

UNIVERSITÀ DEGLI STUDI DI TORINO

Facoltà di Economia
Corso di Laurea in Economia e Commercio

TESI DI LAUREA:

Analisi dei processi aziendali mediante un
modello ad agenti in Java Swarm, con scorte

Relatore:
Prof. Pietro Terna

Correlatore:
Prof. Sergio Margarita

Candidato:
Marco Remondino

ANNO ACCADEMICO 1999 - 2000

Indice degli Argomenti

Introduzione	1
Capitolo 1 – Azienda Virtuale: trasformazione e modelli	
1.1 Un sistema sociale aperto	10
1.2 Information Technology in azienda	12
1.3 La trasformazione dell'azienda	12
1.4 Cambiamenti nell'interazione con i fornitori	13
1.5 Cambiamenti nella relazione con i clienti	14
1.6 Information Technology e produttività	15
1.7 Stime a livello economico aggregato	17
1.8 Il concetto di azienda virtuale ed il modello MAIS	17
1.9 L'architettura NIIP	22
1.10 Un esempio pratico: BizTalk	25
Capitolo 2 – Evoluzione del sistema informativo in azienda	
2.1 Sistema informativo aziendale	26
2.2 Il processo di produzione delle informazioni	28
2.3 Computer e network	30
2.4 Commercio via internet	32
Capitolo 3 – La Supply Chain	
3.1 Introduzione	34
3.2 La Supply Chain: dal passato al futuro	35
3.3 Gestione ed organizzazione di una catena di fornitura	36
3.4 E-business e nuove tecnologie	38
3.5 Il Business to Business come aspetto cruciale	40
3.6 Sistemi multi-agente per lo studio di una Supply Chain	41
3.7 Struttura di una Supply Chain tradizionale	42
3.8 La gestione del flusso informativo	45
3.9 Il just in time ed i sistemi EDI	47

Capitolo 4 – Logistica e gestione delle scorte

4.1 Evoluzione della logistica aziendale.....	50
4.2 Le scorte nella realtà aziendale	52
4.3 Gestione delle scorte: funzioni ed obiettivi.....	53
4.4 Sistemi di controllo delle scorte: una classificazione.....	55
4.5 Modelli di tipo deterministico: lotto economico	57
4.6 Magazzini nella realtà aziendale	59
4.7 Magazzini e teoria delle code	60
4.8 Legge di Little ampliata.....	61
4.9 Attività dei magazzini e costi corrispondenti	62

Capitolo 5 – Complessità e simulazione sociale

5.1 Introduzione ai modelli.....	65
5.2 Simulazione sociale al calcolatore	67
5.3 La teoria della complessità	68
5.4 Sistemi complessi di tipo adattivo	72
5.5 Analisi dei sistemi adattivi complessi (CAS)	74
5.6 Agent-based computational economics	77
5.7 Identificazione ed acquisizione della conoscenza	80
5.8 Come un'organizzazione crea conoscenza.....	82
5.9 Formalizzazione e trasmissione della conoscenza	83

Capitolo 6 – Swarm, Java e simulazione

6.1 La simulazione al calcolatore.....	87
6.2 Il progetto Swarm	88
6.3 Object Oriented Programming	90
6.4 Gli errori di coding ed il debugging	91
6.5 La gestione del tempo nelle simulazioni	92
6.6 Costruzione di un modello in Swarm	93
6.7 Le librerie di Swarm.....	95
6.8 Linguaggio Java in Swarm.....	96
6.9 Lo schema E.R.A.	98

Capitolo 7 – Un modello di Virtual Enterprise: JVE

7.1 Introduzione al modello	100
-----------------------------------	-----

7.2 Le unità produttive	101
7.3 L'unità "Front End"	101
7.4 Tempi di produzione	103
7.5 I magazzini	103
7.6 Gli ordini ed il mercato	104
7.7 La computazione dei costi	105
7.8 Il modello in Swarm	106
7.9 La formazione di code	109
7.10 Magazzini nel modello VE.....	110
7.11 Creazione tecnica dei magazzini	111
7.12 Le regole per il funzionamento	112
7.13 La rappresentazione del carico dei magazzini	113
7.14 Realizzazione tecnica	115
 Capitolo 8 – Conclusioni ed espansione del modello	
8.1 Il modello base	118
8.2 Introduzione dei magazzini	120
8.3 Espansione dei magazzini	121
8.4 Introduzione dei tempi di produzione	123
8.5 Introduzione della contabilità dei costi	125
 Bibliografia	128
 Appendice A – Elementi di programmazione in Java.....	I
Appendice B – Programmazione ad oggetti in Java	XX
Appendice C – Il codice di JVE con scorte	XXIX

INTRODUZIONE

Il presente lavoro è volto a studiare la fattibilità di un modello di impresa virtuale, ottenuto scomponendo l'attività dell'azienda in processi elementari, attraverso l'utilizzo di Java Swarm. Lo scopo finale è quello di poter studiare l'azienda da un punto di vista economico organizzativo, considerando l'impresa come un fenomeno sociale. Non si osservano dunque solamente i processi aziendali che danno origine alla *supply chain* tradizionale, ma viene anche studiata la complessità all'interno dei modelli sociali.

Nel primo capitolo, viene brevemente esaminata la struttura dell'azienda tradizionale, intesa quale sistema sociale aperto. Per studiare l'azienda, quale produttrice di beni e servizi, occorre ricordare che essa fa parte di un ambiente: l'azienda può infatti essere vista come un sub-sistema, all'interno di un sistema più ampio. L'ambiente in cui opera l'impresa non è solamente circoscritto al settore merceologico in cui opera strettamente, ma arriva ad interessare anche aspetti di osservazione più ampi, quali per esempio l'economia di un paese, risvolti legali, politici e altro.

Viene poi descritto il processo di transizione delle aziende, avvenuto a seguito dell'introduzione nell'organizzazione della struttura informatica. La diffusione di strumenti informatici potenti e dal costo relativamente ridotto, ha profondamente mutato le metodologie di studio inerenti a questo sistema così complesso e mutevole. L'utilizzo del computer e mezzi informatici in genere all'interno di un'organizzazione economica non si limita più solamente al puro calcolo e computazione, ma si estende ormai sempre più verso elevati livelli strategici dell'azienda, come valido supporto decisionale. Tuttavia, il passaggio all'*Information Technology*, non accompagnato da cambiamenti complementari non porterebbe benefici e potrebbe addirittura causare perdite. Questo avviene perché le difficili interazioni tra le pesanti pratiche esistenti all'interno dell'organizzazione e i nuovi sistemi introdotti compenserebbero e spesso supererebbero i benefici ottenuti. E' dunque necessario adeguare l'intera organizzazione e struttura aziendale, in modo tale da sfruttare al meglio le nuove opportunità, rese possibili dagli strumenti tecnologici.

A questo proposito viene esaminato il concetto di azienda virtuale, nella sua duplice accezione: da una parte, riferito all'azienda vera e propria, vista come componente di un organismo sovraziendale virtuale che attraverso la collaborazione di tutte le unità della

catena di produzione svolge quello che svolgerebbe un'azienda totalmente integrata verticalmente, cioè capace di svolgere tutte le fasi del ciclo di produzione al suo interno.

Il concetto di azienda virtuale ha anche significato di modello d'azienda, creato attraverso una rappresentazione informatica, con lo scopo di consentirne la simulazione.

Per quanto riguarda il primo caso, ho esaminato il modello teorico NIIP ed un tentativo di applicazione pratica, BizTalk, mentre nel secondo caso mi sono occupato dell'architettura MAIS (Multi-Agent Information System), un'interessante applicazione della metodologia di simulazione basata su agenti, nell'ambito della gestione delle catene di fornitura

Nel secondo capitolo, a supporto del precedente, si effettua un breve excursus su quella che è stata l'evoluzione del sistema informativo aziendale, dall'introduzione di pochi computer con scopi puramente computazionali, fino agli attuali sviluppi, resi possibili dall'Internet e dai concetti di E-commerce ed E-business. Sin dagli anni '50, infatti, il sistema informativo ha occupato una posizione di rilievo all'interno delle aziende; inizialmente si trattava semplicemente di un rimpiazzo elettronico per i tabulati e gli altri metodi meccanici di contabilità di base, ma ben presto diventò più sofisticato e complesso, tanto da coprire tutti gli aspetti economici dell'impresa.

L'informazione è la base fondamentale per una consapevole amministrazione d'azienda; attengono al sistema informativo tutte le attività di raccolta, elaborazione e trasmissione dei dati realizzate al fine di soddisfare le esigenze conoscitive interne ed esterne.

L'obiettivo fondamentale del sistema informativo aziendale è quello di fornire, a chi opera nell'azienda, le informazioni necessarie per lo svolgimento delle mansioni, attraverso un processo continuo di trattamento di dati rappresentativi della realtà aziendale esterna.

I dati costituiscono la materia prima del processo di produzione delle informazioni, in quanto rappresentazione originaria e non decifrata di un fenomeno, attuata attraverso simboli (lettere, numeri, caratteri speciali). Essi acquisiscono un significato sul piano economico in seguito ad una appropriata classificazione, organizzazione ed al loro effettivo impiego nei processi decisionali e di controllo.

Fino a pochi anni or sono, le unità di business comunicavano tra loro principalmente attraverso l'uso del fax, del telefono e del contatto umano diretto. Alcuni dei sistemi ideati nel passato, per permettere lo scambio di informazioni per via elettronica, attraverso reti di computer, possono essere considerati come una base per l'espansione dell'Internet durante la seconda metà degli anni '90.

Le innovazioni informatiche hanno avuto una ricaduta notevole anche sul commercio, sia verso i clienti finali (*Business to Consumer*) sia attraverso altre imprese (*Business to Business*). Il primo tipo di commercio, svolto utilizzando la rete, è definito *e-commerce*, mentre il secondo è noto come *e-business*.

Poiché lo scopo primario del presente lavoro è giungere ad un modello di azienda da simulare al calcolatore, nel terzo capitolo viene esaminata la struttura di una *Supply Chain*, cioè di una catena di fornitura di tipo tradizionale. La *Supply Chain* parte da materiali grezzi, e passa attraverso la produzione, distribuzione, trasporto, magazzino per giungere alla vendita dei prodotti finiti. Esistono aziende in cui la *Supply Chain* è interamente presente, ma più spesso essa è suddivisa tra diverse unità aziendali, in modo tale da ottenere specializzazione maggiore e minori costi.

La gestione ottimale di una *Supply Chain* è sicuramente un punto cruciale per l'organizzazione aziendale. Non è infatti sufficiente ottimizzare l'efficienza operativa delle varie unità, in quanto diviene critico lo scambio di informazioni, non solo all'interno di una singola azienda, ma anche tra diverse realtà produttive o di distribuzione.

Recentemente, l'aumento della competizione, provocato dalla globalizzazione e dalle innovazioni tecnologiche, ha accresciuto la necessità di migliorare l'efficienza con cui le imprese producono e distribuiscono beni e servizi ai clienti. Questo ha portato le aziende di tutto il mondo a cercare nuovi modi di organizzare la forza lavoro, utilizzare le risorse disponibili, interfacciarsi verso i fornitori e organizzare la produzione, in modo tale da risparmiare sui costi e guadagnare competitività nei confronti di un mercato sempre più aggressivo e sempre più orientato verso concetti di *Customer Service*.

Gli sviluppi nel campo dei trasporti, *Information Technology* e telecomunicazioni hanno permesso un crescente livello di scambi sociali tra paesi diversi, attraverso il commercio. Poiché la comunicazione riduce le differenze sociali tra diversi paesi, il mondo è ora più omogeneo ed è dunque possibile condividere beni e servizi, stile di vita, abitudini e anche ideologie. In passato, la competitività delle imprese non era ad un livello uguale per tutte. La fase iniziale dell'ottimizzazione del business portò ad un aumento generalizzato della competitività, attraverso l'industrializzazione e l'automazione dei processi produttivi. In seguito, vennero introdotti concetti quali il *Total Quality Management* (TQM), e questo portò, per le imprese che lo impiegarono, nuovi vantaggi competitivi. Nella fase attuale, l'elevata produttività ed il livello qualitativo non sono più considerabili come vantaggi, ma devono essere visti come veri e propri aspetti necessari per un business.

I clienti sanno ormai esattamente quello che desiderano e spesso sono loro a fissarne il prezzo, e dunque le aziende sono costrette ad utilizzare la pianificazione attenta delle catene di fornitura, per anticipare le condizioni del mercato.

La gestione ottimale di una Supply Chain è sicuramente un punto cruciale per l'organizzazione aziendale. Non è infatti sufficiente ottimizzare l'efficienza operativa delle varie unità, in quanto diviene critico lo scambio di informazioni, non solo all'interno di una singola azienda, ma anche tra diverse realtà produttive o di distribuzione.

La chiave di successo nelle strategie di business degli anni '90 è stata non soltanto l'evoluzione delle tecnologie produttive e il miglioramento dei meccanismi distributivi, ma soprattutto la crescita del coordinamento tra aziende o tra funzioni aziendali. Quando le persone collaborano, devono in qualche modo comunicare, prendere decisioni, predisporre le risorse al momento giusto e nel posto giusto: i manager, gli addetti al settore commerciale, i clienti, gli intermediari, devono dunque coordinarsi.

In questa parte ho anche considerato i costi derivanti dai singoli processi, e dalle modificazioni che l'introduzione dell'*e-business* potrebbe potenzialmente portare.

Restando ancora in tema descrittivo di ciò che si vorrà simulare nel modello, nel quarto capitolo si passa a studiare l'aspetto su cui l'autore del presente lavoro si è maggiormente soffermato, cioè l'effetto e la funzione dei magazzini all'interno di un'impresa. Lo scopo è quello di giungere ad una formalizzazione del funzionamento dei magazzini nella realtà aziendale, per poterli inserire nel modello JVE. A questo fine viene presa in esame l'evoluzione della funzione logistica in azienda, nonché vari metodi statistici per ottimizzare la gestione delle scorte; a questo proposito, si accenna brevemente alla teoria delle code.

Si considerano poi ad esempi concreti, esaminando quali grandezze variano inserendo magazzini in una realtà aziendale, confrontando i risultati pratici con quelli ottenuti con i metodi statistici prima descritti, considerando anche le varie attività svolte da un magazzino ed i costi da esse derivanti.

La gestione delle scorte non è sicuramente un argomento recente: esso ha infatti una lunga tradizione aziendale di oltre settant'anni. Recentemente, con il fiorire di concetti quali il *Just in Time* e l'azzeramento dei livelli di magazzino, le teorie alla base della gestione delle scorte sono profondamente mutate ed evolute, integrandosi spesso con la realtà informatica degli anni '90.

Il sottosistema magazzino può assumere, in una azienda di piccole dimensioni o prevalentemente artigiana, una importanza marginale, mentre è maggiore nelle imprese di

carattere commerciale, nelle quali da una buona gestione della funzione dipende gran parte del successo dell'impresa.

Con il crescere delle dimensioni e l'articolarsi del processo produttivo i flussi di materiali che interessano la logistica si fanno più complessi: ogni flusso è caratterizzato da distinte problematiche e particolarmente complessa è la gestione della tempistica delle diverse operazioni.

Poiché lo scopo del lavoro è la costruzione di un modello per la simulazione di un'impresa, integrata in un contesto sociale, nel quinto capitolo si trovano cenni alla teoria della complessità, applicata ai modelli di simulazione sociale.

La simulazione sociale è un insieme di discipline, che si trovano al confine tra la scienza computazionale, tipica dei modelli matematici e le scienze sociali tradizionali. Con l'evoluzione delle tecnologie informatiche e la crescita esponenziale della potenza di elaborazione dei calcolatori elettronici, negli ultimi anni la ricerca nel campo della simulazione dei modelli sociali ha subito un impulso notevole. Simulazioni praticamente impossibili da computare con i mezzi disponibili negli anni '80, possono ora essere elaborate da un normale personal computer di basso costo e si aprono prospettive ancor più interessanti per gli anni a venire. In questo ambito ho esaminato la teoria dei sistemi adattivi di tipo complesso, ed in particolare i modelli fondati su agenti.

Molti processi di tipo sociale sono complessi, poiché non si possono scomporre in sotto-processi elementari. Gli aspetti economici, demografici, culturali, spaziali, possono solamente essere aggregati, per fornire una visione di insieme dei processi sociali, ma la loro analisi isolata non consente di spiegare e prevedere i fenomeni che si osservano dall'esterno.

Il termine "complesso", riferito allo studio dei modelli dinamici basati su agenti, non è sinonimo di "complicato", nell'accezione di "difficile". Per complessità si intende un fenomeno matematicamente definibile, una caratteristica qualitativa di un sistema, cioè di un aggregato organico e strutturato di parti tra loro interagenti, che gli fa assumere proprietà che non derivano dalla semplice giustapposizione delle parti stesse.

Al di là dei numerosi studi accademici, condotti in particolare negli USA in questo campo, negli ultimi dieci anni si sono sviluppate applicazioni pratiche nel campo del management e della gestione in generale di sistemi complessi, considerati alla stregua di ecosistemi non necessariamente di natura biologica, e basate sui principi della Scienza della Complessità.

Il punto centrale della Scienza della Complessità è nella definizione e descrizione del CAS (*Complex Adaptive System*). Il CAS è un Sistema Complesso Adattativo, cioè che ha la proprietà di adattarsi al suo ambiente per perseguire i propri scopi. Esso può essere moltissime cose, come un'impresa, una multinazionale, un mercato borsistico, un ospedale, internet, un distretto industriale, un'innovazione tecnologica. Un CAS può essere costituito da altri CAS al suo interno e allo stesso tempo costituire un elemento di un CAS più grande, inoltre, un CAS non evolve mai da solo ma evolve con altri CAS.

Uno dei più importanti effetti risultanti dallo studio della dinamica dei modelli economici complessi è stato il cambiamento nel metodo e nella direzione della ricerca. Nei nuovi modelli vengono descritte soltanto le interrelazioni a livello locale tra agenti individuali, mentre i comportamenti aggregati o le strutture emergono dall'auto-organizzazione, invece che essere semplicemente imposti o accettati. Ciò che risulta da un modello aggregato potrebbe non essere la semplice somma di quello che avviene a livello individuale: ciò è in contrasto con il metodo dei modelli ad agenti rappresentativi, nei quali la somma degli individui equivale all'aggregato.

Poiché un modello di un'azienda virtuale non può prescindere dal reale, ho esaminato come si crea e si trasmette la conoscenza all'interno di un'organizzazione. La costruzione di modelli di simulazione fondati su agenti, è un metodo decisamente teorico di analisi dinamica delle strutture sociali, sebbene l'obiettivo sia la plausibilità. Attraverso un modello così costruito, si può predisporre una rappresentazione della realtà che ha il vantaggio di riprodurre le regole di funzionamento e di consentire un'analisi dell'impatto che talune modificazioni del sistema causano e quindi la metodologia può essere applicata non soltanto allo studio teorico del sistema stesso, ma anche per simulare un sistema aziendale concreto. La conoscenza è una manifestazione del meccanismo dei modelli interni; essa dovrebbe intendersi come una sintesi di informazioni elementari, secondo un percorso di astrazione utile ad ogni singolo operatore e tale da rappresentare per costui un "abilitatore" nello svolgimento di attività complesse.

La conoscenza "esplicita" o codificata è trasmissibile con un linguaggio formale, sistematico. C'è però anche una conoscenza "implicita", che, in quanto appartenente ad una natura personale, è arduo formalizzare e comunicare. Un modello di simulazione basato su agenti adattivi complessi potrebbe aiutare a comprendere come si forma e trasmette questo tipo di conoscenza, così difficile da codificare.

Come ultimo passo prima di descrivere tecnicamente il modello di impresa virtuale costruito, nel sesto capitolo viene esaminato il concetto di simulazione al calcolatore, e si

descrivono l'ambiente Swarm, i linguaggi di programmazione Object Oriented ed in particolare Java.

Con la diffusione del computer, uno strumento di calcolo potente e versatile, la ricerca nelle scienze sociali ha subito interessanti sviluppi. Esistono numerosi strumenti software di simulazione, sviluppati parallelamente in ambito scientifico ed in ambito commerciale, che permettono di indagare la natura di modelli dinamici con l'uso della simulazione informatica. Oltre a consentire l'analisi di problemi che matematicamente sono intrattabili, tali strumenti consentono di dare maggiore rilievo al processo dinamico che conduce a situazioni di equilibrio. Tale metodo non porta necessariamente a migliorare i risultati ottenuti con le tradizionali tecniche statistico-matematiche o descrittive, ma fornisce decisamente una migliore plausibilità nel processo di formulazione del modello. Ciò significa che al fine di rendere matematicamente trattabile il modello spesso vengono poste ipotesi forti, la cui dimostrabilità è assai ardua. I risultati del modello sono sempre condizionati dagli assunti che vengono imposti per giungere alla soluzione.

Le problematiche da affrontare, se si intende costruire un modello economico attraverso l'uso del calcolatore, spesso riguardano aspetti che non sono propriamente inerenti alla sostanza di ciò che si vuole studiare. Diventa infatti necessario occuparsi degli aspetti tecnici relativi all'utilizzo del software. Lo strumento che si propone di risolvere parzialmente questo problema è chiamato Swarm.

Swarm non è un'applicazione, non un linguaggio di programmazione, neanche un ambiente di sviluppo, ma una vastissima libreria di potenti funzioni scritte in *Objective-C* in continua espansione ed una filosofia progettuale. L'unità fondamentale nel contesto del progetto è l'agente. La simulazione consiste di gruppi di molti agenti che interagiscono tra loro attraverso un meccanismo di notifica di messaggi.

Gli agenti costituiscono gli oggetti fondamentali del sistema; un orologio che funziona ad eventi discreti definisce i processi che essi devono eseguire allo scorrere del tempo. Ogni azione deve essere svolta in un punto preciso della lista di attività ed il tempo scorre in funzione dell'esecuzione di tali attività, per cui la rappresentazione temporale è relativa alle azioni e non al reale tempo di esecuzione della macchina.

La maggior parte dei linguaggi di programmazione sviluppatasi nel corso degli anni, era di tipo sequenziale. E' possibile descrivere un problema, o creare un modello, con una logica di tipo top-down, in cui cioè si segue una coerenza piuttosto rigida e lineare. In questi linguaggi, è sicuramente possibile creare *subroutines*, cioè parti di codice da eseguire più volte ed in maniera quasi autonoma, tuttavia per molti scopi della simulazione sociale questo tipo di programmazione si rivela particolarmente ostica.

La struttura logica degli sciame di agenti che interagiscono in base allo scatenarsi di eventi discreti è realizzata utilizzando il linguaggio di programmazione *Objective-C* o in alternativa *Java*, ma soprattutto il paradigma della programmazione ad oggetti. Essa consiste nella definizione di varie classi di oggetti, che possiamo pensare come descrizioni di diverse parti della realtà.

Un oggetto è la combinazione di variabili di stato che ne definiscono le caratteristiche interne ed i metodi, che forniscono ad esso la capacità di agire. In Swarm l'agente è costruito come un vero e proprio oggetto software.

Recentemente è possibile abbinare alla librerie di Swarm il linguaggio Java, come alternativa al tradizionale Objective-C. La caratteristica più rilevante di Java è l'alta portabilità del software prodotto: infatti esso promette di essere l'unico linguaggio, nel panorama informatico, ad essere indipendente dalla piattaforma in cui è eseguito. Questo elemento consente in prospettiva di diffondere il modello di simulazione con grande facilità all'interno della comunità di studiosi, eliminando il fastidioso compito di ricompilare il codice nelle diverse versioni Windows/Unix. Inoltre, è un linguaggio più diffuso poiché deriva sintatticamente dal C++. Per approfondire il linguaggio Java e per offrire agli utilizzatori di Swarm un prontuario di programmazione in questo linguaggio, ho scritto le appendici A e B, che lo esamino sotto il profilo sintattico e concettuale. In tali appendici sono contenuti i concetti fondamentali per costruire praticamente ogni simulazione attraverso Swarm.

Il settimo capitolo è sicuramente quello fondamentale del presente lavoro: esso descrive infatti il modello di impresa virtuale, chiamato JVE, costruito utilizzando Swarm.

Un modello deve essere il più possibile semplice, ma allo stesso tempo deve poter riprodurre la realtà, in modo da fornire risultati utilizzabili e studiabili. Nel nostro caso, ci troviamo a creare un modello per un'azienda di tipo tradizionale, facente parte del settore produttivo manifatturiero.

E' dunque necessario creare un insieme di unità produttive, ognuna delle quali è in grado di costruire un particolare componente, l'insieme dei quali costituirà l'oggetto finito da vendere al cliente finale.

Poiché il modello deve essere creato avvalendosi di un calcolatore, un'efficace similitudine è quella tra prodotto finito e numeri composta da diverse cifre. Il nostro scopo è dunque quello di "costruire" numeri di un numero predefinito di cifre: essi costituiscono il nostro prodotto finito ed ogni cifra è il componente singolo.

Le singole cifre vengono costruite dalle unità produttive, ognuna delle quali è in grado di svolgere solamente un processo produttivo, corrispondente, appunto, ad un

singolo componente (cifra). Ci troviamo di fronte dunque ad un'azienda con unità produttive altamente specializzate, ma integrate tra loro. E' chiaro dunque che le unità produttive devono esistere in numero maggiore, o al limite uguale, a quello delle cifre esistenti.

In Swarm, i prodotti devono essere configurati come oggetti, contenenti dunque diverse informazioni sotto forma di variabili, matrici e vettori; le unità produttive sono agenti, in grado di ricevere input ed inviare output.

A questo punto, si trova un'approfondita descrizione del codice Java che costituisce la simulazione stessa; il listato del programma è invece contenuto nell'appendice C.

In seguito, viene approfondita la trattazione dei magazzini, quale parte del modello, cioè dal punto di vista della simulazione. La programmazione ad oggetti concede la possibilità di creare con semplicità una moltitudine di unità di notevole complessità.

Nonostante ciò, in questa prima fase del modello, è preferibile inserire agenti semplici, in modo da rendere più lineare l'interpretazione dei risultati. Per questo motivo, i magazzini sono fondamentalmente dei contatori, che vengono incrementati ogni volta che viene inserito un componente, e decrementati quando l'oggetto è utilizzato per la produzione. Per decentrare il più possibile l'informazione, ogni unità produttiva ha un proprio magazzino, in cui vengono depositati i componenti da essa prodotti.

Nell'ottavo e ultimo capitolo vengono esaminati i risultati ottenuti attraverso il modello JVE e le possibilità di espansione futura, molto ampie grazie all'utilizzo di un linguaggio ad oggetti quale Java, e alle librerie di Swarm. Inizialmente sarà possibile introdurre nel modello la contabilità dei costi: poiché le aziende devono creare profitti, o per lo meno cercare di evitare perdite, una buon denominatore comune per verificare l'efficienza complessiva dell'impresa consiste nell'esaminare i costi e i ricavi.

Nel modello di impresa virtuale presentato, è necessario ricercare un modo piuttosto semplice per contabilizzare i costi, in modo tale da ottenere un indicatore, e non un valore congruente con un ipotetico mercato. Si terrà conto di costi fissi e variabili, ma anche di costi di diversa natura, quali quelli dovuti al ritardo nella consegna delle merci che causano insoddisfazione da parte del cliente.

Altra possibile evoluzione del modello consiste nell'introduzione: nelle aziende reali, non tutti i componenti richiedono un tempo di lavorazione uguale. Anzi, il tempo di lavorazione è spesso uno degli aspetti più cruciali per la programmazione della produzione, tanto da far spesso decidere in favore di eventuali *outsourcing*, nel caso in cui il componente da produrre costituisca un'incombenza troppo pesante per l'azienda, creando colli di bottiglia rilevanti all'interno della produzione.

AZIENDA VIRTUALE: TRASFORMAZIONE E MODELLI

1.1 Un sistema sociale aperto

Per studiare l'azienda, quale produttrice di beni e servizi, occorre ricordare che essa fa parte di un ambiente: l'azienda può infatti essere vista come un sub-sistema, all'interno di un sistema più ampio. L'ambiente in cui opera l'impresa non è solamente circoscritto al settore merceologico in cui opera strettamente, ma arriva ad interessare anche aspetti di osservazione più ampi, quali per esempio l'economia di un paese, risvolti legali, politici e altro.

L'azienda intrattiene dunque con il citato ambiente esterno complesse relazioni di interscambio, di tipo sociale, culturale ed economico, e questo ci permette di far riferimento ad essa come sistema "aperto", non ristretto cioè alla sola dimensione più contingente.

Schematizzando, potremmo dire che l'azienda riceve dall'esterno degli *input* (sotto forma di fattori produttivi, ordini, tecnologie, influssi di altro genere), i quali vengono opportunamente rielaborati e convertiti negli *output*, che a loro volta verranno trasmessi all'ambiente in qualità principalmente di prodotti e servizi, ma anche di conoscenze tecnologiche nuove, influenze politiche, e altro.

La diffusione di strumenti informatici potenti e dal costo relativamente ridotto, ha profondamente mutato le metodologie di studio inerenti a questo sistema così complesso e mutevole. Come si legge in Brynjolfsson e Hitt (2000), durante la Seconda Guerra Mondiale gli Stati Uniti investirono molto nello sviluppo di strumenti in grado di calcolare la traiettoria dei proiettili di artiglieria, contribuendo in parte allo sviluppo dei primi calcolatori digitali, già da tempo teorizzati. Non si deve però pensare ai moderni calcolatori come strumenti in grado solamente di facilitare i compiti che implicano grossi e pesanti calcoli, nonostante questo sia sicuramente un importante compito che sono in grado di svolgere. I moderni computer sono principalmente elaboratori multifunzione: le stesse tecnologie base possono infatti essere utilizzate per archiviare, richiamare, organizzare, trasmettere e trasformare ogni tipo di informazione che possa essere digitalizzata, cioè convertita in codice binario. La maggior parte delle situazioni che si affrontano per mezzo

di un computer non sono infatti semplici problemi di calcolo: le attività giornaliere della maggior parte dei manager, professionisti e persone coinvolte nella gestione di un'impresa richiedono un diverso tipo di pensiero.

Attualmente, grazie alla diminuzione del costo dell'hardware, cioè della componente fisica dei computer e all'aumento incredibile della potenza degli elaboratori, il limite maggiore non è più costituito dalla potenzialità di elaborazione, ma piuttosto dall'abilità degli operatori di inventare nuovi processi, procedure e strutture organizzative, che possano sfruttare adeguatamente i nuovi mezzi.

Il ruolo fondamentale dei calcolatori in economia è chiaro pensando alle organizzazioni ed ai mercati come elaboratori di informazioni (Galbraith, 1977; Simon, 1976; Hayek, 1945). La maggior parte delle istituzioni ed intuizioni economiche sono emerse in un periodo in cui i costi della comunicazione erano ancora relativamente alti e vi era una limitata capacità computazionale. La *Information Technology*, definita come l'insieme dei calcolatori e di tutte le tecnologie digitali ad essi correlate, ha l'incredibile potere di abbassare significativamente i costi della comunicazione e di conseguenza della coordinazione e dello scambio di informazioni. Per questo motivo, non deve sorprendere che la riduzione rilevante nei costi di calcolo e di comunicazione abbia portato ad una evidente ristrutturazione dell'economia: la maggior parte delle industrie attuali utilizzano infatti ampiamente le nuove tecnologie a loro disposizione.

Per questo motivo, in Bresnahan e Trajtenberg (1995), l'*Information Technology* è descritta non come un tradizionale investimento in capitale, ma come tecnologia *general purpose*, cioè di utilizzo generico. Le tecnologie di questo tipo forniscono un contributo economico più elevato di quanto si potrebbe prevedere semplicemente applicando al capitale investito un tasso di interesse medio: esse infatti facilitano l'adozione di innovazioni complementari.

Le antiche tecnologie ad ampio utilizzo, quali il telegrafo, il motore a vapore e quello elettrico, mostrano un insieme di innovazioni complementari che portarono ad aumenti di produttività notevoli. Molte di queste furono anch'esse di tipo tecnologico, ma gli sviluppi forse più interessanti e sicuramente più produttivi, furono innovazioni nel campo dell'organizzazione aziendale.

E' dunque facilmente comprensibile il motivo per cui gli investimenti in *Information Technology* stanno portando a produttività più elevata e trasformazioni nell'organizzazione delle imprese.

1.2 Information Technology in azienda

In Milgrom e Roberts (1990) leggiamo che per avere successo, un'azienda dovrebbe inserire i computer e la struttura informatica all'interno del sistema già esistente, introducendo come conseguenza i cambiamenti naturali nell'organizzazione; il passaggio all'*Information Technology*, non accompagnato da cambiamenti complementari non porterebbe benefici e potrebbe addirittura causare perdite. Questo avviene perché le difficili interazioni tra le pesanti pratiche esistenti all'interno dell'organizzazione e i nuovi sistemi introdotti compenserebbero e spesso supererebbero i benefici ottenuti.

Molte tra le pratiche attuate con più successo nel secolo scorso riflettono gli alti costi del processare informazioni. Per esempio in Malone (1987) e Radner (1993) si legge che le strutture organizzative gerarchiche possono ridurre i costi della comunicazione rendendo minimo il numero di legami richiesti per connettere più agenti economici. Similmente, un modo efficiente per utilizzare una tecnologia poco flessibile e ad alta intensità di scala è quello di produrre semplici prodotti standardizzati. Poiché il costo del elaborare informazioni in modo automatico è sceso di oltre 99 per cento dal 1960 in poi, difficilmente le pratiche attuate nei periodi precedenti potranno essere utilizzate anche ora, per massimizzare il valore della produzione flessibile e del basso costo delle informazioni.

1.3 La trasformazione dell'azienda

Molte imprese condividono la necessità di far coincidere la struttura organizzativa alle capacità della tecnologia e la volontà di attuare la transazione verso un processo produttivo ad alta intensità tecnologica. Per fornire maggior configurabilità e varietà dei prodotti offerti, negli anni '90, molte imprese effettuarono grossi investimenti in processi produttivi integrati con computer. Questo investimento veniva spesso accompagnato da una lunga lista di cambiamenti complementari, quali l'eliminazione di tabelle di produzione rigide, innovazioni di processi e flussi, maggior interazione tra clienti e fornitori, aumento di lavoro di squadra e altre modifiche per quanto riguarda le abilità dei singoli agenti, processi, cultura e struttura. Tuttavia, inizialmente, molte imprese rimasero deluse dai risultati forniti da queste innovazioni: questo avvenne soprattutto perché molti lavoratori mantenevano ancora elementi delle ormai obsolete pratiche precedenti al cambiamento, per una sorta di conoscenza involontaria. Quasi per ironia della sorte, le

nuove attrezzature erano talmente versatili, che i lavoratori potevano utilizzarle esattamente come facevano con quelle vecchie.

In alcune imprese, si concluse una buona tattica era quella di introdurre il nuovo equipaggiamento dapprima solo in una parte dell'azienda, inserendo nell'organizzazione nuova forza lavoro, che dunque non aveva potuto assorbire la conoscenza creata precedentemente.

Gli aumenti di produttività risultanti da questa strategia furono significativi e questo dimostrò che solo una forza lavoro addestrata a sfruttare al meglio le nuove tecnologie poteva portare a veri risultati.

1.4 Cambiamenti nell'interazione con i fornitori

A causa dei problemi di coordinazione con fornitori esterni, spesso le grandi imprese producono esse stesse molte tra le componenti più utilizzate. Un esempio significativo è costituito dalla General Motors, compagnia il cui successo venne in parte favorito dagli altissimi livelli di integrazione verticale. Tuttavia, le tecnologie quali lo scambio elettronico dei dati, sistemi di approvvigionamento basati sulla rete, e altri sistemi informativi inter-aziendali hanno singificativamente ridotto il costo, tempo e difficoltà di comunicazione con i fornitori.

Per esempio, le imprese possono effettuare ordini verso fornitori e ricevere conferma per via telematica, eliminando la necessità di documenti scritti, riducendo i ritardi e gli errori associati al processo manuale di gestione ordini.

L'integrazione elettronica di una catena di fornitura si è particolarmente evoluta nel settore industriale della merce confezionata. Tradizionalmente, i produttori promuovevano prodotti come sapone o detersivi offrendo sconti, o promozioni nei confronti degli stessi rivenditori. Poiché molti prodotti di consumo hanno lunga vita di scaffale, i venditori tendevano ad acquistarne grandi quantità durante questi periodi di promozione, il che poteva distorcere le previsioni di vendita e la visione del mercato del produttore. Per questo motivo, i produttori resero più frequenti le modifiche alle confezioni, al fine di scoraggiare l'accumulazione di grosse quantità di merce e crearono reparti interni con il compito di controllare il comportamento degli acquirenti ed eventuali inadempienze contrattuali (Clemons, 1993).

Per questo motivo, molte imprese di questo tipo investirono in sistemi informativi in grado di trasmettere al produttore ogni transazione effettuata, tramite uno scanner

montato nel punto vendita. Inoltre, resero automatici gli inventari, e la gestione dei magazzini.

Questi cambiamenti, combinati, aumentarono moltissimo l'efficienza; i clienti avevano come beneficio il prezzo inferiore e l'aumento della varietà dei beni, la convenienza e l'innovazione. Senza i collegamenti tra computer che trasmettevano gli acquisti effettuati, non si sarebbe potuta raggiungere l'alta efficienza e l'accuratezza attuali.

Le innovazioni tecnologiche derivate dalla diffusione dell'internet hanno diminuito incredibilmente il costo da sostenere per la creazione di collegamenti tra diversi agenti. L'approvvigionamento attraverso la rete ha portato ad una riduzione dei costi attraverso una riduzione del tempo necessario e a consegne più prevedibili, il che riduce la necessità dei cosiddetti *buffered inventories*, cioè inventari in cui si teneva conto dell'incertezza delle consegne e dunque si ordinava più dello stretto necessario.

Queste innovazioni, quando possono essere introdotte, riducono i costi generali dei prodotti acquistati nella misura dal 10 al 40%, secondo il tipo di settore merceologico considerato. Molti di questi risparmi rappresentano effettivamente una ridistribuzione tra fornitori ed acquirenti, con piccole conseguenze sul risultato economico complessivo del sistema; ciò nonostante, molti altri cambiamenti rappresentano veri miglioramenti in produttività, attraverso una maggior efficienza produttiva e indirettamente portano ad un miglioramento nella qualità del prodotto e nella varietà delle offerte senza costi eccessivi.

Per questo motivo, sempre più aziende hanno ridotto l'integrazione verticale, rivolgendosi sempre più verso l'esterno. Riprendendo l'esempio della General Motors, in Schnapp (1998) si legge che una delle maggiori debolezze del colosso economico è proprio il costruire parti all'interno, aspetto che precedentemente costituiva un punto di forza.

1.5 Cambiamenti nella relazione con i clienti

L'internet ha aperto una nuova gamma di possibilità per rendere più ricche le interazioni con i clienti. Per esempio, Dell Computer ha aumentato moltissimo la propria clientela, permettendo agli acquirenti di creare la configurazione preferita direttamente da un sito web, dal 1998. Quest'innovazione è stata accompagnata da cambiamenti nei sistemi e nelle pratiche aziendali, che potessero enfatizzare la gestione dell'inventario con tecnica *just-in-time*, e della produzione di tipo *build-to-order*. Per questo motivo, Dell ha rinunciato alla consueta rete di vendita, tipica di altri concorrenti, ottenendo circa il 10% di vantaggio per quanto concerne i costi di produzione. Parte di questi risparmi

rappresentano l'eliminazione della distribuzione e della vendita al dettaglio; altri riflettono invece i più bassi livelli di scorte lungo il canale di distribuzione. Il maggior vantaggio conseguito a questa innovazione è comunque costituito dalla rapidità di risposta verso i clienti, aumentata incredibilmente rispetto ad un'organizzazione di tipo tradizionale: per esempio, nel momento stesso in cui Intel o AMD immettono un nuovo processore sul mercato, più volte durante l'anno, Dell riesce a inserirlo nelle configurazioni proposte entro sette giorni. Si tratta di un periodo assolutamente basso soprattutto se paragonato alle varie settimane necessarie ad alte compagnie concorrenti, di tipo più tradizionale. Questo diviene un vantaggio competitivo di assoluto rilievo, soprattutto in un campo in cui le innovazioni si succedono con tanta rapidità e l'obsolescenza della tecnologia è molto rapida, con la conseguenza che i prezzi dei componenti hardware diminuiscono del 4% ogni mese, in media.

Altre imprese, attraverso la rete, hanno instaurato migliori relazioni con la propria clientela; per esempio venditori come Amazon, che fornisce suggerimenti personalizzati ai propri visitatori e permette di personalizzare numerosi aspetti dei loro acquisti.

Realtà quali le compagnie di trasporti o corrieri, hanno inserito sistemi di *tracking* basati su web, che permettono di tener sotto controllo le spedizioni effettuate in tempo reale. Il costo di questo servizio basato su web è pari al 5% di quello che si sosteneva precedentemente, per fornire le stesse informazioni attraverso il telefono; inoltre, molti clienti che precedentemente non avrebbero richiesto tali informazioni, ora possono ottenerle semplicemente, con un aumento evidente della loro soddisfazione e snellimento delle procedure.

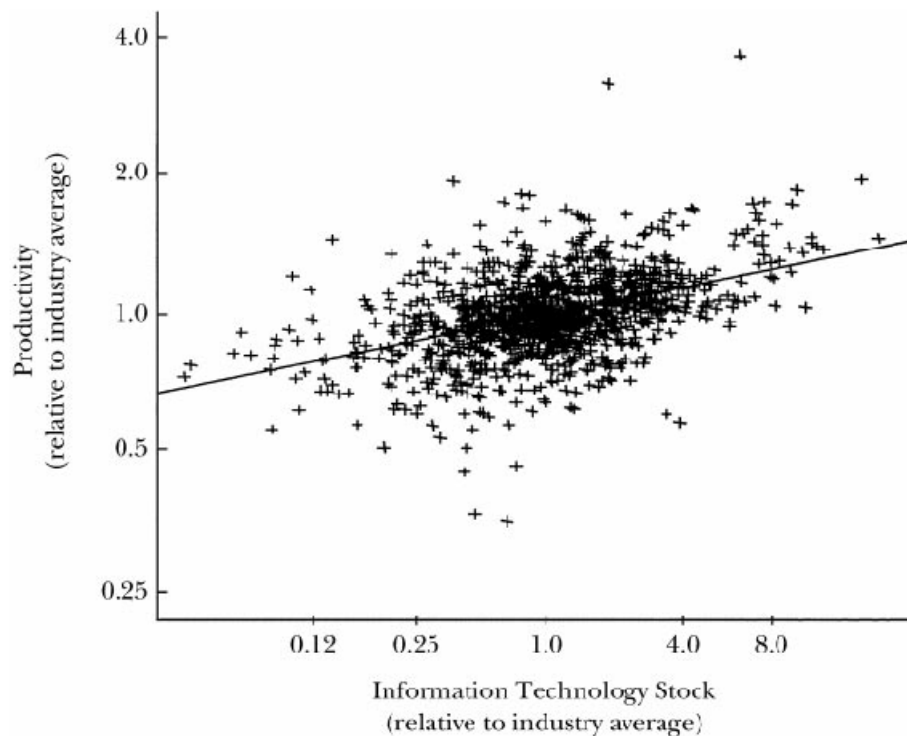
1.6 Information Technology e produttività

Gran parte delle prime ricerche, condotte negli anni '80, sulla relazione tra tecnologie e produttività, utilizzava solamente dati di tipo economico o settoriale; per questo motivo, spesso si concludeva che, mentre l'investimento in tecnologia rapportato alla dimensione della forza lavoro era aumentato incredibilmente dal 1977 al 1989, il prodotto per lavoratore, misurato in modo convenzionale, non era cresciuto in modo apprezzabile.

Nei primi anni '90, le analisi condotte a livello delle imprese, iniziarono a fornire le prove che i computer aumentavano sensibilmente il livello di produttività dell'azienda. Utilizzando i dati provenienti da oltre trecento grandi imprese, Brynjolfsson e Hitt (1995,

1996) e Lichtenberg (1995) stimarono delle funzioni di produzione che utilizzavano il valore aggiunto dell'impresa come variabile dipendente e includevano capitale tradizionale, capitale investito in IT, lavoro di tipo tradizionale, lavoro di tipo tecnologico ed una moltitudine di altre variabili indipendenti per rappresentare il tempo, il settore industriale e l'azienda stessa.

Uno dei risultati fu quello di legare la produttività aziendale al livello di *Information Technology* impiegato, come da figura seguente.



Sicuramente, si può notare una relazione generalmente positiva tra le due grandezze, ma anche una grande variabilità di risultati tra le diverse aziende oggetto dello studio. Questo può derivare dal fatto che in questa ricerca si è tenuto conto della quantità di capitale investito nel passaggio all'*Information Technology*, senza però considerare il tipo di processo informatico introdotto; ci possono dunque essere, a parità di investimento, processi dotati di maggior ricaduta sulla produttività generale dell'azienda.

1.7 Stime a livello economico aggregato

Esistono molti dati che provano l'esistenza di un valore aggiunto creato dall'introduzione dell'*Information Technology* a livello della singola impresa; è tuttavia estremamente difficile legare questi benefici ad una visione macroeconomica dell'economia d'impresa. La ragione di questo è costituita dal fatto che le tecniche di contabilità della crescita tradizionali hanno come oggetto gli aspetti dell'output più facilmente osservabili, quali prezzo e quantità, mentre non tiene conto di benefici difficilmente quantificabili, quali miglioramento della qualità, introduzione di nuovi prodotti, attenzione al cliente e aumento della rapidità delle catene di forniture.

Allo stesso modo, quando si traccia una statistica sul cambiamento dell'economia a seguito dell'introduzione dell'*Information Technology*, si studiano aspetti quali il prezzo e la quantità del materiale elettronico acquistato, mentre non si considerano sviluppi collaterali di nuovi prodotti, servizi, mercati, processi di business e cambiamento delle competenze dei lavoratori.

Tuttavia, i tradizionali sistemi di misurazione costituiscono un punto di partenza utile per capire il vero impatto dell'*Information Technology* sull'economia a livello aggregato. Alcuni studi (Jorgenson e Stiroh, 1995; Oliner e Sichel, 1994), sono giunti a concludere che il progresso tecnologico ha portato un contributo pari a circa lo 0.3% annuo alla crescita economica, considerando dati compresi tra il 1970 ed il 1980. Gli stessi studiosi, in ricerche più recenti, sono giunti alla conclusione che nella seconda metà degli anni '90, il contributo delle nuove tecnologie all'economia aggregata è di circa 1.1%.

1.8 Il concetto di azienda virtuale ed il modello MAIS

Con il termine di azienda virtuale si fa riferimento a due concetti: da una parte, esso è riferito all'azienda vera e propria, vista come componente di un organismo sovraziendale virtuale, che attraverso la collaborazione di tutte le unità della catena di produzione svolge quello che svolgerebbe un'azienda totalmente integrata verticalmente; essa è capace di svolgere tutte le fasi del ciclo di produzione al suo interno.

Il concetto di azienda virtuale ha anche significato di modello d'azienda, creato attraverso una rappresentazione informatica con lo scopo di consentire la simulazione.

Facendo riferimento al concetto di azienda virtuale come aggregazione di strutture aziendali reali all'interno di una struttura informatica comune, esistono varie tecnologie e progetti finalizzati a costruire un'infrastruttura che dia vita al modello. Nella seconda accezione, invece, la simulazione dell'azienda attraverso un modello al calcolatore richiede la definizione di un protocollo di integrazione delle informazioni e la progettazione degli agenti che rifletta le capacità di partecipare alla simulazione, rispecchiando al contempo gli agenti reali a cui fanno riferimento. In sostanza deve essere implementato un agente che contenga contemporaneamente gli elementi degli agenti logici e fisici dei modelli MAIS (*Multi-Agent Information System*), applicazione della metodologia di simulazione basata su agenti, nell'ambito della gestione delle catene di fornitura, proposta da Lin, Tan e Shaw (1996).

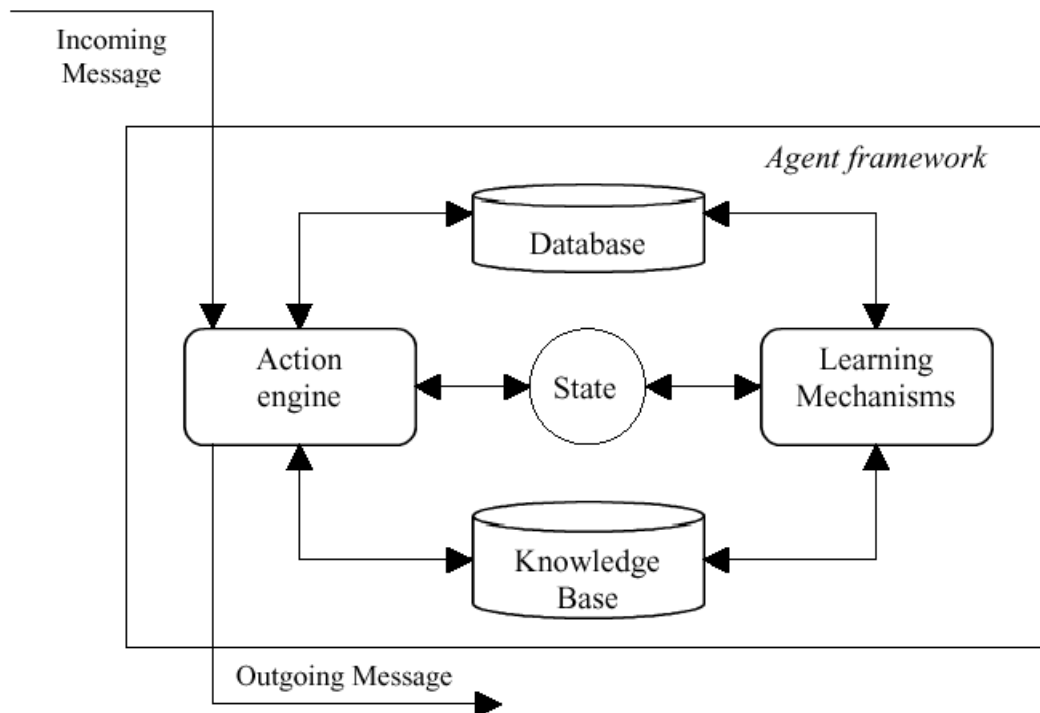
Il modello è composto da quattro componenti: gli agenti, le attività, le organizzazioni e l'infrastruttura informativa. Le componenti del MAIS sono così descritte:

1. Un agente è un oggetto attivo che possiede capacità di compiere operazioni e di comunicare con altri agenti sulla base di strutture organizzative, che favoriscano la cooperazione al fine di portare a termine le attività.
2. Le attività sono compiti affidati agli agenti. In funzione della struttura degli agenti nella organizzazione, le attività possono essere suddivise e distribuite tra più agenti, oppure aggregate in una singola attività, affidata ad un solo agente.
3. Le organizzazioni sono formate in base alle relazioni tra gli agenti. Un'organizzazione è formata dalle relazioni di controllo e di informazioni che si instaurano tra gli agenti. Esistono due tipi di controllo: centralizzato e decentralizzato.
4. L'infrastruttura informativa che garantisce agli agenti il flusso delle informazioni di cui necessitano. Nei MAIS la memorizzazione numerica delle informazioni, delle operazioni e delle comunicazioni sono utilizzate per garantire le interazioni tra gli agenti. Il flusso di informazioni è realizzato attraverso *database* condivisi o attraverso il passaggio di messaggi.

In questo modello è abbandonata la descrizione delle informazioni attraverso gli schemi dei *database* aziendali. Le informazioni diventano un prodotto dell'azione dell'agente, quindi un risultato del modello e non più un punto di partenza.

Esistono due tipologie di agenti: fisici e logici. La struttura interna degli agenti logici è rappresentata nella figura seguente e ha una rappresentazione dello stato dell'agente in base alle variabili interne (*state*). Il motore delle azioni (*action engine*)

decide i comportamenti dell'agente e le procedure per metterli in atto. Il *database* memorizza le informazioni relative alle attività svolte, al fine di renderle disponibili ai processi interni e agli altri agenti. La *knowledge base* memorizza la conoscenza, come prodotto del processo di apprendimento e la rende disponibile al motore delle azioni. Essa è rappresentata da un *database* in cui sono registrate le informazioni in una metrica compatibile con le regole o con i processi adattivi messi in atto dai *learning mechanisms*. Infatti, i meccanismi di apprendimento producono le informazioni registrate nella *knowledge base*.



Gli agenti fisici non sono dotati di capacità adattiva: svolgono semplicemente le attività quando stimolati dagli eventi del sistema, usando una scansione del tempo ad eventi discreti. Nelle catene di fornitura si identificano i seguenti agenti fisici: fornitori, magazzini, produttori, distributori e rivenditori. Essi si occupano soltanto di garantire i flussi di informazioni relativi al ciclo di evasione degli ordini, il quale consiste nel sistema di raccolta ordini, di pianificazione della produzione, delle materie prime, della produzione ed il controllore del flusso di attività. Gli agenti logici sono più attivi, ed adattivi, poiché usano un modello di tempo a passi successivi, con eventi non generati dall'esterno e sono quindi in grado di mettere in atto attività sulla base di una decisione autonoma, elaborata dall'*action engine*.

Il meccanismo di passaggio dei messaggi e delle informazioni, le tecniche di immagazzinamento delle stesse e la capacità adattiva e di apprendimento devono essere integrate nella struttura del singolo agente e devono possedere un'interfaccia di scambio comune.

Il modello comprende in modo molto approfondito tutti gli elementi che influiscono sulla dinamica della catena di fornitura. L'intera struttura è disegnata per essere codificata nel sistema di simulazione *Swarm*, del quale si parlerà approfonditamente in seguito. Da un punto di vista concettuale è stata realizzata una corrispondenza tra gli elementi fondamentali che caratterizzano la catena di fornitura con gli oggetti che *Swarm* mette a disposizione.

Dopo aver costruito il modello MAIS è necessario identificare i fattori ambientali che caratterizzano l'intera catena di fornitura e che devono essere trasposti nel modello come parametri di simulazione. Essi consentono di diversificare i risultati dell'esperimento dimostrando le conseguenze della scelta di particolari strutture organizzative o politiche di gestione. Sono rappresentati da:

1. dati relativi alla domanda di beni finali. Può consistere nella variazione quantitativa di prodotti domandati, nella variazione di tipologia e caratteristiche di prodotto, oppure nella variazione dei tempi di attesa dei consumatori.
2. meccanismi di controllo delle strutture organizzative. La variazione dei meccanismi di controllo e della struttura delle relazioni tra entità della catena di fornitura provocano una variazione nei flussi delle informazioni. Si possono contemplare diverse configurazioni strutturali dell'organizzazione alle quali corrispondono altrettante strutture nel modello di simulazione.
3. infrastruttura delle informazioni. Essa è finalizzata a semplificare le comunicazioni e la coordinazione tra gli agenti. Differenti infrastrutture e differenti livelli di trasparenza delle informazioni producono sensibili differenze nei risultati dell'intero modello.
4. Modelli di azione degli agenti. Differenti strategie in termini di politiche di produzione e stoccaggio devono essere inserite nei modelli di comportamento degli agenti e producono differenti risultati nell'evasione degli ordini.
5. Incertezza. Si possono osservare variazioni inattese della domanda, nella struttura degli ordini, nei tempi di attesa, interruzioni della produzione dovuta a rottura delle macchine oppure ritardi nella consegna dei prodotti finiti.

Il modello è sviluppato come una generica rappresentazione di una catena del valore e può essere usato per sperimentare diverse strategie al fine di migliorare le performance del processo di evasione degli ordini. Gli esperimenti analizzano il coordinamento tra le componenti del sistema informativo all'interno di ciascuna unità di business, il coordinamento tra le strategie operative tra le diverse unità di produzione e le potenzialità della gestione dell'intera catena di produzione.

La simulazione avviene sulla base della scelta degli opportuni parametri di input, i quali determinano la configurazione dell'esperimento in termini di struttura dell'azienda e di combinazione di strategie e politiche di gestione. I risultati ottenuti riguardano la valutazione del trade-off tra tempi di produzione e livello delle scorte, l'efficienza delle diverse combinazioni che si possono realizzare in relazione alle politiche operative, le variazioni della domanda ed il suo effetto sulle performance dell'azienda.

I parametri di input danno origine a strutture organizzative e comportamentali diverse. Ogni agente è dotato di differenti metodi per agire in funzione di differenti politiche aziendali o strategie ed è capace di generare autonomamente i dati e gli stimoli per innescare il ciclo di evasione degli ordini.

Attraverso il meccanismo del passaggio dei messaggi vengono attivati taluni metodi piuttosto che altri in funzione dei parametri globali del modello.

L'ipotesi MAIS è principalmente accademica: i modelli di questo tipo rendono infatti troppo complessa la realizzazione di simulazioni di realtà concrete, poiché la descrizione del modello impone di realizzare una congettura sulle variabili esogene ad esso. Le variazioni della struttura della domanda, la possibilità di rottura delle macchine, i ritardi nei tempi di consegna, devono essere generati come fenomeni casuali all'interno del sistema. Le preferenze e le attitudini degli agenti a valle della catena, cioè i clienti o i rivenditori, devono essere colte dallo sperimentatore ed incluse nel progetto degli agenti.

Per affrontare la simulazione sul "campo" è necessario affidarsi ad un modello che si fondi su presupposti diversi. Gli agenti devono essere progettati per dare vita ad un modello di "azienda virtuale", inteso come modello che riproduce il funzionamento dell'azienda su presupposti realistici, pur includendo la possibilità di simulare ipotesi di intervento sulla struttura o sui processi.

1.9 L'architettura NIIP

La realizzazione di un modello di azienda virtuale intesa come aggregato di aziende reali dipende dalla capacità di integrazione delle tecnologie che consentono lo scambio di materiali e soprattutto di informazioni. In questo caso gioca un ruolo fondamentale la tecnologia di memorizzazione delle informazioni e l'elemento innovativo rappresentato dall'internet. Gli elementi chiave della realizzazione di una tale infrastruttura forniscono un prezioso modello, applicabile anche nella descrizione dell'azienda, intesa come organismo individuale. Tra i modelli che descrivono le caratteristiche di un'azienda virtuale industriale, il più interessanti è il *NIIP*.

L'obiettivo primario del *National Industrial Information Infrastructure Protocols Consortium* (NIIP) è lo sviluppo di una piattaforma tecnologica per la realizzazione della *Industrial Virtual Enterprise*. Con questo termine si intende un'organizzazione temporanea di aziende che si coalizzano per condividere i costi infrastrutturali e costruire le opportunità che da sole non saprebbero cogliere. Tale metodologia incoraggia gli sforzi nella condivisione delle informazioni relative alle tecnologie produttive.

Il consorzio è stato sovvenzionato dal governo americano e dimostra l'attenzione e l'interesse da parte dell'economia alla nuova frontiera della comunicazione globale. Il NIIP è definito come un'iniziativa nata dalla collaborazione tra Industria e Governo, atta a sviluppare un'infrastruttura informativa che permetta la collaborazione tra le imprese che lavorano su diversi aspetti dello stesso processo produttivo. Questa collaborazione è volta a diminuire il tempo richiesto dall'intera catena di fornitura e aumentare la rapidità di cambiamento.

Le aziende che partecipano a questo progetto eseguono le proprie attività come se facessero parte di un'unica organizzazione, attraverso l'uso di un potente sistema di accesso e gestione a tutte le informazioni necessarie allo svolgimento del ciclo produttivo.

A livello più astratto si identificano quattro tecnologie necessarie alla realizzazione di una azienda virtuale:

1. Protocolli di comunicazione comuni.
2. Tecnologia ad oggetti uniforme per consentire la comunicazione tra i sistemi e tra le applicazioni.
3. Modello di scambio e di dettaglio delle informazioni comune.
4. Gestione cooperativa dei processi integrati della *virtual enterprise* (VE).

Il primo livello di definizione delle tecnologie che consentono di dar vita alla VE è costituito dallo strato che garantisce lo scambio di informazioni. L'internet è ridefinito all'interno di questo paradigma come "la super-autostrada nazionale delle informazioni".

Esso è l'elemento chiave di questa tecnologia. Sulla base del paradigma del NIIP, un punto della VE invoca un servizio con l'invio di un messaggio attraverso questa infrastruttura. Esso è implementato attraverso un componente software compatibile con la tecnologia CORBA 2.0 (Common Object Request Broker Architecture). Ogni messaggio deve essere inviato:

1. Confidenzialmente - senza divulgazione a soggetti terzi non autorizzati.
2. Integralmente – con la certezza che i dati non vengano alterati.
3. Autenticamente – la fonte deve essere identificabile in modo attendibile.

La sicurezza delle informazioni può essere garantita dai protocolli di comunicazione dell'internet (*TCP/IP*), all'interno dei componenti software che realizzano lo strato di interscambio delle informazioni, o dalla stessa architettura del NIIP che fa riferimento ai meccanismi di criptazione a chiave pubblica.

Il ruolo svolto dallo strato delle comunicazioni è la facilitazione nell'accesso e nella localizzazione delle informazioni. I servizi di ricerca testuale forniti dall'internet garantiscono tale necessità.

Il secondo livello dell'architettura definisce la tecnologia degli oggetti software. Essi devono rispondere alle specifiche denominate CORBA 2.0. Tale tecnologia permette di creare strati di software che eseguono operazioni e sono in grado di comunicare all'esterno grazie ad un linguaggio detto IDL (*interface definition language*). Esso fornisce agli oggetti un'interfaccia che garantisce la loro visibilità attraverso la rete.

Per completezza va ricordato che esiste un'altra tecnologia che fornisce questo tipo di servizi ed è indicata con la sigla DCOM (*distributed component object model*), che però non è ritenuta standard nel progetto NIIP.

Il terzo livello si occupa di definire le modalità attraverso le quali deve essere definito lo schema di descrizione delle informazioni. Attraverso il modello STEP (*the international standard for exchange of products*) tutti gli stadi del processo di produzione del bene o del servizio devono possedere una struttura specifica di informazioni disponibili relative al prodotto, in modo da garantire la compatibilità con le altre realtà produttive.

Il protocollo richiede che siano rispettate queste specifiche:

1. Interoperabilità. Le informazioni relative ai differenti gruppi di lavoro devono poter operare tra loro in modo da costituire un'unica struttura logica di informazioni relative al prodotto.
2. Progettazione concorrente. Tutte le applicazioni devono sviluppare differenti aspetti del progetto, compresi i processi di produzione.
3. Documentazione del progetto. La documentazione relativa a ciascun prodotto deve comprendere tutti gli aspetti, ma essere integrata in un unico documento.

L'ultimo livello riguarda la predisposizione delle strutture di descrizione delle informazioni che garantiscono la possibilità di collaborazione tra aziende. I servizi di gestione delle attività forniscono uno strumento che permette l'attiva collaborazione tra le parti, attraverso la realizzazione di un modello che unifica i flussi di dati, i flussi di operazioni, le relazioni semantiche tra le attività, i ruoli, i gruppi, le applicazioni ed i dati. Tale modello fornisce ai membri dell'azienda virtuale le risorse necessarie all'identificazione delle sequenze di attività in cui devono operare e della documentazione necessaria. Nella prima fase il modello identifica le risorse di ciascun componente. I dati, le associazioni tra essi, le regole, le operazioni ed i metodi sono definiti da uno schema ad oggetti che ha valore soltanto localmente all'azienda. Questi schemi catturano il contenuto semantico delle risorse di ciascun membro dell'organizzazione.

Nella seconda fase è definito un schema globale che descrive la totalità delle risorse dell'organizzazione virtuale. La terza fase prevede la descrizione dei meccanismi che permettono di mediare i contenuti dei singoli schemi locali nella metrica dello schema globale. I nomi delle risorse devono essere rimappati in caso di potenziali sovrapposizioni e devono essere risolte le discrepanze nella semantica di rappresentazione delle stesse.

Il risultato di questo processo è la descrizione unitaria di un meta-modello dell'azienda virtuale. Il progetto NIIP rappresenta un grande sforzo collettivo delle imprese americane per realizzare sul campo ciò che viene da tempo promesso dal *business-to-business*. Il potenziale difetto di questa metodologia sta nell'enorme impegno di risorse progettuali e contemporaneamente nella scarsa flessibilità del risultato. I progettisti del paradigma NIIP ha dovuto scegliere alcune tecnologie e definire gli standard in una realtà che è in continua e turbolenta evoluzione. Le tecnologie informatiche migliorano e proliferano con una frequenza tale che la definizione degli standard rischia di essere obsoleta ancor prima del loro definitivo rilascio. Un esempio è fornito dalla constatazione che nello standard NIIP non è previsto l'utilizzo del linguaggio XML come formato universale di descrizione e trasporto delle informazioni, mentre ciò si sta realizzando di fatto nel mercato.

1.10 Un esempio pratico: BizTalk

In Saltarin (2000) si fa notare che si tratta di una vera e propria filosofia, di un modo di pensare, di una nuova visione dell'antico problema dell'EDI. Per scambiare informazioni occorre che i sistemi coinvolti nello scambio siano, in qualche modo, integrati. Ogni azienda porta con sé un patrimonio di decisioni informatiche che rendono il suo sistema informativo assolutamente unico: si va dalla scelta delle macchine, a quella dei sistemi operativi a quella dei database.

Le soluzioni basate sull'infrastruttura BizTalk sembrano rispondere ai requisiti di semplicità ed integrazione, semplicemente disaccoppiando definitivamente le applicazioni dai dati. Questi viaggiano da e per ciascuna applicazione in un formato standard e comprensibile da tutti gli attori del sistema. Lo strato di "colla" è il parser XML.

In questo scenario BizTalk riveste un ruolo importante: regolamentare il modo in cui questi documenti XML dovranno essere scritti. In questa infrastruttura è previsto che tutti i dati viaggino attraverso comunicazioni sicure (cifrati all'origine e decifrati a destinazione).

Da un punto di vista tecnico BizTalk è una serie di raccomandazioni e specifiche per utilizzare XML (e Schema) in modo coerente allo scopo di scambiare dati tra differenti sistemi aziendali.

Un apposito portale (<http://www.biztalk.org>) raccoglie poi tutte le soluzioni di integrazione che aderiscono al progetto, in modo da creare una base di soluzioni pronte all'uso. I motivi del fallimento delle soluzioni di EDI nel passato costituiscono il punto di partenza da cui sono stati derivati gli obiettivi di BizTalk.

L'obiettivo principale, quindi, consiste nel costruire una soluzione standard di EDI indipendente dai protocolli di trasporto. Questo si ottiene rendendo ogni documento aziendale in un documento XML ben formato ed infine aggiungervi alcune caratteristiche specifiche di BizTalk che definiscono l'interfaccia tra il documento ed il resto del mondo.

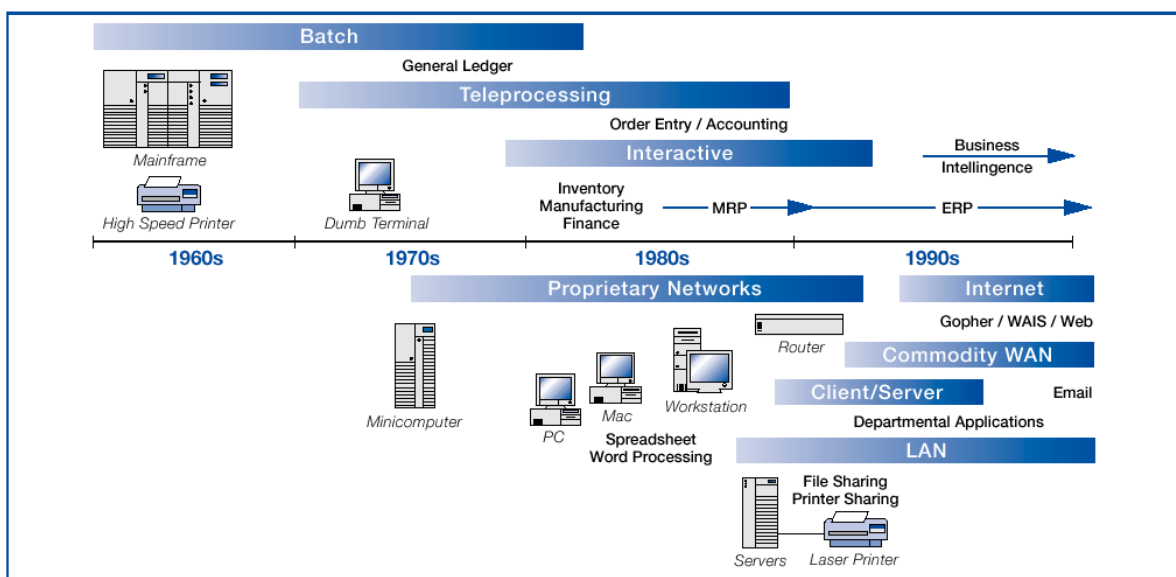
Il futuro dell'e-commerce e dell'integrazione delle informazioni si gioca sull'affidabilità dello scambio di dati tra aziende. Questo scambio deve avvenire in maniera sicura, asincrona, standardizzata. Deve essere disponibile una soluzione poco costosa, scalabile e facile da mantenere.

La soluzione vincente, forse, nascerà da un'infrastruttura che permetterà ad ogni azienda di fare e-commerce mantenendo la propria piattaforma informativa. BizTalk ed il progetto NIIIP sono due tentativi in questa direzione.

EVOLUZIONE DEL SISTEMA INFORMATIVO IN AZIENDA

2.1 Sistema informativo aziendale

Sin dagli anni '50, il sistema informativo ha occupato una posizione di rilievo all'interno delle aziende. Inizialmente si trattava semplicemente di un rimpiazzo elettronico per i tabulati e gli altri metodi meccanici di contabilità di base, ma ben presto diventò più sofisticato e complesso, tanto da coprire tutti gli aspetti economici dell'impresa. Inizialmente, il sistema informativo si occupava maggiormente dell'organizzazione delle singole imprese e non dell'interazione tra diverse realtà economiche. Semplici sistemi, quali l'OLTP (*on-line transaction processing*), in grado di tener sotto controllo lo stato dell'intera azienda, costituivano già un'evoluzione eccezionale rispetto alle metodologie manuali. Le applicazioni ed i sistemi sono divenuti sempre meno costosi, ma più affidabili e potenti: come indicato dalla legge di Moore (1965), i circuiti integrati sono cresciuti del 100%, come numero di transistor, ogni 18 mesi circa e questo ha portato a processori sempre più potenti, ma anche a memorie di massa più ampie e veloci. Nella figura seguente, è sono riprodotte schematicamente le tappe della crescita dell'*Information Technology* in azienda, tratte da un paper di Intel del 2001.



L'informazione è la base fondamentale per una consapevole amministrazione d'azienda; attengono al sistema informativo tutte le attività di raccolta, elaborazione e trasmissione dei dati realizzate al fine di soddisfare le esigenze conoscitive interne ed esterne.

Il sistema informativo aziendale, complesso di elementi la cui localizzazione è spesso variamente frazionata, opera per produrre e distribuire informazioni rilevando ed elaborando sinteticamente ed in modo organizzato i fenomeni economici di interesse aziendale.

Gli elementi che costituiscono il sistema informativo aziendale possono essere distinti in:

1. un insieme di dati rappresentativi della realtà aziendale ed ambientale
2. un complesso di procedure per la raccolta dei dati, la realizzazione e distribuzione delle informazioni
3. risorse umane e tecnologie impiegate nell'attuazione del processo informativo

L'obiettivo fondamentale del sistema informativo aziendale è quello di fornire, a chi opera nell'azienda, le informazioni necessarie per lo svolgimento delle mansioni, attraverso un processo continuo di trattamento di dati rappresentativi della realtà aziendale esterna.

I dati costituiscono la materia prima del processo di produzione delle informazioni, in quanto rappresentazione originaria e non decifrata di un fenomeno, attuata attraverso simboli (lettere, numeri, caratteri speciali). Essi acquisiscono un significato sul piano economico in seguito ad una appropriata classificazione, organizzazione ed al loro effettivo impiego nei processi decisionali e di controllo.

La distinzione tra dato (che costituisce l'input del processo informativo) e informazione (l'output), ha carattere di relatività e può assumere diverse connotazioni a seconda del destinatario e della modalità di utilizzo.

Ogni sistema informativo, considerando le esigenze dei soggetti interni ed esterni all'azienda, deve perseguire le seguenti finalità:

1. fornire ad ogni centro decisionale tutte le informazioni di cui necessita al fine di razionalizzare la relativa assunzione dei provvedimenti
2. preordinare i dati e le conoscenze per soddisfare i fabbisogni informativi manifestati dall'ambiente esterno

La richiesta di informazioni all'interno dell'azienda dipende dalle caratteristiche del sistema delle decisioni e dei controlli aziendali; verso l'esterno, invece, i flussi informativi sono collegati, prevalentemente, a disposizioni legislative (obblighi di pubblicazione dei bilanci d'esercizio, legislazione fiscale, dichiarazione dei redditi e così via), a rapporti di natura contrattuale con soggetti che hanno relazioni con l'azienda (fornitori, clienti, istituti bancari, pubblica amministrazione) ed a politiche che pongono la comunicazione in primo piano.

2.2 Il processo di produzione delle informazioni

Il processo di produzione delle informazioni si articola nelle fasi di rilevazione dei dati, elaborazione degli stessi e trasmissione delle informazioni.

La rilevazione dei dati è il momento iniziale nel cui ambito è opportuno effettuare: una scelta dei dati da raccogliere, in funzione della rilevanza attribuita ai fenomeni oggetto di osservazione; una classificazione, intesa come l'attività di ordinamento logico di entità in classi; una codificazione, realizzata mediante l'attribuzione di simboli ad eventi o entità.

L'elaborazione è il complesso di operazioni necessarie per trasformare i dati grezzi in informazioni. Il grado di elaborazione e la tipologia di informazione, variano in funzione delle esigenze informative dei destinatari. I livelli decisionali di tipo operativo necessitano di informazioni analitiche, fornite tempestivamente e a cui si ricorre frequentemente ed in misura continuativa; le decisioni operative si basano su flussi informativi inerenti ad attività quali la vendita, la distribuzione, la produzione, l'approvvigionamento.

La trasmissione delle informazioni riguarda la comunicazione del risultato finale del processo informativo all'utente; in tale fase il destinatario può accedere alla banca dati nella quale risiedono le conoscenze in attesa di utilizzo.

I primi investimenti in ambito informatico sono stati effettuati dalle imprese con riguardo ad attività di tipo operativo. L'entità dei volumi di dati da elaborare e la ripetitività delle operazioni di trattamento degli stessi hanno costituito dei validi presupposti per l'automazione delle procedure operative aziendali quali, ad esempio, la fatturazione e la contabilità delle paghe e stipendi. Il campo di utilizzo delle procedure di

trattamento automatizzato dei dati si è poi ampliato comprendendo anche operazioni più complesse quali la gestione degli ordini da clienti, la gestione del magazzino, la gestione dei beni patrimoniali. L'espressione *Electronic Data Processing* (EDP) definisce l'insieme di queste applicazioni.

Successivamente, l'esigenza di programmare e controllare in modo continuativo le attività aziendali ha condotto alla costruzione di strumenti informatici finalizzati a supportare le attività direzionali. L'automazione delle procedure è stata effettuata con riferimento al controllo di gestione ed ai sistemi di *reporting*. I sistemi informativi realizzati in tale ambito vengono indicati con l'espressione *Management Information System* (MIS) e forniscono informazioni tali da agevolare l'assunzione di decisioni programmate, cioè ripetitive e di routine, per le quali esiste una procedura consolidata, volte al conseguimento dei risultati indicati nei programmi di medio e lungo termine.

In seguito, la complessità ed il dinamismo del quadro competitivo hanno imposto l'adozione di sistemi atti a favorire la pianificazione strategica: la definizione degli obiettivi aziendali, la formulazione di politiche di gestione e la redazione di piani operativi sono attività decisionali non strutturate e non di routine, la cui peculiarità consiste nell'assenza di una precisa procedura da eseguire; per rispondere a queste esigenze sono stati realizzati i sistemi di supporto alle decisioni (*Decision Support System*, DDS). L'utente si avvale, in questi casi, delle informazioni fornite dal sistema per interpretare razionalmente ed efficacemente il processo decisionale; il coinvolgimento è molto elevato in quanto si interviene nella definizione dei modelli e delle procedure per la produzione delle informazioni.

L'EDP System si propone di automatizzare l'attività di trattamento dei dati producendo informazioni strutturate per i livelli inferiori dell'organizzazione. Le conoscenze prodotte devono raggiungere il massimo grado di esattezza, oggettività e tempestività.

Il MIS fornisce alle direzioni funzionali o di divisione, informazioni di routine, per realizzare le attività di programmazione e controllo. I requisiti richiesti sono tempestività ed affidabilità.

Il DDS ha come scopo la produzione di informazioni destinate a facilitare decisioni non programmate e non strutturate: viene dunque privilegiato l'aspetto della qualità delle informazioni ottenibili, ammettendo un certo grado di approssimazione delle stesse.

Nella realtà aziendale è difficile stabilire classificazioni così precise dei processi informatici utilizzati; per questo motivo, spesso, si considera il sistema in modo unitario, utilizzando l'espressione *Executive Information System*, EIS, che identifica l'insieme del sistema di *reporting* (MIS) e di quello di supporto delle decisioni (DDS).

Gli strumenti EIS, per esercitare le loro funzioni, devono essere supportati da adeguate architetture informatiche: una valida soluzione può essere fornita da un sistema di tipo distribuito, caratterizzato dall'attribuzione di autonomia ai singoli nuclei operativi, dalla possibilità di condivisione delle risorse e dall'indispensabile attività di coordinamento di tutti gli utenti del sistema. Questa tipologia di architettura trova una vasta attuazione nella *Local Area Network* (LAN), rete locale che collega i componenti del sistema collocati in uno stesso edificio o stabilimento.

2.3 Computer e network

La figura precedente non mostra chiaramente il contributo apportato dalla rete nel fenomeno dell'*Information Technology*. Fino a pochi anni or sono, le unità di business comunicavano tra loro principalmente attraverso l'uso del fax, del telefono e del contatto umano diretto. Alcuni dei sistemi ideati nel passato, per permettere lo scambio di informazioni per via elettronica, attraverso reti di computer, possono essere considerati come una base per l'espansione dell'internet durante la seconda metà degli anni '90.

Gli esempi più noti dell'*Inter Enterprise Information Technology* sono:

1. *Electronic Data Interchange* (EDI): da un punto di vista logico, i sistemi di scambio dei dati dovrebbero essere collegati tra le diverse imprese. Negli ultimi vent'anni, le specifiche dell'EDI, X12 e varie derivazioni, furono discusse e rese standard per permettere l'interscambio di documenti. Nello stesso periodo, nacquero i network a valore aggiunto (VAN), cioè i network privati di terza parte, affittati alle aziende; il difetto maggiore dei VAN era l'elevato costo di installazione e manutenzione, che li resero utilizzabili solo da imprese di grandi dimensioni, quando la potenza di un network consiste proprio nell'ampia base di utilizzatori. In ogni caso, le imprese che riuscirono ad inserire sistemi EDI al proprio interno, automatizzarono con successo parte della propria catena di fornitura, eliminando molto lavoro manuale, che consisteva nell'inserire una gran

quantità di dati, con il rischio di errori; tutto questo portò a delle interazioni più rapide ed accurate con altri agenti.

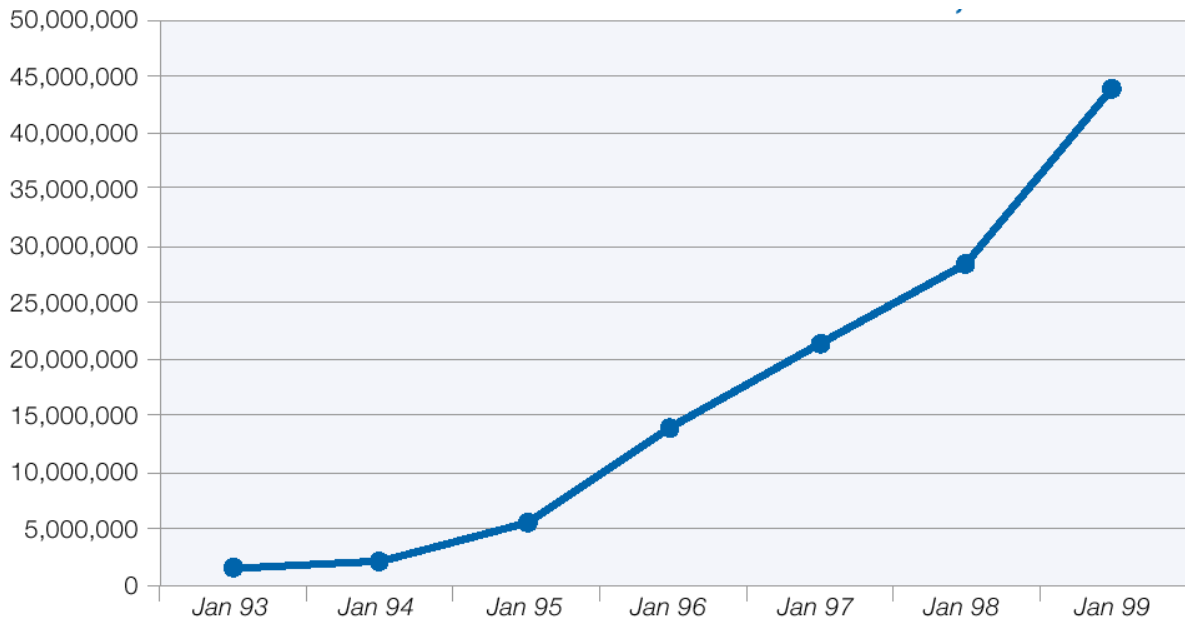
2. *Electronic Funds Transfer (EFT)*: è attualmente il mezzo più utilizzato per il trasferimento dei fondi tra clienti degli istituti finanziari. l'EFT è stato reso possibile dalla nascita di standard per la sicura criptazione delle transazioni.

Fino ad ora, non esiste un meccanismo universale ed assolutamente sicuro per la comunicazione tra imprese, a livello più ampio. In ogni caso, l'EDI e l'EFT costituiscono ottimi esempi dei risultati che si possono ottenere quando le imprese cercano soluzioni per scambiare dati attraverso reti elettroniche. Questi sistemi hanno richiesto la costruzione di network costruiti appositamente per scopi aziendali, che tuttavia sono troppo costosi e specifici per permettere a tutte le imprese di utilizzarli: questa è la maggior motivazione della limitata adozione dei sistemi EDI.

L'internet appare come il sistema migliore per espandere senza grossi costi le interazioni tra imprese, per un rapido scambio di informazioni, coinvolgendo anche le realtà economiche minori, quali le piccole imprese.

Si prevede, nei prossimi dieci anni, un aumento della capacità di banda per le telecomunicazioni di circa mille volte rispetto ai valori attuali, dato eclatante soprattutto quando paragonato alla crescita prevista nella potenza di elaborazione dei computer, quantificata in circa cento volte rispetto ad oggi. La banda disponibile non è solamente alla base del progresso elettronico: se adeguatamente sfruttata essa potrebbe fornire opportunità finora inimmaginabili all'economia.

A questo proposito è utile enunciare un postulato, noto come legge di Metcalfe, dal nome del fondatore di 3Com, che per primo lo ha espresso: "l'utilità di ogni rete di comunicazione (sia essa composta da strade, ferrovie, telefoni, computer), aumenta in proporzione al numero elevato al quadrato degli utilizzatori". Il numero di utilizzatori dell'internet, siano essi privati o aziende, sta crescendo molto rapidamente e prove empiriche suggeriscono che l'ampiezza di banda richiesta dai network principali raddoppia ogni cento giorni. Nella figura seguente, è rappresentata la crescita annuale degli host internet, dal 1993 al 1999:



2.4 Commercio via internet

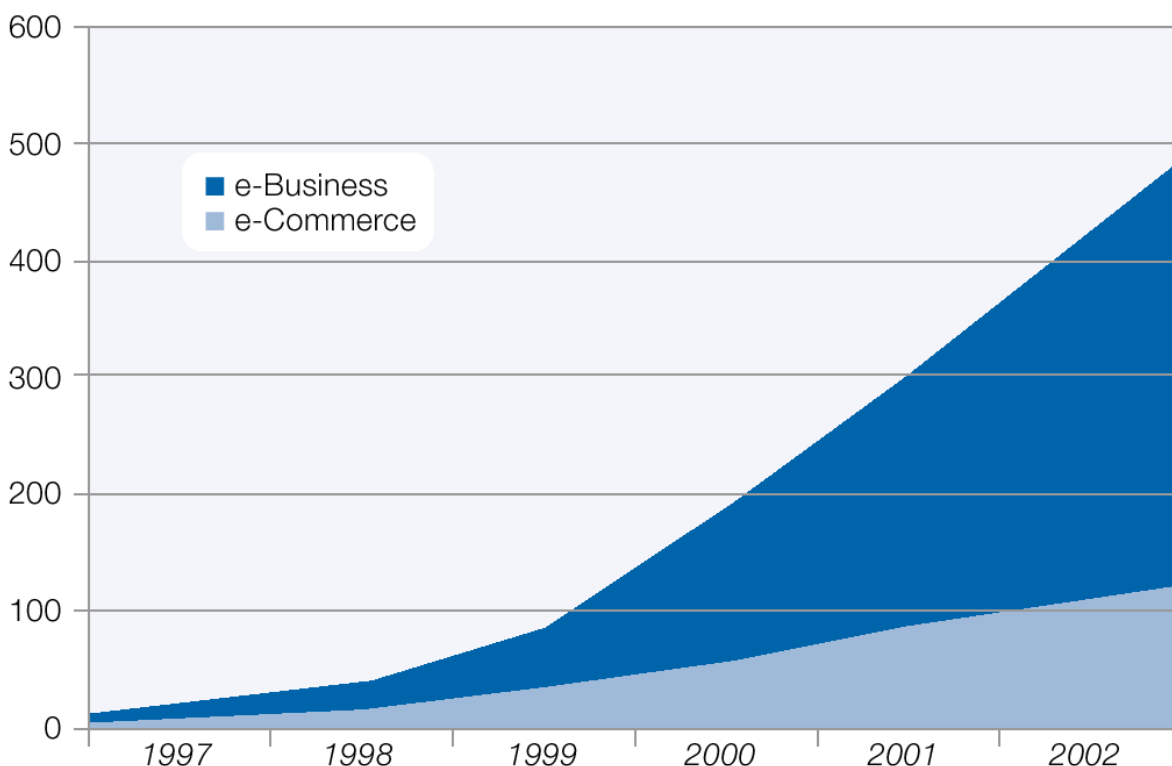
Due sono i principali aspetti legati al mondo degli scambi economici, nati in conseguenza della diffusione dell'internet:

1. *e-commerce*: definito come il vendere e comprare beni e servizi, ed il trasferire somme di denaro, attraverso l'internet
2. *e-business*: definito come l'utilizzare l'internet per processi chiave per il business

Questi due aspetti del commercio elettronico si applicano ad ambiti profondamente differenti e cioè rispettivamente il *Business to Consumer* (B2C) ed il *Business to Business* (B2B). Molte imprese nate a seguito dell'incredibile diffusione dell'internet, sono basate interamente sull'*e-commerce*. Il mercato consumer non è di certo ristretto: una stima del Giga Information Group indica che la quota delle vendite negli Stati Uniti, effettuate attraverso il canale dell'*e-commerce*, arriveranno a 126 miliardi di dollari nel 2002, nonostante le recenti flessioni. Per questo motivo, molti siti web stanno investendo denaro per ampliare la banda disponibile e per rendere sempre più sicuri i sistemi di pagamento.

Un'impresa che si basa esclusivamente sull'internet, può fallire molto facilmente, se non mantiene elevato il livello del servizio offerto.

Il settore B2B è ancor più interessante dal punto di vista delle cifre: nell'anno 2000 le transazioni concluse tramite e-business negli Stati Uniti hanno superato i 205 miliardi di dollari, e le stime parlano di 489 miliardi di dollari nel 2002. Sicuramente questo sarà il mercato principale per il commercio via internet. La figura seguente paragona i due segmenti di mercato ora esaminati.



3.1 Introduzione

L'aumento della competizione, provocato dalla globalizzazione e dalle innovazioni tecnologiche, ha accresciuto la necessità di migliorare l'efficienza con cui le imprese producono e distribuiscono beni e servizi ai clienti. Questo ha portato le aziende di tutto il mondo a cercare nuovi modi di organizzare la forza lavoro, utilizzare le risorse disponibili, interfacciarsi verso i fornitori e organizzare la produzione, in modo tale da risparmiare sui costi e guadagnare competitività nei confronti di un mercato sempre più aggressivo e sempre più orientato verso concetti di *Customer Service*.

Spesso, però, cercare nuove strategie di competizione non è sufficiente e diventa necessario trovare nuove strade, che possano superare le limitazioni di pratiche tradizionali, che portavano a pesanti burocrazie ed impiegati insoddisfatti e non motivati.

Gli sviluppi nel campo dei trasporti, *Information Technology* e telecomunicazioni hanno permesso un crescente livello di scambi sociali tra paesi diversi, attraverso il commercio. Poiché la comunicazione riduce le differenze sociali tra diversi paesi, il mondo è ora più omogeneo ed è dunque possibile condividere beni e servizi, stile di vita, abitudini e anche ideologie.

In effetti, molti cambiamenti strategici che un'impresa deve effettuare sono mirati o a sfruttare l'ambiente più strutturato derivante dalla globalizzazione, o a rispondere a minacce provenienti da concorrenti, le quali, a loro volta, cercano di ottenere dei vantaggi.

Per le imprese leader nel settore prezzo, lo sforzo principale è quello di rafforzare ed ottimizzare i controlli sui costi operativi; poiché l'efficienza e la produttività sono una fonte sempre più importante di vantaggi competitivi, la tendenza è di riscoprire il settore della produzione e dunque di impiegare ingenti somme di denaro per ristrutturare il processo produttivo.

3.2 La Supply Chain: dal passato al futuro

Definiamo *Supply Chain* un processo che permette di creare beni per i clienti finali. La *Supply Chain* parte da materiali grezzi, e passa attraverso la produzione, distribuzione, trasporto, magazzino per giungere alla vendita dei prodotti finiti.

Esistono aziende in cui la *Supply Chain* è interamente presente, ma più spesso essa è suddivisa tra diverse unità aziendali, in modo tale da ottenere specializzazione maggiore e minori costi.

I clienti sanno ormai esattamente quello che desiderano e spesso sono loro a fissarne il prezzo, e dunque le aziende sono costrette ad utilizzare la pianificazione attenta delle catene di fornitura, per anticipare le condizioni del mercato.

Le aziende che si concentrano su produzione e distribuzione, devono pianificare attentamente come raggiungere il cliente nel migliore dei modi, ma anche assicurarsi materie prime da fornitori affidabili, costruire i prodotti con la massima efficienza, e consegnare i prodotti finiti, in accordo con le aspettative dell'acquirente.

Il miglior modo con il quale un'impresa può mantenere la propria competitività consiste nell'assicurarsi di avere le giuste conoscenze ed abilità per utilizzare al massimo le risorse di cui dispone.

Durante i primi anni del Novecento, tutto questo era molto più semplice e lineare: vi era una distinzione molto netta e precisa tra "interno" ed "esterno". Le imprese produttive integrate verticalmente facevano parte dell'interno, e costruivano prodotti per un mercato massificato e facilmente prevedibile, che costituiva l'esterno. Una grossa impresa, in quel periodo, poteva avere una gamma molto ridotta di prodotti, e nonostante questo, avere praticamente la certezza di riuscire a vendere ad ogni potenziale cliente. Quando la Ford iniziò a produrre il famosissimo modello T, il mercato delle automobili era così nuovo ed esotico che ogni acquirente si sentiva unico e privilegiato, nonostante le vetture fossero tutte assolutamente identiche, persino nel colore della carrozzeria (*"Puoi avere il colore che desideri, a patto che si tratti del nero"*).

La componente innovativa del mercato di massa si è però esaurita da tempo: ora molti clienti non sono soddisfatti dei prodotti destinati indistintamente a tutti, anche se spesso se ne trovano sul mercato moltissime varietà e tipologie.

Molti acquirenti desiderano aver la possibilità di definire delle specifiche per i prodotti ai quali sono interessati. Già oggi esistono aziende che permettono ai propri clienti di configurare nei minimi particolari i prodotti, prima di un acquisto, magari attraverso un sito Web. Questo permette anche di ottenere, in tempo reale, il prezzo esatto di quello che acquisteranno, comprese eventuali spese di spedizione.

Tutto ciò porterà ad una variazione delle priorità per i clienti, che non valuteranno più quanto velocemente ed in modo efficiente un'azienda riesca a portare sul mercato un prodotto, ma daranno rilievo alla possibilità di consegnare ogni cosa, dovunque ed in qualsiasi momento.

Sicuramente la *Supply Chain* continuerà ad esistere, ma diverrà un processo sempre più integrato tra imprese, ed altamente specializzato, nonché incentrato sui servizi aggiuntivi.

3.3 Gestione ed organizzazione di una catena di fornitura

La gestione ottimale di una *Supply Chain* è sicuramente un punto cruciale per l'organizzazione aziendale. Non è infatti sufficiente ottimizzare l'efficienza operativa delle varie unità, in quanto diviene critico lo scambio di informazioni, non solo all'interno di una singola azienda, ma anche tra diverse realtà produttive o di distribuzione.

La chiave di successo nelle strategie di business degli anni '90 è stata non soltanto l'evoluzione delle tecnologie produttive e il miglioramento dei meccanismi distributivi, ma soprattutto la crescita del coordinamento tra aziende o tra funzioni aziendali. Quando le persone collaborano, devono in qualche modo comunicare, prendere decisioni, predisporre le risorse al momento giusto e nel posto giusto: i manager, gli addetti al settore commerciale, i clienti, gli intermediari, devono dunque coordinarsi.

Se vogliono rimanere competitive, le organizzazioni industriali non hanno altra scelta se non quella di ridurre i tempi di produzione, migliorare la qualità dei prodotti e ridurre i costi. Tale crescita non può avvenire soltanto in un'unità produttiva isolata, ma deve dipendere in modo critico dalle relazioni che intercorrono tra le organizzazioni.

La struttura industriale, nel tempo, si è evoluta dal livello dell'integrazione a quello della specializzazione. Si può trovare riscontro di questo nella realtà, oltre che nella letteratura aziendale ed economica (Chandler, 1990; Arora, 1997, 1998; Farrel, 1998).

La trasformazione, avvenuta per gradi, da una struttura integrata in grado di realizzare internamente la maggior parte delle attività di produzione, ad una in cui l'interesse dell'impresa si concentra su alcune attività principali, permette di definire il concetto di *core business*, cioè una serie di processi cruciali, che devono assolutamente rimanere di competenza dell'azienda stessa. Ci si potrà dunque rivolgere all'esterno per le attività considerate collaterali, o che comunque non sono gestibili economicamente dall'impresa stessa.

Nella fase di nascita di un mercato, le industrie assumono una struttura integrata, poiché la specializzazione è troppo pericolosa. Quando il mercato entra in fase di maturità, ovvero quando l'economia tende a globalizzarsi, la specializzazione diventa una strada obbligata per le aziende che vogliono rimanere competitive. Nella prima fase, l'impresa opera in un mercato poco strutturato e deve rispondere all'interrogativo *make or buy*, con l'imperativo *make*. Infatti, la struttura della catena di fornitura non è sufficientemente matura e non ci sono alternative nei mercati di approvvigionamento. L'azienda spesso può scegliere tra pochi fornitori di un prodotto o di un servizio e rischia di trovarsi in una situazione di difficoltà, poiché non esistono gli standard qualitativi ed economici ricercati.

Con la nascita di standard di tipo qualitativo, e con sempre più imponente rivoluzione tecnologica, i costi di produzione si abbassano e i vantaggi dell'integrazione diminuiscono progressivamente. In Arora (1997) si mostra attraverso un'analisi statistica che l'emergenza di standard e la nascita di componenti modulari coincide proprio con la crescita del numero delle aziende specializzate.

Si osserva dunque l'effetto, della *divisione del lavoro*, per il quale le imprese tendono a concentrarsi sulle attività svolte con maggior efficienza (Arora, 1997); questo effetto è giustificato da una curva di costi marginale diversa per le singole attività.

Se un produttore conserva all'interno tutte le attività, esso non può concentrarsi su quelle in cui sarebbe più efficiente, in quanto deve tener conto anche delle curve dei costi marginali delle attività meno profittevoli, che diventerebbero piatte attraverso la specializzazione dell'azienda.

Esistono tuttavia anche aspetti che limitano e frenano la specializzazione all'interno di un'impresa; il più significativo è sicuramente costituito dai *costi di transazione*. Si dimostra in Farrel, Monroe, Saloner (1998) che ipotizzando costi di transazione nulli la configurazione dell'industria in caso di specializzazione è più efficiente.

Inoltre, come già accennato, la gestione organizzativa delle *Supply Chain* non può limitarsi a considerare solamente l'azienda, ma anche le relazioni con altre realtà, prime tra tutte l'ambiente esterno, le altre imprese e così via. Molte imprese, strutturate in modo

rigido ed altamente gerarchico fino a poco tempo fa, sono ora unità organizzative indipendenti: senza adeguate metodologie e un'attenta gestione, si possono ottenere risultati inaccettabili o insoddisfacenti, pur disponendo di *business unit* singole molto efficienti. In Lee e Billington (1993), si legge che in passato la gestione delle catene di fornitura si è concentrata soprattutto sull'analisi delle necessità informative dei nodi della catena nei quali vengono prese decisioni critiche.

Lo studio di una *Supply Chain* richiede dunque un'analisi approfondita, a causa dell'evidente complessità e poliedricità dell'argomento; citando Sadeh, Smith e Swaminathan (1998), tale studio può essere condotto su tre livelli distinti e complementari:

1. **La struttura delle catene di fornitura**, che consiste nella ricerca del numero ottimale, della logistica territoriale delle unità produttive e delle strutture di scambio delle informazioni.
2. **L'incertezza delle forniture**. E' molto importante comprendere i fattori chiave che regolano le relazioni tra unità produttive, in quanto la conoscenza di tale modello potenzia la capacità predittiva e conseguentemente la capacità di gestione delle incertezze del mercato.
3. **Le politiche operative**. Ci si deve concentrare sulla dinamica operativa per migliorare l'efficienza nel movimento dei materiali lungo la catena produttiva. Per fare ciò i processi produttivi devono essere armonizzati.

3.4 E-business e nuove tecnologie

Nella seconda metà degli anni '90, l'arrivo di nuove tecnologie, prime tra tutte l'utilizzo dell'Internet su larga scala, unito alla diffusione di connessioni veloci ed efficienti, ha portato grossi cambiamenti in campo economico ed aziendale.

La maggior parte delle aziende attuali posseggono siti *web*, anche se la maggior parte di essi sono decisamente sottoutilizzati. I siti di tipo commerciale, solitamente, permettono ai potenziali acquirenti di effettuare acquisti *on-line*, ma pochi hanno realmente trasformato i processi interni di un'impresa. La maggior parte delle volte, un

ordine effettuato attraverso un sito raggiunge un impiegato addetto alla vendita, il quale svolge funzioni assolutamente identiche a quelle che gli competevano nel business tradizionale.

Questa realtà sta però mutando profondamente: stanno infatti nascendo applicazioni dette di *storefront*, le quali permettono di interfacciare direttamente ed automaticamente le varie aree funzionali di un'impresa. Questo significa sistemi di controllo ed organizzazione di una *Supply Chain* attraverso il Web, con sistemi unificati, che diventano il cuore del business stesso.

Citando Seema Mozaffar (fonte internet), Business System Manager di Unilever Ceylon Limited, i due maggiori benefici dell'uso del Web in una catena di fornitura sono:

1. Un costo ridotto per il possesso di architetture informatiche basate su sistemi PC
2. Un uso ottimizzato dei dati in tutti i sistemi informatici, in un ambiente automatizzato che comprende fornitori, partner commerciali e clienti

Non ci sono però solamente lati positivi: la rete permette di ricevere una quantità di informazioni di proporzioni gigantesche e molte imprese non sono preparate per gestirle e poter separare quelle fondamentali da quelle di scarsa importanza. Per questo motivo è necessario:

1. Definire accuratamente il ruolo dei singoli utilizzatori del sistema informativo, ed identificare le informazioni che aggiungono o creano realmente valore
2. Stabilire e mettere a punto politiche di sicurezza avanzate
3. Stanziare budget per sistemi hardware e software adeguati e per un supporto tecnico specializzato: è necessario che un network sia in grado di trasmettere grandi quantità di dati e di rispondere alle necessità degli utenti
4. Per mantenere una notevole flessibilità, è necessario non effettuare previsione di lungo termine; la connettività è da considerarsi come l'opposto dell'integrazione, per l'accesso alla informazioni all'esterno

5. E' consigliabile impiegare, fin da subito, un'architettura informatica aperta, in grado di accettare espansioni e personalizzazioni
6. Promuovere il sempre maggior utilizzo del Web per rendere possibile il modello di un'impresa virtuale

3.5 Il Business to Business come aspetto cruciale

Il campo in cui le innovazioni tecnologiche degli ultimo anni porteranno i maggiori cambiamenti è sicuramente quello del *Business to Business* (B2B), cioè comunicazione e commercio tra diverse imprese. Questo perché in questo settore è molto più semplice costruire applicazioni software in grado di mettere in comunicazione diverse realtà aziendali, senza l'intervento umano.

Si stima che, nell'emergente scenario basato sul commercio elettronico, per ogni transazione in ambito *Business to Consumer* (B2C), cioè verso l'utente finale, ci saranno da cinquanta a oltre duecento transazioni B2B. Per questo motivo, è assolutamente necessario preparare sistemi di B2B efficienti per poter dialogare al meglio con partner commerciali, attraverso applicazioni informatiche.

Le relazioni commerciali sono dinamiche e cambiano con frequenza: questo richiede che vengano impiegate tecnologie flessibili e che si cerchino collaboratori pronti ad affrontare il cambiamento. Tutto questo si può riassumere in un nuovo concetto, di *Enterprise to Enterprise* (E2E), che costituisce l'evoluzione logica del B2B. Un tipico esempio di E2E è costituito da Amazon.com, noto venditore di libri, CD ed altri prodotti on-line: oltre a fornire un catalogo aggiornato in tempo reale, essa fornisce anche liste di prodotti di diverso tipo, che riflettono le preferenze dei consumatori, secondo le scelte da questi effettuate in passato. Questo servizio, semplice ma efficace, aumenta in modo simultaneo l'attenzione verso il cliente (*customer service*) e permette di vendere più prodotti.

Ovviamente questo costituisce solamente un esempio: in un futuro recente, esisteranno sistemi di E2E, in grado di interfacciarsi direttamente a più imprese. Per esempio, un fornitore potrà fornire un servizio ai propri clienti, che permette in tempo reale di ottenere la lettura dell'inventario clienti. Se certe scorte raggiungono livelli critici, e se quel prodotto è particolarmente richiesto, la nuova fornitura sarà automaticamente autorizzata, dati certi parametri decisi a priori.

Tutto questo permetterà di raggiungere il mercato con eccezionale rapidità, limitando le giacenze di magazzino e riducendo i costi di comunicazione, scorte ed insoddisfazione dei clienti.

3.6 Sistemi multi-agente per lo studio di una Supply Chain

Considerata la complessità di una catena di fornitura, resa ancora più evidente dall'introduzione di nuove tecnologie, risulta chiara l'importanza di uno studio approfondito di questo aspetto aziendale.

Al fine di fornire uno strumento concettuale per supportare l'attività dei manager che ha come oggetto l'analisi della struttura delle catene di fornitura, si propone la strada dei modelli di simulazione basate su agenti. In Lin (1996) si dimostrano le potenzialità della simulazione, utilizzando un modello fondato su un *sistema multi-agente* (MAS). Tale modello descrive il processo di evasione degli ordinativi in una catena di fornitura e dimostra che i sistemi multi-agente rivelano una buona capacità analitica, perché sono in grado di far emergere quei processi di creazione e di diffusione della conoscenza che nascono dall'interazione dei protagonisti del sistema e la cui comprensione è assai complessa.

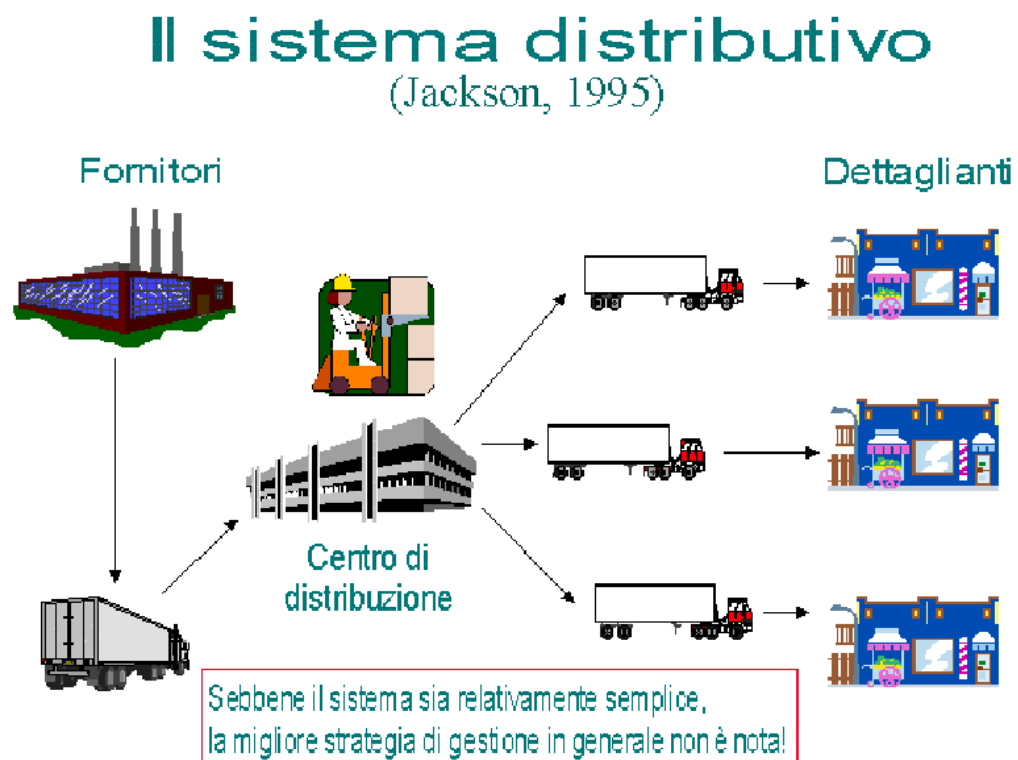
Le ricerche tradizionali, basate su sistemi di equazioni o metodi statistici, erano utili a simulare sistemi produttivi molto gerarchici, in cui le decisioni erano centralizzate. Una catena di fornitura, caratterizzata da bassa controllabilità, è più appropriatamente modellabile alla stregua di una simulazione sociale. Il supporto teorico a questa metodologia di indagine sarà presentato nei capitoli successivi.

Il più arduo problema da affrontare nell'attività di gestione strategica di una catena produttiva è la individuazione e l'opportuna gestione delle incertezze nella prevedibilità dei fenomeni che caratterizzano il sistema. L'imperativo del *just in time*, sempre più pressante per controllare i costi ed assicurare fluidità negli approvvigionamenti, si scontra spesso con la necessità di produrre con anticipo merci che i clienti richiederanno. La produzione dei beni in quantità e tempistiche corrispondenti alle richieste del pubblico è frutto di una stretta armonia tra tutti gli elementi critici della linea di fornitura.

3.7 Struttura di una Supply Chain tradizionale

Possiamo interpretare una tipica *Supply Chain* come una rete, più o meno estesa, attraverso la quale operano diversi agenti eterogenei; l'interazione tra di essi origina un flusso bi-direzionale di materie prime, informazioni, prodotti, servizi e pagamenti.

Nella figura seguente è schematizzata una *Supply Chain* di tipo tradizionale: dai fornitori, attraverso aziende di trasporto, si raggiunge un centro di distribuzione, dal quale partono le spedizioni verso i dettaglianti. Ovviamente possono esserci molti più livelli, ma comunque lo schema di massima è questo:



La definizione di catena di fornitura usata per indicare un sistema produttivo distribuito, fornisce un'interpretazione poco attuale. Quando si parla di catena produttiva, le si attribuisce un significato di linea lungo la quale si snoda il processo di creazione del valore. Si pensa ad un percorso obbligato, attraverso il quale fluiscono informazioni e beni. Ciò è valido soltanto se si considera la sequenza di stati che il prodotto o il servizio assumono.

Ma se si sposta il punto di vista, emerge chiaramente che esiste una fitta rete di relazioni che legano gli agenti del sistema e quelli che venivano definiti anelli della catena diventano inevitabilmente nodi di una rete.

La definizione parla, inoltre, di fornitura. In realtà l'elemento che giustifica la nascita e il funzionamento di una catena di produzione è la domanda dei consumatori. Si sposta dunque il punto di vista da un sistema orientato prettamente alla fornitura, verso una struttura disegnata intorno agli obiettivi di soddisfacimento della domanda, in termini di diversificazione, tempistiche di consegna, prezzo e qualità. Tutti questi elementi dipendono fortemente dal modo in cui si disegnano le relazioni tra gli attori della rete di produzione.

Uno dei maggiori limiti, per quanto concerne la gestione di una *Supply Chain*, consiste nei difetti di comunicazione tra le varie imprese, e anche tra le diverse aree di una medesima azienda. Tradizionalmente, infatti, le organizzazioni che si occupano di marketing, vendite, distribuzione, produzione e pianificazione operano in modo indipendente all'interno della rete produttiva. Gli obiettivi di queste diverse realtà sono spesso definiti senza tener conto del quadro completo e spesso sono in conflitto tra loro: questo fa sì che non esista una strategia unificata ed integrata per l'organizzazione, ma piuttosto una serie di *business plan* e programmi diversi.

Uno dei primi obiettivi è dunque quello di raggiungere un'integrazione tra i vari aspetti della stessa realtà. Secondo Cooper ed Ellram (1993), una *Supply Chain* deve essere gestita come se si trattasse di una squadra di staffetta affiatata; è necessario predisporre il tutto affinché esista uno scambio di informazioni ed una coordinazione perfetta tra gli attori.

In Ganesan, Harrison (1995) si indicano le quattro categorie di attività nelle quali devono essere prese le decisioni ed in ognuna esistono contemporaneamente elementi strategici ed operativi:

1. **Le decisioni logistiche.** Il posizionamento geografico delle unità produttive, dei magazzini, dei punti di approvvigionamento sono il primo naturale passo nella creazione di una catena produttiva. Esse rappresentano il primo livello strategico che consiste nell'ottimizzare l'accesso ai mercati di sbocco ed ha un considerevole impatto sui costi e sul livello dei servizi.
2. **Le decisioni di produzione.** Esse includono la scelta di quali beni produrre ed in quali stabilimenti. Il punto critico è la capacità di produrre i semilavorati o la necessità di ricorrere a fornitori esteri e ciò dipende dal grado di integrazione verticale. Le decisioni comportano l'elaborazione di un documento unico che stabilisce i tempi e le date obiettivo. Inoltre, deve

essere considerato il problema di bilanciare il carico delle unità di produzione e la possibilità di tenere sotto controllo la qualità.

3. **Le decisioni di stoccaggio.** Un elemento molto importante è la collocazione ed il livello medio di utilizzo dei magazzini, elemento sviluppato nel modello JVE dall'autore della Tesi. Esse rappresentano il trade-off tra la necessità di avere un meccanismo per fronteggiare la variabilità della domanda e la necessità di controllare i costi di magazzino. Sono determinanti perché influenzano il livello qualitativo del servizio al consumatore.
4. **Le decisioni sulla distribuzione.** Esse sono strettamente correlate alle decisioni relative allo stoccaggio. Dipendono dal livello dei costi dei mezzi che assicurano lo spostamento delle merci.

Da quanto espresso sino ad ora, appare chiara la difficoltà di gestire una realtà così complessa e mutevole, rimasta per lungo tempo ai margini dell'attenzione dei manager. Solo negli anni '90, infatti, le strategie aziendali hanno cominciato a focalizzarsi sulla gestione delle *Supply Chain*, e ci si è dunque trovati a dover rimuovere le vistose inefficienze dovute ad anni di funzionamento non ottimizzato.

Oltre alla difficoltà oggettiva nell'unificare i dati e gestire una realtà complessa, vi sono altri problemi reali, evidenziati da Billington, Lee (1992), riassumibili nei seguenti punti:

1. **Mancanza di indicatori di monitoraggio delle performance.** Quando le operazioni vengono svolte da unità separate tra loro, la capacità di aggregare le informazioni diminuiscono. I costi relativi alle informazioni aumentano proporzionalmente.
2. **Informazioni relative alle spedizioni inadeguate.** Il sistema di distribuzione rende disponibili ai clienti soltanto una parte delle informazioni relative alle spedizioni. Ciò frena le potenzialità organizzative dei nodi da cui dipende la distribuzione, i quali potrebbero programmare con maggiore precisione le operazioni, se conoscessero la posizione delle merci in tempo reale.
3. **Database non collegati.** Le informazioni spesso sono incompatibili tra loro, perché la struttura in cui sono memorizzate non le rende omogenee e quindi

trattabili automaticamente. Ciò implica l'intervento umano di interpretazione e trascrizione delle informazioni da un sistema all'altro, con inevitabile rallentamento ed introduzione di errori.

4. **Discriminazione verso i clienti interni alla catena.** Ogni unità della catena deve essere considerata allo stesso modo del consumatore finale. Spesso gli agenti del sistema danno la precedenza alla fornitura dei consumatori finali, rispetto a quelli intermedi. Se si formalizzano i rapporti, il sistema diventa maggiormente misurabile e coordinato.

5. **Il progetto di prodotto-processo non tiene conto della catena di fornitura.** La catena di fornitura oggi diventa elemento strategico della gestione produttiva, pertanto la progettazione dei prodotti, dei servizi e dei processi produttivi deve essere realizzata in modo da tener presente dei limiti della catena produttiva, ma anche dei vantaggi competitivi che essa può generare.

6. **Scarsa analisi degli effetti di una catena sui costi.** Risulta molto più agevole monitorare i costi fissi di una struttura o quelli variabili di un singolo processo, rispetto alla valutazione dei costi che si generano nell'interazione tra le diverse organizzazioni produttive. Tali costi risultano essere importanti, sebbene scarsamente monitorati.

7. **Catene incomplete.** Gli effetti di un anello della catena produttiva, genera effetti indiretti su tutto il sistema. Spesso il disegno e l'analisi delle relazioni tra aziende non tiene conto dei clienti di un cliente, piuttosto che dei fornitori di un fornitore. Si tende a controllare le variabili degli elementi esterni limitrofi alla propria realtà.

3.8 La gestione del flusso informativo

Considerata la complessità di una *Supply Chain* tradizionale, e le problematiche portate alla luce dalla carenza di dati ed indicatori sull'efficienza della stessa, risulta chiaro come sia fondamentale, per una gestione ottimale, un attento sistema informativo, in grado di assicurare lo scambio delle informazioni cruciali all'interno di un'azienda, e verso

l'ambiente esterno. Sicuramente, i sistemi informatici che da qualche anno si stanno diffondendo, rendono più semplice e meno arduo questo compito. Per questo motivo, l'*Information Technology* (IT) è sempre maggiormente considerata la strada da percorrere per organizzare al meglio una catena di fornitura.

Per poter efficacemente analizzare la dinamica dell'evoluzione strutturale di aziende che utilizzano l'IT, è utile prendere in esame la scelta di strategie specializzate. Le aziende possono infatti essere specializzate (concentrate, cioè, solamente su un core business) oppure integrate su più livelli. Anche l'ambiente in cui le imprese operano può essere di tipo *chiuso*, con poche aziende, strettamente correlate tra loro ed integrate all'interno, oppure *aperto*, con tantissime aziende specializzate.

Il grado di integrazione è una variabile fondamentale per la scelta del tipo di modello di IT da utilizzare; in questo settore possiamo distinguere le *business unit* dotate di un sistema informativo tradizionale e quelle innovative, dotate di *digital interactive services* (DIS). I sistemi di tipo DIS sono fondamentalmente un'evoluzione concettuale dei precedenti, basati su architetture proprietarie e rigide, difficilmente espandibili e quasi mai compatibili tra loro. Ora, grazie alle tecnologie Internet, Intranet ed Extranet, è possibile costruire sistemi snelli e aperti, che possono dialogare grazie alla rete mantenendo così ampia compatibilità ed espandibilità.

Un sistema informativo di tipo DIS è essenziale nel caso in cui l'azienda operi in una catena di fornitura fortemente decentrata e specializzata, ma questa scelta è anche influenzata dal grado di informatizzazione degli altri agenti del mercato. La tecnologia deve essere ovviamente compatibile e portatile. Un sistema informativo così modulare comporta costi molto elevati, poiché necessita frequentemente di un parziale ridisegno dei processi aziendali in senso più formalmente strutturato e non tutte le aziende ritengono tale scelta tanto strategica da giustificare i costi. Tradizionalmente i sistemi informativi monolitici si possono plasmare con maggiore semplicità di un sistema che impone l'interfacciamento con gli altri sistemi.

I servizi denominati DIS vengono realizzati attraverso lo scambio elettronico di informazioni, denominato EDI (*electronic data interchange*). Si può definire come EDI, qualsiasi sistema informatico che permetta di realizzare le transazioni elettroniche. Esso prevede l'invio diretto delle informazioni da un computer all'altro senza l'intervento umano: per questo motivo i dati devono essere strutturati secondo formati predefiniti e universalmente riconosciuti.

3.9 Il just in time e i sistemi EDI

Il principio del *just in time* è quello di diminuire al massimo le fonti di sprechi ed inefficienze del sistema produttivo, ricevendo una giusta quantità di materie prime e producendo l'ammontare richiesto di prodotti da consegnare in maniera e tempo ottimale al cliente finale.

Il JIT non è un concetto nuovo: esso infatti è un metodo di management nato negli anni '70 in Giappone, adottato inizialmente dalla Toyota; lo scopo principale di quel periodo era incontrare la domanda dei clienti.

Con l'evoluzione dell'ambiente competitivo, gli scopi principali del JIT si sono ampliati. Nonostante dunque il JIT non sia un concetto nuovo, è nuovamente molto importante, grazie alle nuove tecnologie che rendono possibile uno scambio in tempo reale di dati ed informazioni.

Attualmente i sistemi EDI sono considerati tra i metodi più efficienti per la soluzione dei problemi di gestione degli ordini per la maggior parte dei venditori, distributori e produttori, anche se sono in via di superamento.

EDI è un sistema costruito in modo tale da permettere a diversi agenti di scambiarsi dati in un formato comune. Esistono due organizzazioni che controllano il formato dei dati scambiati: la più importante è la ASC X12, che controlla il commercio nel Nord America.

L'EDI, come concetto, nasce nel lontano 1948, quando questo sistema venne usato per scambiare messaggi all'interno delle truppe militari. L'EDI è fondamentalmente una forma elettronica per i documenti standard, utilizzati quotidianamente dalle varie imprese.

Nelle comunicazioni interaziendali tradizionali, i clienti inviano un ordine di acquisto ad un venditore; questo ordine può essere inviato tramite posta, fax o trasmesso telefonicamente. Il venditore legge l'ordine e basandosi sull'inventario, lo invia al magazzino; una volta spedito, viene creata la ricevuta commerciale, che viene spedita all'acquirente, il quale comunica di aver ricevuto il prodotto.

Un sistema EDI permette di scrivere una volta sola, su un computer, le informazioni e i dati dell'ordine. In questo modo si eliminano i ritardi e gli sprechi di tempo e si evitano errori: per esempio, con un catalogo elettronico, è sufficiente un'interfaccia compatibile con il mouse per selezionare il prodotto desiderato.

Da quanto detto sinora, si deduce quanto un sistema informatico integrato possa ottimizzare i tempi, permettendo di raggiungere obiettivi di JIT impensabili fino all'inizio degli anni '90.

Questi effetti sono evidenti quando l'information technology è usata non solo per aumentare la velocità delle comunicazioni, ma soprattutto per cambiare i processi che creano e usano le informazioni. L'informatica genera, di fatto, nuove opportunità di ridisegnare le catene di fornitura.

Sulla base di queste considerazioni emerge l'importanza di utilizzare tecnologie standard, per dare vita ad una infrastruttura informativa che permetta lo sviluppo futuro dell'integrazione di unità specializzate e indipendenti, che migliori l'efficienza della rete produttiva. La soluzione che sembra oggi più promettente per realizzare tale obiettivo è la tecnologia dell'internet e dei browsers.

L'internet è un affascinante esempio di rete globale delle informazioni, nato come strumento di connessione tra piattaforme e tecnologie profondamente diverse tra loro, che possono finalmente condividere informazioni.

L'internet si è inizialmente sviluppato attraverso la condivisione di progetti di ricerca da parte di studiosi che collaboravano da punti geografici distinti e spesso molto lontani tra loro. Data la somiglianza di una catena di fornitura ad un progetto di ricerca, per quanto concerne la diversificazione dei compiti tra gli agenti e la loro distanza geografica, il web si presta ad essere un ottimo strumento per la crescita e la gestione delle catene produttive e distributive.

Come già detto, il paradigma dell'EDI è, in realtà, diffuso da molto più tempo dell'affermazione dell'internet.

Nel passato, i sistemi informativi venivano realizzati in modo da poter comunicare tra loro sulla base di tecnologie proprietarie: la comunicazione era possibile perché la struttura multi-azienda era progettata preventivamente e l'integrazione informatica di sistemi già installati risultava un problema legato alla conversione dei sistemi preesistenti verso piattaforme compatibili. Con l'avvento dell'internet, la tecnologia ha finalmente aperto la strada alla compiuta realizzazione dello scambio elettronico di informazioni. Qualunque realtà preesistente può essere collegata ad altre con semplici interventi di modifica e customizzazione. Ogni azienda può essere integrata in una catena di transito elettronico delle informazioni, pur non possedendo un sistema informativo standard, ma soprattutto non diventa più necessario prevedere l'infrastruttura informativa in fase di progetto iniziale.

Il problema più significativo nell'approdo delle catene di fornitura sull'internet è senza dubbio il problema della sicurezza delle informazioni: spesso le transazioni richiedono il trasporto di informazioni delicate, che le aziende non sono disposte a mettere a disposizione di un pubblico indefinito. Per superare tale barriera si ricorre spesso alla

realizzazione di una sotto-rete, che sfrutta l'infrastruttura della grande rete telematica, ma permette l'accesso ad un numero limitato ed identificato di utenti, le cosiddette intranet.

Tale infrastruttura informativa sta producendo lo sviluppo di una nuova frontiera nella distribuzione: l'e-commerce. Quella che viene oggi chiamata con l'infelice e poco condivisibile termine di new economy, consiste proprio nella compiuta realizzazione dell'EDI, il quale consente la nascita di imprese, la cui attività è completamente orientata allo sfruttamento delle nuove tecnologie. Le imprese tendono a smaterializzarsi, orientandosi all'uso della rete come core business e propongono prodotti e servizi direttamente al cliente secondo il paradigma del business-to-consumer, ma soprattutto stanno sempre più automatizzando le relazioni nei confronti dei propri diretti interlocutori nella catena distributiva. Si affidano, cioè, sempre più al business-to-business, realizzando ciò che è stato teorizzato dai modelli multi-agente sulle catene di fornitura e arrivando al concetto di Enterprise-to-Enterprise già descritto.

La strategia che è risultata essere vincente nel condurre le aziende verso l'e-business, si è basata su un totale ridisegno degli obiettivi aziendali. Da un lato si è mantenuta la struttura preesistente che garantiva un flusso di capitali e una posizione di mercato consolidati, dall'altro il top-management ha riprogettato l'azienda, costituendone una nuova struttura smaterializzata e completamente orientata alle opportunità offerte dal nuovo sistema di comunicazione.

LOGISTICA E GESTIONE DELLE SCORTE

4.1 Evoluzione della logistica aziendale

L'esordio della funzione logistica è avvenuto nella conduzione delle funzioni di magazzino e dei trasporti; questo è ancora vero per le aziende di dimensione minore. In questo ambito, si cerca di effettuare la gestione del materiale che l'azienda acquista, vende o semplicemente utilizza, con particolare riguardo sia per la collocazione fisica delle diverse partite, sia per il controllo delle giacenze e la eventuale ordinazione delle quantità necessarie, sia per la conservazione merceologica dei materiali immagazzinati e per il loro trasporto da e per l'azienda.

In questo caso, il sottosistema magazzino può assumere, in una azienda di piccole dimensioni o prevalentemente artigiana, una importanza marginale, mentre è maggiore nelle imprese di carattere commerciale, nelle quali da una buona gestione della funzione dipende gran parte del successo dell'impresa.

Con il crescere delle dimensioni e l'articolarsi del processo produttivo i flussi di materiali che interessano la logistica si fanno più complessi: ogni flusso è caratterizzato da distinte problematiche e particolarmente complessa è la gestione della tempistica delle diverse operazioni.

Nasce quindi l'esigenza di un controllo attuato non più "a vista", ma secondo criteri programmati e che, almeno prima che si instaurino routine soddisfacenti, frutto anche di aggiustamenti derivanti dall'esperienza effettuata sul campo, possono essere adeguatamente analizzati attraverso l'impiego di strumenti rivolti al calcolo dei lotti di riordino o di produzione.

La necessità di tenere sotto controllo non solo tipologie diverse di flusso, ma articoli di magazzino abbastanza numerosi, emerge anche dalla difficoltà di seguire tutti questi andamenti contemporaneamente e, pur con l'impiego di sistemi informatici all'altezza della situazione, di concentrare la propria attenzione su alcuni aspetti più importanti riguardo interventi, scelte e metodologie d'azione.

Una prima selezione delle problematiche segue la logica della focalizzazione della gestione logistica sulle classi di fenomeni più significative attraverso una distinzione in tre gruppi, aventi una diversa rilevanza nei confronti del processo di gestione.

Per raggiungere tali scopi, nel caso delle scorte di materiali, si utilizza il Diagramma di Pareto nel quale in ascissa sono riportati, in percentuale sul totale, la numerosità delle classi di oggetti da gestire e, in ordinata, i corrispondenti valori di accumulo, in percentuale sul valore totale dei materiali (Bellandi, 1991).

In questa maniera è possibile distinguere tre classi fondamentali:

1. **Classe A:** I materiali che, in numero limitato, coprono la maggior parte del consumo totale
2. **Classe B:** I materiali che in numerosità media contribuiscono a livelli medi di consumo totale
3. **Classe C:** I materiali che, in numero maggiore, contribuiscono alla minor percentuale del consumo totale

Il controllo delle scorte, che può avvenire secondo diversi criteri, ha lo scopo di evitare che la scorta si esaurisca nel tempo che intercorre prima che l'ordine di acquisto, o produzione (*lead time*), venga soddisfatto, determinando, in caso si tratti di scorte di materie prime o semilavorati, difficoltà o interruzioni della produzione o, in caso di prodotti finiti, ritardi della consegna ai clienti.

In generale, la gestione della produzione può avvenire:

1. sulla base di gestioni basate sul livello delle scorte, applicando un sistema di controllo che permetta di ricostituire le scorte prima che queste si esauriscano
2. sulla base dei futuri fabbisogni

D'altra parte i sistemi di controllo possono seguire due strade diverse:

1. prendere come base un lotto di riordino costante che viene calcolato in base al costo ottimale di gestione delle scorte;
2. prendere come base intervalli di tempo costante, sempre al fine di minimizzare i tempi di giacenza

Si tratta comunque di sistemi prevalentemente orientati a risolvere problemi della produzione con particolare attenzione alle richieste del mercato.

In questi termini la funzione della logistica appare comunque passiva e prevalentemente tecnica, come dimostrano i diversi modelli e la differenziata casistica che essi intendono affrontare.

Un notevole progresso nello studio di questa problematica è avvenuto attraverso l'affinamento degli studi di ricerca operativa e l'applicazione delle scienze matematiche al *problem solving* relativo alle reti di produzione.

4.2 Le scorte nella realtà aziendale

Consideriamo un sistema molto semplice, con un magazzino contenente prodotti finiti, alimentato da un sistema produttivo rappresentato da una scatola nera, nel senso che rappresenta un'entità in grado di soddisfare, con un certo ritardo, ordini di riempimento del magazzino, non distinguendo ancora tra ordini di acquisto (in cui la scatola è un fornitore) o di produzione.

Il livello di magazzino “W” scende nel tempo, con un tasso che dipende dalla frequenza e dall'entità degli ordini da parte dei clienti. Un problema di controllo delle scorte richiede di stabilire *quando* occorre lanciare un ordine e *quanto* occorre ordinare. Un possibile approccio consiste di determinare un punto di riordino R: quando il livello del magazzino scende sotto questo livello prestabilito, si lancia un ordine per un quantitativo Q; dopo un intervallo di tempo (*lead time*), il lotto prodotto o acquistato viene versato a magazzino.

Per determinare i parametri R e Q occorre tener conto di esigenze spesso contrastanti: da una parte, sarebbe opportuno ordinare piccoli quantitativi di prodotto con una frequenza abbastanza alta, in modo da tenere il livello del magazzino più basso possibile, minimizzando i costi fissi, per lo più quelli di giacenza. Dall'altra, occorre evitare di lanciare ordini con frequenza eccessiva, a causa dei costi di riordino o, nel caso di ordini di produzione, alla necessità di attrezzare le macchine. Si tratta dunque di trovare il miglior compromesso tra queste esigenze, considerando anche fattori quali la variabilità della domanda ed il *lead time*.

Esistono fondamentalmente tre tipi di scorte:

1. scorte di materie prime
2. scorte di semilavorati

3. scorte di prodotti finiti

Le materie prime sono da intendersi tali dal punto di vista dell'azienda che si studia: possono infatti essere anche dai prodotti elaborati e di una certa complessità, che però costituiscono la base della produzione aziendale e vengono acquistati come punto di partenza.

Occorre controllare i flussi di materiali e le giacenze a questi tre livelli, in modo da trovare compromessi ottimali alle esigenze contrastanti di cui si diceva prima: il livello delle scorte deve essere tale da assicurare un buon flusso di materiali, evitando di bloccare o rallentare la produzione. Un livello di scorte eccessivo, d'altra parte, comporterebbe costi elevati di giacenza e un aumento del tempo di attraversamento del sistema.

Questo, definito come *flow time*, è il tempo intercorrente tra l'ingresso delle materie prime nel sistema di produzione e l'uscita dei prodotti finiti. Ovviamente, una perfetta sincronizzazione tra le diverse fasi di gestione delle scorte, diventa cruciale in aziende che si occupano prevalentemente di assemblaggio.

4.3 Gestione delle scorte: funzioni ed obiettivi

La gestione delle scorte non è sicuramente un argomento recente: esso ha infatti una lunga tradizione aziendale di oltre settant'anni. Recentemente, con il fiorire di concetti quali il *Just in Time* e l'azzeramento dei livelli di magazzino, le teorie alla base della gestione delle scorte sono profondamente mutate ed evolute, integrandosi spesso con la realtà informatica degli anni '90.

Oltre alla classificazione delle scorte in materie prime, semilavorati e prodotti finiti, si trova in Brandimarte, Villa (1995) un ulteriore criterio di classificazione:

1. **Cycle stock:** La funzione di questo tipo di scorta è di raccordare il processo di riempimento del magazzino con il processo di domanda. Il primo è caratterizzato dall'arrivo infrequente di lotti relativamente grandi; il secondo da richieste frequenti e per piccole quantità. La presenza di costi e tempi di setup, o costi fissi di ordinazione, rende necessaria la creazione di scorte temporanee.
2. **Safety stock:** Si introduce un livello di scorta di sicurezza, per far fronte a ritardi di consegna, guasti nel sistema produttivo, picchi di domanda.

3. **Seasonal stock:** Nel caso di prodotti con domanda stagionale, può essere necessario costruire una sorta nel periodo di scarsa domanda, per poter soddisfare le richieste nel periodo in cui la domanda è alta.
4. **Pipeline stock:** In un sistema produttivo multi-stadio (che cioè opera su materie prime, semilavorati e prodotti finiti), vi sono dei magazzini di semilavorati che disaccoppiano parzialmente gli stadi; a volte gli stadi sono geograficamente separati, nel senso che si tratta di stabilimenti diversi. In questo caso divengono cruciali aspetti logistici legati al trasporto. non è ovviamente possibile trasportare materiali in modo sincrono con la lavorazione; l'uscita di uno stadio si accumula prima di esser trasferita allo stadio successivo. Maggiore è la distanza tra gli stadi, maggiore è il livello di scorte di questo tipo.

In una situazione reale, spesso ci si trova a lavorare con più tipi di scorte diverse e ciò richiede di operare a più livelli decisionali: la prima decisione concerne cosa tenere in magazzino. Ad un livello più operativo è richiesto il monitoraggio dei versamenti e dei prelievi, della movimentazione dei materiali, del controllo dei codici soggetti a deperimento. Tali operazioni sono normalmente definite *warehousing*, mentre con *inventory control* si intendono le funzioni decisionali relative a quando e quanto ordinare.

Uno dei principali obiettivi del controllo delle scorte è sicuramente quello di ottimizzare i costi ed i tempi della produzione. Esistono diversi elementi di costo da considerare nella gestione delle scorte, così classificabili:

Costi associati al livello del magazzino: dovuti a immobilizzo del capitale, rischio di obsolescenza, affitto di locali, eventuali costi di assicurazione

Costi associati al lancio di ordini, siano essi di produzione o di acquisto: essenzialmente costi fissi, dovuti al trasporto, setup delle macchine, pratiche amministrative e burocratiche

Costi di *shortage*: subentrano nel caso in cui le scorte non sono sufficienti per soddisfare la domanda. Nel caso di domanda per prodotti finiti, tali costi sono principalmente legati alla perdita di immagine e all'insoddisfazione del cliente.

Gli ultimi costi sono sicuramente i meno agevoli da quantificare; per questo motivo, spesso, le aziende fissano un obiettivo di servizio minimo, cioè in un numero di ordini che può sicuramente ed immediatamente essere soddisfatto dal magazzino. Altre volte, le

aziende si auto-multano per un ritardo, trasformando il costo in uno sconto da praticare al cliente per ogni giorno di attesa.

Sicuramente, gestire le scorte, significa trovare il miglior compromesso tra livello di servizio e costi. Un indicatore per misurare i costi di giacenza è il *turnover*, definito come il rapporto tra costo del fabbisogno anno ed il valore medio di magazzino. Il turnover misura quante volte il magazzino viene rinnovato nel periodo di riferimento; il valore ideale di turnover dipende solitamente dal settore merceologico dell'azienda ed è utile misurarlo per misurarsi con aziende simili e per valutare i progressi a seguito di innovazioni nel sistema di controllo delle scorte.

4.4 Sistemi di controllo delle scorte: una classificazione

I sistemi di controllo delle scorte si differenziano fundamentalmente nel modo in cui sono determinati gli istanti di lancio degli ordini e la quantità ordinata.

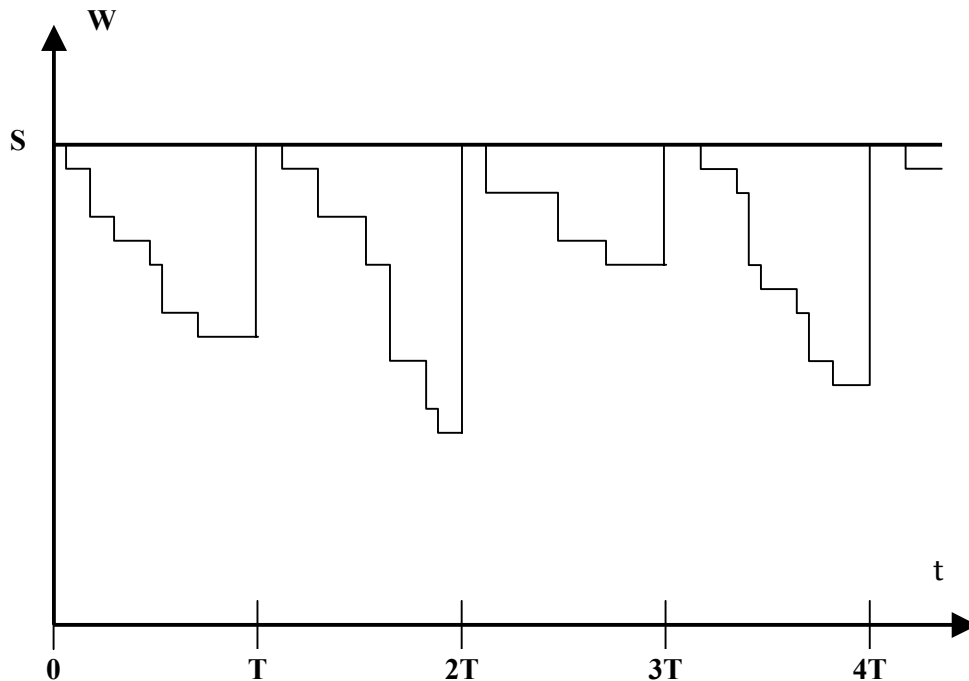
Sistemi periodic review: il livello di magazzino viene esaminato in istanti di tempo prefissati e si decide se e quanto ordinare. Se si adotta questo tipo di sistema, il “quando” ordinare è prefissato a priori, lasciando come unica decisione il “quanto”. Considerando dunque un livello target “S”, un livello “I” di magazzino e un livello “R” di riordino, come leggiamo in Brandimarte, Villa (1995), la quantità “Q” da ordinare è data da:

$$(4.1) \quad Q = \begin{cases} 0 & I > R \\ S - I & I \leq R \end{cases}$$

Il livello di riordino ha la funzione di impedire il lancio di ordini per quantità troppo piccole, che comporterebbero un incremento dei costi fissi. Caso particolare è costituito dalla politica *order-up-to-S*, in cui si pone $S=R$, lanciando dunque un ordine ad ogni ciclo. Il risultato è rappresentato nella pagina seguente.

Sistemi fixed quality: in questo caso l'unità da ordinare “Q” è fissa e si deve determinare l'istante di lancio dell'ordine. Si tratta dunque di una politica di gestione inversa rispetto a quella di *periodic review*: in questo caso il livello dei magazzini è controllato continuamente e si lancia un ordine quando esso scende sotto un punto di riordino “R” (*continuous review*).

Sistemi base stock: si lancia un ordine in corrispondenza ad ogni prelievo dal magazzino, pari alla quantità prelevata.



I modelli sulla base dei quali determinare i parametri delle politiche decisionali sono spesso molto complessi. Principalmente, i criteri che si utilizzano per classificare tali modelli sono i caratteri *statico/dinamico* e *deterministico/stocastico*. I modelli dinamici sono in grado di trattare domande varianti nel tempo, mentre per quelli statici il livello della domanda si assume piuttosto stabile. Le variazioni della domanda nei modelli statici possono essere modellate assumendo che il tasso di domanda sia una variabile aleatoria, cioè caratterizzata in termini probabilistici. Anche per il *lead time* si può utilizzare una variabile di questo tipo.

E' interessante, per le finalità del presente lavoro, analizzare delle tecniche di gestione che possano essere implementate in un modello software basato su agenti, dunque piuttosto basilari, di tipo statico e deterministico.

4.5 Modelli di tipo deterministico: lotto economico

Come si legge in Graves e Kan (1993), i modelli deterministici sono caratterizzati da una domanda il cui tasso “D”, cioè la quantità assorbita nell’unità di tempo, è costante. I sistemi *fixed quantity* e *periodic review* descritti prima sono dunque equivalenti, in questa realtà: il periodo “T” degli ordini e la quantità “Q” sono legati dalla relazione seguente:

$$(4.2) \quad T = \frac{Q}{D}$$

Questa equivalenza è valida per un modello, mentre non lo potrebbe essere nella realtà a causa delle fluttuazioni, ancorché minime, nel tasso di domanda.

Il principale modello di tipo statico deterministico, facilmente inseribile in un software basato su agenti, è quello del *lotto economico*, noto anche come *Economic Order Quantity*. Esso ha lo scopo di determinare una quantità “Q” da ordinare quando il livello di riordino viene raggiunto. I dati del problema sono la domanda media “D” nell’unità di tempo, il costo unitario di magazzino “h” ed il costo per il lancio di un nuovo ordine, indicato con “s”. Ovviamente lo scopo è minimizzare i costi, dati dalla somma di quelli di magazzino e di lancio degli ordini. Alla base di questo tipo di modello, vi sono alcune assunzioni fondamentali, che ne semplificano l’esposizione:

1. la domanda è da considerarsi nota, continua e costante nel tempo
2. ogni prodotto viene considerato indipendentemente dagli altri
3. lo riempimento del magazzino è istantaneo
4. il costo del prodotto non dipende dalla quantità ordinata (non esistono sconti)
5. non si ammettono ritardi di consegna

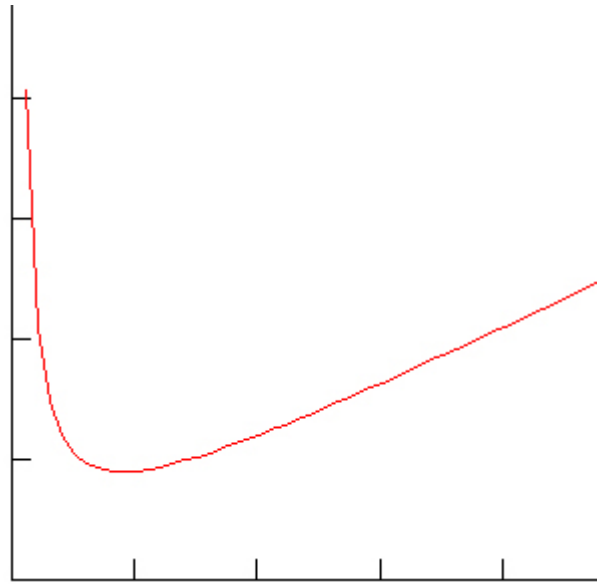
Ogni ciclo di magazzino, cioè ogni periodo compreso tra due ordini, ha una durata Q/D , e dunque il costo di magazzino per un ciclo sarà:

$$(4.3) \quad C_m = \frac{1}{2} Q \frac{Q}{D} h$$

Per ottenere il costo totale, per unità di tempo, sarà necessario sommando l'espressione precedente ad “s”, cioè il costo del lancio di un ordine per un ciclo, e dividere il tutto per la durata del ciclo stesso. Si ottiene:

$$(4.4) \quad C(Q) = \frac{1}{2}Qh + \frac{sD}{Q}$$

Vi è dunque un termine linearmente crescente rispetto a “Q” (il primo) ed uno decrescente. La funzione appare come in figura:

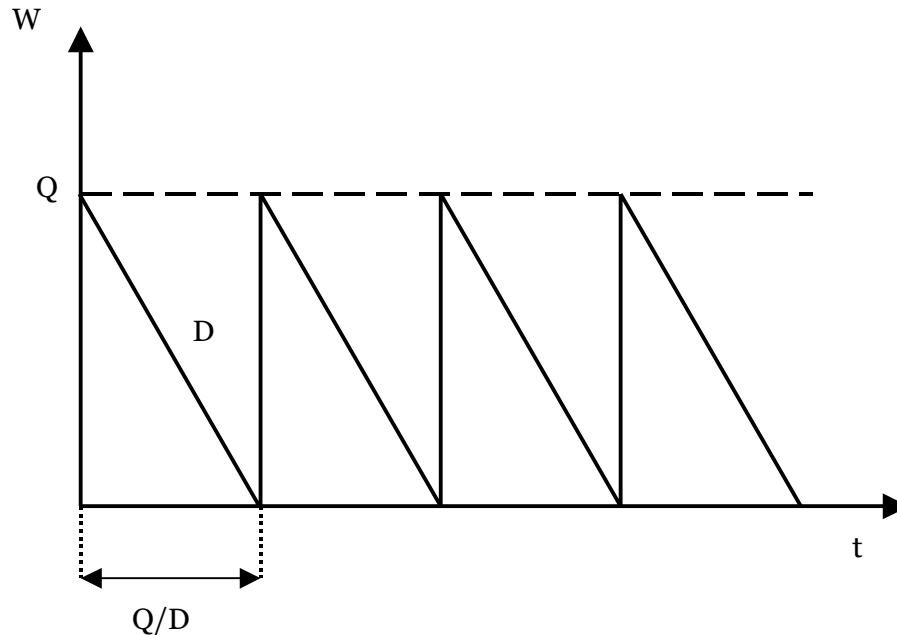


Per ottenere la quantità ottima, è sufficiente calcolare la derivata prima dell'espressione 4.4, e porla uguale a zero. Si otterrà:

$$(4.5) \quad Q^* = \sqrt{\frac{2sD}{h}}$$

Dalla 4.5 si evince che la dimensione del lotto economico aumenta al crescere di “s” e diminuisce al crescere di “h”; in corrispondenza del lotto economico ottimale, il costo di giacenza e il costo fisso si equivalgono. Il costo totale per unità di tempo, sostituendo la 4.5 nella 4.4, sarà semplicemente \sqrt{hsD} .

Il funzionamento del metodo del lotto economico è rappresentato nella figura seguente:



In Brandimarte e Villa (1995) si prendono in esame alcune varianti al modello base del lotto economico descritto prima: ne esistono, ad esempio, in grado di tener conto del costo di *backlog*, cioè quello associato ad un ordine servito in ritardo, che può essere rappresentato da un magazzino negativo, con la duplice possibilità di cliente paziente, che dunque attende l'ordine in ritardo, o impaziente, che ritira l'ordine.

Un'altra variante che può essere presa in considerazione riguarda la possibilità di praticare sconti in base alle quantità ordinate: questo porta sicuramente un'aggiunta verosimile al modello, ma complica notevolmente i calcoli, in quanto obbliga a calcolare il lotto economico ottimale per tutte le eventuali fasce di prezzo. Infine, si può presupporre come finita la velocità di riempimento dei magazzini, nel caso produttore e magazzino siano logisticamente vicini.

4.6 Magazzini nella realtà aziendale

Un magazzino, come ogni parte fisica di un'azienda, richiede lavoro, capitale e sistemi di gestione: tutto questo comporta dei costi. Per la maggior parte delle operazioni

che assolvono, i magazzini sono assolutamente necessari e difficilmente vi si potrà rinunciare. Alcuni usi tipici di un magazzino sono i seguenti:

1. Ridurre i costi di trasporto e migliorare il servizio nei confronti del cliente
2. Realizzare economie di scala nella produzione o nell'acquisto
3. Gestione ottimale dei prodotti stagionali
4. Creare valore aggiunto ai processi, aiutando a creare prodotti sempre più in linea con i bisogni della clientela e differenziati
5. Ridurre il tempo di risposta, quando l'infrastruttura dei trasporti è poco sviluppata o troppo carica

La scelta dell'organizzazione dei flussi di materiali è ampiamente determinata da alcuni fattori, quali:

1. le caratteristiche dell'inventario (numero dei prodotti, velocità di circolazione)
2. requisiti di flusso, per esempio in numero di linee produttive e gli ordini da evadere al giorno
3. la struttura dell'edificio stesso in cui si svolge l'attività

4.7 Magazzini e teoria delle code

Un sistema di code (*queueing system*) è un modello con la seguente struttura: gli agenti arrivano e si mettono in coda, per attendere un servizio offerto da uno o più fornitori. Dopo aver ricevuto ciò per cui attendevano, gli agenti escono dal sistema. Un risultato fondamentale della teoria delle code è noto come legge di Little, dal nome di colui il quale ha formalizzato ciò che prima era solo conoscenza aziendale implicita.

Questo teorema sostiene che, per un sistema di code in stato stazionario, la lunghezza media della coda è uguale al tasso medio di arrivo moltiplicato per il tempo medio di attesa. Esprimendolo con una formula si ha:

$$(4.6) \quad L = \lambda W$$

Un magazzino può essere modellato come un sistema di code nel quale ogni *SKU* (*stock keeping unit*) costituisce un agente che arriva e si aggiunge alla coda, cioè viene

aggiunta al magazzino, in attesa di essere consegnata o utilizzata. Se il magazzino è in stato stazionario, il prodotto verrà consegnato nello stesso tempo medio in cui arriva: applicando la legge di Little, si ottiene che la media di prodotti in magazzino sarà uguale al tasso di arrivo moltiplicato per il tempo medio in cui i componenti rimangono in attesa.

In questo modo si riesce a stimare la richiesta media di forza lavoro per la gestione dell'attività durante un periodo esaminato.

4.8 Legge di Little ampliata

Ciò che rende la legge di Little veramente utile ed importante è il fatto che essa può essere applicata anche quando esistono diversi tipi di agenti, ognuno dei quali presenta parametri diversi per quello che riguarda tasso di arrivo (λ_i), tempo di attesa (W_i) e lunghezza della coda (L_i). La regola può dunque essere applicata ad una *SKU* singola, ad una serie di *SKU*, ad una porzione di un magazzino oppure ad un magazzino intero.

Si divida un periodo di tempo in n intervalli uguali, e si definisca $A(n)$ come il numero totale di arrivi in ognuno di queste unità e T_j come il tempo (assunto come numero intero di periodi) nel sistema del j^{mo} arrivo. Vi sono arrivi solamente all'inizio di ogni periodo. Sia I_i l'inventario del sistema al tempo i e per ora sia $I_0 = I_n = 0$. Se ogni agente deve pagare una lira per periodo, si può calcolare quanto denaro il sistema raccoglie in due modi diversi: ogni agente j paga T_j lire e dunque il risultato sarà $\sum_{j=1}^{A(n)} T_j$. Allo stesso tempo, il sistema raccoglie lire per ogni periodo, quindi il risultato è anche $\sum_{i=1}^n I_i$.

Abbiamo dunque la relazione:

$$(4.7) \quad \sum_{j=1}^{A(n)} T_j = \sum_{i=1}^n I_i$$

o, in modo equivalente:

$$(4.8) \quad \frac{\sum_{i=1}^n I_i}{n} = \left(\frac{\sum_{j=1}^{A(n)} T_j}{A(n)} \right) \left(\frac{A(n)}{n} \right)$$

L'equazione (4.8) può essere interpretata come la legge di little con tasso di arrivo $A(n)/n$ e tempo medio nel sistema pari a $\sum_{j=1}^{A(n)} T_j / A(n)$.

In un sistema realistico, non possiamo ovviamente avere $I_0 = I_n = 0$ e la quantità reale di denaro raccolto dovrà essere corretta, aggiungendo la quantità di denaro raccolto dagli agenti iniziali nel sistema al tempo 0 e sottraendo la quantità di denaro che deve ancora essere raccolto dai nuovi arrivi che non hanno finito il servizio al tempo n . Queste modifiche andranno divise per $A(n)$ nell'equazione (4.8).

4.9 Attività dei magazzini e costi corrispondenti

Solitamente, un magazzino riorganizza e riconfeziona i prodotti, i quali solitamente giungono imballati su larga scala, e ripartono con un diverso package. Una funzione importante del magazzino è dunque quella di dividere grosse partite di prodotti per distribuirli in quantità inferiori. Solitamente, più è ridotta la quantità trattata singolarmente dal magazzino, più i costi relativi sono elevati.

La riorganizzazione dei prodotti si articola nelle seguenti fasi:

Processi in entrata:

1. **Arrivo della merce.** Solitamente si inizia con la notifica di arrivo di prodotti, il che permette al magazzino di liberare risorse e prepararsi al meglio a ricevere il nuovo carico. Quando il prodotto è arrivato, viene scaricato e preparato per l'immagazzinamento. Solitamente viene inserito in un database in modo da poter venire utilizzato e rintracciato rapidamente quando ve ne sarà la necessità. Vi sarà un'ispezione dei prodotti per verificare eventuali difetti o errori nella consegna.

Solitamente la merce arriva imballata in grandi quantità. La ricezione ed il controllo preliminare della merce influiscono per circa il 10% sui costi totali.

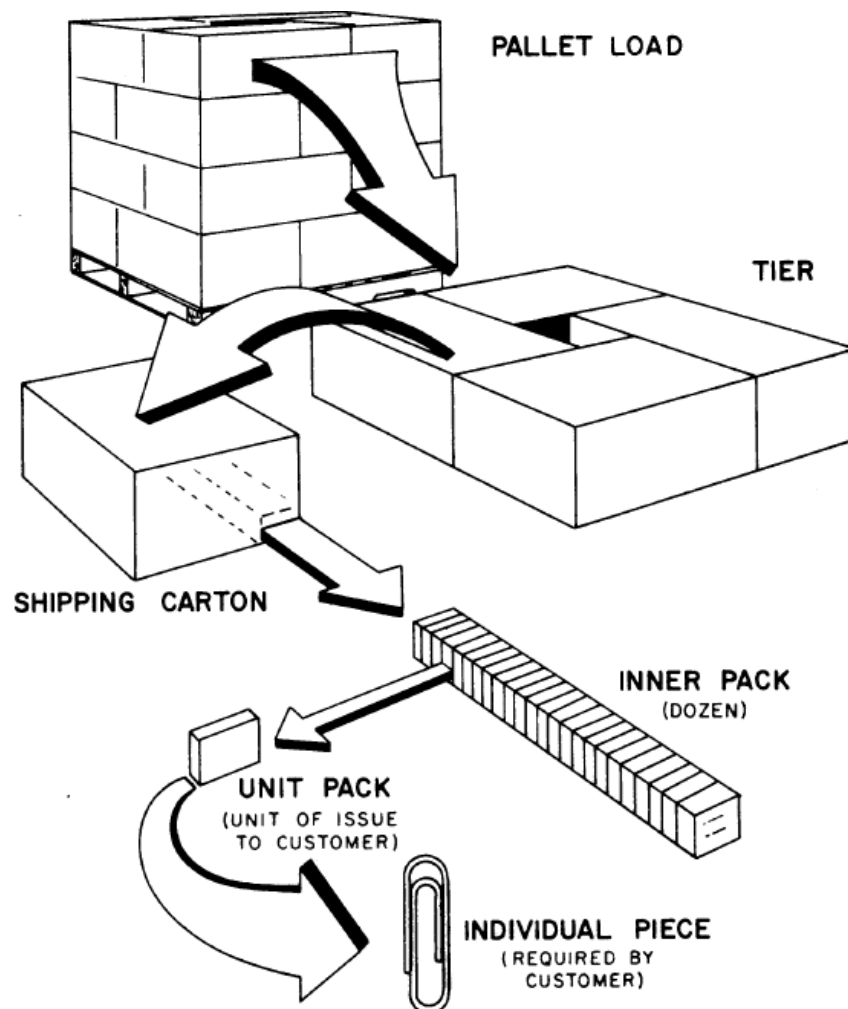
2. **Immagazzinamento.** Prima che la merce possa essere inserita nel magazzino, occorre determinare la miglior disposizione all'interno dello stesso. Questa fase è molto importante, poiché il posizionamento della merce in magazzino determina la rapidità con cui vi si potrà avere accesso e di conseguenza i costi, diretti e indiretti. E' dunque necessario conoscere in ogni momento la disponibilità di zone all'interno del magazzino, la loro grandezza e quanto peso possono supportare. Quando la merce è immagazzinata, si inserisce nel database anche la locazione corrispondente: questa informazione verrà utilizzata in una fase seguente, per costruire liste ottimizzate per prelevare la merce, quando sia richiesta dai clienti o dalle unità produttive. Solitamente la fase di immagazzinamento influisce per circa il 15% sui costi totali del magazzino.

Processi in uscita:

1. **Arrivo degli ordini.** Alla ricezione dell'ordine da parte del cliente, sia esso finale o intermedio, il magazzino deve effettuare controlli per verificare se l'oggetto richiesto è disponibile. Il magazzino deve poi procedere a creare una lista delle richieste da evadere, e infine a produrre la documentazione necessaria per la spedizione. Queste attività sono svolte solitamente da un sistema di gestione dei magazzini, cioè un software che coordina le attività del magazzino stesso.
2. **Evasione degli ordini.** Questo processo solitamente incide per circa 55% dei costi operativi tipici di un magazzino, e può essere a sua volta suddiviso in sottoprocessi quali la ricerca, l'estrazione, la creazione dei documenti e gli eventuali viaggi, nel caso i magazzini non siano vicini alle unità produttive. La maggior parte del processo di evasione degli ordini è volta a cercare di ridurre al massimo il tempo improduttivo, attraverso approcci di ottimizzazione e riduzione dei costi. Un esempio può essere quello di avere il magazzino il più vicino possibile agli impianti, in modo da minimizzare i costi relativi agli spostamenti della merce, che spesso richiedono l'utilizzo di mezzi pesanti.
3. **Controllo ed imballaggio.** L'imballaggio può richiedere un'elevata intensità di lavoro, poiché è necessario trattare separatamente ogni pezzo facente parte dell'ordine del cliente. In questa fase si controlla anche la perfetta corrispondenza della merce con l'ordine ricevuto, e l'accuratezza della fornitura: si tratta sicuramente di una pratica attinente alla *customer satisfaction*, che tuttavia migliora anche l'organizzazione all'interno del magazzino stesso, eliminando gli

sprechi e abbreviando i tempi. Gli ordini errati, infatti, non solo peggiorano la qualità percepita dal cliente, ma generano anche dei resi, che sono spesso costosi da gestire, fino a dieci volte il costo di spedizione della merce. Una complicazione attinente all'imballaggio è costituita dal fatto che spesso i clienti preferiscono ricevere la merce in pochi contenitori, in quanto questo riduce i costi che devono sostenere e facilita la gestione degli oggetti ricevuti. E' necessario dunque sincronizzare, all'interno del magazzino, l'invio al reparto imballaggio di tutti i prodotti facenti parte di uno stesso ordine.

Spedizione. La fase finale dell'intero processo, attraverso la quale la merce raggiungerà il cliente, che come già detto può essere finale oppure una parte dell'azienda stessa, come un impianto o un'unità produttiva. In questa fase si cancellano dal database i codici degli oggetti uscenti e se necessario si rimettono in ordine. Solitamente, questa fase è meno costosa della precedente, in quanto gli oggetti sono stati raggruppati in base agli ordini e dunque si hanno quantitativi inferiori di imballaggi fisici da gestire.



COMPLESSITÀ E SIMULAZIONE SOCIALE

5.1 Introduzione ai Modelli

In Conte (1997), leggiamo che la simulazione sociale è un insieme di discipline che si trovano al confine tra la scienza computazionale, tipica dei modelli matematici, e le scienze sociali tradizionali. Con l'evoluzione delle tecnologie informatiche e la crescita esponenziale della potenza di elaborazione dei calcolatori elettronici, negli ultimi anni la ricerca nel campo della simulazione dei modelli sociali ha subito un impulso notevole. Simulazioni praticamente impossibili da computare con i mezzi disponibili negli anni '80, possono ora essere elaborate da un normale personal computer di basso costo e si aprono prospettive ancor più interessanti per gli anni a venire.

Vi sono diversi impieghi della simulazione eseguita su calcolatori; i principali obiettivi sono i seguenti:

1. **Proiezione:** la previsione degli esiti di interventi di politica economica, istituzionale o sociale a volte richiede un'applicazione e una verifica empirica, e quindi più affidabile, dei modelli elaborati. Diventa essenziale quando i processi indagati sono per loro natura molto complessi, come è il caso dei processi non lineari, stocastici, multidimensionali e così via. Queste simulazioni, tuttavia, non sono necessariamente basate su un modello ad agenti. In esse, l'unità di analisi è una operazione (una transizione fra stati), non il sistema che effettua tale operazione.
2. **Sperimentazione:** per i fenomeni sociali non è possibile, come invece avviene per scienze esatte quali la fisica e la chimica, riprodurre fenomeni in laboratorio, da poter poi estendere alla realtà. La verifica dei modelli sociali al calcolatore, dunque, non è effettuata solamente allo scopo di ottenere risultati immediati, affidabili e precisi, ma anche allo scopo di coprire tale lacuna: l'obiettivo in questo caso non è la previsione, ma la spiegazione di fenomeni della realtà. Contribuisce all'indagine la possibilità di manipolare a piacere le variabili del modello

3. costruito, ottenendo varianti della simulazione e paragonare i risultati con quelli ottenuti in precedenza. Il computer diventa così una sorta di laboratorio delle scienze sociali.
4. **Esplorazione:** mentre gli aspetti precedenti hanno stretta correlazione con l'implementazione su computer del modello, o di un set di modelli, di un fenomeno reale, la simulazione al calcolatore può avere una funzione strettamente esplorativa. Più recentemente, questo tipo di piattaforma è stata utilizzata per lo studio di diversi aspetti della dinamica sociale e, soprattutto, per l'evoluzione della cooperazione, nonché di modelli storici ed evolutivi.

Con il termine "simulazione sociale" si definisce dunque l'uso di modelli precisamente specificati, formulati ed eseguiti su computer, per ricreare e studiare aspetti essenziali della socialità e delle società naturali, siano esse umane, non umane o artificiali.

Per agente si intende l'unità di popolazione osservata: una popolazione può essere una società umana, una specie animale, un'azienda, una catena di fornitura.

Lo studio simulativo consta solitamente di due fasi distinte: il cosiddetto modeling, ossia l'ideazione e la costruzione concettuale del modello da simulare e la simulazione vera e propria con il supporto di strumenti informatici.

Il modello, anche se dovrà rappresentare una realtà concreta, non deve necessariamente essere troppo complesso: esso deve descrivere gli aspetti fondamentali di ciò che si vuole descrivere, trascurando tuttavia quelli che finirebbero per rendere difficoltosa la lettura e l'interpretazione dei risultati ottenuti. Esistono approfonditi studi riguardanti l'affidabilità e la stima dei risultati ottenuti da modelli simulati al computer. Il limite più evidente consiste nella possibilità di ottenere previsioni non verificate nel reale, che portano ad incoerenze e risultati inutili. L'aspetto più critico consiste invece nel rischio di avere previsioni troppo generiche, che dunque non consentono una stima efficace dell'affidabilità del modello in questione: in questi casi risulta molto difficile decidere se tale modello sia utilizzabile per previsioni teoriche di aspetti non osservabili direttamente e dunque non confrontabili con la realtà in modo immediato.

5.2 Simulazione sociale al calcolatore

Esistono comunque almeno due campi in cui la simulazione al calcolatore risulta necessaria, o comunque molto utile:

1. Nei modelli dotati di capacità di apprendimento, in cui, cioè, gli agenti sono capaci di modificarsi e adattarsi in base all'interazione con l'ambiente fisico o sociale. Questo tipo di simulazione è resa possibile grazie a sistemi basati su reti neurali, o algoritmi basati su regole, in grado di riprodurre, anche se in maniera molto semplificata, i comportamenti umani.
2. negli studi sull'emergenza di fenomeni particolari, quali ad esempio la cooperazione, simulando la crescita di società. Fenomeni emergenti vengono osservati in sistemi dinamici e capaci di auto-organizzazione, i quali si modificano per rispondere a cambiamenti sopravvenuti nell'ambiente, contribuendo, così, a modificare l'ambiente stesso.

In questi tipi di studi, l'analisi meramente descrittiva di un modello non potrebbe sostituire la simulazione al calcolatore, poiché quest'ultima permette realmente al sistema di agire in un mondo artificiale, che costituisce la riproduzione in scala di una situazione ed un ambiente realmente osservati.

Questo tipo di simulazione ha certamente dei costi, rintracciabili nella disponibilità di strumenti tecnologici necessari, nel tempo e risorse necessarie per la costruzione del modello, l'apprendimento e la programmazione al computer, nei costi di elaborazione in senso statistico, di interpretazione dei risultati, di *debugging*, ed infine di comprensione.

Un secondo gruppo di costi è rappresentato dall'arbitrarietà della scelta del modello, del linguaggio di programmazione e dei valori da assegnare ai parametri della simulazione.

Un terzo costo è rappresentato dalla “trappola della verosimiglianza”, ossia dalla moltiplicazione delle categorie descrittive, nel tentativo di rendere la simulazione più realistica possibile: questo può richiedere un grande sforzo progettuale aggiuntivo, non sempre ripagato dai risultati ottenuti.

Inoltre, in Terna (1998) leggiamo che scegliendo il paradigma di simulazione di modelli basati su agenti, entriamo in un mondo molto ampio, e per la maggior parte inesplorato, dove la metodologie e le tecniche sono ampiamente ancora da costruire.

Ancora in Conte (1997), si identificano i seguenti temi unificanti, emersi da vari studi:

1. emergenza: insorgenza spontanea ma non necessariamente graduale di effetti rilevanti per il successivo livello di adattamento di un sistema;
2. complessità: ovvero differenziazione, specializzazione funzionale, *clustering*, formazione di coalizioni, organizzazioni.
3. evoluzione: modifiche adattive dei vari sistemi in risposta al loro ambiente;
4. distribuzione: condivisione di proprietà e comportamenti (strategie stabili, conformità) da parte delle unità del sistema;
5. micro-macro: meccanismi e fattori di raccordo fra vari livelli di complessità (ad esempio, fra agenti individuali e sovra-individuali, oppure fra coalizioni spontanee all'interno di un'organizzazione);
6. comunicazione: necessaria alla coordinazione, ma anche alla diffusione di certe regolarità, ad esempio del conformismo; un problema che sta emergendo nell'ambito della simulazione sociale è il ruolo degli altri nell'influenza normativa, ad esempio nella punizione dei trasgressori.

5.3 La teoria della complessità

In Epstein e Axtell (1994) si osserva che molti processi di tipo sociale sono complessi, poiché non si possono scomporre in sotto-processi elementari. Gli aspetti economici, demografici, culturali, spaziali, possono solamente essere aggregati, per fornire una visione di insieme dei processi sociali, ma la loro analisi isolata non consente di spiegare e prevedere i fenomeni che si osservano dall'esterno.

In un volume non recente, ma ancora molto attuale di Gleick (1987), si osserva che dove inizia il caos, la scienza classica si ferma. Mentre i lati più regolari della natura sono stati spiegati da fisici e scienziati in genere, gli aspetti irregolari hanno sempre costituito veri enigmi per la scienza.

Il termine “complesso”, riferito allo studio dei modelli dinamici basati su agenti, non è sinonimo di “complicato”, nell’accezione di “difficile”. Per complessità si intende un fenomeno matematicamente definibile, una caratteristica qualitativa di un sistema, cioè di un aggregato organico e strutturato di parti tra loro interagenti, che gli fa assumere proprietà che non derivano dalla semplice giustapposizione delle parti stesse.

Il concetto di *caos* è nato negli anni ‘60 da studi su calcolatore di un modello di simulazione della meteorologia del globo. Durante questi studi si è scoperto che cambiamenti minimi delle condizioni meteorologiche iniziali potevano avere come conseguenza enormi differenze di clima dopo un certo tempo. Si era così scoperto il cosiddetto "Effetto farfalla" (*Butterfly effect*) descritto paradossalmente come: a seconda del movimento che farà una farfalla in una piazza di Pechino, questo potrà influenzare, alcune settimane dopo, la nascita o no di un uragano nel Golfo del Messico. La scoperta di questi e altri comportamenti inattesi, combinata con altre problematiche esistenti nell'aria già da tempo in numerose branche della scienza, ha catalizzato la nascita di questa disciplina. Due premi Nobel: Ilya Prigogine, chimico belga di origine russa, conosciuto per i suoi studi sui processi irreversibili, e il fisico americano di origine ungherese Murray Gell-Mann, scopritore dei quark, hanno appoggiato in particolare lo sviluppo di questa scienza.

La Scienza della Complessità è una scienza generale che trova applicazione praticamente in tutti i campi scientifici come la fisica, la chimica, la biologia, la medicina e anche in campi come l'economia, la sociologia e la psicologia. La trasformazione del moto di un fluido da lineare in turbolento, il comportamento di certe reazioni chimiche o di ecosistemi biologici, la fibrillazione del cuore, l'andamento dei prezzi di borsa, i conflitti e i processi decisionali sono tutti esempi di fenomeni che possono essere studiati con successo dalla Teoria della Complessità.

La Scienza della Complessità studia per esempio come un sistema altamente disordinato possa in certe condizioni sviluppare spontaneamente dei sistemi organizzati in grado di adattarsi all'ambiente che li circonda. La Scienza della Complessità studia anche le condizioni di trasformazione di sistemi ordinati in sistemi caotici (Teoria del caos).

Al di là dei numerosi studi accademici, condotti in particolare negli USA in questo campo, negli ultimi dieci anni si sono sviluppate, essenzialmente negli Stati Uniti, applicazioni pratiche nel campo del management e della gestione in generale di sistemi complessi, considerati alla stregua di ecosistemi non necessariamente di natura biologica, e basate sui principi della Scienza della Complessità.

Il punto centrale della Scienza della Complessità è nella definizione e descrizione del CAS (*Complex Adaptive System*) . Il CAS è un Sistema Complesso Adattativo, cioè che

ha la proprietà di adattarsi al suo ambiente per perseguire i propri scopi. Esso può essere moltissime cose, come un'impresa, una multinazionale, un mercato borsistico, un ospedale, internet, un distretto industriale, un'innovazione tecnologica. Un CAS può essere costituito da altri CAS al suo interno e allo stesso tempo costituire un elemento di un CAS più grande, inoltre, un CAS non evolve mai da solo ma evolve con altri CAS.

Un CAS è composto da un sistema di Agents ovvero di Agenti individuali o Attori che hanno la libertà di agire in maniera non totalmente prevedibile e le cui azioni sono interconnesse in modo che una certa azione di un agente possa influenzare le azioni degli altri agenti. Ad esempio, nel caso di un mercato borsistico gli attori sono i venditori, i compratori, le società quotate e le organizzazioni regolatrici del mercato. In un CAS ogni attore opera secondo le sue proprie strategie o modelli mentali, questi possono essere condivisi con altri attori o essere individuali. Il comportamento di un CAS emerge dall'interazione tra gli attori ed è tipicamente non lineare. Grandi cambiamenti possono essere assorbiti senza grandi conseguenze mentre piccoli cambiamenti possono causare ristrutturazioni maggiori. Le caratteristiche del CAS rendono praticamente imprevedibile la sua evoluzione. Tuttavia, in alcuni casi, è possibile identificare comportamenti ciclici e situazioni ricorrenti che permettono una certa prevedibilità a corto termine e a lungo termine esattamente come in meteorologia è possibile prevedere il tempo dell'indomani, l'approssimarsi di un temporale o il variare delle stagioni.

Esistono molteplici altre definizioni che mettono in luce la difficoltà a delimitare un preciso ambito di applicazione della teoria della complessità: a causa della molteplicità definitoria Horgan (1995) ha affermato ironicamente che si sta passando “dalla complessità alla perplessità”.

Una buona definizione del fenomeno è data da Day (1994): un sistema dinamico è complesso se dal punto di vista endogeno non tende asintoticamente ad un punto fisso, ad un ciclo limitato, o ad un'esplosione. Tale sistema può mostrare un comportamento discontinuo e può essere descritto da un sistema di equazioni differenziali o di equazioni alle differenze finite, potenzialmente con elementi stocastici. Ma non tutti i sistemi con tali caratteristiche genereranno la complessità. La funzione esponenziale, molto usata per descrivere i modelli di crescita, è un esempio di sistema non complesso e non lineare, perché esplode.

In Rosser (1999) si afferma che questa definizione è molto valida dal punto di vista matematico, ma non tiene conto che, sebbene la complessità sia un fenomeno multidisciplinare che deriva dalla matematica e dalla fisica, nell'economia emergono

complicazioni aggiuntive non presenti in altri campi di studio, a causa dell'interazione di calcoli umani nelle decisioni.

Pur tenendo presente tale limitazione, la definizione di Day è l'unica che permette di includere nel fenomeno tutte le teorie precedenti sui sistemi dinamici non lineari: la cibernetica, la teoria delle catastrofi e la teoria del caos.

Le quattro correnti di studio, i cui nomi cominciano tutti con la "C", costituiscono il percorso evolutivo degli studi sulla non linearità del XX secolo:

1. La cibernetica (Forrester 1961) si sofferma sulla possibilità che all'interno di sistemi multipli di equazioni non lineari possano emergere risultati sorprendenti e controintuitivi. Tale idea è rimasta un principio fondamentale di tutto lo studio sui modelli dinamici, fino ai più moderni modelli sulla complessità, nella quale è presente il concetto di auto-organizzazione ed emergenza di strutture organiche dall'interazione di sistemi semplici
2. La teoria sulle catastrofi, elaborata da Thom nel 1975 sulla base delle precedenti teorie di sistemi dinamici. Una catastrofe è un particolare tipo di discontinuità in un tale sistema; le discontinuità dipendono da punti di equilibrio multipli e distinti tra loro e comportano il salto da un punto all'altro con il variare dei parametri del sistema
3. La teoria del caos ha influenzato lo studio di molte discipline scientifiche. Le dinamiche caotiche sono generate da un processo deterministico, sebbene ad un'analisi ad occhio nudo o ad un'analisi statistica non approfondita esse appaiano meramente casuali
4. La complessità accoglie tutte le premesse teoriche delle precedenti correnti di studio ed evolve l'analisi in diverse discipline. Presso l'istituto per gli studi sulla complessità di Santa Fe è stata elaborata una elencazione di caratteristiche presenti nei modelli complessi, quale surrogato di una formula definitoria di difficile enucleazione

Arthur, Durlauf e Lane (1997) suggeriscono che un sistema complesso consiste di sei caratteristiche:

1. interazione diffusa tra agenti eterogenei, che agiscono localmente in uno spazio;
2. nessun controllore centrale del modello, sebbene siano consentite deboli interazioni a livello globale;
3. organizzazione gerarchica multi-livello, con interazioni distribuite a diversi livelli;

4. adattamento continuo degli agenti, che sono in grado di imparare ed evolvere;
5. continue novità come nuovi mercati, tecnologie e comportamenti, creano nuove nicchie nell'“ecologia” del sistema;
6. dinamica priva di un equilibrio globale, ma con molti punti di equilibrio instabili.

Come risultato di tali caratteristiche si ottiene un ambiente caratterizzato da razionalità limitata e da aspettative non razionali.

5.4 Sistemi complessi di tipo adattivo

In Vriend (1999) si afferma che un sistema complesso è costituito da un gran numero di parti tra loro indipendenti che sono interconnesse ed interattive. Si dice adattivo un tale sistema, se le sue parti sono costituite da agenti che modificano le loro azioni in funzione di eventi che si scatenano durante il processo di interazione. Alcuni esempi di sistemi complessi sono i sistemi biologici, il sistema immunitario, il cervello, il sistema meteorologico, le società. Un'economia decentralizzata, la quale consiste in un grande numero di agenti razionali interconnessi ed interagenti, che ricercano continuamente vantaggi ed opportunità, è un buon esempio di sistema adattivo complesso. Una caratteristica fondamentale di questi sistemi è l'impossibilità di ricavare le proprietà globali dalla semplice analisi dei suoi componenti.

Un sistema complesso si differenzia da un sistema caotico per la sua tendenza ad evolvere non raggiungendo né equilibri ottimi e stabili, né una casualità completa. Ciò è dovuto alla capacità di auto-organizzarsi ed al meccanismo della selezione: la selezione sembra agire spingendo il sistema lontano dagli estremi di ordine e caos. Lungo questa media i sistemi sembrano comportarsi in modo molto complesso e adattarsi molto rapidamente alle variazioni ambientali.

Poiché le interazioni tra gli agenti individuali sono in generale non lineari, da un punto di vista matematico rappresentano spesso problemi intrattabili. L'attuale apparato analitico che consiste nei meccanismi statistici, nella teoria delle interazioni fra particelle è restrittivo rispetto ai contenuti economici dei modelli.

Ciò accade particolarmente perché le interazioni fra agenti non avvengono nella realtà in funzione della loro posizione nello spazio. Molte interazioni avvengono poiché gli stessi agenti ricercano le condizioni più vantaggiose. All'interno di una economia di

mercato le interazioni avvengono soprattutto perché gli agenti conoscono i potenziali partner con cui scambiare.

Un'economia decentralizzata non è un luogo dove un certo bene è scambiato, né la domanda e l'offerta aggregata di tale bene. In generale, i mercati emergono come risultato di interazioni a livello locale tra agenti individuali che ricercano scambi vantaggiosi. Essi sono auto organizzati e durante i loro processi di interazione avviene l'evoluzione, ovvero l'apprendimento, da parte degli agenti. Una caratteristica degli agenti che vivono nella complessità di un mondo così ampio è che non hanno un vero modello con il quale lavorare.

McKelvey (1997) afferma che l'attività di interpretazione dal punto di vista del comportamento sociale delle relazioni e delle interazioni tra agenti richiede l'analisi contemporanea di quattro elementi:

1. la fisica, per quanto riguarda l'analisi delle quattro forze della teoria dei campi elettromagnetici;
2. la biologia, relativamente ai principi della selezione naturale;
3. la razionalità, per gli effetti delle decisioni assunte dagli attori del sistema;
4. la complessità.

Le ultime tre discipline sono rilevanti nello studio delle scienze sociali.

Se almeno alcuni fenomeni sociali, che sono tipicamente considerati derivanti dal comportamento razionale, emergono invece dalle dinamiche complesse che sono parzialmente influenzate da un'azione conscia, allora dobbiamo includere questi principi fondanti nel disegno dei modelli. Tutta l'attenzione va dunque riposta nella progettazione dei singoli agenti. Ferber (1989) identifica un agente come "un'entità reale o astratta". Tutti gli agenti utilizzati nelle simulazioni al computer sono artificiali e sono usati per produrre società virtuali. Queste società possono assomigliare alle analoghe società reali ed in tal caso la realtà va interpretata tenendo conto che le conclusioni sono realistiche, cioè assomigliano alla realtà, ma non necessariamente corrispondono a quest'ultima.

In base a questo principio un agente deve possedere "la capacità di agire su se stesso e sull'ambiente". Ciò implica la capacità dell'agente di realizzare semplici processi di auto-regolazione basati su meccanismi di feedback derivanti dall'ambiente. La capacità di agire sull'ambiente indica banalmente che l'agente deve essere integrato nel sistema, con il quale è in grado di interagire. In altre parole, non è lecito progettare agenti come entità astratte, prive di plausibilità nel confronto con i sistemi che rappresentano. L'ambiente

non deve avere diretto controllo sull'agente, ma si deve realizzare un meccanismo di interazione che permette in entrambe le direzioni di influenzare indirettamente il cambiamento di stato interno all'ambiente ed al singolo agente.

Il problema di dotare di intelligenza gli agenti del sistema è risolto in Minsky (1987) con l'affermazione che "l'intelligenza può emergere dalla non-intelligenza". Si può infatti rappresentare la mente come una società di agenti, ognuno con funzioni distinte. La mente emerge dall'interazione tra questi agenti dalle funzionalità differenziate, i quali sono entità autonome ed hanno la proprietà di poter essere utilizzati in molte differenti sequenze di interazione per realizzare attività diverse. Tale capacità di eseguire una molteplicità di funzioni da parte del sistema viene identificata come capacità intellettuale. Gli agenti comunicano tra loro ad un livello locale e non è necessario il controllo da un livello più elevato. In altre parole dal punto di vista biologico l'intelligenza deriva da processi autocatalitici, che sviluppano capacità molto sofisticate, sotto l'influenza di forze ambientali selettive.

In un'economia decentralizzata ogni attività individuale è sviluppata dalle attività e dalle decisioni degli altri agenti. Ogni agente possiede un ambiente profondamente diverso a seconda del tipo di attività che deve svolgere. In altre parole, mentre un agente individuale si adatta all'ambiente, parte dell'ambiente si adatta ad esso. In biologia questo fenomeno è detto "coevoluzione".

5.5 Analisi dei sistemi adattivi complessi (CAS)

Holland (1995) sostiene che l'analisi dei CAS deve essere condotta a due livelli: lo studio dei modelli interni e i *building block*. I modelli interni si identificano con i meccanismi che generano il fenomeno dell'anticipazione, nei sistemi adattivi complessi (CAS). Attraverso questo elemento si possono interpretare meglio i meccanismi correlati con l'adattamento e con la capacità di apprendimento che si osserva a diversi livelli di complessità: essi dipendono dall'esperienza e dalla sensibilità di ciascun agente.

Esistono due tipi di modelli interni: impliciti ed espliciti. I primi, semplicemente condizionano le azioni degli agenti sulla base di un'implicita previsione di alcuni eventi futuri. I secondi presuppongono un'esplicita, sebbene interna, esplorazione delle alternative, processo noto anche come *lookahead*.

Ciò è giustificato dalla constatazione che la struttura dell'ambiente agisce attivamente a determinare il comportamento del singolo agente. Quindi, se le azioni

intraprese sono frutto di un'utile anticipazione delle conseguenze future, l'agente possiede effettivamente un modello interno. Quando esiste un efficace meccanismo che consente di collegare le conseguenze future ad un'azione corrente, l'evoluzione può favorire i modelli interni efficienti ed eliminare quelli inefficienti.

Le speranze di sopravvivenza dell'agente dipendono criticamente dalla sua capacità di prevedere, implicitamente o esplicitamente, i dettagli del modello, elaborando congetture coerenti. L'intero schema di aspettative abituali, o di *meaning perspectives*, costituisce il codice che governa le attività di percezione, comprensione e memorizzazione: i simboli che noi proiettiamo all'interno delle nostre percezioni sensoriali sono filtrati attraverso le percezioni di significato.

Un CAS acquisisce informazioni sull'ambiente e sulle sue interazioni con l'ambiente stesso, identifica le regolarità di queste informazioni, condensa tali regolarità in una sorta di "schema" o modello e agisce nell'ambiente sulla base di esso. Esistono diversi schemi, i quali competono sulla base dei risultati che provocano sul sistema: il meccanismo di feedback dell'ambiente influenza la competizione tra tali modelli interpretativi.

Gli schemi sono teorie e ciò che avviene nella relazione con l'ambiente è il confronto tra la teoria e le osservazioni; le nuove teorie devono competere con quelle già esistenti, in parte sulla base della coerenza e della capacità di generalizzazione, ma soprattutto sulla base della loro capacità di interpretare le osservazioni e di prevedere quelle future.

Il secondo livello di analisi è costituito dai *building block*: per Holland (1995) i *building block* sono un elemento riutilizzabile e ripetitivo di un sistema. Essi costituiscono una strada per costruire le gerarchie senza spendere troppe risorse. Un esempio sono i neuroni di una rete neurale, i singoli geni all'interno di un algoritmo genetico o gli oggetti in una rappresentazione visiva. I modelli interni danno forma ai *building block*.

Nelle situazioni reali un modello interno deve essere fondato su un numero limitato di esempi relativi ad un ambiente in continuo mutamento. Il modello può essere utile solo se si possono rilevare alcuni tipi di ricorrenze nelle situazioni descritte. Poiché il riutilizzo significa ripetizione, è necessario ricercare una metodologia in grado di cogliere queste ripetizioni durante il confronto con la realtà. È possibile acquisire esperienza attraverso l'uso ripetuto dei *building block*, anche se essi potrebbero non riapparire mai nella stesso ordine sequenziale.

Se la costruzione dei modelli, largamente interpretata, comprende molte attività scientifiche, allora la ricerca attraverso i *building block* diventa la tecnica per aumentare queste attività.

Si ottengono significativi risultati quando si riducono i blocchi ad un unico livello, cioè alle sole interazioni, oppure ad uno stadio più basso, cioè alle combinazioni tra essi: la legge si ricava, da un punto di vista più astratto, dalle leggi che governano i singoli *building block*. Ciò fornisce un'incredibile forza di legame alla struttura scientifica.

Possono essere riutilizzati e ricombinati per creare scenari completamente nuovi: attraverso la loro scomposizione e l'uso ricombinato, si può osservare l'emergenza di nuovi fenomeni. Possono essere prodotte un grande numero di combinazioni diverse da un piccolo insieme di blocchi e di regole per combinarli.

Diventano molto potenti quando sono applicati a progetti in cui è necessaria l'aggregazione e la descrizione a vari livelli di astrazione. Il processo di aggregazione (Holland 1995), riguarda l'emergenza di comportamenti complessi su larga scala, dalle interazioni aggregate di un numero inferiore di agenti complessi. Questa è considerata una caratteristica fondamentale di tutti i sistemi adattivi complessi.

In Merry (1999) è interessante osservare le differenze tra un sistema adattivo complesso di tipo naturale rispetto ad uno di tipo umano. Le organizzazioni sociali possiedono un livello di coscienza ed hanno una migliore capacità di manipolazione interna delle informazioni: esiste un più elevato livello di comunicazione e di cooperazione tra gli agenti. Mentre nei sistemi naturali gli agenti operano secondo leggi naturali, le organizzazioni umane sono coordinate da regole costruite socialmente.

Una descrizione degli elementi che caratterizzano i sistemi CAS è data da Merry (1999):

1. Ogni sistema di questo tipo è composto da una rete di componenti che costantemente e reciprocamente si influenzano tra loro. Esse possono essere tanto i neuroni di un cervello, quanto gli operatori di un mercato economico.
2. Gli agenti non sono controllati centralmente, i fenomeni emergono dalla loro interazione: nessuna componente è in grado di controllare le altre (sistema decentrato). I comportamenti, apparentemente coerenti, sono frutto della competizione e della collaborazione tra i componenti.
3. Ogni livello rappresenta un *building block* per il livello successivo, proprio come un gruppo di cellule creano un tessuto o un gruppo di professionisti formano un team.

4. Il meccanismo fondamentale è l'adattamento, che si realizza in seguito alla costante riorganizzazione interna delle componenti.
5. Essi sono in grado di prevedere gli accadimenti futuri. Le previsioni sono basate sui cambiamenti che avvengono nei modelli interni. Essi sono la rappresentazione che ogni agente ha dell'ambiente in cui opera. Tali modelli sono costantemente modificati in seguito all'esperienza.
6. Le opportunità sono create dallo stesso sistema che sviluppa costantemente nuovi fenomeni o situazioni.
7. Ogni processo è in costante cambiamento. Non si raggiunge mai un equilibrio ottimale perché esiste un continuo flusso di relazioni con gli altri sistemi, i quali provocano perturbazioni.

5.6 Agent-based computational economics

Uno dei più importanti effetti risultanti dallo studio della dinamica dei modelli economici complessi è stato il cambiamento nel metodo e nella direzione della ricerca. Nei nuovi modelli vengono descritte soltanto le interrelazioni a livello locale tra agenti individuali, mentre i comportamenti aggregati o le strutture emergono dall'auto-organizzazione, invece che essere semplicemente imposti o accettati. Ciò che emerge in un modello, aggregato potrebbe non essere la semplice somma di quello che avviene a livello individuale: ciò è in contrasto con il metodo dei modelli ad agenti rappresentativi, nei quali la somma degli individui equivale all'aggregato.

Non è comunque certo che questo tipo di studio fornisca risultati superiori a quelli ottenuti con altri sistemi: la giustificazione all'uso di una metodologia che a priori non garantisce risultati migliori rispetto a tecniche consolidate, di cui si conoscono i limiti e le potenzialità risiede nella capacità teorica di abbattere il muro concettuale che separa il "mondo reale" da quello teorico. Questi modelli impongono la plausibilità nella costruzione degli agenti, in contrasto con l'uso frequente di semplificazioni ed assiomi della metodologia teorica classica.

In Testfatsion (1998) viene data una definizione per la cosiddetta Agent-based computational economics (ACE): si tratta dello studio computazionale dell'economia, attraverso l'uso di sistemi decentralizzati di agenti interagenti ed autonomi.

L'obiettivo di tale metodologia è la comprensione della comparsa apparentemente spontanea di regolarità a livello globale nei processi economici, nello stesso modo in cui opera la coordinazione non pianificata degli scambi in mercati economici decentralizzati, che gli economisti spiegano con la metafora della mano invisibile di Adam Smith. La sfida è la spiegazione di queste regolarità con un procedimento che va dal basso verso l'alto: questo significa che le regolarità emergono dalle interazioni globali di agenti autonomi coordinati da attuali o potenziali istituzioni economiche, piuttosto che attraverso i meccanismi di coordinamento di tipo *top-down* come avviene nel modello del consumatore rappresentativo. In Testfatsion (1998) si trova una buona definizione della materia: la Agent-based computational economics è lo studio computazionale di economie modellate come sistemi decentralizzati in evoluzione, composti da agenti autonomi ed interagenti tra loro.

In Terna (1997) si legge che l'introduzione della metodologia di tipo ACE è anche legata alla necessità di incorporare la razionalità limitata nei nostri modelli economici, con la complessità tipica della realtà che emerge dalla interazione tra agenti e non dalla complessità intrinseca dell'agente stesso.

Invece di analizzare il processo che conduce all'equilibrio economico date alcune ipotesi, nella metodologia ACE tali equilibri emergono come prodotto dell'interazione tra gli agenti individuali.

Al fine di sviluppare gli esperimenti di tipo ACE, introduciamo le seguenti ipotesi generali:

1. un agente che agisce in un ambiente economico deve svilupparsi ed adattarsi coerentemente, soprattutto quando deve interagire con altri agenti
2. la complessità può essere riscontrata più frequentemente al di fuori degli agenti, in un contesto che emerge dall'interazione, dall'adattamento e dall'apprendimento, piuttosto che al loro interno
3. allo stesso modo anche la razionalità può essere osservata al di fuori degli agenti, semplicemente come prodotto delle caratteristiche dell'ambiente e delle capacità limitate degli agenti

4. non è corretto dedurre dalla descrizione dell'agente ogni apparato formale ricercando la complessità al loro interno

Nel disegnare un metodo di apprendimento nella rappresentazione del comportamento umano all'interno di un particolare contesto, non si deve cercare di riprodurre soltanto i tassi umani di apprendimento, ma anche lo "stile" con il quale gli uomini imparano e possibilmente anche il modo in cui essi si discostano dalla razionalità perfetta. L'obiettivo è, dunque, non soltanto riprodurre la curva di apprendimento che rappresenta una buona approssimazione di quella umana, ma più ambiziosamente, un comportamento nell'apprendimento che permetta di superare il test di Turing, il quale consiste nel rendere il comportamento simulato indistinguibile da quello umano. (Arthur 1990).

Per evitare di costruire modelli troppo complessi da gestire, si considerano agenti dotati di intelligenza artificiale, fondata su algoritmi che consentono l'apprendimento mediante un processo di prove e correzioni progressive.

Gli agenti dotati di tali limitazioni, ma con la capacità di adattamento, costituiscono la base per la costruzione di modelli di interazione molto sofisticati, ispirati direttamente ad un ambiente economico o finanziario, o in modo più astratto a modelli di vita artificiale. Essi sono modelli finalizzati all'osservazione del fenomeno di emergenza di organizzazioni strutturate gerarchicamente.

Il fenomeno che giustifica questa metodologia di indagine è spiegato da Hayek (1948) come una conseguenza del cambiamento: se i fenomeni sociali fossero sempre uguali o presentassero forti regolarità, non ci sarebbe la necessità di prendere decisioni e non sarebbero richieste particolari strategie. In generale, la giustificazione della capacità adattiva degli agenti sta proprio nella irregolarità dei fenomeni osservati.

Le sole informazioni di cui gli agenti sono a disposizione corrispondono alle proprie esperienze e, soprattutto, alle esperienze di agenti che hanno dovuto affrontare una decisione analoga in passato: ciò è descritto in letteratura come fenomeno del contagio informativo.

Sempre Hayek (1948) fornisce una convincente spiegazione sul vantaggio di questo metodo di indagine scientifica, quando paragonato a quelli tradizionali. I fenomeni naturali non possono essere integralmente spiegati dalla statistica ed i dati ad essi relativi non sono naturalmente convogliati ad un punto centrale di raccolta. È il centro in cui si assumono le decisioni che deve raccogliere tali informazioni, astraendo dalle differenze di ordine minore, al fine di ottenere dati omogenei. Ma spesso il processo di astrazione

elimina informazioni significative ai fini della decisione. Gli agenti per propria natura sono in grado di tenere traccia di queste informazioni e di contribuire attivamente all'analisi aggregata dei dati, grazie alla loro capacità adattiva.

5.7 Identificazione ed acquisizione della conoscenza

La costruzione di modelli di simulazione fondati su agenti, è un metodo decisamente teorico di analisi dinamica delle strutture sociali, sebbene l'obiettivo sia la plausibilità. Attraverso un modello così costruito, si può predisporre una rappresentazione della realtà che ha il vantaggio di riprodurre le regole di funzionamento e di consentire un'analisi dell'impatto che talune modificazioni del sistema causano e quindi la metodologia può essere applicata non soltanto allo studio teorico del sistema stesso, ma anche per simulare un sistema aziendale concreto. L'indagine basata sulla simulazione apporta un vantaggio sensibile alle decisioni dei manager, poiché tali decisioni comportano la comprensione di realtà complesse, in quanto sociali. Le tecniche descrittive, o matematico-statistiche, non forniscono neppure un surrogato di sperimentazione delle soluzioni ipotizzate.

Nel processo di formulazione del modello emerge la necessità di rappresentare innanzitutto la conoscenza posseduta dall'organizzazione, oltre che la semplice descrizione delle strutture tangibili.

La conoscenza è una manifestazione del meccanismo dei modelli interni. Essa dovrebbe intendersi come una sintesi di informazioni elementari, secondo un percorso di astrazione utile ad ogni singolo operatore e tale da rappresentare per costui un abilitatore nello svolgimento di attività complesse (Takada, 1994).

La necessità di rivolgere l'attenzione al meccanismo di memorizzazione della conoscenza all'interno di un sistema, è importante quando gli agenti del sistema siano caratterizzati da comportamenti cognitivi. Nel descrivere un modello ad agenti che rappresenta la una realtà aziendale, comprendere le caratteristiche dei processi di apprendimento e creazione della conoscenza nell'organizzazione nel suo complesso è fondamentale per poter descrivere il modello in termini di *building block* e di schemi di apprendimento che definiscano gli "atomi" del sistema.

Molto frequentemente in azienda si considera conoscenza un ammasso, spesso impressionante, di materiale informativo come messaggi, relazioni, procedure, listini e

simili. La quantità di sforzi e di insuccessi cui un soggetto deve sottostare durante la ricerca e l'astrazione da queste informazioni, delle parti necessarie allo svolgimento delle proprie attività, dimostra che non si tratta della vera conoscenza. Se si ha l'obiettivo di simulare il comportamento di tale soggetto è necessario prima formalizzarne i contenuti conoscitivi.

La conoscenza, può essere a questi fini disposta in una gerarchia del tipo seguente, secondo quanto schematizzato da Pepe (2000):

1. Conoscenza elementare, la conoscenza desumibile dalla ricerca per tentativi di informazioni grezze (esempio classico, la conoscenza derivata dall'uso dell'internet). È una conoscenza largamente pubblica, non distintiva, facilmente condivisibile: non è considerata da nessuno un patrimonio tesaurizzabile.

2. Conoscenza tecnica, cioè l'insieme di nozioni e di abilità riguardanti l'uso di una tecnica o metodo di pubblico dominio (ad esempio: conoscenza di programmazione in un certo linguaggio per computer) o sulle modalità di esecuzione di processi canonici d'impresa (ciclo attivo, ciclo passivo, ecc.). È una conoscenza di tipo quasi pubblico, facilmente condivisibile e viene tesaurizzata dalle persone soltanto in ambienti poveri di investimenti in formazione e di attenzione alla crescita delle persone.

3. Conoscenza aziendale, l'insieme di informazioni, competenze nell'usarle e canali di informazione che consentono ad una persona di svolgere la propria attività in termini distintivi costituendo, a parità di processo di business, un elemento di maggiore o minore competitività. Si tratta dunque di conoscenze molto specifiche all'azienda, che possono essere condivise con una certa difficoltà e che le persone mettono in comune solo nell'"*old boys network*", ossia con altre persone che vengono considerate degne della loro confidenza, indipendentemente dagli schemi che sarebbero desiderati dall'azienda. Questa conoscenza dovrebbe rappresentare il punto di massimo sforzo aziendale di costituzione di una *knowledge base*, ma non è sempre possibile codificarla.

4. Conoscenza creativa, che costituisce un patrimonio esclusivamente personale di astrazioni e modalità di manipolazione. Consente di generare una conoscenza non preesistente partendo dall'informazione (ad esempio: modalità di concettualizzazione di un progettista). Sono conoscenze intrinseche alla persona,

non condivisibili e fortemente protette: ogni tentativo di strutturazione è tendenzialmente infruttuoso e genera facilmente conflitti.

5.8 Come un'organizzazione crea conoscenza

Il sistema aziendale, deve tenere presente ed utilizzare la precedente classificazione per strutturarsi, ottenendo con organizzazioni semplici la condivisione per i primi due livelli; per entrare nel terzo dovrà innovare la propria organizzazione (comunità di interesse, *think tanks*, *guru farm*, ecc.). Il quarto livello sarà, invece, rigorosamente rispettato come individuale.

Uno dei fattori che costituiscono il valore complessivo di un'azienda è proprio l'identificazione della conoscenza. Essa rappresenta il patrimonio che un sistema ha acquisito e che gli può permettere di mantenere nel medio periodo un vantaggio competitivo rispetto alla concorrenza e quindi la sopravvivenza.

La teoria dell'organizzazione è stata a lungo dominata da un paradigma nel quale l'organizzazione stessa è concettualizzata come un sistema che “elabora” informazioni o “risolve” problemi. Secondo tale paradigma, uno dei compiti fondamentali di un'organizzazione consiste nella ricerca, attraverso la quale trattare informazioni e prendere decisioni in situazioni di incertezza. Una più completa comprensione dell'attività e dello sviluppo organizzativo richiede una valutazione dinamica di come l'organizzazione stessa interagisce con l'ambiente e dei processi attraverso cui crea conoscenza. Ad esempio l'innovazione, che è una forma tipica di creazione di conoscenza da parte dell'organizzazione, può essere meglio interpretata come processo in cui essa crea e delinea i problemi e quindi sviluppa attivamente nuove informazioni per risolverli. La conoscenza è un concetto ricco di sfaccettature con significati a più livelli. In breve, basti dire che le informazioni sono un flusso di messaggi, mentre la conoscenza viene creata ed organizzata da un flusso di informazioni, a sua volta ancorato all'impegno ed alle opinioni del suo detentore.

In Merry (1999) si tenta di descrivere la differenza tra dati, informazioni, conoscenza e apprendimento:

1. **Informazioni:** la parte della comunicazione che riduce l'incertezza. È la “differenza che fa la differenza”.

2. **Dati:** diventano informazione quando sono significativi.
3. **Conoscenza:** è informazione organizzata in un modello o schema attraverso il quale la realtà è percepita. Essa è l'informazione che possiamo integrare con altre informazioni ed avere a disposizione per agire.
4. **Apprendimento:** l'abilità di identificare percorsi nel flusso delle informazioni ricevute, di dare ad esse un significato, di erborarle e conservarle come esperienza. L'apprendimento è la capacità di assorbire, processare, salvare e adottare le informazioni nella pratica.

5.9 Formalizzazione e trasmissione della conoscenza

Come afferma Polanyi (1966), noi conosciamo più di quanto sappiamo dire. La conoscenza “esplicita” o codificata è trasmissibile con un linguaggio formale, sistematico. C'è però anche una conoscenza “implicita”, che, in quanto appartenente ad una natura personale, è arduo formalizzare e comunicare. L'idea è che essa possa essere creata e condivisa a diversi livelli sociali. Tali livelli sono l'individuo, il gruppo, l'organizzazione e i rapporti tra organizzazioni diverse. È evidente che l'interpretazione che l'individuo dà dell'ambiente organizzativo è un processo interattivo, soggetto a continui aggiustamenti e revisioni. Vi è poi il processo di comunicazione di tali informazioni, il quale avviene nella forma del dialogo. Come tale, il dialogo possiede virtù co-generative: così i partecipanti il dialogo si aiutano reciprocamente nel loro co-sviluppo.

Altro grande vantaggio della formalizzazione della conoscenza è quello di rendere molto più evidenti e manipolabili i legami tra processo umano e supporto computerizzato offerto dai sistemi informativi; facilitazione che nasce dal poter ragionare in modo coniugato su di un dominio interpretabile sui due versanti, utenti e informatici, con relativa univocità di linguaggio.

Secondo Gilbert e Terna (2000) il fattore conoscenza si può rappresentare secondo uno dei formalismi che si classificano in:

1. modelli letterari-descrittivi
2. modelli matematico-statistici

3. modelli realizzati con codice informatico, che forniscono gli strumenti per la simulazione.

Il terzo tipo di rappresentazione fornisce una rappresentazione virtuale del sistema che può diventare oggetto di indagine e di sperimentazione, come se si trattasse di una struttura riprodotta in laboratorio.

Nel processo di formazione della conoscenza si attuano meccanismi non apparenti. Gli agenti di un sistema apprendono memorizzando la conoscenza nel tessuto delle relazioni. Si parla di memoria distribuita. Questo comporta la difficoltà ad esternare in modo strutturato la conoscenza da parte di chi la possiede. Nella fase di formulazione del modello che si propone di descrivere formalmente il sistema è dunque problematico osservare direttamente le caratteristiche della conoscenza. Accade frequentemente che chi ha l'incarico di descrivere il modello della realtà aziendale non coincida con chi ne è l'attore: di conseguenza è necessario uno scambio di informazioni che obbliga gli agenti del sistema a riflettere sulla struttura e sul contenuto delle informazioni stesse, per poterle tramandare. In questo processo di riflessione spesso l'agente scopre quanto la conoscenza sia distribuita nel sistema e quanto sia difficile formalizzarla. Proprio durante questo processo cognitivo, che impone di ripercorrere l'intera struttura delle relazioni, emergono le incongruenze e le inefficienze. Per questa ragione si può asserire che la stessa fase di progettazione di un modello di simulazione può condurre a risolvere o comunque migliorare le problematiche insite nel sistema.

Partendo dai propri modelli interni un'organizzazione agisce nell'ambiente comparando con essi le proprie esperienze che a loro volta, sommate con altre fonti di informazione, arricchiscono il modello stesso, costruendo conoscenza. Per raggiungere tale scopo l'organizzazione deve interagire con i dati che ha a disposizione e con la propria capacità di dare a queste informazioni un significato.

L'apprendimento porta le organizzazioni ad interagire con l'ambiente comparando le informazioni ottenute con la conoscenza e le esperienze di cui è internamente dotata.

La conoscenza è racchiusa nelle relazioni tra chi detiene il sapere e ciò che egli è interessato a conoscere; essa ha un significato soltanto nello schema mentale di chi la possiede.

Al fine di raggiungere e di mantenere un vantaggio in termini di sapere, le organizzazioni devono diventare adattive. Una tale organizzazione crea le strutture, la cultura, le politiche di gestione che incoraggiano l'apprendimento sia a livello individuale che di gruppo, combina continuamente l'adattamento con il rinnovamento, è sempre

aperta alle nuove informazioni, sviluppa strumenti per la formazione, ha un metodo non lineare di apprendimento.

Linearità significa proporzione tra causa ed effetto; l'apprendimento è un processo iterativo non lineare, nel quale i prodotti di un ciclo diventano materie prime per il ciclo successivo. Noi impariamo confrontando le informazioni nuove con la conoscenza e le esperienze di cui siamo dotati.

La conoscenza posseduta in un determinato punto del tempo non è la banale somma dei processi di apprendimento che lo hanno preceduto: nel processo di apprendimento un piccolo input può generare grandi effetti.

In Eoyang (1999) è descritto il processo lungo il quale di forma la conoscenza, cioè il ciclo di apprendimento”:

È un modello semplice per capire come avviene l'apprendimento. Esso è visto come un processo auto-organizzante, che può essere usato nelle comunità virtuali e consiste essenzialmente di tre passi:

1. Discernimento: la differenza tra nuove informazioni e modelli esistenti.
2. Collegamento: chi svolge il ciclo mette il nuovo sapere nel contesto di quello precedente e prova a collegarli
3. Auto-organizzazione: un nuovo modello di sintesi autonoma che crea ordine dalla confusione di dati e notizie.

In Delaini et al. (2000) si afferma che le imprese esternalizzano, decentralizzando segmenti del processo produttivo e restringendo i loro “confini”. Di conseguenza aumentano le transazioni con il mercato per migliorare il loro livello di efficienza e potenziare la produttività. In relazione a ciò, si possono studiare le caratteristiche di particolari decomposizioni. La sfida è dunque capire se e in quali circostanze i mercati – e cioè le “decomposizioni”, le “distribuzioni decentralizzate di conoscenza” evidenzino vantaggi differenziali nella risoluzione di problemi.

Sempre in Delaini et al. (2000) si dimostra che la decomponibilità della conoscenza è la proprietà fondamentale che la rende analizzabile in un contesto di simulazione ad agenti: da un esame, anche solo superficiale, emergono forti somiglianze fra i concetti da noi utilizzati a proposito dell'analisi delle organizzazioni e quelli usati in relazione alla conoscenza ed al *problem solving individuale*. (...) suggeriamo l'idea che ci sia molto più che un'analogia metaforica tra la decomposizione dei problemi nelle organizzazioni

collettive e le decomposizioni ed altre euristiche che gli individui tipicamente impiegano nella loro attività cognitiva. A livello definitorio le decomposizioni sono un particolare tipo di euristiche di ricerca, che a loro volta sono insiemi di regole di esplorazione per limitare lo spazio delle soluzioni che possono essere generate e testate.

In termini astratti decomposizioni ed euristiche hanno la stessa identica funzione: riducono lo spazio di ricerca. Effettivamente, possiamo immaginare un agente che non decompone per nulla ma che ha una potente (efficace) euristica che può limitare notevolmente l'insieme di combinazioni da testare.

In altre parole, una decomposizione non è altro che, dopo tutto, un particolare esempio di euristica. Questo lavoro suggerisce che le organizzazioni realizzino meccanismi collettivi di apprendimento combinando semplici euristiche individuali. Una delle risorse dell'efficacia evolutiva delle organizzazioni è il fatto che esse scompongono enormi problemi tecnici secondo procedure che spesso coincidono con i processi di esplorazione euristica locale e con le caratteristiche di adattamento degli individui.

Dunque, la vera sfida nella costruzione di un modello di simulazione di un'organizzazione produttiva è riposta nella capacità di formalizzare i meccanismi relativi alla conoscenza nei comportamenti degli agenti di tale modello.

6.1 La simulazione al calcolatore

Con la diffusione del computer, uno strumento di calcolo potente e versatile, la ricerca nelle scienze sociali ha subito interessanti sviluppi. Esistono numerosi strumenti software di simulazione, sviluppati parallelamente in ambito scientifico ed in ambito commerciale, che permettono di indagare la natura di modelli dinamici con l'uso della simulazione informatica. Oltre a consentire l'analisi di problemi che matematicamente sono intrattabili, tali strumenti consentono di dare maggiore rilievo al processo dinamico che conduce a situazioni di equilibrio. Tale metodo non porta necessariamente a migliorare i risultati ottenuti con le tradizionali tecniche statistico-matematiche o descrittive, ma fornisce decisamente una migliore plausibilità nel processo di formulazione del modello. Ciò significa che al fine di rendere matematicamente trattabile il modello spesso vengono poste ipotesi forti, la cui dimostrabilità è assai ardua. I risultati del modello sono sempre condizionati dagli assunti che vengono imposti per giungere alla soluzione.

Nello studio dei sistemi economici, i fenomeni che si riscontrano sono frutto dell'interazione parallela di differenti processi. Ciò rende spesso intrattabile il modello in termini di sistema di equazioni che descrivono tali processi e diventa dunque necessario concentrare l'analisi verso un ambito più circoscritto.

Nella simulazione con il calcolatore, il disegno del modello richiede che si faccia una descrizione plausibile e realistica degli agenti (le unità minime del sistema), i cui modelli di comportamento sono più facilmente identificabili e descrivibili, anche da un punto di vista matematico. Le conseguenze a livello aggregato emergono come prodotto dell'interazione tra queste unità minime e riducono il compito del ricercatore alla spiegazione dell'emergenza dei fenomeni osservati in aggregato. Ciò che accade a livello aggregato non è mai frutto di regole o ipotesi imposte dallo sperimentatore.

L'uso dello strumento informatico rende più efficace il lavoro di ricerca, poiché osservare il processo dinamico attraverso il quale il sistema raggiunge l'equilibrio migliora la capacità di comprensione del fenomeno. È inoltre possibile aggiungere a piacere elementi che complicano il modello, al fine di valutarne l'impatto sull'intero sistema.

6.2 Il progetto Swarm

Le problematiche da affrontare, se si intende costruire un modello economico attraverso l'uso del calcolatore, spesso riguardano aspetti che non sono propriamente inerenti alla sostanza di ciò che si vuole studiare. Diventa infatti necessario occuparsi degli aspetti tecnici relativi all'utilizzo del software. Lo strumento che si propone di risolvere parzialmente questo problema e che presenta soluzioni di grande interesse per la comunità scientifica è chiamato Swarm. Esso rappresenta lo strumento di riferimento nello sviluppo di modelli informatici basati su agenti. L'uso di un qualunque linguaggio di programmazione risolve il limite della mancanza di flessibilità di programmi creati ad hoc per una qualche applicazione particolare, ma accresce enormemente la difficoltà e, soprattutto, la limitazione alla potenziale diffusione del modello all'interno della comunità scientifica.

Se si decide di aderire al progetto Swarm si sceglie la strada più difficile da un punto di vista tecnico (bisogna imparare un linguaggio di programmazione a basso livello), ma più rigorosa e positivamente accettata dalla comunità scientifica. Swarm non è un'applicazione, non un linguaggio di programmazione, neanche un ambiente di sviluppo: consiste in una libreria di potenti funzioni scritte in *Objective-C* ed una filosofia progettuale.

In Askenazi et al. (1996) si traccia un breve profilo nell'evoluzione del rapporto tra ricercatori e tecnologia. Agli albori dello sviluppo di una disciplina scientifica, i primi ricercatori sono costretti a sviluppare i propri strumenti sperimentali: molando le lenti, costruendo le proprie particolari sonde di misurazione, talvolta costruendo gli stessi calcolatori. Essi devono diventare ingegneri, meccanici, elettronici, oltre ad essere semplicemente scienziati. Quando un campo di ricerca diventa maturo, la collaborazione tra scienziati e ingegneri conduce a sviluppare strumenti standard ed affidabili, permettendo ai primi di concentrarsi sulla ricerca piuttosto che sulla costruzione degli strumenti. Non soltanto l'uso di strumenti scientifici diffusi permette di aumentare la quantità di tempo speso dai ricercatori nelle ricerche, ma soprattutto all'ottenimento di risultati ripetibili e comparabili.

Sfortunatamente, la modellazione al computer "...ha trasformato spesso buoni scienziati in pessimi programmatori". La conseguenza di ciò è che studi concettuali di alto livello sono progettati e realizzati in modo non adeguato. In questo contesto è nato il progetto Swarm, dal lavoro di un gruppo di ricercatori dell'Istituto degli Studi sulla

Complessità di Santa Fe, al fine di produrre strumenti che potenzino il lavoro di ricerca, attraverso la collaborazione tra scienziati ed ingegneri. Si tratta di uno strumento software finalizzato alla sperimentazione efficiente, affidabile e riutilizzabile.

In questo paradigma progettuale non è richiesto di realizzare specifiche ipotesi quali la presenza dello spazio, la necessità di fenomeni fisici, una particolare rappresentazione interna degli agenti o particolari interazioni. Con Swarm si può realizzare praticamente qualsiasi modello ad agenti relativo a fenomeni chimici, fisici, economici, dovuti ad interazione delle leggi naturali, antropologiche, di modelli ecologici o relativi alle scienze politiche. Ciò è possibile poiché tutto il codice informatico relativo al comportamento degli agenti deve essere integralmente scritto dallo sperimentatore; Swarm fornisce soltanto tutte le funzioni generiche di gestione dei tempi, dei cicli di funzionamento e di interazione grafica con l'utente.

L'unità fondamentale nel contesto del progetto è l'agente. La simulazione consiste di gruppi di molti agenti che interagiscono tra loro attraverso un meccanismo di notifica di messaggi.

Gli agenti costituiscono gli oggetti fondamentali del sistema; un orologio che funziona ad eventi discreti definisce i processi che essi devono eseguire allo scorrere del tempo. Ogni azione deve essere svolta in un punto preciso della lista di attività ed il tempo scorre in funzione dell'esecuzione di tali attività, per cui la rappresentazione temporale è relativa alle azioni e non al reale tempo di esecuzione della macchina.

Il componente fondamentale che organizza gli agenti del modello è un oggetto chiamato "sciame" (swarm). Uno sciame è una collezione di agenti con un calendario di attività ed eventi che sono notificati agli stessi agenti. Oltre ad essere contenitori di agenti, gli sciami sono essi stessi agenti.

Un agente è costituito da un insieme di regole e di capacità di risposta agli stimoli, ma può essere anche uno sciame: una collezione di oggetti e di "orologi", cioè strumenti software per simulare il trascorrere del tempo nel modello. In questo caso, il comportamento dell'agente è definito dal fenomeno che emerge dall'interazione degli agenti in esso contenuti. Attraverso questa metodologia concettuale è possibile costruire gerarchie di modelli che si contengono l'un l'altro, secondo la teoria dei *building-block*, di cui si è discusso nel capitolo precedente. Di fatto fornisce lo strumento pratico per realizzare ciò che la teoria dei modelli adattivi complessi definisce.

Infine, grazie alla diffusione negli ambienti scientifici, Swarm si propone come "idioma" comune per facilitare lo scambio di modelli tra studiosi, aspetto precedentemente reso molto delicato dall'enorme varietà di "dialetti" esistenti.

6.3 Object Oriented Programming

La maggior parte dei linguaggi di programmazione sviluppatisi nel corso degli anni, era di tipo sequenziale. E' possibile descrivere un problema, o creare un modello, con una logica di tipo top-down, in cui cioè si segue una coerenza piuttosto rigida e lineare. In questi linguaggi, è sicuramente possibile creare *subroutines*, cioè parti di codice da eseguire più volte ed in maniera quasi autonoma, tuttavia per molti scopi della simulazione sociale questo tipo di programmazione si rivela particolarmente ostica.

La struttura logica degli sciami di agenti che interagiscono in base allo scatenarsi di eventi discreti è realizzata utilizzando il linguaggio di programmazione *Objective-C* o in alternativa *Java*, ma soprattutto il paradigma della programmazione ad oggetti. Essa consiste nella definizione di varie classi di oggetti, che possiamo pensare come descrizioni di diverse parti della realtà.

Un oggetto è la combinazione di variabili di stato che ne definiscono le caratteristiche interne ed i metodi, che forniscono ad esso la capacità di agire. In Swarm l'agente è costruito come un vero e proprio oggetto software.

Il vantaggio principale di questa tecnica di programmazione è la capacità di incapsulare il codice relativo a ciascuna unità logica all'interno della definizione della classe, potendo esporre all'esterno soltanto un'interfaccia. Ciò implica che quando l'oggetto deve essere utilizzato è sufficiente conoscere a quali messaggi sa rispondere e che tipo di azioni è in grado di compiere. Il modo specifico con cui lo fa non è interessante e viene nascosto nella definizione di classe. Questo meccanismo è il nodo chiave della filosofia di Swarm: tutti quegli oggetti "tecnici", che svolgono funzioni ripetitive ed esterne da un punto di vista logico del modello, sono forniti dalle librerie di Swarm. Essi svolgono gran parte delle operazioni che non sono inerenti al problema della definizione degli agenti del modello e alle loro regole di interazione. La gestione del tempo, dello scatenarsi di eventi discreti, della rappresentazione grafica dei dati, delle funzioni matematiche e dell'interazione con l'utente (interfaccia grafica) sono delegate a queste ultime.

Un elemento molto interessante è costituito dalle *probe*, ovvero delle sonde, che la libreria apposita di funzioni mette a disposizione. Esse consentono all'utilizzatore del modello, in modo del tutto automatico e trasparente, di mostrare ed eventualmente modificare il contenuto dei dati dei singoli oggetti (agenti) presenti nel modello, in modo interattivo. Questo elemento è innovativo e potente dal punto di vista della simulazione ad agenti, poiché, mentre il software tradizionale nasconde il funzionamento interno all'utente per motivi estetici ed ergonomici, nel caso dell'analisi scientifica di un modello

simulato dal computer è molto importante poter osservare ciò che realmente accade “dietro le quinte”.

Da quanto detto risulta chiaro che la programmazione ad oggetti è la strada ideale per creare al calcolatore modelli basati su agenti, compito reso ancor meno gravoso dalle efficientissime librerie di Swarm, che permettono di sveltire la programmazione e riutilizzare codice creato da altri studiosi.

6.4 Gli errori di coding ed il debugging

Poiché per utilizzare Swarm è necessario creare un codice, cioè utilizzare un linguaggio di programmazione, è necessario prestare particolare attenzione all’insorgere di errori software.

In Axelrod (1997) si fa notare che un nodo fondamentale della ricerca scientifica in questo campo è riposto nell’attenzione alla qualità dei modelli: per poter raggiungere obiettivi quali validità, utilizzabilità e espandibilità, deve essere riposta particolare attenzione in tutta l’attività di ricerca. Questo non include solamente la programmazione, ma anche la documentazione e l’accurata analisi dei dati. Spesso, i risultati che si ottengono con più rapidità sono infatti i meno affidabili.

In Gilbert e Terna (2000) si osserva che il *debugging* è sempre un’attività molto delicata e non si deve mai pretendere di aver prodotto un codice privo di errori. Ciò è valido sempre, ma nel contesto dei modelli basati su agenti diventa un serio problema, poiché i risultati delle simulazioni possono essere inattesi e non si può essere sicuri che essi emergano dalle caratteristiche degli agenti, piuttosto che da qualche *bug* nascosto.

Axtell ed Epstein (1994) affermano a tal proposito che tali problemi software sono difficili da scoprire a causa della natura altamente decentrata dei modelli basati su agenti.

Ancora in Gilbert e Terna (1999), si ricorda che esiste un meccanismo di allarme relativo alla possibile presenza di errori: per esempio, se i risultati variano di molto quando si cambia un singolo parametro che non si ritiene essere critico, la prima spiegazione che si deve ricercare è sicuramente in un possibile errore e non in qualche fenomeno esotico emergente.

Forse l’unica seria metodologia di indagine per la ricerca degli errori consiste nell’attenta analisi interna dei dati di ciascun agente, durante la simulazione ed al termine della stessa. Grazie proprio alle librerie di *probe* che Swarm mette a disposizione tale

attività risulta molto facilitata rispetto all'ipotesi di dover disseminare il codice di istruzioni che mostrino sul terminale i risultati di ciascuna istruzione.

Nell'ambito della simulazione informatica ad agenti il meccanismo di ispezione delle variabili di ciascun è dunque essenziale. La necessità di poter eseguire operazioni di *debugging* è necessaria in ogni campo informatico, ma in questo essa diventa uno dei fattori essenziali e va ricordato che dotare un ambiente di sviluppo di tale funzionalità richiede un grande sforzo.

6.5 La gestione del tempo nelle simulazioni

Uno dei maggiori problemi, quando si affrontano problemi di simulazione al calcolatore, consiste nel gestire il trascorrere del tempo. Risulta infatti molto complesso, nei linguaggi di programmazione tradizionale, sincronizzare gli eventi in modo tale da non creare sovrapposizioni o inesattezze. Solitamente, per ottenere questo, si ricorre a serie di cicli, spesso annidati, in cui si compiono le varie azioni.

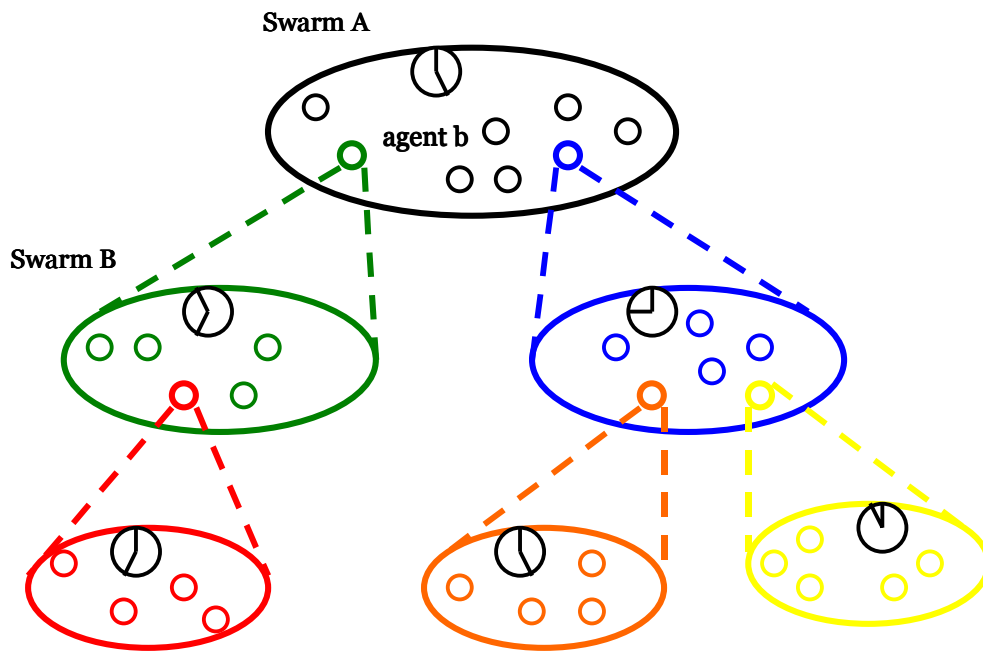
Quando però gli agenti diventano molti, e la simulazione assume rilevante complessità a causa di azioni sempre più perfezionate, questa tecnica diviene particolarmente difficile da implementare. Swarm risolve in modo molto elegante questi problemi.

In Lin, Tan e Shaw (1996), si afferma che uno sciame non è soltanto una collezione di oggetti ha anche un orologio. Ogni sciame contiene un orologio indipendente dagli altri sciami e allo stesso tempo tutti gli agenti dello sciame fanno riferimento allo stesso orologio. Una struttura gerarchica di sciami è essa stessa una struttura gerarchica di orologi, come si può osservare nella figura che segue.

Tutte le attività sono eseguite in riferimento ad un orologio globale, sebbene durante l'esecuzione differenti sciami hanno la possibilità di non compiere nessuna operazione rispetto alla sincronizzazione globale.

La gestione ricorsiva del tempo è talmente flessibile da consentire sia una gestione totalmente accentrata e sincronizzata dell'elenco di operazioni da svolgere, sia una gestione parallela delle attività.

Durante l'esecuzione il nucleo di Swarm scorre la lista degli orologi per informare gli agenti di svolgere le operazioni previste, mandando loro un messaggio nel momento opportuno.



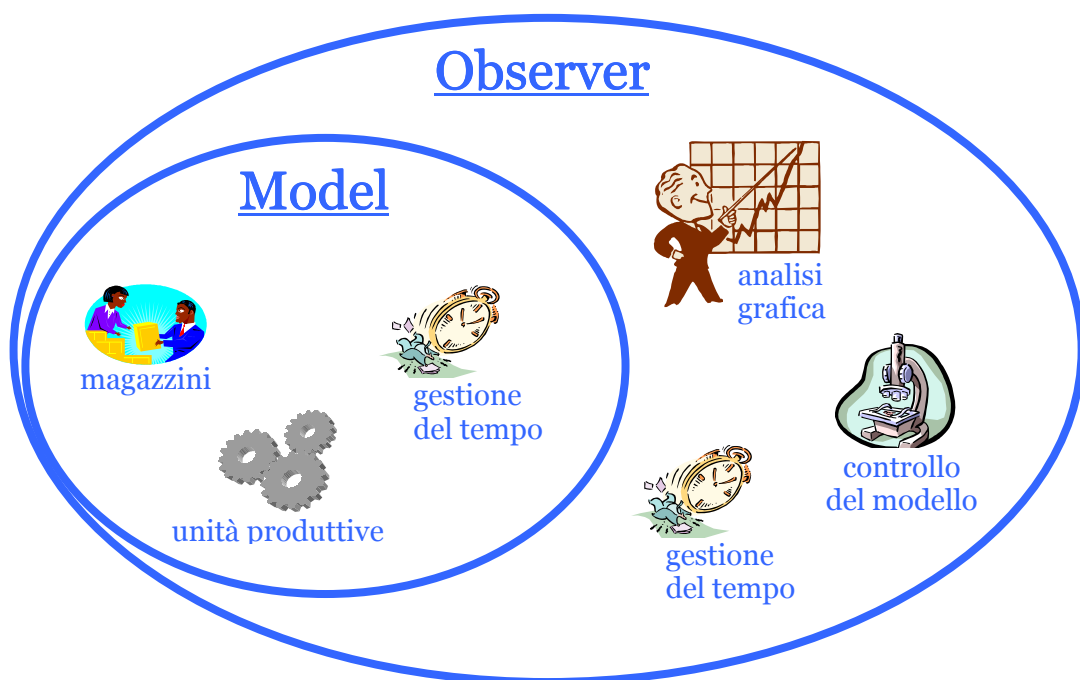
6.6 Costruzione di un modello in Swarm

La realizzazione di un modello di simulazione secondo la metodologia introdotta dagli autori di Swarm prevede la realizzazione di tre passi successivi (questa è la filosofia di Swarm):

1. La prima operazione da compiere è la scrittura del codice relativo alle classi che corrispondono alle categorie di agenti previsti dal modello.
2. La seconda operazione prevede la scrittura della classe denominata *ModelSwarm*, la quale contiene tutto il codice relativo alla costruzione ed alla gestione da un punto di vista aggregato degli agenti del modello. Essa corrisponde al modello vero e proprio ed è dotata della descrizione della sequenza di azioni che ciascun agente dovrà compiere in ogni ciclo temporale.
3. L'ultima operazione consiste nella realizzazione della classe *ObserverSwarm*, composta da un oggetto che si occupa di esaminare il comportamento dell'agente *ModelSwarm* nel suo comportamento aggregato. Ciò significa che in questa classe

vengono costruiti il modello di simulazione, tutti gli oggetti di analisi grafica e la lista delle azioni di ciascun componente che “osserva” il comportamento di un oggetto, sia esso il modello o un singolo agente.

Questo schema è assolutamente “pulito” da un punto di vista logico, perché viene richiesto di costruire un oggetto che realizza e gestisce le interazioni tra gli agenti, ed un altro oggetto che incorpora il primo e di occupa di effettuare le operazioni di analisi statistica sui dati aggregati che emergono dal modello.



Il model costruisce un numero di agenti pari al parametro opportuno delle categorie previste. Un oggetto detto *schedule* gestisce lo scorrere del tempo e lo scatenarsi relativo degli eventi. Nel suo complesso questo oggetto dà vita al modello e ne contiene tutte le componenti necessarie al funzionamento. Al fine, però, di permettere allo sperimentatore di osservare e comprendere ciò che accade durante la simulazione il *model* diventa un semplice oggetto che, attraverso la filosofia dei *building block*, è incorporato da un altro oggetto detto *observer*, il quale è in grado, attraverso le sonde, di osservare ciò avviene all'interno del modello e di elaborare i dati per l'analisi grafica o di altro tipo. Anche quest'ultimo è dotato di un oggetto *schedule* che gestisce il succedersi degli eventi, poiché l'osservazione dei dati può avvenire con una frequenza diversa da quella che è utilizzata dal modello. È importante notare, però, che gli oggetti che svolgono la funzione

di “orologi informatici” all’interno del sistema Swarm sono tutti sincronizzati e quindi gli eventi si scatenano secondo una sequenza totalmente controllabile da un punto di vista aggregato del modello dallo sperimentatore.

6.7 Le librerie di Swarm

Come si legge in Minar et al. (1996), le librerie di Swarm svolgono due funzioni principali. Esse sono una serie di classi che i costruttori di modelli possono utilizzare direttamente, creando esemplari di oggetti derivati da esse. Per la maggior parte degli oggetti, in particolar modo quelli particolarmente tecnici, come strutture di dati, è probabile che tutti gli utenti utilizzino questo semplice metodo.

Oltre a questo utilizzo, le librerie di Swarm possono essere utilizzando creando delle sottoclassi, specializzate in particolari funzioni, in base al tipo di modello che si desidera costruire. Entrambe queste modalità d’uso delle librerie di Swarm sono importanti, e Swarm stesso è stato creato per renderle più immediate possibile.

La maggior novità all’interno di Swarm, è l’esistenza di librerie specifiche per l’uso simulativo. Le librerie *swarmobject*, *activity* e *simtools* sono il centro del paradigma di Swarm, orientato ai modelli. La prima contiene le classi fondamentali su cui sono basati gli oggetti presenti nelle simulazioni effettuate con Swarm: al centro di essa vi sono due classi, *SwarmObjects*, che definisce l’interfaccia base per la gestione della memoria, come anche il supporto delle probe, e *Swarm*, da cui derivano il *Model* e l’*Observer*.

La libreria *activity* contiene il cuore del meccanismo simulativo, le strutture dello *schedule*, il supporto per l’esecuzione e la gestione del tempo.

L’ultima libreria, specifica per scopi simulativi, è *simtools*, un insieme di classi necessarie per costruire le simulazioni: essa contiene classi per controllare l’intero modello. Esistono due modalità operative: una interamente grafica, per l’esplorazione interattiva ed un *batch mode*, per la raccolta e l’elaborazione dei dati. Questa libreria contiene classi in grado di generare insiemi di dati statistici, disegnare grafici e così via.

Oltre a quelle utili direttamente per le simulazioni, Swarm contiene anche librerie scritte per i modelli, ma utilizzabili anche per scopi diversi. Librerie quali *defobj*, *collections*, *random* e *tkobjc* incorporano oggetti utili per scrivere qualunque tipo di software. Le prime due sono utili per costruire programmi *Object Oriented* di tipo

generico; *collections* permette di rintracciare gli oggetti in un sistema complesso attraverso mappe, liste e così via. La libreria *random* offre all'utilizzatore una serie molto ampia di generatori di numeri casuali, assolutamente essenziali per la qualità di un software. Poiché la libreria stessa è *Object Oriented*, è possibile avere diversi tipi di generatori in uno stesso programma, in modo da simulare comportamenti diversi di agenti, tutti comunque di tipo casuale. Infine, la libreria *tkobjc* è un'interfaccia grafica molto potente, basata sul *toolkit Tcl/Tk*, che a sua volta si appoggia su *X-windows*. Attraverso questa classe è possibile creare bottoni, istogrammi, grafici e schermi.

Esistono anche librerie create per determinati tipi di simulazione: per esempio vi sono prototipi di oggetti per gestire gli spazi in due dimensioni, algoritmi genetici e reti neurali. Proprio questo è il campo in maggiore evoluzione: costruiti gli oggetti più generici, diventa infatti utile crearne di specifici, per particolari scopi.

6.8 Linguaggio Java in Swarm

Swarm è stato scritto originariamente utilizzando il linguaggio di programmazione *Objective-C*, un dialetto del C, diffuso inizialmente soltanto nell'ambito della piattaforma *Unix*. Questo linguaggio si propone di essere un'alternativa più aderente al paradigma della programmazione ad oggetti rispetto al più diffuso C++. Infatti, l'*Objective-C* utilizza uno stile parzialmente ereditato da *Small-Talk*, il primo linguaggio ad essere completamente pensato ad oggetti. L'impossibilità di eseguire Swarm nelle macchine dotate del sistema Windows costituiva un limite alla diffusione dello strumento, considerando che Swarm, come si è detto, si propone di essere uno strumento di riferimento nella scrittura dei modelli di simulazione. La sua diffusione ha seguito due tappe. La prima conquista fu raggiunta quando fu rilasciato il compilatore *Objective-C* per il sistema operativo Windows. La seconda è rappresentata dal recente rilascio di una versione di Swarm nel linguaggio Java.

La caratteristica più rilevante di Java è l'alta portabilità del software prodotto: infatti esso promette di essere l'unico linguaggio, nel panorama informatico, ad essere indipendente dalla piattaforma in cui è eseguito. Questo elemento consente in prospettiva di diffondere il modello di simulazione con grande facilità all'interno della comunità di studiosi, eliminando il fastidioso compito di ricompilare il codice nelle diverse versioni Windows/Unix. Inoltre, è un linguaggio più diffuso poiché deriva sintatticamente dal C++.

È necessario notare che la scelta tra i due linguaggi di programmazione non è definitivamente risolta a favore di questo. Esistono vari motivi che impongono un'accurata valutazione tra le due opportunità. *Objective-C* è oggi più diffuso all'interno della comunità di utilizzatori di Swarm: molti ricercatori hanno investito molto tempo nell'apprendimento del C ad oggetti e sono restii a mettere in discussione le conoscenze acquisite se non in presenza di rilevanti vantaggi della nuova soluzione.

Tutti i modelli già scritti dovrebbero essere inoltre tradotti e poiché il compilatore *Objective-C* produce codice binario specifico per la piattaforma in cui è eseguito, permette di ottenere alte prestazioni in termini di velocità. Java, al contrario, è un linguaggio in parte interpretato, quindi più lento.

Questo può rappresentare un problema se si pensa che il codice relativo ai modelli Swarm richiede molti cicli iterativi di calcolo. È necessario notare, però, che Java è dotato di un compilatore *just-in-time* che sfrutta proprio i cicli ripetuti di istruzioni per aumentare le prestazioni: per questo la differenza in termini di velocità di esecuzione non è un fattore determinante.

La motivazione che ha determinato la scelta di Java per questo lavoro è legata al fortissimo sviluppo dell'Internet, media strettamente legato al Java per ovvi motivi; inoltre, la portabilità su diversi sistemi renderà possibile l'espansione del modello anche da parte di altri sviluppatori.

Inoltre, nel caso si decidesse di utilizzare dati reali per la simulazione, sarà possibile interfacciare più facilmente il programma con dati presenti in rete.

È utile ricordare che Swarm mette a disposizione una libreria di gestione dei dati denominata HDF5, un robusto motore di gestione di dati per le elaborazioni scientifiche. Essa è assai complessa e non risolve il problema di dovere intervenire manualmente nel trasferimento dei dati dal database al modello.

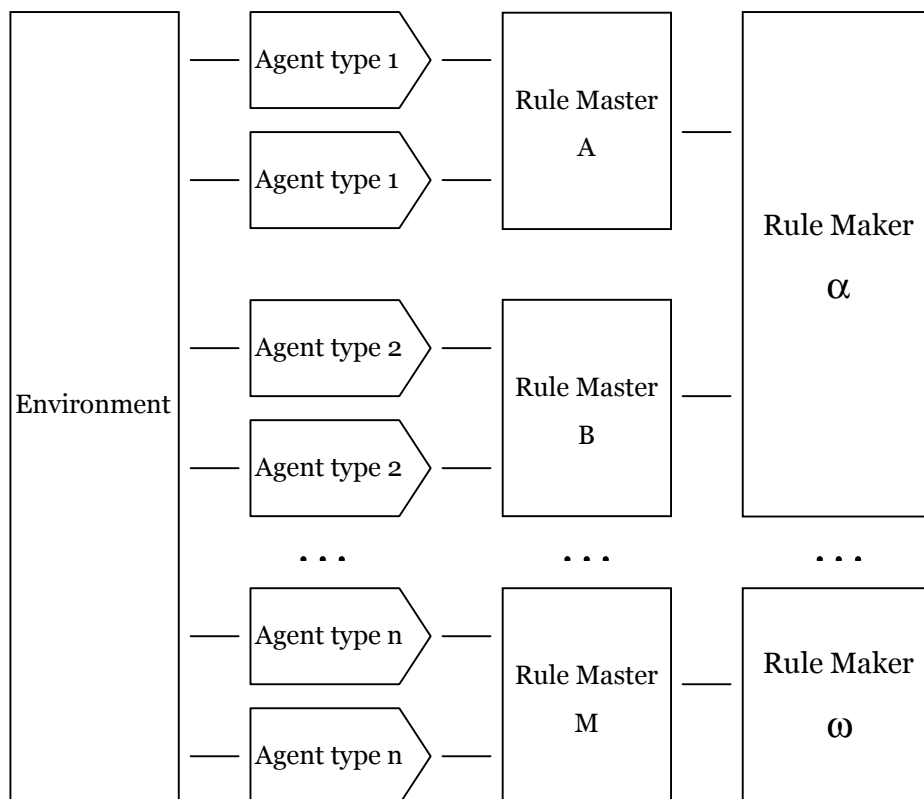
Java è dotato nella sua libreria standard di potenti funzioni di accesso e manipolazione dei dati: `java.sql.*`. Queste API permettono di accedere alla fonte dei dati utilizzando la tecnologia di accesso ODBC che consiste in uno strato software che interpreta e traduce le richieste al database da un linguaggio universale, l'SQL (*Structured Query Language*), nelle opportune chiamate al tipo di database in cui i dati sono memorizzati.

Una ulteriore caratteristica interessante del linguaggio Java è la capacità di creare le *applet*. Un oggetto Java, può essere eseguito all'interno di un browser, con un minimo intervento da parte dell'utente. Questa caratteristica apre le porte ad un nuovo sviluppo nelle simulazioni di modelli sociali: si configura la possibilità di realizzare modelli aperti che sono eseguiti su un server, e possono interagire con utenti remoti di tipo artificiale

(oggetti software), ma anche umani. Inoltre, si è scelto questo linguaggio per la sua maggiore prospettiva di evoluzione, la capacità di gestire oggetti come le stringhe di caratteri con estrema semplicità, la capacità di produrre la documentazione tecnica del software in formato HTML, semplicemente annegando i commenti all'interno del codice sorgente, la sua innata integrabilità con l'internet e la notevole dotazioni di funzioni sofisticate all'interno della libreria standard.

6.9 Lo schema E.R.A.

Spesso, costruendo un modello di certe dimensioni, diviene problematico tener sotto controllo tutte le parti. Secondo Gilbert e Terna (2000), passando da semplici modelli a risultati complessi, sorgono problemi, i quali suggeriscono che un ruolo cruciale per l'utilità e l'accettabilità degli esperimenti, viene giocato dalla struttura con che descrive i modelli stessi. Per questo motivo, si introduce uno schema generale, che può essere utilizzato per costruire simulazioni basate su agenti. La struttura originale dello schema *Environment-Rules-Agents*, proposto dagli autori è descritto nella figura seguente.



Lo schema prevede di collocare gli oggetti in tre distinte sezioni. L'ambiente (*Environment*) è un componente che contiene le informazioni condivisibili da tutti gli

agenti del modello. Il “colloquio” diretto tra essi è realizzabile nello schema Swarm, ma quando il modello presenta molte tipologie di agenti si rischia di non riuscire a comprendere l'intero schema delle relazioni tra le diverse componenti, poiché sono distribuite in punti diversi del codice. Gli oggetti che dell'ambiente svolgono una funzione di intermediazione nella comunicazione tra agenti e mette a disposizione di tutti le informazioni globali.

Lo strato denominato *Agent* contiene la descrizione di tutti gli agenti del modello e corrisponde alla rappresentazione logica dello stesso. Gli agenti però 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 in oggetti esterni all'agente stesso. La sezione *Rules* contiene proprio gli oggetti in grado di applicare le regole di comportamento degli agenti ed eventualmente di modificarle. Sarà possibile inserire regole originali, ma anche meta-regole, vale a dire regole in grado di modificarne altre: per questo motivo, l'oggetto *RuleMaster*, cioè il depositario delle regole, 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.

Attraverso questo schema la variazione dei meccanismi cognitivi o adattivi degli agenti non richiede la modifica dell'infrastruttura del modello. Le regole di interazione sono mantenute costanti, è sufficiente modificare o sostituire gli esecutori ed i produttori delle regole per cambiare il volto degli agenti senza apportare modifiche al loro codice informatico.

Questo schema di progetto produce vantaggi enormi nella manutenzione e nella comprensibilità del codice, permettendone soprattutto una notevole espandibilità, senza dover spendere tempo a comprendere difficili interazioni tra agenti, mascherate in qualche oscura parte del software. Inoltre, se si volessero solo modificare alcune regole alla base del funzionamento degli agenti, basterebbe editare il *RuleMaster* corrispondente, senza modificare il codice dei singoli oggetti coinvolti nella simulazione.

UN MODELLO DI VIRTUAL ENTERPRISE: JVE

7.1 Introduzione al modello

Un modello deve essere il più possibile semplice, ma allo stesso tempo deve poter riprodurre la realtà, in modo da fornire risultati utilizzabili e studiabili. Nel nostro caso, ci troviamo a creare un modello per un'azienda di tipo tradizionale, facente parte del settore produttivo manifatturiero. E' dunque necessario creare un insieme di unità produttive, ognuna delle quali è in grado di costruire un particolare componente, l'insieme dei quali costituirà l'oggetto finito da vendere al cliente finale. Si rende necessario simulare l'arrivo di un ordine per un prodotto finito, non predeterminato, composto da sequenze sempre diverse e variabili di componenti.

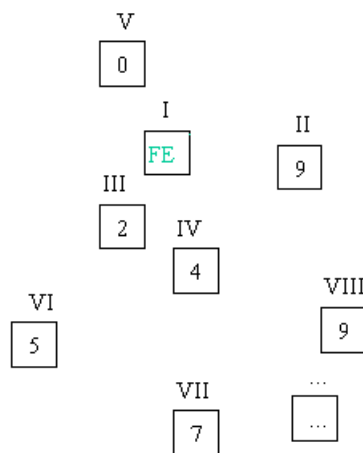
L'ordine deve dunque circolare tra le varie unità, secondo il concetto di catena di montaggio, ancora molto diffuso nelle aziende organizzate ad isole di produzione. Poiché il modello deve essere creato avvalendosi di un calcolatore, un'efficace similitudine è quella tra prodotto finito e numeri composta da diverse cifre. Il nostro scopo è dunque quello di "costruire" numeri di un numero predefinito di cifre: essi costituiscono il nostro prodotto finito ed ogni cifra è il componente singolo. Ogni cifra, componente il numero, è prodotta singolarmente e solo in seguito, oppure sequenzialmente secondo l'idea di catena di montaggio, avviene l'assemblaggio. Possiamo dunque considerare le singole cifre come componenti del prodotto; per questo motivo, consideriamo importante l'ordine e la posizione in cui le cifre devono essere assemblate: ad ogni sequenza di cifre, corrisponde un prodotto diverso.

	Prodotti
	14372
	41909
	. . .

7.2 Le unità produttive

Le singole cifre vengono costruite dalle unità produttive, ognuna delle quali è in grado di svolgere solamente un processo produttivo, corrispondente, appunto, ad un singolo componente (cifra). Ci troviamo di fronte dunque ad un'azienda con unità produttive altamente specializzate, ma integrate tra loro. E' chiaro dunque che le unità produttive devono esistere in numero maggiore, o al limite uguale, a quello delle cifre esistenti. Per catalogare e differenziare le singole unità produttive, esse vengono contraddistinte da un numero espresso in cifre romane, che non coincide necessariamente con l'identificativo del componente che esse sono in grado di produrre. Il numero romano identifica l'unità produttiva all'interno delle “*matrici di legami*”, di cui si parlerà più avanti. Non possono esistere due unità produttive con lo stesso indice espresso in numeri romani.

Ogni unità produttiva, inoltre, è contraddistinta da un numero arabo, che indica quale processo essa è in grado di svolgere, cioè quale cifra (componente) essa produrrà. Possono esistere più unità produttive in grado di produrre la stessa cifra, dunque con uguale indice espresso in numero arabo.



7.3 L'unità “Front End”

Una unità produttiva, anziché costruire un componente, funziona da “*front end*” verso il mercato.

L'unità “front end” può operare in molti modi differenti: essa deve raccogliere gli ordini dei prodotti finiti, cioè dei numeri a cinque cifre. A questo punto le possibilità sono:

1. Essa può trasmettere l'ordine, indipendentemente dalla posizione delle singole cifre, contemporaneamente alle singole unità produttive capaci di costruire i componenti in questione. Quando i componenti singoli saranno pronti, verrà data comunicazione all'unità "front end", la quale è depositaria della posizione che le cifre dovranno avere per la costituzione del prodotto finito desiderato. Questa invierà il dato all'unità di assemblaggio, che, ricevuti i singoli componenti, procederà alla fase di montaggio finale.

2. Essa può altresì comunicare solo con l'unità capace di produrre la prima cifra desiderata, dando ad essa anche indicazione dell'intera sequenza di cifre. Sarà poi quest'ultima che, ultimata la costruzione del componente a suo carico, passerà l'ordine alla seconda unità produttiva in sequenza, insieme all'informazione sull'ordine delle cifre rimanenti, e così via. E' chiaro che, in questo modo, si vengono ad allungare i tempi, poiché ogni unità in grado di un processo produttivo seguente dovrà aspettare il completamento di quello precedente per poter iniziare la costruzione. Tuttavia, in questo modo, il prodotto finale sarà già nella sequenza desiderata, e avremo uno scenario del tipo "catena di montaggio".

3. Come nel precedente caso, essa può dare comunicazione solo alla prima unità produttiva della sequenza da produrre, ma questa unità potrebbe non attendere la fine della costruzione del proprio oggetto, per passare l'ordine alle seguenti unità.

4. Essa potrebbe tener conto dei diversi tempi di produzione, e delle code esistenti in sede di costruzione dei singoli componenti (grazie ad un sistema informativo). A questo punto, per cercare di eliminare i tempi morti, potrebbe inviare l'ordine di produrre le singole componenti in primo luogo alle unità produttive con tempi di costruzione più lunghi, e solo quando queste sono in fase avanzata di completamento, a quelle con tempi più brevi, in maniera intelligente. Questo modello potrebbe costituire una modificazione di quello al punto uno, derivante dall'introduzione di un sistema informativo integrato.

Infine, potrebbe esistere un'altra unità, che potremmo definire di staff, a cui viene passata semplicemente la gerarchia delle componenti da produrre, mentre gli ordini vengono inviati singolarmente alle unità produttive incaricate. Ogni qualvolta una

componente sia completata, essa viene inviata a questa unità, che provvederà all'assemblaggio secondo l'ordine ricevuto inizialmente.

7.4 Tempi di produzione

Dato quanto detto in precedenza, risulta chiaro come le singole unità produttive abbiano tempi diversi per la costruzione dei singoli componenti (cifre). Questa problematica è inserita per conferire maggior realismo al modello, e per creare code in certi punti del sistema. Oltre ai tempi di produzione veri e propri, esistono anche dei cosiddetti tempi di movimentazione, che consistono nel tempo richiesto all'unità produttiva per mettersi in moto. Il tempo di movimentazione, sommato a quello richiesto dalla costruzione vera e propria, costituirà il tempo totale richiesto per la produzione dei singoli componenti.

Poiché esiste questa differenza nei tempi di produzione delle singole componenti, alcune unità produttive potrebbero essere duplicate, cioè più unità produttive, contraddistinte comunque da numeri romani diversi per mantenere la loro identità, possiedono lo stesso processo produttivo (stesso numero arabo). E' chiaro che le unità da duplicare dovrebbero essere quelle con tempi produttivi totali più lunghi, o quelle che costruiscono le componenti più spesso richieste. A tal proposito si può scegliere se dare una distribuzione normale alla probabilità di estrazione casuale delle cifre che compongono un prodotto finito, oppure di prediligerne alcune, che potrebbero avere la funzione di componenti comuni di ampio utilizzo.

Possono poi essere introdotti tempi di consegna del prodotto finito, anch'essi diversi sulla base delle singole cifre che lo compongono (si potrebbe per esempio ipotizzare la fragilità di un dato componente, che si trasferirebbe sull'intero prodotto, il quale dovrebbe dunque essere trasportato con maggiori cautele).

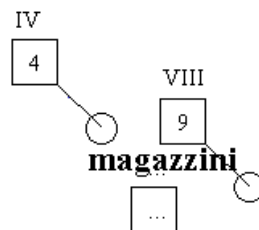
7.5 I magazzini

Un ulteriore modo per ridurre le code e i tempi morti, consiste nell'istituzione di magazzini, in cui verrebbero depositate le singole componenti prodotte in eccedenza. In questo caso, si dovrebbe decidere l'organizzazione delle entrate/uscite in magazzino (libero, FIFO, LIFO, ecc.). Con l'esistenza di uno o più magazzini sicuramente si

ridurrebbero i tempi necessari alla costruzione del prodotto finito, però si alzerebbero considerevolmente i costi, rispetto alla produzione “*just in time*”.

Potrebbero anche esistere più magazzini, magari anche uno per ogni componente, o comunque magazzini mono-componente solo per le cifre di più ampio utilizzo. In questo modo si avrebbe un ulteriore snellimento, a fronte però di costi ancora più elevati.

Nel caso dell’esistenza di un magazzino, l’unità front end descritta precedentemente, prima di passare gli ordini alle singole unità produttive, dovrebbe interrogare il sistema informativo, per verificare se certe componenti richieste sono già state prodotte in precedenza. In questo caso, esse possono essere semplicemente prelevate dal magazzino ed inviate all’assemblaggio, o semplicemente inviate in catena di montaggio, a seconda del procedimento di funzionamento scelto per l’unità front end.



7.6 Gli ordini ed il mercato

Gli ordini, provenienti dal mercato esterno, sono in numero variabile ogni giorno; questo numero potrebbe essere scelto a caso, oppure potrebbe seguire dei trend stagionali. Nel secondo caso potrebbe essere inserito un sistema di previsione basato su reti neurali, in grado anche di tener conto di quali sono le cifre più richieste all’interno dei singoli ordini. Con un tale sistema, l’azienda potrebbe “decidere” di chiudere momentaneamente alcune unità produttive duplicate, che dunque non genererebbero costi, nei periodi in cui si prevede scarsa produzione di quel determinato componente, oppure potrebbe decidere di produrre per il magazzino, prima di un periodo in cui si ritiene ci saranno forti richieste di quella data cifra. Si possono introdurre mezzi di trasmissione degli ordini basati su sistemi informativi integrati, che rendono più rapido il complesso della comunicazione, sia tra azienda e mercato, che all’interno dell’azienda stessa.

7.7 La computazione dei costi

Ogni unità è in grado, al proprio interno, di eseguire un “accounting” dei costi generati. Esisteranno costi fissi, dovuti alla semplice esistenza e manutenzione dell’unità in questione, e costi variabili, dipendenti dal numero delle componenti prodotti dalle singole unità. Possono poi esistere dei costi aggiuntivi, quale quello di un eventuale magazzino integrato, dedicato ai soli componenti prodotti dall’unità. Ovviamente, i costi, siano essi fissi o variabili, sono diversi per ogni unità: ci saranno dunque componenti più o meno costose da produrre e altresì esisteranno unità che per il proprio mantenimento necessiteranno cure maggiori, e che dunque genereranno costi più ingenti. Si possono introdurre guasti casuali alle singole unità produttive, allorché la riparazione sarà imputata ad essa, e genererà un costo che l’azienda potrà ammortizzare aumentando il prezzo della componente prodotta.

I costi dei singoli componenti si riflettono poi ovviamente sul prezzo del prodotto finito, che incorporerà anche il costo di assemblaggio delle componenti ed il valore aggiunto.

I costi, tuttavia, non sono solamente fisici. Per l’azienda, costituisce un costo anche l’insoddisfazione del cliente. E’ necessario dunque badare ai costi fisici, senza tuttavia perdere di vista i tempi di consegna: se questi infatti si innalzano troppo, rendono il cliente insoddisfatto, e dunque portano un danno derivato all’azienda (il non riacquisto). Inoltre, si potrebbe immaginare, proporzionalmente al ritardo, una diminuzione del prezzo del prodotto finito, una sorta di “penale” per non aver consegnato il prodotto nel momento concordato. Questo si riflette soprattutto nelle strategie di gestione dei magazzini, secondo lo schema seguente:

Ulteriore costo è poi costituito dalla consegna. Anche qui, come già per i tempi, i costi possono variare a seconda della tipologia del prodotto finito (a sua volta derivante dalle diverse componenti impiegate).

7.8 Il modello in Swarm

In Swarm, i prodotti devono essere configurati come oggetti, contenenti dunque diverse informazioni sotto forma di variabili, matrici e vettori. Le informazioni contenute devono essere:

1. **il codice prodotto**, un numero di cinque cifre che identifica e costituisce il prodotto stesso. Nel modello, infatti, ogni cifra è la componente stessa, e la posizione da essa occupata è determinante. L'informazione potrebbe essere contenuta in un vettore a cinque posizioni, in modo da avere un indice identificativo per l'ordine delle cifre stesse.
2. **Stato di completamento**, che indica quante componenti del prodotto finito siano già state costruite. Potrebbe anch'esso essere costituito da un vettore a cinque posizioni, corrispondenti alle cifre da produrre, contenente "0" se la componente non è ancora stata prodotta, oppure "1" se questa è già completata.
3. **Posizione**. L'implementazione di questa variabile cambia a seconda del modo di funzionamento scelto per il front end. Nel caso la nostra produzione funzioni a catena di montaggio, la posizione indicherà presso quale unità produttiva si trova il nostro prodotto. Nel caso invece di comunicazione contemporanea a tutte le unità produttive, la variabile posizione potrebbe assumere l'aspetto di un vettore a cinque posizioni, nelle cui caselle sono inserite le percentuali di completamento dei singoli componenti.
4. **Costo**, sia quello totale del prodotto, che quello delle singole voci che lo compongono (componenti, assemblaggio, eventuali costi aggiuntivi, trasporto, ecc.)

Le unità produttive sono agenti, in grado di ricevere input ed inviare output. Per compiere ogni azione, esse interrogano un RuleMaster, il quale a sua volta interagisce con un RuleMaker, secondo lo schema ERA. Il tempo ed i costi di produzione sono valori intrinseci e caratteristici delle singole unità produttive. I magazzini interni alle unità possono essere configurati come agenti autonomi, di una diversa fattispecie, oppure

semplicemente come matrici interne all'agente "unità produttiva" in cui vengono immagazzinate le componenti uguali.

Il magazzino indifferenziato, se esistente, va invece configurato come un agente con le proprie regole, scelte in base al tipo di funzionamento (FIFO, LIFO, a schema libero, ecc).

Per gestire al meglio i flussi e le comunicazioni tra le diverse parti del modello, possono essere utilizzate le "matrici di legami".

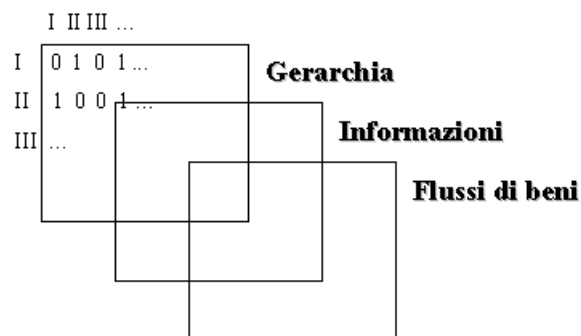
Si tratta di tre tabelle a doppia entrata, che hanno sulle due dimensioni gli indici che identificano le unità produttive, e nelle caselle possono avere valori di "0" od "1".

Le matrici indicano i legami di:

1. **Gerarchia:** indica quali agenti possono comunicare gli ordini a chi. Assegnando alle righe il significato "da" e alle colonne il significato "a" possiamo interpretare ogni incrocio come la possibilità o l'impossibilità dell'unità su quella riga di passare un ordine all'agente sulla colonna considerata. Per esempio, l'unità "Front End" potrà comunicare ordini a tutte le singole unità produttive, mentre queste ultime potranno o meno comunicare tra loro in base alla tipologia di trasmissione degli ordini scelta. Data questa matrice, sarà interessante verificare come l'efficienza dell'azienda può variare, in presenza di diverse gerarchie.
2. **Scambio di informazioni:** in maniera analoga alla precedente, questa matrice opera sui flussi informativi, indicando chi può mandare informazioni, e verso chi. Per esempio, le unità produttive possono essere contattate da altre unità, o dal front end, per comunicare loro in anticipo quante componenti saranno da produrre, cioè quante sono già in coda d'ordine. Inoltre, informazioni possono anche essere le previsioni sulla produzione, nel caso di un previsore per mezzo di reti neurali.
3. **Flussi di beni:** questa matrice indica chi può inviare il componente a chi. La struttura sarà strettamente dipendente dal meccanismo scelto per il funzionamento dell'azienda, e varierà dunque se si predilige la catena di montaggio, o se si inseriscono i magazzini, o ancora se sono presenti una o più unità preposte all'assemblaggio. In quest'ultimo caso, per esempio, le unità assemblaggio potranno ricevere componenti da ogni unità produttiva, ma non

potranno inviare. Nel caso in cui si scelga la catena di montaggio, invece, tutte le unità potranno scambiare beni tra loro. Il front end non può ovviamente inviare componenti alle unità, in quanto non produce. Le unità possono invece inviare componenti ai magazzini, i quali possono inviare all'unità assemblaggio, se esistente, o anche alle unità produttive stesse, in caso di catena di montaggio. La tabella dei flussi di beni può essere anche integrata con i tempi necessari al loro transito.

La sovrapposizione e la combinazione delle tre matrici di cui sopra, determinerà l'organizzazione dell'azienda virtuale. Questo ci permetterà di vedere gli effetti di eventuali perturbazioni nella comunicazione delle informazioni, o variazioni nella gerarchia, o ancora in guasti nella catena dei flussi di beni. Potrà anche essere messo in evidenza come un adeguato sistema informativo possa migliorare questi aspetti, aumentando le possibilità di scambio di informazioni, e rendendo il sistema più snello, ma anche meno gerarchico ed interpretabile.



Sempre sotto il profilo tecnico, può poi esistere una sorta di router, utile soprattutto tra l'unità front end ed il resto del modello, utilizzato per raccogliere ordini in apposite liste, ed indirizzarli verso i destinatari prescelti nella maniera più lineare possibile. Il router genera ogni giorno un numero casuale (oppure scelto dall'utente) di messaggi "propagate" verso gli agenti. Attraverso il router possono anche essere creati, artificialmente, dei colli di bottiglia dovuti all'impossibilità di inviare un numero troppo elevato di "propagate" al giorno, per quanto riguarda gli ordini e le informazioni. Aumentando i propagate si diminuiscono i colli di bottiglia, ma si aumentano i costi di gestione aziendale.

Attraverso il router, lo Schedule riproduce gli effetti:

1. di simultaneità, in quanto tutti eseguono il messaggio propagate ad uno stesso tic dell'orologio, anche più volte nello stesso giorno
2. di difficoltà di collegamento, creando i colli di bottiglia di cui dicevo sopra
3. Le sequenze per le comunicazioni sono invece contenute nelle matrici.

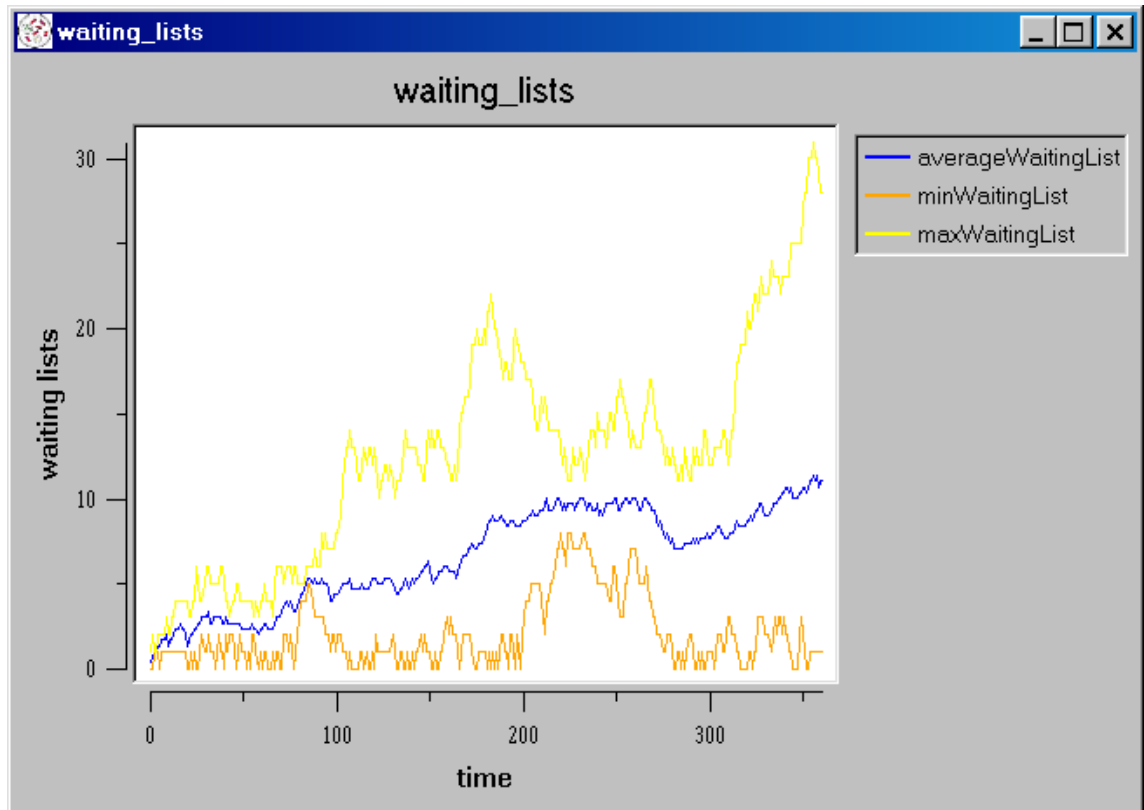
Potrebbe anche esistere un agente, preposto al solo accounting, in grado di mantenere una sorta di contabilità aziendale, indicata come “costi” e “ricavi”. Questo potrebbe ricevere informazioni dalle singole unità produttive per quanto riguarda i costi, e dai prodotti finiti per quanto riguarda il prezzo. Questo agente potrebbe fornire un'indicazione della salute economica dell'azienda, al variare di parametri quali il flusso informativo e quello di beni.

7.9 La formazione di code

Già durante la prima fase di sviluppo del modello di azienda virtuale, la presenza contemporanea di un numero ridotto di unità produttive, per esempio tre e di un prodotto potenzialmente costituito da più componenti, provoca l'insorgere di code e ritardi.

Dopo pochi passi della simulazione, infatti, si creano delle liste di attesa contenenti anche trenta ordini in coda per ogni unità produttiva.

Questo è verosimile, in quanto nessuna azienda attende di avere assoluta necessità di un componente, per costruirlo o comunque per procurarselo: in questo modo si creerebbero infatti dei ritardi assolutamente insostenibili per la produzione, che porterebbero a non rispettare i tempi di consegna e costituirebbero dunque un costo per l'azienda stessa.



Molta attenzione è ormai rivolta al concetto del “Just in Time”, cioè un’organizzazione della produzione tale da permettere la riduzione delle scorte, e da rendere più snello e rapido possibile il percorso che va dall’ordine al prodotto finito, riducendo al massimo l’intervallo di programmazione.

Nonostante questo l’esigenza di un magazzino è comunque sempre presente, anche se l’attenzione è focalizzata maggiormente sull’organizzazione dello stesso, piuttosto che sulla pura capienza.

La presenza di magazzini nel modello è dunque un qualcosa di concreto, che trova corrispettivi nella realtà aziendale.

7.10 Magazzini nel modello VE

La programmazione ad oggetti concede la possibilità di creare con semplicità una moltitudine di unità di notevole complessità.

Nonostante ciò, in questa prima fase del modello, è preferibile inserire agenti semplici, in modo da rendere più lineare l’interpretazione dei risultati.

Per questo motivo, i magazzini sono fondamentalmente dei contatori, che vengono incrementati ogni volta che viene inserito un componente, e decrementati quando l'oggetto è utilizzato per la produzione.

Per decentrare il più possibile l'informazione, ogni unità produttiva ha un proprio magazzino, in cui vengono depositati i componenti da essa prodotti.

I magazzini sono in grado di rispondere a quattro messaggi fondamentali, rappresentati da metodi in JAVA Swarm:

1. **getWarehouseNumber()** : restituisce il numero del magazzino. Questo permette ovviamente di identificare il magazzino, e di associarlo all'unità produttiva corrispondente.
2. **getCounterValue()** : restituisce il numero di pezzi contenuti all'interno del magazzino. Fondamentalmente, viene letto il contatore all'interno della struttura del magazzino.
3. **addCounterValue()** : incrementa il contatore. Questo avviene ogni volta che si produce per il magazzino, e si introduce dunque un nuovo componente.
4. **subCounterValue()** : decrementa il contatore. E' l'operazione inversa alla precedente, che viene eseguita quando si utilizza un componente contenuto nel magazzino.

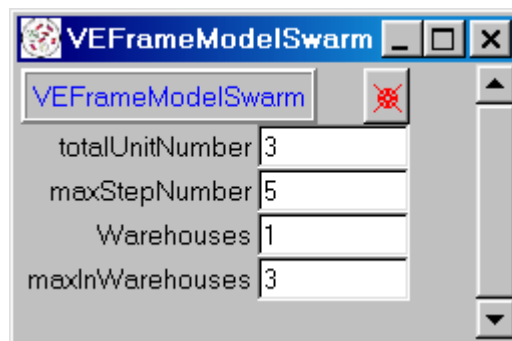
7.11 Creazione tecnica dei magazzini

Seguendo la struttura tipica di Swarm, la creazione dei magazzini avviene all'interno del "Model". Qui viene invocato il costruttore della classe, e i singoli oggetti creati sono inseriti in una lista (warehouseList), per potervi accedere con facilità e costruire il grafico corrispondente. L'indirizzo dei magazzini viene inviato all'unità corrispondente. Viene dunque associato ad ogni magazzino un numero, corrispondente a quello delle unità produttive, e viene inizializzato il contatore, che è posto uguale a "0".

Per poter introdurre i magazzini nel modello, ho introdotto due ulteriori parametri della simulazione. Il primo, chiamato semplicemente “useWarehouses”, può essere posto uguale ad “1” oppure a “0” e determina la presenza o meno dei magazzini.

Il secondo è denominato “maxInWarehouses” e permette di scegliere la capienza massima dei magazzini. Può essere un qualunque numero intero superiore a 0, e nel caso sia posto uguale a 0 la capienza sarà infinita, cioè non c’è limite al riempimento del magazzino.

Per semplicità e uniformità, i magazzini hanno tutti la stessa capienza, determinata all’inizio della simulazione.



7.12 Le regole per il funzionamento

Per funzionare, i magazzini hanno bisogno di regole, che indicano in che circostanze si devono riempire o svuotare.

Per rendere pulito e comprensibile in codice, queste regole sono contenute all’interno di un oggetto chiamato “RuleMaster”, in accordo con lo schema ERA già descritto.

Il “RuleMaster” è un oggetto, anch’esso costruito dal Model, al quale vengono passati l’indirizzo della lista contenente i magazzini e la variabile che indica la massima capienza.

Quando un’unità produttiva è libera, cioè non ha nulla da produrre per il mercato, interroga il RuleMaster. Questo sa rispondere al messaggio “checkNumber”, lanciato dalla singola “unit”, che passa come parametri il proprio numero ed il numero totale delle unità presenti nella simulazione.

A questo punto, il RuleMaster annuncia che l'unità da cui è stato chiamato è libera, e chiede al magazzino corrispondente, cioè con lo stesso numero identificativo, quanti pezzi contiene.

Nel caso i pezzi già contenuti siano inferiori alla capienza, il RuleMaster ordina la produzione per il magazzino; se invece la capacità massima è già stata raggiunta, si comunicherà che il magazzino è già pieno e non può ospitare altri componenti.

La scelta di una regola assolutamente semplice per la gestione dei magazzini è stata dettata dalla fase in cui si trova il progetto; per ora è infatti più importante verificare i risultati ottenuti con l'aggiunta di parti al modello, piuttosto che cercare di ottimizzare le decisioni di gestione.

In una versione futura si potrà per esempio aggiungere la possibilità di guardare la ricetta dell'ordine e ordinare all'unità di produrre per il magazzino solo se il componente sarà richiesto in una fase successiva. In questo modo si potrebbe realizzare una sorta di "Just in Time".

Il RuleMaster è anche in grado di rispondere al messaggio "checkWarehouse", lanciato dall'unità quando questa riceve l'ordine di produrre un componente.

Anche in questo caso, i parametri passati sono il numero dell'unità che inoltra la richiesta ed il numero totale delle unità presenti nella simulazione.

Il RuleMaster controllerà nel magazzino corrispondente, e nel caso vi siano contenuti dei componenti, sottrae un'unità al contatore.

L'unità a questo punto non dovrà produrre, sveltendo il processo e saltando di fatto uno step.

In questo modo dovrebbe essere assolta la funzione dei magazzini che consiste nel ridurre le code di produzione, colmando i tempi morti, in cui cioè le unità sarebbero potenzialmente libere e inutilizzate.

7.13 La rappresentazione del carico dei magazzini

Nel caso in cui i magazzini siano presenti nella simulazione, è utile avere una rappresentazione grafica, aggiornata dinamicamente, del carico istantaneo minimo, massimo e medio.

Per questo motivo ho inserito un grafico che presenta il trend del carico dei magazzini. Sull'asse delle ascisse è raffigurato il trascorrere del tempo, mentre sull'asse delle ordinate si misura il carico dei magazzini.



La linea gialla è il carico massimo dei magazzini, mentre quella blu rappresenta il carico medio, e quella arancione il minimo. Questo permette di tenere sotto controllo la movimentazione dei magazzini, e notare come il flusso di prodotti possa effettivamente sveltire la produzione finale.

Per realizzare praticamente il grafico, ho modificato l'Observer e ho inserito un oggetto di tipo "EZGraphImpl", chiamato "warehouseGraph". Questo tipo di oggetto è in grado di leggere il valore del contatore dei singoli magazzini, e di restituire il valore minimo, medio e massimo riscontrabile interrogando tutti gli oggetti.

7.14 Realizzazione tecnica

Utilizzando la programmazione ad oggetti, tipica di JAVA e molto utile per costruire modelli basati su agenti, è possibile descrivere con semplicità il funzionamento della simulazione, in quanto ogni singolo oggetto dispone di un'interfaccia che ne definisce le caratteristiche.

E' possibile dividere l'interfaccia in due categorie: la prima definisce le caratteristiche in termini di memoria, attraverso le variabili di stato utilizzate.

La seconda descrive invece il comportamento degli agenti, ed è costituita dai metodi.

L'agente "Warehouse" è piuttosto semplice, in quanto sono poche le variabili ed i metodi che lo contraddistinguono. Questo non impedisce tuttavia ai magazzini di svolgere la propria funzione nel modello, identificabile con la riduzione delle code che si creano presso le singole unità produttive.

Le variabili di stato:

1. **public int warehouseNumber;** rappresenta il numero del magazzino, in modo da permetterne l'identificazione da parte dell'unità produttiva corrispondente
2. **public int counter;** il contatore che misura il numero dei componenti contenuti nel magazzino
3. **public ListImpl warehouseList;** la lista contenente gli indirizzi in memoria di tutti i magazzini presenti nella simulazione

I metodi:

1. **public int getCounterValue()** restituisce il valore del contatore, dunque la quantità dei componenti contenuti nel magazzino
2. **public int getWarehouseNumber()** restituisce l'etichetta numerica del magazzino

3. **public void addCounterValue()** incrementa il contatore, cioè aggiunge un componente al magazzino
4. **public void subCounterValue()** operazione inversa di quella descritta prima: diminuisce il contatore, cioè sottrae un componente dal magazzino

In questa prima fase, in cui il debug è molto importante, le variabili sono di tipo “public”, cioè accessibili dall'esterno, anche se la loro modificazione è gestita da meccanismi interni all'agente stesso.

Il RuleMaster è per ora utilizzato solamente per le regole facenti capo ai magazzini, ma può facilmente essere modificato, al fine di introdurvi precetti anche per altre parti della simulazione, per esempio per contabilizzare i costi generati.

Anche qui è possibile identificare l'interfaccia, divisa in variabili statici e metodi.

Le variabili di stato:

1. **public int unitNumber;** il numero che identifica l'unità produttiva, per ora coincidente anche con il processo produttivo di cui l'unità è capace
2. **public int warehouseNumber;** rappresenta il numero del magazzino, in modo da permetterne l'identificazione da parte dell'unità produttiva corrispondente
3. **public int counter;** il contatore che misura il numero dei componenti contenuti nei magazzini
4. **public int i;** variabile contatore, utile per i cicli “for”
5. **public int totalUnitNumber;** numero totale delle unità presenti nella simulazione, coincidente anche con il numero dei magazzini
6. **public int maxInWarehouses;** variabile identificativa della massima capienza gestibile dai magazzini, definibile dall'utente ad inizio simulazione

I metodi:

1. **public void checkNumber(int un, int totNum)** metodo invocato quando un'unità produttiva è libera, che controlla il magazzino, ed incrementa il contatore di un'unità nel caso la massima capienza non sia ancora raggiunta

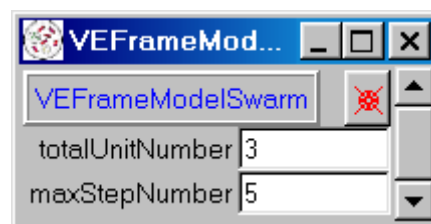
2. **public int checkWarehouse(int un, int totNum)** metodo invocato dalla Unit quando è richiesta la produzione di un componente. Nel caso sia presente in magazzino, viene impiegato questo, sollevando l'unità dal lavoro.

CONCLUSIONI ED ESPANSIONE DEL MODELLO

8.1 Il modello base

Nonostante il modello JVE sia molto semplice, fornisce già fin d'ora risultati interessanti. La possibilità di inserire una quantità di unità produttive ed una lunghezza dei prodotti a scelta dell'utente permette di studiare situazioni diverse.

La situazione di default prevede la presenza di tre unità e di un numero massimo di componenti, per ogni prodotto, pari a cinque.

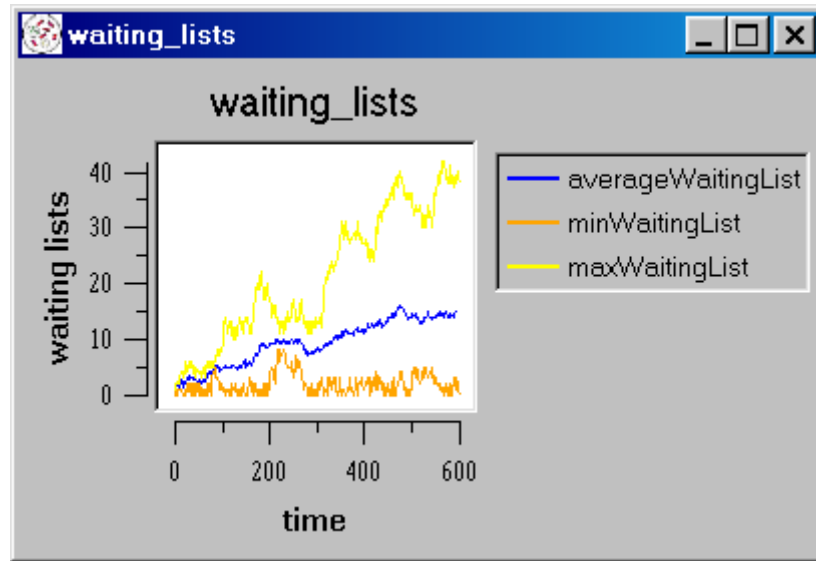


Questo significa che il numero medio di componenti, casuale, è pari a tre, cioè uguale al numero di unità produttive presenti nella simulazione.

Tuttavia, il singolo componente può, all'interno dello stesso prodotto, essere ripetuto più volte anche se le componenti sono meno di tre. Si potrebbe, per esempio, verificare prodotto del tipo "1 1" che dovrebbe, dunque, passare due volte presso l'unità produttiva che costruisce il componente "1".

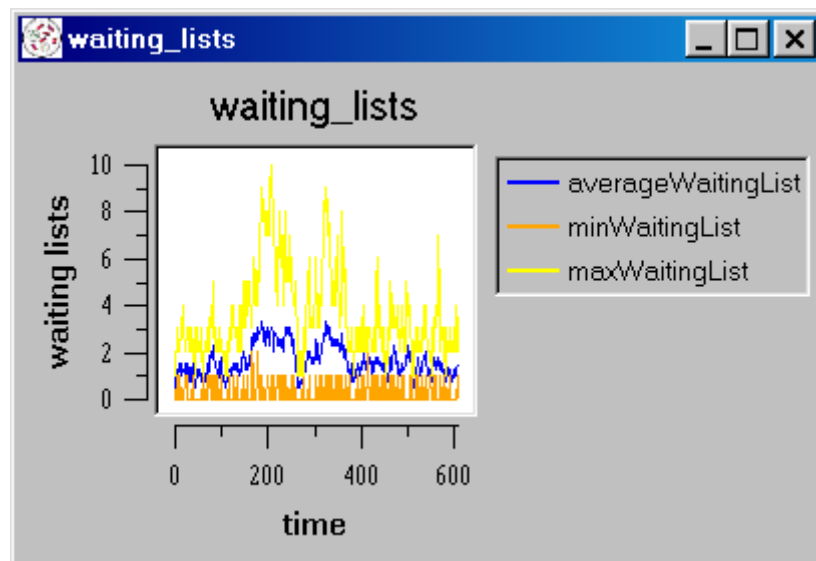
Questo fa sì che anche in questa situazione così semplice si verifichi una situazione che a priori potrebbe essere difficilmente prevista: con un numero medio di componenti per ogni prodotto pari al numero delle unità presenti nella simulazione si assiste alla formazione di code crescenti presso tutte le unità.

In pratica, si osserva quanto riportato nel seguente grafico:



L'unità produttiva più carica, dopo seicento passi della simulazione, rimane definitivamente oberata di lavoro, con una coda di attesa superiore ai quaranta ordini da evadere. La media di attesa è invece pari a quindici ordini da evadere, comunque molti, considerando che nella realtà il ritardo di consegna si traduce in un costo per l'azienda, sotto forma di insoddisfazione del cliente.

La situazione muta profondamente se, mantenendo il numero massimo di componenti immutato, inseriamo una quarta unità produttiva nell'azienda virtuale. Il risultato, a parità di passi della simulazione, è rappresentato nel grafico seguente:



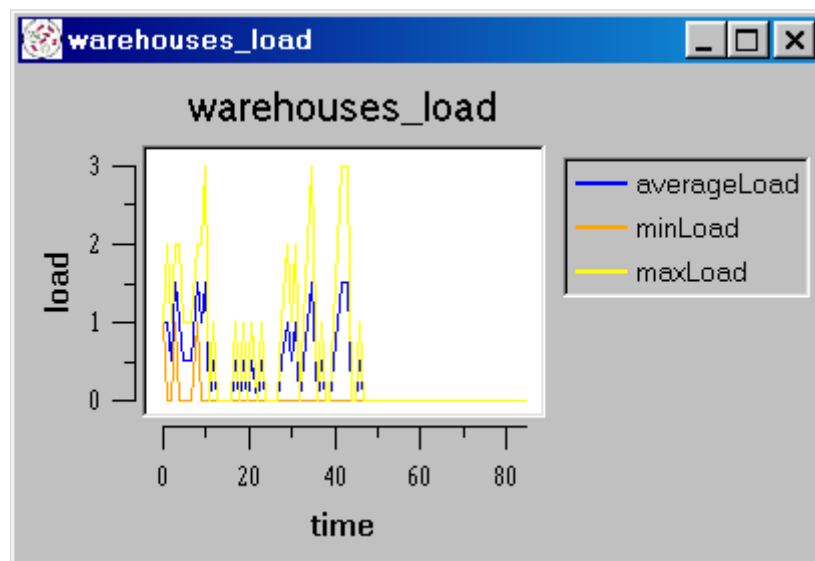
Risulta chiaro che in questo caso non si formano sistematicamente delle code: si possono verificare dei rallentamenti di produzione, che però vengono immediatamente riassorbiti.

8.2 Introduzione dei magazzini

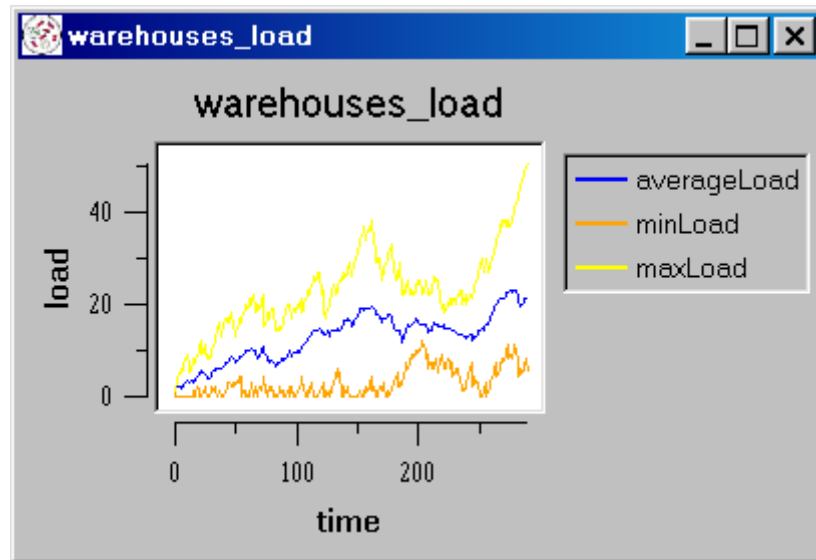
Con l'introduzione dei magazzini, descritta tecnicamente nel capitolo precedente, ho cercato di verificare se le code che si formano nel modello base vengono attenuate. Anche i magazzini sono oggetti molto semplici, in quanto funzionano come dei contatori, che vengono incrementati quando si introduce un componente, e decrementati quando lo stesso viene utilizzato per la produzione.

La regola di riempimento è altrettanto semplice: se l'unità non produce per il mercato, lo fa per il magazzino. In questo modo le unità lavorano sempre, mentre nel modello base potevano trascorrere cicli di inattività se non vi erano componenti corrispondenti da produrre. Introducendo i magazzini si verificano due fatti interessanti: principalmente, come previsto, le code si attenuano, anche se non spariscono, nel caso in cui il numero di unità sia uguale al numero medio di componenti del prodotto finito.

Il risultato più interessante riguarda invece la logica con cui i magazzini vengono utilizzati effettivamente: nel caso in cui le unità produttive siano troppo poche rispetto al numero di componenti medio (per esempio due unità produttive a fronte di un numero massimo di componenti pari a quattro, che si traduce in una media di 2.5), dopo pochi passi le unità, oberate di lavoro per il mercato, smettono di produrre per i magazzini. Il tutto risulta chiaro del seguente grafico:



La situazione opposta si verifica invece quando le unità nella simulazione eccedono in numero la quantità media di componenti. In questo caso, infatti, se non definiamo una massima capienza dei magazzini, notiamo che essi si riempiono indefinitamente, tendendo ad infinito. Anche questo esempio è spiegato dal grafico che segue:



Dopo soli trecento cicli, il carico medio dei magazzini è pari a venti componenti. Questa situazione è assolutamente realistica, in base alle regole di funzionamento che abbiamo attribuito ai magazzini.

8.3 Espansione dei magazzini

Da quanto esposto sinora, si evince come il modello di impresa virtuale elaborato sia molto versatile e si presti ad essere espanso in più direzioni, al fine di ricercare un contatto sempre più stretto con quanto si può osservare nella realtà.

Un'interessante prima evoluzione consiste nell'aggiunta di magazzini all'interno dell'azienda simulata.

Nonostante le imprese perseguano sempre più il "just in time", cercando di inseguire la domanda e di ridurre al minimo le scorte, i magazzini sono infatti una realtà innegabile, soprattutto per abbreviare i tempi di consegna e non creare colli di bottiglia alla produzione.

Nel modello, i magazzini possono essere implementati in diversi modi, ognuno dei quali si rifletterà sulla flessibilità aziendale e sui costi complessivi generati.

La prima possibilità consiste nel dotare ogni unità produttiva di un proprio magazzino, com'è stato fatto.

In questo modo, le unità produrranno componenti per il magazzino, anche quando non è espressamente richiesto dal mercato, finché non si raggiunge un numero di scorte determinato (in una fase iniziale, per non togliere equilibrio al modello, si potrebbe ipotizzare che nel magazzino debba essere contenuto un solo componente di riserva).

Avere scorte in magazzino, significa ovviamente una maggior rapidità nella produzione, anche quando si formano code.

I magazzini possono essere rappresentati da liste, che si riempiranno e vuoteranno con i puntatori agli oggetti creati (i componenti), oppure da semplici contatori: in quest'ultimo caso "o" rappresenterà il magazzino vuoto, mentre ogni oggetto inserito incrementerà il contatore di una unità, finché tale numero non arriva a coincidere con il massimo prefissato.

Durante il "DO", l'ordine di produrre il componente, rivolto ad ogni unità, è contenuto nella "listaCoseDaFare" dell'unità stessa: se questa lista è vuota e se nel magazzino le scorte sono inferiori al massimo prefissato, avverrà comunque la produzione, ma l'oggetto prodotto sarà introdotto nella "listaMagazzino" (oppure il contatore sarà incrementato di una unità se si sceglie questo procedimento).

Se invece la "listaCoseDaFare" contiene ordini, l'unità controllerà all'interno della "listaMagazzino": se è già presente il componente finito, l'ordine viene immediatamente completato, e l'oggetto viene inviato alla "listaCoseFatte" (e ovviamente sottratto alla "listaMagazzino"). Se invece il magazzino non contiene oggetti già pronti, l'ordine sarà classificato come "in lavorazione".

Nel caso in cui il magazzino fosse rappresentato da un contatore, se quest'ultimo segnerà un numero positivo, si creerà un oggetto che verrà immediatamente inserito nella "listaCoseFatte", e si diminuirà il contatore. Se invece il contatore segna "o", si passerà alla lavorazione.

Questa soluzione permette sicuramente di simulare l'effettivo miglioramento dell'efficienza aziendale, in termini di rapidità costruttiva ed eliminazione dei colli di bottiglia, rappresentato dall'introduzione di magazzini.

Inoltre, questa organizzazione dei magazzini, permette di far emergere i vantaggi dell'introduzione di E-business nella gestione aziendale: le unità devono decidere se lavorare al massimo della capacità o no (indipendentemente da ciò che hanno nella lista di ordini da evadere) e quindi per il proprio magazzino, in funzione di informazioni sugli ordini in arrivo e delle previsioni degli ordini stessi.

Se i magazzini sono costituiti nel modo descritto, non è cruciale la conoscenza delle modalità con le quali si attinge ad essi: l'organizzazione delle entrate ed uscite può infatti essere indifferentemente di tipo LIFO o FIFO, ma il risultato nel modello sarà sempre identico (a livello aggregato, almeno), in quanto ogni unità produttiva dispone di un proprio magazzino, in cui è contenuto un solo tipo di componente.

Idea decisamente diversa, sicuramente meno efficiente, è invece quella di avere un solo magazzino, che in tal caso ovviamente deve essere costituito da una lista, per riflettere le diversità degli oggetti in essa contenuti. Il magazzino potrebbe avere una capienza limite, e da esso potrebbero essere attinti solo alcuni dei componenti pronti, a causa dell'organizzazione interna (sia essa LIFO o FIFO). Per esempio, decidendo che solo i primi tre oggetti entrati possono essere utilizzati, se il componente richiesto fa parte di essi, viene automaticamente inserito nel prodotto finito. Se invece esso è presente nel magazzino, ma è impossibilitato ad uscire, non potrà comunque essere utilizzato. A seguito dell'uscita di un componente dal magazzino tutti le altre scorte presenti potrebbero fare un passo avanti, e lasciare libera l'ultima posizione, che verrà occupata dalla produzione della prima unità livra (ipotesi FIFO).

Quest'organizzazione del magazzino è tuttavia meno utile per la simulazione: l'informazione dovrebbe infatti essere decentrata, per poter cogliere la complessità della realtà. E' dunque preferibile che ogni unità conosca il numero dei prodotti contenuti nel proprio magazzino, il che ci riporta all'idea di un magazzino per ogni unità.

Inoltre un magazzino unico potrebbe generare meno costi fissi, ma più costi di gestione ed eventuali difficoltà di comunicazione.

8.4 Introduzione dei tempi di produzione

Nelle aziende reali, non tutti i componenti richiedono un tempo di lavorazione uguale. Anzi, il tempo di lavorazione è spesso uno degli aspetti più cruciali per la programmazione della produzione, tanto da far spesso decidere in favore di eventuali

“outsourcing”, nel caso in cui il componente da produrre costituisca un’incombenza troppo pesante per l’azienda, creando colli di bottiglia rilevanti all’interno della produzione.

Mentre in una fase iniziale del modello non si terrà conto di questo (almeno non formalmente), uno sviluppo futuro potrebbe comprendere questo aspetto.

Propongo dunque una modalità per tener conto dei tempi all’interno dell’impresa virtuale.

Si potrebbe inserire un semplice vettore che riporta i tempi di produzione per ogni singola componente (questo è fattibile, in quanto il numero di componenti è finito e predeterminato, nella fattispecie numeri interi ad una cifra).

Praticamente, avremo un oggetto di questo tipo:

1	2	3	4	5	6	7	8	9
a	b	c	d	e	f	g	h	i

dove i numeri (indici del vettore stesso) indicano la componente, e la lettera indica il tempo di produzione necessario per essa.

Il tempo di produzione di ogni componente potrebbe non essere necessariamente statico, ma magari variare all’interno di un “range” predefinito, utilizzando numeri casuali.

Questo vettore potrebbe essere noto anche al “Front End”, che dunque avrebbe la possibilità di "ordinare" a certe unità di produrre per il magazzino, nel caso i tempi di produzione della componente ad essa imputata fossero particolarmente lunghi.

Ogni unità potrebbe essere poi dotata di un massimo di energia totale (dotazione giornaliera), per la produzione.

Quando l’unità guarda nella "listaCoseDaFare", fa anche un computo del tempo totale richiesto, e lo sottrae dalla dotazione giornaliera. Questo procedimento continua finché non ci sono più ordini o finché l’energia è terminata.

Altro approccio, sicuramente più semplice da integrare nel modello già in una fase iniziale, è quello di deformare artificialmente la “ricetta”, per tener conto dei diversi tempi di produzione.

Se dobbiamo produrre 12364 e solo il processo 3 dura due unità di tempo, la ricetta può essere 123364: questo è sicuramente un approccio più sintetico, meno generale del

precedente, ma sicuramente efficace nel rappresentare le differenze nel tempo richiesto per la produzione dei singoli componenti.

8.5 Introduzione della contabilità dei costi

Poiché le aziende devono creare profitti, o per lo meno cercare di evitare perdite, una buon denominatore comune per verificare l'efficienza complessiva dell'impresa consiste nell'esaminare i costi e i ricavi.

Nel modello di impresa virtuale presentato, è necessario ricercare un modo piuttosto semplice per contabilizzare i costi, in modo tale da ottenere un indicatore, e non un valore congruente con un ipotetico mercato.

I costi sono fondamentalmente di due tipo: fissi, cioè dovuti alla semplice esistenza e manutenzione delle parti aziendali, oppure variabili, cioè dipendenti dal numero di componenti prodotti dalle singole unità e da altri aspetti della produzione. Esistono poi costi generati dal mantenimento di magazzini, e dalla permanenza di un prodotto negli stessi, che si può supporre direttamente proporzionale al tempo.

Ovviamente, i costi, siano essi fissi o variabili, sono diversi per ogni unità: ci saranno dunque componenti più o meno costosi da produrre e altresì esisteranno unità che per il proprio mantenimento necessiteranno cure maggiori e che dunque genereranno costi più ingenti. Si possono introdurre guasti casuali alle singole unità produttive, allorché la riparazione sarà imputata ad essa, e genererà un costo che l'azienda potrà ammortizzare (in media) tenendo conti di questi imprevisti nel prezzo del prodotto finito.

I costi, tuttavia, non sono solamente fisici: per l'azienda, infatti, costituisce un costo anche l'insoddisfazione del cliente. E' necessario dunque badare ai costi concreti, senza tuttavia perdere di vista i tempi di consegna: se questi infatti si innalzano troppo, rendono il cliente insoddisfatto, e dunque portano un danno derivato all'azienda (il non riacquisto). Inoltre, si potrebbe immaginare, proporzionalmente al ritardo, una diminuzione del prezzo del prodotto finito, una sorta di "penale" per non aver consegnato il prodotto nel momento concordato. Questo si riflette soprattutto sulle strategie di gestione magazzini, secondo lo schema seguente:



Per mantenere il modello snello e facilmente osservabile, ogni unità sarà in grado di gestire l'account dei costi ad essa genericamente imputabili (fissi, variabili, di magazzino). In questo modo si decentra l'informazione, che può essere osservata nella sua totalità, oppure per ogni unità produttiva.

Passando a considerare i ricavi, è importante decidere come determinare il prezzo dei prodotti finiti. Una strategia comunemente utilizzata, e facilmente modellizzabile, è la regola del “mark-up”: il prezzo del prodotto finito è dato dai costi marginali di lavorazione, approssimabili nei costi medi, moltiplicati per una costante con valore superiore all'unità (realisticamente compresa tra 1 e 2). Questa premessa è realistica soprattutto se l'azienda che consideriamo ha potere di mercato; nel caso di una concorrenza perfetta il mark-up è decisamente prossimo all'unità.

Come costo medio di produzione si deve considerare quello dovuto al costo di ogni singolo componente, dato approssimativamente dal costo variabile direttamente imputabile (costi da lavoro, variazioni di produttività, eventuali aspettative inflazionistiche), più una percentuale di costo fisso, derivato da una produzione media considerata nell'arco di tempo considerato.

Poiché nel modello non è semplice assegnare valori realistici a queste variabili, potremmo semplicemente considerare che, quando un componente richiede più tempo per essere prodotto, esso costa di più, oppure assegnare arbitrariamente un costo ad ognuno di essi.

La somma dei costi medi dei singoli componenti darà dunque il costo totale, che, moltiplicato per la costante di mark-up, fornirà il prezzo finale di vendita.

Da quanto esposto, si capisce subito che eventuali ritardi creeranno costi non previsti, che eroderanno i guadagni, e così avverrà anche per i costi generati

dall'insoddisfazione dei clienti o quelli dovuti ad imprevisti di produzione o ad un sovraccarico dei magazzini.

Normalmente, le aziende, sul lungo periodo, modificano i loro prezzi per venire incontro alle diverse aspettative inflazionistiche, e ad eventuali shock sui costi delle materie prime. Poiché noi consideriamo il breve periodo, tali grandezze non hanno rilievo e dunque non è errato considerare costanti i prezzi. Inoltre, una variazione troppo frequente dei prezzi, crea dei costi ("menu cost") e dunque non è desiderabile per un'impresa.

E' infine possibile generare artificialmente dei costi tagliando la comunicazione tra le unità produttive, oppure rendendola poco efficiente, magari per mezzo di errori casuali nella trasmissione dei dati. Questi costi potrebbero essere abbattuti con l'integrazione di un'efficiente sistema di E-business che terrebbe sotto controllo tutti gli scambi di informazione all'interno dell'azienda.

Ogni unità invierà i costi generati al front end, che terrà conto anche dei ricavi, e calcolerà un indice sintetico delle prestazioni aziendali basato sulla differenza tra ricavi e costi.

Bibliografia in ordine alfabetico

ARNTZEN, B. C., G. G. BROWN, T. P. HARRISON, and L. TRAFTON, 1995, Global Supply Chain Management at Digital Equipment Corporation. *Interfaces*, Jan.-Feb.

ARORA A., BOKHARI F., MOREL B. 1997, 1998, Returns to specialization, transaction costs, and the dynamics of industrial evolution, Working paper Carnegie Mellon University

ARTHUR W.B., 1990, A Learning Algorithm that Mimics Human Learning, Santa Fe Institute Working Paper 90-026

ARTHUR W.B., DURLAUF S.N., LANE D.A., 1997, *The Economy as an Evolving Complex System II*, Reading: Addison-Wesley

ASKENAZI M., BURKHART R., LANGTON C., MINAR N., 1996, The Swarm Simulation System: A Toolkit For Building Multi-Agent Simulations, Santa Fe Working Paper (<http://nelson.www.media.mit.edu/people/nelson/research/swarm>)

AXELROD R., 1997, *The Complexity of Cooperation*, Princeton, Princeton University Press

AXTELL R.L., EPSTEIN J.M., 1994, Agent Based Modeling: Understanding Our Creations, *The Bulletin of the Santa Fe Institute*, Winter

BELLANDI G., 1991, *La gestione dei rapporti con il mercato*, Il Borghetto, Milano

BILLINGTON C., LEE H.L., 1992, Managing Supply Chain Inventory: Pitfalls and Opportunities, *Sloan Management Review*, spring

BILLINGTON C., LEE H.L., 1993, Material Management in Decentralized Supply Chains, *Operations Research*

BRANDIMARTE P., A.VILLA, 1995, *Gestione della Produzione Industriale*, UTET, Italy

BRYNJOLFSSON E., L. M. HITT, 2000, *Beyond Computation: Information Technology, Organizational Transformation and Business Performance*

CHANDLER A.D. 1990, Scale and Scope: The Dynamics of Industrial Capitalism, Harvard University Press

CONTE R., 1997, Il metodo simulativo, Istituto di Psicologia, Cnr Roma, Paper

COOPER, M. C., ELLRAM, L. M., 1993. Characteristics of Supply Chain Management and the Implications for Purchasing and Logistics Strategy. The International Journal of Logistics Management

DAY R.H., 1994, Complex Economic Dynamics, Volume I: An Introduction to Dynamical Systems and Market Mechanisms, MIT Press

DELAINI C., LEGRENZI, MARENGO, ORSENIGO, 2000, La simulazione della divisione del lavoro, Mente e società.

ELSAYED E.A., T.O. BOUCHER, 1995, Analysis and Control of Production Systems Management, CRC Press, Boca Raton (Fl)

EOYANG G., 1999, Coping with Chaos: Seven Simple Tools by Glenda Holladay Eoyang. Reviewed by Wei-Bin Zhang Institute for Future Studies, Stockholm, Sweden

FARRELL J., MONROE H.K., SALONER G. 1998, The Vertical Organization of Industry: Systems Competition versus Component Competition, Journal of Economics and Management Strategy

FORRESTER J.W., 1961, Industrial Dynamics, MIT Press

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

(http://silmaril.smeal.psu.edu/misc/supply_chain_intro.html)

GILBERT N., TERNA P., 1999, How to build and use agent-based models in social science, Mind & Society, 1

GLEICK J., 1987, Chaos: Making A New Science, Viking ed.

GRAVES S.C., A.H.G. RINNOOY KAN, P.H. ZYPKIN, 1993, Logistics of Production and Inventory, North Holland, Amsterdam

HAX A.C., D. CANDEA, 1984, Production and Inventory Management, Prentice-Hall, Englewood Cliffs (NJ)

HAYEK, F.A. (1948e). The Use of Knowledge in Society (reprint). In F.A. Hayek (Ed.), Individualism and Economic Order, Chicago: University of Chicago Press

HOLLAND, H. John, 1995, Hidden Order: How adaptation builds complexity. Addison-Wesley.

HORGAN J., 1995, From Complexity to Perplexity, Scientific American, June 1995

KAUFFMAN, 1995, At Home in the Universe: The Search for the Laws of Self-Organization and Complexity, Oxford University Press

LIN F.-R., SHAW M.J., STRADER T.J., 1999, The Impact of Information Sharing on Order Fulfillment in Divergent Differentiation Supply Chain, Journal of Global Information Management. 7 . 1. January - March

LIN F.-R., TAN G.W., SHAW M.J., 1996, Multi-agent enterprise modeling. Working Paper 96-0314, University of Illinois at Urbana-Campaign

LIN F.-R., TAN G.W., SHAW M.J., 1996, Multi-agent enterprise modeling. Working Paper 96-0314, University of Illinois at Urbana-Campaign

MCKELVEY, 1997, Quasi-Natural Organisation Science, Organization Science No.8

MERRY U., 1999, Practically Applying the New Science to Organizations, KE Workgroup Paper, (<http://pw2.netcom.com/~nmerry/pract1.htm>)

MINSKY M., 1987, The Society of Mind, Picador, London

NIIP Consortium, 1998, NIIP Reference Architecture, (<http://www.niip.org>)

- PEPE G., 2000, La conoscenza per il posizionamento strategico. Un nuovo modello di analisi organizzativa, (<http://www.eng.it/Ingenium/ingenium.html>)
- POLANYI K., 1966, The Tacit Dimension, London: Routledge and Keagan
- ROSSER B.J., 1999, On the complexity of complex economic dynamics, Journal of economic perspectives, vol.13, n.3
- SADEH N.M., SMITH S.F. e SWAMINATHAN J.M., 1998, Modeling SupplyChain Dynamics: a Multi-Agent Approach, Decision Sciences Vol.29
- SALTARIN A., 2000, BizTalk: fare EDI con XML e Schema, Computer Programmino, Maggio
- SCHEER A.W., 1993, Architecture of integrated information systems (aris). In Yoshikawa H., Goossenaerts J., Information Infrastructure Systems for Manufacturing, Elsevier Science B.V.
- TAKADA, 1994, In PEPE G., 2000.
- TERNA P., 1994, Reti neurali artificiali e modelli con agenti adattivi, XXXV Riunione scientifica annuale della società italiana degli economisti
- TERNA P., 1997, A Laboratory for Agent Based Computational Economics: The Self-development of Consistency in Agents' Behaviour, University of Turin
- TERNA P., 1998, Simulation Tools for Social Scientists: Building Agent Based Models with SWARM, JASSS vol.1, no.2
- TERNA P., 1999, Economic Experiments with Swarm: a Neural Network Approach to the Self-Development of Consistency in Agents' Behaviour, in LUNA F., STEFANSSON B., Economic Simulations in Swarm: Agent Based Modelling and Object Oriented Programming. Kluwer Academic.
- TERNA P., 1999, How to use bp-ct, (terna@econ.unito.it).
- VRIEND NJ, 1999, Was Hayek an Ace?, University of London [Paper]

Appendice A

ELEMENTI DI PROGRAMMAZIONE IN JAVA

A.1 Una premessa

Quest'appendice costituisce un riferimento per la programmazione in linguaggio Java. Gli argomenti trattati sono utili al fine di acquisire una conoscenza di questo linguaggio, sufficiente per affrontare la costruzione di programmi, anche complessi, atti a svolgere le funzioni più usate.

Nota: nei prossimi paragrafi si troverà spesso il sostantivo “istanza” o il verbo “istanziare”, sicuramente poco piacevoli da un punto di vista linguistico; preciso che si tratta di termini, tipici dei linguaggi Object Oriented, che derivano dall'inglese *instance*, letteralmente traducibile con esempio, esemplare. Una “istanza” di una classe è dunque un oggetto derivato da essa.

A.2 Per iniziare

I programmi Java sono solitamente divisi in diversi files. Il file principale deve contenere il cosiddetto metodo `main`, secondo la seguente sintassi:

```
class nomeprogramma {  
    public static void main (String args[])  
    {  
  
        ...  
        ...  
  
    }  
}
```

Per dichiarare il metodo `main()`, l'unico presente in questo sorgente, occorre includerlo in una classe; il nome della classe e quello del file in cui è dichiarata devono essere uguali per compilare senza errori. Java è *case sensitive*, il ch  significa che due parole sono diverse, se hanno caratteri maiuscoli e minuscoli in posizione differente.

Il comando `String args[]`, compreso tra parentesi, identifica i parametri che si passeranno alla classe in fase di esecuzione del programma.

A.3 Comando Import

Per utilizzare comandi e funzioni, è necessario importare delle librerie, in cui sono contenuti. Gli import devono essere effettuati prima di ogni altro comando, all'interno del listato. Import è analogo al comando `#include` del linguaggio C. Seguono alcune librerie, tra le più usate:

<code>java.io.*;</code>	per le operazioni I/O
<code>java.lang.*;</code>	tipi di base sempre importati in ogni programma.
<code>java.net.*;</code>	librerie di classi che provvedono a gestire socket, interfacce telnet, URL.
<code>java.util.*;</code>	librerie di utilità varie, come data, orologio, tecniche di codifica e decodifica, numeri casuali e altro
<code>java.awt.*;</code>	Abstract Windowing Toolkit. Libreria di classi adatta a sviluppare applicazioni per ambienti a finestra. Contiene interfacce grafiche standard come gestione degli eventi, colori, barre di scorrimento e pulsanti.

A.4 Variabili in Java

La dichiarazione di variabili in Java è simile a quella del C. Non è possibile utilizzare un nome che inizi per un numero o con simboli come `&/,!`; si deve inoltre ricordare che le lettere maiuscole sono diverse dalle minuscole (PLUTO è diverso da Pluto).

Per dichiarare una variabile si utilizza la seguente sintassi:

```
tipo_variabile nome_variabile = contenuto;
es. String name = "Marco";
```

oppure

```
int var1, var2;
var1= valore1;
var2= valore2;
```

A.5 Numeri casuali

Come in ogni linguaggio di programmazione evoluto, anche in Java è possibile fare uso di numeri casuali. Esiste un tipo di variabile `Random`, per utilizzare la quale deve essere importata la libreria `Java.util.*`

```
Random myRand = new Random();    come tipo variabile

a = myRand.nextInt();             dove a è una variabile intera
                                   precedentemente definita
```

A.6 Tipi di variabile

Tipo	Lungh.	Range (valori minimi e massimi)
byte	8 bits	-128 ÷ 127
char	16 bits	Caratteri in formato UNICODE (non ASCII)
short	16 bits	-32,768 ÷ 32,767
int	32 bits	-2.147.483.648 ÷ 2.147.483.647
long	64 bits	± 9 miliardi di miliardi
float	32 bits	1.4e-45 ÷ 3.4e+38
double	64 bits	4.9e-324d ÷ 1.7e+308d

A.7 Stringhe di testo

In Java le stringhe sono degli oggetti, a differenza di quanto avviene nel linguaggio C. I caratteri in Java sono in formato UNICODE a 16 bit, non ASCII a 8bit. Per dichiarare una stringa si utilizza il seguente costrutto:

```
String stringa = "prova";
```

Le stringhe non possono essere cambiate dopo la loro creazione. La classe *String* ha molti metodi; ne cito alcuni:

<code>charAt(int);</code>	Restituisce il carattere all'indice specificato. 0 = primo carattere, <code>length()-1</code> = ultimo.
<code>equals(String);</code>	Confronta questa stringa con un'altra. È case sensitive, ossia Pippo e PiPpO sono diversi.
<code>length();</code>	Restituisce la lunghezza della stringa.
<code>toLowerCase();</code>	Converte la stringa in soli caratteri minuscoli.
<code>toUpperCase();</code>	Converte la stringa in soli caratteri maiuscoli

Inoltre esistono classi riguardanti le stringhe. Una interessante è `Java.util.StringTokenizer`, che ha dei metodi per separare i *token* (alla lettera segni, simboli) in un testo: le singole parole di una frase, ad esempio. Senza parametri addizionali, i caratteri riconosciuti come separatori sono gli spazi, i *tab*, e gli "a capo" (*return* e *newline*).

Avendo la stringa "Questa è una prova 1 2 3", si ottengono i *token* "Questa", "è", "una", "prova", "1", "2", "3".

Per poter usare `StringTokenizer`, occorre fare un `import`, e poi "istanziarlo":

```
import Java.util.StringTokenizer;

String prova1 = "Questa è una prova 1 2 3";
StringTokenizer miostoke = new StringTokenizer(prova1);
```

Il costruttore vuole in entrata la stringa su cui operare.

Si può anche aggiungere un'altra stringa ai parametri del costruttore, con i caratteri da considerare delimitatori di *token*, se si vuole.

A questo punto, ecco i metodi più utili:

<code>nextToken();</code>	Restituisce il prossimo <i>token</i> della stringa. Se viene chiamato quando non ci sono più <i>token</i> , genera l'eccezione <code>NoSuchElementException</code> .
<code>countTokens();</code>	Restituisce il numero di <i>token</i> ancora disponibili prima che <code>nextToken</code> generi un'eccezione per fine <i>tokens</i> .

`hasMoreTokens()` ; Restituisce `true` (vero) se ci sono ancora *token*

A.8 Stampa a schermo

Si tratta di uno dei comandi base per ogni linguaggio di programmazione. L'operatore “.”, in Java, serve per richiamare metodi di una determinata classe. Questo sarà più chiaro in seguito.

Per stampare un testo sullo schermo occorre la seguente sintassi:

```
system.out.println("testo da scrivere");
```

per stampare senza cambiare riga, si utilizza `print` anziché `println`.

Utilizzando `print`, è possibile comunque mandare a capo scrivendo `\n` alla fine del testo da stampare:

```
system.out.print("testo da scrivere\n");
```

Per stampare a video il contenuto di una variabile si usa:

```
system.out.println(var);
```

Omettendo dunque “ ”.

A.9 Input da tastiera

Le stringhe in Java sono considerate come oggetti: non si tratta dunque di *array* (vettori) di `char` con zero terminale. La dichiarazione di una stringa è simile a quella di un `int` o un `float`:

```
String miastringa;
```

Per leggere una stringa da tastiera, occorre usare il metodo `readLine()` di `DataInputStream`, derivato da `FilterInputStream`, che è a sua volta contenuto in `Java.io.InputStream`.

Per accedere a `DataInputStream.readLine()` occorre creare un esempio di un oggetto di tipo `DataInputStream`. In Java per “istanziare” un oggetto si usa l'operatore `new`, in questo modo:

```
DataInputStream leggiLo = new DataInputStream(System.in);
```

Al costruttore passiamo lo stream (flusso) da cui prendere input, che nel nostro caso è `System.in`.

Ora abbiamo un `DataInputStream` chiamato `leggiLo`, che possiamo usare per leggere una stringa:

```
miaStringa = leggiLo.readLine(); // leggo stringa da tastiera
```

Ora dobbiamo convertire `miaStringa` in un valore `float`, e salvarla.

```
vara = Float.valueOf(stringa).floatValue(); // conversione
in float
```

Sono disponibili anche `Integer.valueOf()`, `Double.valueOf()`, eccetera.

Il `.floatValue()` finale indica che deve essere considerata anche la parte decimale, dopo la virgola (o, meglio, il punto). Se si fosse specificato:

```
vara = Float.valueOf(stringa).intValue();
```

La stringa sarebbe convertita in `float`, ma senza la parte dopo la virgola, ad esempio `23.43` sarebbe convertito come `23`.

Occorre anche gestire l'eventualità di un errore, che genererebbe un'eccezione. Le eccezioni vengono generate in caso di errore durante l'esecuzione (non si tratta dunque di errori puramente di sintassi, rivelati in fase di compilazione) e servono per far eseguire delle istruzioni di emergenza, come la stampa di messaggi di errore o per causare l'interruzione del programma.

Per gestire gli errori si usa la coppia di istruzioni `try` e `catch`. *Try* significa "prova", e *catch* "afferra, cogli".

Occorre inserire l'istruzione che può generare eccezioni nel blocco di `try`: "prova ad eseguire questa istruzione". Subito dopo, occorre mettere il `catch`, e nel suo blocco le istruzioni da eseguire in caso di errore:

"Se è avvenuta una eccezione, intercettata e fai le operazioni di emergenza predefinite". Il tutto applicato al nostro input:

```
try {
    // leggo stringa da tastiera
    stringa = leggilo.readLine();
    // e la converto in float.
    vara = Float.valueOf(stringa).floatValue();
}
catch (Exception e) {
    System.out.println("Errore: " + e + " nella lettura da
    tastiera");
    System.exit(0);
}
```

Come si vede, si provano ad eseguire la lettura e la conversione da tastiera e in caso di errore di stampa un messaggio, contenente il nome dell'eccezione (al catch è passata l'eccezione, che ha il nome "e"), e si termina l'esecuzione con la funzione `System.exit()`

A.9 Input di stringhe

Il linguaggio Java legge i valori in *bytes*, quindi è necessario definire un vettore di tipo `byte`:

```
byte name[] = new byte[100];
```

deve poi venir specificato un contatore, per vedere la lunghezza della stringa immessa:

```
int nr_read = 0;
```

poi si passa all'input vero e proprio:

```
nr_read = System.in.read(name);
```

per stamparlo:

```
System.out.write(name, 0, nr_read);
```

A.11 Per l'input di numeri

Si utilizza il seguente metodo, per i numeri interi:

```
static int getNextInteger() {
    String line;
    DataInputStream in = new DataInputStream(System.in);

    try {
        line = in.readLine();
        int i = Integer.valueOf(line).intValue();
        return i;
    }
    catch (Exception e) {
        return -1;
    }
}
```

il seguente serve invece per input di numeri di tipo float:

```
static float getNextfloat() {
    String line;
    DataInputStream in = new DataInputStream(System.in);

    try {
        line = in.readLine();
        float i = Float.valueOf(line).floatValue();
        return i;
    }
    catch (Exception e) {
        return -1;
    }
}
```

A.12 Vettori

Un *array* (vettore) è un insieme di elementi dello stesso tipo, cui è possibile accedere tramite un indice, che è il numero dell'elemento, partendo da zero. Per esempio con indice 0 si accede al primo elemento, con indice 3 si accede al quarto elemento. Questi elementi possono essere delle variabili `int`, `float` o di altro tipo.

Da notare che gli indici sono numeri interi non negativi, e vanno da 0 a $n-1$, dove n è il numero di elementi.

Gli indici possono essere variabili di tipo `int`, `short`, `byte`.

Agli *array* si può dare un nome a piacere, col quale si identificano tutti gli elementi (variabili) che lo compongono. In altre parole gli elementi condividono lo stesso nome (quello dell'*array* stesso).

Gli *array* in Java sono degli oggetti, a differenza del linguaggio C, ed esiste un controllo su di essi, in modo da non poter scrivere oltre il loro limite: se si tenta di accedere ad un *array* con un indice superiore all'ultimo elemento, si genera un'eccezione `ArrayIndexOutOfBoundsException` (indice fuori dai margini). Essendo oggetti non basta dichiararli con:

```
Tipo[] nome;      Esempio:   int[] mioarray;
```

Occorre anche “istanziarli”, allocando memoria per essi, tramite l'operatore `new`, in questo modo:

```
int[] mioarray = new int[10];
```

In questo caso dichiariamo e allochiamo un *array*, composto da 10 interi, di nome `mioarray`. Il suo indice andrà da 0 a 9.

Si può anche inizializzare subito, inserendo i valori all'interno:

```
int[] mioarray = {35,21,28,123,321};
```

Dichiarando i suoi elementi, si alloca come con l'operatore `new`: questo *array* avrà 5 elementi, con indice da 0 a 4.

Per leggere e scrivere nell'*array* si userà questa forma:

```
mioarray[0]=3;//Metto nel primo elemento il valore 3
```

Il numero che mettiamo tra parentesi quadre è l'indice, ossia il numero di elemento dell'*array*, partendo da 0.

Per definire un vettore è necessario mettere il simbolo `[]` dopo il tipo di variabile. Esistono anche vettori di stringhe:

```
String[] nomi;      crea un vettore che si chiama nomi, i cui elementi sono
                    nomi[0], nomi[1],
```


si deve poi allocare nel seguente modo:

```
nomi = new String[n];
```

dove n è la lunghezza desiderata

è poi necessario inizializzare i vettori, dando ad ogni componente un valore iniziale:

```
nomi[0] = "momo";
....
```

`nomevettore.length` è la variabile che ci dà la lunghezza di un vettore.

Si può anche dichiarare ed allocare in un colpo solo:

```
String[] names = new String[50];
```

Come indice di un vettore è possibile utilizzare una variabile, per poter riempire, con un ciclo iterativo, tutti gli elementi.

A.13 Variabile argomento:

<code>args[n]</code>	è vettore dove n da 0 a k identifica la posizione.
<code>args.length</code>	è la variabile che ci dà la lunghezza del vettore.

A.14 Matrici

Un *array* a 2 dimensioni si può considerare un *array* di *array* monodimensionali o, in altri termini, una matrice righe-colonne, molto simile ad una scacchiera o al reticolo della battaglia navale.

In realtà Java non gestisce *array* multidimensionali "veri" come il C, ma *array* di *array*, che però nella pratica funzionano allo stesso modo.

La differenza consiste anche nel fatto che le dimensioni si possono allocare separatamente, anziché tutte insieme.

Ecco come si definisce e alloca un *array* a 2 dimensioni:

```
tipo[] nome = new tipo[dimensione1] [dimensione2];
```

Come esempio, definiamo un *array* con 3 righe e 5 colonne, ossia $3*5=15$ elementi:

```
int[] tabelluccia = new int[3][5];
```

In questo modo creeremo una tabella del genere:

```
[0][0]  [0][1]  [0][2]  [0][3]  [0][4]
[1][0]  [1][1]  [1][2]  [1][3]  [1][4]
[2][0]  [2][1]  [2][2]  [2][3]  [2][4]
```

Con `[0][0]` si intende il primo elemento, e per raggiungere gli altri si devono variare i due indici opportunamente.

Per scrivere un valore nell'elemento `[1][2]`, è sufficiente scrivere:

```
tabelluccia[1][2]=10;           //Scrivo nell'elemento [1][2]
```

Il primo indice è quello che corrisponde alle righe, mentre il secondo corrisponde alle colonne.

A.15 Condizioni (comando if)

Controlliamo il flusso del programma, saltando istruzioni o eseguendole a seconda di qualche condizione, col costrutto `if`, che ha la forma:

```
if (condizione) istruzione;
```

La condizione è una espressione che può risultare vera o falsa, come vedremo per il `for`, ad esempio (`pippo<10`). *If* in inglese significa "se", quindi potremmo tradurre in: "se la condizione è vera, sarà eseguita l'istruzione, altrimenti no".

Il costrutto `if` ha un ramo opzionale, chiamato `else`, che in italiano significa "altrimenti":

```
if (condizione) istruzione1;
else istruzione2;
```

La traduzione potrebbe essere: "se la condizione è vera, esegui istruzione1, altrimenti esegui istruzione2". Quindi si tratta di una scelta: l'esecuzione di un'istruzione esclude l'altra. In altri termini, se la condizione è falsa viene eseguita istruzione2.

Per riassumere:

```
if (condizione) {comando da eseguire;}
else {comando 2;}
...
```

oppure

```
if (condizione) {comando da eseguire;}
else if (condizione 2) {comando2;}
...
```

Per completare la panoramica sull'if, vedremo come annidarlo. Con ciò si intende mettere una condizione if all'interno di altre condizioni if, ossia fissare più di una condizione per l'esecuzione delle istruzioni.

Ad esempio:

```
if (pippo>10)    // Prima condizione da superare
{
    if (pippo<20)    // Seconda condizione
        System.out.println("\nPippo è compreso tra 11
        e 19");
    else System.out.println("\nPippo è grande, non mi
        interessa");
}
else System.out.println("\nPippo è piccolo, non mi
interessa");
```

La traduzione potrebbe essere: "se pippo è maggiore di 10, esegui la seconda condizione if: se anche questa è vera, ossia se pippo è minore di 20, allora esegui il println relativo. Altrimenti stampiamo il printf relativo al nostro scarso interesse per pippo". In pratica la variabile pippo è sottoposta a due condizioni if.

A.16 Condizioni (comando switch)

In Java, oltre all'istruzione `if` tradizionale, esiste il costrutto `switch`, che permette di effettuare facilmente test multipli su una variabile. Abbiamo visto come fare dei test multipli tramite molti `if` annidati, ma quando vi sono molti "casi" in cui agiamo diversamente, è meglio usare `switch`, soprattutto per pulizia di codice.

Ecco la sua forma:

```
switch(variabile)
{
    case costante1:
        istruzione1;
        break;
    case costante2:
        istruzione2;
        break;
    case costante3:
        istruzione3;
        break;
    default:
        istruzione4;
}
```

In questo caso verranno effettuati tre confronti. Se la variabile non soddisfa nessuno dei tre requisiti di uguaglianza viene eseguita l'istruzione sotto, denominata `default`.

I `break` servono per fermare il confronto qualora uno dei casi sia risultato vero. Se non si mettono i `break`, i confronti continuano al `case` successivo. Se non si mette la parte `default`, nel caso in cui nessun confronto sia vero, non viene fatto niente.

Da notare che lo `switch` può verificare solo l'uguaglianza, a differenza degli `if` che possono verificare condizioni qualsiasi.

Potremmo usare lo `switch` per fare una specie di menù di selezione, in cui controlliamo quale voce (per esempio corrispondente ad un numero), è stata selezionata.

A.17 Loops (cicli)

Anche in Java, come nella maggior parte dei linguaggi di programmazione, esiste l'istruzione `for`, che ha questa forma:

```
for(inizializzazione ; condizione ; incremento)
    istruzione da eseguire;
```

- inizializzazione: istruzione di assegnamento, es: `pippo=1`
- condizione: espressione per valutare quando terminare, es: `pippo<10`
- incremento: come modifichiamo la variabile ogni *loop*, es: `pippo=pippo+1`

Nel caso in cui si desideri eseguire un blocco di istruzioni, queste vanno tra `{ }`.

Questi sono gli operatori condizionali (o relazionali) in Java:

```
>    Maggiore
<    Minore
>=   Maggiore o uguale
<=   Minore o uguale
==    Uguale
!=    Diverso
```

Gli ultimi due, `==` e `!=`, sono detti anche operatori di eguaglianza. In generale, possiamo dire che il *loop* prosegue se la condizione è vera, qualsiasi essa sia, e si ferma quando diviene "falsa".

```
for (indice=valore iniziale; indice < variabile; indice =
    indice + 1) {comando/i da ripetere;}
```

Per uscire da un ciclo `for` prima che l'indice abbia raggiunto il proprio valore finale, si utilizza il comando `break`. L'uso della istruzione è molto semplice: ogni volta che ci si trova all'interno di un ciclo, se si raggiunge una istruzione `break` si interrompe il ciclo e si passa direttamente alla prima istruzione che segue.

Oltre al `for`, in Java esiste anche il comando `while` con il seguente costrutto:

```
while (indice < variabile) {comando/i da ripetere; indice =
    indice + 1;}
```

Se invece si volesse fare in modo che le istruzioni fossero eseguite una volta almeno, e solo dopo fosse controllata la condizione per un eventuale *loop*, si dovrebbe usare il comando `do`, che in inglese significa "fai, esegui":

```
do
{
    istruzioni;
}
while (espressione);
```

In questo caso eseguiamo le istruzioni, e solamente dopo controlliamo se è vera la condizione e di conseguenza ripetiamo l'esecuzione delle istruzioni racchiuse tra `{ }`.

Per incrementare o diminuire l'indice del ciclo, si utilizzano le seguenti forme:

```
indice = indice + 1    si può scrivere come indice ++ (o --
                      all'occorrenza)
indice = indice + n    si può scrivere come indice += n (o -=
                      all'occorrenza)
```

Le istruzioni da eseguire in loop devono essere sempre comprese tra `{ }`.

A.18 Metodi

I metodi, chiamati funzioni in C, iniziano con una dichiarazione. Questa può includere da tre a cinque parti: la prima è uno *specifier* opzionale che può essere `public`, `private` o `protected`. Un metodo `public` può essere chiamato da ogni altra parte del programma. Un metodo dichiarato `private` può solamente essere utilizzato nel package in cui è definito. I metodi che non vengono specificatamente definiti come `public` o `private`, sono `protected` per default.

A questo punto si deve decidere se il metodo sia o meno `static`. I metodi `static` hanno solamente un esempio (*instance*) per classe, anziché un esempio (*instance*) per oggetto. Tutti gli oggetti di una stessa classe condividono una singola copia di un metodo `static`. Per default, i metodi non sono `static`.

In seguito, si specifica il cosiddetto *return type*, cioè il tipo di variabile che verrà restituito alla fine dell'esecuzione del metodo stesso. Se il *return type* è `void`, allora non viene restituito nessun valore.

A questo punto si inserisce il nome del metodo, a piacere.

Seguono delle parentesi `()` in cui inseriamo i nomi ed i tipi degli argomenti di un metodo. Un metodo può non avere argomenti o può averne molti: questi possono essere utilizzati all'interno del metodo, esattamente come si fa con le variabili locali.

Infine, il resto del metodo è racchiuso tra parentesi graffe `{ }` come un unico blocco. La parte centrale è il corpo del metodo, e termina con l'istruzione `return`, che invia un valore al metodo chiamante.

```
TipoRitorno NomeMetodo(Lista tipo e nome argomenti passati)
{
    corpo del metodo
    return;
}
```

A.19 I/O su files

In ogni linguaggio di programmazione, esistono istruzioni atte a gestire l'I/O, ossia Input/Output (Entrata/Uscita).

Il sistema I/O del Java ha un'interfaccia indipendente dall'hardware, analoga a quella UNIX del C, che pone un certo livello di astrazione tra il programmatore e il dispositivo in uso: è perciò analogo accedere a terminali connessi in rete, unità a disco, a nastro, e così via.

I diversi dispositivi sono considerati come un dispositivo logico chiamato flusso (detto *stream* o *pipe*). I flussi partono da una sorgente ed arrivano ad una destinazione, astraendosi dai dispositivi fisici su cui avvengono le operazioni.

I tipi di flusso in UNIX/ANSI C sono due:

1. **FLUSSO DI TESTO:** è una sequenza di caratteri, che però durante il trasferimento può subire anche delle conversioni secondo le necessità dell'ambiente di

destinazione (ad esempio il carattere di fine riga è diverso in certi casi: può essere di *newline* o *cr+lf*) il che implica che il flusso eventualmente convertito può essere anche di dimensione diversa. Non si deve far passare da questo flusso un file non di testo.

2. **FLUSSO BINARIO:** è una sequenza di byte avente una corrispondenza uno a uno con la sequenza ricevuta dal dispositivo esterno. Non avvenendo alcuna conversione, il numero di byte scritti/letti rimane uguale.

In Java sono disponibili anche flussi filtrati più sofisticati. In precedenza ho trattato comandi per I/O da e verso la console (sottosistema tastiera/video), ora passo al caso di lettura/scrittura da/verso file.

In Java le superclassi da cui sono derivati i vari flussi particolari sono `InputStream` e `OutputStream`. Le subclassi per i flussi su file sono `FileInputStream` e `FileOutputStream`, dedicate al flusso di file.

Ad esempio:

```
FileInputStream MioInFil = new FileInputStream("Prog.Java");
FileOutputStream MioOutFil = new FileOutputStream("Prog.bak");
```

Queste classi però hanno metodi che possono leggere e scrivere soltanto *bytes*, e non altri tipi di dati. Se si desiderano leggere/scrivere degli `int`, dei `float`, o altri dati, si può convertire/filtrare lo *stream* usando delle classi come

```
DataInputStream MioInDat = new DataInputStream(MioInFil);
DataOutputStream MioOutDat = new DataOutputStream(MioOutFil);
```

A questo punto saranno disponibili dei metodi più sofisticati: `readChar()`, `readBoolean()`, `readFloat()`, `readLine()`; `writeChar()`, `writeFloat()`, e così via.

Le eccezioni generate da queste classi sono di tipo `IOException`:

```
try {
    MioInFil = new FileInputStream("Prog.Java");
}
catch (IOException e) {
```



```

        System.out.println("Errore: " + e + " nella lettura di
        un file");
        System.exit(1);
    }

```

A.20 Gestione file

Il Java ha una classe `File`, molto meno astratta di `FileInputStream` e `FileOutputStream`, specializzata nel gestire files sulla macchina *host* (quella su cui viene eseguito il programma).

Essa fa riferimento ad un file sul disco, e serve in particolare per il supporto di *path* e nomi file particolari della macchina specifica.

È possibile anche avere informazioni sugli attributi del file, e le directory sono considerate file listabili.

I metodi più importanti della classe `File` sono:

<code>canRead()</code> ;	Restituisce <code>true</code> se è leggibile, altrimenti <code>false</code>
<code>canWrite()</code> ;	Restituisce <code>true</code> se è scrivibile, altrimenti <code>false</code>
<code>equals(Object)</code> ;	Confronta il file con un altro
<code>exists()</code> ;	Restituisce <code>true</code> se il file esiste, altrimenti <code>false</code>
<code>getPath()</code> ;	Restituisce il <i>path</i> (solo quello indicato da noi)
<code>getAbsolutePath()</code> ;	Restituisce il <i>path</i> assoluto (es: <code>c:\javcors\..\..\</code>)
<code>isDirectory()</code> ;	Ritorna <code>true</code> se esiste ed è una directory, alt. <code>false</code>
<code>isFile()</code> ;	Ritorna <code>true</code> se esiste ed è un file, altrim. <code>false</code> .
<code>length()</code> ;	Restituisce la lunghezza del file
<code>list()</code> ;	Lista i file di una directory. Restituisce un array di Stringhe con i nomi.
<code>mkdir()</code> ;	Crea una dir e restit. <code>true</code> se ha avuto successo.
<code>renameTo(File)</code> ;	Rinomina un file e restit. <code>true</code> se ha successo.

Per “istanziare” un `File` basta passare il nome e/o *path* al costruttore:

```
File MioFile1 = new File("prova.java");
File MioFile2 = new File("c:/marco/appunti/prova2.txt");
```

I metodi di `java.io.File` non causano eccezioni `IOException`, quindi non sono necessari i `try-catch`. Se un'operazione non ha avuto successo i metodi restituiscono `false` anzichè `true`, per informarci.

NOTA: I separatori di percorso su Unix (e dunque Linux) sono `/`, mentre su Ms-Dos sono costituiti *backslash* `\`. In questo caso Java permette di usare `/` anche su Ms-Dos.

Appendice B

PROGRAMMAZIONE AD OGGETTI IN JAVA

B.1 Classi

Java è un linguaggio *Object Oriented*, e ogni oggetto può essere costituito come una classe: ogni classe deve essere contenuta in un file con estensione `.class` e nome della classe stessa. Se si definiscono più classi nello stesso sorgente `.java`, automaticamente, dopo la compilazione verranno creati i vari *file class*.

La classe è un tipo astratto usato per definire soltanto l'organizzazione di variabili e istruzioni di un dato oggetto, e non è l'oggetto fisico utilizzabile.

Si dovrà istanziare (ossia creare fisicamente) un oggetto da una classe, cioè un oggetto, di cui decidiamo il nome, la cui organizzazione interna è stata definita astrattamente nella classe di appartenenza.

Una classe si crea in questo modo:

```
[accesso] [modificatori] class nomeclasse { ... }
```

In `[accesso]` si può facoltativamente mettere `public`, `protected` o `private`, che determinano la visibilità della classe rispetto alle altre. In altre parole determinano il livello di accesso.

Una classe/variabile/metodo di tipo `public` è visibile da tutte le classi esterne, ossia i metodi delle altre classi possono accedervi.

Dichiarare tutto `public` semplifica le cose, ma rende scarsa la sicurezza di un programma.

Se non si indica nulla, si ha un livello di protezione simile a `public`, ma un po' più ristretto, in quanto non si può accedere dall'esterno del package.

Non è possibile specificare "manualmente" questo tipo di accesso con una parola chiave: l'unico modo è non metterne alcuna.

Un po' più restrittivo è l'attributo di tipo `protected`, che permette l'accesso solo alla stessa classe e alle classi derivate, cioè figlie.

Occorre scrivere `private protected`, e non solo `protected` per ottenere questo livello di protezione.

Il livello di protezione più alto è `private`, che rende visibili metodi e variabili soltanto alle classi in cui sono definite.

Tutto ciò che non serve alle classi esterne e che non deve essere condiviso con esse, deve essere reso `private`, in modo da rendere effettivamente impossibile un caso di accesso dall'esterno.

Rese `private` queste variabili, saranno utilizzabili dalle classi esterne attraverso i metodi pubblici della classe, mai direttamente.

Sarà necessario definire due classi anziché una nel sorgente: la prima sarà quella principale, con il `main()`, nella quale “istanzieremo” oggetti della seconda classe.

Definire tutti i metodi e le variabili come `public` e/o `static` porta a usare il Java come un linguaggio non Object Oriented, e ciò limita molto le sue possibilità. Definire una classe di tipo `static`, permette infatti di utilizzarla senza dover creare una instance.

B.2 Costruttore di classe

Il costruttore è un metodo speciale che ha lo stesso nome della classe:

```
animale() {...}           // Costruttore
```

Il costruttore non può restituire alcun valore, per cui non si fa precedere dal tipo di restituzione (nemmeno da `void`):

```
class animale {           // Definisco la classe animale

    private float lunghezza;    // Le variabili private
    private int numerozampe;

    animale() {              // Il costruttore. Non ha parametri in
                              entrata
        lunghezza = 0;
        numerozampe = 0;
        System.out.println("animale inizializzato");
    }
}
```

```

        } // Fine costruttore
// Eventuali altri metodi

    } // Fine classe

```

Il costruttore viene eseguito attraverso il comando `new`.

È possibile passare dei parametri al costruttore di una classe; rispetto all'esempio precedente avremo:

```

animale(int zamp) { // Il costruttore. 1 parametro in
    entrata
    lunghezza = 0;
    numerozampe = zamp;
    System.out.println("animale inizializzato");
} // Fine costruttore

```

Il parametro verrà passato in sede di istruzione `new`, con il seguente costrutto:

```

animale cane = new animale(4);

```

B.3 Polimorfismo dei metodi

Si definisce polimorfismo la possibilità che si ha di effettuare una sovrapposizione (*overload*) di metodi aventi lo stesso nome, ma con parametri di tipo diverso.

```

int somma(int a,int b) {...}
long somma(long c,long d) {...}
float somma(float e,float f) {...}

```

vengono considerati solo i tipi dei parametri (`int`,`long`,`float`), non i nomi delle variabili (`a`,`b`,`c`); è dunque errore dichiarare due metodi come questi:

```

int somma(int a,int b) {...}
int somma(int c,int d) {...}

```

Da notare che il tipo di valore restituito dal metodo (ritorno) non conta quando il compilatore deve scegliere che metodo chiamare.

B.4 Polimorfismo delle classi

Come già per i metodi, è possibile effettuare una sovrapposizione (*overload*) anche dei costruttori di classe.

Per richiamare una classe polimorfica, è sufficiente utilizzare più di un `new` sullo stesso costruttore, ma con parametri diversi, nel modo seguente:

```
animale cane1 = new animale();           // Nessun parametro
animale cane2 = new animale(4);         // 1 parametro
animale millepiedi = new animale(1000,3); // 2 parametri
```

Mentre per costruire la classe polimorfica, basterà inserire più costruttori all'interno del codice.

```
class animale {           // Definisco la classe animale

    private float lunghezza=0; // Variabile privata
    private int numerozampe=0; // Variabile privata

    animale() {           // Il costruttore. Nessun
                        parametro in entrata
        lunghezza = 0;
        numerozampe = 0;
    }           // Fine costruttore

    animale(int zamp) {   // Il costruttore. 1
                        parametro in entrata
        lunghezza = 0;
        numerozampe = zamp;
    } // Fine costruttore

    animale(int zamp, float lung) { // Il costruttore.
                                    2 parametri
        lunghezza = lung;
        numerozampe = zamp;
    } // Fine costruttore
    .....
    .....
}           // Fine della classe animale
```

B.5 Ereditarietà (inheritance) delle classi

Come anche in C, in Java è possibile derivare delle sottoclassi da una classe precedentemente definita, che si differenziano da quella *parent* per alcuni dettagli. Le sottoclassi erediteranno variabili e metodi dalla classe *parent* e si potranno aggiungere manualmente altre variabili e metodi, specifici della sola classe derivata.

Per derivare una classe da un'altra, si usa questa sintassi:

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

Nel listato sarà dunque possibile utilizzare contemporaneamente istanze della superclasse e delle classi derivate.

È possibile derivare classi da sottoclassi già derivate, creando dunque una sorta di “albero genealogico”.

B.6 Variabile di tipo static

Quando si “istanziano” degli oggetti da una classe, ognuno di questi dispone di copie personali dei metodi e delle variabili definite dalla classe di appartenenza.

Cioè, se *a* e *b* sono due classi di tipo “animale”, le due variabili *a.numero zampe* e *b.numero zampe* sono autonome, per cui cambiando una non si modifica l'altra.

È possibile però definire *static* degli elementi di una classe: ciò significa che se ne fa esistere una sola copia, condivisa da tutti gli oggetti “istanziati”.

In questo modo, tale variabile sarà condivisa da tutti gli oggetti “istanziati” dalla classe, e scrivere in essa da uno degli oggetti significherà cambiarne il valore in tutti gli altri. Per dichiarare statica una variabile, basta usare *static*:

```
static int variabile;
```

È necessario prestare molta attenzione all'uso di variabili di tipo *static*, in quanto spesso non sono necessarie e se utilizzate possono compromettere il funzionamento corretto del programma, venendo modificate da più di un metodo o più di un oggetto.

B.7 Fermare un programma Java

In Java non esiste un'istruzione END, come in molti altri linguaggi. Per bloccare l'esecuzione del programma è necessario utilizzare il metodo `System.exit(0)`.

Con *Jdk1.3* e' stato aggiunto il metodo `Runtime.addShutdownHook()` che consente di specificare un codice per un chiaro *clean up* e dovrebbe girare prima di uscire da un' applicazione.

B.8 Thread e Multithreading

Un *thread* è una parte di codice che può essere eseguito senza monopolizzare il computer: diventa così possibile svolgere diverse attività contemporaneamente, condividendo in modo ottimale le risorse di sistema.

In Java è assolutamente necessario far eseguire più operazioni allo stesso tempo, perchè si possono avere più applet che girano in modo indipendente nello stesso istante, per esempio un orologio, un gioco e così via ed è molto importante soprattutto nelle animazioni.

Consideriamo infatti il caso di un'applet che esegue un *thread* di animazione. Mentre il *thread* viene eseguito, l'utente deve poter usare il mouse per svolgere altre operazioni, per esempio per continuare a scorrere una pagina HTML, e questo è possibile solo grazie ai *thread*.

Una delle classi che fanno parte delle librerie standard di Java è la classe `Thread`. Essa possiede tutti i metodi necessari per far partire contemporaneamente più *thread* in un unico programma Java, nonché di arrestare gli stessi, di farli ripartire e fornire indicazioni sul loro stato. Java gestisce i vari *thread* in maniera *pre-emptive*, ovvero essi possono essere controllati completamente. Un *thread* può essere interrotto in qualsiasi momento e al suo posto ne può partire un altro. Per aumentare la velocità di esecuzione in compiti particolarmente gravosi, è possibile anche sospendere il controllo di un *thread* per mezzo del metodo `yield`.

Attraverso l'istruzione `synchronized` è possibile mandare in esecuzione più metodi che accedono alle stesse variabili "istanziate" senza che vi sia il pericolo che il valore delle variabili a cui un metodo in esecuzione sta accedendo sia cambiato da un altro metodo. Più in generale, attraverso l'istruzione `synchronized` si impone che lo stato di

un oggetto sia cambiato solo da un metodo alla volta. Buona parte delle caratteristiche *multi-thread* di Java sono state fornite dai linguaggi di programmazione *Mesa* e *Cedar* (un superset del *Mesa*, sempre implementato dalla *Xerox*).

Un linguaggio di programmazione che consente l'utilizzo dei *thread*, si dice che supporta il *multithreading*.

A volte, un programma che è in grado di implementare i *thread* viene definito *multitasking*, perchè ogni *thread* è visto come un *task* (un compito, un processo) che può essere svolto simultaneamente ed in concorrenza con altri.

La programmazione in *multithreading* offre grandi potenzialità, perché consente di sfruttare al massimo le risorse del sistema a disposizione, però occorre tenere ben presente anche i numerosi problemi che esso comporta. Infatti l'utilizzo condiviso delle risorse del sistema può causare un conflitto fra *thread*, ad esempio per il tentativo di leggere o scrivere sullo stesso file.

Java mette a disposizione tutte le funzionalità necessarie per implementare il *multithreading*, comprese quelle che consentono di applicare le necessarie protezioni.

Il principale problema posto dal *multithreading* è quello della sincronizzazione, che può essere chiarito meglio con un esempio; consideriamo questo stralcio di codice:

```
public void metodo() {
    if (valore < 0) {
        // istruzioni
        valore += 1;
    }
}
```

Anche se si tratta di istruzioni semplicissime, considerate dal punto di vista del *multithreading*, possono introdurre un problema molto grave. Infatti lo stesso metodo potrebbe venire richiamato da due *thread* distinti, e mentre un *thread* ha appena effettuato un controllo di valore e ha determinato che è minore di zero, l'altro potrebbe incrementare lo stesso valore facendolo divenire uguale o maggiore di zero senza che il primo se ne accorga, causando quindi un'errore alla successiva esecuzione. Questo problema nei casi più semplici si può risolvere utilizzando la parola chiave *synchronized* nella dichiarazione del metodo, come nel seguente esempio:

```
public synchronized void metodo () { ... istruzioni ... }
```

In questo caso le istruzioni vengono protette dai *thread*, nel senso che soltanto un *thread* per volta può eseguire il corpo del metodo, gli altri devono rimanere in attesa. Naturalmente questo può portare ad un peggioramento delle prestazioni complessive, ma non c'è alternativa.

L'oggetto `java.util.ThreadGroup` ci viene in aiuto in quanto ci permette di unire i vari *thread* in un unico gruppo. Per fermare tutti i *thread* del gruppo, basterà chiamare il metodo `stop()` dell'oggetto `ThreadGroup`. I gruppi di *thread* non sono solo insiemi di *thread* ma permettono di organizzare i *thread* secondo una struttura ad albero. Ogni gruppo di *thread*, tranne quello principale (*root*) dispone di un gruppo genitore.

B.9 Utilizzo pratico dei thread

In Java, si possono creare *thread* in due modi; il più semplice è quello di trasformare una classe già esistente in un *thread*. Per far questo si inserisce un'interfaccia all'interno della classe stessa, che dichiara il metodo `run()` richiesto da tutti i tipi di *thread*. Questo metodo contiene il codice che verrà eseguito dal *thread* stesso.

Il secondo metodo per creare un *thread* consiste nello scrivere una classe separata, derivata dalla superclasse `Thread` presente in Java. Poiché quest'ultima comprende già l'interfaccia richiesta, le classi derivate avranno un metodo `run()` senza doverlo creare. Tuttavia questo metodo non fa nulla: è necessario riscriverlo per poter creare il tipo di *thread* desiderato.

B.10 Conversione di una classe in thread

Per attuare la conversione sono necessari i seguenti passaggi:

1. Si dichiara la classe e si "implementa" l'interfaccia `Runnable`

```
public class sottoClasse extends SuperClasse
    implements Runnable
```

2. Si costruisce il metodo `run()`

```
public void run()
{
}
```

all'interno di questo metodo si inserisce il codice da far eseguire al *thread*

3. Si dichiara l'oggetto di tipo `Thread`

```
Thread nomethread;
```

4. Si inserisce il costruttore in un metodo `startThread()`

```
public void startThread()
{
    thread = new Thread(this);
    thread.start();
}
```

da qui si lancia il *thread*, che andrà ad eseguire ciò che è contenuto nel metodo `run()` prima definito.

5. Si definisce un metodo `stop ()` per terminare un *thread*

```
public void stop ()
{
    thread.stop();
}
```

Appendice C

IL CODICE DI JVE CON SCORTE

C.1 StartVEFrame.java

```
// StartVEFrame.java - jveframe application
// Pietro Terna (October 2000-January 2001)
// Dipartimento di Scienze economiche e finanziarie G.Prato
// Università di Torino

// Some building ideas come from jHeatbugs (Java Heatbugs application.
// Copyright (c) 1999-2000 Swarm Development Group.)

import swarm.Globals;

/**
 * The StartVEFrame class contains main().
 * We follow here the typical Swarm structure with main() (in Start...
 * as a Java convention) generating the observer and the observer generating the model.
 * [We could (?) also use Observer or Model as a class directly as classes defining
 * methods and classes outside methods, in a static way]
 *
 * @author Pietro Terna
 */
public class StartVEFrame
{
    /** The main() function is the top-level place where everything starts.
     * For a typical Swarm simulation, in main() you create a toplevel Swarm (the
     * observer), let it build and activate, and set it to running.
     */

    public static void main (String[] args) {
        // Swarm initialization: all Swarm apps must call this first.
        Globals.env.initSwarm ("jveframe", "0.3.3", "pietro.terna@unito.it",
                               args);

        //creating the VEFrameObserver
        VEFrameObserverSwarm vEFrameObserver = new
            VEFrameObserverSwarm (Globals.env.globalZone);
        // to save window position
        Globals.env.setWindowGeometryRecordName
            (vEFrameObserver, "vEFrameObserver");

        // build objects into the observer
        vEFrameObserver.buildObjects ();

        // build actions into the observer
        vEFrameObserver.buildActions ();

        // activate
        vEFrameObserver.activateIn (null);

        // go() in vEFrameObserver is a method inherited from class
        // swarm.simtoolsgui.GUISwarmImpl
        // go() start the activity running, and also handle user requests
        // via the control panel.
        vEFrameObserver.go ();

        // concluding the activity we drop the observer; is this drop message
        // absolutely necessary?
        vEFrameObserver.drop ();}}
}
```

C.2 VEFrameObserverSwarm.java

```
// VEFrameObserverSwarm.java

import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;

import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;

import swarm.objectbase.Swarm;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;

import swarm.simtoolsgui.GUISwarm;
import swarm.simtoolsgui.GUISwarmImpl;

import swarm.analysis.EZGraph;
import swarm.analysis.EZGraphImpl;

import java.util.List;

public class VEFrameObserverSwarm extends GUISwarmImpl {
    /** one parameter: update freq */
    public int displayFrequency;

    /** ActionGroup for sequence of GUI events */
    public ActionGroup displayActions;
    /** the single Schedule instance */
    public Schedule displaySchedule;

    /** the Swarm we're observing */
    public VEFrameModelSwarm vEFrameModelSwarm;

    // a graph
    EZGraphImpl waitingListGraph, warehouseGraph;

    /** Constructor for class */
    public VEFrameObserverSwarm (Zone aZone) {
        super(aZone);

        // Fill in the relevant parameters (only one, in this case).
        displayFrequency = 1;

        // Now, build a customized probe map using a 'local' subclass
        // (a special kind of Java 'inner class') of the
        // EmptyProbeMapImpl class. Without a probe map, the default
        // is to show all variables and messages. Here we choose to
        // customize the appearance of the probe, give a nicer
        // interface.

        // NB genera un file class a parte, con nome xxx$yyy.class

        class VEFrameObserverProbeMap extends EmptyProbeMapImpl {
            private VarProbe probeVariable (String name) {
                return
                    Globals.env.probeLibrary.getProbeForVariable$inClass
                        (name, VEFrameObserverSwarm.this.getClass ());
            }
            private MessageProbe probeMessage (String name) {
                return
                    Globals.env.probeLibrary.getProbeForMessage$inClass
                        (name, VEFrameObserverSwarm.this.getClass ());
            }
            private void addVar (String name) {
```

```

        addProbe (probeVariable (name));
    }
    private void addMessage (String name) {
        addProbe (probeMessage (name));
    }
    public VEFrameObserverProbeMap (Zone _aZone, Class aClass) {
        super (_aZone, aClass);
        addVar ("displayFrequency");
    }
}

// Install our custom probeMap class directly into the
// probeLibrary
Globals.env.probeLibrary.setProbeMap$For
    (new VEFrameObserverProbeMap (aZone, getClass ()), getClass ());
}

/**
 * Create the objects used in the display of the model. */
public Object buildObjects () {

    super.buildObjects ();

    // First, we create the model that we're actually observing. The
    // model is a subswarm of the observer.

// ?? provare a usare static e non creare l'instance

    VEFrameModelSwarm = new VEFrameModelSwarm (getZone ());

    // Now create probe objects on the model and ourselves. This gives a
    // simple user interface to let the user change parameters.

    Globals.env.createArchivedProbeDisplay (vEFrameModelSwarm,
        "vEFrameModelSwarm");
    Globals.env.createArchivedProbeDisplay (this, "vEFrameObserverSwarm");

    // Instruct the control panel to wait for a button event: we
    // halt here until someone hits a control panel button so the
    // user can get a chance to fill in parameters before the
    // simulation runs
    getControlPanel ().setStateStopped ();

    // OK - the user has specified all the parameters for the
    // simulation. Now we're ready to start.

    // First, let the model swarm build its objects.
    vEFrameModelSwarm.buildObjects ();

    // Create the graph widget to display waitingLists
    waitingListGraph = new EZGraphImpl
        (getZone (),
        "waiting_lists",
        "time", "waiting lists",
        "waiting_lists");

    Globals.env.setWindowGeometryRecordName (waitingListGraph,
        "waitingListGraph");

    // create the data for the waitingListGraph
    try {
        waitingListGraph.createAverageSequence$withFeedFrom$andSelector
            ("averageWaitingList", vEFrameModelSwarm.getUnitListStatic (),
            new Selector (Class.forName ("Unit"), "getWaitingListLength",
                false));
        waitingListGraph.createMinSequence$withFeedFrom$andSelector
            ("minWaitingList", vEFrameModelSwarm.getUnitListStatic (),
            new Selector (Class.forName ("Unit"), "getWaitingListLength",
                false));
        waitingListGraph.createMaxSequence$withFeedFrom$andSelector
            ("maxWaitingList", vEFrameModelSwarm.getUnitListStatic (),
            new Selector (Class.forName ("Unit"), "getWaitingListLength",
                false));
    }
}

```

```

    } catch (Exception e) {
        System.err.println ("Exception getWaitingListLength: "
            + e.getMessage ());
    }

    // Create the graph widget to display warehouses' load (Remondino)

    if (vEFrameModelSwarm.getWarehouse()==1)    // if warehouses are present, draws the
graph (Remondino)
    {
        warehouseGraph = new EZGraphImpl
        (getZone (),
        "warehouses_load",
        "time", "load",
        "warehouses_load");

        Globals.env.setWindowGeometryRecordName (warehouseGraph, //Remondino
            "warehouseGraph");

        // create the data for the warehouseGraph (Remondino)
        try {
            warehouseGraph.createAverageSequence$withFeedFrom$andSelector
            ("averageLoad", vEFrameModelSwarm.getWarehouseListStatic (),
            new Selector (Class.forName ("Warehouse"), "getCounterValue",
            false));
            warehouseGraph.createMinSequence$withFeedFrom$andSelector
            ("minLoad", vEFrameModelSwarm.getWarehouseListStatic (),
            new Selector (Class.forName ("Warehouse"), "getCounterValue",
            false));

            warehouseGraph.createMaxSequence$withFeedFrom$andSelector
            ("maxLoad", vEFrameModelSwarm.getWarehouseListStatic (),
            new Selector (Class.forName ("Warehouse"), "getCounterValue",
            false));

        } catch (Exception e) {
            System.err.println ("getCounterValue: "
                + e.getMessage ());
        }
    }

    return this;
}

/**
    Create the actions necessary for the simulation. This is where
    the schedule is built (but not run!) Here we create a display
    schedule - this is used to display the state of the world and
    check for user input. This schedule should be thought of as
    independent from the model - in particular, you will also want
    to run the model without any display. */
public Object buildActions () {
    super.buildActions();

    // First, let our model swarm build its own schedule.
    vEFrameModelSwarm.buildActions();

    // Create an ActionGroup for display: a bunch of things that
    // occur in a specific order, but at one step of simulation
    // time. Some of these actions could be executed in parallel,
    // but we don't explicitly notate that here.
    displayActions = new ActionGroupImpl (getZone());

    // Add the methods to the ActionGroup
    try {
        // Schedule the update of the probe displays
        displayActions.createActionTo$message
        (Globals.env.probeDisplayManager,
        new Selector (Globals.env.probeDisplayManager.getClass (),

```

```

        "update", true));

displayActions.createActionTo$message
    (waitingListGraph,
     new Selector (waitingListGraph.getClass(), "step", true));

    displayActions.createActionTo$message
    (warehouseGraph,
     new Selector (warehouseGraph.getClass(), "step", true));

// Finally, schedule an update for the whole user
// interface code. This is crucial: without this, no
// graphics update and the control panel will be
// dead. It's best to put it at the end of the display
// schedule
displayActions.createActionTo$message
    (getActionCache (), new Selector
     (getActionCache ().getClass (), "doTkEvents", true));
} catch (Exception e) {
    System.err.println ("Exception in setting up displayActions in " +
                        "VEFrameObserverSwarm: "
                        + e.getMessage ());
}

// And the display schedule. Note the repeat interval is set
// from our own Swarm data structure. Display is frequently
// the slowest part of a simulation, so redrawing less
// frequently can be a help.
// note frequency!
displaySchedule = new ScheduleImpl (getZone (), displayFrequency);
// insert ActionGroup instance on the repeating Schedule
// instance
displaySchedule.at$createAction (0, displayActions);

return this;
}
/**
    activateIn: - activate the schedules so they're ready to run.
    The swarmContext argument has to do with what we were activated
    *in*. Typically the ObserverSwarm is the top-level Swarm, so
    it's activated in "null". But other Swarms and Schedules and
    such will be activated inside of us. */

public Activity activateIn (Swarm swarmContext) {
    // First, activate ourselves (just pass along the context).
    super.activateIn (swarmContext);

    // Activate the model swarm in ourselves. The model swarm is a
    // subswarm of the observer swarm.
    VEFrameModelSwarm.activateIn (this);

    // Now activate our schedule in ourselves. This arranges for
    // the execution of the schedule we built.
    displaySchedule.activateIn (this);

    // Activate returns the swarm activity - the thing that's ready to run.
    return getActivity();
}}

```


C.3 VEFrameModelSwarm.java

```
// VEFrameModelSwarm.java

import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;
import swarm.defobj.SymbolImpl;

import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;

import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;

import swarm.collections.ListImpl;

public class VEFrameModelSwarm extends SwarmImpl
{
    // simulation parameters
    public int totalUnitNumber, maxStepNumber, maxInWarehouses;
    public int useWarehouses; // Remondino

    // simulation public object instances
    public OrderGenerator orderGenerator;

    /** ActionGroup for holding an ordered sequence of action */
    public ActionGroup modelActions;
    /** the single Schedule */
    public Schedule modelSchedule;

    private RuleMaster ruleMaster; // Remondino

    /** the list of the units */
    public ListImpl unitList, unitListStatic, warehouseListStatic;

    public VEFrameModelSwarm (Zone aZone) {
        super (aZone);

        // Now fill in various simulation parameters with default values.
        totalUnitNumber = 3;
        maxStepNumber=5;
        useWarehouses=1; // Remondino
        maxInWarehouses=3; // Remondino

        // Now, build a customized probe map using a 'local' subclass (a
        // special kind of Java 'inner class') of the EmptyProbeMapImpl
        // class. Without a probe map, the default is to show all
        // variables and messages. Here we choose to customize the
        // appearance of the probe, to display a nicer interface.

        class VEFrameModelProbeMap extends EmptyProbeMapImpl {
            private VarProbe probeVariable (String name) {
                return
                    Globals.env.probeLibrary.getProbeForVariable$inClass
                        (name, VEFrameModelSwarm.this.getClass ());
            }
            private MessageProbe probeMessage (String name) {
                return
                    Globals.env.probeLibrary.getProbeForMessage$inClass
                        (name, VEFrameModelSwarm.this.getClass ());
            }
        }

        private void addVar (String name) {
```

```

        addProbe (probeVariable (name));
    }
    private void addMessage (String name) {
        addProbe (probeMessage (name));
    }
    public VEFrameModelProbeMap (Zone _aZone, Class aClass) {
        super (_aZone, aClass);
        addVar ("totalUnitNumber");
        addVar ("maxStepNumber");
        addVar ("useWarehouses"); // Remondino
        addVar ("maxInWarehouses"); // Remondino
    }
}

// Now, install our custom probeMap class directly into the
// probeLibrary
Globals.env.probeLibrary.setProbeMap$For
    (new VEFrameModelProbeMap (aZone, getClass ()), getClass ());
}

/**
 * Now it's time to build the model objects. We use various
 * parameters inside ourselves to choose how to create things.
 */
public Object buildObjects ()
{
    int i;

    // allow our parent class to build anything.
    super.buildObjects();

    // the list of the production units
    unitList = new ListImpl (Globals.env.globalZone);
    unitListStatic = new ListImpl (Globals.env.globalZone);
    warehouseListStatic = new ListImpl (Globals.env.globalZone); // (Remondino)

    // Create the orderGenerator
    orderGenerator = new OrderGenerator
        (Globals.env.globalZone, totalUnitNumber, maxStepNumber, unitList);

    // Create RuleMaster. (Remondino)

    ruleMaster = new RuleMaster(maxInWarehouses);

    // if warehouses are not present, Create only units.

    if (useWarehouses==0)
    {

        for (i = 1; i <= totalUnitNumber; i++) {

            Unit aUnit;

            aUnit = new Unit (Globals.env.globalZone, i, totalUnitNumber, unitList, ruleMaster,
useWarehouses);

            // Add the unit to the end of the list.
            unitList.addLast (aUnit);
            unitListStatic.addLast (aUnit);

        }

    }

    // if warehouses are present, Create warehouses and units. (Remondino)

    if (useWarehouses==1)
    {
        for (i = 1; i <= totalUnitNumber; i++)
        {
            Warehouse myWarehouse;
            Unit aUnit;

```

```

        myWarehouse = new Warehouse (Globals.env.globalZone, i);
        System.out.println (myWarehouse.getCounterValue()+" pieces in warehouse "+
i); // Prints how many objects are in the warehouse.

        aUnit = new Unit (Globals.env.globalZone, i, totalUnitNumber,
unitList, ruleMaster, useWarehouses, myWarehouse);

        // Add the unit to the end of the list.

        unitList.addLast (aUnit);
        unitListStatic.addLast (aUnit);

        warehouseListStatic.addLast (myWarehouse);
    }
    else System.out.println ("No warehouses in simulation");

    return this;
}

/**
 * Here is where the model schedule is built, the data structures
 * that define the simulation of time in the mode. The core is an
 * actionGroup that has a list of actions. That's then put in a
 * Schedule. */
public Object buildActions ()
{
    Selector sel;

    super.buildActions();

    // Create the list of simulation actions. We put these in
    // an action group, because we want these actions to be
    // executed in a specific order, but these steps should
    // take no (simulated) time. The M(foo) means "The message
    // called <foo>". You can send a message To a particular
    // object, or ForEach object in a collection.

    modelActions = new ActionGroupImpl (getZone ());

    try
    {
        // Note the use here of the forName() method of the
        // "Class" class to pass the first argument to the
        // creation of the selector. The first argument to the
        // creation of a selector requires an object of type Class
        // that identifies the class of the object to which the
        // message should be sent, in this case a Unit. If we
        // had an instance of a Unit available, say unit,
        // that first argument could use the getClass() method
        // derived for all classes from the superclass, Object.
        // The argument would be "unit.getClass()".
        sel = new Selector(Class.forName("Unit"),
            "unitStep1", false);
        // keep the 3th argument always to false
        modelActions.createActionForEach$message(unitListStatic, sel);
    } catch (Exception e)
    {
        System.err.println("Exception unitStep1: " +
            e.getMessage ());
        System.exit(1);
    }

    try
    {
        // We have here an instance of OrderGenerator, named
        // orderGenerator, so we use the getClass() method
        // to check for the consistency of the message sent to the object
        sel = new Selector(orderGenerator.getClass(),
            "createRandomOrderWithNSteps", false);
        modelActions.createActionTo$message(orderGenerator, sel);
    }
}

```

```

    } catch (Exception e)
    {
        System.err.println("Exception createRandomOrderWithNSteps: " +
            e.getMessage ());
        System.exit(1);
    }

try
{
    sel = new Selector(Class.forName("Unit"),
        "unitStep2", false);
    // keep the 3th argument always to false
    modelActions.createActionForEach$message(unitListStatic, sel);
} catch (Exception e)
{
    System.err.println("Exception unitStep2: " +
        e.getMessage ());
    System.exit(1);
}

// 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. This
// schedule has a repeat interval of 1, it will loop every
// time step. The action is executed at time 0 relative to
// the beginning of the loop.

// This is a simple schedule, with only one action that is
// just repeated every time. See jmousetrap for more
// complicated schedules.

modelSchedule = new ScheduleImpl (getZone (), 1);
modelSchedule.at$createAction (0, modelActions);

return this;
}

/**
 * Now set up the model's activation. swarmContext indicates where
 * we're being started in - typically, this model is run as a
 * subswarm of an observer swarm. */
public Activity activateIn (Swarm swarmContext) {
    // First, activate ourselves via the superclass
    // activateIn: method. Just pass along the context: the
    // activity library does the right thing.
    super.activateIn (swarmContext);

    // Now activate our own schedule.
    modelSchedule.activateIn (this);

    // Finally, return our activity.
    return getActivity ();
}

public Object getUnitListStatic (){
    return unitListStatic;
}

public Object getWarehouseListStatic (){ // Remondino
    return warehouseListStatic;
}

    public int getWarehouse()
    {
        return useWarehouses;
    }
}

```

C.4 Unit.java

```
// Unit.java

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

import swarm.collections.ListImpl;

public class Unit extends SwarmObjectImpl{

    public int unitNumber, noOrder, totalUnitNumber, dummy, flag;
    public int useWarehouses;
    public ListImpl unitList, waitingList;
    Unit aUnit, thisUnit;
    Warehouse myWarehouse;
    Order firstOrder;
    public RuleMaster ruleMaster; // Remondino

    /**
     * First Constructor for Unit (with warehouses in simulation) ; Remondino
     */
    public Unit (Zone aZone, int un, int totNum, ListImpl ul, RuleMaster aRM, int wh,
    Warehouse aW) {

        // Call the constructor for the unit's parent class.
        super(aZone);

        ruleMaster = aRM; // RuleMaster address ; Remondino
        useWarehouses = wh; // if 0, warehouses are not present; if 1 warehouses are
        present ; Remondino
        myWarehouse = aW; // warehouse address ; Remondino

        // the production waiting list of the unit
        waitingList = new ListImpl (getZone());

        // Record the unit's id number.
        unitNumber=un;
        unitList=ul;
        totalUnitNumber=totNum; // the number of the units present in simulation ;
        Remondino

        // Announce the unit's presence to the console.
        System.out.println("Unit number " + unitNumber + " has been created.");
    }

    /**
     * Second Constructor for Unit (no warehouses in simulation) ; Remondino
     */

    public Unit (Zone aZone, int un, int totNum, ListImpl ul, RuleMaster aRM, int wh) {

        // Call the constructor for the unit's parent class.
        super(aZone);

        ruleMaster = aRM; // RuleMaster address ; Remondino
        useWarehouses = wh; // if 0, warehouses are not present; if 1 warehouses are
        present ; Remondino

        // the production waiting list of the unit
        waitingList = new ListImpl (getZone());

        // Record the unit's id number.
        unitNumber=un;
        unitList=ul;
        totalUnitNumber=totNum; // the number of the units present in simulation ;
        Remondino

        // Announce the unit's presence to the console.
        System.out.println("Unit number " + unitNumber + " has been created."); }
}
```

```

public void unitStep1 () {

    //System.out.println("I'm the unit number " + unitNumber + ".");

    /**
     * the order to be executed today (if any)
     */

    flag=0;    // Remondino
    noOrder=0;
    if(waitingList.getCount() != 0)
    {
        firstOrder=(Order)waitingList.removeFirst();

        if (useWarehouses==1) // start of Remondino's part: if warehouses are
present, execute
        {
            dummy=0;
            // unit asks rulemaster what to do
            dummy=ruleMaster.checkWarehouse(unitNumber, totalUnitNumber, myWarehouse);
        }
    }
    // end of Remondino's part
    else
    {
        noOrder=1;

        if (useWarehouses==1)
        {
            // if unit is free, asks rulemaster what to do
            ruleMaster.checkNumber(unitNumber, totalUnitNumber, myWarehouse);
        }
    }
}

public void unitStep2 () {

    int i, operatingUnit;

    if (noOrder != 1)
    {
        /**
         * sending the order to the successive production unit
         */

        if (dummy==1)
        {
            firstOrder.setDoneStep();
            flag=1;
            if (firstOrder.getNextStep()==0)
            {
                firstOrder.drop();
                System.out.println("Order #"+firstOrder.orderNumber+" is complete");
            }
        }
        else
        {
            operatingUnit=0;
            for (i=1;i<=unitList.getCount() && operatingUnit==0;i++)
            {
                aUnit=(Unit) unitList.removeFirst();
                unitList.addLast(aUnit);

                if
                (firstOrder.getNextStep()==aUnit.getProductionPhase())
                {operatingUnit=1;
                aUnit.setWaitingList(firstOrder);
                }
            }
        }
    }
}

```

```

        if (flag==1 && waitingList.getCount() != 0)
        {
            firstOrder=(Order)waitingList.removeFirst();
            flag=0;
            operatingUnit=0;
            for (i=1;i<=unitList.getCount() && operatingUnit==0;i++)
            {
                aUnit=(Unit) unitList.removeFirst();
                unitList.addLast(aUnit);

                if
                (firstOrder.getNextStep()==aUnit.getProductionPhase())
                {operatingUnit=1;
                 thisUnit=aUnit;
                 }
            }

            thisUnit.warehouseWork();
        }

        if (flag==0)
        {firstOrder.setDoneStep(); // Remondino
        if(firstOrder.getNextStep()==0)

            {
                firstOrder.drop();
                System.out.println("Order #"+firstOrder.orderNumber+" is complete");
            }

        else
        {
            operatingUnit=0;
            for (i=1;i<=unitList.getCount() && operatingUnit==0;i++)
            {
                aUnit=(Unit) unitList.removeFirst();
                unitList.addLast(aUnit);

                if (firstOrder.getNextStep()==aUnit.getProductionPhase())
                {operatingUnit=1;
                 aUnit.setWaitingList(firstOrder);}
            }
        }
    }
}

/**
 * Return the production phase of the unit
 */
public int getProductionPhase()
{
    return unitNumber; // at present unitNumber and production phase are
                       // the same thing; this is a temporary situation
}

/**
 * Putting an order in the production waitingList of the unit
 */
public void setWaitingList(Order anOrder)
{
    waitingList.addLast(anOrder);
    System.out.println("The unit " + unitNumber +
                       " has received the order # "
                       + anOrder.getOrderNumber());
}

/**
 * Returning the waiting list length
 */
public int getWaitingListLength()
{

```

```

    return waitingList.getCount();
}

// using warehouses ; Remondino
public void warehouseWork()
{
    int i, operatingUnit;
    dummy=0;
    if (useWarehouses==1)
    {
        dummy=0;
        dummy=ruleMaster.checkWarehouse(unitNumber, totalUnitNumber, myWarehouse);
        ruleMaster.checkWarehouse(unitNumber, totalUnitNumber, myWarehouse);
    }

    if (dummy==1)
    {
        firstOrder.setDoneStep();
        flag=1;
        if (firstOrder.getNextStep()==0)
        {
            firstOrder.drop();
            System.out.println("Order #"+firstOrder.orderNumber+" is complete");
        }
        else
        {
            operatingUnit=0;
            for (i=1;i<=unitList.getCount() && operatingUnit==0;i++)
            {
                aUnit=(Unit) unitList.removeFirst();
                unitList.addLast(aUnit);

                if
                (firstOrder.getNextStep()==aUnit.getProductionPhase())
                {operatingUnit=1;
                 aUnit.setWaitingList(firstOrder);
                }
            }
        }
    }
    else
    {
        flag=0;
        return;
    }
    if (useWarehouses==1)
    {
        if (waitingList.getCount() != 0)
        {
            firstOrder=(Order)waitingList.removeFirst();

            flag=0;
            operatingUnit=0;
            for (i=1;i<=unitList.getCount() && operatingUnit==0;i++)
            {
                aUnit=(Unit) unitList.removeFirst();
                unitList.addLast(aUnit);

                if
                (firstOrder.getNextStep()==aUnit.getProductionPhase())
                {operatingUnit=1;
                 thisUnit=aUnit;
                }
            }

            thisUnit.warehouseWork();
        }
    }
}

```


C.5 RuleMaster.java

```
// RuleMaster.java
// Class created by Marco Remondino

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

import swarm.collections.ListImpl;

/**
 * Constructor for RuleMaster
 */
public class RuleMaster {
    public RuleMaster (int max)
    {
        maxInWarehouses=max;
        if (maxInWarehouses==0)
        {maxInWarehouses=9999;
        }
    }
    public int unitNumber, warehouseNumber, counter, i, totalUnitNumber, maxInWarehouses;

    public Warehouse myWarehouse;

    // if a Unit is free, asks RuleMaster what do do

    public void checkNumber(int un, int totNum, Warehouse aW)
    {
        unitNumber=un;
        totalUnitNumber=totNum;
        myWarehouse=aW;

        System.out.println("* Unit #"+unitNumber+" is free");

        System.out.println("* Warehouse #" + myWarehouse.getWarehouseNumber() + " has
"+myWarehouse.getCounterValue()+" pieces");

        if (myWarehouse.getCounterValue()<maxInWarehouses)
        {System.out.println("* Unit #"+unitNumber+" is putting one piece in
its warehouse");
        myWarehouse.addCounterValue();
        }
        else System.out.println("* Warehouse #"+unitNumber+" is already
full.");
    }

    // if a Unit has an order in waiting list, asks RuleMaster what to do

    public int checkWarehouse(int un, int totNum, Warehouse aW)

    {unitNumber=un;
    totalUnitNumber=totNum;
    myWarehouse=aW;

    if (myWarehouse.getCounterValue()>0)
    {
        System.out.println("* Unit #"+unitNumber+ " has pieces in
warehouse: using stored piece.");
        myWarehouse.subCounterValue();
        System.out.println("** "+myWarehouse.getCounterValue()+"
pieces left in warehouse #" +unitNumber);

        return 1;}

    return 0;}
}
```

C.6 Warehouse

```
// Warehouse.java
// Class created by Marco Remondino

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

import swarm.collections.ListImpl;

public class Warehouse extends SwarmObjectImpl{

    public int warehouseNumber, counter;
    public VEFrameModelSwarm vEFrameModelSwarm;

    /**
     * Constructor for Warehouse
     */
    public Warehouse (Zone aZone, int sn) {

        // Call the constructor for the warehouse's parent class.

        super(aZone);

        // Record the warehouse's id number.
        warehouseNumber=sn;
        counter = 0;
        vEFrameModelSwarm = new VEFrameModelSwarm (getZone ());

        // Announce the warehouse's presence to the console.
        if (vEFrameModelSwarm.getWarehouse()==1) System.out.println("Warehouse number " +
warehouseNumber + " has been created.");
    }

    public int getCounterValue()
    {
        return counter; // Returns the counter value.
    }

    public int getWarehouseNumber()
    {
        return warehouseNumber; // Returns the warehouse number.
    }

    public void addCounterValue()
    {
        counter=counter+1;
        return; // Add one to the counter value.
    }

    public void subCounterValue()
    {
        counter=counter-1;
        return; // Subtracts one to the counter value.
    }

}
```

C.7 OrderGenerator

```
// OrderGenerator.java

import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;
import swarm.collections.ListImpl;

public class OrderGenerator extends SwarmObjectImpl{

    int unitNumber, maxStepNumber, orderCount;

    public Order anOrder;
    public ListImpl unitList;
    public int[] orderRecipe;

    /**
     * Constructor for OrderGenerator
     */
    public OrderGenerator (Zone aZone, int un, int msn, ListImpl ul)
    {
        // Call the constructor for the unit's parent class.
        super(aZone);

        orderCount=0;
        unitNumber=un;
        maxStepNumber=msn;
        unitList=ul;
        orderRecipe= new int[maxStepNumber];
    }
    public void createRandomOrderWithNSteps()
    {
        int i, randomStepNumber, operatingUnit;

        Unit aUnit;

        orderCount++;
        randomStepNumber=Globals.env.uniformIntRand.getIntegerWithMin$withMax
            (1, maxStepNumber);
        for (i=1;i<=randomStepNumber;i++)
        {
            orderRecipe[i-1]=
                Globals.env.uniformIntRand.getIntegerWithMin$withMax
                (1, unitNumber);
        }
        anOrder = new Order(getZone(), orderCount, randomStepNumber,
            orderRecipe);

        /**
         * sending the order to the 1st production unit
         */
        operatingUnit=0;
        for (i=1;i<=unitList.getCount() && operatingUnit==0;i++)
        {
            aUnit=(Unit) unitList.removeFirst();
            unitList.addLast(aUnit);

            if (anOrder.getNextStep()==aUnit.getProductionPhase())
            {
                operatingUnit=1;
                aUnit.setWaitingList(anOrder);
            }
        }

        // anOrder.drop();
    }
}
```

C.8 Order.java

```
// Order.java

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

public class Order extends SwarmObjectImpl{

    public int stepNumber, orderNumber;
    public int[] orderRecipe, orderState;

    /**
     * Constructor for Order
     */
    public Order (Zone aZone, int n, int sn, int [] r) {

        // Call the constructor for the unit's parent class.
        super(aZone);

        orderNumber=n;
        stepNumber=sn;
        orderRecipe= new int[stepNumber];
        orderState = new int[stepNumber];

        int i;

        for (i=0;i<stepNumber;i++)orderRecipe[i]=r[i];

        // Announce the order's presence to the console.
        System.out.println("New order (#" + orderNumber + ")");

        // setting the state vector (0=empty; 1=done)
        System.out.print("orderState vector ");
        for (i=1;i<=stepNumber;i++)
        {
            orderState[i-1]=0;
            System.out.print(orderState[i-1] + " ");
        }
        System.out.println();

        // announcing the recipe vector
        System.out.print("orderRecipe vector ");
        for (i=1;i<=stepNumber;i++)
            System.out.print(orderRecipe[i-1] + " ");
        System.out.println();

    }

    /**
     * Production next step in Order
     */
    public int getNextStep () {

        int i, stepN;

        stepN=0;
        for (i=0;i< stepNumber && stepN==0;i++)
            if (orderState[i]==0)stepN=orderRecipe[i];

        return stepN;

    }

    /**
     * Done next step in Order
     */
    public void setDoneStep () {

        int i, done;
```

```

        done=0;

        for (i=0;i<= stepNumber && done==0;i++)
            if (orderState[i]==0)
                {orderState[i]=1;
                 done=1;
                }

//        return;
    }

/**
 * returning orderNumber of this order
 */
    public int getOrderNumber () {
        return orderNumber;
    }
}

```