poseidon for UML is developed by <A
HREF="http://www.gentleware.com">Gentleware</A>.
 *  Generated with <A HREF="http://jakarta.apache.org/velocity/">velocity</A>
template engine.
 */
import java.util.*;

/**
 * <p>
 *
 * @author Antonella Borra
 * </p>
 */
public class HelloWorld {

  ////////////////////////////////////////
  // operations


/**
 * <p>
 * Does ...
```

```
 * </p><p>
 *
 * </p><p>
 *
 * @param args ...
 * </p>
 */
    public static void main(String[] args) {
        System.out.println("Hello World!! I'm here now");
    } // end main

} // end HelloWorld
```

In conclusione di questo capitolo si può affermare che la nuova metodologia di XP è da considerarsi effettivamente più efficiente, rispetto ai metodi di sviluppo di software tradizionali, specialmente nei casi, come quello oggetto di questa tesi, in cui il lavoro di sviluppo interessi un gruppo di lavoro e il codice sia strettamente legato con una realtà aziendale che, per sua natura mutevole, può determinare necessità di adattamento in corso d'opera.

In questo contesto la notazione UML si pone come un valido supporto per la comunicazione tra gli sviluppatori del codice e l'impresa reale cui esso fa riferimento, in quanto consente di fornire una rappresentazione "visuale" del sistema e del codice in oggetto.

L'importanza del ricorso alla notazione UML non si riduce, tuttavia, alla sola fase di disegno e comunicazione della struttura del sistema, ma risulta apprezzabile per gli stessi sviluppatori di codice nella misura in cui consente loro di scegliere molteplici punti di vista del sistema e permette quindi di comprenderne meglio il funzionamento e gestirne la complessità.

Bibliografia

Bianchi C. (2001), Processi di apprendimento nel governo dello viluppo della piccola impresa – Una prospettiva basata sull'integrazione tra modelli contabili e di system dynamics attraverso i micromondi. Milano, Giuffrè

Boyd D. E., Spekman R. E. (2001), Internet Usage Within B2B Relationships and Its Impact on Value Creation: A Conceptual Model and Research Propositions, Darden Graduate School of Business Administration, Working Paper n. 01-17

Chen J., Lauschke J. J. (july 2001), What Makes Knowledge Tacit? On the Evolution of nowledge-based Industrial Clusters, Department of Finance and Accounting National University of Singapore

Coase R.H. (1960), "The problem of socialcost", Journal of Law and Economics, vol.3, pp. 1-44

Colombatto E. (agosto 2001), Dall'impresa dei neo-classici all'impresa di Kirzner, in Economia Politica, n.2, pp.157-179

Christopher, M. (1992). Logistics and Supply Chain Management: Strategies for Reducing Costs and Improving Service. Pitman Publishing, London, UK.

Do V. T., Halatchev M., Neumann D. A Context-based Approach to Support Virtual Enterprises**,** Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, 4-7 January 2000, Maui, Hawaii, IEEE Computer Society, 2000

Eymann T., Padovan B., Schoder D. (1998), Artificial Coordination - Simulation Organizational Change with Artificial Life Agents, Proceedings of the IFAC Symposium on Computation in Economics, Finance, and Engineering: Economic Systems (CEFES '98), Cambridge, june 29-july 1, 1998

Eyman T., Schoder D., Padovan B. (1998), The Living Value Chain-Coordinating Business Processes with Artificial Life Agents, Proceedings of the Third International Conference and Exhibition on the Practical Applicatio of Intelligent Agents and Multi – Agents, march 23-25, London

Ganeshan R., Harrison T.P., 1995, An Introduction to Supply Chain Management, Penn State University, Working Paper

Garicano L., and Kaplan S. N. (November 2000), The Effects of Business-to-Business E-Commerce on Transaction Costs, in NBER Working Paper, n. W8017

Gilibert N., Pyka A., Ahrweiler P. (2001),Innovation Networks - A Simulation Approach, in Journal of Artificial Societies and Social Simulation, vol. 4, n. 3

v. Hayek F. (1948), Individualism and Economic Order, London, University of Chicago Press.

Hunt J. (2000), The Unified Process for Practitioners-Object Oriented Design, UML and Java. London, Springer

Kimbrough S., Wu D.J., Zhong F. (2001), Computers Play the Beer Game: Can Artificial Agents Manage the Supply Chain?, in R.H. Sprague, Jr. (Ed.), Proceedings of the Thirty-fourth Annual Hawaii International Conference on System Sciences, Los Alamitos, California, IEEE Computer Society Press

Kirzner I. M. (March 1997), Entrepreneurial Discovery and the Competitive Market Process: An Austrian Approach, in Journal of Economic Literature, Vol. XXXV pp. 60–85

Kirzner I. (1973), Competition and Entrepreneurship, Chicago, University of Chicago Press [Concorrenza e Imprenditorialità, Messina, Rubbettino, 1997].

Luna F., Stefansonn B. (eds.) (2000), Economic Simulations in Swarm: Agent-Based Modelling and Object Oriented Programming. Boston Dordrecht and London, Kluwer Academic

Meister D. B. Assessing an Organization's Preparedness for the Virtual Enterprise: The TEMPLET Model, Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, 4-7 January 2000, Maui, Hawaii, IEEE Computer Society, 2000

v. Mises L. (1949), Human Action, London, William Hodge and Company.

NIIIP Consortium, 1998, NIIIP Reference Architecture, (http://www.niiip.org)

Notazione UML
http://www.uml.org/
http://argouml.tigris.org/
http://www.togethersoft.com/
http://www.rational.com/index.jsp

Russel, S.J. (1998), Norvig, P., Artificial Intelligence, a modern approach, Prentice Hall International

Parisi D. (2001), Simulazioni - La realtà rifatta nel computer. Bologna, Il Mulino

Pozzali A., Viale R., Cognizione e Conoscenza Tacita nei Processi Innovativi, Università degli Studi di Milano – Bicocca e Fondazione Rosselli

Rullani E., Il distretto industriale come sistema adattivo complesso, in Complessità e distretti industriali: dinamiche, modelli, casi reali, Fondazione Montedison, Milano 19-20 giugno 2001

Sabbatini P. (giugno 2001), Il B2B e il paradigma dei costi di transazione, in Moneta e credito, n.214, pp139-173

Siems T. F., Gruben W. C., Koo J., Saving J. L. (july 2001), B2B E-commerce: why the new economy lives, in The Southwest Economy, n.4, pp. 1-12

Shapiro C., Varian H. R. (1999), Information Rules - Le regole dell'economia dell'informazione. Milano, Etas Libri

Terna P. (2002), Simulazione ad agenti in contesti di impresa. Sistemi intelligenti, 1, XVI,pp.33-51

Terna P. (2000a), Economic Experiments with Swarm: a Neural Network Approach to the Self-Development of Consistency in Agents' Behavior, in F. Luna and B. Stefansson (eds.), Economic Simulations in Swarm: Agent-Based Modelling and Object Oriented Programming. Dordrecht and London, Kluwer Academic.

Trento S., Wrglien M., Nuove tecnologie e cambiamenti organizzativi: alcune implicazioni per le imprese italiane, Temi di discussione Servizio studi banca d'Italia n.428, dicembre 2001

Ulieru M., Stefanoiu D., Norrie D., Holonic Self-Organization of Multi-Agent Systems by Fuzzy Modeling with Application to Intelligent Manufacturing, in Joint Conference on Information Sciences, JCIS-2000, February 27 -March 3, Atlantic City, USA.

Vir S.p.a. il sito dell'azienda oggetto della simulazione : http://www.moredata.it/anima/vir

VonKortzfleisch H.F.O., Al-Laham A. Structurization and Formalization of Knowledge Management in Virtual Organizations: The Case of a Medium-Sized Consulting Company, Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, 4-7 January 2000, Maui, Hawaii, IEEE Computer Society, 2000

Wu D.J. (2000), Artificial Agents for Discovering Business Strategies for Network Industries, in International Journal of Electronic Commerce, Vol. 5, No. 1, pp. 9-36

Wu D.J., Sun Y., Zhong F. (November 2000), Organizational Agent Systems for Intelligent Enterprise Modeling, in International Journal Electronic Markets, Vol. 10, No. 4, pp. 272–281

APPENDICE A – codice jVE-Vir

-le nuove classi introdotte nel modello jVE-

```
import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;

import swarm.collections.ListImpl;

/**
 * OrderDistiller.java
 *
 * This class is used to read data from two worksheets.
 * The first one contains the list of recipes of our
virtual enterprise.
 * The second one contains a sequence of orders to be
launched,
 * shift by shift, in order to make the daily
production activities
 *
 * Created: Wed May 08 15:29:12 2002
 *
 * @author Cristian Barreca dgbarrec@libero.it Elena
Bonessa elena.bonessa@infinito.it Antonella Borra
anborra@libero.it
 */
public class OrderDistiller extends OrderGenerator{

    // INSTANCE VARIABLES
    /**A flag to check if the orderSequences worksheet
file is open
     */
    public boolean worksheetOrderSequenceFileOpen =
false, worksheetRecipeFileOpen = false;

    /**The arrays containing: the sequence of
orders to be done in the related shift and the
respective quantity
     */
    int[] orderSequence1, orderSequence2;

    /**The variable referring to the
orderSequences worksheet file
     */
    ExcelReader orderSequenceWorksheet = null,
recipeWorksheet = null;
    /** The name of the recipe
     */
    String recipeName;

    /**Flags to operate checks during the reading
     */
    public String semicolon = ";", checkTheCell,
gate = "#", p = "p", sec = "s", min = "m", end =
"e", slash = "/", backslash = "\\", or = "||";

    /**An object to record the array containing
the steps of the recipes in a List
     */
    Recipe aRecipe;

    /** used to record the number of generated
orders
     */
    public int orderCount = 0;

    /** a specific order
     */
    public Order anOrder;

    /** the list containig the operating units
     */
```

```
    public ListImpl unitList;

    /** the list containig the end units
     */
    public ListImpl endUnitList;

    /** the list containig all the orders
     */
    public ListImpl orderList;

    /** the list containig all the orders
     */
    public ListImpl recipeList;

    /**The array containing the steps of the recipe and
the steps of procurement
     */
    int[] orderRecipe;

    // units
    Unit aUnit;
    EndUnit anEndUnit;

    public boolean unitNotFound;
    int j, row, length, choice;

    // CONSTRUCTOR
    public OrderDistiller (Zone aZone, int msn, int
msl, ListImpl ul,
                       ListImpl eul, ListImpl ol,
VEFrameModelSwarm mo){

      super(aZone, msn, msl, ul, eul, ol, mo);

      unitList=ul;
      endUnitList=eul;
        orderList=ol;
    }

    /**This method is used to collect the names
of the units and of the end units, so that it can
operate the check of
     * corrispondency between the production
phases required by recipes and the phases of
production the units can do.
     */
    public void setDictionary(){

      super.setDictionary();

      // It will contain the ID codes of the
recipes worksheet file
      orderSequence1 = new int[dictionaryLength];
      // It will contain the quantity of each
recipe to be done
      orderSequence2 = new int[dictionaryLength];
      for( int i = 0; i < dictionaryLength; i++){
          orderSequence1[i] = 0;
          orderSequence2[i] = 0;
      }

      setRecipeContainers();
      readRecipes();
    }

    /**This method is used to set the length of
each recipe
     */
    public void setRecipeContainers(){

      recipeList = new ListImpl(getZone());

      if(! worksheetRecipeFileOpen){
```

```
          recipeWorksheet = new
ExcelReader("recipeData/recipes.xls");
          worksheetRecipeFileOpen = true;
      }

      checkTheCell = recipeWorksheet.getStrValue();

      row = 0;
      while(! recipeWorksheet.eof()){
          checkForComments(checkTheCell);
          calculateLength(checkTheCell);
      }

      worksheetRecipeFileOpen = false;
    }

    /** This is the method needed to read and store the
recipes. The procedure follows this steps:<br/> It
opens the worksheet file recipes.xls, in which are
stored the sequences of steps of all the
recipes;<br/>It makes some check of the routines;
<br/>It search the object containig the same code of
the recipe we are considering;<br/>
     * Finally it substitutes the values of the array
of that object with the new ones.
     */
    public void readRecipes(){
      //local variables
      int code = 0, i;
      boolean recipeCodeNotFound;

      if(! worksheetRecipeFileOpen){
          recipeWorksheet = new
ExcelReader("recipeData/recipes.xls");
          worksheetRecipeFileOpen = true;
      }
      row = 0;
      checkTheCell =
recipeWorksheet.getStrValue();
      while(! recipeWorksheet.eof()){

          checkForComments(checkTheCell);
          code =
errorIsNotAnInteger(recipeWorksheet);

          recipeCodeNotFound=true;
          for (i = 0; i < recipeList.getCount()
&& recipeCodeNotFound; i++)
            {
                aRecipe = (Recipe)
recipeList.atOffset(i);
                if(aRecipe.getCodeNumber() ==
code)recipeCodeNotFound = false;
            }

          aRecipe.setSteps(checkTheCell,
recipeWorksheet);
          checkTheCell =
aRecipe.getCheckTheCell();
      }
    }

    /**This is the method containing the iterator
needed to launch the daily production of recipes.
     * It take a look at the orderSequence arrays
to determine which recipes must be done and how
many times.
     * A request for which unit can do the first
production phase of each recipe will be done to
units or endUnits.
     */
    public void distill(){
      //local counters
      int i, ii, iii, k, code, quantity;
```

```
      boolean recipeCodeNotFound;
      i=0;

      readOrderSequence();
      while(i < dictionaryLength && orderSequence1[i]
!= 0){
           code =  getOrderSequence1(i);
           quantity = getOrderSequence2(i);

           recipeCodeNotFound = true;
           for (ii = 0; ii < recipeList.getCount() &&
recipeCodeNotFound; ii++)
             {
                 aRecipe = (Recipe)
recipeList.atOffset(ii);
                 if(aRecipe.getCodeNumber() ==
code)recipeCodeNotFound = false;
             }
           for(k = 0; k < quantity; k++){

             orderCount++;
             // creating an order
             anOrder = new Order(getZone(), orderCount,

Globals.env.getCurrentTime(),
                                 aRecipe.getLength(),
aRecipe.getOrderRecipe(),
                                 vEFrameModelSwarm,
endUnitList);


      anOrder.setRecipeName(aRecipe.getRecipeName());

           // add the active orders to the general
order list (they will be
           // eliminated when dropped in a unit, being
finished); this
           // list has been introduced for
accounting purposes [may be it would
           // be better subtitute it with a get
to the units to know their
           // waiting lists]
           orderList.addLast(anOrder);

           /**
            * sending the order to the first
production unit
            * (we are acting as the Front End of
the VE)
            */
           unitNotFound = true;
           for (ii = 0; ii < unitList.getCount()
&& unitNotFound; ii++)
               {
                 aUnit=(Unit)
unitList.atOffset(ii);
                 for (iii = 0; iii <
aUnit.getNOfPhasesToDealWith() && unitNotFound;
iii++)
                   {
                     if(anOrder.getNextStep()
== aUnit.getProductionPhase(iii))
                       {
                         unitNotFound =
false;

      aUnit.setWaitingList(anOrder);
                       }
                   }
               }

           // considering also endUnits
           for (ii = 0; ii <
endUnitList.getCount() && unitNotFound; ii++)
```

```
                {
                  anEndUnit = (EndUnit)
endUnitList.atOffset(ii);
                  if(anOrder.getNextStep() ==
anEndUnit.getUnitNumber())
                      {
                        unitNotFound=false;

      anEndUnit.setTemporaryList(anOrder);
                      }
                }

            if(unitNotFound){
                System.out.println("OrderDistiller
internal error: receiving "+
                                   "unit does not exixts");
                System.exit(1);
            }


          }

          i++;
      }


    }


    /**This method reads from the worksheet containing,
shift by shift, the sequence of orders to be launched
and fills
     * in the orderSequence1 with the ID codes of
recipes and the orderSequence2 with the quantities of
each recipe.
     */
    public void readOrderSequence(){

      //local variables
      int i, ii,numberOfShift;

      i = 0;
      if(! worksheetOrderSequenceFileOpen){
          orderSequenceWorksheet = new
ExcelReader("recipeData/orderSequences.xls");
          worksheetOrderSequenceFileOpen = true;
      }

      numberOfShift =
orderSequenceWorksheet.getIntValue();
      if(StartVEFrame.verbose)
          System.out.println("The shift #" +
numberOfShift + "has begun ");

      while(! orderSequenceWorksheet.eol()){

          orderSequence1[i] =
errorIsNotAnInteger(orderSequenceWorksheet);


errorIsNotAString(orderSequenceWorksheet);
          orderSequenceWorksheet.getStrValue();

          orderSequence2[i] =
errorIsNotAnInteger(orderSequenceWorksheet);
          i++;
      }

      errorIsNotAString(orderSequenceWorksheet);
      orderSequenceWorksheet.getStrValue();



      if(orderSequenceWorksheet.eof()){
```

```
          //   System.out.println("The simulation
finishes here. It is not the right and normal way, but
is only for explanation. Thank you for your attention
and interest.");
          // System.exit(1);
          worksheetOrderSequenceFileOpen = false;
      }
    }


    /**This method is used to check if there are
comments in the cells
     */
    public void checkForComments(String cTC){

      checkTheCell = cTC;
      while(checkTheCell.equals(gate)){
          row++;
          if(StartVEFrame.verbose)
            System.out.println("The row # " + row + "
contains a comment");

          recipeWorksheet.getStrValue();
          checkTheCell = recipeWorksheet.getStrValue();
          if(!
checkTheCell.equals(semicolon))errorIsNotAString(recipe
Worksheet);
          checkTheCell = recipeWorksheet.getStrValue();
      }
    }

    /**This method is used to calculate the length of
each row after a strong check of the elements
     */
    public void calculateLength(String cTC){
      int code = 0;
      checkTheCell = cTC;
      recipeName = checkTheCell;
      code =
errorIsNotAnInteger(recipeWorksheet);
      row++;
      length = 0;
      checkTheCell =
recipeWorksheet.getStrValue();


      while(! checkTheCell.equals(semicolon)){
          if(StartVEFrame.verbose)
            System.out.println("CheckTheCell
contains "+checkTheCell);

          if(checkTheCell.equals(p))   choice =
1;
          if(checkTheCell.equals(or))  choice =
2;
          if(checkTheCell.equals(end)) choice =
3;
          if(recipeWorksheet.checkForLabelCell()
&& !checkTheCell.equals(semicolon))   choice = 4;

          switch(choice){
          case 1:
            if(StartVEFrame.verbose)
            System.out.println("The choice is for
a Procurement");
            procurement(recipeWorksheet);
            break;
          case 2:
            if(StartVEFrame.verbose)
            System.out.println("The choice is for
an Or ");
            oR(recipeWorksheet);
            break;
          case 3:
```

```
            if(StartVEFrame.verbose)
            System.out.println("The choice is for an
End ");
            end(recipeWorksheet);
            break;
          case 4:
            if(StartVEFrame.verbose)
            System.out.println("The choice is for a
Number");
            number(recipeWorksheet);
            break;
          default:
      if(StartVEFrame.verbose)
            System.out.println("No matches were found
reading the worksheet, check for errors inside it");
            System.exit(1);


          }
      }

      if(! recipeWorksheet.eof())
          checkTheCell = recipeWorksheet.getStrValue();



      aRecipe = new Recipe(getZone(), code, length,
recipeName);
      if( StartVEFrame.verbose)
          System.out.println("OrderDistiller: a Recipe
named "+ recipeName+ " with code " + code + "was
born");
      recipeList.addLast(aRecipe);
    }

    /**This method is used to obtain the elements of
the orderSequence1 array
     */
    public int getOrderSequence1(int j){

      return orderSequence1[j];

    }

    /**This method is used to obtain the elements
of the orderSequence2 array
     */
    public int getOrderSequence2(int j){

      return orderSequence2[j];

    }

    /**This method is used to check the
corrispondence with the dictionary of production
phases
     */
    public int checkTheExistence(int c){
      int i, check;
      boolean dictionaryVoice = false;

      check = c;

      for(i = 0; i < dictionaryLength; i++){
          if(check ==
dictionary[i])dictionaryVoice = true;
      }

      if(dictionaryVoice == false){
            System.out.println("The value found
is not a valid number of production");
            System.exit(1);
      }
```

```
      return check;
    }

    /**This method is used to check if a type error
occurs
     */
    public int errorIsNotAnInteger(ExcelReader e){

      if(e.checkForLabelCell())
       {
           System.out.println("The cell should contain
an Integer Value");
           System.exit(1);
       }
      return e.getIntValue();
    }

    /**This method is used to check if a type error
occurs
     */
    public void  errorIsNotAString(ExcelReader e){

      if(e.checkForLabelCell());
      else {
          System.out.println("The cell should contain a
String Value");
          System.exit(1);
      }
    }

    /**This method dial with the procurement choice
     */
    public void procurement(ExcelReader e){
      int numberOfStepsForProcurement = 0;

      numberOfStepsForProcurement = e.getIntValue();
      length += (2 + numberOfStepsForProcurement);
      for(int h = 0; h <
numberOfStepsForProcurement; h++) checkTheCell =
e.getStrValue();
      checkTheCell = e.getStrValue();

    }

    /**This method dial with the or choice
     */
    public void oR(ExcelReader e){

      length +=2;
      e.getStrValue();
      checkTheCell = e.getStrValue();

    }

    /**This method dial with the end choice
     */
    public void end(ExcelReader e){

      length++;
      e.getStrValue();
      checkTheCell = e.getStrValue();

    }

    /**This method dial with normal or batch
choice
     */
    public void number(ExcelReader e){

      checkTheCell = e.getStrValue();
      if(checkTheCell.equals(sec)) second(e);
      else if(checkTheCell.equals(min))minute(e);
      else{
          if(StartVEFrame.verbose)
```

```
          System.out.println("Time is not expressed in
minutes or seconds. Check the worksheet");
          System.exit(1);
      }

    }


    public void second(ExcelReader e){
      int numberOfSteps = 0;

          numberOfSteps = e.getIntValue();
          checkTheCell = e.getStrValue();

          if(checkTheCell.equals(slash) ||
checkTheCell.equals(backslash)){

            length +=4;
            e.getStrValue();
            checkTheCell = e.getStrValue();

          }

          else {
            length += numberOfSteps;
          }

    }

   public void minute(ExcelReader e){
      int numberOfSteps = 0;

          numberOfSteps = (e.getIntValue() * 60);
          checkTheCell = e.getStrValue();

          if(checkTheCell.equals(slash) ||
checkTheCell.equals(backslash)){

            length +=4;
            e.getStrValue();
            checkTheCell = e.getStrValue();

          }

          else {
            length += numberOfSteps;
          }


    }

} // OrderDistiller
```

```
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;

/**
 * Recipe.java
 *
 *
 * Created: Wed May  13 15:29:12 2002
 *
 * @author Cristian Barreca<a
href="mailto:dgbarrec@libero.it"></a><br/>Elena
Bonessa<a
href="mailto:elena.bonessa@infinito.it"></a><br/>
Antonella Borra<a
href="mailto:anborra@libero.it"></a>
 * @version 0.6.2.a
 */
```

```
/**This class is used to record the recipes and their
referring number (ID), so we can assign them to a
List*/
public class Recipe extends SwarmObjectImpl{
    //INSTANCE VARIABLES
    /**the array containing the recipe*/
    int[] orderRecipe;

    /**the referring number of the recipe in the
worksheet file*/
    int code;

    /**The length of the array
     */
    int length;

    /**Flags to operate checks during the reading
     */
    public String semicolon = ";", checkTheCell, gate =
"#", p = "p", end = "e", sec = "s", min = "m", slash =
"/", backslash = "\\", or = "||";
    /** The name of the recipe
     */
    String recipeName;

    /** the dictionary to be used to choose the steps
to be included in
     *    a recipe
     */
    int[] dictionary;

    /** length of the dictionary */
    int dictionaryLength;

    OrderDistiller orderDistiller;

    int j, choice, step;
    //CONSTRUCTOR
    public Recipe(Zone aZone, int c, int l,
String rN){

      super(aZone);

      code = c;
      orderRecipe = new int[l];
      for(int j = 0; j < l; j++)orderRecipe[j] =
0;
      length = l;
      recipeName = rN;
    }

    public int getCodeNumber(){

      return code;

    }

    public int[] getOrderRecipe(){

      return orderRecipe;

    }

    public int getLength(){

      return length;

    }

    public String getRecipeName(){

      return recipeName;
```

```
    }

    public void setSteps(String cTC, ExcelReader
recipeWorksheet){


      checkTheCell = cTC;

      checkTheCell = recipeWorksheet.getStrValue();
      j = 0;
      if(StartVEFrame.verbose)
      System.out.println(checkTheCell +" The length of
the recipe n° " + getCodeNumber() + " is " +
getLength());

      while(! checkTheCell.equals(semicolon)){
          if(checkTheCell.equals(p))   choice = 1;
          if(checkTheCell.equals(or))  choice = 2;
          if(checkTheCell.equals(end)) choice = 3;
          if(recipeWorksheet.checkForLabelCell() &&
!checkTheCell.equals(semicolon))   choice = 4;

          switch(choice){
          case 1:
            if(StartVEFrame.verbose)
            System.out.println("The choice is for a
Procurement");
            procurement(recipeWorksheet);
            break;
          case 2:
            if(StartVEFrame.verbose)
            System.out.println("The choice is for an Or
");
            oR(recipeWorksheet);
            break;
          case 3:
            if(StartVEFrame.verbose)
            System.out.println("The choice is for
an End ");
            end(recipeWorksheet);
            break;
          case 4:
            if(StartVEFrame.verbose)
            System.out.println("The choice is for
a Number");
            number(recipeWorksheet);
            break;
          default:
      if(StartVEFrame.verbose)
            System.out.println("No matches were
found reading the worksheet, check for errors
inside it");
            System.exit(1);



          }
      }

      for(int h = 0; h < length; h++)
          if(StartVEFrame.verbose)
          System.out.println("The recipe contains
" + orderRecipe[h] + " in position " + h);
      if(! recipeWorksheet.eof())
          checkTheCell =
recipeWorksheet.getStrValue();

    }

    public String getCheckTheCell(){

      return checkTheCell;

    }
```

```
    /**This method is used to check if a type error
occur
     */
    public int errorIsNotAnInteger(ExcelReader e){

      if(e.checkForLabelCell())
          {
            if(StartVEFrame.verbose)
            System.out.println("The cell should contain
an Integer Value");
            System.exit(1);
          }
      return e.getIntValue();
    }

    /**This method is used to check if a type error
occur
     */
    public void errorIsNotAString(ExcelReader e){

      if(e.checkForLabelCell());
      {
          if(StartVEFrame.verbose)
          System.out.println("The cell should contain a
String Value");
          System.exit(1);
      }
    }
    /**This method dial with the procurement choice
     */
    public void procurement(ExcelReader e){
      int numberOfStepsForProcurement = 0;

      orderRecipe[j] = -1;
      numberOfStepsForProcurement = e.getIntValue();
      orderRecipe[++j] = numberOfStepsForProcurement;
      for(int h = 0; h <
numberOfStepsForProcurement; h++)
          orderRecipe[++j] = e.getIntValue();
      j++;
      checkTheCell = e.getStrValue();

    }

    /**This method dial with the or choice
     */
    public void oR(ExcelReader e){

      orderRecipe[j] = -10;
      orderRecipe[++j] = e.getIntValue();
      j++;
      checkTheCell = e.getStrValue();

    }

    /**This method dial with the end choice
     */
    public void end(ExcelReader e){

      orderRecipe[j] = e.getIntValue();
      checkTheCell = e.getStrValue();

    }

    /**This method dial with normal or batch
choice
     */
    public void number(ExcelReader e){

      step = Integer.parseInt(checkTheCell);
      checkTheCell = e.getStrValue();
      if(checkTheCell.equals(sec)) second(e,
step);
```

```
      else if(checkTheCell.equals(min))minute(e, step);
      else{
          if(StartVEFrame.verbose)
          System.out.println("Time is not expressed in
minutes or seconds. Check the worksheet");
          System.exit(1);
      }
      j++;
    }


    public void second(ExcelReader e, int step){
      int numberOfSteps = 0;

          numberOfSteps = e.getIntValue();
          checkTheCell = e.getStrValue();

          if(checkTheCell.equals(slash) ||
checkTheCell.equals(backslash)){

            if(checkTheCell.equals(slash)){
                orderRecipe[j] = -2;
                orderRecipe[++j] = numberOfSteps;
                orderRecipe[++j] = e.getIntValue();
                orderRecipe[++j] = step;
            }

            else if(checkTheCell.equals(backslash)){
                orderRecipe[j] = -3;
                orderRecipe[++j] = numberOfSteps;
                orderRecipe[++j] = e.getIntValue();
                orderRecipe[++j] = step;
            }
            checkTheCell = e.getStrValue();
          }

          else {
              orderRecipe[j] = step;
              for(int jj = 0; jj < numberOfSteps -
1; jj++)
                  orderRecipe[++j] = step;
          }

     }

   public void minute(ExcelReader e, int step){
       int numberOfSteps = 0;

          numberOfSteps = e.getIntValue();
          numberOfSteps = numberOfSteps * 60;
          checkTheCell = e.getStrValue();

          if(checkTheCell.equals(slash) ||
checkTheCell.equals(backslash)){

            if(checkTheCell.equals(slash)){
                orderRecipe[j] = -2;
                orderRecipe[++j] = numberOfSteps;
                orderRecipe[++j] =
e.getIntValue();
                orderRecipe[++j] = step;
            }

            else
if(checkTheCell.equals(backslash)){
                orderRecipe[j] = -3;
                orderRecipe[++j] = numberOfSteps;
                orderRecipe[++j] =
e.getIntValue();
                orderRecipe[++j] = step;
            }
            checkTheCell = e.getStrValue();
          }
```

```
          else {
            orderRecipe[j] = step;
            for(int jj = 0; jj < numberOfSteps - 1;
jj++)
                orderRecipe[++j] = step;
          }

    }

}//Recipe.java
```

```
//  ProcurementAssembler.java

import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;

import swarm.collections.ListImpl;

/**
 * The ProcurementAssembler class instances are unit
 * assembling the elements of a procurement process
 *
 * @author Pietro Terna
 */
public class ProcurementAssembler extends
SwarmObjectImpl implements
    PTHistogramPlottable{


    /** the unit we are assembling for */
    Unit myUnit;
    /** the list of the order to be assembled with the
procurements */
    public ListImpl waitingList;
    /** the list of end units */
    public ListImpl endUnitList;
    /** the vector of the quantities in the end
units */
    public int [] quantitiesInEndUnits;
    /** the vector of the labels fo the end units
*/
    public int [] endUnitLabels;
    /** a procurement specification set */
    ProcurementSpecificationSet
pendingProcurementSpecificationSet;

    /**
     * the constructor for ProcurementAssembler
     */
    public ProcurementAssembler (Zone aZone,
ListImpl eul)
    {
      // Call the constructor for the parent
class.
      super(aZone);
      int i;

      // the end unit list
      endUnitList = eul;
      quantitiesInEndUnits = new
int[endUnitList.getCount()];
      endUnitLabels = new int
[endUnitList.getCount()];

      for (i=0; i<endUnitList.getCount(); i++)
          endUnitLabels[i]= ((EndUnit)
endUnitList.atOffset(i)).
            getUnitNumber();
```

```
      // the list procurement processes to be assembled
      waitingList = new ListImpl (getZone());

    }

    /** setting the unit we are assembling for */
    public void setUnit(Unit u){
      myUnit = u;
    }

    /** adding an order to the waitingList */
    public boolean setProcurementWaitingList(Order
anOrder){
      waitingList.addLast(anOrder);
      return true;
    }

    /** verifying procurements and eliminating the
orders from our
     *  waitingList */
    public void checkingProcurementsAndFreeingOrders(){
      int n, i, ii, nmax;
      boolean completed;
      Order anOrder;

      nmax = waitingList.getCount();
      if(nmax>0)
          for (n=0;n<nmax;n++)
            {
      if(StartVEFrame.verbose)
                System.out.println(" ");
                // create the table of the quantities
into the end units
                for (i=0;
i<endUnitList.getCount();i++){
                  quantitiesInEndUnits[i]=
                        ((EndUnit)
endUnitList.atOffset(i)).
                        getTemporaryListCount();
      if(StartVEFrame.verbose)
                  System.out.print("eu " +
endUnitLabels[i] + " " +

quantitiesInEndUnits[i] + "   ");
                }
      if(StartVEFrame.verbose)
                System.out.println(
                          " as seen in
procurementAssembler of unit "
                          +
myUnit.getUnitNumber());

                anOrder = (Order)
waitingList.removeFirst();

pendingProcurementSpecificationSet =
                 (ProcurementSpecificationSet)

anOrder.getPendingProcurementSpecificationSet();


if(pendingProcurementSpecificationSet == null){
                 System.out.println("Internal
error in" +
                                    "
procurementAssembler of unit " +

myUnit.getUnitNumber() +
                                    " - order # " +

anOrder.getOrderNumber() +
                                    "waiting for
procurement "+
```

```
                                "has no procurement
specifications.");
                  System.exit(1);
                }

      if(StartVEFrame.verbose)
                {
                System.out.print("proc. spec. of order
" +

pendingProcurementSpecificationSet.
                                getNumber() + " at pos. "
+

pendingProcurementSpecificationSet.
                                getPositionInRecipe() + "
items ");
                for
(i=0;i<pendingProcurementSpecificationSet.

getNumberOfItemsToBeProcured();i++)

      System.out.print(pendingProcurementSpecificationS
et.
                                getItemToBeProcured(i) +
" ");
                System.out.println(" ");
                  }

                // checking endUnits for the required
procurements
                for
(i=0;i<pendingProcurementSpecificationSet.

getNumberOfItemsToBeProcured();i++)
                  for (ii=0; ii<endUnitList.getCount();
ii++)
                        if(endUnitLabels[ii]==

pendingProcurementSpecificationSet.
                          getItemToBeProcured(i))
                         quantitiesInEndUnits[ii]-
-;
      if(StartVEFrame.verbose)
                for(ii=0;
ii<endUnitList.getCount(); ii++)
                  System.out.print("eu " +
endUnitLabels[ii] + " " +

quantitiesInEndUnits[ii] + "   ");
      if(StartVEFrame.verbose)
                System.out.println(
                        " as residual in
procurementAssembler of unit "
                        +
myUnit.getUnitNumber());

                completed=true;
                for (ii=0;
ii<endUnitList.getCount(); ii++)

      if(quantitiesInEndUnits[ii]<0)completed=fal
se;

                if(completed)
                  {
      if(StartVEFrame.verbose)

System.out.println("completed");
                        for
(i=0;i<pendingProcurementSpecificationSet.

getNumberOfItemsToBeProcured();i++)
```

```
                        for (ii=0;
ii<endUnitList.getCount(); ii++)
                            if(endUnitLabels[ii]==

pendingProcurementSpecificationSet.
                             getItemToBeProcured(i))
                            // set the procured items
into
                            // anOrder

      anOrder.setProcuredItemList((Order)
                            ((EndUnit)
endUnitList.atOffset(ii)).

removeFromTemporaryList() );
                  }

                else waitingList.addLast(anOrder);

            }
    }

    /**
     * return the waiting list length
     */
    public int getWaitingListLength()
    {
      return waitingList.getCount();
    }

    /**
     * checking if an order is in the waitingList
     */
    public boolean thisOrderIsInTheWaitingList(Order
o){
      // the result is true if o is a member of
waitingList; false in
      // the opposite case
      return waitingList.contains(o);
    }


    /**
     * PTHistogramPlottable interface method:
getLabel()
     */
    public String getLabel(){
      return "";
    }

    /**
     * PTHistogramPlottable interface method:
getValueToPlot()
     */
    public double getValueToPlot(){
      return (double) getWaitingListLength();
    }

}
```

```
// ProcurementSpecificationSet.java

import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;


/**
```

```
 * The container generated by an order and containing
the spefications about
 * procument necessities
 *
 * @author Pietro Terna
 */
public class ProcurementSpecificationSet  extends
SwarmObjectImpl{

    /** number (i.e. that of the order which has
generated this object */
    public int myOrderNumber;
    /** the position of this sequence in the recipe */
    public int pos;
    /** the specifications */
    public int [] specifications;

  /**
   * constructor for ProcurementSpecificationSet
   */
  public ProcurementSpecificationSet (Zone aZone, int
n, int p) {

     // Call the constructor for the parent class
     super(aZone);

     myOrderNumber=n;
     pos=p;

      if(StartVEFrame.verbose)
     System.out.println("Proc. spec. created " +
                  "(by order " + myOrderNumber + " at
pos. " +
                  pos + "; NB first pos. is 0)");
  }

    /** setting the procurement specifications */
    public void setSpecificationSet(int jj, int
[] r)
    {
      int i;
      specifications = new int[r[jj+1]];

      for (i=0; i<specifications.length; i++)
          specifications[i] = r[jj+2+i];

      if(StartVEFrame.verbose)
      {
      System.out.print("item(s) to be procured
");
      for (i=0; i<specifications.length; i++)
          System.out.print(specifications[i] + "
");
      System.out.println(" ");
      }

    }

    /** returning the number of items to be
procured */
    public int getNumberOfItemsToBeProcured(){
      return specifications.length;
    }

    /** returning the item to be procured at pos.
i of the vector
     *  containg the items
(0<=i<specifications.length) */
    public int getItemToBeProcured(int i){
      return specifications[i];
    }

    /**
```

```
     * returning the number of the order of this
procurement specification
     */
    public int getNumber () {
      return myOrderNumber;
   }

    /**
     * returning the position of this procurement
specification into its recipe
     */
    public int getPositionInRecipe() {
      return pos;
   }

}
```

// **StandAloneBatchAssembler.java**

```
import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;

import swarm.collections.ListImpl;

/**
 * The  StandAloneBatchAssembler class instances are
unit
 * assembling stand alone batch processes
 *
 * @author Pietro Terna
 */
public class StandAloneBatchAssembler extends
SwarmObjectImpl implements
    PTHistogramPlottable{

    /** the unit we are assembling for */
    Unit myUnit;
    /** the list of the order to be assembled
with a stand alone batch
      process */
    public ListImpl waitingList;

    /**
     * the constructor for
StandAloneBatchAssembler
     */
    public StandAloneBatchAssembler (Zone aZone)
    {
      // Call the constructor for the parent
class.
      super(aZone);

      // the list of the stand alone batch
processes to be assembled
      waitingList = new ListImpl (getZone());
    }

    /** setting the unit we are assembling for */
    public void setUnit(Unit u){
      myUnit = u;
    }

    /** adding an order to the waitingList */
    public boolean
setStandAloneBatchWaitingList(Order anOrder){
      waitingList.addLast(anOrder);
      return true;
    }

    /** verifying stand alone batches and
eliminating the orders from our
     *  waitingList in scheduling */
```

```
    public void
checkingStandAloneBatchAndFreeingOrders(){
      // empty method, included for simmetry reasons
and for future use
    }

    /**
     * return the waiting list length
     */
    public int getWaitingListLength()
    {
      return waitingList.getCount();
    }

    /**
     * checking if an order is in the waitingList
     */
    public boolean thisOrderIsInTheWaitingList(Order
o){

      // the result is true if o is a member of
waitingList; false in
      // the opposite case
      return waitingList.contains(o);
    }

    /**
     * removing the order from the waitingList
     */
    public void removeThisOrderFromTheWaitingList(Order
o){

      waitingList.remove(o);
    }


    /**
     * PTHistogramPlottable interface method:
getLabel()
     */
    public String getLabel(){
      return "";
    }

    /**
     * PTHistogramPlottable interface method:
getValueToPlot()
     */
    public double getValueToPlot(){
      return (double) getWaitingListLength();
    }

}
```

// **StandAloneBatchSpecificationSet.java**

```
import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;


/**
 * The container generated by an order and
containing the spefications about
 * a stand alone batch process
 *
 * @author Pietro Terna
 */
public class StandAloneBatchSpecificationSet
extends SwarmObjectImpl{

    /** number (i.e. that of the order which has
generated this object */
```

```
    public int myOrderNumber;
    /** the position of this sequence in the recipe */
    public int pos;
    /** the specifications */
    public int productionTimeInTicks;

  /**
   * constructor for StandAloneBatchSpecificationSet
   */
  public StandAloneBatchSpecificationSet (Zone aZone,
int n, int p) {

     // Call the constructor for the parent class
     super(aZone);

     myOrderNumber=n;
     pos=p;

      if(StartVEFrame.verbose)
          System.out.println("Stand alone b. spec.
created " +
                         "(by order " + myOrderNumber +
" at pos. " +
                         pos + "; NB first pos. is
0)");
  }

    /** setting the stand alone batch specifications */
    public void setSpecifications(int k)
    {
      productionTimeInTicks = k;

      if(StartVEFrame.verbose)
      System.out.println("Stand alone batch production
time in ticks " +
                  productionTimeInTicks);
    }
```

```
    /** decrease production time */
    public int decreaseProductionTime(){

      if(StartVEFrame.verbose)
          System.out.println(
             "Residual production time in stand
alone production set from"+
           " order " + myOrderNumber+" at pos " +
pos +": " +
           productionTimeInTicks);
      productionTimeInTicks--;
      return productionTimeInTicks;
    }


    /**
     * returning the number of the order of this
procurement specification
     */
    public int getNumber () {
      return myOrderNumber;
   }

    /**
     * returning the position of this procurement
specification into its recipe
     */
    public int getPositionInRecipe() {
      return pos;
   }

}
```

// **SequentialBatchAssembler.java**

```
import java.util.Arrays;

import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;

import swarm.collections.ListImpl;

/**
 * The  SequentialBatchAssembler class instances are
unit
 * assembling sequential batch processes
 *
 * @author Pietro Terna
 */
public class SequentialBatchAssembler extends
SwarmObjectImpl implements
    PTHistogramPlottable{

    /** the unit we are assembling for */
    Unit myUnit;
    /** the list of the order to be assembled with a
sequential batch
      process */
    public ListImpl waitingList;

    /**
     * the constructor for SequentialBatchAssembler
     */
    public SequentialBatchAssembler (Zone aZone)
    {
      // Call the constructor for the parent class.
      super(aZone);

      // the list of the sequential batch processes to
be assembled
      waitingList = new ListImpl (getZone());
    }

    /** setting the unit we are assembling for */
    public void setUnit(Unit u){
      myUnit = u;
    }

    /** adding an order to the waitingList */
    public boolean
setSequentialBatchWaitingList(Order anOrder){
      waitingList.addLast(anOrder);
      return true;
    }

    /** verifying sequential batches and
eliminating the orders from our
     *  waitingList in scheduling */
    public void
checkingSequentialBatchAndFreeingOrders(){
      // empty method, included for simmetry
reasons and for future use
    }

    /**
     * return the waiting list length
     */
    public int getWaitingListLength()
    {
      return waitingList.getCount();
    }

    /**
     * checking if an order is in the waitingList
     */
    public boolean
thisOrderIsInTheWaitingList(Order o){
```

```
      // the result is true if o is a member of
waitingList; false in
      // the opposite case
      if(StartVEFrame.verbose)
          if( waitingList.contains(o))

      System.out.println("SequentialBatchAssembler:time
"

+Globals.env.getCurrentTime()+
                           " order "+o.getOrderNumber()+
                           " trapped from unit "+
                           myUnit.getUnitNumber());

      return waitingList.contains(o);
    }

    /**
     * removing the order from the waitingList
     */
    public void removeThisOrderFromTheWaitingList(Order
o){

      waitingList.remove(o);
    }

    /**
     *  if the whole batch exists, we can clear it to
the production
     */
    public void clearToProduce()
    {
      int i, ii, count, countNeeded, countToBeFound,
listCount;
      boolean batchNotFound;
      Order orderToBeChecked;

      for (i=0;i<waitingList.getCount();i++){
      ((Order)
waitingList.atOffset(i)).setInSequentialBatch(fal
se);}

      if(waitingList.getCount()<2)return;
//nothing to do
      batchNotFound=true; countToBeFound=0;

      for (i=0;i<waitingList.getCount()-1 &&
batchNotFound;i++)
          {
            count=0;
            for
(ii=i+1;ii<waitingList.getCount();ii++)
                {

                  if(sameOrderInSequentialBatch(
                                       (Order)
waitingList.atOffset(i),
                                       (Order)
waitingList.atOffset(ii)
                                       ))count++;
                }

            // count is the number of order
equals to the first one

      countNeeded=((SequentialBatchSpecificationS
et)
                ((Order) waitingList.atOffset(i))

.getPendingSequentialBatchSpecificationSet())

.getQuantityInSequentialBatch()-1;
```

```
            if(count>=countNeeded){
                batchNotFound=false;
                countToBeFound=countNeeded;
                ii=i+1;
            }

            if(count>=countNeeded)
                while(countToBeFound>0){
                  if(sameOrderInSequentialBatch(
                                    (Order)
waitingList.atOffset(i),
                                    (Order)
waitingList.atOffset(ii)
                                    ))
                        {
                          ((Order)
                           waitingList.atOffset(ii)).
                              setInSequentialBatch(true);
                          countToBeFound--;
                        }
                  ii++;
                }
            if(! batchNotFound)((Order)
                            waitingList.atOffset(i)).

setInSequentialBatch(true);
          }

      // eliminating from the local list the order
found to complete a batch
      // eliminating them also from the unit
waitingList and setting them
      // together in the list at the current position
      listCount=waitingList.getCount();
      for (i=0;i<listCount;i++){
          orderToBeChecked=(Order)
waitingList.removeFirst();
          // locally
          if(!
orderToBeChecked.getInSequentialBatch())

      waitingList.addLast(orderToBeChecked);
          // in my unit

if(orderToBeChecked.getInSequentialBatch()){

      myUnit.removeFromWaitingList(orderToBeCheck
ed);

      myUnit.setWaitingList(orderToBeChecked);
          }

      }

    }

    /**
     *  if the whole batch is produced, we can
clear it to the propagation
     */
    public void clearToPropagate()
    {
    }

    /**
     *  checking if two orders are in the same
batch
     */

    public boolean
sameOrderInSequentialBatch(Order o1, Order o2)
    {
      if(
```

```
        o1.getRecipeName()==o2.getRecipeName() &&
        Arrays.equals(o1.getRecipeVector(),
o2.getRecipeVector()) &&
        Arrays.equals(o1.getStateVector(),
o2.getStateVector()) &&
        ((SequentialBatchSpecificationSet)

o1.getPendingSequentialBatchSpecificationSet()).

getProductionTimeInTicks() ==
        ((SequentialBatchSpecificationSet)

o2.getPendingSequentialBatchSpecificationSet()).

getProductionTimeInTicks() &&
        ((SequentialBatchSpecificationSet)

o1.getPendingSequentialBatchSpecificationSet()).

getQuantityInSequentialBatch() ==
        ((SequentialBatchSpecificationSet)

o2.getPendingSequentialBatchSpecificationSet()).

getQuantityInSequentialBatch() &&
        (! o1.getInSequentialBatch()) && (!
o2.getInSequentialBatch())
        )
         return true;
      else
         return false;
    }



    /**
     * PTHistogramPlottable interface method:
getLabel()
     */
    public String getLabel(){
      return "";
    }

    /**
     * PTHistogramPlottable interface method:
getValueToPlot()
     */
    public double getValueToPlot(){
      return (double) getWaitingListLength();
    }

}
```

```
// SequentialBatchSpecificationSet.java

import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;


/**
 * The container generated by an order and
containing the spefications about
 * a sequential batch process
 *
 * @author Pietro Terna
 */
public class SequentialBatchSpecificationSet
extends SwarmObjectImpl{

    /** number (i.e. that of the order which has
generated this object */
    public int myOrderNumber;
```

```
    /** the position of this sequence in the recipe */
    public int pos;
    /** the specifications */
    public int productionTimeInTicks;
    public int quantityInSequentialBatch;

  /**
   * constructor for SequentialBatchSpecificationSet
   */
  public SequentialBatchSpecificationSet (Zone aZone,
int n, int p) {

     // Call the constructor for the parent class
     super(aZone);

     myOrderNumber=n;
     pos=p;

      if(StartVEFrame.verbose)
          System.out.println("Sequential b. spec.
created " +
                         "(by order " + myOrderNumber +
" at pos. " +
                         pos + "; NB first pos. is
0)");
  }

    /** setting the sequential batch specifications */
    public void setSpecifications(int k1, int k2)
    {
      productionTimeInTicks = k1;
      quantityInSequentialBatch = k2;

      if(StartVEFrame.verbose)
          {
            System.out.println(
                "Sequential batch production time
in ticks " +
              productionTimeInTicks);
             System.out.println(
                "Quantity in sequential batch
production " +
              quantityInSequentialBatch + "
orderNumber "+myOrderNumber);
           }
    }

    /** ???????? decrease production time */
    public int decreaseProductionTime(){

      if(StartVEFrame.verbose)
          System.out.println(
             "Residual production time in stand
alone production set from"+
           " order " + myOrderNumber+" at pos " +
pos +": " +
            productionTimeInTicks);
      productionTimeInTicks--;
      return productionTimeInTicks;
    }


    /**
     * returning the number of the order of this
procurement specification
     */
    public int getNumber () {
      return myOrderNumber;
   }

    /**
     * returning the position of this procurement
specification into its recipe
```

```
     */
    public int getPositionInRecipe() {
      return pos;
   }

    /** returning production time in ticks (original or
residual) */
    public int getProductionTimeInTicks(){
      return productionTimeInTicks;
    }

    /** settinging production time in ticks (residual)
*/
    public void setProductionTimeInTicks(int t){
      productionTimeInTicks=t;
    }

    /** returning quantity in sequential batch */
    public int getQuantityInSequentialBatch(){
      return quantityInSequentialBatch;
    }

}
```

// **Or.java**

```
import swarm.Globals;
import swarm.collections.ListImpl;

/**
 * An Or manager for our orders<br><br>
 *
 * orCriterion<br>
 *                0 - all 'or' branches in sequence<br>
 *                1 - choosing first branch<br>
 *                2 - choosing second branch<br>
 *                3 - a random branch<br>
 *                4 - the branch whose first step
 *                    has the shortest waiting
list<br><br>
 *
 * @author Pietro Terna
 */
public class Or
{
    // the class is used in a static way, so we
have no construtors
    // here and we have to declare static all the
methods

    public static int orCriterion;
    /** the list of the operating units */
    public static ListImpl unitList;

    public static void setOrCriterion(int k){
      orCriterion=k;
    }

    public static void setUnitList(ListImpl ul){
      unitList=ul;
    }

    public static void applyOr(Order o){
      OrSpecificationSet spec;
      Unit aUnit;
      boolean unitNotFound;
      int i, iii, node, chosenNode, length,
length0;
      int [] recipe, state;

      // the or specification set to be used
```

```
      spec=(OrSpecificationSet)
o.getPendingOrSpecificationSet();
      if(spec == null){
          System.out.println("Internal error: found an
order x in " +
                         "Or.applyOr(x) without 'or'
specifications");
          System.exit(1);
      }

      // applying the 'or' criterion
      if(orCriterion == 0)return;

      chosenNode=0;
      recipe = (int []) o.getRecipeVector();
      state =  (int []) o.getStateVector();

      if(orCriterion == 1)chosenNode=1;
      if(orCriterion == 2)chosenNode=2;
      if(orCriterion ==
3)chosenNode=Globals.env.uniformIntRand.

getIntegerWithMin$withMax(1,spec.getNodeNumber());

      if(orCriterion == 4)
          {
            length=1000000000;
            // looking for the first unit in each
branch
            for (node=1;
node<=spec.getNodeNumber();node++)
                {
                  length0=length;
                  unitNotFound=true;
                  for (i=0;i<unitList.getCount() &&
unitNotFound;i++)
                      {
                          aUnit=(Unit)
unitList.atOffset(i);
                          for
(iii=0;iii<aUnit.getNOfPhasesToDealWith()
                                  &&
unitNotFound;iii++)
                              {

      if(recipe[spec.getNodePosition(node)]==

aUnit.getProductionPhase(iii))
                                  {

      unitNotFound=false;
                                    length=aUnit.

getWaitingListLength();

      if(length<length0)chosenNode=

node;
                                  }
                              }
                      }
                }
          }

      if(StartVEFrame.verbose)
          System.out.println("Applying Or to
order number "+
                          o.getOrderNumber()+
                          " at pos.
"+spec.getPositionInRecipe());

      for (node=1;
node<=spec.getNodeNumber();node++)
```

```
          {
            for(i=spec.getNodePosition(node);
                i<spec.getNodePosition(node+1);i++)
                {
                  if(chosenNode != node) state[i]=-1;
                }
          }

    }

}
```

// **OrSpecificationSet.java**

```
import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;


/**
 * The container generated by an order and containing
the spefications about
 * an or stepe in a process
 *
 * @author Pietro Terna
 */
public class OrSpecificationSet extends
SwarmObjectImpl{

    /** number (i.e. that of the order which has
generated this object */
    public int myOrderNumber;
    /** the position of this sequence in the
recipe */
    public int pos;
    /** the specification */
    public int nodeNumber;
    /** the vector of node positions in 'or'
sequence (max 100) */
    public int[] nodePosition;


  /**
   * constructor for OrSpecificationSet
   */
  public OrSpecificationSet (Zone aZone, int n,
int p) {

     // Call the constructor for the parent class
     super(aZone);

     nodePosition = new int[100];
     myOrderNumber=n;
     pos=p;

     if(StartVEFrame.verbose)
       System.out.println("Or spec. created " +
                          "(by order " +
myOrderNumber + " at pos. " +
                           pos + "; NB first pos. is
0)");
  }

    /** setting the number if nodes */
    public void setNodeNumber(int k)
    {
      int i;
      nodeNumber = k;
```

```
      if(StartVEFrame.verbose)
          {
            System.out.print("In orSpecificationSet of
order # "+
                           myOrderNumber+
                           ", number of # "+nodeNumber+
                           "; node posistions:");
            for (i=1;i<=nodeNumber+1;i++)
                System.out.print("
"+getNodePosition(i));
            System.out.println(" ");
          }

    }

    /** setting the node position */
    public void setNodePosition(int node, int
position){

      if(node>100){
          System.out.println("More than 100 nodes in an
'or' sequence in "+
                           "orden # "+myOrderNumber);
          System.exit(1);
      }
      nodePosition[node-1]=position;
    }

    /** getting the node position; position of node # 1
is in
     *  nodePosition[0] etc.
     */
    public int getNodePosition(int node){
      return nodePosition[node-1];
    }
```

```
    /**
     * returning the number of the order of this
procurement specification
     */
    public int getNumber () {
      return myOrderNumber;
   }

    /**
     * returning the position of this procurement
specification into its recipe
     */
    public int getPositionInRecipe() {
      return pos;
   }

    /**
     * returning the number of nodes (branches)
in this 'or' sequence
     */
    public int getNodeNumber() {
      return nodeNumber;
   }

}
```

```
// EndUnit.java

import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;

import swarm.collections.ListImpl;

/**
```

```
 * The EndUnit class instances represent virtual or
actuale
 * places or warehouse were temporary finished sub-
recipes (produced internally
 * or obtained as procurements) wait to be used in
other recipes via a "p"
 * (procurement) step<br><br>
 *
 * an endUnit has its internal number equal to the
concluding code
 * used to finish/identify a temporary finished sub-
recipe
 *
 * @author Pietro Terna
 */
public class EndUnit extends SwarmObjectImpl implements
PTHistogramPlottable
{

    /** the number identifying the unit which is also
its end code */
    public int unitNumber;
    /** the list containing order temporary kept in
this virtual (or actual)
     *  place */
    public ListImpl  temporaryList;
    /** the modelSwarm (to which we can apply the
     * setProductionTimeAndRecipeLength(...) method, if
any */
    VEFrameModelSwarm vEFrameModelSwarm;


    /**
     * the constructor for EndUnit
     */
    public EndUnit (Zone aZone, VEFrameModelSwarm mo)
    {
      // Call the constructor for the Unit parent
class.
      super(aZone);

      // the model (for future use)
      vEFrameModelSwarm = mo;

      // creating the internal list temporary the
items temporary in hold
      // hold
      temporaryList = new ListImpl (getZone());
    }

    /** set unit number */
    public void setUnitNumber(int un)
    {
      // the unit id number.
      unitNumber=un;
      // announce the unit to the console
      if(StartVEFrame.verbose)
      System.out.println("End unit number " +
unitNumber +
                         " has been created.");
     }

    /** get unit number */
    public int getUnitNumber()
    {
      // the unit id number.
      return unitNumber;
    }


    /**
     * Putting an order in the temporaryList of
the unit
     */
```

```
    public void setTemporaryList(Order anOrder)
    {
        temporaryList.addLast(anOrder);
      if(StartVEFrame.verbose)
          System.out.println("The end unit " +
unitNumber +
                         " has received the order # "
                         + anOrder.getOrderNumber());
    }

    /**
     * Removing an order from the temporaryList of the
unit
     * and returning its address of the order or of its
clone if
     * the original order has multiplicity > 1
     */
    public Object removeFromTemporaryList()
    {
      Order removedOrder, orderClone;

      removedOrder = (Order)
temporaryList.removeFirst();
      if(StartVEFrame.verbose)
          System.out.println("The end unit " +
unitNumber +
                         " has removed the order # "
                         +
removedOrder.getOrderNumber());

      if(removedOrder.getMultiplicity() == 1) return
removedOrder;

      else
          {

            if(StartVEFrame.verbose)
                System.out.println("A clone of
the order # " +
                 removedOrder.getOrderNumber() +
                 " has been created; the original
order has now " +
                 "multiplicity " +
(removedOrder.getMultiplicity()-1));
            orderClone = (Order)
removedOrder.clone();

      removedOrder.setMultiplicity(removedOrder.g
etMultiplicity()-1);
            temporaryList.addFirst(removedOrder);
            return orderClone;
          }
    }

    /**
     * return the temporary list length
     */
    public int getTemporaryListLength()
    {
      return temporaryList.getCount();
    }

    /**
     * return the count of the order in the list,
with multiplicity
     */
    public int getTemporaryListCount()
    {
      int i, c;

      c = 0;
      for (i=0; i< getTemporaryListLength();i++)
          c += ((Order)
temporaryList.atOffset(i)).getMultiplicity();
```

```
      return c;
    }


    /**
     * PTHistogramPlottable interface method:
getLabel()
     */
    public String getLabel(){
      return "" + getUnitNumber();
    }

    /**
     * PTHistogramPlottable interface method:
getValueToPlot()
     */
    public double getValueToPlot(){
      return (double) getTemporaryListCount();
    }

}
```

Appendice B: Analisi del corretto funzionamento della versione jVEFrame-0.7.0.a

Per verificare l'esattezza della versione *jVEFrame-0.7.0.a* riportiamo l'output dell'esecuzione passo per passo effettuata con riferimento alla programmazione dei materiali e delle componenti basando l'analisi su un numero di turni ridotti, come riportato in tabella 1 esemplificativa del file *orderSequences.xls*.

1	100	*	2	200	*	3							;
2	100	*	2	200	*	3	300	*	1	400	*	5	;
3							300	*	3	400	*	1	;

Tabella 1: Programmazione dei materiali e delle componenti, esempio su tre turni.

Le sequenze produttive, con i relativi tempi di esecuzione, sono estratti dalla tabella 2, anche questa esemplificativa del file *recipes.xls*.

#	comment	;																		
#	comment	;																		
	alfa	100	1	s	10	2	s	2	3	s	4	4	s	3	5	s	1	e	100	;	
	beta	200	6	s	5	2	s	1	7	s	1	8	m	1	4	s	2	e	200	;	
	teta	300	p	1	100	9	s	10												;
#	comment	;																		
	sigma	400	p	1	200	10	s	1												;

Tabella 2: Sequenze di fasi di produzione, esempio di quattro ricette.

per poter impostare gli oggetti rappresentativi delle singole unità di produzione la struttura della simulazione prevede, dopo aver creato le sonde necessarie per impostare i parametri, la lettura di dati dai file riportati nelle tabelle che seguono.

unit_#
100

200

Tabella 3: Esempio del file endUnit.txt.m

unit_#__useWarehouse____prod.phase_#____fixed_costs_____variable_costs								
1	1		1		1		1	
2	1		2		1		1	
3	1		3		1		1	
4	1		0		0		0	
5	1		5		1		1	
6	1		6		1		1	
7	1		7		1		1	
8	1		8		1		1	
9	1		0		0		0	
10	1		0		0		0	

Tabella 4: Esempio del file unitBasicData.txt.

3				
	4	1	1	1
	1	1	1	1
	8	1	1	1

Tabella 5: Esempio del file units.xls, unità complessa n°4.

2				
	9	1	1	1
	8	1	1	1

Tabella 6: Esempio del file units.xls, unità complessa n°9.

4				
	10	1	1	1
	7	1	1	1
	4	1	1	1

	8	1	1	1

Tabella 7: Esempio del file units.xls, unità complessa n°10.

La simulazione inizia con la creazione degli oggetti necessari al funzionamento all'interno del metodo buildObject() del VEFrameModelSwarm.java:

- Sono create le unità finali;

End unit number 100 has been created.

End unit number 200 has been created.

- Sono create le unità semplici;

Unit number 1 has been created.

Unit number 2 has been created.

Unit number 3 has been created.

Unit number 4 has been created.

- Come si può osservare nella tabella n° 4 la quarta unità, così come la nona e la decima, è una unità complessa…

Unit 4 is a special one.

…in grado di effettuare tre fasi di lavorazione differenti…

special unit 4 nOfPhasesToDealWith 3

…ognuna delle quali con propri costi fissi e variabili.

special unit 4 fixed costs 1.0 variable costs 1.0

special unit 4 fixed costs 1.0 variable costs 1.0

special unit 4 fixed costs 1.0 variable costs 1.0

- Tutte le unità sono state create

Unit number 5 has been created…

… … … … …

…Unit 10 is a special one.

- Come spiegato in precedenza (vedi capitolo 5) non sono stati utilizzati i magazzini e le informazioni;

No warehouses generated

- Si effettua il richiamo al metodo, presente nella classe OrderDistiller.java, necessario per definire il dizionario delle possibili fasi di lavorazione. Avviene la determinazione della lunghezza e l'archiviazione dei nomi delle fasi.

Length of the dictionary (# of codes identifying steps in recipes): 18

0 100

1 200

2 1

3 2

4 3

5 4

6 1

7 8

8 5

9 6

10 7

11 8

12 9

13 8

14 10

15 7

16 4

17 8

- Lo stesso richiamo al metodo setDictionary() avvia anche la lettura del file recipes (vedi tabella 1). Se la riga contiene un commento questo è segnalato…

The row # 1 contains a comment

The row # 2 contains a comment

…altrimenti se la riga contiene una ricetta si determina la lunghezza della stessa ed è creato un esemplare della classe Recipe.java.

The row # 3 contains a recipe named alfa

OrderDistiller: aRecipe with code 100was born

The row # 4 contains a recipe named beta

OrderDistiller: aRecipe with code 200was born

The row # 5 contains a recipe named teta

OrderDistiller: aRecipe with code 300was born

The row # 6 contains a comment

The row # 7 contains a recipe named sigma

OrderDistiller: aRecipe with code 400was born

The row # 1 contains a comment

The row # 2 contains a comment

The row # 3 contains a comment

- Una volta che lo schedule è avviato sono richiamate tutte le funzioni della simulazione. Anzitutto al primo *tick* del turno vengono azzerati i costi della contabilità e poi è eseguita la produzione assegnando ad ogni unità il primo passo della ricetta che è in grado di fare;

Step 1 in unit 1; waitingList count 0
Step 1 in unit 2; waitingList count 0
Step 1 in unit 3; waitingList count 0
Step 1 in unit 4; waitingList count 0
Step 1 in unit 5; waitingList count 0
Step 1 in unit 6; waitingList count 0
Step 1 in unit 7; waitingList count 0
Step 1 in unit 8; waitingList count 0
Step 1 in unit 9; waitingList count 0
Step 1 in unit 10; waitingList count 0

- Dopo che il primo passo della ricetta di produzione è stato assegnato, gli ordini sono assegnati ad una lista di produzione giornaliera;

Step 2 in unit 1 dailyProductionList count 0
Step 2 in unit 2 dailyProductionList count 0
Step 2 in unit 3 dailyProductionList count 0
Step 2 in unit 4 dailyProductionList count 0
Step 2 in unit 5 dailyProductionList count 0
Step 2 in unit 6 dailyProductionList count 0
Step 2 in unit 7 dailyProductionList count 0
Step 2 in unit 8 dailyProductionList count 0

Step 2 in unit 9 dailyProductionList count 0

Step 2 in unit 10 dailyProductionList count 0

Lo schedule richiama il metodo distill() della classe OrderDistiller.java, il quale avvia la lettura del file orderSequences (visto in tabella 1) e si avvia così il turno relativo…

The shift #1has begun

…di conseguenza tutti gli ordini di produzione che devono essere evasi nel turno vengono instanziati e si comunica la loro presenza alla console;

New order (#1)

- Il programma Order,java crea due vettori che definiscono: lo stato di completamento delle fasi di lavorazione…

orderState vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

…il contenuto del vettore delle fasi di lavorazione;

orderRecipe vector 1 1 1 1 1 1 1 1 1 1 2 2 3 3 3 3 4 4 4 5 100

- Gli ordini sono poi propagati verso le unità che rispondono se sono in grado di eseguire una determinata lavorazione;

The unit 1 has received the order # 1

New order (#2)

orderState vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

orderRecipe vector 1 1 1 1 1 1 1 1 1 1 2 2 3 3 3 3 4 4 4 5 100

The unit 1 has received the order # 2

New order (#3)

```
orderState  vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
orderRecipe vector 6 6 6 6 6 2 7 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 4 4 200
The unit 6 has received the order # 3
New order (#4)
orderState  vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
orderRecipe vector 6 6 6 6 6 2 7 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 4 4 200
The unit 6 has received the order # 4
New order (#5)
orderState  vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
orderRecipe vector 6 6 6 6 6 2 7 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 4 4 200
The unit 6 has received the order # 5
```

- Dal secondo *tick* della giornata sino all'ultimo la produzione delle singole fasi di lavorazione è attuata ripetendo i due cicli già descritti:

il primo di assegnazione dei passi alle unità con eventuale creazione delle code…

Step 1 in unit 1; waitingList count 2
Step 1 in unit 2; waitingList count 0
Step 1 in unit 3; waitingList count 0
Step 1 in unit 4; waitingList count 0
Step 1 in unit 5; waitingList count 0
Step 1 in unit 6; waitingList count 3
Step 1 in unit 7; waitingList count 0
Step 1 in unit 8; waitingList count 0
Step 1 in unit 9; waitingList count 0
Step 1 in unit 10; waitingList count 0

…il secondo di assegnazione alle unità degli ordini;

Step 2 in unit 1 dailyProductionList count 1
The unit 1 has received the order # 1
Step 2 in unit 2 dailyProductionList count 0
Step 2 in unit 3 dailyProductionList count 0
Step 2 in unit 4 dailyProductionList count 0
Step 2 in unit 5 dailyProductionList count 0
Step 2 in unit 6 dailyProductionList count 1
The unit 6 has received the order # 3
Step 2 in unit 7 dailyProductionList count 0
Step 2 in unit 8 dailyProductionList count 0
Step 2 in unit 9 dailyProductionList count 0
Step 2 in unit 10 dailyProductionList count 0

- Terzo tick…

Step 1 in unit 1; waitingList count 2…

…Step 1 in unit 10; waitingList count 0

Step 2 in unit 1 dailyProductionList count 1

The unit 1 has received the order # 2

Step 2 in unit 2 dailyProductionList count 0

Step 2 in unit 3 dailyProductionList count 0

Step 2 in unit 4 dailyProductionList count 0

Step 2 in unit 5 dailyProductionList count 0

Step 2 in unit 6 dailyProductionList count 1

The unit 6 has received the order # 4

Step 2 in unit 7 dailyProductionList count 0

Step 2 in unit 8 dailyProductionList count 0

Step 2 in unit 9 dailyProductionList count 0

Step 2 in unit 10 dailyProductionList count 0

- Al sesto *tick* l'ordine numero tre passa alla unità numero due per effettuare la lavorazione numero due. Stessa cosa succede poi agli altri ordini;

Step 1 in unit 1; waitingList count 2…

…Step 1 in unit 10; waitingList count 0

Step 2 in unit 1 dailyProductionList count 1

The unit 1 has received the order # 1

Step 2 in unit 2 dailyProductionList count 0

Step 2 in unit 3 dailyProductionList count 0

Step 2 in unit 4 dailyProductionList count 0

Step 2 in unit 5 dailyProductionList count 0

Step 2 in unit 6 dailyProductionList count 1

The unit 2 has received the order # 3

Step 2 in unit 7 dailyProductionList count 0

Step 2 in unit 8 dailyProductionList count 0

Step 2 in unit 9 dailyProductionList count 0

Step 2 in unit 10 dailyProductionList count 0

- All'ottavo *tick* l'ordine numero tre passa alla unità numero quattro per effettuare la lavorazione numero otto. L'unità numero quattro è complessa ed è perciò in grado di effettuare più tipi di lavorazioni. La scelta tra l'unità quattro e la otto avviene considerando la prima delle due che risponde. Stessa cosa succede poi agli altri ordini;

Step 1 in unit 1; waitingList count 2…

…Step 1 in unit 10; waitingList count 0

Step 2 in unit 1 dailyProductionList count 1

The unit 1 has received the order # 1

Step 2 in unit 2 dailyProductionList count 1

The unit 7 has received the order # 4

Step 2 in unit 3 dailyProductionList count 0

Step 2 in unit 4 dailyProductionList count 0

Step 2 in unit 5 dailyProductionList count 0

Step 2 in unit 6 dailyProductionList count 1

The unit 2 has received the order # 5

```
Step 2 in unit 7 dailyProductionList count 1
The unit 4 has received the order #  3
Step 2 in unit 8 dailyProductionList count 0
Step 2 in unit 9 dailyProductionList count 0
Step 2 in unit 10 dailyProductionList count 0
```

- All’inizio del novantunesimo *tick* parte il secondo turno;

```
The shift #2has begun
```

- Nuovi ordini vengono creati esattamente come visto in precedenza;

```
New order (#6)
orderState  vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
orderRecipe vector 1  1  1  1  1  1  1  1  1  1  2 2  3  3  3  3  4  4  4  5  100
The unit 1 has received the order #  6
New order (#7)
orderState  vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
orderRecipe vector 1  1  1  1  1  1  1  1  1  1  2 2  3  3  3  3  4  4  4  5  100
The unit 1 has received the order #  7
New order (#8)
orderState  vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
orderRecipe vector 6  6  6  6  6  2  7  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 4 4 200
The unit 6 has received the order #  8
```

```
New order (#9)
orderState  vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
orderRecipe vector 6 6 6 6 6 2 7 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 4 4 200
The unit 6 has received the order #  9
New order (#10)
orderState  vector 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
orderRecipe vector 6 6 6 6 6 2 7 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
8 4 4 200
The unit 6 has received the order #  10
```

- Poiché vi sono degli ordini che necessitano di procurement, si registrano le ProcurementSpecifications, in questo primo caso si rile va che all'undicesimo ordine in posizione zero è presente un procurement, si necessita del semilavorato con codice 100;

```
Proc. spec. created (by order 11 at pos. 0; NB first pos. is 0)
item(s) to be procured 100
New order (#11)
```

- Il primo passo della ricetta che necessita di un procurement viene indicato con valore negativo;

orderState vector 0 0 0 0 0 0 0 0 0 0

orderRecipe vector -9 9 9 9 9 9 9 9 9 9

Proc. spec. created (by order 12 at pos. 0; NB first pos. is 0)

item(s) to be procured 200

New order (#12)

orderState vector 0

orderRecipe vector -10

Proc. spec. created (by order 13 at pos. 0; NB first pos. is 0)

item(s) to be procured 200

New order (#13)

orderState vector 0

orderRecipe vector -10

Proc. spec. created (by order 14 at pos. 0; NB first pos. is 0)

item(s) to be procured 200

New order (#14)

orderState vector 0

orderRecipe vector -10

Proc. spec. created (by order 15 at pos. 0; NB first pos. is 0)

item(s) to be procured 200

New order (#15)

orderState vector 0

orderRecipe vector -10

Proc. spec. created (by order 16 at pos. 0; NB first pos. is 0)

item(s) to be procured 200

New order (#16)

orderState vector 0

orderRecipe vector -10

- La produzione è quindi assegnata come già visto in precedenza;

Step 1 in unit 1; waitingList count 2…
…Step 1 in unit 10; waitingList count 0

Step 2 in unit 1 dailyProductionList count 1…
…Step 2 in unit 10 dailyProductionList count 0

- *In questa fase si crea una tabella delle quantità presenti nelle end units. In questo caso la end unit 100 contiene due elementi mentre la end unit 200 è vuota. Il passo di produzione identificato con il 9 ed eseguito dall'unità 9 richiede il procurement 100.*

eu 100 2 eu 200 0 as seen in procurementAssembler of unit 9
proc. spec. of order 11 at pos. 0 items 100
eu 100 1 eu 200 0 as residual in procurementAssembler of unit 9
completed
The end unit 100 has removed the order # 1
The unit 9 has received the order # 11

eu 100 1 eu 200 0 as seen in procurementAssembler of unit 10
proc. spec. of order 12 at pos. 0 items 200
eu 100 1 eu 200 -1 as residual in procurementAssembler of unit 10

eu 100 1 eu 200 0 as seen in procurementAssembler of unit 10
proc. spec. of order 13 at pos. 0 items 200
eu 100 1 eu 200 -1 as residual in procurementAssembler of unit 10

eu 100 1 eu 200 0 as seen in procurementAssembler of unit 10
proc. spec. of order 14 at pos. 0 items 200
eu 100 1 eu 200 -1 as residual in procurementAssembler of unit 10

eu 100 1 eu 200 0 as seen in procurementAssembler of unit 10
proc. spec. of order 15 at pos. 0 items 200
eu 100 1 eu 200 -1 as residual in procurementAssembler of unit 10

eu 100 1 eu 200 0 as seen in procurementAssembler of unit 10
proc. spec. of order 16 at pos. 0 items 200
eu 100 1 eu 200 -1 as residual in procurementAssembler of unit 10

Step 1 in unit 1; waitingList count 2…
…Step 1 in unit 10; waitingList count 0

Step 2 in unit 1 dailyProductionList count 1…
…Step 2 in unit 10 dailyProductionList count 0

The shift #3has begun
The simulation finishes here. It is not the right and normal way, but is only for explanation. Thank you for your attention and interest.