UNIVERISTÁ DEGLI STUDI DI TORINO

FACOLTÁ DI ECONOMIA

CORSO DI LAUREA IN ECONOMIA E COMMERCIO

TESI DI LAUREA

LA PREVISIONE DEGLI ORDINI ATTRAVERSO LA SIMULAZIONE DI UNA CATENA DI FORNITURA: UN CASO AZIENDALE.

Relatore:
Prof. PIETRO TERNA

Correlatore:
Dr. SERGIO MARGARITA

Candidato:
MICHELE SONNESSA

Indice degli Argomenti

Introduzione

La rivoluzione tecnologica che ha caratterizzato gli anni ’90 ha contribuito a modificare diverse attività della vita aziendale e, soprattutto, ha creato nuove opportunità per le imprese. Opportunità in termini di nuovi mercati e di nuove attività di *business*. Ma la flessibilità degli strumenti informatici ha consentito soprattutto di aumentare la quantità di conoscenza condivisibile, attraverso un processo di formalizzazione e di codificazione strutturata della stessa. Il fenomeno muta lo scenario e le capacità di analisi della realtà aziendale, in un contesto che abbraccia il fenomeno della globalizzazione.

Le nuove tecnologie aprono di fatto nuovi orizzonti nella capacità di integrazione e di collaborazione tra le imprese. Non solo, esiste anche un interessante sviluppo della ricerca scientifica nell’ambito delle cosiddette scienze sociali. Le materie, cioè, che studiano i sistemi complessi in cui è presente un elemento difficilmente spiegabile da una relazione funzionale, poiché assolutamente non lineare, adattivo e spesso irrazionale: la mente umana. L’economia fa certamente parte di questa categoria e ciò è confermato dalla difficoltà della teoria classica di fornire modelli affidabili che sappiano prevedere con certezza i fenomeni della realtà. Ciò a causa della impossibilità di riprodurre il modello in laboratorio e verificarne le conclusioni. Non potendo avvalorarne le ipotesi attraverso la verifica empirica è ugualmente difficile riporre troppa fiducia nelle sue conclusioni.

Tutti i fenomeni incerti e la loro previsione rappresenta un elemento fondamentale da un punto di vista gestionale, sono fenomeni caratterizzati dall’interazione di elementi non perfettamente razionali e deterministici.

Il presente lavoro nasce dalla riflessione sulle possibilità di rispondere ad un’esigenza ricorrente nel mondo aziendale: la capacità di prevedere i fenomeni per rendere efficace la politica di gestione. In particolare, è necessario trovare lo strumento più idoneo per prevedere la stagione di vendita di un’azienda che opera nel settore tessile: la Boglietti S.p.A.

Lo stretto connubio tra l’uso delle tecnologie informatiche nell’ambito imprenditoriale, come strumento di coordinamento delle attività e di osservazione delle *performance* aziendali, e nell’ambito scientifico, attraverso la metodologia di simulazione dei modelli basati su agenti, ha suggerito l’ipotesi di costruire un modello di simulazione dell’azienda attraverso questa tecnica che sfrutti l’elemento comune: la capacità dei computer di elaborare e manipolare le informazioni.

Il successo nella previsione degli ordini di prodotti di un’azienda è riposto nella comprensione dei fenomeni interni all’azienda che influenzano la domanda ed soprattutto di quelli esterni. Per questo la simulazione dell’azienda deve tenere conto da un punto di vista concettuale dell’intero sistema di produzione a cui l’impresa appartiene. A tal proposito nel capitolo 1 sarà affrontata un’analisi strategica delle strutture organizzative e la loro collocazione all’interno delle

catene di fornitura, al fine di inquadrare il contesto ambientale entro il quale le aziende si collocano e si integrano.

L'analisi parte dalla descrizione delle tecniche statistiche di previsione dei dati, al fine di valutare la loro applicabilità al problema in esame. La riflessione suggerisce, però, che la sola applicazione statistica non è sufficiente a indagare fenomeni complessi.

Si può comprendere che la sola applicazione di metodi statistici sui dati aggregati che pervengono alla Boglietti non rappresenta una metodologia sufficientemente affidabile, a causa delle variazioni dovute ad interazioni di fenomeni non quantificabili: le tendenze moda, l'influenza soggettiva della rete di vendita. Da questo quadro descrittivo emergono gli elementi sufficienti a suggerire di integrare l'intera struttura della rete di vendita dell'azienda in un modello che sia in grado di interpretare le complessità dinamiche del sistema.

Lo strumento che meglio affronta la problematica della complessità è, come si è detto, il paradigma dei modelli ad agenti. Ma tale metodologia necessita di un'approfondita descrizione teorica, affrontata nel capitolo 3, e pratica, nei capitoli 4 e 5.

All'interno di tale modello diventa più plausibile collocare tecniche di previsione, che possono essere poste a diversi livelli di dettaglio. Una volta dimostrato come l'influenza del sistema degli agenti rappresentanti produca un micro-cosmo con caratteristiche differenti da caso a caso, chi disegna la struttura del modello può ragionevolmente ritenere più interessante applicare le tecniche di previsione a livello del singolo agente, per poi aggregarne i risultati.

Come si è ampiamente detto, la ricerca nel campo delle scienze sociali si orienta verso l'uso di modelli ad agenti, in quanto possiedono la capacità di rispecchiare le strutture ed i meccanismi del mondo reale. Inoltre, tale strumento consente di realizzare modelli che descrivono, in modo uniforme ed indipendente dalla natura del problema, ciò che la mente è in grado di percepire dalla realtà.

Essi sono in grado di descrivere la dinamica dei sistemi di carattere teorico con la stessa metodologia con cui possono descrivere realtà concrete. In tale contesto nasce l'idea di sfruttare questo strumento per dare vita ad un modello di azienda virtuale, che come tale consente di essere sottoposta ad esperimenti di simulazione. È un modello molto flessibile e può essere usato come infrastruttura entro la quale collocare il software relativo agli strumenti di indagine tradizionale. Tutto l'apparato informatico utile alla gestione operativa (ordini e produzione) ed all'analisi dei dati (*DDS, decision support system*) si integra nell'infrastruttura fornendo al modello le informazioni di cui necessita. Ciò evita il lavoro di trascrizione ed importazione dei dati. Gli esperimenti e le simulazioni dell'effetto di ipotesi di variazioni nell'organizzazione o nell'utilizzo di nuovi strumenti per le previsioni si effettuano dentro la stessa infrastruttura, usando come supporto un ambiente che rispecchia più o meno fedelmente la dinamica del sistema. Il modello ad agenti aggiunge a questi elementi la capacità di interazione.

L’applicazione pratica presuppone una conoscenza informatica che esula dal patrimonio di conoscenza della teoria economica, ma che è indispensabile. A tale proposito è necessario considerare la scelta più opportuna in termini di *software* per costruire i modelli di simulazione. Nel capitolo 4 si confrontano due *software* scientifici. Il primo, *Starlogo*, è un ambiente progettato appositamente per costruire le simulazioni ad agenti, il secondo, *Mathematica*, non ha questa caratteristica ma, essendo un potentissimo e diffuso programma di calcolo, presenta interessanti caratteristiche in questo ambito di studio.

Il capitolo 5 presenta il *software* scelto per la simulazione aziendale, poiché ritenuto il più idoneo a questo scopo: *Swarm*. Oltre ad essere un interessante oggetto informatico, *Swarm* è un progetto molto diffuso nella comunità scientifica ed esiste in letteratura un interessante modello di simulazione aziendale che costituisce il punto di partenza del presente lavoro. Il MAIS (*multi agent information system*) è un modello molto articolato e ricco di utili riflessioni, descritte approfonditamente nel capitolo 6.

Il capitoli 7, 8 e 9 costituiscono rispettivamente la descrizione teorica, il modello applicativo e la riflessione sui risultati e sulle conclusioni emerse durante la realizzazione del progetto di simulazione dell’azienda e di calcolo delle previsioni.

Naturalmente la realizzazione di un modello capace di contenere tutte le componenti fin qui presentate è un progetto ambizioso. Spesso la realizzazione di progetti ad ampio spettro richiedono la disposizione di tutte le parti del sistema a collaborare alla sua realizzazione. Il paragone con l’*e-business* è illuminante. Portare l’azienda verso un nuovo modo di scambiare informazioni è un progetto strategico che richiede l’intervento del management in prima persona. Allo stesso modo orientare tutta la tecnologia di memorizzazione dei dati verso un modello a componenti “aperti” ed intercomunicanti richiede una rivoluzione nelle attività aziendali e nello scambio delle informazioni, nonché nella stessa organizzazione.

Tutto ciò ha portato a costruire un modello di azienda virtuale che comporta la descrizione degli obiettivi, delle attività che la caratterizzano, delle interdipendenze e del risultato atteso. Questo deve essere il frutto di un opportuno bilanciamento degli obiettivi intermedi, compatibile con le attese dei manager della catena di fornitura. Talvolta si può perseguire l’abbattimento dei costi, talvolta la soddisfazione dei clienti e la qualità. In ognuna di queste configurazioni è necessario che il modello si possa adattare al cambiamento dell’ambiente. Ciò significa che deve essere rispettata la modularità del progetto.

Capitolo 1 Le catene di fornitura: analisi strategica

1.1 La gestione delle catene di fornitura

La gestione delle catene di fornitura è un'attività manageriale critica, poiché riguarda il coordinamento e le decisioni organizzative non soltanto intra-aziendali, ma soprattutto inter-aziendali. Il punto critico è rappresentato oltre che dalla difficoltà nel controllare l'efficienza delle unità operative, anche nella necessità di realizzare una struttura che garantisca un corretto e tempestivo flusso di informazioni tra le unità stesse.

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 letteratura suggerisce (Chandler 1990, Arora 1997, 1998, Farrel 1998) che la struttura industriale tende ad evolvere, nell'arco del tempo, dal livello dell'integrazione a quello della specializzazione. Ciò significa il passaggio da una struttura produttiva in grado di realizzare internamente la maggior parte delle attività di produzione ad una in cui la produzione si concentra nelle attività cruciali, dette di *core business*, rivolgendosi all'esterno per le attività considerate collaterali. Esso si verifica quando più di una attività è richiesta per produrre un bene. Le aziende possono specializzarsi in una di queste attività o integrarle tutte nella propria struttura.

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. La risposta obbligata è, pertanto, quella di costruire le

strutture interne per produrre da sé il bene o il servizio. Quando cominciano ad emergere gli standard e i progressi tecnologici abbassano i costi di produzione, i vantaggi dell'integrazione vengono progressivamente meno. 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.

Inoltre, quando un'impresa ha differenti livelli di efficienza nelle attività che svolge, può decidere di concentrarsi in quella in cui è più efficiente. Questo effetto, detto effetto della divisione del lavoro (Arora 1997), è giustificato da una curva di costo marginale diversa per le due attività. Il produttore che svolge internamente tali attività non può sfruttare il pieno vantaggio che deriva dall'attività in cui è più efficiente, perché è disturbato dalla curva dei costi marginali dell'attività meno profittevole. Si dimostra che attraverso la specializzazione l'azienda può ottenere una curva dei costi relativa a quest'ultimo processo piatta.

L'unico elemento che costituisce un freno alla specializzazione è rappresentato 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.

Quelle che non molti anni fa erano grandi strutture gerarchiche, oggi sono tante piccole unità organizzative indipendenti o piccole aziende, alle quali non si può pretendere di imporre regole di gestione dall'esterno. Ciò ha prodotto una situazione in cui processi aziendali non coordinati possono condurre a risultati inaccettabili, pur disponendo di *business unit* molto efficienti. Per questa ragione la gestione organizzativa delle catene di fornitura deve espandere la propria sfera di competenza alle relazioni inter-aziedali. Lee e Billington (1993) hanno mostrato che la gestione delle catene di fornitura si è concentrata in passato sull'analisi delle necessità informative dei nodi della catena nei quali vengono prese decisioni critiche.

Da questa situazione, emerge inevitabilmente che la linea lungo la quale viene creato il valore aggiunto richieda un'analisi profonda. Secondo Sadeh, Smith e Swaminathan (1998) tale studio può essere condotto su tre distinti livelli:

- ***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.
- ***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.
- ***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.

In questo lavoro, verranno approfonditi i primi due punti, poiché richiedono un supporto teorico, a differenza dell'ultimo punto, che rappresenta un'attività tipicamente operativa, localizzata nelle sole unità che si occupano della distribuzione.

Al fine di fornire uno strumento concettuale per supportare l'attività dei manager di 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. Questo è riscontrabile, ad esempio, nel settore dell'abbigliamento, nel quale la domanda è dipendente da fattori difficili da prevedere: la moda, l'economia, i materiali e la disponibilità di dati è limitata, a causa della breve durata delle stagioni di vendita. La previsione corretta e tempestiva dei prodotti richiesti dal mercato è un elemento che può diventare un vantaggio strategico di una catena di fornitura bene organizzata. I prodotti devono essere messi in produzione con sufficiente anticipo rispetto al momento in cui il cliente manifesterà l'intenzione di acquistare. Gli anelli della catena a diretto contatto con il cliente devono comunicare tempestivamente gli ordinativi ed inoltre, devono coordinare le proprie strategie commerciali con le strategie produttive di chi sta a monte della catena stessa. 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.

1.2 La struttura delle catene di fornitura

Una catena di fornitura si descrive come una rete attraverso la quale operano diversi agenti eterogenei, la cui interazione dà origine ad un flusso bi-direzionale di materie prime, informazioni e pagamenti, un esempio è rappresentato in fig. 1. 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. Va spostato il punto di vista da un sistema orientato 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.

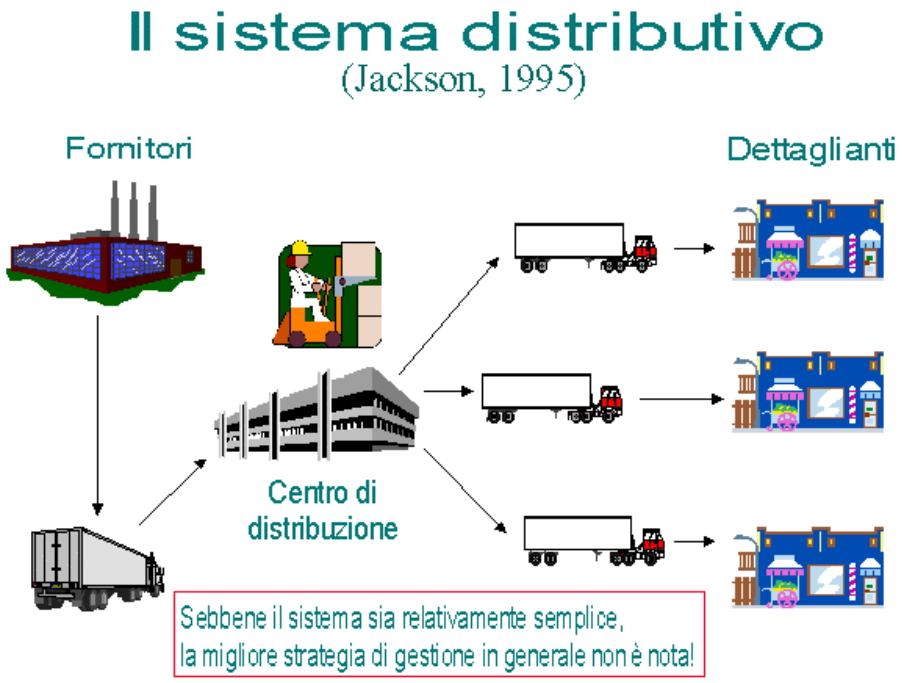

fig. 1 Schema di una catena di fornitura.

Un'interessante definizione è data da Ganeshan, Harrison (1995):

> A *supply chain* is a network of facilities and distribution options that performs the functions of procurement of materials, transformation of these materials into intermediate and finished products, and the distribution of these finished products to customers. Supply chains exist in both service and manufacturing organizations, although the complexity of the chain may vary greatly from industry to industry and firm to firm.

La gestione delle catene di fornitura è un'attività di cui si deve occupare il management di una società. Nel passato, la direzione si poneva come principale obiettivo quello di rendere efficiente ciascun settore dell'azienda. L'incremento della competizione per il miglioramento della qualità, del servizio al consumatore e dell'efficienza operativa, richiedono oggi maggiore

attenzione alle relazioni critiche tra le diverse unità che operano all'interno e all'esterno dell'azienda. L'efficacia di ogni entità del sistema dipende strettamente dalla propria tempestività e dalla propria capacità di coordinazione con le altre entità.

Tradizionalmente, le organizzazioni che si occupano di marketing, distribuzione, produzione e pianificazione operano in modo indipendente all'interno della rete produttiva. Tali organizzazioni hanno i propri obiettivi e spesso sono in conflitto. Gli obiettivi del marketing di soddisfazione del cliente e di massimizzazione delle vendite non sono gli stessi di quelli della produzione.

Spesso non esiste una singola ed integrata strategia per l'intera organizzazione, ma esistono molti *business plan* diversificati. La gestione di una catena di fornitura è una strategia attraverso la quale può essere raggiunto tale obiettivo di integrazione.

Cooper ed Ellram (1993) comparano la gestione delle catene di fornitura ad una ben bilanciata squadra di staffetta. Un tale organico è tanto più competitivo quanto più è ben posizionato per lo scambio. Il legame tra i corridori che si passano direttamente la staffetta è il più stretto, ma tutta l'intera squadra deve cercare di coordinarsi per vincere.

Il *management* che gestisce un'intera catena di fornitura deve prendere due tipologie di decisioni: strategiche ed operative. Le prime devono essere basate su un orizzonte di lungo termine e sono strettamente correlate con le strategie di *corporate*. Costituiscono la prospettiva progettuale della catena. Le seconde sono relative alle particolari attività che sono svolte quotidianamente nella realtà produttiva. L'attenzione in questo caso deve essere riposta alla gestione efficiente ed efficace del flusso di produzione all'interno della prospettiva strategica prima elaborata.

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

- 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.
- 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.

- Le **decisioni di stoccaggio**. Un elemento molto importante è la collocazione ed il livello medio di utilizzo dei magazzini. 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.
- 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.

L'attività di pianificazione strategica ed operativa dell'intera catena produttiva ha il delicato compito di gestire una realtà complessa e sulla quale soltanto negli anni '90 si è indirizzata l'attenzione degli strateghi aziendali. Il primo compito è quello di risolvere le numerose inefficienze che si presentano nella realtà mondiale delle catene produttive. Ciò emerge dall'analisi della situazione reale in Billington , Lee (1992):

- *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.
- *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.
- *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.
- *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.
- *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.

- *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.
- *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.

L'efficacia di una catena di fornitura dipende fortemente da due fattori opposti: l'integrazione e il decentramento. Per fronteggiare la concorrenza di un mercato globale, non si può sostenere una struttura produttiva integrata ed autonoma. Le catene di fornitura si caratterizzano sempre più per un'atomizzazione spinta delle unità produttive. Contemporaneamente il successo dell'intero sistema produttivo, risiede nella sua forte integrazione orizzontale.

Una catena specializzata e fortemente coordinata richiede un sistema di informazioni distribuite. L'informazione è decentralizzata è poco omogenea, poiché proveniente da sorgenti diverse. Nessun anello della catena ha sufficienti informazioni per poter controllare e gestire l'intero processo. L'unico modello che può descrivere ed analizzare una realtà così complessa è certamente un modello distribuito.

1.3 Le prospettive

Nell'era dell'informazione distribuita in cui il cambiamento è moto rapido, la conoscenza è diventata il fattore produttivo più importante, il mercato e la comunicazione sono globalizzati e la crescente differenziazione della domanda impongono nuove strutture aziendali, in cui siano presenti le seguenti caratteristiche (Merry 1999):

- Le **reti**: la struttura delle organizzazioni deve assomigliare sempre di più ad una rete piuttosto che ad una gerarchia centralizzata. La struttura non è una piramide gerarchica con livelli di autorità. È una ricca ragnatela di relazioni e comunicazioni che fluiscono in tutte le direzioni, tra agenti interni ed esterni all'impresa.
- I **livelli "holarchici"**: i livelli esistono come holarchie[1] (si contengono l'un l'altro) e non come gerarchie, in cui una domina l'altra. La struttura è piatta con pochi livelli. Il controllo e la pianificazione avvengono isolatamente in ciascun livello.

[1] Per una trattazione approfondita del concetto di holarchia si veda http://inn.ingrm.it/compsys/holar1.htm o http://capita.wustl.edu/ME567_Informatics/concepts/holarchy.html.

- La predisposizione al **cambiamento**: le strutture non devono essere solidificate, ma cambiare in funzione delle necessità imposte dall'ambiente. Devono essere smantellate le strutture burocratiche, in modo da consentire la delega delle decisioni e delle azioni.
- Le **reti esterne**: le organizzazioni devono essere costruite come parti di una entità più grande, costituita da accordi, *partnership*, contatti, scambi, *joint venture* tra molte aziende.
- Il **rinnovamento**: l'impresa deve possedere meccanismi interni che garantiscano il proprio rinnovamento, eliminando le barriere che ostruiscono l'auto-organizzazione ed il cambiamento. Deve esserci anche una struttura che incoraggi le persone ad intraprendere nuove strade.
- Le **decisioni parallele**: le decisioni devono essere assunte in parallelo in ogni situazione e ad ogni livello dell'organizzazione. Spesso infatti le informazioni per decidere sono possedute dalla struttura più vicina al problema da affrontare ed il parallelismo garantisce una maggiore rapidità.
- L'**autonomia dei sub-sistemi**: ogni sotto-sistema assume le decisioni strategiche nell'ambito dei principi direttivi stabiliti dalla direzione. Vi deve essere maggiore delega di autorità e la funzione direttiva deve concentrarsi sull'analisi aggregata degli obiettivi aziendali. L'autonomia e la responsabilità devono procedere di pari passo.
- La **specializzazione**: le energie devono essere concentrate nei settori produttivi nei quali l'azienda ha speciali competenze. Le altre attività che sono necessarie al funzionamento dell'attività d'impresa devono essere ottenute in *outsourcing*.
- La **velocità**: il tasso di cambiamento della domanda richiede un'organizzazione rapida nell'adattarsi a tali esigenze. Non devono essere sbarrate le porte alle nuove opportunità. L'azienda avrà bisogno di sviluppare strutture e meccanismi che consentano ad essa di essere riprodotta nella realtà virtuale.
- La **scarsa correlazione**: i progetti e le altre strutture all'azienda devono essere collegate in modo lasco. Se ciò non accade esiste il rischio dell'effetto domino. Se una di esse crolla porta con sé tutte le altre. La sfida consiste nel trovare il giusto grado di interdipendenza.
- I **confini sfumati**: ciò indica che i confini dell'organizzazione non devono essere netti, chiari, definiti, ma impermeabili all'interno ed all'esterno dell'organizzazione. Essi devono cambiare in funzione dei cambiamenti che avvengono nelle relazioni tra le persone, nel mercato.
- L'**IT efficiente**: la qualità dell'*information technology* deve essere di alto livello. Esso deve essere capace di fornire i servizi per dialogare, per prendere decisioni, per gestire i progetti, per le conferenze e la formazione.

- La **fiducia come principio strutturale**: Lo sviluppo, la costruzione ed il mantenimento della fiducia tra i sotto-gruppi dell'organizzazione e al di fuori di essa deve essere un principio guida ed un valore che contribuisce alla sinergia, alla creatività ed al cambiamento.
- La **soddisfazione**: non è possibile realizzare tutti gli obiettivi sopra esposti, dunque è necessario saper scegliere quali consentono di raggiungere la migliore struttura per rendere flessibile ed innovativa l'impresa.

Stalk, Evans e Shulman (1992) suggeriscono che la competizione è "un'onda in movimento" nella quale il successo dipende dalla capacità di anticipare i trend di mercato e di rispondere rapidamente ai cambiamenti nei bisogni dei consumatori. In tale scenario la strategia aziendale è guidata da quattro principi su cui si fonda la competizione:

Gli obiettivi delle strategie d'impresa non sono i prodotti o i mercati ma i processi di produzione.

1. Il successo dipende dalla capacità di trasformare i processi fondamentali dell'azienda in capacità strategiche che forniscono un valore consistentemente superiore al consumatore.
2. Le imprese creano queste capacità facendo investimenti strategici nella creazione di infrastrutture capaci di collegare tra loro i tradizionali unità e funzioni di *business* strategico.
3. Il responsabile di tale infrastruttura diventa il nodo chiave di supporto a questa strategia.

L'infrastruttura che consente di collegare i processi aziendali è la strategia vincente in un ambiente produttivo dinamico e mutevole. Al fine di adattarsi e gestire i cambiamenti della realtà in cui l'azienda opera è necessario comprendere la dinamica dei processi produttivi sia da un punto di vista interno che da uno esterno all'impresa. I modelli di integrazione aziendale sono una soluzione per comprendere tali dinamiche e per disegnare le strategie migliori per affrontare questo tipo di competizione.

Le aziende possono essere integrate in tre diverse tipologie di strutture:

1. L'integrazione tra aziende, anche detto *vertical partnering*. Esso aumenta la condivisione di informazioni relative al prodotto ed ai suoi differenti stadi di produzione.
2. L'integrazione delle funzioni aziendali si occupa di migliorare la comunicazione tra processi che danno vita al prodotto. È detta *horizontal partnering.*

Integrazione della catena del valore, o *business integration*. Fornisce una struttura più flessibile ai cambiamenti e più responsabilizzata, poiché permette di configurare una catena produttiva più dinamica.

Ma prima di progettare l'integrazione dell'azienda è necessario comprendere le attività che l'organizzazione svolge al suo interno.

All'interno di un'industria manifatturiera esistono molti buoni motivi per realizzare un'infrastruttura comune finalizzata all'integrazione aziendale. Innanzi tutto, per la capacità di

condividere procedure ed informazioni tra le unità funzionali dell'impresa, indipendentemente dalla collocazione gerarchica di tali unità. In secondo luogo, le imprese possono ridurre i costi infrastrutturali. Invece di realizzare un grande unico stabilimento produttivo, ogni singolo settore di produzione può concentrare le risorse sui fattori ad essa specifici e sfruttare la i vantaggi di un grande e diffuso orizzonte di fornitori. Il terzo vantaggio consiste nel potenziamento della capacità delle imprese di collaborare al progetto dei processi produttivi relativi a prodotti complessi.

1.4 Il flusso delle informazioni

La tecnologia informatica assume un ruolo fondamentale in un sistema di imprese che devono collaborare per dare vita ad un ciclo di produzione distribuito. La struttura tecnologica che assicura il flusso delle informazioni caratterizza il risultato di un catena di fornitura. Inoltre, l'*information technology* permette di ridurre i costi di transazione e di suddividere i compiti degli agenti del mercato a livelli finora impensabili.

Per conoscere quale sia la struttura delle relazioni iter-aziendali è utile distinguere le *business unit* in due macro categorie: quelle dotate di un sistema informativo tradizionale e quelle innovative, dotate dei cosiddetti *digital interactive services* (DIS).

I sistemi informativi di tipo DIS sono il frutto di un'evoluzione che parte da sistemi proprietari e monolitici verso tecnologie a componenti aperti e compatibili tra loro, come la tecnologia Internet-Intranet.

La scelta di un azienda relativa a quale modello di *information technology* affidarsi non è banale. Normalmente questa scelta rispecchia il grado di integrazione dell'azienda. 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 portabile. 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.

Un problema da non sottovalutare è quello per cui molte imprese si trovano nella situazione di doversi trasformare per affrontare la sfida della specializzazione produttiva, dovendo considerare la necessità di realizzare una migrazione del proprio sistema informativo esistente verso le nuove tecnologie. Ciò avviene molto frequentemente poiché, l'evoluzione del mercato impone alle aziende una progressiva specializzazione nella produzione, con la necessità integrarsi in una fitta rete di scambi commerciali.

I servizi denominati DIS vengono realizzati attraverso lo scambio elettronico di informazioni, denominato EDI (*electronic data interchange*). Si può etichettare 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 le informazioni devono essere strutturate secondo formati predefiniti e universalmente riconosciuti

I fornitori che ricevono ordini *just in time* raggiungono senza dubbio risultati migliori, in termini di soddisfazione dei clienti e di aumento dell'efficienza produttiva. Questo è l'effetto dell'integrazione elettronica. 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 collaborano da punti geografici distinti 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 www si presta ad essere un ottimo strumento per la crescita delle catene produttive distribuite.

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. 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 un più leggero intervento di modifica. 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 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, si orientano all'uso della rete come *core business*, propongono i prodotti e i 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.

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.

1.5 I modelli di analisi delle catene di fornitura

La ricerca economico-scientifica ha da tempo affrontato il problema del *suppy chain management* al fine di fornire un insieme di strumenti che abbiano una giustificazione teorica e allo stesso tempo una validità pratica per l'attività di coordinamento e gestione delle strategie tra gruppi di aziende.

In Ganeshan e Harrison (1995) si suggerisce la divisione di questa materia in tre aree di indagine: i modelli descrittivi (*network design*), i metodi di "*rought-cut*" e i metodi basati sulla simulazione.

I primi costituiscono uno strumento per assistere i *manager* nelle decisioni strategiche. Essi si concentrano sull'analisi delle quattro tipologie di decisioni strategiche prima descritte. In questo ambito esiste una vasta ed eterogenea letteratura di studi e modelli. Tra questi si ricordano:

- l'analisi di Geoffrion e Powers (1993) che traccia il percorso evolutivo delle strategie distributive;
- in Breitman e Lucas (1987) si mostra un complesso modello che riproduce l'infrastruttura del sistema di produzione-distribuzione, usato per decidere quali prodotti offrire sul mercato, dove e quando produrli, in quali mercati e quali risorse impiegare. Tale studio è stato applicato con successo alla General Motors;

- in Cohen e Lee (1988) è sviluppato un modello concettuale di analisi delle strategie di produzione. Esso consiste di sub-modelli stocastici che considerano i flussi dei prodotti lungo l'intera catena di produzione. Tali sub-modelli sono ottimizzati e collegati tra loro attraverso metodi euristici;
- in Arntzen, Brown, Harrison, Trafton (1995) si trova il più approfondito modello deterministico per la gestione delle catene di fornitura. La funzione obiettivo minimizza una combinazione di elementi di costo e di tempo. Per la prima volta è stata perso in considerazione la componente rappresentata dalle imposte e dal loro recupero quando i prodotti si muovono attraverso le frontiere. L'applicazione di questo modello alla Digital Equipment Corporation ha prodotto spettacolari risultati, consentendo un risparmio di circa 100 milioni di dollari.

Sebbene in letteratura si dia molto credito alle potenzialità i questa metodologia, essa non è esente da limiti. I problemi di analisi sono affrontati ad un livello a "larga scala", sono spesso difficili da risolvere. Spesso sono troppo deterministici e statici. Spesso usano elementi stocastici che non trovano un corretto raffronto nella realtà. In sintesi, essi non sembrano essere così rappresentativi della vera natura dei flussi di beni che avvengono nelle catene di fornitura. Inoltre, non tengono presente il flusso delle informazioni che, come si è visto, è altrettanto importante.

I secondi forniscono le linee guida per le decisioni di carattere operativo. Essi considerano la realtà di ogni unità operativa relativa ad un singolo stadio produttivo ed applicano ad essa le caratteristiche comuni all'intera rete di fornitura.

I metodi di *rough-cut* sono anche detti modelli di controllo multi-livello delle scorte. La loro gestione nelle catene produttive ha l'obiettivo di ridurre il livello medio delle giacenze di magazzino, migliorando contemporaneamente il servizio al consumatore. Tali studi presentano, però, molti limiti. Spesso ignorano la produzione, concentrando l'analisi sulle code dei soli prodotti finiti. La descrizione è strutturata ad albero: i beni giungono al distributore da un unico punto e vengono distribuiti a più nodi del livello sottostante. Ciò non è plausibile se raffrontato alla realtà che si propone di descrivere, nella quale l'azienda si affida a diversi fornitori. Nella realtà quotidiana devono essere analizzate le relazioni tra i magazzini, il trasporto ed i consumatori, e ciò non è considerato in questa metodologia.

I metodi simulativi spostano l'analisi della catena ad un punto di vista aggregato e prendono in considerazione gli aspetti strategici ed operativi contemporaneamente. La rappresentazione attraverso un modello esplicativo di un sistema produttivo così complesso come la catena di fornitura richiede l'utilizzazione di conoscenze attinenti, come si è visto, a diverse discipline e soprattutto all'approfondimento della ricerca scientifico-sociale fondata sui modelli multi-agente. Questo metodo di indagine consente di descrivere i modelli comportamentali di ciascun elemento del sistema in modo molto plausibile. Permette inoltre di includere nel modello gli elementi di carattere qualitativo che contribuiscono a modificare il comportamento degli

agenti. Attraverso l'uso di un tale metodo di indagine è possibile concentrare l'attenzione sui meccanismi di propagazione delle informazioni e del loro peso sugli equilibri del modello. Spesso descrivere un sistema come quello delle catene di fornitura comporta la scrittura di sistemi di equazioni intrattabili dal punto di vista del calcolo. Spesso il modello risulta complesso, dove per complessità si intende un preciso fenomeno, misurabile matematicamente. La giustificazione dell'uso dei modelli multi-agente necessita di un pesante supporto teorico e pertanto sarà dedicato a questo argomento il capitolo 3, ma ad un'attenta analisi risulta probabilmente l'unica metodologia plausibile di indagine di una realtà così articolata e poco formalizzabile, a causa della presenza di agenti umani, che agiscono secondo comportamenti non perfettamente razionali.

Capitolo 2 Le tecniche di previsione

2.1 L'incertezza delle forniture

Il secondo livello al quale deve essere condotto lo studio delle catene di fornitura è rappresentato dalla ricerca di previsioni accurate dell'andamento delle variabili del sistema. Questa attività richiede un'attenzione particolare quando la necessità di predisporre gli elementi delle strutture produttive si manifesta molto tempo prima del loro reale utilizzo.

Già nel XVIII secolo si era formata la visione secondo la quale (Laplace) la conoscenza di "tutte" le leggi che governano i fenomeni consentirebbe di predire esattamente qualunque evento. In altri termini, il problema di prevedere non sorgerebbe qualora fosse noto il modello esplicativo, completo di tutte le interazioni del fenomeno che interessa. Ma la perfetta razionalità non appartiene al mondo reale.

Si deve accettare l'idea che il passato non può essere osservato in modo univoco. L'analisi di esso conduce, in generale, ad una gamma limitata di varianti interpretative, ciascuna delle quali ha implicazioni nel determinare gli avvenimenti futuri. Questi non possono mai essere anticipati, ma soltanto analizzati all'interno di una selezione relativamente ridotta di *scenari*, sostanzialmente riconducibili allo schema:

what…if

nel quale la relazione tra input d'informazioni ed output possibili è fondata sulla esperienza storica e su di un giudizio di modificabilità del passato. L'ipotesi dell'esistenza di una relazione con parametri fissi appartiene ad una famiglia di simulazioni volte a stimare le condizioni estreme (minimo, massimo) della possibile evoluzione futura.

La previsione non è rivolta a predire, ma a cercare di individuare le scelte alla nostra portata e gli elementi inattesi che possono intervenire a modificare la realtà. Essa contempla anche l'elaborazione di un giudizio di qualità, ossia di ciò che è attualmente auspicabile per l'oggi e di ciò che parrebbe esserlo per il domani.

La previsione ha fini di orientamento e di decisione strategica e si configura pertanto come una costruzione ipotetica tendente a riprodurre con inevitabile approssimazione un modello di comportamento, che può riguardare uno o più fenomeni, entro un intervallo temporale. A tal proposito Marbach (1991) affermava che:

> l'attività di previsione, nel tentativo di fornire una guida alla condotta, inevitabilmente concerne aspetti per i quali si dispone di una spiegazione parziale e di un incompleto patrimonio sperimentale. Ci è dato, infatti, di cogliere una parte soltanto delle circostanze influenti, per ricostruire un modello semplificato e provvisorio dei fatti allo studio. Del resto, ogni osservazione sperimentale è unica e mai esattamente riproducibile, perché l'universo può essere considerato come una entità "solidale", nella quale ogni accadimento è legato a

ciascuno degli altri ed in perpetuo divenire (*panta rei* di Eraclito): nulla è ripetibile, mai, tanto meno gli esperimenti. La cosiddetta ripetizione dell'esperimento altro non è quindi che una operazione approssimata, che avviene in condizioni al più simili, mai identiche. Poiché, dunque, ogni conoscenza è incompleta, per superare un atteggiamento passivo e passare all'azione non resta altro che utilizzare al meglio il patrimonio informativo disponibile.

Pertanto, superata l'illusione di poter arrivare, mediante lo studio del futuro, a previsioni certe, il margine di discrezionalità si riduce alla semplice possibilità di scegliere tra alcuni ben differenziati approcci previsivi. Questi si basano, in genere, sulla formulazione di modelli, che in una prima fase considerano valide nel futuro le relazioni verificate nel passato, ipotizzando successivamente alcune delle possibili variazioni future dei parametri di tali relazioni.

Il metodo *estrapolativo* si basa appunto sull'ipotesi di costanza dell'andamento delle variabili del modello, esplicitamente oppure no, in funzione della variabile *tempo*.

Il metodo denominato, invece, *simulativo*, o anche *proiettivo*, tratta ciò che avverrebbe qualora si verificassero date circostanze: fissati i valori di alcune variabili, il modello fornisce un'analisi di quelle endogene. Questo modo di procedere presuppone, in ultima analisi, la costruzione di sistemi internamente coerenti tra i quali selezionare quello giudicato più appropriato.

Il metodo definito *normativo* comprende, infine, le previsioni che esprimono il desiderio che date circostanze, oppure certi livelli delle variabili in analisi, si realizzino e tentano di individuare nel presente le possibili strategie per conseguire gli obiettivi prefissati, verificando inoltre la percorribilità di tali strategie.

2.2 Classificazioni di alcuni metodi previsionali

I dati relativi ad un fenomeno che è rilevato in funzione del tempo sono detti dati funzionali o serie storiche. Essi costituiscono la rappresentazione a punti di un fenomeno continuo. Spesso non si conosce il modello interno di funzionamento del sistema che genera il fenomeno che si vuole studiare e dunque l'analisi delle serie storiche è finalizzata alla comprensione della legge di causa-effetto che governa il sistema e, soprattutto, alla capacità di previsione degli accadimenti futuri, come dimostrazione del modello che spiega i dati.

L'analisi parte da un campione di curve e ne studia:

1. la rappresentazione grafica mediante tecniche di "lisciatura",
2. tecniche esplorative per l'analisi della variabilità,
3. tecniche di classificazione delle differenti unità statistiche.

La diverse impostazioni prendono spunto dalla classificazione nella quale sono considerati:

- *metodi esplorativi*, che sulla base del passato e del presente analizzano le possibilità future mediante tecniche proiettive;

- *metodi normativi*, che partono da una futura situazione desiderabile ed individuano nel presente le possibili strategie per realizzarla.

I metodi esplorativi, molto più formalizzabili, possono essere a loro volta suddivisi in:

- *puramente esplorativi*, adoperati quando non vi siano indizi di mutamento, né tra gli elementi dei quali si vorrebbe conoscere il comportamento, né tra le relazioni che li legano. Le tecniche di estrapolazione pura più frequentemente utilizzate individuano curve di sviluppo, ovvero il *trend*;
- *esplicativi*, utilizzati quando è disponibile una conoscenza, pure se approssimata, dei fattori che influenzano il mutamento delle relazioni. Le tecniche più adoperate sono, in questo caso, i modelli casuali (ARIMA);
- *ausiliari*, impiegati quando è necessario collegare in appropriata sintesi conoscitiva i risultati di differenti analisi previsionali esplorative, ciascuna applicata ad un ben preciso, e differenziato, ambito di interesse. Le tecniche in questo caso più comuni sono le cosiddette *matrici di impatto incrociato*, una tecnica che permette di attribuire, al verificarsi di un evento, le probabilità che accadano alcuni altri eventi significativi. Ciò permette di esprimere in forma matriciale l'insieme delle probabilità che caratterizzano un determinato scenario.

Al fine di individuare il *trend* si possono utilizzare le seguenti tecniche:

- *I filtri lineari*: tra i quali i più comuni consistono in una trasformazione lineare con una *media aritmetica mobile*. L'uso del filtro lineare dà luogo ad una serie filtrata meno fluttuante di quella di riferimento, ma si perdono le informazioni estreme.
- *I filtri alle differenze*: concentra l'analisi sulla dinamica del sistema, depurando i dati dalle variabili di stock, che possono falsare la metrica dei movimenti del livello.
- *Il metodo di Holt-Winters:* la previsione viene ottenuta estrapolando un *trend* lineare mediante parametri che cambiano nel tempo per adattarsi alle variazioni della serie.

La componente di *trend* può essere utilizzata per effettuare previsioni sui valori futuri della serie, eventualmente dopo aver depurato la serie dalla componente ciclica e stagionale.

La componente ciclica può essere messa in evidenza attraverso l'uso di tecniche, quali:

- *Coefficiente di autocorrelazione di ordine h:* non è altro che il coefficiente di correlazione lineare tra la serie osservata e la stessa sfasata di h posti.
- *Livellamento esponenziale:* ha la caratteristica di dare peso preponderante ai valori più recenti della serie. Ogni metodo di livellamento si basa sulla scelta della funzione matematica per l'estrapolazione.

Un altro interessante metodo di indagine statistica classica, capovolge il processo di analisi dei dati. Esso si basa sull'utilizzo di modelli della classe ARMA. Tale metodologia permette di orientare l'osservatore verso il modello corretto e, quindi, non necessita di una conoscenza preliminare sulla struttura e sulle caratteristiche dei dati. I modelli ARMA, sono fondati su una legge composta da una componente autoregressiva (da cui l'acronimo AR) e da componenti a media mobile (MA, *moving average*). Partendo dal modello, mediante i dati occorre stimare i coefficienti della componente autoregressiva e quelli di media mobile, in modo da poter determinare l'andamento futuro della serie.

Le tecniche quantitative possono essere applicate quando simultaneamente ricorrono le seguenti condizioni:

- sono disponibili informazioni sul passato;
- le medesime possono essere espresse numericamente;
- è plausibile ipotizzare che alcune tendenze del passato continueranno anche nel futuro (ipotesi di continuità).

I principali problemi riguardanti le tecniche previsionali derivano dall'esistenza di numerose limitazioni di carattere generale nello studio del futuro.

Si osserva, anzitutto, che la inadeguatezza quali-quantitativa dei dati di base della previsione può essere così rilevante da inficiare completamente la validità dei risultati che saranno ottenuti, qualunque sia il modello prescelto per effettuarla.

Ciò è, ovviamente, tanto più condizionante quanto maggiore è l'input che un modello richiede. Altri problemi derivano inoltre:

- da carenze nell'ambito della teoria, particolarmente accentuate nel caso dei metodi più complessi, nonché dalle intrinseche difficoltà che si incontrano nella fase esplicativa, soprattutto se i meccanismi interni del sistema considerato non sono ancora sufficientemente noti, situazione molto frequente;
- dalla necessità di considerare la fitta rete di legami, talvolta molto complessi e difficilmente identificabili, tra variabili o gruppi di variabili. Infatti, in una realtà come l'odierna, dove l'interazione tra fenomeni economici, sociali e politici è la regola, spiegare un qualsiasi fattore indipendentemente da tutti gli altri non è plausibile. Questo è il principale motivo che limita le possibilità applicative dei modelli di analisi delle serie storiche, ma, ad un tempo, costituisce una sfida per quelli che utilizzano, invece, il metodo Box-Jenkins;
- dalla necessità di costruire modelli cosiddetti *globali*, non limitati da impostazioni restrittive, che tengano conto della ormai sempre più stretta connessione internazionale dei fenomeni socio-economici.

Si può notare, infine, come sia indispensabile che i modelli incorporino elementi innovativi, di rottura, capaci di fornire informazioni sulle modifiche che interverranno nel sistema a seguito di eventi anomali. Tale necessità, affrontata, nei modelli econometrici, ricorrendo alle cosiddette variabili *dummy*, è inevitabile fonte di decisioni puramente soggettive circa gli eventi da prescegliere, ma risulta di innegabile importanza.

I *metodi quantitativi* richiedono generalmente formulazioni ed elaborazioni molto complesse che, però, non sempre forniscono stime migliori di quelle ottenibili da metodi puramente descrittivi, od addirittura empirici, usualmente denominati *naïf* perché fondati sulla banale estrapolazione delle linee di tendenza osservate. Purtroppo, anche i metodi più raffinati, si pensi, ad esempio, ai modelli pluriequazionali di previsione, soggiacciono alla medesima limitazione nonostante l'apparato formale, talvolta imponente, messo in campo.

Inoltre, le ipotesi alle quali tali metodi, soprattutto, i più complicati, devono solitamente soddisfare sono talmente rigide che i risultati sono affidabili soltanto in un campo piuttosto circoscritto. Molti modelli si fondano su alcuni parametri esogeni, la cui variazione condiziona fortemente il risultato. Ciò rende poco utile il modello e richiede invece la comprensione del sistema ad un livello globale, al fine di congetturare con buona approssimazione le variabili esogene. Solo in seguito a questa attività il modello può risultare utilizzabile.

2.3 Lo studio dei dati longitudinali funzionali

Le tecniche di previsione di tipo esplicativo si propongono di ricercare le relazioni di causa-effetto che danno origine ai fenomeni osservati. Si è dimostrato nella sezione precedente la moltitudine di tecniche a disposizione per l'analisi dei dati. In questo contesto lo sperimentatore deve seguire un percorso concettuale che parte dall'utilizzo di tecniche semplici e che prosegue complicandosi in funzione dei risultati e delle osservazioni via via ottenute. La scelta dei metodi più appropriati è spesso affidata alla sensibilità e all'esperienza dello sperimentatore.

Tale scelta, però, spesso dipende in modo significativo dalla natura dei dati. Se si vuole studiare, ad esempio, le rilevazioni di temperatura in una determinata zona geografica, ci si può attendere ragionevolmente che le osservazioni saranno caratterizzate da una componente di ciclicità stagionale. L'analisi di dati relativi ad un fenomeno sociale, invece, è spesso caratterizzata da forte rumore dovuto all'interazione di agenti cognitivi, cioè di agenti che modificano l'ambiente e si adattano ad esso.

La tecnica statistica della regressione è una metodologia di indagine fondamentale per comprendere la struttura dei dati.

Raramente un fenomeno è frutto di una semplice e perfetta relazione di causa-effetto da poter essere spiegato con una semplice relazione lineare, ma il primo approccio all'analisi dei dati

deve indagare se i dati sono caratterizzati da natura lineare e questa tecnica è un ottimo strumento per fare ciò.

In Grigoletto, Ventura (1998) si trova una breve sintesi teorica. Supponiamo di voler studiare come una variabile *x* influenza una variabile *y*, con *x* e *y* quantitative. La variabile *y* è detta *dipendente*, mentre *x indipendente* o *esplicativa*. Avendo a disposizione un insieme di osservazioni (x_i,y_i), $i=1, \ldots, n$, si vuole descrivere la relazione tra *x* e *y* mediante una funzione matematica rappresentabile da una curva che passa attraverso l'insieme di punti (x_i,y_i). Questa curva viene detta *curva di regressione di y su x* ed è rappresentabile come una funzione del tipo

$$y = a + b\hat{x}\,, \tag{1}$$

con α e β opportuni. Uno dei criteri più usati per trovare gli opportuni parametri della funzione è il metodo dei minimi quadrati, che consiste nello scegliere i valori α e β che minimizzano la somma degli scarti al quadrato dei valori y_i ed i corrispondenti $\hat{y}$ previsti dalla retta di regressione. Si scelgono cioè in base alla relazione:

$$\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - a - bx_i)^2 \tag{2}$$

È possibile dimostrare che i valori di α e β che minimizzano la (2) sono, rispettivamente,

$$\hat{a} = \bar{y} - \hat{b}\bar{x}_i$$

e

$$\hat{b} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i^{\,2} - n\bar{x}^2}\,, \tag{4}$$

dove

$$\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}\,,\ \bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}\,. \tag{5}$$

Al fine di comprendere se i valori α e β ottenuti simulino la natura lineare del fenomeno e risultino, di conseguenza, dei buoni parametri per prevedere la serie storica si deve usare un parametro che quantifichi la forza della relazione funzionale trovata. Per fare ciò si utilizza spesso il coefficiente di correlazione di Bravais. Esso è dato da

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}\,. \tag{6}$$

Il coefficiente di correlazione r_{xy} assume valori nell'intervallo [-1,1], con $r_{xy}=1$ che indica una relazione lineare perfetta, $r_{xy}=0$ nessuna relazione e $r_{xy}=-1$ una perfetta relazione negativa tra

i dati. Se tale coefficiente si avvicina ad 1 in valore assoluto la tecnica della regressione lineare può essere un buon metodo di stima dei dati.

Difficilmente, però, la spiegazione di un fenomeno in termini lineari è un successo, soprattutto in caso di fenomeni complessi. Per questo motivo si deve fare affidamento a tecniche più approfondite che introducono l'ipotesi di non linearità.

In Freund, Wilson (1998) si osserva che

> … we noted that some models that are not linear in the parameters cannot be so by the use of transformations. We called these *intrinsically nonlinear* models.

Si consideri la relazione funzionale generica del tipo

$$y = f(x_1,...,x_m, b_1,...,b_p) + e \,, \tag{7}$$

dove f è una qualsiasi funzione delle m variabili indipendenti $x_1, \ldots, x_m$, e dei p coefficienti $b_1,\ldots, b_p$. Il valore di m può essere diverso da p. Normalmente si considera la componente di rumore ε con media 0 e varianza σ^2. Si noti che in caso sia vera la relazione

$$f(x_1,...,x_m, b_0,...,b_m) = b_0 + b_1 x_1 + ... + b_m x_m \,, \tag{8}$$

ci si trova nel caso della regressione lineare prima descritta.

Il metodo usato per stimare i coefficienti nel caso in cui la funzione f sia non lineare è lo stesso dei modelli di regressione lineare, seppure molto più complesso dal punto di vista del calcolo. Si tratta, cioè, di minimizzare la seguente

$$\sum_{i=1}^{n} [y - f(x_1,...,x_n; b_1,...,b_n)]^2 \,. \tag{9}$$

Il problema nasce dal fatto che ci possono essere più coefficienti che variabili indipendenti ($p>m$) e, soprattutto, che la funzione di minimizzazione non produce equazioni lineari, per cui rendere il problema ridotto ad una forma sintetica, o trovare la soluzione esatta è impossibile. Le soluzioni sono ottenute attraverso un processo di ricerca iterativa.

Un tale processo parte con alcune stime preliminari dei parametri. Esse sono usate per calcolare le somme degli scarti al quadrato e danno un'indicazione su quali modifiche apportare alle stime dei parametri al fine di minimizzare gli scarti. Il processo è ripetuto un numero di volte sufficiente a ridurre significativamente l'errore della funzione di regressione.

In letteratura esistono molti metodi iterativi per risolvere le equazioni relative ai modelli non lineari, in seguito si affronta quello dell'algoritmo di apprendimento delle funzioni a rete neurale.

Il metodo più diffuso in statistica per costruire una relazione funzionale che spieghi i dati di una serie storica è basato sull'uso dei modelli ARMA, i quali si fondano sul concetto di autoregressione e di media mobile.

Se, cioè, si ipotizza che la relazione (1) che descrive le coppie di dati (x_i,y_i) sia sostituita dalle coppie di valori (x_{i-k},x_i) si crea una funzione che dipende intrinsecamente da se stessa. In questo caso il calcolo del coefficiente di correlazione della (6) diventa

$$r_k = \frac{1}{n-k}\sum_{t=1}^{n-k} x_{t+k}x_t - \left(\frac{1}{n}\sum_{t=1}^{n} x_t\right)^2 . \qquad (10)$$

Attraverso questa metodologia, detta di autocorrelazione, si può indagare la natura di stazionarietà della serie storica. La proprietà di decadimento della funzione di autocorrelazione indica se questa condizione è verificata. In tal caso si può considerare l'uso dei modelli ARMA.

In Delgado, Prat (1996) si espone il modello di previsione fondato sui modelli autoregressivi e a media mobile del tipo Box-Jenkins. Il caso più semplice di autoregressione lineare (AR) usa i precedenti *p* valori della serie per definire il modello.

$$z_t = f_1 z_{t-1} + f_2 z_{t-2} + \ldots + f_p z_{t-p} + a_t , \qquad (11)$$

dove f_i sono i parametri di autoregressione e a_t è il rumore bianco.

La media mobile (MA) dipende dai valori della serie definiti dai *q* precedenti errori di previsione $a_t,a_{t*1},\ldots$ del modello

$$z_t = a_t - q_1 a_{t-1} - q_2 a_{t-2} - \ldots - q_q a_{t-q} , \qquad (12)$$

dove q_i sono i parametri di media mobile.

Quando entrambi i metodi vengono combinati si ottiene un modello ARMA(*p*,*q*) definito come

$$z_t = \sum_{i=1}^{p} f_i z_{t-i} + a_t - \sum_{k=1}^{q} q_k a_{t-k} , \qquad \text{con } t=p+1,\ldots,N. \qquad (13)$$

I modelli ARMA sono stazionari nel senso che sono localizzati attorno ad una media μ e ad una varianza σ^2. In molti casi i dati osservati non sono stazionati attorno ad un valore medio. Ma se è possibile osservare un *trend* o una componente ciclica stagionale, la stazionarietà può essere ottenuta con una trasformazione lineare.

Si definisca la funzione ∇ detta *operatore alle differenze prime* come

$$\nabla z_t = z_t - z_{t-1}, \qquad (14)$$

applicando *d* volte tale operatore alla serie storica non stazionaria si ottiene una nuova serie w_t del tipo

$$w_t = \nabla(\nabla(\nabla..\nabla z_t))) = \nabla^d (z_t) . \qquad (15)$$

Il modello ARMA(*p*,*q*) che risulta applicato alla serie w_t assume il nome di ARIMA(*p*,*d*,*q*). Nel caso i dati abbiano una componente stagionale, si introduce l'*operatore alle differenze stagionali* ∇_s,

$$\nabla_s z_t = z_t - z_{t-s} \tag{16}$$

che se applicato *D* volte diventa ∇_s^D e produce un modello SARIMA(*p*,*d*,*D*,*q*).

I modelli ARIMA forniscono una buona risposta al problema della previsione dei dati, però c'è un limite alla loro applicazione dato che essi sono validi solo per processi stazionari. A volte si devono fare molte differenze prime per ottenere dalla serie di dati di partenza una successione stazionaria, il che aumenta la dimensione caratteristica del processo in modo artificiale. Le reti neurali possono essere applicate a qualunque successione senza bisogno della stazionarietà, in generale sono usate per ricostruire qualsiasi serie di dati generati da una qualsiasi trasformazione non lineare e non deterministica.

2.4 Le reti neurali

In Weigend et al. (1992) si afferma che quando la conoscenza delle leggi che regolano un fenomeno è espressa nella forma di equazioni deterministiche che possono essere risolte, almeno in linea teorica, il futuro diventa predicibile, una volta che le condizioni iniziali vengano completamente specificate.

La più comune situazione in cui deve essere svolta una previsione richiede, però, la ricerca di regolarità empiriche nell'osservazione del sistema, poiché il modello in generale non è noto. Le oscillazioni di un pendolo, il ripetersi delle stagioni, possono potenzialmente essere predette, sulla base della conoscenza della loro ciclicità pur non conoscendo i meccanismi interni che li regolano. Purtroppo, le regolarità non sono sempre evidenti, poiché spesso mascherate dal rumore. Ci sono fenomeni che, sebbene ricorrenti in senso generale, appaiono come casuali, privi di apparente periodicità. Attraverso l'uso le reti neurali è possibile estrarre la conoscenza di questi fenomeni osservando il passato, con l'obiettivo di predire la sequenza temporale futura.

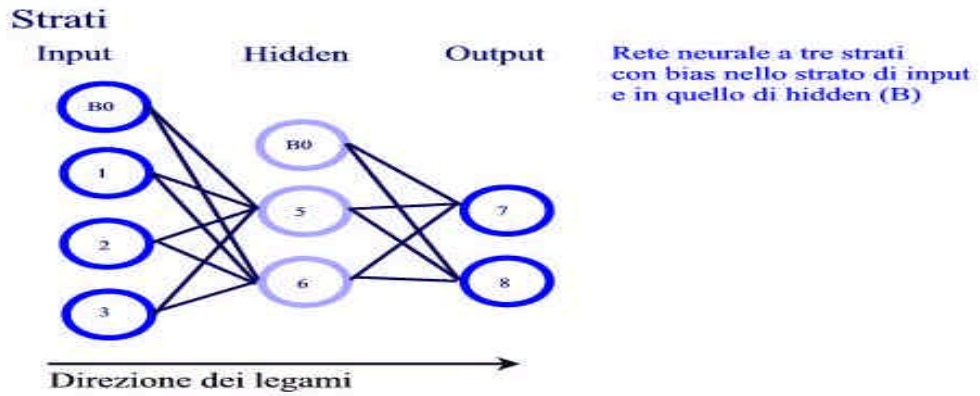

fig. 2 Schema di una RNA

In estrema sintesi, una funzione a rete neurale artificiale (RNA) di tipo *feedforward* è una funzione che trasforma un vettore di *input* in un vettore di *output*, sulla base di una matrice di

parametri detti *pesi* (si veda fig. 2), la cui determinazione non è concettualmente diversa dalla determinazione di stime statistiche in ambito econometrico (multivariato), ma presenta caratteristiche particolari in termini di tempi di calcolo e di difficoltà di convergenza degli algoritmi standard di stima.

I parametri della funzione o pesi, vengono ricavati applicando reiterativamente un algoritmo di propagazione dell'errore (*backpropagation*) generato dalla funzione rispetto all'output atteso, detto *target*. L'algoritmo è applicato ad un insieme di dati di apprendimento dei quali si conosce il *target* (detto *training set*). La matrice dei pesi risultante da tale processo è verificata su dati di controllo (*verification set*) che presentano *pattern* (casi) non utilizzati in fase di apprendimento. Se il risultato è buono, la rete è applicata a dati via via nuovi per fornire previsioni.

Ciò che si ottiene da questo processo è una funzione non lineare del tipo **f(B f(Ax))**, che rappresenta la trasformazione del vettore di input nel vettore di output ed è detta matematicamente funzione a *rete neurale*. Le matrici dei coefficienti **A** e **B** rappresentano i pesi della funzione.

Detto wih_{ij} il peso che collega il nodo *i* dello strato di *input* con il nodo *j* di quello nascosto e who_{ij} il peso tra i nodi *i* e *j* rispettivamente degli strati nascosti e di *output* la funzione si può esprimere nella forma

$$y_o = f\left(\sum_j\left(f\left(\sum_i wih_{ij}\cdot x_i\right)\cdot who_{jo}\right)\right) \tag{17}$$

White (1988) afferma che le procedure di apprendimento delle reti neurali sono di per se stesse delle tecniche statistiche. Le reti neurali possono organizzare i dati, eseguire analisi statistiche, individuare le tendenze, adattarsi e apprendere dai dati, effettuare azioni correttive e prevedere particolarità future che possono manifestarsi, tutto ciò in tempo reale. Le RNA utilizzano operazioni numeriche per effettuare classificazioni, paragoni, previsioni, reperimento di informazioni, azioni, in quanto sono degli stimatori universali.

Dal punto di vista statistico il processo di apprendimento non è altro che un processo iterativo che ricerca la soluzione di un modello a regressione multipla non lineare, minimizzando la (9). Il modello non è altro che un sistema di equazioni del tipo (7) le cui incognite sono le matrici A e B. La rete neurale può essere vista come una scatola nera con un'architettura iniziale che, dopo un processo di apprendimento volto a stimare i pesi, diventa un modello esplicativo messo a punto sui dati che si hanno a disposizione.

L'algoritmo di apprendimento cerca la combinazione ottima della matrice w^* nello spazio dei pesi, minimizzando una funzione di costo detta *E(w)*, definita come

$$E(w) = \frac{1}{2}\sum_p \sum_i (x_i^p - d_i^p)^2 \,, \tag{18}$$

dove x è il risultato prodotto dalla matrice dei pesi w^*, e d è il prodotto desiderato. L'algoritmo di *backpropagation* parte dal calcolo dell'errore $e=x-d$ ottenuto come differenza tra il risultato fornito dalla rete (x) e il *target*, ovvero il risultato atteso (d). Questo errore è "propagato" dallo strato del nodo di uscita all'indietro verso i nodi di ingresso, producendo piccole correzioni. Ogni coefficiente w_{ij} è corretto proporzionalmente all'influenza del suo peso nella funzione E[2].

Per calcolare la correzione del peso w_{ij}, è necessario calcolare il gradiente ∇E come

$$\nabla E = \frac{\partial E}{\partial w_{ij}}, \tag{19}$$

dal quale si ottiene la relazione ricorsiva che modifica il singolo coefficiente

$$w_{ij}(t+1) = w_{ij}(t) - a\,\frac{\partial E_i}{\partial w_{ij}}, \tag{20}$$

dove α è una costante, detta tasso di apprendimento (*learning rate*).

In Beltratti, Margarita, Terna (1996) si ricorda che spesso tale coefficiente è mantenuto sufficientemente piccolo da consentire all'algoritmo di apprendimento di giungere ad una soluzione. Un valore piccolo ha le conseguenze negative di aumentare il numero delle iterazioni necessarie per ottenere la soluzione.

È consuetudine aggiungere un secondo parametro che modifica l'algoritmo di *backpropagation* in modo da utilizzare la formula di correzione:

$$\Delta_t w_{ij} = -a\,\frac{\partial E_t}{\partial w_{ij}} + b\Delta_{t-1} w_{ij}\,, \tag{21}$$

dove β è detto *momentum*. Esso introduce alcuni gradi di persistenza nella modifica dei pesi, poiché la variazione al tempo t dipende anche da quella al tempo $t-1$. È utile a ridurre l'eccessiva oscillazione dei pesi.

2.5 Le caratteristiche delle RNA

L'insieme dei fenomeni che caratterizzano un sistema economico, tende a fare emergere alcune ricorrenze dominanti, dovute alla capacità adattiva degli agenti attori di un tale sistema. Sulla base di questa considerazione la nuova visione dell'economia considera che la crescita dei percorsi del sistema economico non discende direttamente da alcune condizioni iniziali e da leggi di interazione deterministiche. Al contrario, spesso si riscontrano effetti evidenti di eventi poco rilevanti.

[2] Si ricorda che w_{ij} ha un peso sulla funzione di costo E, poiché il valore x_i in essa contenuto è frutto della applicazione della (17).

In una tale visione la strada più promettente nella ricerca e nell'analisi di questi fenomeni è quella dell'adozione di tecniche adattive, basate su meccanismi di intelligenza artificiale. Attraverso l'approccio cognitivista si delega il compito della ricerca delle relazioni intrinseche che danno origine ai fenomeni di un sistema ad agenti in grado di apprendere dal comportamento degli altri agenti e dall'ambiente nel quale sono integrati. Le modalità che regolano il percorso dell'apprendimento diventano un componente fondamentale del modello. Lo sviluppo delle funzioni a rete neurale spesso porta alla definizione di equazioni non lineari, molto complesse che sono in grado di mappare la realtà passata e talvolta a prevedere con sorprendente precisione gli accadimenti futuri.

In Terna (1994) si identificano i vantaggi principali derivanti dall'uso delle funzioni neurali derivano dalla natura *connessionista* di questo sistema. Il connessionismo identifica i sistemi dotati di:

- *Parallelismo*: i carichi computazionali sono ripartiti tra gli elementi di una struttura costituita da legami paralleli. Il risultato è un sistema con capacità di calcolo indipendente ed interattivo, con capacità di risposta immediata.
- *Rappresentazione subsimbolica della conoscenza:* in una struttura connessionista la conoscenza non è memorizzata in modo esplicito o simbolico (Hinton, 1986). Tale conoscenza risulta distribuita all'interno del sistema.
- *Autorganizzazione:* le strutture impersonali del mondo economico reale, come il mercato, sono in realtà frutto dell'azione di collettività di agenti autorganizzanti.
- *Robustezza all'errore e ridondanza*: derivano direttamente dalla capacità di memorizzazione distribuita. Quando la struttura dovesse essere danneggiata, non avverrebbe un blocco, ma soltanto un degrado del funzionamento. Anche per questa caratteristica, quando l'input si presenta disturbato da un rumore, il sistema di risposta riesce in modo più o meno efficiente ad eliminarlo.

Le funzioni a rete neurale, però, non spiegano le relazioni che generano i fenomeni. Esse funzionano perfettamente ma non hanno una plausibilità dal punto di vista del ragionamento umano. I valori contenuti nei nodi nascosti della rete sono strettamente correlati con la rappresentazione distribuita o sub-simbolica dei significati intrinseci, per questo è così difficile interpretarli direttamente. In sintesi si può dire che i nodi nascosti si comportano come una funzione la cui forma, aggregata a quelle prodotte dagli altri nodi, coopera nell'approssimazione della funzione obiettivo. Ciò si può comprendere realizzando un paragone tra le struttura di una RNA e quella di un'organizzazione sociale. Il comportamento generale del sistema emerge spesso dall'interazione apparentemente disordinata delle sue componenti. Gli agenti di un sistema hanno un comportamento di tipo microeconomico, non sono coscienti degli effetti che questo comporta in aggregato. Allo stesso modo i nodi di una rete neurale sono banali "calcolatrici" che applicano regole elementari. Non è immediatamente comprensibile il motivo per cui la rete nel suo

complesso sia in grado di adattarsi ai *pattern* di *input* sulla base di una fitta rete di semplici funzioni non lineari che svolgono banali sommatorie.

Esistono diversi metodi consolidati per la ricerca di una spiegazione interpretativa della struttura assunta dalla matrice dei pesi dopo un processo di apprendimento. Una buona strada di indagine circa l'*innaturale* struttura interna della funzione è fornita dall'interpretazione dei nodi nascosti, la tecnica di *cluster analysis*, l'esame delle derivate tra output ed input della funzione, l'introduzione di un'euristica adatta a estrarre regole da una RNA e la lettura delle interazioni in modelli fondati su agenti.

L'analisi di *cluster* è una tecnica statistica fondata sull'idea di misurare le distanze, in una metrica coerente, tra i punti di uno spazio multidimensionale. Applicandola, si possono formare molti gruppi, o *cluster*, di vettori. Per ogni cluster è interessante investigare le caratteristiche dei pattern associati con i vettori così raggruppati; in molti casi si scopre un *nome* per i gruppi di pattern di un cluster che rappresentano situazioni, oggetti, ecc. con una caratteristica comune. Inoltre, analizzando cluster di cluster, può emergere una completa classificazione gerarchica. Nel caso delle RNA analizzare la struttura dei *cluster* in funzione di una determinata categoria di dati di *input* fornisce l'interpretazione della compito svolto da determinati gruppi di nodi.

Questo è molto interessante da un punto di vista cognitivo, perché può spiegare come le reti neurali inferiscono sui pattern, assimilandoli internamente a differenti cluster, sebbene non diano una spiegazione a livello di singolo nodo. Il vantaggio esplicativo di tale metodo va ricercato nella capacità di spiegazione distribuita tra i nodi nascosti, poiché ciò che si sta misurando è la distanza tra vettori e non tra singoli nodi. Quindi l'analisi a cluster dello strato dei nodi nascosti spiega l'organizzazione strutturale dei pattern.

Un altro limite di questa tecnica di indagine previsionale è data dalla difficoltà di impostazione dei parametri caratterizzanti una RNA. Si può incorrere in un problema di *overfitting*, che può diventare molto serio. In generale, non è noto quale sia l'ampiezza ideale di una rete neurale per risolvere un dato problema. Se la rete è troppo *stretta*, nel senso che possiede pochi nodi nello strato nascosto, essa non sarà sufficientemente flessibile da emulare la dinamica del sistema che produce la serie storica (fenomeno di *underfitting*). Se viene scelta troppo *larga* (troppi nodi hidden), l'eccessiva libertà permette alla rete di apprendere non soltanto il segnale, ma anche il rumore (*overfitting*). Al fine di verificare di non essere incappati nella trappola di una rete eccessivamente estesa, si utilizza la tecnica della *validazione interna*. Tale tecnica consiste nella divisione della serie storica in due serie separate. La prima verrà utilizzata per il procedimento di apprendimento e la seconda, costituita da pattern che la rete non ha mai incontrato prima, per la verifica circa la bontà dell'apprendimento.

Infatti, la presenza di molti nodi fa sì che ogni nodo possa "imparare a rispondere" ad ogni piccolo gruppo di *pattern* ed il comportamento generale che si ottiene è una funzione che apprende esattamente la struttura che ha osservato. Ciò la rende rigida e ne limita la capacità di

generalizzazione. Quando, al contrario, i nodi sono pochi la previsione che la rete effettua può essere valida se i dati non oscillano molto e sono relativi ad una casistica ridotta.

2.6 Il confronto tra i diversi metodi

È interessante in Bellacicco, Lauro (1997) il confronto di alcuni metodi classici di analisi delle serie temporali con i risultati ottenuti attraverso l'uso si reti neurali (*back-propagation*).

Il contesto in cui le reti neurali hanno origine e in cui trovano principale campo di applicazione è quello caratterizzato da applicazioni di tipo *data intensive*, ossia con molti più dati che calcoli, in cui i dati stessi sono spesso incompleti o affetti da "rumore" e non registrati su archivi ma continuamente aggiornati, e per le quali i risultati ottimali sono ottenuti utilizzando macchine con dispositivi di calcolo parallelo. Il metodo, quindi, è prevalentemente *problem-oriented*, rivolto cioè alla ricerca si una soluzione che non sia necessariamente "esatta" ma che consenta di risolvere lo specifico problema considerato con un buon grado di approssimazione.

I principali problemi che è possibile individuare possono essere ricondotti alla mancanza di criteri univoci nella definizione dell'architettura neuronale e alla non assicurata convergenza verso il minimo globale. Al contrario, la metodologia statistica, sia essa riferita ad un contesto descrittivo o inferenziale, si caratterizza per la interpretabilità e la trasparenza non solo del risultato finale ma di tutte le fasi che a questo conducono. La non univocità della soluzione finale, che delle reti costituisce una caratteristica e che in quel contesto non viene considerato un problema determinante, non sembra avere analogo riscontro in statistica dove la prospettiva è invece generalmente di tipo *model-oriented*, rivolta cioè alla individuazione della struttura sottostante ai dati e in cui l'obiettivo può essere rappresentato dalla stima dei parametri dell'unico modello in grado di spiegare i dati stessi sulla base di criteri di ottimizzazione prefissati.

La capacità di generalizzazione della rete, che ne costituisce uno degli indiscussi punti di forza, risulta infatti influenzata dalla scelta della numerosità del campione e dalla rappresentatività di questo. L'aleatorietà legata alla scelta iniziale dei pesi, non bilanciata da un numero sufficiente di *patterns*, può infatti influire in maniera determinante sul risultato finale, ancor più nel caso in cui venga applicata dal regola dello *stopped training* secondo la quale il campione iniziale viene diviso in un *set* di apprendimento ed in uno di verificazione. In questo caso, inoltre, ulteriori problemi possono presentarsi quando si abbia a disposizione un insieme di dati di numerosità limitata in cui una ulteriore divisione potrebbe inficiare l'attendibilità della *performance*. La capacità di generalizzazione di una rete può essere allora valutata ricorrendo ad esempio ai metodi di ricampionamento quali *bootstrap*, *jackknife* e *cross-validation* già ampiamente sperimentati nella metodologia statistica classica.

In Delgado, Prat (1997) si fa notare che:

> Neural Networks have confirmed their flexibility and accuracy in the nonlinear regression modeling and time series forecasting. The NN are only a continuation of the classical statistical point of view including new and promising challenges provided by the Artificial Intelligence paradigm.

Per completezza è opportuno citare l'esistenza di un'altra metodologia di analisi. Essa è costituita dalle cosiddette "tecniche non parametriche per lo studio dei dati longitudinali funzionali" (Bosq 1998). Si fondano sul principio teorico della regressione lineare, ma in questo caso le variabili indipendenti non sono rappresentate da valori osservati relativi ad una serie storica, ma da variabili aleatorie *hilbertiane*, che consistono in funzioni di valori reali con caratteristiche di separabilità.

Tale metodologia è in fase di verifica e dai primi risultati pare essere molto promettente, se paragonata ai tradizionali metodi ARMA.

Capitolo 3 La complessità nella simulazione dei modelli sociali

3.1 La simulazione sociale

Secondo quanto afferma Conte(1997), la simulazione sociale è un raggruppamento di discipline a cavallo fra la scienza computazionale e le scienze sociali, che negli ultimi anni ha ricevuto notevole impulso. Vi sono diversi tipi di impieghi della simulazione su computer.

Distinguiamo almeno i seguenti obiettivi:

- **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, ecc. 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.
- **Sperimentazione**: a volte, la verifica dei modelli non viene effettuata sul computer soltanto allo scopo di ottenere risultati immediati ed affidabili, ma anche allo scopo di riprodurre in "vitro" il fenomeno da interpretare. L'obiettivo in questo caso non è la previsione, ma la spiegazione di fenomeni della realtà. Contribuisce all'indagine la manipolazione delle variabili del modello costruito. Ciò che non è consentito effettuare in natura è possibile ottenere in "laboratorio". Il computer diventa così il laboratorio delle scienze sociali.
- **Esplorazione**: i primi due impieghi hanno a che vedere con l'implementazione sul computer del modello, o di un *set* di modelli, di un fenomeno reale. Ma la simulazione su computer 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.

È a questo tipo di simulazione, ossia alla simulazione di società di agenti, che rivolgeremo ora la nostra attenzione. L'utilizzo del computer a scopo sperimentale appare come un'occasione di emancipazione delle scienze sociali, e in particolare della teoria sociale, da una condizione di relativa infalsificabilità, ovvero dalla difficoltà di sottoporre a verifica empirica gli enunciati teorici. Dall'altro l'implementazione di società di agenti consente una prospettiva altamente interdisciplinare.

Con la simulazione sociale si intende l'uso di modelli precisamente specificati, formulati ed eseguiti su computer, per ricreare e studiare aspetti essenziali della socialità e delle società naturali (umane e non umane) e 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.

Uno studio simulativo consta di solito di due fasi distinte, il cosiddetto *modeling*, ossia la costruzione del modello da simulare e la sua simulazione sul computer.

I modelli matematici, come quelli elaborati dagli economisti, definiscono gli agenti unicamente in base ai valori assegnati a determinate variabili (preferenze, probabilità di occorrenza degli *outcomes*, rischio) e le situazioni in termini di matrici di *pay-off*. I modelli informatici invece non necessariamente definiscono gli agenti in base a variabili numeriche, ma in base alle sequenze di azioni e di operazioni che essi eseguono, e in base a rappresentazioni interne (regole).

Per quanto riguarda l'affidabilità dei modelli, occorre osservare che esistono studi relativi alla stima dei risultati ottenuti attraverso i modelli simulativi. Essi presentano diversi limiti: quello più ovvio consiste nell'elaborazione di previsioni non verificate. L'aspetto forse più critico consiste nella elaborazione di previsioni troppo generiche che non consentono una stima effettiva della affidabilità dei modelli.

Vi sono almeno due casi nei quali appare ragionevolmente necessario eseguire la simulazione:

- Nei modelli dotati di capacità di apprendimento: si tratta di quegli studi nei quali gli agenti sono capaci di apprendere dall'interazione con l'ambiente fisico o sociale. Ciò è possibile grazie ad architetture basate su reti neurali o a sistemi basati su regole.
- Negli studi sull'emergenza di fenomeni nuovi (ad esempio la cooperazione) da fenomeni noti. Fenomeni emergenti vengono osservati in sistemi dinamici e capaci di auto-organizzazione, i quali si modificano per rispondere a cambiamenti sopravvenuti nell'ambiente, con ciò modificando o contribuendo a modificare l'ambiente stesso.

In questi tipi di studi l'analisi descrittiva del modello non potrebbe sostituire la simulazione del programma sul computer poiché quest'ultima è necessaria per consentire ai sistemi di agire effettivamente in un mondo artificiale.

Certamente, la simulazione ha dei costi, che non sempre vengono bilanciati dai risultati ottenuti, i costi di programmazione, di apprendimento, di esecuzione, di elaborazione in senso statistico, di interpretazione, di *debugging*, di trasferimento e infine di comprensione.

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

Un terzo costo è rappresentato dalla "trappola della verosimiglianza", ossia dalla moltiplicazione delle categorie descrittive al fine di rendere la simulazione più realistica, più emulativa, ma che richiede un grande sforzo progettuale.

Inoltre va considerato che (Terna 1998):

> Choosing the agent based model paradigm we enter a wide unexplored world where methodology and techniques are largely "under construction". We are working in the bottom-up direction, putting together pieces of software (out agents) which can react to stimuli from the environment or from other agents.

Sempre in Conte (1997) si identificano i seguenti temi unificanti che sono emersi dai vari studi:

- emergenza: insorgenza spontanea ma non necessariamente graduale di effetti rilevanti per il successivo livello di adattamento di un sistema;
- complessità: ovvero differenziazione, specializzazione funzionale, *clustering*, formazione di coalizioni, organizzazioni.
- evoluzione: modifiche adattive dei vari sistemi in risposta al loro ambiente;
- distribuzione: condivisione di proprietà e comportamenti (strategie stabili, conformità) da parte delle unità del sistemo;
- micro-macro: meccanismi e fattori di raccordo fra vari livelli di complessità (ad esempio, fra agenti individuali e sovraindividuali, oppure fra coalizioni spontanee all'interno di un'organizzazione);
- 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.

3.2 La teoria della Complessità

In Epstein e Axtell (1996) si osserva che molti dei processi oggetto delle nostre analisi sono complessi, poiché non si possono scomporre in sub-processi. 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 Gleick (1987) si osserva che:

> Where chaos begins, classical science stops. For as long as the world has had physicists inquiring into the laws of nature, it has suffered a special ignorance about disorder in the atmosphere, in the fluctuations of the wildlife populations, in the oscillations of the heart and the brain. The irregular side of nature, the discontinuous and erratic side -- these have been puzzles to science, or worse, monstrosities.

È importante puntualizzare che il termine di “complessità” nello studio dei modelli dinamici ad agenti non è un sinonimo di “complicato”, di “difficile da comprendere”. Per complessità si intende un fenomeno matematicamente definibile.

In Waldrop (1992) si definisce come

> ... at a kind of abstract phase transition called "the edge of Chaos," you also find complexity: a class of behaviors in which the components of the system never quite lock into place, yet never quite dissolve into turbulence, either. These are systems that are both stable enough to store information, and yet evanescent enough to transmit it. These are the systems that can be organized to perform complex computations, to react to the world, to be spontaneous, adaptive, and alive.

Oppure in Kauffman (1995)

> ... just near this phase transition, just at the edge of chaos, the most complex behaviour can occur--orderly enough to ensure stability, yet full of flexibility and surprise. Indeed, this is what we mean by complexity

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 altre discipline, a causa dell’interazione di calcoli umani nelle decisioni. Pur tenendo presente tale limitazione, la definizione di Day è l’unica che premette 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 nonlinearità del XX secolo:

- La cibernetica (Forrester 1961) enfatizza la 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.

- La teoria sulle catastrofi fu elaborata da Thom (1975) sulla base delle precedenti teorie dei 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 comporta il salto da un punto all'altro man mano che i parametri del sistema variano.
- 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 una più approfondita analisi statistica esse appaiano casuali. Ne è un esempio il "fenomeno della farfalla", in cui una piccola differenza nei valori iniziali del modello o di alcuni parametri determinano traiettorie vistosamente differenti.
- 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 ad 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, 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.

3.3 Complex adaptive systems

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 cambiano 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:

- la *fisica*, per quanto riguarda l'analisi delle quattro forze della teoria dei campi elettromagnetici;
- la *biologia*, relativamente ai principi della selezione naturale;
- il *razionale*, per gli effetti delle decisioni assunte dagli attori del sistema;
- 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 intellettiva. 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 di adatta ad esso. In biologia questo fenomeno è detto “coevoluzione”.

Holland (1995) sostiene che l’analisi dei CAS deve essere condotta a due livelli: lo studio dei modelli interni e i *building block*.

Identifica i modelli interni come 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 persuppongono un’esplicita, sebbene interna, esplorazione delle alternative, un processo anche detto *lookahead*.

Ciò è giustificato dalla constatazione che la struttura dell’ambiente agisce attivamente a determinare il comportamento del singolo agente. Quindi, se accade che 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 di un'azione corrente, l'evoluzione può favorire i modelli interni efficienti e eliminare quelli inefficienti.

Le speranze di sopravvivenza dell'agente dipendono criticamente dalla sua capacità di prevedere, implicitamente o esplicitamente, i dettagli del modello. 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 questo "schema". 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 riusabile 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 neuronale, 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 la riusabilità significa ripetizione, è necessario ricercare una metodologia in grado di cogliere le 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 livello, alle sole interazioni, e ad un livello più basso, 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)

> concerns the emergence of complex large-scale behaviours from the aggregate interactions of less complex agents.

Essa 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. Nelle organizzazioni 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):

- Ogni sistema di questo tipo è composto da una rete di componenti che costantemente e reciprocamente si influenzano tra loro. Esse possono essere i neuroni di un cervello, come operatori di un mercato economico.
- Gli agenti non sono controllati centralmente, i fenomeni emergono dalla loro interazione. Nessuna componente è in grado di controllare le altre. I comportamenti, apparentemente coerenti, sono frutto della competizione e della collaborazione tra i componenti.
- 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*.
- Il meccanismo fondamentale è l'adattamento, che si realizza in seguito alla costante riorganizzazione interna delle componenti.
- 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.
- Le opportunità sono create dallo stesso sistema che sviluppa costantemente nuovi fenomeni o situazioni.
- 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.

3.4 ACE

Uno dei più importanti effetti dello studio della dinamica dei modelli economici complessi è stato il cambiamento nel metodo e nella direzione della ricerca. In particolar modo con i modelli basati sulla descrizione del comportamento di agenti individuali distribuiti, con l'uso della simulazione al computer. Nei nuovi modelli soltanto le interrelazioni a livello locale tra agenti individuali sono descritte, mentre i comportamenti o le strutture aggregati emergono dall'auto-organizzazione, invece che semplicemente imposti o accettatati. Ciò che può emergere in aggregato può non essere la semplice somma di quello che avviene a livello individuale, in contrasto con il metodo dei modelli ad agenti rappresentativi nei quali gli individui equivalgono all'aggregato. Sebbene questo vantaggio non sia garanzia che i modelli complessi risultino più accurati o utili di quelli prodotti con altre metodologie.

La giustificazione nell'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) si definisce il cosiddetto *Agent-based computational economics (ACE)* come lo 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 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, nel senso 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 il consumatore rappresentativo. In Testfatsion (1996) si trova una buona definizione della materia.

> Agent-based computational economics (ACE) is the computational study of economies modelled as evolving decentralized systems of autonomous interacting agents. ACE is thus specialization to economics of the basic Alife paradigm.

In Terna (1997) 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: un agente che agisce in un ambiente economico deve svilupparsi ed adattarsi coerentemente. Soprattutto quando l'agente deve interagire con altri agenti.

La complessità può essere riscontrata più frequentemente al di fuori degli agenti – in un contesto che emerge dall'interazione, dall'adattamento e dall'apprendimento – che al loro interno. 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.

Non è corretto dedurre dalla descrizione dell'agente ogni apparato formale ricercando la complessità al loro interno, con l'utilizzo della metafora.

> 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, possibilmente anche il modo in cui essi si discostano dalla razionalità perfetta. L'obiettivo è, dunque, non soltanto riprodurre la curva di apprendimento che riproduce 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 indistinguibile da quello umano. (Arthur 1990)

Al fine di evitare di incappare nell'errore di creare 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.

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, non sarebbero più richieste le 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) dà una convincente spiegazione del vantaggio di questo metodo di indagine scientifica. 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.

3.5 Il processo di identificazione e descrizione della conoscenza

La costruzione di modelli di simulazione, fondata su agenti, è un metodo decisamente teorico di analisi dinamica delle strutture sociali, sebbene l'obiettivo sia la plausibilità. Attraverso un siffatto modello 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, ma anche per simulare un sistema aziendale concreto. L'indagine basata sulla simulazione offre 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 porre 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 dell'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. Che tutto ciò non costituisca conoscenza è dimostrato dalla quantità di sforzi e di insuccessi nei quali un soggetto incappa durante la ricerca e l'astrazione della conoscenza necessaria allo svolgimento delle proprie attività. Se si ha l'obiettivo di simulare il comportamento di tale soggetto è necessario prima formalizzarne i contenuti conoscitivi.

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

- ***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.
- ***Conoscenza tecnica,*** l'insieme di nozioni e di abilità sull'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.
- ***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. Sono 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 stimano all'altezza 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 per farlo deve riuscire a decodificare i meccanismi di agnizione all'"*old boys network*".
- ***Conoscenza creativa***, è un patrimonio esclusivamente personale di astrazioni e modalità di manipolazione che consente di generare dall'informazione una conoscenza non preesistente (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 conflitti.

Il sistema aziendale, quindi, deve usare questa 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 rispetto alla concorrenza e quindi la sopravvivenza.

La teoria dell'organizzazione è stata a lungo dominata da un paradigma nel quale l'organizzazione stessa viene concettualizzata come sistema che "elabora" informazioni o "risolve" problemi. Secondo tale paradigma, uno dei compiti fondamentali dell'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:

- Informazioni: la parte della comunicazione che riduce l'incertezza. È la "differenza che fa la differenza".
- Dati: diventano informazione quando sono significativi.
- 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.
- Apprendimento: l'abilità a identificare percorsi nel flusso delle informazioni ricevute, di dare ad esse un significato, di erborarle e conservarle come esperienza. L'apprendimento è l'abilità nell'assorbire, processare, salvare e adattare in pratica le informazioni.

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

1. I modelli letterari-descrittivi

2. I modelli matematico-statistici
3. I modelli realizzati con codice informatico, che forniscono gli strumenti per la simulazione.

Il terzo tipo di rappresentazione fornisce un modello 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 un scambio di informazioni che obbliga gli agenti del sistema a riflettere sulla struttura e sul contenuto delle informazioni 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 progetto di un modello di simulazione può condurre a risolvere o migliorare la problematiche insite nel sistema.

Partendo dai propri modelli interni un'organizzazione agisce nell'ambiente comparando le proprie esperienze con il modello stesso e impara arricchendo il modello con le esperienze ed altre fonti di informazione. Per raggiungere tale scopo 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. La conoscenza è qualcosa che si trova all'interno di tali relazioni. 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 politica 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.

La 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, esso è detto “Ciclo di Apprendimento”:

È un modello semplice per capire come avviene l’apprendimento. Esso è visto come un processo auto-organizzante, 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 Delfini 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 decompongono 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.

Capitolo 4 Il mercato delle auto usate: a market for lemons

4.1 Un modello sulle asimmetrie informative: a market for lemons

Nell'ambito degli studi economici sugli effetti delle asimmetrie informative esiste un interessante modello microeconomico che approfondisce i meccanismi attraverso i quali la selezione avversa, un particolare tipo di asimmetria informativa, agisce sugli scambi.

Lo studio teorico di Akerlof (1970) considera il mercato delle automobili. Ogni vettura venduta è caratterizzata da un prezzo p e la qualità media dei beni sul mercato è pari a μ. Da ciò si ottiene una domanda aggregata $Q^d = D(p, \mu)$. La qualità delle autovetture è funzione del prezzo, per cui si ottiene $\mu = \mu(p)$ e la domanda aggregata si esprime con $S = S(p)$. L'equilibrio di mercato si raggiunge quando

$$S(p) = D(p, \mu(p)). \tag{22}$$

Le conclusioni del modello sono ottenute attraverso la formalizzazione delle funzioni di utilità degli agenti, le quali sono lineari. Gli agenti scambiano massimizzando l'utilità attesa secondo il paradigma di von Neumann-Morgenstern.

In questo lavoro, la finalità consiste nel confronto tra il modello teorico tradizionale del mercato delle auto ed una metodologia orientata alla simulazione. A tal fine è utile considerare la trasposizione numerica del modello appena presentato elaborata da Varian (1987). Essa analizza un mercato formato ipoteticamente da 100 agenti disposti a vendere la propria automobile usata e 100 che desiderano acquistarne una. Il numero degli agenti è simbolico, poiché l'analisi avviene ad un livello microeconomico. Tutti gli agenti si comportano esattamente allo stesso modo e l'equilibrio del mercato si può determinare dall'analisi del singolo agente rappresentativo. Si ipotizza l'esistenza di due tipologie di autovettura, che si differenziano per la qualità. È possibile acquistare auto di buono o di scarso livello (*plum or lemons*). La probabilità di acquistare un'auto di buona qualità è del 50%. I proprietari di tali autovetture conoscono esattamente le caratteristiche qualitative della propria auto, mentre gli acquirenti ne hanno una conoscenza a livello aggregato. Conoscono, cioè, percentualmente la distribuzione qualitativa, ma non quella di ogni singola vettura. Essi non hanno preferenze relative al tipo di auto da acquistare ma vogliono pagare il giusto prezzo per ciò che comprano. Il mercato potenziale, dunque, non esclude a priori nessuna transazione. Tutti gli agenti dovrebbero essere soddisfatti dalla compravendita, se avvenisse ad un prezzo equo.

I venditori sono disposti a cedere le auto di buona qualità ad un prezzo non inferiore a \$2000 e quelle più scadenti a non meno di \$1000. Gli acquirenti fissano un prezzo massimo di \$2400 per le auto buone e \$1200 per quelle non buone.

In questo contesto si osserva che, in ipotesi di diretta misurabilità da parte dei compratori delle caratteristiche di ogni vettura, il mercato funzionerebbe perfettamente. Tutte le auto sarebbero scambiate ad un prezzo compreso tra $2400 e $2000 se di buona qualità e tra $1200 e $1000 nel caso opposto.

Lo studio dimostra che la mancanza di informazione condiziona il mercato, producendo un fenomeno paradossale: avvengono soltanto transazioni relative alle vetture più scadenti. In questa ipotesi si realizza un mercato di "bidoni" (*a market for lemons*), come è stato definito ironicamente nello studio condotto da Akerlof.

Come ciò avvenga si spiega analizzando il comportamento di entrambe le tipologie di agenti. Il venditore deve dichiarare il prezzo di vendita della vettura che possiede e conosce il difetto informativo del proprio interlocutore. Per questa ragione è libero di dichiarare un prezzo maggiore, al fine di approfittare della superiorità informativa. Il compratore, invece, deve decidere se accettare lo scambio. Poiché non conosce la qualità della vettura e ha interesse a pagare il prezzo che si avvicina di più al reale valore, non può fare altro che calcolare matematicamente tale importo. Esso è il valore atteso dei prezzi di tutte le auto, ovvero:

$$\mathrm{E}[P] = p_1 P_1 + p_2 P_2 \tag{23}$$

, dove p_1 è la probabilità di ottenere un'auto di buona qualità, p_2 è la probabilità relativa all'evento contrario e P_n è il prezzo che è disposto ad offrire l'acquirente in corrispondenza di uno dei due eventi. Da cui risulta:

$$0.5 * 2400 + 0.5 * 1200 = 1800.$$

Al prezzo di $1800, però, soltanto i possessori di auto scadenti avranno convenienza ad accettare lo scambio. In base a tale semplice principio il compratore è in grado di sapere che avrà la certezza matematica di ottenere soltanto un'auto di scarsa qualità, poiché nessun venditore cederebbe un'auto buona ad un prezzo inferiore a $2000. Inevitabilmente la probabilità p_1 corrisponde a 0 ed il valore atteso dei prezzi diventa $1200, il prezzo delle sole auto che verranno vendute.

Il modello dimostra il parziale fallimento del mercato a causa della presenza di informazioni non equamente distribuite. L'asimmetria informativa causa sistematicamente un peggioramento dell'equilibrio ottimo che si otterrebbe in sua assenza. In particolare, quando la situazione di sbilancio informativo tra i soggetti si manifesta prima che si perfezioni il contratto di scambio, si realizza una situazione di *adverse selection*, la quale provoca la diminuzione delle transazioni effettuate rispetto a quelle potenziali. Al limite ne può determinare il completo fallimento.

Qualora esistesse la possibilità di colmare il *gap* informativo sostenendo alcuni costi per reperire le informazioni non possedute, l'effetto della selezione avversa si concretizzerebbe in una componente aggiuntiva del prezzo di acquisto. Ciò comporterebbe la diminuzione del numero di

transazioni realizzabili, come si può osservare in fig. 3 dallo spostamento verso il basso della curva di domanda.

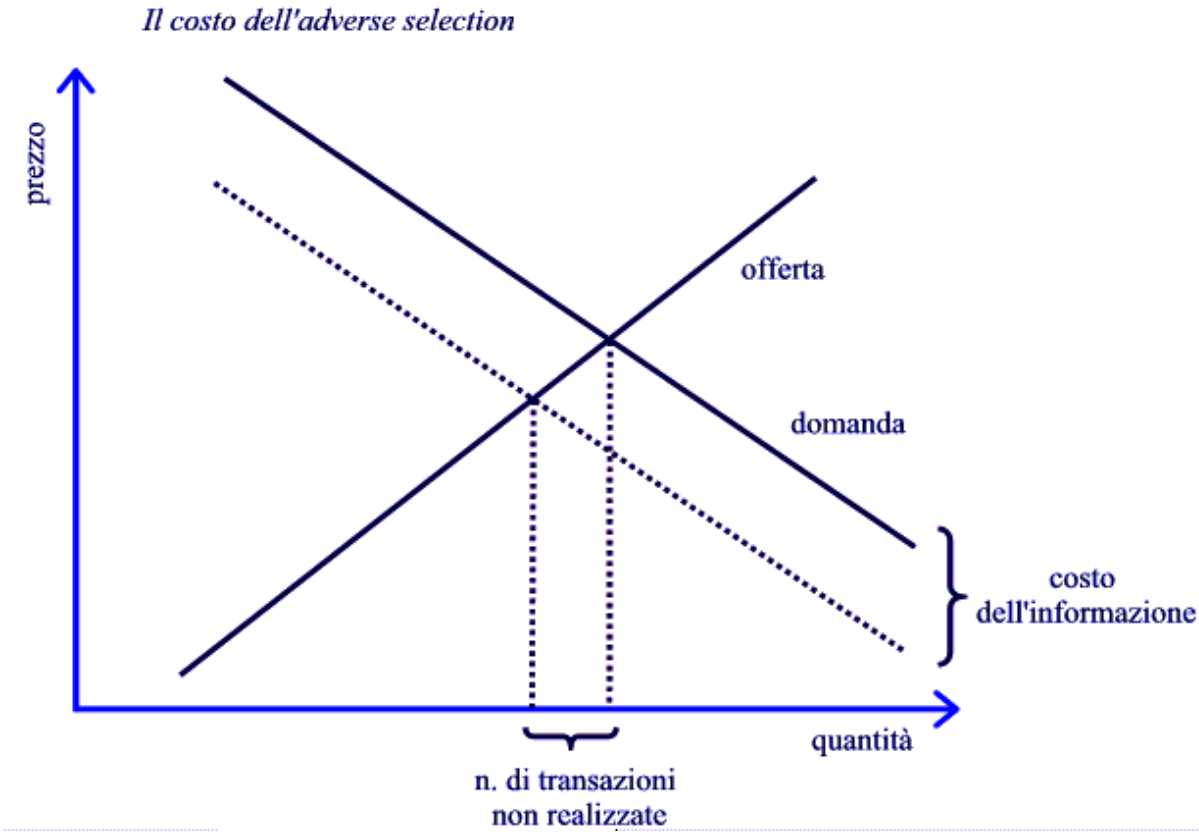

fig. 3 Perdita di mercato dovuta al fenomeno di *adverse selection*

Il modello di Akerlof spiega gli effetti di una distribuzione non equilibrata delle informazioni, ma non fornisce una giustificazione di come nella realtà i mercati funzionino ugualmente in presenza di difetto informativo. In particolare, si osserva come il mercato delle auto non sia un mercato di "bidoni", altrimenti fallirebbe.

Esistono varie ragioni legate a fenomeni che in questa semplificazione non vengono affrontati. Si deve ricordare che nella realtà il meccanismo psicologico che guida gli agenti può comprendere una componente di necessità che qui non è considerata. L'agente può essere disposto a rischiare l'acquisto di una vettura poiché ne ha una necessità oggettiva e quindi il meccanismo di scelta razionale viene abbandonato. Inoltre va considerato l'intervento di una terza tipologia di agente detto intermediario. Esso dispone di strutture e di conoscenze che gli permettono di realizzare economie di scala e di trasformare in un costo lo sbilancio di informazioni. L'intermediario offre garanzia di qualità relativa al bene che cede, poiché ha interesse ad affermare la propria credibilità. È consapevole di poter dichiarare un prezzo superiore ma il danno che gli provocherebbe la perdita di credibilità sulle prospettive future di reddito è certamente maggiore del surplus immediato che deriverebbe dall'utilizzo di una scorretta

informazione. La differenza tra il venditore privato e l'intermediario consiste proprio nella professionalità. Il venditore svolge lo scambio occasionalmente e pertanto è interessato a massimizzare il ricavo. L'intermediario ha obiettivi di medio-lungo termine e può rinunciare al massimo profitto.

Il servizio svolto dall'intermediario, però, si traduce in un costo: il diritto di intermediazione. Nonostante l'interpretazione del funzionamento del mercato è indubitabile che il disequilibrio delle informazioni peggiora l'equilibrio ottimo di mercato.

4.2 La simulazione al computer

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 conduce 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.

Nel caso specifico del modello appena descritto il metodo simulativo permette di osservare quale sia il processo dinamico che conduce all'equilibrio di mercato previsto da Akerlof. 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 necessario concentrare l'analisi in un ambito più circoscritto.

Nella simulazione con il computer 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 estremamente semplice aggiungere elementi che complicano il modello al fine di valutare l'impatto sull'intero sistema.

4.3 La trasposizione ad agenti del modello di Akerlof

Trasporre in termini informatici le premesse del modello descritto poc'anzi comporta la necessità di eliminare le ipotesi e le forzature che limitano la capacità di interazione tra gli agenti. In particolare, è necessario considerare i seguenti limiti nelle ipotesi sul mercato delle auto usate:

1. I prezzi limite sono uguali per tutti gli agenti compratori e per tutti i venditori sono scelti in modo da garantire l'esecuzione della transazione. Tale ipotesi è implausibile perché il prezzo è normalmente diverso da agente ad agente. Nella trasposizione informatica, infatti, si elabora una funzione che determina un prezzo diverso da agente ad agente sulla base delle diverse esperienze e di fattori casuali. Il principio di comportamento è lo stesso, ma l'ambiente osservato da ciascun agente non è percepito allo stesso modo.
2. Il compratore conosce la struttura dei prezzi dei venditori, perché è in grado di congetturare che al prezzo di $1800 verrebbero vendute solamente auto scadenti. Ma se il compratore conosce i prezzi del venditore non c'è alcuna ragione perché sia disposto ad offrire di più di $1000 e di $2000 per auto rispettivamente di scarsa e di buona qualità. Nel modello dinamico viene abbandonata questa forzatura e si dimostra il fallimento del mercato dall'osservazione del comportamento del sistema, non più dalle congetture del singolo agente. I compratori simulano un semplice comportamento istintivo. Decidono se accettare o meno il prezzo offertogli in base ad un meccanismo a tentativi ripetuti.
3. Non esiste un meccanismo che induca gli agenti a realizzare ugualmente la transazione poiché costretti dalla necessità. Nella realtà spesso le transazioni avvengono in situazione di asimmetria, perché al trascorrere del tempo ed in funzione dei fallimenti gli agenti correggono le proprie stime di prezzo. Inoltre, spesso le transazioni avvengono soltanto dopo che la parte ha valutato un buon campione di offerte alternative.
4. Nel modello di Akerlof viene ipotizzato un numero di agenti pari a 100. In realtà tale informazione è del tutto irrilevante perché ogni agente ha la medesima struttura interna e quindi la moltiplicazione di tali unità non provoca variazioni nel sistema. Nel modello informatico ogni agente ha variabili di stato diversificate in base a numeri casuali ed a meccanismi di correzione fondati sull'esperienza di ciascuno.
5. L'analisi del modello di riferimento deduce il risultato generale dalla formalizzazione del comportamento dell'individuo rappresentativo. Nel modello informatico l'equilibrio non è perfetto ed univocamente determinato, ma dipende dal caso, dalla variabilità degli agenti, nonché dal loro numero.

6. Non è specificato il meccanismo che determina il prezzo della transazione. Se, infatti, i compratori sono disposti a scambiare ad un prezzo massimo di $2400 e i venditori ad un prezzo minimo di $2000, non è spiegato quale sia il prezzo di vendita. Viene qui introdotto il principio per cui il prezzo di scambio è pari al prezzo dell'offerta.

Tenendo presenti tali differenze, il modello ad agenti è costituito da 100 agenti che posseggono un'autovettura ed hanno intenzione di venderla, proprio come nel modello di riferimento. Si contemplano 100 agenti che non posseggono l'autovettura ed hanno intenzione di acquistarne una. Sono consapevoli della possibilità di acquistare una vettura soddisfacente oppure un "bidone" e conoscono anche la probabilità del 50% di effettuare un acquisto insoddisfacente. Negli elementi nei quali è possibile mantenere le premesse di Akerlof si adottano gli stessi parametri.

Il venditore propone un prezzo di vendita che è funzione della qualità dell'auto che possiede, includendo però una componente di sincerità diversa da agente ad agente.

Il compratore congettura il prezzo come valore casuale nella fase iniziale, in cui si ipotizza sia inesperto. Nelle fasi successive il prezzo è funzione delle esperienze trascorse. È calcolato un valore di disagio/soddisfazione post-acquisto che modifica la tendenza di fiducia dell'acquirente.

La dinamica del modello è basata su un ciclo temporale unico, ripetuto all'infinito. Ad ognuno di questi cicli ogni agente incontra un agente con intenzioni opposte.

Il prezzo è stabilito dal mercato.

4.4 Il meta-modello

Prima di utilizzare qualsiasi strumento software per realizzare un modello informatico di simulazione ad agenti, è importante costruire un meta-modello (Goldspink 2000), ovvero una descrizione formale degli agenti attori del modello, delle variabili che caratterizzano ciascun agente, delle regole che ne consentono la variazione, dei messaggi ai quali un agente deve saper rispondere e della lista di azioni che devono essere compiute in ogni ciclo temporale di esecuzione. Tale meta-modello fornisce una descrizione comprensibile a chiunque e permette di seguire un processo rigoroso nella scrittura del codice informatico, nel quale spesso è necessario ricorrere a compromessi per realizzare quanto ci si era proposti.

Il passo successivo consiste proprio nella elaborazione del codice informatico che produce la simulazione, attraverso l'uso di uno strumento software appropriato. Le regole dell'ambiente di sviluppo sono molto importanti perché realizzano uno strato attraverso il quale la descrizione informatica del modello è trasformata in una metrica compatibile con l'architettura di funzionamento del computer. Occorre, cioè, scrivere il software tenendo presente che si deve far eseguire una simulazione di una realtà intrinsecamente parallela e caratterizzata dalla continuità

del tempo ad uno strumento la cui struttura prevede la scansione del tempo discontinua e soprattutto un'elaborazione strettamente sequenziale delle operazioni.

Il primo fondamentale aspetto del modello è costituito dalle variabili del sistema:

- Il **numero degli agenti.** Nel modello simulato ciascun agente possiede variabili di stato che ne descrivono le caratteristiche e che si modificano in base alle sue esperienze. In generale, dunque, la quantità di agenti può essere un fattore rilevante per interpretare i risultati. Nel caso specifico ci si aspetta che al variare del numero di agenti il comportamento complessivo del modello sia costante a causa della semplicità del sistema.
- Le variabili **envSincerityMin** ed **envSincerityMax** sono i valori soglia entro i quali devono essere generate casualmente le determinazioni del coefficiente di sincerità di ciascun agente. Se si ipotizza, ad esempio, che tali limiti siano [1,1], si determina un sistema in cui tutti gli agenti sono totalmente onesti e dichiarano sempre la reale qualità della vettura. Una coppia di valori di [1,1.5] indica invece che mediamente gli agenti venditori dichiarano un valore un po' più alto di quello effettivo.
- Le variabili **envFaithMin** e **envFaithMax** sono i limiti del supporto di generazione dei numeri casuali relativi alla fiducia di ciascun compratore. Nell'ipotesi di valori [1,1] si impone che nel mercato avvengano tutte le transazioni potenziali, poiché i compratori avranno sempre la massima fiducia nella qualità dichiarata dai venditori e poiché il compratore non ha preferenze rispetto all'auto da acquistare, sarà sempre disponibile allo scambio.
- La variabile **envBasisPrice** costituisce il prezzo base di una vettura scadente. Tale informazione è utile ad imporre un limite nella caduta dei prezzi, sulla base della considerazione che i compratori nel peggiore dei casi non possono ricevere una vettura che vale meno di una vettura scadente. Se si eliminasse questo limite inferiore alla discesa dei prezzi, potrebbe verificarsi il completo fallimento del mercato. Ciò sarebbe plausibile, ma nel modello di riferimento si impone che nonostante il venditore possa essere disonesto la qualità intrinseca del bene venduto non può essere inferiore ad un valore che corrisponde al prezzo minimo.
- **envFaithCorrection** e **envMissingCorrection** sono i parametri relativi al meccanismo di correzione delle congetture degli agenti. Il primo parametro è utilizzato dal compratore per correggere la sua stima della fiducia nel sistema in base all'esperienza accumulata. Il secondo parametro agisce su entrambe le tipologie di agenti variando il prezzo congetturato in funzione del numero di cicli nei quali l'agente non è riuscito a scambiare. Quest'ultimo tende a far convergere domanda e offerta, ma non produce effetti sull'interpretazione del risultato di equilibrio.

Il meta-modello di Akerlof prevede solamente due tipologie di agenti:

- L'***agente venditore***, il quale possiede le seguenti variabili di stato:

1. Il **coefficiente di sincerità**, caratteristico di ciascuno. Esso viene assegnato in modo casuale in fase di creazione. Deve essere necessariamente incluso nell'intervallo [*envSincerityMin, envSincerityMax*]. Rappresenta il livello di sincerità del venditore. Proprio come avviene nella realtà, spesso il venditore cerca di nobilitare le virtù del prodotto conscio del *gap* informativo che lo separa dal compratore. Tale coefficiente viene moltiplicato alla qualità della vettura per ottenere il livello qualitativo che l'agente comunica al suo interlocutore.
2. Il **prezzo** congetturato dal venditore relativamente alla qualità dell'auto che possiede nel ciclo corrente di simulazione, in funzione della sua sincerità.
3. La **qualità** dell'auto che possiede. Essa è generata casualmente ad ogni ciclo di simulazione tra i valori {1,2}. Quando la qualità vale 1 corrisponde alla tipologia di auto di scarsa qualità. Nel secondo caso, il valore 2 sta a significare che l'auto ha un valore qualitativo doppio rispetto alla prima. Ciò non è perfettamente plausibile, poiché nel mondo reale le auto possono avere infiniti gradi di qualità. Al fine di poter esprimere un corretto confronto con il modello di riferimento si preferisce mantenere il più possibile le ipotesi che non comportano un'influenza significativa sul risultato. La potenza della simulazione ad agenti consiste proprio nell'estrema facilità ad introdurre l'ipotesi di continuità di questa variabile, nel momento in cui la si ritenesse significativa ai fini del risultato.
4. Il **numero di cicli di astinenza da transazione** è un banale contatore del tempo in cui l'agente non è riuscito a scambiare. Tale variabile è usata come coefficiente di costo nella funzione di congettura del prezzo. L'idea è che al passare del tempo accresca il desiderio di scambiare dell'agente, il quale è quindi disposto a diminuire le sue pretese. Questa ipotesi di plausibilità è stata introdotta rispetto al modello teorico di riferimento per garantire che domanda ed offerta si incontrino. In sua assenza si potrebbe verificare un'immobilità del sistema, con il completo fallimento.
5. Il **prezzo della transazione** è il prezzo al quale è stato eseguito lo scambio. Se lo scambio non è avvenuto esso vale 0. Il prezzo della transazione è una variabile critica, poiché nella realtà si realizza un complesso meccanismo psicologico negli agenti, che cercano di non scoprire le proprie congetture sui prezzi limite al fine di spuntare le condizioni migliori. In questo mercato il venditore fa un'offerta di vendita ed il compratore si limita ad accettarla o meno, pertanto il prezzo della transazione è sempre il prezzo espresso dal venditore.

- L'***agente compratore*** possiede le seguenti variabili di stato:

1. Il **coefficiente di fiducia**, caratteristico di ciascun agente. Esso viene assegnato il modo casuale in fase di creazione. Deve essere necessariamente incluso nell'intervallo

[*envFaithMin, envFaithMax*]. Tale coefficiente rappresenta il livello di fiducia del compratore nei confronti del mercato dei venditori. Nella realtà l'idea di fiducia è legata alla percezione che il compratore ha del singolo venditore, è meno importante la fiducia generica dell'insieme dei venditori. Realizzare tale soluzione dal punto di vista informatico sarebbe stato complesso e poco significativo ai fini dell'analisi sugli effetti dell'asimmetria informativa. In questo modello la fiducia varia in funzione delle esperienze che il compratore fa nel corso dei cicli di simulazione.

2. Il **prezzo** congetturato dal compratore relativamente alla qualità dell'auto che intende comprare nel ciclo corrente di simulazione. Esso dipende dalla qualità che il venditore promette e dal livello di fiducia che il compratore ha nei suoi confronti.
3. La **qualità comunicata** dal venditore circa l'auto in vendita. Essa è comunicata dal venditore in funzione della qualità reale e del suo grado di sincerità. Può assumere infiniti valori sul supporto [1,2]. In questo caso la variabile è continua poiché frutto del prodotto tra la qualità reale e il coefficiente di sincerità (continuo per definizione). Si può considerare un difetto di costruzione del modello poiché non omogeneo con il valore di riferimento, ma anche in questo caso tale caratteristica non influisce sulla qualità dei risultati. Inoltre è possibile renderlo intero con un procedimento di approssimazione.
4. La **qualità reale** dell'auto in vendita. Questa informazione è disponibile solamente nel momento in cui viene perfezionato lo scambio. In caso di mancata transazione il compratore non saprà mai quale fosse realmente la qualità dell'auto. Sulla base di questa informazione l'agente modifica il proprio livello di fiducia nel mercato di una quantità pari al coefficiente [*envFaithCorrection*], in positivo se la qualità reale è superiore a quella comunicata, in senso opposto nell'altro caso.
5. Il **numero di cicli di astinenza da transazione.** Ha lo stesso significato dell'analoga variabile del venditore. In questo caso l'astinenza agisce incrementando il prezzo congetturato, nel primo caso lo diminuisce.
6. Il **prezzo comunicato** è il prezzo al quale il venditore è disposto a vendere.
7. Il **prezzo della transazione** è il prezzo al quale è stato eseguito lo scambio. Se lo scambio non è avvenuto questo prezzo è 0. È uguale al prezzo di transazione dell'agente venditore.

Le variabili degli agenti vengono modificate nel corso della simulazione in base alle seguenti regole:

- Il prezzo del venditore è calcolato in base alla formula:

  Prezzo= envBasisPrice qualitàCorrente* sincerità- envMissingCorrection* astinenza*

- Il prezzo del compratore è calcolato in base alla formula:

  Prezzo= envBasisPrice qualitàComunicata * fiducia + envMissingCorrection* astinenza*

- La qualità comunicata dal venditore è calcolata con:

$$qualitàComunicata = qualitàCorrente * sincerità$$

- La fiducia del compratore viene così modificata:

$$fiducia = fiducia – (qualitàComunicata - qualitàReale) * envFaithCorrection$$

La dinamica temporale delle azioni svolte in un ciclo di simulazione è la seguente:

1. Predisposizione di tutte le variabili di stato degli agenti.
2. Tutti gli agenti venditori effettuano la congettura sul prezzo in base al valore corrente di qualità.
3. Ogni venditore contatta il compratore attraverso un meccanismo di *grabbing*, che consiste nella cattura di un agente che non sia stato contattato da altri venditori. Dal punto di vista teorico dovrebbe essere garantito un contatto tra ogni venditore con un diverso compratore ad ogni ciclo. Come si vedrà ciò non è garantito nella rappresentazione informatica, ma il fenomeno si può considerare solamente se si sceglie un coefficiente *envMissingCorrection* elevato.
4. Il venditore, dopo aver catturato, un compratore, comunica a quest'ultimo due informazioni: il prezzo al quale è disposto a vendere e la qualità della vettura in vendita. Nel modello informatico è trasferita anche l'informazione relativa alla reale qualità della vettura, informazione che però non è utilizzata dall'agente fino a che non si sia compiuta la transazione.
5. A questo punto il compratore possiede le informazioni per realizzare una congettura sul prezzo al quale è disposto ad acquistare.
6. Se il prezzo proposto dal venditore è inferiore a quello al quale il compratore è disposto ad acquistare, la transazione viene perfezionata al prezzo del compratore.
7. Tutti gli agenti svolgono le operazioni di riflessione sull'esperienza relativa al ciclo corrente. In particolare, i compratori stimano l'effetto di soddisfazione/insoddisfazione post-acquisto per correggere la propria stima di fiducia.
8. In questo punto sono collocate le procedure relative ai calcoli statistici da mostrare visivamente all'osservatore del modello. Esse sono dette procedure di *probing*.

4.5 Il software

La rappresentazione del meta-modello fornisce una descrizione che trascende dal piano puramente tecnico-informatico. Esso fornisce la rappresentazione concettuale degli agenti e l'analisi degli algoritmi che determinano il comportamento di questi nell'ambito della simulazione. Sono elencate in forma dettagliata tutte le operazioni che devono essere svolte all'interno di ciascun ciclo di funzionamento dell'esperimento. Il meta-modello fornisce al

progettista del software tutte le informazioni necessarie a realizzare il codice informatico che esegue la simulazione.

Come è stato accennato precedentemente, il modello è progettato per essere eseguito in forma sequenziale. Ogni operazione deve essere svolta integralmente prima che cominci l'esecuzione della successiva. La realtà a cui il modello si ispira non si svolge in sequenza, poiché ogni entità che opera nel sistema è indipendente rispetto al tempo. Le operazioni che implicano l'interazione tra gli agenti avvengono spesso in sequenza, ma la totalità degli attori del sistema non usa come riferimento di tempo l'unità detta ciclo, all'interno della quale si svolge un preciso numero di operazioni. Nel modello informatico tutto ciò è assolutamente schematizzato. Tale differenza deve essere tenuta presente durante l'analisi della simulazione.

La rappresentazione sequenziale del problema rende il modello potenzialmente compatibile con il calcolatore, il quale ha il compito di svolgere l'esperimento simulativo. Al fine di realizzare compiutamente tale esperimento è necessario costruire il software. Il computer ed il sistema operativo forniscono l'infrastruttura e le funzioni di calcolo numerico, ma è necessario produrre il codice informatico che governi lo svolgimento delle operazioni.

Il metodo che consente di realizzare ciò non è unicamente determinato. Esiste una metodo più diretto e potente: la scrittura attraverso un linguaggio di programmazione di tutto il codice che regola lo scatenarsi degli eventi, la coordinazione tra gli oggetti del sistema, gli algoritmi di funzionamento degli agenti. Un qualsiasi linguaggio di programmazione, sia esso imperativo o dichiarativo, rappresenta certamente un ottimo strumento, ma allo stesso tempo una strada impervia da percorrere. Il programmatore deve tenere sotto controllo molti fattori di carattere tecnico, legati cioè alle caratteristiche del sistema operativo e non direttamente attinenti al modello. Ogni aspetto relativo alla gestione delle liste di agenti, delle variabili di stato che regolano il trascorrere del tempo, dei meccanismi che scatenano degli eventi deve essere scritto dal programmatore, sottoponendosi al rischio di introdurre *bugs* nel codice.

Deve essere tenuto presente un ulteriore aspetto negativo. La finalità della simulazione sociale consiste anche della sua capacità di convalidare la teoria alla quale il modello fa riferimento. Un importante proprietà di tutti gli studi di carattere scientifico è una forma espressiva universale che consenta all'osservatore di comprendere ed eventualmente confutare il lavoro. L'uso di un linguaggio di programmazione per realizzare un modello di simulazione totalmente ideato dall'autore impone a chi osserva di conoscere il linguaggio di programmazione e di analizzare la validità delle soluzioni informatiche. Tale premessa deve essere evitata al fine di rendere universalmente condivisibile il lavoro.

Il metodo che si adotta in alternativa al primo porta alla realizzazione di quanto ci si è proposti attraverso l'uso di software finalizzati alla simulazione. Essi forniscono alcuni strumenti e facilitazioni per realizzare il modello, permettendo di esimersi dall'affrontare problematiche tecniche che si discostano dai proponimenti di analisi dell'autore. Questa seconda opportunità

appare indubbiamente la più idonea. Esiste però l'altro lato della medaglia. I modelli di simulazione in ambito sociale devono poter affrontare problematiche profondamente diversificate. Si affrontano spesso problemi di carattere teorico che richiedono un ambito di osservazione molto ampio, come ad esempio un mercato in concorrenza. Parimenti si analizzano frequentemente realtà molto più circoscritte ed approfondite. Si può, ad esempio, mirare l'osservazione su un particolare processo che regola il sistema (il meccanismo dell'asimmetria informativa). Per queste ragioni è impensabile che un software che renda semplice la creazione del modello ad agenti possa consentire una flessibilità tale da consentire l'analisi di ogni tipo di modello dinamico.

È importante, dunque, conoscere gli strumenti per comprendere quale sia il più idoneo ad affrontare uno specifico obiettivo. In questo lavoro sono confrontati due strumenti profondamente diversi fra loro, al fine di dimostrare come la realizzazione di modelli sia possibile in differenti modalità e con differenti risultati.

4.6 Dettagli implementativi in Starlogo

Starlogo è un software accademico realizzato nell'ambito di un progetto del MIT di Boston. È un ambiente specifico per la simulazione di modelli ad agenti, in cui il concetto di spazio è un fattore importante. Nasce come evoluzione del celebre linguaggio di programmazione denominato *Logo*. Esso è un dialetto del *Lisp*, un noto linguaggio dichiarativo, e fu sviluppato da Feurzeig (1967) con l'obiettivo di consentire ai giovani di conoscere i rudimenti della programmazione informatica. È caratterizzato da una superficie toroidale proiettata su di un piano bidimensionale. Ciò significa che lo schermo rettangolare non ha idealmente confini: i due lati opposti sono congiunti. Su tale superficie è disegnata una "tartaruga" che è capace di interpretare ed eseguire comandi di movimento, ha una direzione e la capacità di lasciare un segno sulla superficie che attraversa se riceve il comando opportuno. Quando giunge al confine dello schermo ricompare nella posizione simmetrica sul lato opposto, per effetto dell'ideale continuità dello spazio.

I progettisti di *Starlogo* hanno mutuato due principi da questo linguaggio antenato. L'idea che li ha spinti a scriverlo è stato quella di realizzare uno strumento che permettesse di avvicinare all'universo della simulazione informatica di modelli sociali tutti coloro che non avessero dimestichezza con la programmazione. L'altra idea è stata di moltiplicare il concetto semplice di "tartaruga". È sufficiente creare più di uno di questi elementi sulla superficie toroidale e di dotarlo di capacità elementari di comunicazione per ottenere un potente strumento di analisi dinamica dei sistemi. *Starlogo* conserva quasi integralmente il bagaglio di istruzioni del *Logo* relative alla gestione delle tartarughe ed aggiunge una serie si strumenti e funzioni per controllare l'interazione tra queste. Resnick (1994) descrive *Starlogo* in questi termini:

> Starlogo is a programmable modeling environment for exploring the workings of decentralized systems – systems that are organized without an organizer, coordinated without a coordinator. With Starlogo, you can model (and gain insights into) many real-life phenomena, such as bird flocks, traffic jams, and market economies.

Come accennato, questo strumento è fortemente legato all'idea di spazio. Le tartarughe sono localizzate sulla superficie toroidale e la loro interazione avviene quando si incontrano nella stessa posizione spaziale. In realtà esiste un altro metodo di interazione non basato sullo spazio. É un *set* di istruzioni che permette di gestire le liste di oggetti con una sintassi ereditata dal *Lisp*. Ma il loro uso comporta alcuni problemi. Partendo dalla constatazione che non è fornito l'intero complesso di istruzioni del *Lisp*, se l'obiettivo di *Starlogo* è avvicinare chi non ha dimestichezza con la programmazione, l'uso di istruzioni ereditate da un linguaggio dichiarativo allontana da tale obiettivo.

Inoltre, nell'ipotesi di conoscere la programmazione e quindi di saper manipolare opportunamente tali liste va tenuto in conto il problema che nasce dalla caratteristica di architettura parallela dell'ambiente. Ogni agente viene eseguito in un processo (*thread*) isolato ed il processore del computer simula l'esecuzione non sequenziale delle operazioni. Ciò implica che se un agente invia un messaggio ad un altro agente, questo risponde, ma nel frattempo l'esecuzione del codice del chiamante viene proseguita. Il fenomeno rende assai complessa la gestione delle comunicazioni tra "tartarughe". Il meccanismo che consente di superare tale limite sta nello sfruttare la possibilità di programmare la superficie, detta *patch*, la quale è in grado di contenere informazioni ed eseguire operazioni. Con tale stratagemma è possibile ordinare l'esecuzione di un comando ad una categoria di agenti, i quali "depositano" le informazioni sulla superficie in cui si trovano e la successiva esecuzione di un'altra categoria di agenti, i quali possono leggere tali informazioni. Questi due gruppi di operazioni avvengono in sequenza poiché la loro esecuzione è ordinata dalle istruzioni dell'*observer*, il controllore del sistema. Questo meccanismo evita il contatto diretto degli agenti e consente di rendere sequenziale e, quindi, più controllabile l'esecuzione del modello.

Starlogo è stato originariamente scritto per essere eseguito nell'ambiente *Apple Macintosh*, ma ciò ne ha limitato la diffusione, poiché tale sistema è poco utilizzato. Recentemente è stata rilasciata una versione scritta con il linguaggio *Java* e che pertanto può essere eseguita su qualsiasi computer dotato di una *Java Virtual Machine*. Questo elemento ha notevolmente aumentato il numero di utilizzatori dello strumento. Data la sua recente diffusione non sono state ancora pubblicate applicazioni in campo economico.

La realizzazione del modello di Akerlof in *Starlogo* ha richiesto un piccolo sforzo di adattamento degli elementi descritti dal meta-modello. Come è già stato rilevato, nei modelli nei quali gli agenti posseggono caratteristiche legate allo spazio lo strumento facilita enormemente il lavoro del programmatore. Soprattutto se si effettua il confronto con la mole di lavoro che l'utilizzo di un linguaggio di programmazione tradizionale richiederebbe nel realizzare il codice

che controlla il movimento degli agenti e la rappresentazione grafica dello spazio. Qualunque linguaggio comporterebbe un tempo di lavoro molto più grande. Nel caso specifico però il modello delle auto usate non prende in considerazione la collocazione spaziale degli agenti: l'incontro tra questi avviene in modo casuale. Al fine di rendere tale soluzione in *Starlogo* si è dovuto utilizzare lo stratagemma sopra descritto. Gli agenti compratori si muovono nella posizione di un venditore. Questo comunica alla superficie sottostante il prezzo di vendita e la qualità dell'auto. Nella fase successiva il compratore legge i dati dalla *patch* e decide se realizzare la transazione. In caso affermativo colloca l'informazione nella variabile apposita del "terreno", affinché nel passo successivo il compratore possa essere informato dell'avvenuta transazione.

Nella programmazione ad oggetti esiste l'opportunità di creare un componente software che contenga i metodi e le variabili capaci di determinarne il comportamento. Lo strumento in esame, purtroppo, non fornisce la possibilità di programmare ad oggetti, ma consente di separare il codice in due macro sezioni. L'*observer* è il luogo in cui vengono scritte tutte le funzioni che si occupano di regolare lo svolgimento delle operazioni da un punto di vista aggregato. Esso è in grado di notificare uno o più comandi ad un insieme di agenti. Ad esempio, possono essere mandati comandi a tutte le "tartarughe" che si trovano all'interno di un'area, che hanno una variabile di stato con determinate caratteristiche o che appartengano ad una categoria detta *breed*.

Il codice che viene eseguito dal singolo agente va collocato all'interno della sezione ad esso relativa (*turtle procedures*). Purtroppo il codice è relativo a tutti gli agenti, indipendentemente dalla categoria alla quale appartengono. Nel modello di Akerlof esistono due tipologie di agenti: i compratori e i venditori. Sarebbe molto comodo separare il codice relativo ai due tipi, invece la scrittura del codice comporta l'inevitabile commistione tra le funzionalità dei compratori e quelle dei venditori. Per questa ragione nel codice le variabili di stato delle due categorie di agenti sono elencate congiuntamente. È compito del programmatore usare le une o le altre a seconda del *breed* dell'agente che sta eseguendo il codice. Va ricordato che anche i metodi relativi al funzionamento delle celle che compongono la superficie devono essere collocati nella sezione delle "tartarughe", aumentando inevitabilmente il livello di confusione.

L'ambiente consente quattro metodi per mostrare i risultati all'osservatore. Da questo punto di vista chi realizza il modello è enormemente facilitato rispetto all'uso di altri strumenti.

Il primo fondamentale canale che consente di mostrare il funzionamento è lo schermo sul quale interagiscono le "tartarughe". Esse possono assumere svariati colori e possono determinare la colorazione delle stesse *patch*. Tale soluzione ha un valore decorativo e talvolta anche ludico, ma può anche essere un potente strumento di analisi della dinamica del modello durante la sua esecuzione.

Esiste un'altra finestra che mostra i risultati in forma grafica. La *plot window* è in grado di disegnare grafici funzionali dei risultati con l'uso di semplici istruzioni.

I risultati in forma testuale sono mostrati in una finestra di terminale, attraverso l'invio di istruzioni di stampa delle informazioni o attraverso sonde che mostrano il valore corrente di una variabile di stato in ogni istante di tempo.

4.7 Dettagli implementativi in Mathematica

Mathematica è un'applicazione molto nota in campo scientifico per la sua grande potenza di calcolo e l'indubbia versatilità. Il nucleo del programma è un potente elaboratore che è in grado di eseguire calcoli con precisione decimale infinita e, soprattutto, è in grado di realizzare calcolo simbolico. *Mathematica* sceglie automaticamente il migliore algoritmo per risolvere il problema e lo applica in modo adattivo. Al contrario di tutti gli altri software sul mercato esso può eseguire i calcoli con una precisione decimale infinita. I suoi autori sostengono che:

> We've designed Mathematica to be useful throughout as much of a technical person's workday as possible, not just for one or two tasks. Our customers find that using one technical computing system, fully integrated between different aspects, is many times more efficient than using several specialized systems. Mathematica is unique in having taken this approach to assisting technical professionals.

In seguito al successo riscontrato nel mondo scientifico, questo software è stato sviluppato fino a diventare un vero e proprio linguaggio scientifico di programmazione. Esistono migliaia di librerie e di esempi che ne arricchiscono le potenzialità.

Da una prima osservazione non sembra essere uno strumento adatto alla simulazione, ma grazie all'idea di Maeder (1993) è stata dotata di una libreria che consente di sviluppare codice ad oggetti. Tale metodo fornisce gli elementi per realizzare un modello di simulazione con un software nato per rispondere ad esigenze totalmente differenti. L'idea di programmare *Mathematica* come ambiente di simulazione è sicuramente una sfida, una curiosità, ma presenta anche alcuni elementi di interesse. Infatti, in tutti i modelli di simulazione nei quali è necessario realizzare il codice che svolga pesanti operazioni di carattere algebrico-matematico, lo strumento fornisce un indubbio vantaggio rispetto a strumenti più tradizionali. Ad esempio, il prodotto di due matrici si realizza in *Mathematica* attraverso una semplice istruzione, mentre richiede diverse righe di codice iterativo in tutti gli altri linguaggi.

Va ricordato un interessante sviluppo di questa idea. Recentemente è stata fornita un'interfaccia nel linguaggio *Java*, *J/Link*, che permette di comunicare con il nucleo di *Mathematica* e utilizzarlo per svolgere i calcoli nei quali è "esperto". Attraverso questo metodo si può realizzare un modello di simulazione in un ambiente più idoneo, senza dover rinunciare alle peculiarità di questo strumento.

Mathematica rappresenta un punto di riferimento tra i software scientifici e pertanto è utile indagarne le capacità nell'ambito della simulazione sociale.

Il modello di Akerlof è semplice poiché contiene pochi oggetti e questi svolgono operazioni limitate. La realizzazione in M*athematica* è risultata molto lineare, ma al complicarsi del modello probabilmente la realizzazione diverrebbe troppo complessa. Tale affermazione è confermata dal fatto che il modello di programmazione ad oggetti non è integrato nel codice del software, ma è ottenuto con un parziale mascheramento. Il risultato è che le variabili di stato ed i nomi dei metodi degli oggetti non sono realmente incapsulati al loro interno, ma l'ambiente li tratta come elementi globali e quindi è necessario diversificare i nomi delle funzioni in modo da non incappare in un fenomeno di sovrapposizione.

Questo ambiente è strutturato come un grande foglio sul quale vengono collocati in sequenza il codice ed i risultati delle elaborazioni del nucleo. Non esistono strumenti di visualizzazione esterni, pertanto i risultati di una simulazione sono inevitabilmente presentati in successione. Quando la simulazione richiede numerosi cicli di esecuzioni ciò comporta inevitabili problemi di gestione grafica dei risultati.

Un elemento che diversamente da altri strumenti lo rende funzionale è la grande potenza nell'uso del concetto di lista che permette all'utilizzatore di manipolare con estrema potenza i dati raccolti durante la simulazione. Infatti, anche *Mathematica* eredita molte istruzioni dal *Lisp*. Inoltre, si può osservare il modello, disegnarne i risultati e continuare a trasformarli simbolicamente senza perdere precisione di calcolo.

Mathematica ha la capacità di salvare il lavoro in differenti formati: HTML, TeX, RTF. È capace inoltre di riconoscere ed utilizzare il formato MathML, il nuovo standard creato dal consorzio W3C per il *web* che permette di presentare le espressioni matematiche in un formato universale per il trasferimento attraverso l'internet.

4.8 Confronto tra i due ambienti.

Si è dimostrato che a partire da una descrizione nella forma di meta-modello informatico del problema possono essere sviluppati diversi esempi realizzativi. Con riferimento al modello del mercato di auto usate i due strumenti utilizzati si sono dimostrati idonei alla produzione di una simulazione economica di carattere teorico. Il confronto non si risolve del tutto a favore di uno dei due strumenti, poiché entrambi possiedono caratteristiche interessanti, non riscontrabili nell'altro ambiente.

Starlogo è stato progettato per affrontare lo specifico tema della simulazione, ma si propone di essere semplice. Spesso la semplicità si traduce in scarsa disponibilità di strumenti, ma allo stesso tempo possiede la dote di rappresentare molto efficacemente l'evoluzione dinamica del sistema. Attraverso la collocazione spaziale degli agenti e la possibilità di colorarli, si osserva con un colpo d'occhio ciò che sta accadendo. Gli agenti compratori, che normalmente sono rappresentati dal colore blu, assumono una tonalità giallastra quando hanno acquistato la vettura.

Questa caratteristica di analisi grafica non rende soltanto piacevole l'esecuzione del modello, ma consente di osservare fenomeni che non sono previsti a priori. Attraverso tale indagine lo sperimentatore è in grado di variare il modello sulla base dei fenomeni osservati.

Mathematica, al contrario del suo antagonista, non è stato pensato come strumento di analisi dinamica dei sistemi sociali. Essendo dotato di un linguaggio di programmazione consente di essere utilizzato per qualunque scopo. Rispetto ad un linguaggio tradizionale, però, mette a disposizione funzioni di calcolo e strumenti di visualizzazione grafica molto semplici da usare e potenti allo stesso tempo. Inoltre, la estrema potenza nella gestione delle liste permette di conservare i risultati prodotti durante l'esecuzione del modello e di elaborarli successivamente. *Starlogo* non ha la capacità di memorizzare la serie storica delle variabili osservate dal modello. Questa funzionalità deve essere interamente costruita e gestita dall'utente.

Un altro elemento di differenza da esaminare è la capacità di programmazione dell'interfaccia grafica che permette all'utente di controllare l'esecuzione della simulazione e di modificarne i parametri. *Starlogo* fornisce un metodo assolutamente semplice per disegnare bottoni e oggetti grafici che permettono di variare i parametri (detti *slider*). In modo assolutamente rapido ed intuitivo è possibile rendere il modello utilizzabile anche da chi non conosce il codice. In *Matematica* questa opportunità non esiste. L'uso del modello, però, non è difficile, poiché l'utente deve eseguire un semplice comando per creare il mondo degli agenti (*setupModel*) ed un altro per eseguirlo (*runModel*). A queste istruzioni sono passati tutti i parametri necessari. L'impressione generale è comunque che nel secondo caso l'osservatore deve avere un minimo di dimestichezza con la sintassi del programma, mentre nel primo tale capacità non è necessaria. Infatti, con *Starlogo*, è possibile eseguire tutti i comandi attraverso la pressione degli appositi pulsanti, si possono variare i parametri utilizzando le *slider*, ed è possibile indagare e modificare la struttura di un singolo agente effettuando un doppio *click* con il *mouse* in corrispondenza della sua posizione. È possibile creare e distruggere le "tartarughe" sempre con l'uso dei comandi interattivi.

Un'altra differenza consiste nella durata dell'esperimento. *Matematica* richiede che lo sperimentatore indichi a priori quanti periodi di simulazione devono essere utilizzati, poiché non ha la capacità di interazione con l'utente durante il calcolo. Starlogo sfrutta il concetto di comando *forever*. Ogni operazione è scritta in modo atomico, l'intero ciclo di simulazione è una funzione che può essere eseguita singolarmente. Attraverso la proprietà *forever* del pulsante che attiva la funzione l'ambiente la esegue fino a che l'utente non comanda l'interruzione della stessa. Ciò migliora la capacità di indagare i fenomeni, potendo interrompere, effettuare modifiche e poi riprendere l'esecuzione più volte senza dover ripartire dal punto iniziale.

4.9 Risultato finale

Il modello è stato eseguito in entrambi gli ambienti ed il risultato fornisce un'interpretazione univoca: le conclusioni sugli studi teorici del mercato delle auto usate sono stati confermati anche dalla simulazione informatica. Il risultato del prezzo di equilibrio e della quantità di scambi realizzati è differente in valore assoluto. Mentre nel modello teorico, il prezzo finale in presenza di selezione avversa è esattamente pari al prezzo di vendita di un'auto scadente, nella simulazione al computer il prezzo non è univocamente determinato. L'equilibrio nel mercato non viene mai raggiunto in termini definitivi, ad ogni ciclo di esecuzione del modello il prezzo medio è sempre diverso, a causa della dinamica interna del sistema. Si può quindi affermare che i risultati sono analoghi soltanto osservando il *trend* dei prezzi medi raggiunti in ciascun ciclo. Il risultato di un modello simulato presenta sempre delle conclusioni sfumate, non si può pretendere di osservare un equilibrio perfetto nel sistema. Questa caratteristica amplifica la similitudine con la realtà, la quale non presenta mai fenomeni dai contorni netti, ma sempre situazioni che si possono spiegare in termini statistici.

I modelli sono stati eseguiti in due differenti configurazioni. Il primo esperimento è stato compiuto in ipotetica assenza di asimmetria informativa. L'obiettivo era quello di convalidare l'ipotesi che in assenza di questo fenomeno il mercato permettesse la realizzazione della totalità delle transazioni ad un prezzo medio tra i due prezzi estremi. In questa configurazione i parametri sono stati così impostati:

- Il numero di agenti è di 46, un numero scelto casualmente.
- I limiti di sincerità dei venditori [*envSincerityMin, envSincerityMax*] sono [1,1]. Ciò impone massima onestà nella dichiarazione dei prezzi.
- I limiti di fiducia del compratore [*envFaithMin, envFaithMax*] sono [1,1]. La ipotesi più forte che ci si aspetti il verificarsi della totalità delle transazioni è che il compratore abbia la massima fiducia di quanto dichiarato dal proprio interlocutore.
- Il parametro *envFaithCorrection* è stato impostato a 0.02, comportando un adattamento molto graduale del compratore ai risultati della sua esperienza. In questo primo esperimento questa variabile è ininfluente poiché i venditori sono onesti e quindi non si verifica mai la dissonanza post-acquisto.
- Il parametro *envMissingCorrection* è impostato a 10. In questo caso l'agente deve attendere 10 cicli di fallimento da transazione consecutivi per correggere di soli $100 il proprio prezzo limite.
- Il prezzo base *envBasisPrice* è 1070, un valore casuale. Da cui deriva il prezzo di un'auto di buona qualità pari a 2140 ed il prezzo atteso pari a 1605.
- In fig. 4 è possibile osservare il risultato dell'esecuzione di circa 200 periodi del modello realizzato con *Starlogo*, nel quale emerge una certa stabilità del trend attorno al valore

medio di 1600. Nell'ultimo periodo di calcolo si sono riscontrate 46 transazioni, ovvero la totalità di quelle potenziali. Il prezzo medio è stato rilevato esattamente pari a 1533.9, un valore prossimo al prezzo atteso.

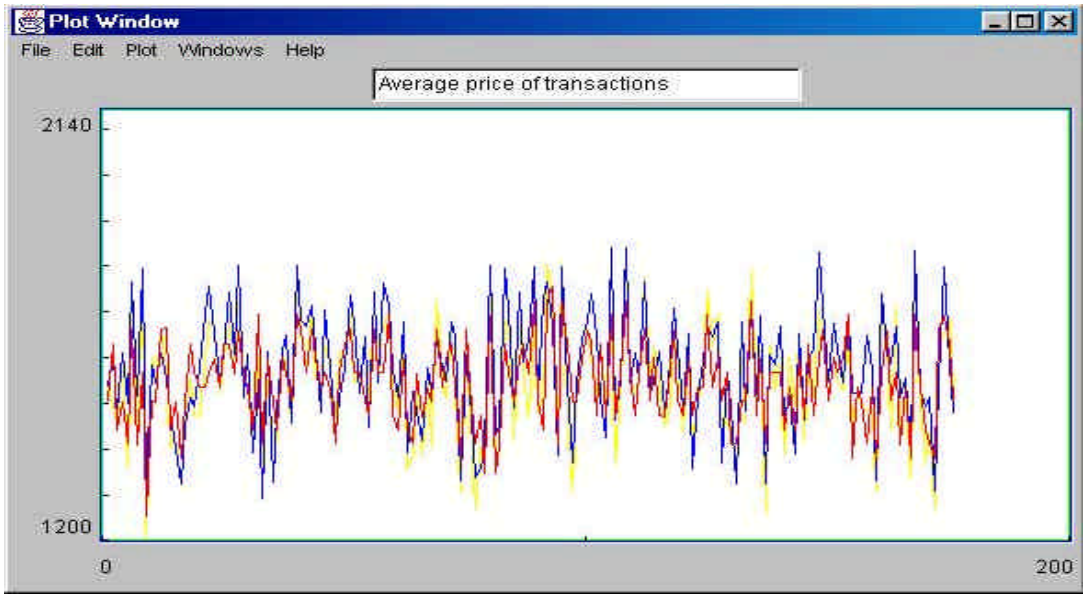

fig. 4 Il risultato dei prezzi di mercato in ipotesi di assenza di asimmetria informativa (*Starlogo*)

Il modello è stato eseguito mantenendo gli stessi parametri con *Matematica*. Le uniche differenze sono il numero degli agenti, pari a 50, ed il numero di cicli di esecuzione è stato ridotto a 100.

Il risultato può essere apprezzato in fig. 5, oltre ad esso lo strumento permette di calcolare agilmente la media globale di tutte le rilevazioni, al contrario dello strumento concorrente. Essa risulta 1605, confermando ulteriormente le ipotesi iniziali del modello di riferimento.

Dopo aver confermato la validità delle ipotesi di modello perfetto non resta che misurare gli effetti prodotti dall'introduzione del disequilibrio nella distribuzione delle informazioni. Ciò si realizza introducendo in media la tendenza dei venditori a dichiarare un valore di qualità relativo alla vettura posseduta più elevato di quello reale. Contemporaneamente è necessario introdurre un elemento di sfiducia dei compratori nei confronti delle dichiarazioni ricevute dalla controparte.

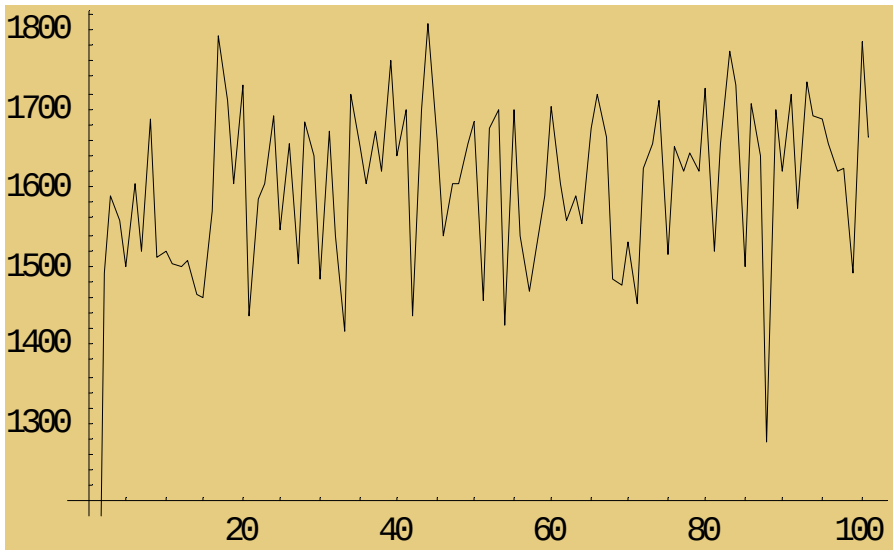

fig. 5 Il risultato dei prezzi di mercato in ipotesi di assenza di asimmetria informativa (*Mathematica*)

Per raggiungere questo risultato sono mantenuti costanti tutti i parametri prima elencati, variando soltanto i coefficienti di fiducia e di sincerità:

- I limiti di sincerità dei venditori [*envSincerityMin, envSincerityMax*] sono [1,1.9]. Ciò significa che il prezzo (e quindi la qualità) da loro dichiarato è mediamente molto più elevato di quello reale. È importante rilevare che i valori sono stati scelti totalmente in modo casuale.

  I limiti di fiducia del compratore [*envFaithMin, envFaithMax*] sono [0.3,1].

Il risultato di tali parametri imposti al modello in *Starlogo* sono visibili in fig. 6 e producono un prezzo medio nell'ultima rilevazione pari a 1165.2 ed un numero di transazioni realizzate pari a 8, ovvero il 17,3% delle transazioni potenziali. Anche in questo caso il risultato è compatibile con ciò che ci si attendeva. Sebbene questo strumento di analisi non fornisca un valore medio relativo a tutto l'arco della simulazione, in figura si nota che il numero di transazioni è sempre molto ridotto ed il prezzo al quale avvengono gli scambi è prossimo al prezzo base, come dimostra la linea gialla.

In sintesi, le ipotesi di equilibrio del modello di riferimento sono confermate. Si osserva, inoltre, che a differenza dell'esperimento in assenza di selezione avversa nel quale la fiducia media dei compratori era pari al 100%, in questo caso il modello osserva un valore pari al 54%. Esso è più basso del valore medio iniziale che si attesta intorno al livello del 35% (la media tra i valori 0.3 e 1), poiché la dissonanza post-acquisto ha prodotto un peggioramento della stima di fiducia dei compratori.

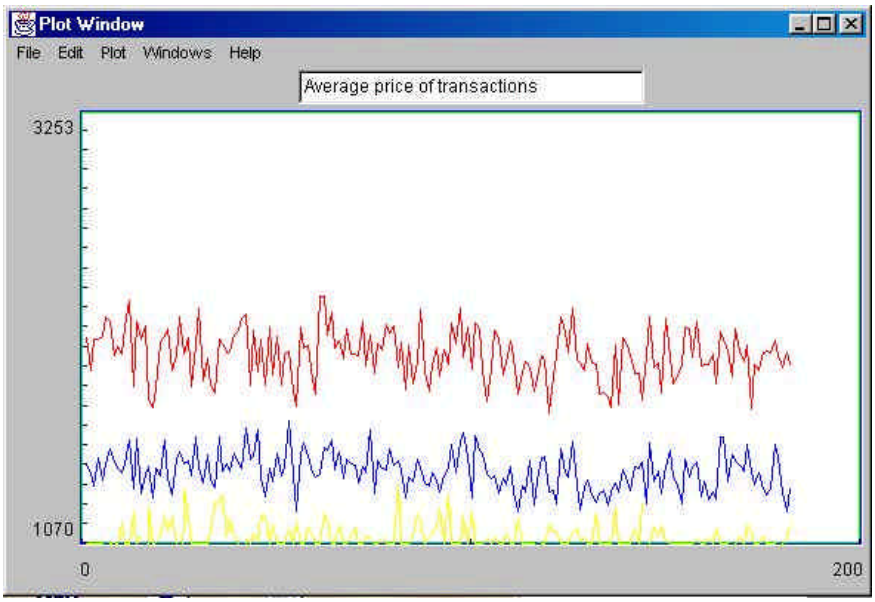

fig. 6 Il risultato dei prezzi di mercato in ipotesi di presenza di asimmetria informativa (*Starlogo*)

L'esecuzione del modello nel secondo ambiente di sviluppo utilizzato presenta un risultato leggermente diverso da quello poc'anzi presentato, ma conferma in linea generale le conclusioni. In fig. 7 sono presentati in forma grafica i risultati della simulazione. In questo grafico si comprende meglio ciò che è accaduto durante la simulazione. Spesso nel mercato non è avvenuta nessuna transazione (quando la linea scende al di sotto del prezzo base) e la maggior parte dei cicli di simulazione ha realizzato un prezzo medio che si avvicina molto al prezzo minimo. Il valore medio complessivo è risultato pari a 1280. Ciò si discosta leggermente dal valore ipotizzato dal modello di Akerlof (1070, in questo caso). La conclusione che se ne può trarre è che sebbene le conclusioni teoriche siano confermate in senso generale, il reale valore di mercato delle auto è determinato sulla base di numerosi fattori, la cui interazione produce valori non necessariamente corrispondenti a quelli attesi.

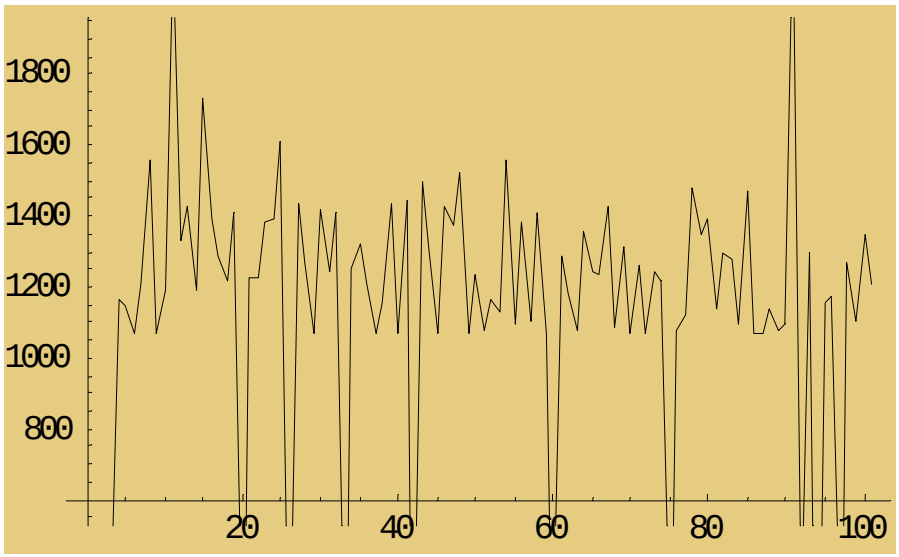

fig. 7 Il risultato dei prezzi di mercato in ipotesi di presenza di asimmetria informativa (*Mathematica*)

Il fascino della simulazione di modelli distribuiti per mezzo dell'informatica è legato ai risultati molto più plausibili a cui si giunge, ma soprattutto alla possibilità di osservare con molta facilità cosa può accadere modificando le ipotesi.

4.10 Conclusioni

In sintesi, il confronto tra i due ambienti non si risolve a favore di nessuno, poiché le differenze che li caratterizzano sono molte. Si è osservato come la realizzazione pratica di un modello economico a partire da un meta-modello informatico che ne ha stabilito i principi realizzativi abbia portato a conclusioni compatibili dal punto di vista economico, ma molte differenze nella capacità di indagare i risultati prodotti nei due casi e della capacità di interagire con il modello stesso.

È indubbio che *Starlogo* fornisca un ambiente più idoneo alla costruzione di simulazioni informatiche, ma il suo ambito di applicazione è piuttosto ridotto. Inoltre, lo strumento non è ancora molto diffuso nella comunità scientifica, a causa della sua recente introduzione.

Matematica è senza ombra di dubbio uno strumento più standard, poiché maggiormente diffuso. Il suo ambito applicativo è pressoché infinto e le capacità di calcolo di cui è dotato lo rendono comunque uno strumento interessante anche ai fini della simulazione sociale. L'apprendimento di quest'ultimo strumento è sicuramente un ottimo investimento in termini di conoscenza, poiché può essere utilizzato per risolvere un grande numero di problemi scientifici. Inoltre, l'interfaccia *J/Link* che permette di utilizzare il nucleo di calcolo nelle applicazioni *Java* lo rende assai aperto all'integrazione con gli altri strumenti di analisi. La sua capacità di esportare i risultati in un formato compatibile con l'internet è certamente un fattore importante. *Starlogo* è

molto limitato da questo punto di vista, poiché usa un formato di memorizzazione proprietario ed i risultati sono salvati in forma testuale (ASCII).

Dal punto di vista del supporto fornito dai progettisti del software agli utilizzatori non ci sono molte differenze. Entrambi gli strumenti hanno a disposizione una lista di discussione sull'internet che mette in comunicazione la comunità degli utenti. Anche le prospettive di sviluppo e di manutenzione futura dei due software è garantita dalle strutture che hanno dato loro vita.

Nella valutazione deve essere tenuto presente, però, che mentre *Starlogo* è un software accademico completamente gratuito, *Matematica* è un software commerciale e viene concesso in licenza d'uso ad un prezzo piuttosto elevato. Tale elemento, date le finalità di ricerca di coloro che svolgono simulazioni sociali, può costituire un limite nella scelta di uno strumento che non costituisce un ambiente ideale per questo tipo di ricerca.

Capitolo 5 *Swarm*

5.1 Introduzione

Nel capitolo precedente è stata affrontata nella pratica la metodologia di simulazione ad agenti di un modello economico attraverso l'uso del computer. Non si può nascondere che le problematiche da affrontare, se si segue questo percorso di analisi, spesso riguardano aspetti che non sono propriamente inerenti alla sostanza di ciò che si vuole studiare. È necessario occuparsi delle cosiddette *technicalities*, ovvero gli aspetti tecnici relativi al software. Lo strumento che si propone di risolvere parzialmente questo problema e che sta diventando lo strumento più usato nel campo della simulazione di modelli distribuiti è chiamato *Swarm*. Esso rappresenta lo strumento di riferimento nello sviluppo di modelli informatici basati su agenti. Si è visto in precedenza quali vantaggi, ma anche quali limiti, presenti l'uso di strumenti progettati *ad hoc*. È stato anche accennato al fatto che l'uso di un qualunque linguaggio di programmazione risolve il limite della mancanza di flessibilità di questi ultimi, 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, è semplicemente una libreria di funzioni scritte in *Objective-C* ed una filosofia progettuale.

5.2 Swarm

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. È 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, 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 ogetti 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. 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 3. Di fatto fornisce lo strumento pratico per realizzare ciò che la teoria dei modelli adattivi complessi definisce.

5.3 La tecnologia ad oggetti

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, di cui si è visto un esempio nel modello sulle asimmetrie presentato nel capitolo precedente. Un oggetto è la combinazione di variabili di stato che ne definiscono le caratteristiche interne e di 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 faccia 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 dalle 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) del modello, in modo interattivo. Questo elemento è assolutamente rivoluzionario 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".

5.4 I bugs

In Axelrod (1997) si fa notare che un nodo fondamentale della ricerca scientifica in questo campo è riposto nell'attenzione alla qualità dei modelli.

> In order to be able to achieve the goals of validation, usability and extendibility, considerable care must be taken in the entire research enterprise. This includes not just the programming, but also the documentation and data analysis. (...) I have learned the hard way that haste does indeed make waste. Quick results are often unreliable. Good habits slow things down at first, but speed things up in the long run.

In Gilbert, Terna (1999) 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, Epstein (1994) affermano a tal proposito che

> Such software problems are difficult to discover due precisely to the highly distributed nature of agent-based models. Indeed, the "robustness" of macro-structures to perturbations in individual agent performance – "graceful degradation" to borrow a phrase from neural networks – is often a property of agent-based models and exacerbates the problem of detecting "bugs".

Sempre in Gilbert, 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.

Si è già fatto notare che anche in *Starlogo* esiste questa opportunità e ciò testimonia come nell'ambito della simulazione informatica ad agenti il meccanismo di ispezione delle variabili di ciascun agente sia essenziale. Ciò è ulteriormente dimostrato dal fatto che un'applicazione scientifica "tradizionale" come *Mathematica* non prevede questa funzionalità. 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.

5.5 La gestione del tempo

In Lin, Tan e Shaw (1996) si afferma che uno sciame non è soltanto una collezione di oggetti ma 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 in fig. 8. 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 totalmente 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 opportuno messaggio nel momento opportuno.

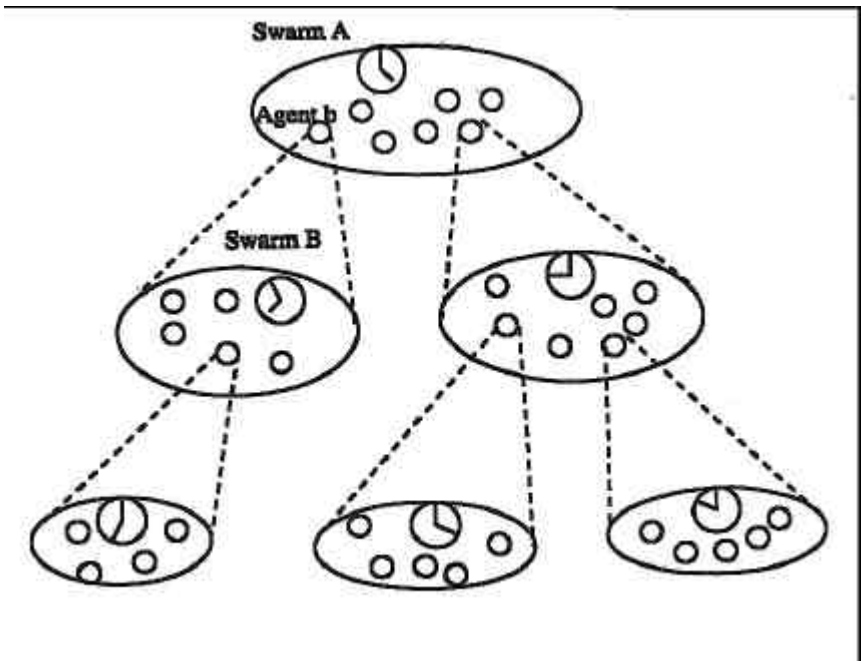

fig. 8 L'albero gerarchico nidificato degli sciami

5.6 Il modello Akerlof

Il modello di Akerlof è stato scritto utilizzando *Swarm* al fine di confrontare le caratteristiche di questo ambiente con i software presentati nel capitolo precedente. Da un punto di vista teorico dovrebbero essere avvantaggiati i primi, poiché il modello di riferimento è piuttosto semplice. Un sistema così articolato e potente come *Swarm* è stato progettato per consentire la realizzazione di progetti ambiziosi, caratterizzati da molte tipologie di agenti e, soprattutto, da meccanismi cognitivi fondati su regole o su algoritmi adattivi, i quali richiedono molte istruzioni informatiche e quindi un ambiente robusto. In realtà, tralasciando che questo ambiente richiede la conoscenza di linguaggi di programmazione impegnativi come *Objective-C* o *Java*, la realizzazione del modello è assolutamente lineare e non impone alcuno dei compromessi necessari in *Starlogo* e *Mathematica*. Ricordiamo che nel primo è stato necessario introdurre un sofisticato meccanismo concettuale per realizzare la comunicazione sequenziale tra gli agenti e che nel secondo si è dovuto limitare notevolmente il numero di cicli di calcolo ed il numero di agenti a causa della lentezza nello svolgimento dei calcoli (precisione infinita, calcolo simbolico).

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. Nel nostro caso esistono due classi. La prima è la classe dei venditori (*seller*) che possiedono le autovetture e la seconda è quella dei potenziali compratori (*buyer*).
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*, che consiste in un oggetto che si occupa di esaminare il comportamento dell'oggetto/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.

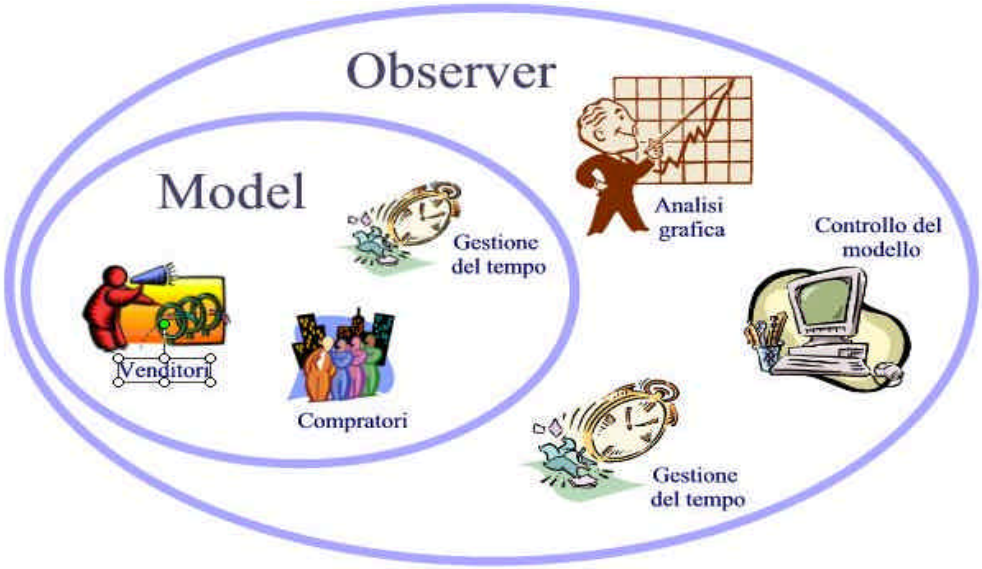

fig. 9 Il modello ad oggetti del software jAkerlof

In fig. 9 è rappresentato lo schema che descriverla struttura del modello Akerlof secondo la metodologia *Swarm*. Il *model* costruisce un numero pari al parametro opportuno di agenti delle due categorie previste (compratori e venditori). 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 lo scatenarsi 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 tutti 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.

5.7 Le interfacce

Attraverso la tecnologia di programmazione ad oggetti è molto semplice ed efficace descrivere il funzionamento del modello, poiché ogni oggetto espone all'esterno un'interfaccia che ne definisce in modo semplice le caratteristiche in termini di memoria (le variabili di stato) e di comportamento (i metodi).

L'agente compratore (***buyer***) dispone della seguente interfaccia:

Le variabili di stato

private float	**faith**;
private float	**price**;
private float	**qualityComunicated**;
private int	**qualityReal**;
private int	**missing**;
private float	**sellerPrice**;
private float	**transactionPrice**;
private float	**envBasisPrice**;
private float	**envFaithCorrection**;
private int	**envMissingCorrection**;

I metodi

public float	**getFaith**()
public float	**getPrice**()
public int	**getMissing**()
public float	**getTransactionPrice**()
public float	**getSellerPrice**()
public Object	**thinkPrice**()
public Object	**initCycle** ()
public Object	**makeCorrections**()
public Object	**setSellerPrice$Quality$QualityReal**()
public boolean	**doYouLikeMyPrice**()
public Object	**step1** ()
public Object	**step2** ()

L'agente venditore (***seller***) dispone della seguente interfaccia:

Le variabili di stato

private float	**sincerity**;
private float	**price**;
private int	**quality**;
private int	**missing**;
private float	**transactionPrice**;
private float	**envBasisPrice**;
private int	**envMissingCorrection**;
private Buyer	**myBuyer**;

I metodi

public float	**getSincerity**()
public float	**getPrice**()
public int	**getMissing**()
public float	**getTransactionPrice**()
public float	**getSellerPrice**()
public Object	**thinkPrice**()
public Object	**initCycle** ()
public Object	**makeCorrections**()
public Object	**comunicatePriceToBuyer** ()
public boolean	**setCurrentBuyer** ()
public Object	**step1** ()
public Object	**step2** ()

Attraverso questa rappresentazione si è a conoscenza dei messaggi (i metodi) ai quali ciascun agente è in grado di rispondere. In questo caso tutte le variabili di stato sono "private" e quindi inaccessibili dall'esterno. La loro modificazione è gestita dai meccanismi interni dell'agente sulla base dei messaggi ricevuti. I metodi chiamati *step1* e *step2* posseduti da entrambe le tipologie di agenti sono i metodi generici richiesti dalla filosofia *Swarm*. Nell'oggetto che si occupa di notificare gli eventi è prescritto di chiamare in sequenza questi metodi, all'interno dei quali sono svolte le operazioni necessarie allo svolgimento del modello.

Nell'oggetto *ModelSwarm* l'interfaccia è molto uniforme per ogni modello, poiché la filosofia *Swarm* impone di usare metodi predefiniti, le variabili invece dipendono in modo determinante dal modello, esse sono le variabili di ambiente.

Il *ModelSwarm* ha la seguente interfaccia:

Le variabili di stato

public int	**agentNumber**;
public float	**envBasisPrice**;
public float	**envFaithCorrection**;
public int	**envMissingCorrection**;
public float	**envSincerityMin**, **envSincerityMax**;
public float	**envFaithMin**, **envFaithMax**;
public ActionGroup	**modelActions**;
public Schedule	**modelSchedule**;
public List	**buyerList**;
public List	**sellerList**;
private ListShufflerImpl	**listShuffler**;

I metodi

public List	**getBuyerList** ()
public List	**getSellerList** ()
public Object	**meetAgents** ()
public Object	**buildObjects** ()
public Object	**buildActions** ()
public Activity	**activateIn** ()

Si può osservare che sono presenti i parametri della simulazione in qualità di variabili di stato dell'oggetto che realizza la simulazione. Questo oggetto possiede due metodi standard nella filosofia di progetto di *Swarm*: *buildObject* e *buildActions*. Il primo costruisce il numero di agenti

necessari alla simulazione ed il secondo la lista di attività che devono essere svolte in ogni ciclo, il metodo *ActivateIn* esegue il modello.

Il metodo *meetAgents* sfrutta un oggetto della libreria di *Swarm* chiamato *ListShuffler*, il quale con una semplice istruzione mescola casualmente una lista, garantendo l'incontro casuale di tutti gli agenti in ogni ciclo di simulazione. Questa operazione è assai più complessa nei precedenti software analizzati e mette in luce la potenza di *Swarm.*

La lista delle azioni genera in sequenza i seguenti messaggi:

- il messaggio meetAgents è inviato al *ModelSwarm*: mescola la lista dei compratori;
- è inviato il messaggio *step1*, il quale invoca il metodo *initCycle*, alla lista dei compratori;
- è inviato il messaggio *step1*, il quale invoca il metodo *initCycle*, *thinkPrice*, *comunicatePriceToBuyer*, alla lista dei venditori;
- è inviato il messaggio *step2*, il quale invoca il metodo *thinkPrice*, alla lista dei compratori;
- è inviato il messaggio *step2*, il quale invoca il metodo *doYouLikeMyPrice* dell'agente compratore, alla lista dei venditori, se la risposta è positiva la transazione è eseguita;
- è inviato il messaggio *makeCorrections*, alla lista dei compratori;
- è inviato il messaggio *makeCorrections*, alla lista dei venditori.

Anche l'oggetto *ObserverSwarm* ha una struttura predefinita e spesso possiede solo alcuni piccoli elementi che fanno riferimento al modello sottostante, ma per la gran parte del codice contiene istruzioni delle librerie fornite dal sistema. Ciò riduce la necessità da parte della comunità scientifica di comprendere le scelte concettuali di progettazione del software da parte del programmatore, poiché una buona dose di oggetti e di istruzioni sono universalmente conosciute.

5.8 I risultati

La simulazione è stata eseguita mantenendo la stessa struttura di quelle del capitolo precedente. Le due esecuzioni sono state effettuata con i parametri in fig. 10.

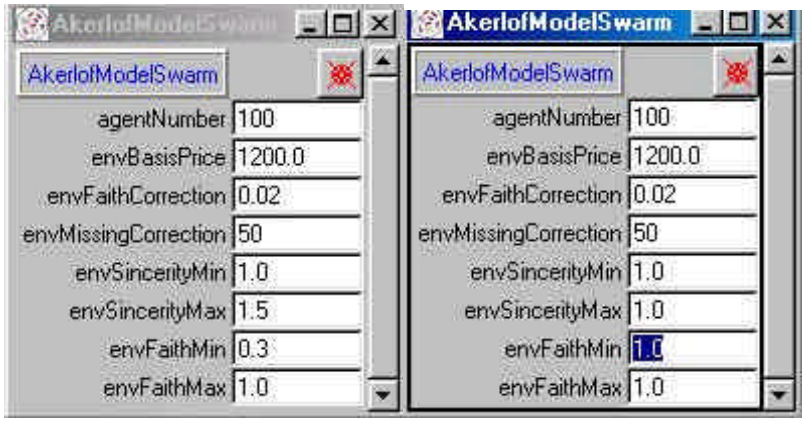

fig. 10 Parametri relativi all'ipotesi di mercato imperfetto (a sinistra) e di mercato perfetto (a destra)

Il risultato della prima, in ipotesi di assenza di asimmetria informativa è apprezzabile in fig. 11 e della seconda, in ipotesi di presenza di asimmetria informativa, in fig. 12.

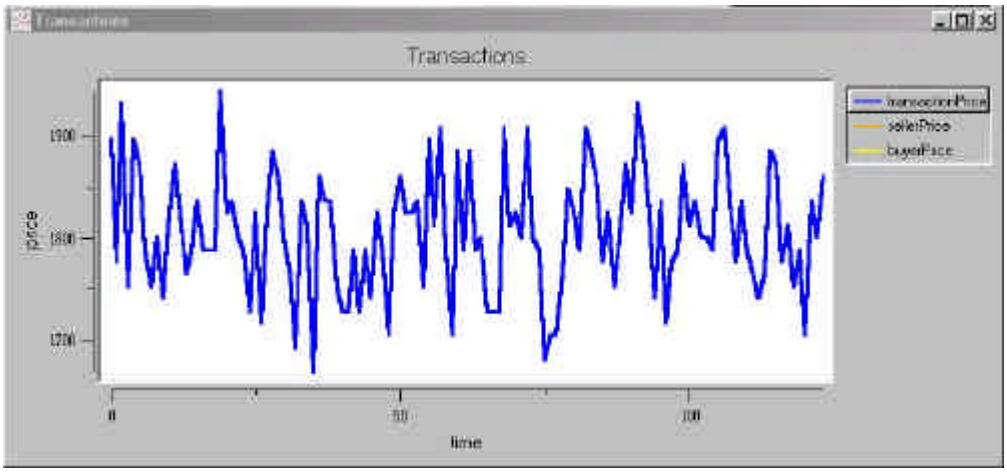

fig. 11 Il risultato dei prezzi di mercato in ipotesi di assenza di asimmetria informativa (*Swarm*)

Si può osservare che i risultati rispecchiano le attese. Nel primo caso la scala delle ordinate è compresa tra 1700 e 1900, per cui si può osservare come la serie dei prezzi medi di transazione sia stazionaria attorno alla media di 1800. Si nota che tutte le transazioni sono avvenute, perché le tre serie storiche dei prezzi (prezzi medi dei compratori, dei venditori e delle transazioni) si sovrappongono. Ciò è garantito dal meccanismo che assicura l'incontro uno ad uno di tutti gli agenti. Questo non accadeva nei precedenti esperimenti, provocando alcune differenze nelle rilevazioni dei prezzi delle due categorie di agenti.

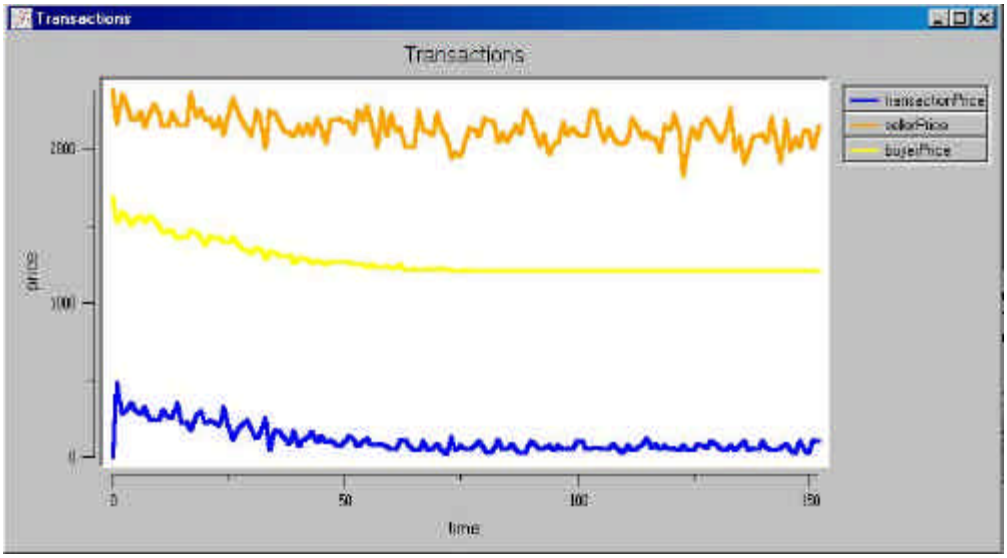

fig. 12 Il risultato dei prezzi di mercato in ipotesi di presenza di asimmetria informativa (*Swarm*)

Nel secondo caso il prezzo medio si avvicina a 0 perché la media dei prezzi è calcolata tenendo presente anche gli agenti che non hanno scambiato (prezzo = 0). Il fenomeno testimonia che sono avvenuti pochi scambi e tutti ad un prezzo vicino al prezzo base (1200). Infatti, la media dei prezzi dei compratori (giallo) si avvicina ad esso.

5.9 Il confronto con Starlogo

In sintesi, si può affermare che *Swarm* sia candidato ad imporsi come lo strumento di riferimento nel campo della simulazione sociale. Esso non impone compromessi e consente di realizzare ogni tipo di simulazione.

Indubbiamente, il costo più grande è rappresentato dalla necessità di conoscere, seppure in modo non approfondito, un linguaggio di programmazione sofisticato. Questa necessità, ad un'analisi più attenta, risulta essere un buon investimento in termini di conoscenza.

Infatti, se la difficoltà del modello è bassa, la realizzazione in *Starlogo* è per alcuni versi più intuitiva, ma al crescere della complessità e del numero di categorie di agenti il problema esplode e può essere affrontato solamente con uno strumento solido e rigoroso come *Swarm*.

Capitolo 6 I modelli di simulazione aziendale: l'azienda virtuale

6.1 La simulazione dell'azienda

La metodologia di simulazione ad agenti è stata presentata finora come strumento finalizzato all'analisi della dinamica di sistemi fondati su agenti. Per agente si intende una rappresentazione informatica schematizzata di entità reali, nella quale sono introdotte le regole e le ipotesi che tentano di spiegarne il comportamento e attraverso la cui interazione si tenta di far emergere il complesso dei fenomeni del sistema. Nulla impedisce di applicare tale metodo per rappresentare una realtà concreta, costituita da agenti fisici.

L'obiettivo è applicare questa metodologia nell'ambito della simulazione delle organizzazioni aziendali, intese come singole unità di *business* o come intere catene produttive.

Non esiste una esatta definizione di impresa. È una buona interpretazione quella in Lin, Tan Shaw (1996) che descrive l'impresa come una collezione di entità di *business* che cooperano allo scopo di fornire un prodotto o un servizio al consumatore. Queste entità possono essere gli elementi costitutivi di una singola azienda oppure essere aziende che costituiscono l'intera catena del valore.

Dal punto di vista del prodotto, la catena del valore consiste in un processo che realizza come risultato il bene o il servizio ed ogni singola entità aggiunge valore ad esso. Un esempio di catena del valore formato da industrie tessili è rappresentato in fig. 13.

fig. 13 L'industria tessile

Dal punto di vista del processo, la catena del valore può essere vista come una sequenza di operazioni che trasformano progressivamente le materie prime in prodotti finali. Considerando ad esempio un processo come il singolo blocco di produzione (*building block*), che aggiunge un unità di valore al prodotto, si può descrivere il processo.

Sia definito con *p* l'unità di processo che riceve in ingresso un *input (i)* e lo trasforma producendo un *output (O)*, esso è descritto dalla formula $O=p(i)$. In una catena di processi, il prodotto (O) del processo p^i diventa l'*input* del processo p^{i+1}. Tutti gli *output* del tipo O^i sono prodotti intermedi, mentre l'*output* O^n è il prodotto finale.

La catena dei processi non è necessariamente costituita da una struttura lineare. Un processo può essere parzialmente ordinato. Possono riscontrarsi situazioni in cui molti *input* sono

trasformati in un singolo *output*. Oppure si può verificare la situazione inversa (un *input* trasformato in molti *output*). Poiché è possibile raggruppare più unità di processo in un processo più grande, attraverso questo metodo si realizza un gerarchia di processi che trasformano i materiali grezzi in prodotti finiti, attraverso una sequenza lineare.

Al fine di descrivere un modello di simulazione dell'impresa occorre decomporre la catena del valore in singole fasi di prodotto o in singoli processi produttivi, per poter identificare le categorie di entità che costituiscono il modello e le azioni che queste devono essere capaci di svolgere. È anche necessario identificare la prospettiva dalla quale si vuole studiare l'intero sistema. Le possibili prospettive sono:

1. Da un punto di vista del **flusso dei materiali**. Una catena di fornitura è una rete che mette in comunicazione entità di *business* attraverso varie attività. Un'entità può essere un'industria che trasforma le materie prime in prodotti finiti (o semilavorati), un distributore che consegna i prodotti ai clienti, dotato o meno di capacità di assemblaggio, un dettagliante che vende i prodotti ai clienti finali o un magazzino di stoccaggio dei prodotti. Tali attività di raccolta ordini, produzione e distribuzione sono finalizzate a facilitare il flusso dei prodotti verso i consumatori.
2. Dal punto di vista dei **ruoli** svolti dalle entità. Il compito di ciascuna unità è di spostare le materie prime nelle due direzioni lungo le quali si snoda la catena produttiva. Ciò si riferisce al ruolo non assolutamente definito che svolge la singola entità: un produttore è contemporaneamente un fornitore nei confronti delle unità che stanno a valle della catena e un cliente di coloro che sono posizionati a monte.
3. Da una prospettiva relativa ai **confini organizzativi**. Un'unità di *business* è un partecipante della rete e può essere rappresentata secondo differenti gradi di astrazione in funzione dei confini delle organizzazioni. Ciò si traduce in una rappresentazione della catena di fornitura in uno schema a molti strati (*multiple abstraction layer*). I processi aziendali possono essere osservati nella loro interezza, superando i confini che delimitano le singole entità, come frutto della cooperazione di differenti funzioni aziendali appartenenti ad organizzazioni diverse ed indipendenti tra loro
4. Dal grado di **visibilità** delle entità nella rete. Esse sono collegate tra loro attraverso informazioni. Ciò fornisce gli elementi per disegnare la rete delle relazioni che intercorrono tra esse. Attraverso il passaggio di informazioni, un'entità può migliorare la conoscenza e l'interazione con l'ambiente e quindi prendere decisioni in modo più flessibile. Tali informazioni aumentano, dunque, la visibilità della catena di fornitura. In funzione dei diversi gradi di visibilità, in aggregato, emergono diverse strutture organizzative e diverse caratteristiche dei processi di produzione.

Gli elementi che devono essere descritti per costruire una visione completa del sistema sono:

- le *business units*, che rappresentano un livello concettuale di astrazione che può corrispondere all'intera organizzazione o ai singoli dipartimenti produttivi;
- le attrezzature, che corrispondono alle risorse fisiche dell'organizzazione. Parallelamente al punto precedente, esse possono rappresentare l'intero stabilimento di produzione o le singole linee di assemblaggio;
- i processi, i quali rappresentano le attività di produzione e possono essere scomposti in sub-processi, in funzione del grado di dettaglio necessario;
- gli *input/output*, che costituiscono l'insieme delle informazioni e dei materiali che entrano ed escono dall'impresa;
- ed infine, le relazioni e le interazioni tra i singoli dipartimenti o tra le aziende che collaborano.

6.2 I modelli di integrazione aziendale

In letteratura esistono diversi tentativi di definire un metodo o un'architettura di riferimento per la descrizione dell'organizzazione, secondo l'identificazione delle componenti appena citate. I più significativi sono raccolti in Lin, Tan, Shaw (1996):

1. *The open system architecture for computer integrated manufacturing (CIMOSA)*. È una struttura di modellazione, conosciuta come cubo CIMOSA, che definisce livelli descrittivi separati ma correlati (definizione degli obiettivi, specifiche di progetto e descrizione dell'implementazione), tre strati (i *building block* generici, i modelli parziali ed i modelli particolari) e quattro visuali (funzioni, informazioni, risorse e organizzazioni). (Vernadat 1993).
2. *The Architecture of Integrated Information Systems (ARIS)*. Sviluppata da Scheer (1993,1994) come modello di riferimento per le imprese industriali. Descrive i processi di *business* utilizzando quattro visioni: le funzioni, le organizzazioni, i dati ed il controllo. Questi processi sono messi in relazione con il tecnico dell'*information technology(IT)* in tre livelli: definizione degli obiettivi, specifiche di progetto e descrizione dell'implementazione.
3. *The GRAI Integrated Mehodology (GIM)*. Questo modello si basa su un quadro descrittivo a due dimensioni: le viste ed i livelli di astrazione. Dal punto di vista della prima dimensione, il modello consiste in quattro sistemi: informativo, decisionale, fisico e funzionale. Dal punto di vista tecnico consiste nella descrizione dell'organizzazione, dell'*information technology* e nelle tecnologie produttive. (Doumeingts, 1993).
4. *The Purdue Enterprise Reference Architecture (PERA)*. Esso è definito da cinque fasi:
    a. *Fase concettuale*: identificazione delle entità produttive, descrizione della missione aziendale, visione dei valori del *management*, filosofia aziendale.

b. *Fase definitoria*: definizione degli obiettivi, dei moduli di *building block*, processi di formazione della conoscenza e processi di produzione.
c. *Fase progettuale*: disegno dettagliato delle funzioni del sistema informativo, delle strutture organizzative relative alle persone ed agli apparati produttivi.
d. *Fase di costruzione ed installazione*: durante la quale viene costruito ciò che è stato definito nelle fasi precedenti.
e. *Fase operativa*: le tre architetture sono utilizzate e sviluppate continuamente. (Williams, 1993).

5. *The Integrated Enterprise Modeling (IEM)*. Un modello orientato alla modellazione ad oggetti. Gli elementi della azienda sono suddivise in tre categorie: i prodotti, gli ordini e le risorse. Il modello generico sulle attività cattura i processi di produzione in descrizioni di attività. Sono sviluppati due punti di vista: la prospettiva funzionale che descrive gli aspetti funzionali degli oggetti della catena di produzione e la prospettiva informativa che descrive le caratteristiche di tali oggetti. (Mertins et al., 1992).

Tutti questi modelli di descrizione delle aziende mettono in risalto il ruolo dell'*IT*. Esso rappresenta il punto di incontro tra la realtà dell'organizzazione e la sua rappresentazione schematica. Soltanto attraverso un meccanismo che formalizzi la conoscenza ed i flussi di merci ed informazioni è possibile costruire un modello che descriva correttamente il sistema. Questi modelli, però, non possiedono caratteristiche innovative quali la rappresentazione dell'astrazione multi-livello, la capacità di rappresentare il passaggio delle informazioni attraverso l'uso dei messaggi e la capacità di adattamento. Inoltre, non consentono la simulazione di ciò che descrivono. La descrizione è statica, non permette di effettuare un'analisi degli effetti prodotti dall'introduzione di cambiamenti in uno degli elementi descritti. Tutto ciò è stato incluso nel modello MAIS realizzato con *Swarm*, nel quale si descrivono le entità della catena di fornitura come oggetti software che possono essere simulati.

6.3 Il Multi-Agent Information System (MAIS).

Lin, Tan e Shaw (1996) propongono un'interessante applicazione della metodologia di simulazione ad agenti nell'ambito della gestione delle catene di fornitura. Il modello è denominato MAIS (*multi-agent information system*) ed è composto di 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 o aggregate in una singola attività affidata ad un singolo 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.

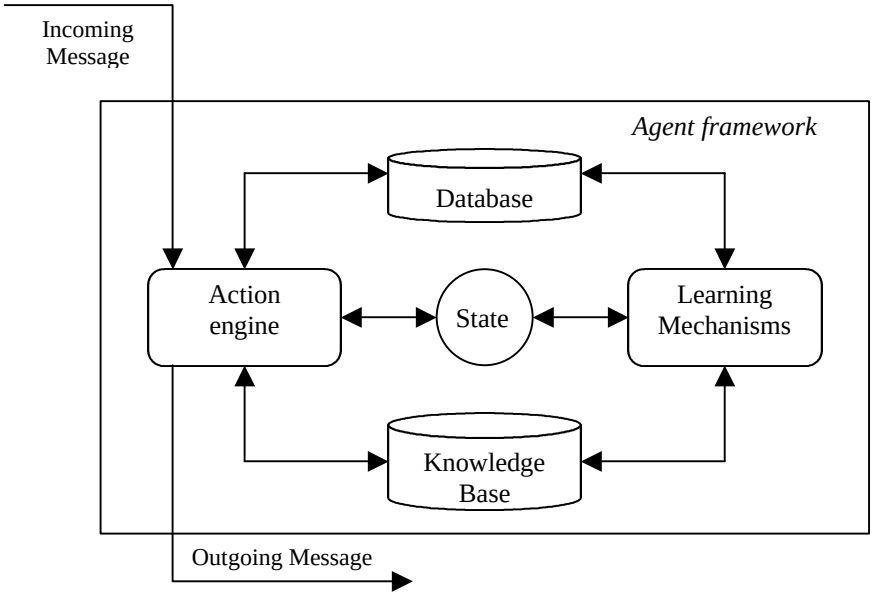

fig. 14 I componenti di un agente

Esistono due tipologie di agenti: fisici e logici. La struttura interna degli agenti logici è rappresentata in fig. 14 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, usano 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, quindi, sono capaci 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. Ciò che lo differenzia nella sostanza dai modelli citati in precedenza è il suo orientamento applicativo. Infatti, l'intera struttura è disegnata per essere codificata nel sistema di simulazione *Swarm*. 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. Il risultato si può apprezzare in Tabella 1

Tabella 1 La relazione tra le proprietà del modello aziendale e Swarm

Modello di azienda	Swarm
Unità di *business*	Una collezione di agenti
Entità fisica	Agente
Processi	Metodi
Suddivisione dei processi	Gerarchia intrinsecamente nidificata
I/O dei materiali e delle informazioni	Messaggi
Relazioni tra unità di *business*	Relazioni tra agenti
Flusso delle informazioni	Passaggio dei messaggi
Misurazione delle performance	Misurazioni statistiche
Interazioni tra ed all'interno delle *business units*	Elaborazione parallela e concorrente degli agenti

6.4 I risultati

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:

- I 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.
- I meccanismi di controllo delle strutture organizzative. La variazione dei meccanismi di controllo e nella struttura delle relazioni tra entità della catena di fornitura provocano una variazione nei flussi delle informazioni. Si possono contemplare diverse configurazioni strutturali dell'organizzazioni alle quali corrispondono altrettante strutture nel modello di simulazione.
- L'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.
- 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.
- 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 variazione 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.

Va tenuto presente, però, che la natura di questa metodologia è del tutto accademica, come fanno notare i suoi autori:

> The sucessful implementation of the MAIS verifies that out model is an accurate representation of the order fulfillment process (OFP). The simulation captures the behaviors of individual agents and allows them to interact dynamically. Thus, we are able to study not just the interactions and effects of pairs of agents or a small group of agents, but the overall effect of the whole system. It is an invaluable tool that allows us to test out hypotheses without resorting to real experiments. Lastly, the MAIS demonstrates the features of Swarm and establishes its value as a simulation tool in the academic world.

Il valore del modello è legato all'indagine e alla verificazione delle ipotesi sugli effetti delle politiche di gestione della catena di fornitura, non ha lo scopo di simulare una realtà concreta, poiché i dati sono generati internamente sulla base di regole.

I modelli MAIS rendono troppo complessa la realizzazione di simulazioni di realtà concrete, poiché la descrizione del modello impone di realizzare una congettura sulle variabili esogene al modello. 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. Ciò è assai improbabile.

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.

6.5 Il concetto di azienda virtuale

Con il termine di azienda virtuale si fa riferimento a due concetti.

In una prima accezione si può ritenere un modello di azienda attraverso una rappresentazione informatica che ha lo scopo di consentire la simulazione, ovvero il concetto finora considerato. Essa è intesa come l'aggregazione e l'interazione delle sue componenti costitutive. La seconda accezione fa riferimento all'azienda vera e propria 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.

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 prima accezione, invece, la simulazione dell'azienda in un modello di azienda virtuale 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. Esso sarà affrontato nel capitolo successivo.

6.6 L'architettura NIIIP e BizTalk

La realizzazione di un modello di azienda virtuale intesa come aggregato di aziende reali dipende dalla capacità di integrazione delle tecnologie che consentono di scambiare i materiali e soprattutto le 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, i più interessanti sono il NIIIP e *BizTalk*.

L'obiettivo primario del *National Industrial Information Infrastructure Protocols Consortium (NIIIP)* 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 costituito dal governo americano e dimostra l'attenzione e l'interesse da parte dell'economia alla nuova frontiera della comunicazione globale. Il *NIIIP* è così definito:

> The National Industrial Information Infrastructure Protocols (NIIIP) Consortium is an Industry/Government initiative to develop an information infrastructure that enabled collaboration between companies working together on different aspects of a manufacturing process. This new form of collaborative computing will decrease cycle-time and increase responsiveness to change.

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

Al 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 NIIIP 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[3]. Ogni messaggio deve essere inviato:

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

La sicurezza delle informazioni può essere garantita dai protocolli di comunicazione dell'internet (IP), all'interno dei componenti software che realizzano lo strato di interscambio delle informazioni, o dalla stessa architettura del NIIIP 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 NIIIP.

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.

[3] CORBA (*Common Object Request Broker Architecture*) è un modello ad oggetti basato su interfaccia definita dall'OMG (*Object Management Group*). La fonte di riferimento è la *OMA Guide*.

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. Ognuno di 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 NIIIP 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 NIIIP 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 NIIIP non è previsto l'utilizzo del linguaggio XML come formato universale di descrizione e trasporto delle informazioni, mentre ciò sta realizzando di fatto nel mercato. A conferma di tale visione risulta interessante descrivere sinteticamente una prospettiva di soluzione alternativa al progetto NIIIP: il *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* gioca 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 manutenere. 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.

Capitolo 7 Il modello di azienda virtuale Boglietti

7.1 La previsione degli ordini

La premessa teorica che precede il presente capitolo ha lo scopo di fornire gli strumenti concettuali utili ad affrontare l'analisi qui condotta. Tale indagine trae origine dall'esigenza di studiare alcune problematiche di una realtà aziendale concreta. Si pone come obiettivo la necessità di approfondire la conoscenza dei fattori che determinano la struttura della domanda, al fine di migliorare la previsione delle quantità di beni domandati. Ciò fornisce uno strumento di supporto alla gestione delle politiche di acquisto. Il comportamento della domanda è influenzato da fattori esterni che dipendono dalla struttura e dalla dinamica dell'intera catena produttiva in cui l'azienda opera.

La Boglietti S.p.A. è un produttore di capi di abbigliamento. In particolare opera nel settore della produzione di biancheria intima ed è situata all'interno di una zona geografica caratterizzata da alta densità di produttori tessili: il territorio circostante la città di Biella. Esso rappresenta un importante polo produttivo italiano nel campo tessile. Possiede non soltanto un alto numero di aziende ma anche ampia eterogeneità di fasi produttive rappresentate. Per questa ragione, nel biellese è possibile trovare esempi di complete catene di fornitura. Molte aziende, infatti, collaborano strettamente lungo il percorso della produzione, coprendo quasi integralmente l'intera catena del valore aggiunto.

L'intera area di Biella, costituita da 82 comuni, costituisce fin dagli inizi dell'ottocento uno dei maggiori centri mondiali dell'industria laniera. Tale sistema produttivo ha subito una profonda evoluzione nel corso degli anni 70, passando da un'organizzazione della produzione di tipo "verticale", cioè con tutto il ciclo produttivo svolto all'interno della stessa impresa, ad una di tipo "orizzontale", con la specializzazione delle singole aziende per fasi di produzione. Ciò ha comportato un forte rinnovamento tecnologico, in cui rivestono un ruolo strategico i fattori legati da un lato al design e dall'altro ai canali di commercializzazione.

Al di là delle dimensioni delle singole imprese, i "distretti industriali" italiani sono veri giganti economici con quote del 30% sul commercio internazionale di tessuti di lana e di seta. Il solo distretto di Biella impiega circa 29.000 addetti, con 6.500 miliardi di fatturato.

A livello mondiale, il settore di produzione tessile sta iniziando a sfruttare le opportunità del *business-to-business*, per migliorare la visibilità delle aziende nel mercato e per accelerare i tempi di scambio delle informazioni. Esistono esempi come www.wool.com.au, un portale costituito da organizzazioni di produttori tessili che forniscono la possibilità di acquistare *on-line* le materie prime o i semilavorati a prezzi concordati con il meccanismo delle aste e con standard

qualitativi garantiti. In questo settore produttivo è sempre stata avvertita l'esigenza di poter ridurre i tempi di negoziazione dei beni per una ragione di ordine geografico. I produttori di materie prime, in particolare della lana, operano in zone in cui esistono le caratteristiche ambientali ideali per produrre tali beni e spesso queste zone sono distanti dai mercati di sbocco. La lana ad esempio, molto presente nella produzione dell'interland biellese, viene fornita soprattutto dai produttori australiani. A causa della distanza, la stagione in cui avvengono gli scambi è ridotta a tre mesi l'anno. Ciò al fine di formare un prezzo significativo in sede d'asta, il quale può emergere se alla contrattazione partecipa un numero elevato di acquirenti. Quindi, si riducono i tempi di contrattazione per concentrare la presenza di compratori. I produttori e gli acquirenti hanno l'esigenza di estendere il mercato in senso temporale e di restringerlo dal punto di vista spaziale. Questa esigenza ha contribuito alla nascita delle prime aste virtuali secondo il modello del *business-to-business*.

Un altro fattore caratteristico della produzione tessile è la lunga tradizione, che si traduce, come si è detto, in termini economici in un mercato fortemente specializzato. I produttori cercano di differenziare l'offerta per creare vantaggi competitivi sulle caratteristiche del prodotto, perché il miglioramento nelle tecnologie produttive o nelle tecniche distributive non può più rappresentare il solo elemento strategico. Dunque, concentrano l'attività su quello che considerano *core business* e acquistano dall'esterno i fattori di produzione relativamente ai quali non ritengono di poter acquisire vantaggi rispetto alla concorrenza. La specializzazione rappresenta un'altra giustificazione del crescente interesse nei confronti delle nuove tecnologie di scambio.

Una parte non secondaria alla base della competitività e della flessibilità del sistema moda si sostiene proprio sulle interrelazioni tra le imprese tessili e quelle dell'abbigliamento, insieme danno vita a un grande laboratorio specializzato nella creazione e nella realizzazione rapida di una gamma ampia e variata di articoli sempre nuovi.

Nell'analisi dell'azienda Boglietti si devono tenere presenti le caratteristiche strutturali dell'industria appena descritte. Il settore della biancheria intima copre due mercati molto differenziati tra loro. L'abbigliamento intimo classico concentra la competizione sul rapporto qualità-prezzo, mentre elementi quali il colore, il tipo di confezione, l'aspetto stilistico non sono fattori determinanti. I consumatori percepiscono questo tipo di abbigliamento come un bene di prima necessità, nel quale conta la sostanza del prodotto, non il suo lato esteriore. In questo mercato non si possono generare interessanti vantaggi comparativi rispetto ai concorrenti, a causa dell'impossibilità di differenziare il prodotto. Ciò si traduce in un mercato fortemente in concorrenza, nel quale il prezzo dei prodotti è una variabile esogena, non condizionabile dal fornitore. Il vantaggio potenziale in termini di profitto può giungere dalla riduzione dei costi. A causa dell'alta specializzazione però, i produttori hanno raggiunto forti economie di scala che limitano l'intervento anche su questo versante. Emerge, dunque, un mercato che offre poco

margine di profitto. Il secondo mercato è invece formato da consumatori che possiedono differenti aspettative nei confronti del prodotto. Sono interessati alle caratteristiche estetiche della biancheria, poiché lo ritengono un elemento che “fa moda”. Esso è legato al bisogno di “immagine” che la moda appaga, determinando un mercato molto più dinamico e differenziato. In questo caso i beni sono soggetti ad una rapida obsolescenza, poiché il colore di moda varia da una stagione all’altra e le tendenze all’uso di particolari confezioni o modelli sono soggette a rapide mutazioni. In questo contesto ci sono grossi margini di differenziazione del prodotto e quindi la possibilità di dare vita a vantaggi competitivi. In funzione del vantaggio il produttore può conquistare una nicchia di mercato che assicura una certa libertà nella politica dei prezzi e dunque una buona possibilità di profitto.

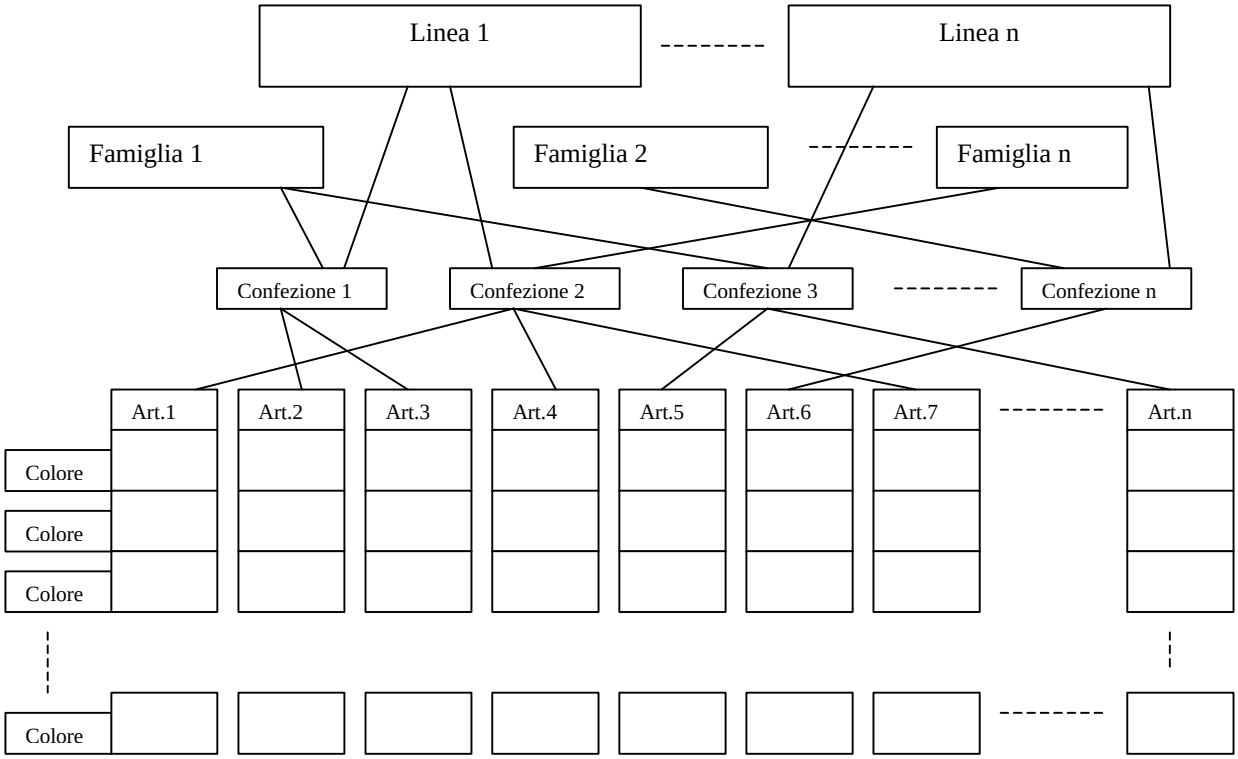

fig. 15 Il modello di classificazione degli articoli di campionario.

Il management della Boglietti deve determinare la politica di acquisto all’inizio di ogni stagione di vendita. Nel caso di un’azienda specializzata tale attività è molto rischiosa. Infatti, si ha la necessità di concentrare il più possibile gli acquisti di materie prime in modo da abbattere i costi di acquisto e rimanere competitivi sui prezzi finali. Quando l’azienda è specializzata deve acquistare semilavorati. L’acquisto di beni che hanno già subito una lavorazione riduce le capacità di utilizzo di tale prodotto in differenti processi produttivi. Nel caso dell’azienda integrata, avvengono acquisti di materie prime grezze, che possono subire differenti processi di lavorazione ed essere destinate a più prodotti finali. Questo rappresenta un vantaggio, perché durante la stagione di produzione il materiale può mutare destinazione al mutare della domanda del mercato.

Nel primo caso, dunque, la politica degli acquisti diventa un'attività più delicata, perché ha minori margini di intervento. Al fine di garantire una consegna tempestiva della merce ai clienti, la produzione deve iniziare la propria attività nei tempi più rapidi possibili. Questi ingredienti obbligano a scegliere il migliore *trade-off* tra la necessità di attendere, per avere una buona sensazione delle intenzioni del mercato, ovvero per avere una serie di dati sufficiente a fornire una previsione affidabile e tra la necessità di accorciare i tempi per dare inizio alla produzione.

Questa decisione è critica perché non è possibile adottare la soluzione di un acquisto superiore alla quantità suggerita dalla previsione. Se le materie prime eccedenti potessero essere riutilizzate nelle stagioni produttive seguenti, si potrebbe adottare tale semplice soluzione. Purtroppo le caratteristiche del mercato moda poc'anzi descritte rendono inutilizzabili le rimanenze di magazzino a fine stagione, comportando un minor profitto. Ciò ha un peso rilevante poiché il mercato moda possiede margini di profitto interessanti ed è quello su cui puntano le strategie aziendali di sviluppo.

Per affrontare la previsione degli ordini è necessario premettere una riflessione sulle caratteristiche dell'evoluzione dei trend stagionali, sulla struttura di vendita e sul campionario di prodotti venduti. Le variabili fondamentali che identificano un singolo prodotto sono cinque (fig. 15):

1. La **linea di produzione** suddivide il campionario in macro categorie di prodotti in base al tipo di mercato a cui si orienta. Si suddividono le linee donna, uomo e bambino. Differenzia anche i due mercati poc'anzi descritti: il mercato moda e quello tradizionale. Le linee di produzione rappresentano una suddivisione stabile nel tempo, poiché al variare delle caratteristiche dei prodotti le possibili macro-classificazioni immaginabili sono le stesse.
2. La **famiglia** del prodotto è una classificazione che si può ottenere aggregando tutti gli articoli nella cui produzione si usano materie prime assimilabili. La struttura delle famiglia è completamente stravolta di stagione in stagione, perché il campionario di prodotti è rielaborato in funzione delle tendenze della moda, con l'applicazione ad articoli esistenti di accessori che ne mutano la collocazione nel raggruppamento delle famiglie.
3. La **confezione** è una caratteristica che differenzia il prodotto sulla base del taglio di alcune sue parti o a seconda degli accessori che vengono applicati. Di fatto, la confezione rappresenta una sotto-aggregazione delle famiglie. Ogni famiglia può raggruppare al suo interno più confezioni. Rappresentando una sottoclasse delle famiglie, anche le confezioni non hanno una struttura stabile al mutare delle stagioni.
4. L'**articolo** è il nome che identifica ogni singolo prodotto del campionario. Come si può immaginare esistono innumerevoli articoli, molti dei quali vengono usati solamente per una sola stagione. La loro numerosità giustifica la necessità di classificarli nelle macro-classi appena elencate.

5. Il **colore** è un elemento che moltiplica il numero di articoli producendo una matrice di prodotti. Questa variabile agisce in modo trasversale rispetto alle classificazioni di cui sopra. Normalmente esistono colori dominanti (il bianco, il nero) nella vendita di linee di prodotto tradizionali, con una suddivisione interna piuttosto stabile lungo l'arco delle stagioni. Al contrario nelle linee "moda" emergono alcuni colori dominanti per una singola stagione, non ripetuti in quelle successive.

Dal punto di vista di chi che deve predisporre gli ordinativi di materie prime da utilizzare nella produzione devono essere tenuti presenti i diversi materiali di cui i prodotti necessitano. Per comprendere appieno il problema è utile un esempio. Molti prodotti vengono arricchiti con l'uso di un tessuto ornamentale, il cosiddetto "pizzo", il quale è acquistato in rotoli. Il pizzo è tagliato ed applicato sui semilavorati determinando la confezione e la famiglia alla quale il prodotto appartiene. La previsione sull'acquisto della materia prima "pizzo" deve essere effettuata rispetto alla variabile aggregata "famiglia", poiché nel corso della stagione produttiva può essere decisa di volta in volta la destinazione del "pizzo" ad un prodotto rispetto ad un altro.

Fino a questo punto è stata considerata unica variabile esogena a condizionare la struttura della domanda la tendenza della moda, in termini di capi e di colori preferiti. Attraverso l'analisi dei dati emerge una realtà non così semplice. Si può osservare che la domanda è spesso condizionata dalla forza di vendita. Gli agenti rappresentanti sono in grado di influire sulle decisioni di acquisto dei rivenditori in funzione della propria sensibilità e delle proprie necessità o interessi. Non bisogna dimenticare che l'agente svolge spesso il ruolo di consulente, in forza della sua conoscenza del mercato. Ciò induce l'acquirente ad adattare le proprie convinzioni ai suggerimenti di questi. L'adeguata analisi e comprensione dei meccanismi che regolano il comportamento della rete di vendita rappresenta un importante elemento ai fini di una corretta previsione, come osservato nel capitolo sulle catene di fornitura.

L'analisi sommaria dei dati ha fornito una situazione che può apparire paradossale: gli agenti che operano in zone geografiche limitrofe hanno una distribuzione percentuale delle linee vendute profondamente differente l'una dall'altra. Questo conferma l'ipotesi per cui l'intervento dell'agente influisce sulle decisioni di acquisto del dettagliante. Infatti, non possono essere addotte giustificazioni legate ai fattori culturali e geografici. Un elemento che rafforza questa idea è fornito dal tipo di mandato che lega gli agenti alla Boglietti: essi sono agenti plurimandatari. L'agente, oltre a negoziare i prodotti della Boglietti, tratta articoli complementari e non concorrenziali alla gamma da essa offerta. Ciò può provocare correlazioni tra i prodotti di biancheria intima e il resto dei campionari offerti dal singolo venditore e fa sì che questo favorisca le vendite di certi prodotti rispetto ad altri.

Il totale dei clienti serviti si attesta attorno alle 1800 unità. I clienti hanno caratteristiche eterogenee tra loro e sono distribuiti lungo tutto il territorio nazionale. Si spazia da negozi che trattano griffe e che si rivolgono ad una clientela sofisticata, fino ai punti vendita tradizionali,

come le mercerie e le piccole boutique, con un target di clientela segmentato sul fattore prezzo. Per una precisa scelta di immagine e di convenienza economica, la Boglietti non serve la grande distribuzione e, inoltre, tratta quasi esclusivamente il settore dell'intimo femminile.

Ogni esercizio è suddiviso in due stagioni che presentano tra loro una dinamica interna molto differenziata. Ogni anno, a febbraio, inizia la produzione della collezione estiva, seguita da quella invernale, che inizia a settembre. I due periodi produttivi non sono tra loro confrontabili in termini di previsione degli ordini, poiché le variabili determinanti sono diverse.

Spesso, la scelta di capi da acquistare da parte del rivenditore ricade su caratteristiche compatibili all'immagine che il punto vendita comunica al pubblico, poiché si rivolgono ad una clientela piuttosto diversificata nei gusti e nelle attese. Nella prima fase, gli agenti contattano i clienti più rappresentativi per quantità ordinate nel passato. I primi ordini raccolti forniscono le informazioni sui gusti del mercato nella stagione in corso. Sulla base di questi dati vengono decise le quantità di materie prime da acquistare. Sono pertanto i grandi clienti a saggiare le intenzioni del mercato e a condizionarlo. Successivamente vengono contattati i clienti più piccoli e l'ordinativo medio si riduce. La previsione si fonda sulla moltiplicazione delle quantità già ordinate per coefficienti che si differenziano in base alla variabilità presunta negli ordinativi della linea di prodotto. La produzione più tradizionale presenta una varianza più contenuta di quella innovativa.

7.2 Il modello di riclassificazione dei dati

Al fine di realizzare previsioni corrette è necessario valutare le caratteristiche dei dati disponibili. In riferimento all'azienda Boglietti, il sistema informativo, sede della maggior parte dell'informazione codificata, è integralmente orientato a soddisfare le esigenze delle attività svolte quotidianamente dalle diverse funzioni aziendali. Non esistono meccanismi negli strumenti di memorizzazione dei dati che predispongano i dati stessi per il supporto alle decisioni. Ciò si evince osservando come le classificazioni del campionario prodotti, sopra descritte, sono spesso manipolate al fine di adattarsi alle mutate esigenze della stagione produttiva. Nessuno strumento si occupa di riclassificare i dati in una metrica coerente, quale supporto all'attività di analisi successiva degli stessi.

Ad ogni stagione la definizione delle famiglie è completamente ridisegnata e ciascuna categoria assume un nome differente. Non esiste nessuna relazione codificata nei database che consenta di realizzare una corrispondenza tra famiglie omogenee appartenenti a due stagioni diverse. Al fine di comprendere l'evoluzione del mercato è necessario possedere una serie storica dei dati che sia confrontabile in quanto abbia elementi omogenei.

Il management Boglietti svolge l'attività di previsione utilizzando un mix di informazioni disponibili *on-line*, provenienti cioè dal sistema di memorizzazione dei dati, e di una ricca dose di

informazioni non codificate, ma distribuite nella conoscenza implicita del previsore, informazioni spesso di carattere qualitativo.

La previsione è dunque realizzata con un metodo non schematico e rigoroso. Il punto di partenza per disegnare una metodologia formale è l'utilizzo di strumenti che consentano di analizzare in modo efficace i dati. L'idea di costruire un modello di simulazione consente di estrarre potenzialmente alcune delle informazioni implicite e di includere nei meccanismi degli agenti la capacità di aggregare e riclassificare i dati. Di fatto, il modello deve svolgere l'attività che è comunemente detta di *on-line analytical processing (OLAP).*

In Chaudhuri, Dayal (1996) si definisce strumento OLAP un software in grado di riclassificare i dati, collezionati durante l'attività di produzione, secondo uno schema denominato a "cubi multidimensionali". La riclassificazione permette di realizzare numerose preaggregazioni di dati in modo da fornire al decisore la possibilità di interrogare il database producendo aggregazioni in tempo reale lungo alcune dimensioni del cubo (il tempo, le aree geografiche, le unità di vendita, ecc.). Attraverso la tecnica del *drill-down* lo strumento OLAP fornisce la possibilità di approfondire l'analisi lungo una dimensione, de-aggregando i dati. Le situazioni che difficilmente emergono ad un livello aggregato di analisi, si possono osservare con un procedimento di *drill-down* progressivo, ovvero ricercando le situazioni critiche ad un più basso livello di dettaglio.

Indagando con tale tecnica i dati di vendita, si può, ad esempio, scoprire che la *performance* di un particolare rivenditore o rappresentante è dissonante rispetto all'indicatore medio del settore e ne influenza il risultato complessivo.

L'uso degli strumenti OLAP nasce da un lato dall'esigenza dei manager di avere la situazione aziendale costantemente sotto controllo e dall'altro lato dallo sforzo delle aziende di formalizzare il più possibile la conoscenza, codificandola nelle architetture fornite dall'*information technology*, per superare la difficoltà nell'analisi a posteriori dei processi e dei dati.

Le tecnologie del *data warehousing* e del *on-line analytical processing (OLAP)* sono elementi essenziali nel supporto alle decisioni. Tale attività richiede caratteristiche diverse rispetto alla tecnologia di immagazzinamento dei dati tipica delle applicazioni utilizzate per le transazioni relative all'attività quotidiana di produzione. Sempre in Chaudhuri, Dayal (1996) si definisce *data warehouse* (magazzino di dati) un contenitore di informazioni "orientate al soggetto, integrate, variabili nel tempo, non volatili" usate primariamente per prendere decisioni all'interno delle organizzazioni. Generalmente il *data warehouse* è mantenuto separato dai database operativi dell'azienda. Ciò avviene per molte ragioni. Il *data warehouse* è utilizzato per fare processi analitici sui dati (OLAP), il database tradizionale si occupa di elaborare le transazioni (*OLTP, on-line transaction processing*). Quest'ultimo registra le operazioni che avvengono quotidianamente nell'azienda. Esse sono ripetitive e consistono in transazioni brevi ed atomiche. Il *data warehouse*

in contrasto necessita di dati aggregati e consolidati. Le operazioni caratteristiche di un processo OLAP includono il *rollup* (meccanismo di aggregazione progressiva dei dati, al crescere del punto di osservazione) e di *drill-down* (al crescere dell'esigenza di dettaglio, come processo di de-aggregazione).

Attraverso l'integrazione delle tecnologie informatiche nell'infrastruttura del modello ad agenti si può superare il limite strutturale delle tecniche OLAP. Esse infatti, nonostante impongano all'organizzazione un rigore formale nella gestione delle informazioni, delegano all'abilità del manager la capacità di saper analizzare i dati riconducendoli alle informazioni non codificate. In sostanza, il decisore ha a disposizione uno strumento potente di indagine, ma la ricerca dei fattori determinanti deve essere condotta sulla base delle sue "sensazioni".

Se si sceglie il modello di azienda virtuale è plausibile pensare di integrare nel software che determina il comportamento degli agenti i meccanismi che facciano emergere le situazioni critiche dal "basso". La ricerca di tipo *bottom-up* di tali punti critici è sicuramente più efficace di quella tradizionale, realizzata da un punto di vista aggregato (*top-down*).

Inoltre, è fondamentale notare che la previsione della stagione di vendita dipende non soltanto dalla struttura della domanda e dalle caratteristiche ambientali: ciò che influisce nella determinazione del modello è proprio la qualità e la struttura degli stessi.

Le tecniche statistiche di previsione presuppongono che i dati da analizzare siano pre-elaborati, al fine di renderli compatibili con le caratteristiche della tecnica previsiva utilizzata. Nel caso dei modelli ARIMA o delle reti neurali, ad esempio, spesso è consigliabile applicare un operatore alle differenze prime.

Non si può pensare di applicare in modo acritico tutte le tecniche di previsione ai dati posseduti, al fine di determinare quale dia i migliori risultati. Prevedere i dati significa possedere una giusta chiave di lettura della loro natura. La tecnica di previsione deve essere scelta in funzione del tipo di risultato che si vuole ottenere.

Nel caso specifico dei dati di vendita di un bene di consumo così mutevole come i capi moda di biancheria intima, le variabili esogene hanno un forte peso sulla domanda. Esse sono legate a fattori di scelta irrazionali e quindi i dati devono essere analizzati in uno schema che ne colga le componenti più stabili e consenta di osservare più chiaramente i fenomeni ricorrenti. Da ciò nasce l'idea di integrare le tecniche di previsione all'interno di un modello di simulazione che sia in grado di fornire i dati "riclassificati" in modo autonomo.

Il problema di tali modelli è, però, legato alla loro separazione logica dai dati reali dell'azienda. I modelli consentono di emulare in modo realistico le attività, i processi, i flussi di informazione codificata che quotidianamente si osservano all'interno della realtà aziendale. Il limite è rappresentato dai dati esterni al modello come la domanda, gli eventi ambientali, i blocchi di produzione causati da incidenti, i quali sono normalmente introdotti attraverso formule aleatorie in cui la distribuzione di probabilità è stimata sui dati rilevati nel passato. Esiste una

sensibile differenza tra l'utilizzo dei dati veri e propri e dei dati generati casualmente sulla base della struttura probabilistica del fenomeno. La previsione delle vendite dipende intimamente dalle rilevazioni dei dati reali e allo stesso tempo dall'influenza di fenomeni non quantificabili, ma riproducibili attraverso un modello basato su regole plausibili.

7.3 Lo schema del modello di simulazione

Il modello di azienda virtuale qui proposto ha lo scopo di estendere il concetto di modello aziendale elaborato da Lin, Tan e Shaw (1996) con l'introduzione dei dati reali attraverso il collegamento degli agenti software con i database aziendali.

In questo modo è possibile superare il limite di applicabilità di tali modelli ad esempi concreti ed è anche possibile ipotizzare di posizionare il modello di simulazione all'interno dell'infrastruttura informatica e quindi sfruttare le eventuali informazioni che giungono dalla rete distribuita, di cui l'azienda fa parte[4].

Questo elemento ha una notevole rilevanza. Infatti, la progettazione e la realizzazione di un modello che sia in grado di replicare tutti i processi e le attività che costituiscono una complessa catena di fornitura richiede uno sforzo elevato e dipende da troppe variabili. Il risultato che se ne può ottenere non può mai essere troppo specifico.

Se si riflette sul ruolo svolto dal database dell'azienda, si nota che esso possiede molte informazioni relative all'attività svolta dalle altre organizzazioni della catena del valore. Inoltre, se le aziende hanno una struttura condivisibile delle informazioni, il patrimonio informativo disponibile è ancora più completo. Sfruttando proprio il magazzino delle informazioni è possibile sostituire i blocchi rappresentati dalle unità esterne all'organizzazione con semplici agenti che ne replicano il comportamento attingendo ai dati passati.

Nella fig. 16 è rappresentato lo schema del modello integrato nella catena di fornitura. I suoi elementi fondamentali sono:

1. Il ***building-block*** che corrisponde ad un componente logicamente indipendente dell'azienda o, attraverso una gerarchia nidificata, all'intero complesso aziendale. La rete di fornitura o il mercato di vendita possono essere visti come singoli blocchi che producono flussi di informazioni. Ciascun blocco può contenere un modello di simulazione.
2. Le **tecnologie** che elaborano i dati e li rendono disponibili in forma codificata all'organizzazione. Sono gli elementi che rendono possibile la comunicazione da e verso l'esterno. Si raggruppano nelle due macro categorie: OLAP e OLTP[5]. All'interno

[4] A tal proposito si vedano anche i paradigmi NIIIP e BizTalk presentati nel capitolo precedente.
[5] Si ricorda che OLAP è uno strumento riclassificazione dei dati per il supporto alle decisioni, OLTP è l'insieme degli strumenti che si occupano di gestire i dati per l'attività quotidiana dell'azienda.

dell'OLTP si colloca generalmente l'infrastruttura che ospita le tecnologie dell'*e-business*. L'OLAP può essere presente come database separato. Se, viceversa, non è presente può essere sostituito dalle capacità del modello stesso di elaborare le informazioni.

3. Gli **agenti** costituiscono la struttura interna del modello. Essi sono i contenitori del software che costituisce il modello informatico di funzionamento dell'azienda. Al loro interno vengono collocate le regole di comportamento, gli algoritmi di previsione e la capacità di catturare l'informazione non codificata in forma esplicita, come effetto della loro interazione. Sono caratterizzati da una doppia natura: devono possedere la capacità di prelevare i dati dall'esterno ed eventualmente di produrli con l'applicazione di regole.

Il modello, costruito secondo la filosofia *Swarm*, è costituito da agenti che interagiscono replicando il comportamento delle entità fisiche a cui fanno riferimento. Alcuni di essi attraverso un'interfaccia universale che consente loro di collegarsi alla fonte dei dati operativi dell'azienda, prelevano le informazioni e le introducono nel modello. Gli agenti che, invece, rappresentano i processi di produzione sono introdotti come gli agenti fisici dei modelli MAIS[6], secondo la relazione $O=p(i)$.

Tale schema consente di focalizzare l'analisi dei problemi in modo selettivo. Infatti, se si vuole studiare l'effetto di una diversa struttura organizzativa o di una diversa politica di gestione è sufficiente dotare i soli agenti che sono interessati all'indagine di regole che modificano il loro comportamento rispetto al passato. In tal caso l'agente interrompe la comunicazione con il database aziendale e produce i dati internamente.

L'azienda virtuale è un modello molto flessibile le cui caratteristiche di indagine non sono univocamente definite. La sua capacità analitica deve nascere da un progetto a passi successivi. In una prima fare la progettazione si occupa della riproduzione il più possibile fedele del sistema a cui fa riferimento. Successivamente si aggiungono livelli più astratti che svolgono tipologie di indagine diversificate, a diversi gradi di approfondimento. Ciò implica la necessità che il disegno del modello sia realizzato totalmente a componenti, per consentire di adattarsi alle più diversificate esigenze di analisi.

[6]Si ricorda che nello schema MAIS gli agenti fisici includono metodi che emulano i processi aziendali in termini di costi e tempi. Si veda a tal proposito il paragrafo 6.3.

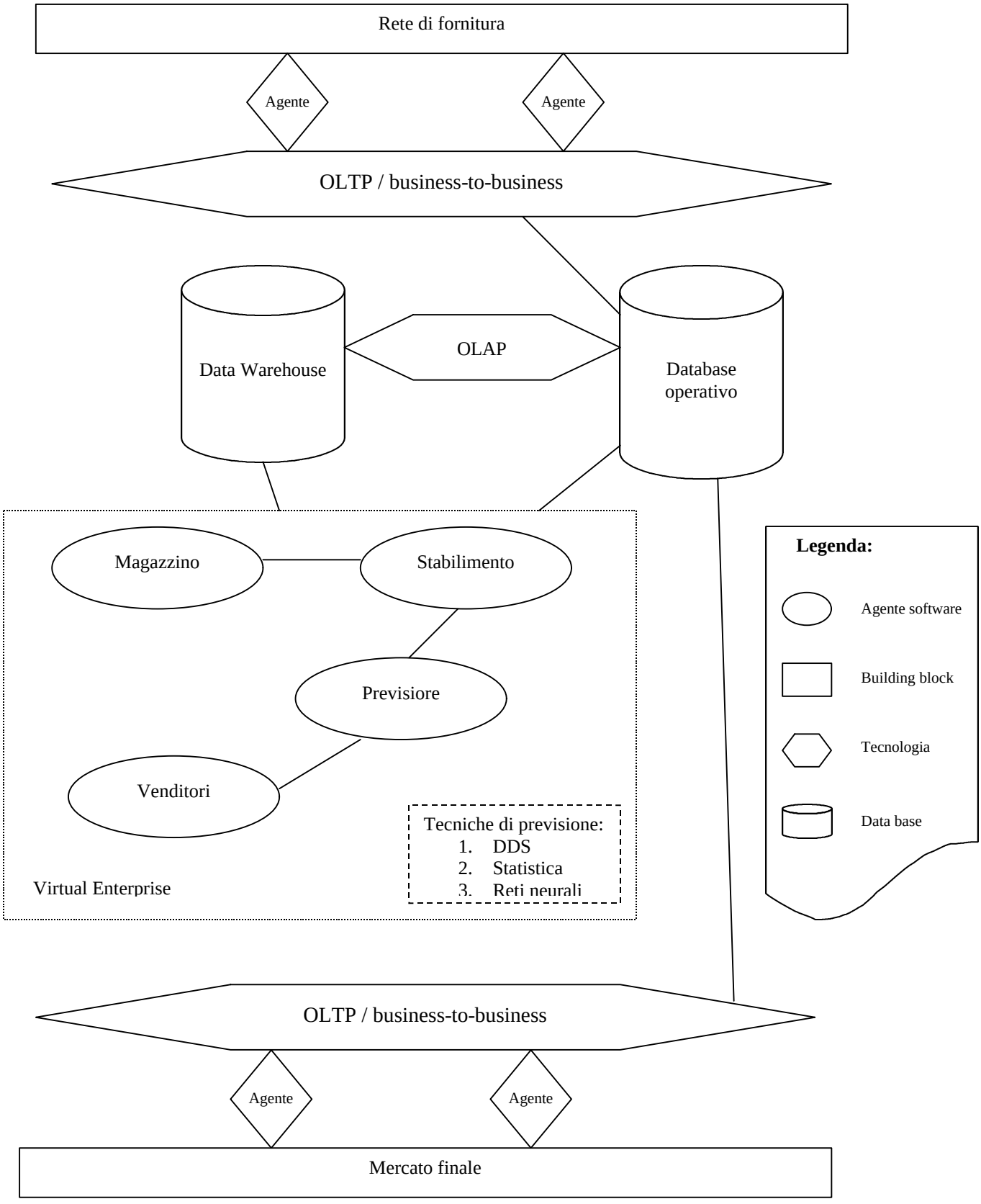

fig. 16 Schema di una Virtual Enterprise.

7.4 La rappresentazione a strati sovrapposti

Gli agenti software, che interpretano il ruolo di analoghi agenti reali, sono dotati di una duplice natura. La prima consiste nella capacità di decodificare le informazioni provenienti dai database aziendali e di introdurle nel modello di simulazione. La seconda è legata alla tradizionale interpretazione di agente software. Sono, cioè, dotati di meccanismi di elaborazione delle informazioni e hanno la capacità di modificare il proprio comportamento in funzione di tali informazioni. I dati relativi ai due piani di operatività sono tenuti rigorosamente distinti.

Un agente di questo tipo non è una banale estensione di un tradizionale agente privo di collegamento con i dati reali. In questo caso si impone l'abbandono delle congetture sui meccanismi comportamentali degli attori del sistema. Infatti, nel momento in cui l'agente riceve le informazioni e le trasmette al modello svolge una semplice funzione di trasferimento. Il suo comportamento non è per nulla spiegato dal modello, ma ha la massima plausibilità, poiché si comporta come si sono comportati nella realtà gli agenti che vuole descrivere.

Da ciò il modello dell'azienda virtuale costituisce una diverso metodo di indagine nella simulazione ad agenti. Porta inevitabilmente a spostare l'analisi dei meccanismi cognitivi ed interattivi delle realtà economiche, verso la ricerca empirica di soluzioni aziendali.

La rappresentazione a strati sovrapposti, rappresentata in fig. 17, garantisce la possibilità di avere un modello che emula ciò che è avvenuto realmente nell'organizzazione di riferimento, senza introdurre nessun tipo di ipotesi descrittiva. A questo modello che replica la realtà si sovrappone il secondo strato. Esso consente di fornire ipotesi sull'effetto che talune modificazioni nel modello comportano in aggregato, di simulare, di provare soluzioni, di analizzare i dati e proporre variazioni.

Ci si propone di costruire uno strato che rappresenti l'azienda e nel quale gli agenti siano dotati di semplici regole ed un altro stato che svolga il ruolo di simulazione in modo del tutto indipendente e che eventualmente fornisca i dati al modello sottostante attraverso degli appositi "punti di contatto".

Nella base dei dati Boglietti non è registrata l'informazione relativa alla previsione che il management ha realizzato, ma questa informazione è necessaria al modello perché il blocco che rappresenta la rete di vendita deve comunicare come *output* la quantità di beni da mettere in produzione per sostenere la successiva attività di fornitura. In tal caso il modello base deve essere dotato della capacità di fornire una rudimentale previsione sulla base di una formula di calcolo.

Poiché la capacità di realizzare previsioni affidabili nasce dall'uso di tecniche sofisticate all'interno degli agenti, il meccanismo che realizza le previsioni è introdotto nel secondo strato, ovvero nel modello di simulazione. Attraverso opportuni punti di contatto è possibile imporre che il modello di simulazione introduca i dati che ha prodotto durante l'esecuzione nel magazzino dei

dati degli agenti del modello base, sostituendo di fatto la previsione che l'agente privo di meccanismi sofisticati aveva realizzato.

Questa impostazione rende estremamente flessibile il progetto, perché il modello base pur essendo dotato di scarso interesse dal punto di vista della simulazione fornisce gli elementi infrastrutturali fondamentali. Con la semplice introduzione di meccanismi che da uno strato più elevato di astrazione modifichino i dati ed, eventualmente, le regole degli agenti appartenenti al modello base, si ottengono diverse configurazioni e possibilità di indagine con una struttura che fa sempre e comunque riferimento ai dati reali, quando questi sono disponibili.

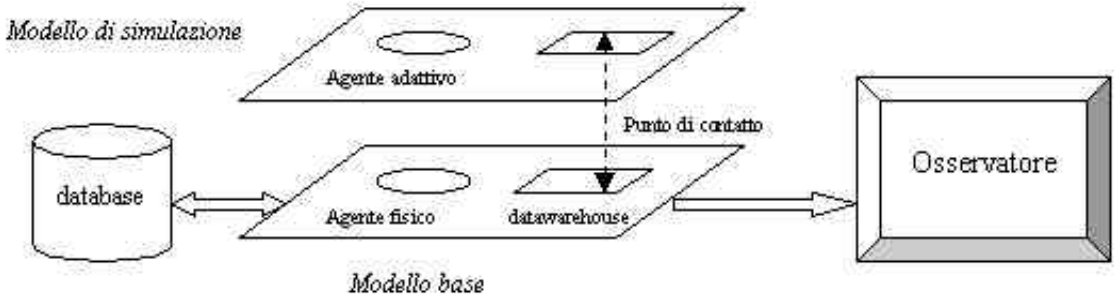

fig. 17 Lo schema a doppio strato del modello di azienda virtuale

7.5 Il modello Boglietti

Lo schema è stato applicato al problema in esame attraverso la costruzione di un modello basato su *Swarm* e attraverso l'uso della tecnologia ODBC[7], la quale ha lo scopo di fornire il metodo di accesso ai database.

Il modello base è costituito da un unico *building block* che rappresenta il settore dell'azienda che si occupa di raccogliere gli ordini e di decidere le quantità di beni da produrre, ovvero il *marketing* aziendale. Esso è costituito da tre categorie di agenti: i rivenditori, i rappresentanti ed il previsore.

I rivenditori rappresentano i negozi e le *boutique* che ordinano le forniture per tutta la stagione. La loro collocazione all'interno del blocco *marketing* è dovuto alla loro funzione di strato di *input*. Il modello che ne descrive dettagliatamente la natura dovrebbe essere collocato nel blocco che rappresenta il mercato finale della Boglietti. In questo caso essi costituiscono l'elemento di contatto che collega i due blocchi ed è fondato sui dati prelevati direttamente dal database.

[7] *Open data-base connectivity* è una tecnologia introdotta da Microsoft come strato software che attraverso il linguaggio SQL consente di interagire con qualsiasi formato di database che sia dotato di un *driver* compatibile con ODBC.

I rappresentanti sono agenti molto semplici: essi raccolgono gli ordini e li trasmettono all'agente previsore. Nella realtà essi svolgono un ruolo che influenza la struttura della domanda, pertanto i fattori che ne determinano il comportamento dovrebbero essere descritti e riprodotti nel loro funzionamento. La filosofia di azienda virtuale consente di evitare questo compito, spostando la ricerca sul solo obiettivo di costruire un meccanismo che riesce a prevedere gli ordini, attraverso l'interpretazione implicita dei fenomeni che influenzano la domanda.

Il previsore è l'agente che deve possedere la capacità di interpretare questi fattori e fornire i valori di *output* del blocco: la stima delle quantità di beni da produrre fino a fine stagione.

In realtà per migliorare il livello di affidabilità delle previsioni, il meccanismo di previsione è integrato negli agenti rappresentanti. Il previsore, contrariamente al suo nome, si occupa soltanto di raccogliere ed aggregare le previsioni fornite da questi ultimi.

Ciò è stato realizzato per sfruttare la capacità dei modelli di realizzare meccanismi di analisi dal "basso", cioè localizzati ad un livello di dettaglio inferiore di quanto non possa fare un organo centralizzato.

La struttura logica del blocco è diversa dalla sua realizzazione pratica, ma è necessario fare sempre riferimento alla prima per comprendere la struttura del progetto.

Accanto a questo *building block* sarebbe interessante affiancarne uno in grado di rappresentare lo stabilimento di produzione ed uno che rappresenti il mercato di fornitura, al fine di osservare la dinamica del livello delle scorte. Data la finalità previsiva di questo modello e la mancanza dei dati relativi, questo elemento è stato volutamente omesso.

Gli agenti che comunicano con il database posseggono un oggetto denominato *datawarehouse* il quale rappresenta una semplice versione del *data warehouse* costruito con gli strumenti OLAP tradizionali e contiene le informazioni lette dall'ambiente ed elaborate al fine di renderle omogenee e funzionali per gli scopi di analisi dei dati. Le vendite, ad esempio, sono raggruppate per settimana e per colore o per linea e sono gestite in modo distinto per ogni singolo agente rappresentante.

Si potrebbe ipotizzare di introdurre un meccanismo che registri in un database esterno queste informazioni trasformando il modello in un semplice e rudimentale strumento di analisi *on-line* dei dati.

Data la struttura degli agenti rivenditori, priva di qualsiasi regola, il modello è strutturato in modo da replicare gli avvenimenti passati. Ciò avviene perché la previsione si può soltanto fondare sui dati passati. Nell'ipotesi di introduzione del secondo blocco nel modello sarebbe interessante effettuare una stima di giacenza del magazzino a fine stagione. In tal caso gli agenti rivenditori dovrebbero possedere una regola per fornire i dati che il database non possiede, esattamente come nei modelli MAIS. Al limite dovrebbe essere costruito il relativo blocco.

Attraverso il modello che si collega con la realtà aziendale è possibile eseguire la simulazione quotidianamente, senza la necessità di prelevare manualmente i dati aziendali dal loro

"deposito", aggregarli ed applicare su di essi le tecniche di previsione. Le previsioni possono essere aggiornate settimanalmente con la semplice ri-esecuzione del modello che possiede un numero sempre maggiore di informazioni, prelevandole autonomamente dal database aziendale.

7.6 Le tecniche di previsione utilizzate

Nel modello è stato scelto l'uso delle reti neurali come tecnica di previsione, per le caratteristiche descritte nel capitolo ad esse dedicato. L'agente rappresentante possiede la doppia interfaccia poc'anzi descritta. Nel modello base raccoglie gli ordini dei clienti ed è dotato di una banale regola di previsione che si fonda su una relazione lineare con i dati rilevati nella stagione precedente. Con questa semplice regola fornisce al previsore una stima "grezza" delle vendite a fine stagione.

È dotato, però, di una interfaccia nel modello di simulazione attraverso la quale apprende ed adatta la propria struttura ai dati che giungono nell'apposito magazzino (*datawarehouse*), con una regola fondata sulle funzioni a rete neurale.

Esiste un doppio punto di contatto. In una prima fase l'agente del modello di simulazione legge i dati contenuti nel *datawarehouse* dell'agente del modello base e dopo aver elaborato una previsione attraverso la rete neurale li modifica opportunamente.

In questo modo il modello base continua a funzionare utilizzando i dati previsti dalla rete neurale senza "esserne a conoscenza". Il meccanismo del modello di simulazione introduce in modo trasparente i risultati nel modello base.

Questo schema consente di inserire altre tecniche di previsione senza dover modificare la struttura fin qui delineata. Infatti è sufficiente creare un altro modello di simulazione da sovrapporre o sostituire a quello fondato sulle reti neurali per ottenere un nuovo "simulatore" basato su altre tecniche previsive.

Il modello base usa un sistema di previsione che si fonda intimamente sui dati rilevati nella stagione precedente, secondo l'ipotesi di stabilità dei flussi di ordini.

La previsione si ottiene moltiplicando la congettura sul numero di clienti che si ipotizza di contattare fino a fine stagione ed il numero di capi ordinati per cliente contattato.

Sia s_i il totale di capi venduti durante la settimana i, sia $c_{a,i}$ il numero di clienti contattati nella stagione a, durante la settimana i, la previsione si ottiene con la seguente

$$ {}_{a}P_t^T = \sum_{i=0}^{t} s_i + \frac{\sum_{i=0}^{t} s_i}{\sum_{i=0}^{t} c_{a,i}} \cdot \sum_{i=t+1}^{T} c_{a-1,i} \cdot \frac{\sum_{i=0}^{t} c_{a,i}}{\sum_{i=0}^{t} c_{a-1,i}}, \qquad (24) $$

la quale è costituita da una componente rappresentata dai capi venduti fino alla settimana t, alla quale si somma il prodotto il numero di capi per cliente misurato fino al tempo t ed in numero di clienti da contattare fino a fine stagione. Questo elemento a sua volta è il prodotto tra il numero di

clienti contattati nella stagione precedente nelle settimane successive alla *t*, ponderato per un coefficiente che esprime il rapporto tra i clienti contattati nella stagione corrente e quelli contattati nella precedente. La (24) si può anche esprimere con la più sintetica

$$ {}_{a}P_{t}^{T} = \sum_{i=0}^{t} s_i \cdot \left(1 + \frac{\sum_{i=t+1}^{T} c_{a-1,i}}{\sum_{i=0}^{t} c_{a-1,i}} \right). $$

Il difetto fondamentale di questa tecnica è l'ipotesi forte che il numero di capi per cliente sia costante durante tutto l'arco della stagione e che il numero di clienti contattati abbia una struttura proporzionale.

La rete neurale non richiede che si elabori nessuna funzione ben determinata per effettuare questo calcolo. Essa adatta una generica funzione non lineare a molti gradi di libertà alla struttura dei dati. Ciò che è importante nella definizione della struttura dei dati da usare come *pattern* di apprendimento è che questi siano dinamici. Si deve fare in modo che il valore di *output* atteso non sia costante per tutta la stagione e, quindi, si chiede che la funzione effettui una congettura sui dati relativi alla sola componente futura. I capi venduti fino alla settimana *t* di previsione sono sommati al dato previsto dalla funzione. La struttura dei dati è rappresentata in Tabella 2.

Tabella 2 Struttura dei *pattern* di input e di output usati nelle funzioni a rete neurale

INPUT	*OUTPUT*
Numero della settimana	Numero clienti dalla settimana *t+1* fino a *T*
Numero della linea di prodotto	Numero capi / cliente degli ordini relativi al periodo che va da *t+1* a *T*
Numero clienti contattati nella settimana *t*	
Totale capi venduti dalla settimana *0* a *t*	

La scelta delle reti neurali è dovuta alla possibilità di evitare la definizione di un'ipotesi sulla dinamica dei dati, con la sola necessità di effettuare diversi tentativi per trovare una buona configurazione dei parametri di apprendimento e dei dati su cui poggiare le previsioni.

Nel capitolo conclusivo saranno affrontati i risultati e le riflessioni sul comportamento di questa tecnica di previsione integrata in un modello di simulazione e sarà confortata con gli altri metodi.

Capitolo 8 jBoglietti Swarm

8.1 Introduzione

In questo capitolo sono analizzate nel dettaglio la struttura, le caratteristiche e le tecnologie impiegate nel modello. *jBoglietti Swarm* è un "simulatore" scritto con il linguaggio di programmazione *Java*, secondo la filosofia progettuale *Swarm* e attraverso l'uso delle librerie da questo messe a disposizione. Il software esegue la simulazione di una stagione di vendita dell'azienda Boglietti, utilizzando dati prelevati da un database in formato Microsoft Access.

I dati dell'azienda non sono registrati in questo formato, ma essendo pubblicati in formato testuale, si è optato per l'uso di un software che possiede una buona capacità di elaborazione (*query*). Il formato Access non è una scelta obbligata, in ragione del fatto che i dati possono derivare da qualsiasi altro formato, purché sia compatibile con lo standard ODBC.

Per esigenze di "trasportabilità" da un computer all'altro il software è in grado di leggere in alternativa i dati da un gruppo di file. Infatti, il database occupa molto spazio (circa 45 MB) e, dunque, utilizzare i file con i dati già aggregati e opportunamente predisposti consente la possibilità di eseguire il software su qualsiasi computer e in qualsiasi sistema operativo.

Questa scelta deriva dal fatto che *jBoglietti Swarm* è un'applicazione di carattere scientifico e deve poter essere riprodotto senza l'uso del database, come software *stand-alone*.

8.2 La scelta del linguaggio Java

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, 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 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* è un linguaggio 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.

Dovendo affrontare in questa sede una problematica di simulazione legata al mondo aziendale, la scelta di Java permette sfruttare più agevolmente le risorse dell'azienda, poiché ha la capacità nativa di collegarsi al database. Quindi, la motivazione che ha determinato la scelta di Java per questo lavoro è legata alle *API*[8] di accesso ai dati, di cui è dotato.

In *jBogliettiSwarm* si deve gestire l'approvvigionamento di moltissimi dati agli agenti del modello: basti pensare che l'intera tabella che contiene gli ordini relativi a tre stagioni contiene circa 250.000 righe d'ordine. Questi dati devono essere anche aggregati e pre-elaborati prima di essere trasmessi agli oggetti software. Per questo non è pensabile di poter gestire tutta la problematica di accesso ai dati con le sole funzioni di *input/output* da file fornite con il linguaggio *Objective-C*. Inoltre, nella seconda ipotesi si sarebbe dovuto abbandonare l'interessante possibilità di eseguire il modello sfruttando i dati disponibili in linea senza doverli pre-elaborare.

È utile ricordare che Swarm mette a disposizione una libreria di gestione dei dati denominata HDF5[9], 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.

[8] API (*application program interface*) è una sigla che indica l'insieme delle funzioni che un linguaggio, una libreria o un sistema operativo mettono a disposizione del programmatore.
[9] HDF è prodotto dal NCSA (*National Center for Supercomputing Applications*, http://hdf.ncsa.uiuc.edu/)

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[10].

Il bilancio è rivolto a favore di questo linguaggio per la sua maggiore prospettiva di evoluzione, la capacità di gestire oggetti come le stringhe di caratteri con estrema semplicità, la capcità 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.

8.3 Lo schema E.R.A.

Il modello *jBogliettiSwarm* è realizzato con la stessa filosofia progettuale del modello *jAkerlof* presentato nel capitolo 5, ma il numero delle righe di codice, la quantità di classi, la complessità degli algoritmi utilizzati richiedono una maggiore attenzione nel progetto dello schema teorico e nella definizione dei componenti per evitare di incorrere nell'errore di costruire un complesso di istruzioni troppo sofisticato da manutenere, da modificare, da riutilizzare e da verificare durante il *debug*.

In Gilbert, Terna (1999) si delinea un utile metodo di descrizione dei modelli *Swarm* come dagli stessi autori è osservato:

> The problems arising when we go from simple models to complex results suggest that a crucial role for the usefulness and the acceptability of the experiments is played by the structure of the underlying models. For this reason, we introduce here a general scheme that can be employed in building agent-based simulations. Similar schemes are advocated and implemented in Moss (1998) and Gilbert and Troitzsch (1999).

La struttura originale dello schema *Environment-Rules-Agents*. proposto dagli autori è descritto in fig. 18. 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. La sezione *Rules* contiene proprio gli oggetti in grado di applicare le regole di comportamento degli agenti ed eventualmente di modificarle. 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.

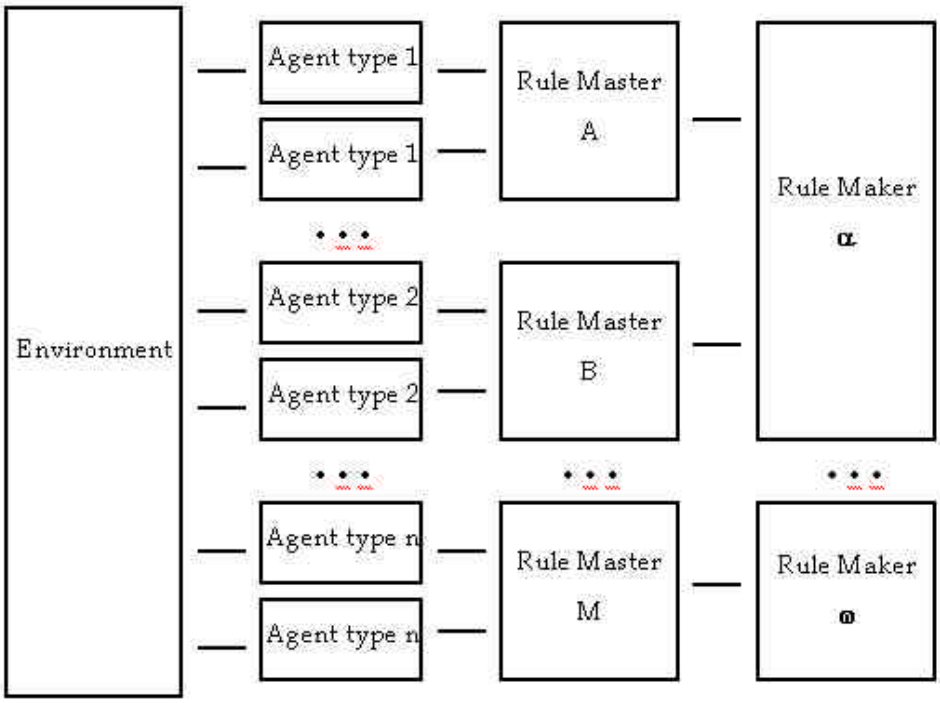

fig. 18 Lo schema Environment-Rules-Agents per costruire modelli basati su agenti

Questo schema di progetto produce vantaggi enormi nella manutenzione e nella comprensibilità del codice. Per questa ragione il modello *jBogliettiSwarm* è stato descritto e realizzato secondo questa tecnica. Il risultato è osservabile in fig. 19.

[10] A tal proposito si veda l'interessante progetto *SWIEE*, (http://swiee.econ.unito.it)

Lo schema del modello base (*BaseModel*) appare molto più complesso del modello di previsione. In realtà, sebbene quest'ultimo compia poche operazioni, il suo livello di complessità concettuale e realizzativa è assai superiore.

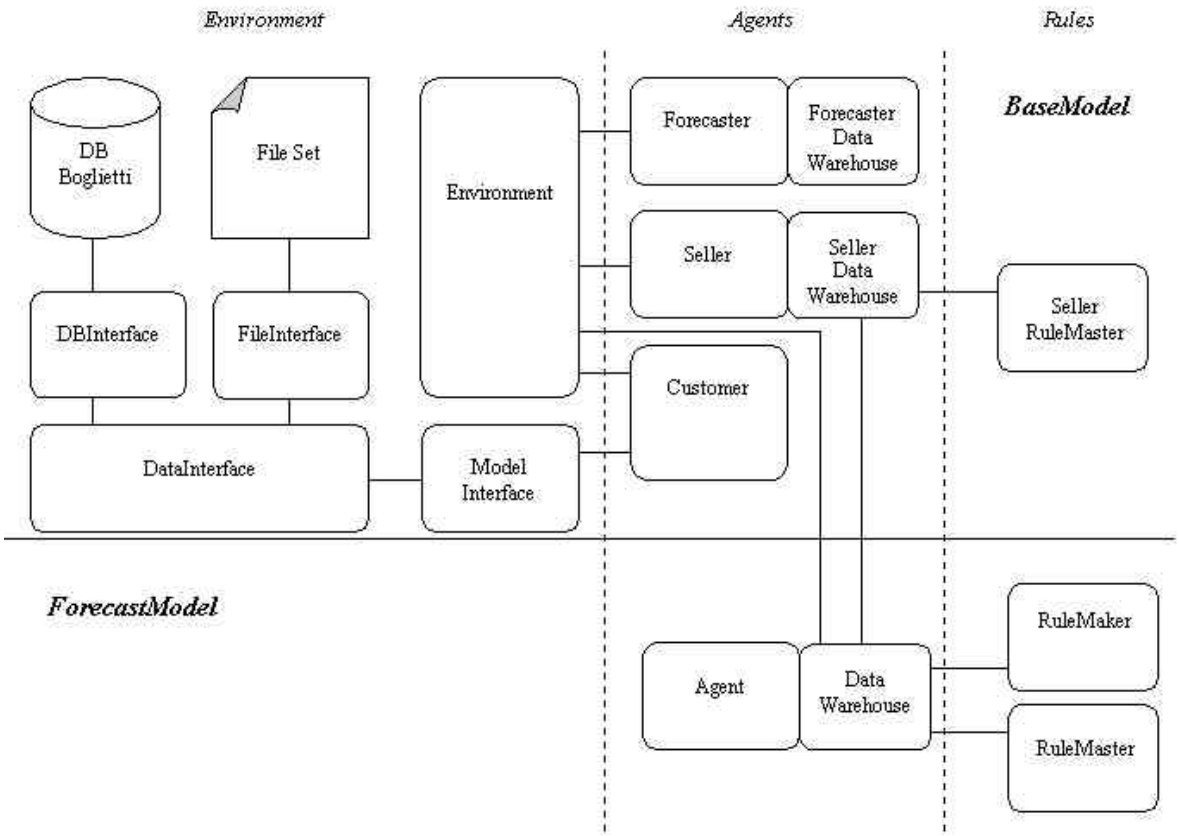

fig. 19 Lo schema E.R.A. del modello jBoglietti Swarm

8.4 Il modello base

Il percorso che le informazioni compiono dal database fino al modello è assai articolato. Come è stato accennato in precedenza, il modello è in grado di utilizzare i dati provenienti dal database o da un file che li contiene in forma aggregata per ridurne le dimensioni. La doppia sorgente dei dati è stata unificata sfruttando l'elegante meccanismo dell'ereditarietà fornito dalla programmazione ad oggetti. È definita una classe astratta *DataInterface* che contiene tutti metodi necessari al colloquio tra il modello ed il mondo esterno. Questi metodi però non contengono il codice necessario alla lettura dei dati, sono semplici definizioni dei messaggi ai quali l'oggetto deve saper rispondere. Nella fase di esecuzione del modello, l'utente sceglie quale sorgente dei dati usare selezionando l'opportuno parametro *readFromFile*. In base a questo valore l'oggetto *ModelSwarm* che costruisce gli oggetti per la simulazione crea un oggetto di tipo *DBInterface* o *FileInterface*, i quali ereditano le caratteristiche della classe "padre" *DataInterface*. Gli agenti del

modello chiamano il metodo della classe *DataInterface,* non conoscendo quale dei due oggetti (*DBInterface* o *FileInterface*) risponderà alla richiesta.

I dati provenienti da un qualsiasi oggetto che fa parte della gerarchia di classi *DataInterface* sono filtrati dalla classe *ModelInterface*, la quale crea una mappa di corrispondenze tra i nomi dei colori, degli agenti rappresentanti, delle linee di prodotti e tra il relativo numero indice usato nelle matrici che memorizzano le informazioni all'interno del modello. Attraverso questo metodo è possibile rappresentare le vendite di tutte le linee di prodotto con un vettore le cui posizioni corrispondono ad una descrizione testuale comprensibile all'utente. La corrispondenza è garantita dalle mappe contenute nell'oggetto *ModelInteface*.

I dati così filtrati sono trasferiti nella memoria temporanea degli agenti che rappresentano i rivenditori di biancheria intima (*customer*), i quali attraverso l'oggetto *Environment,* che rappresenta l'ambiente secondo le specifiche dello schema E.R.A., garantisce il trasferimento degli ordini ai rappresentanti (*seller*).

Per comprendere in modo chiaro il funzionamento del modello e le interazioni tra gli agenti è necessario osservare la struttura degli orologi che gestiscono la sequenza delle operazioni.

8.5 Lo schedule del modello

Dal punto di vista aggregato la scansione del tempo avviene in due passi, tra i quali è garantita "l'intrusione" del modello di simulazione (*ForecastModel*). Nella fig. 20 si può osservare lo schema che garantisce la scansione degli eventi.

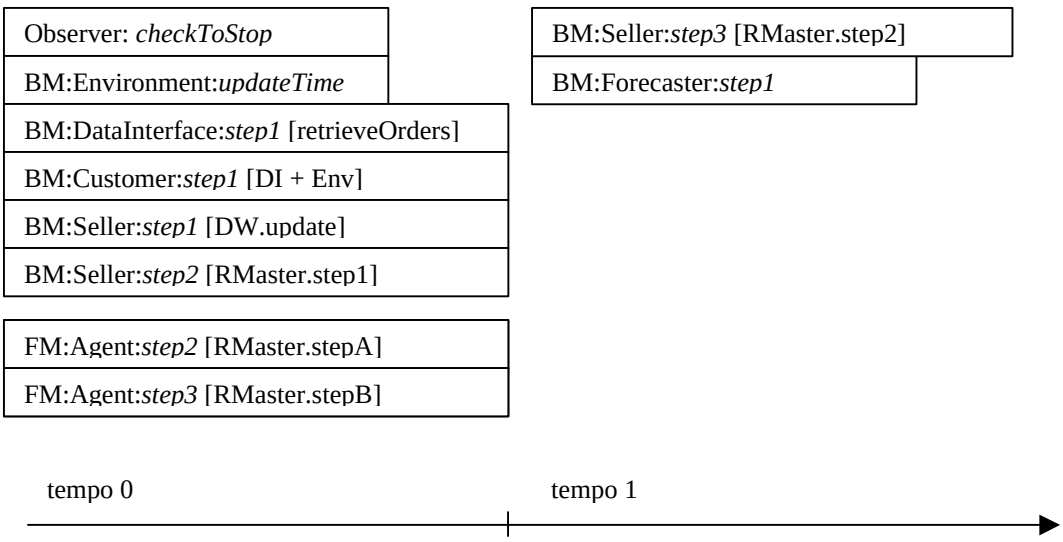

fig. 20 Lo schema sequenziale degli eventi generati dallo *schedule* del modello

Impostando le azione al tempo 0 in ogni modello si possono strutturare le sequenze di eventi in modo completamente centralizzato senza la necessità di modificare tutti i modelli. In questo caso, pur essendo elencate nella stessa funzione, le azioni del modello base sono scandite al tempo 0 e al tempo 1. In questo modo qualsiasi altro modello che inserica eventi al tempo 0 provocherà l'esecuzione delle azioni prima di quelle previste dal modello base al tempo 1. Questo meccanismo esalta la potenza e la flessibilità di *Swarm* nel realizzare soluzioni che concettualmente non sono affatto banali.

Dall'elenco delle operazioni si può comprendere la totalità della struttura della simulazione. Il modello base notifica all'oggetto *DataInterface* di approvvigionarsi dei dati necessari all'esecuzione di un ciclo di simulazione con il metodo *retrieveOrders*, dopo essersi assicurato che l'oggetto *Environment* abbia aggiornato il proprio orologio interno che simula lo scorrere delle settimane.

A tutti gli oggetti rivenditori è richiesto di "fare un passo" (*step*), nel senso generico di compiere un blocco di operazioni da eseguire in sequenza. In questo caso ciascun *customer* richiede i dati che lo interessano all'oggetto *DataInteface* e contestualmente li invia all'agente rappresentante (*seller*) in base all'informazione contenuta nella riga d'ordine. Il cliente non è rigidamente legato ad un rappresentante, ma seleziona quello corretto in base all'informazione contenuta nel database, al fine di riprodurre la realtà.

A questo punto gli agenti seller eseguono due passi. Nel primo aggiornano i dati nel loro personale magazzino, eseguendo un'operazione del tipo OLAP, attraverso il metodo *stepEndWeek*. Ad ogni ordine ricevuto il dati sono raggruppati progressivamente, al termine della settimana il magazzino dei dati è riorganizzato. Nel secondo passo il *seller* svolge una tipica operazione dello schema E.R.A.: richiama un opportuno metodo dell'oggetto *SellerRuleMaster* che si occupa di determinarne il comportamento. In questo caso l'applicatore di regole svolge le operazioni previsione delle vendite a fine stagione utilizzando l'algoritmo di calcolo descritto nel capitolo precedente.

Ma se il modello di simulazione (*ForecastModel*) è attivo, entra in funzione facendo svolgere all'agente due operazioni. Nella prima è richiamato il *RuleMaster* che applica la rete neurale per ottenere una previsione. Nella seconda il *RuleMaster* interroga l'oggetto *Environment* (metodo *isTheForecastTime*) per sapere se è il momento di effettuare l'apprendimento della rete da parte dell'oggetto *RuleMaker*.

Dopo queste operazioni il controllo ripassa al modello base, il quale chiama il metodo *step3* dell'agente rivenditore. Il funzionamento di tale metodo è analogo al secondo *step* di *SellerRuleMaster*. Infatti, se è giunta la settimana in cui eseguire l'apprendimento, costruisce l'insieme dei file che contengono le informazioni della stagione trascorsa al fine di utilizzarli nelle previsioni di quella successiva.

L'ultima operazione consiste nella chiamata all'oggetto *Forecaster*, il quale aggrega i dati di previsione e li ripone nel magazzino dei dati (*ForecasterDataWarehouse*). Da questo magazzino l'osservatore preleva le informazioni da mostrare all'utente.

8.6 Il modello bp-ct

Si è già detto che il modello di previsione è fondato sulle reti neurali, grazie al paradigma della programmazione ad oggetti spesso il codice scritto da altre persone può essere facilemente riutilizzato per scopi differenti. In questo caso è stato sfruttato un modello scritto in *Swarm* da Terna (1999) e chiamato *bp-ct*. Il modello, pensato secondo lo schema E.R.A., è un modello generico e flessibile che permette di effettuare diversi compiti: l'apprendimento di reti neurali con l'algoritmo di *backpropagation*[11] (BP), come semplice strumento di calcolo, la costruzione di un modello fondato su agenti dotati di una rete neurale, con la modifica di parte del codice, ed infine permette di costruire i *pattern* di apprendimento sfruttando la costruzione interna di regole con la tecnica dei *cross-target* (CT), proposta dallo stesso autore.

Come fa notare l'autore

> We introduce here the bp-ct code as a package useful to develop and run ANNs in the Swarm framework. A complete "How to use bp-ct" explanation is included as a text file in the distribution of the package.

Purtroppo il modello è scritto con *Objective-C* e ciò costituisce un problema considerando che il modello *jBogliettiSwarm* è realizzato in *Java*. Esistevano due soluzioni per utilizzare *bp-ct*. La prima consisteva in una soluzione un po' artigianale, poiché il modello avrebbe dovuto produrre i dati per l'apprendimento in opportuni file che poi sarebbero stati usati da *bp-ct* per eseguire l'apprendimento, ma si sarebbe dovuto scrivere il codice relativo al calcolo interattivo della rete per eseguire le previsioni nel modello. In questo caso, *bp-ct* sarebbe stato usato secondo il primo metodo descritto dettagliatamente nella guida del *software* di cui si fa riferimento in Terna (1999).

La seconda soluzione, quella utilizzata, consisteva nella traduzione del codice sorgente originale *bp-ct* in una versione *Java*. Questa soluzione, seppure costosa in termini di tempo e risorse, ha permesso di modificare parte del codice per consentire di costruire la matrice dei dati di apprendimento in modo interattivo, senza l'uso dei file esterni, e, soprattutto, ha reso il risultato molto più elegante. Esiste, quindi, una versione semplicemente tradotta e denominata *jBPCT* ed una versione adattata per essere integrata nel modello *jBoglietti Swarm* e che è rappresentata dal cosiddetto *ForecastModel*.

Gli interventi sul *package* hanno riguardato pochi e specifici elementi:

[11] Si veda a tal proposito il capitolo 2.

- Durante la simulazione i dati su cui base la previsione sono costruiti in tempo reale, *bp-ct* è in grado di eseguire il calcolo dei valori di *output* su una matrice di verificazione. Al fine di sfruttare la capacità di previsione della rete è stato scritto un metodo nel *RuleMaster* detto *fillVerificationTrainingMatrix* che ha la funzione di punto di contatto con il modello base, poiché preleva i dati dal magazzino degli agenti rappresentanti e li usa per costruire la matrice di verificazione. Dopo questa operazione il modello continua a funzionare in modo tradizionale, producendo i risultati.
- Dopo l'esecuzione del calcolo il *RuleMaster* chiama il nuovo metodo *copyOutputVerificationToBaseModel* il quale rappresenta il secondo punto di contatto. Esso sovrascrive i risultati di previsione forniti dall'oggetto *SellerRuleMaster* del modello base.
- Al termine della stagione il *ForecastModel* deve eseguire l'apprendimento delle reti neurali. La matrice di apprendimento è costruita dal modello stesso durante la simulazione. Al temine, quando si conoscono i risultati finali della stagione è possibile calcolare i valori di *target* per l'apprendimento. Il metodo *makeTargetsForLearning* dell'oggetto *RuleMaker*, compie questa operazione prima di eseguire l'apprendimento. Nel modello "classico" *bp-ct* la matrice di apprendimento è prelevata da un file che deve essere opportunamente predisposto dall'utente. In questo caso è necessario che il modello di previsione sia in grado di interagire opportunamente con il modello base e sappia costruire la matrice di *training*.
- L'algoritmo di apprendimento delle reti neurali richiede che i dati siano scalati in un intervallo di valori continui compreso tra 0 e 1. Ciò si realizza con una trasformazione lineare che utilizza una matrice detta di *minmax*. In essa sono contenuti i valori limite di scala di ciascun nodo di *input* e di *output*. Nel problema in esame questi valori non sono univocamente determinabili. Gli ordini di capi possono ragionevolmente essere compresi tra 0 ed 300.000 in una singola stagione, ma ciò non è necessariamente obbligatorio. Nell'impostazione di *bp-ct*, l'utente che predispone i file con la matrice dei dati deve anche preoccuparsi di verificare i valori di scala. Nel caso del modello *jBogliettiSwarm* i dati sono generati internamente e quindi è stato costruito un metodo *checkNewMinMax* che verifica i valori ed in caso essi oltrepassino i limiti, la matrice è aggiornata e viene notificato all'utente che è necessario rieseguire l'apprendimento perché i valori dei pesi non sono significativi. Essi sono messi appunto su dati non scalati correttamente.
- L'ultime modifica consiste nella dotazione di un metodo *saveTrainingMatrix* che è in grado di salvare su file la matrice di apprendimento costruita dal metodo *makeTargetsForLearning*. Nell'impostazione tradizionale del modello non è necessario poiché i dati giunono necessariamente dall'esterno. In questo caso è utile per verificare il

funzionamento corretto del modello ed evntualemente di riutilizzare la matrice nello schema classico di *bp-ct*.

8.7 Lo schema di funzionamento dei modelli sovrapposti

È utile analizzare il meccanismo che consente il funzionamento ed coordinamento dei due modelli. La classe *BogliettiObserverSwarm* contiene al suo interno due variabili che contengono l'indirizzo degli oggetti *model* che racchiudono lo schema di funzionamento di entrambi i modelli.

La costruzione dell'oggeto consiste di tre fasi: la fabbricazione degli oggetti, la costruzione dello schema sequenziale degli eventi ed infine di un comando di attivazione del tutto.

In fase di costruzione, quando è eseguito il metodo *buildObjects* che fabbrica tutti gli oggetti necessari al modello, il modello base è costruito in modo tradizionale, il modello di simulazione, invece, ponendosi ad un livello di astrazione più elevato, riceve l'indirizzo del modello base in fase di costruzione. Ciò consente ad esso di "osservare" dall'interno le componenti del modello e di modificarne i dati.

La costruzione dello schema sequenziale degli eventi è, come poc'anzi detto, molto semplice, poiché la gestione della sincronia è affidata interamente a *Swarm*.

La fase di attivazione consiste in una banale chiamata al metodo *activateIn* che conclude la fase di creazione. Specificando un valore all'opportuno parametro *useDoubleModel* si agisce proprio su questo metodo. Se l'utente specifica che non il modello sovrapposto non deve essere eseguito, la classe *Observer* non dà il comando di attivazione a quest'ultimo e tutti gli eventi relativi al *ForecastModel* non sono attivati. In questa configurazione le previsioni sono effettuate, come si è visto, dall'oggetto *SellerRuleMaster*.

Il modello di simulazione, possedendo l'indirizzo del modello base, è in grado di chiamare i suoi metodi, di chiedere l'indirizzo degli oggetti in esso contenuto, risalendo al magazzino dei dati degli agenti rappresentanti. Con questa soluzione i dati sono modificati in modo trasparente.

Capitolo 9 I risultati

9.1 Le modalità di esecuzione del modello

Il software *jBoglietti Swarm* può essere utilizzato in diverse configurazioni. Si è già osservato che ha la capacità di riprodurre nei dettagli la campagna di vendita di una stagione. Durante questa operazione svolge un'importante funzione di elaborazione delle informazioni.

Il database dell'azienda è un database di tipo non relazionale, privo della capacità di elaborare le *query*, cioè di estrarre automaticamente i dati eseguendo operazioni su di essi. Per ottenere i dati aggregati da utilizzare nelle previsioni è necessario l'intervento di un operatore che li estrae attraverso la scrittura di codice informatico. Ciò richiede del tempo e può essere eseguito soltanto periodicamente. In questo contesto, dunque, la capacità del modello di fornire automaticamente i dati elaborati con frequenza settimanale lo rendono un importante strumento di supporto alle decisioni.

Attraverso l'uso di un database relazionale è stato molto semplice suddividere per famiglie i prodotti, mentre nel database aziendale questa informazione non è disponibile. La percentuale di distribuzione delle famiglie di prodotto all'interno di una linea è un dato molto significativo ai fini delle previsioni. Il software crea automaticamente questo raggruppamento fornendo al *management* un'informazione ulteriore che costituisce un elemento di riflessione per comprendere meglio la natura delle previsioni.

La caratteristica principale del software è la capacità di usare tali informazioni per formulare le previsioni di vendita per linea di prodotto. La capacità di realizzare previsioni dipende strettamente dalle vendite della stagione precedente. Per questa ragione al termine di un esercizio, quando tutta la serie storica dei dati è disponibile, è necessario eseguire il modello impostando l'opportuno parametro *onlyVerificationRun* al valore *false*. Con tale operazione si comunica al programma che al termine dell'esecuzione del modello deve innescare gli algoritmi che si occupano di osservare i dati e modificare le regole di previsione. Dopo avere svolto questa operazione, il modello è in grado di elaborare previsioni sui dati, anche parziali, della stagione successiva.

È importante notare che il modello base adotta una metodologia di previsione a "memoria corta", ovvero utilizza soltanto le informazioni della stagione precedente. Il modello fondato sulle reti neurali può eseguire l'apprendimento sui dati usando un percorso più sofisticato. Infatti, quando i pesi della rete sono formati su una stagione di vendita, è possibile eseguire l'apprendimento sui nuovi dati partendo da una struttura della funzione a rete neurale già formata su quelli della stagione precedente. Con questa metodologia si garantisce, a patto di non eseguire un apprendimento troppo prolungato, una certa inerzia della rete ad "imparare" i dati più recenti mantenendo, quanto possibile, un'influenza della sua precedente struttura interna.

Sarebbe stato possibile eseguire l'intero apprendimento utilizzando tutte le informazioni a disposizione, ma la scelta è ricaduta sull'uso dei dati relativi all'ultima stagione, perché sull'arco di tre esercizi la struttura del mercato varia notevolmente, per prodotti e per tendenze della moda e quindi i dati di due esercizi precedenti avrebbero avuto lo stesso peso di quelli della stagione più recente nell'apprendimento.

9.2 Il pannello di controllo

Il programma consente di impostare tutti i parametri della simulazione in modo interattivo. Esistono tre pannelli che controllano i parametri dell'osservatore, del modello base e del modello di simulazione.

- Il pannello chiamato *BogliettiObserverSwarm* è presentato in fig. 21.

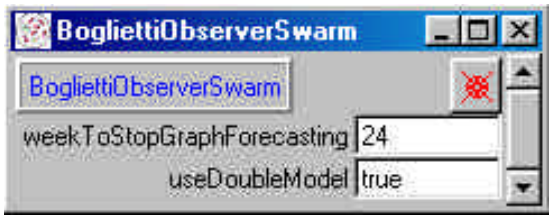

fig. 21 Il pannello di controllo di BoglettiObserverSwarm

Impostando il parametro *weekToStopGraphForecasting* il programma interrompe il disegno della previsione di ogni singolo agente (area rossa del grafico a barre) alla settimana desiderata, in modo da fornire un confronto visivo della struttura delle previsioni in termini di volumi di vendita per singolo agente.

Il parametro *useDoubleModel* decide se il modello di simulazione fondato sulle reti neurali deve essere attivato o meno. Nel caso il valore sia *true* il modello è eseguito e sovrappone le previsioni a quelle del modello base.

- Il pannello *BoglettiModelSwarm* consente di modificare i parametri del modello base ed è presentato in fig. 22.

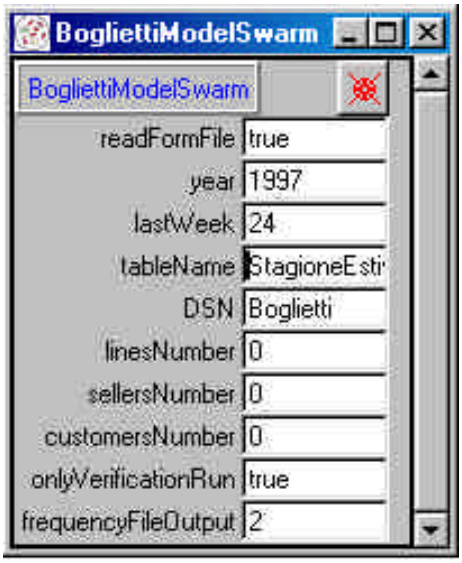

fig. 22 Il pannello di controllo di BogliettiModelSwarm

Il parametro *readFromFile* è stato presentato nel capitolo precedente. Se impostato a *true* legge i dati da i file con estensione ".data" (*StagioneEstiva1997.data*, *StagioneEstiva_Clienti.data*, ecc.), usando la classe *FileInterface*. Se impostato a *false* usa un database che corrisponde al *DataSourceName* (*DSN*) specificato nel parametro *DSN*. Tale parametro cerca la voce corrispondente nell'elenco delle sorgenti di dati ODBC[12]. Attraverso questo meccanismo il database può avere un qualsiasi nome ed una qualsiasi collocazione all'interno del computer, ma attraverso il *DSN* il programma è in grado di identificarlo univocamente.

Il parametro *year* decide l'anno della stagione che deve essere simulata, il parametro *tableName* indica la sigla della tabella che contiene i dati: essa può essere *StagioneEstiva* o *StagioneInverno*.

Il parametro *lastWeek* è un elemento fondamentale, poiché stabilisce in quale settimana deve considerarsi terminata la stagione. L'utilizzatore può considerare ininfluente la parte delle vendite che si realizza dopo una certa data ai fini delle previsioni. Con questo parametro è possibile stabilire una finestra entro la quale il modello osserva i dati. Le simulazioni qui condotte sono state riferite al valore *24* che corrisponde per la stagione estiva al 31 dicembre.

Il parametro *onlyVerificationRun* è stato già presentato: esso determina se alla settimana *lastWeek* il modello deve memorizzare i dati della stagione appena simulata, al fine di utilizzarli in quella successiva. I parametri *linesNumber*, *sellersNumber*, *customersNumber* sono parametri di sola lettura che indicano quanti agenti e quante linee di prodotto sono analizzate dal modello. I parametri sono determinati automaticamente dal programma quando si connette alla base dati.

Infine, il parametro *frequencyFileOutput* indica ogni quante settimane si desidera che l'oggetto *forecaster* registri sul file denominato *forecaster.output* i risultati delle vendite, le previsioni correntemente effettuate ed il numero di clienti contattati. Questo file rappresenta il risultato dettagliato

[12] Questo valore è impostabile nelle piattaforme *Windows* dal pannello di controllo alla voce "*ODBC Data Sources*".

dell'esecuzione del modello ed è salvato in formato testuale, quindi facilmente importabile da un'applicazione che presenti i dati in una forma più accattivante.

- Il pannello *ForecastModelSwarm* è presentato in fig. 23.

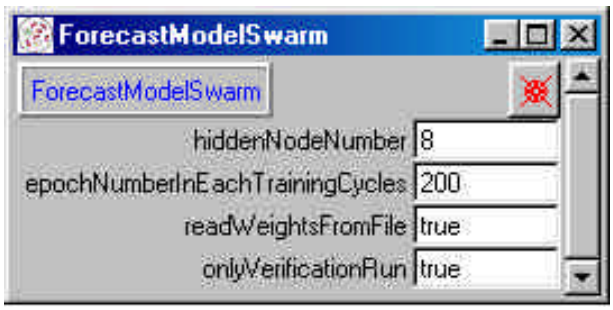

fig. 23 Il pannello di controllo di BoglettiModelSwarm

Esso rappresenta il punto più delicato del software. A parte il noto significato del parametro *onlyVerificationRun* i tre parametri determinano il comportamento delle reti neurali.

Il parametro *hiddenNodeNumber* consente di decidere di quanti nodi nascosti ciascuna rete neurale deve essere dotata. Si ricorda che aumentandone il numero si ottiene come risultato una funzione RNA che apprende molto bene i dati ma ha poca capacità di generalizzazione, mentre un numero troppo ristretto provoca risultati imprecisi. Le prove qui condotte hanno identificato un numero che oscilla tra 5 e 6 nodi come la soluzione ideale per le previsioni dei dati.

Qualora questo parametro fosse cambiato sarebbe necessario rieseguire l'apprendimento specificando il valore *readWeightsFromFile* a *false*, poiché i file che contengono gli attuali parametri (pesi) della rete non sarebbero più significativi. Quest'ultimo parametro indica se i pesi della rete neurale devono essere generati casualmente o letti dagli appositi file (*agent#.wih*, *agent#.who*). Nel caso si sia eseguito l'apprendimento è necessario indicare il valore *true* per osservare il risultato in termini di previsioni.

Il parametro *epochNumberInEachTrainingCycles* indica quante volte il modello deve eseguire l'algoritmo di apprendimento delle reti neurali sull'intera matrice di *training*. Anche in questo caso vale il discorso sulla scelta del parametro in funzione del possibile *overfitting* o *underfitting* di cui si è discusso nel capitolo 2. Il parametro rivelatosi più interessante è pari a 80. È utile notare che con un valore di 80, considerando che nella stagione estiva esistono 14 linee di prodotti e 24 settimane di dati, l'algoritmo è applicato per ciascun agente per 80 * 14 * 24 volte, ovvero per 26.880. In aggregato per tutti gli agenti (in totale ci sono 34 agenti) è applicato ben 913.920 volte!

Ciò implica che il tempo necessario all'apprendimento dell'intero insieme delle reti neurali renderebbe difficilmente impiegabili le tecniche senza l'ausilio di un computer, che impiega circa 5 minuti di calcolo.

9.3 La procedura di previsione e i primi risultati

L'analisi dei dati è stata condotta sulla stagione estiva degli anni 1997, 1998 e 1999. Le previsioni ottenute sono relative agli anni '98 e '99. La stagione '97 è usata come base di confronto per le previsioni della stagione '98.

La prima operazione consiste nell'esecuzione del programma impostando il valore *false* del parametro *onlyVerificationRun* di entrambi i modelli. La simulazione è eseguita sulla stagione 1997, all'ultima settimana avviene l'apprendimento dei dati.

Successivamente il modello è eseguito due volte sulla stagione 1998. La prima esecuzione è effettuata con il parametro *useDoubleModel* impostato al valore *false*. Il risultati così ottenuti sono relativi alle previsioni effettuate dal solo modello base. Il primo indicatore del risultato è apprezzabile graficamente in fig. 24 dove sono confrontate le vendite rappresentate dalla linea blu e le previsioni che il modello congettura fino al termine della stagione, rappresentate dalla linea arancione. Il risultato dà soltanto un'impressione grafica, ma i reali risultati della previsione possono essere valutati soltanto analizzando nel dettaglio i risultati contenuti nel file *forecaster.output*. Infatti la previsione per ogni linea di prodotto può essere apprezzata solo in quest'ultimo.

In fig. 25 è presentato il grafico del modello eseguito con il parametro *useDoubleModel* impostato a *true*. I risultati sono quindi relativi all'uso delle reti neurali.

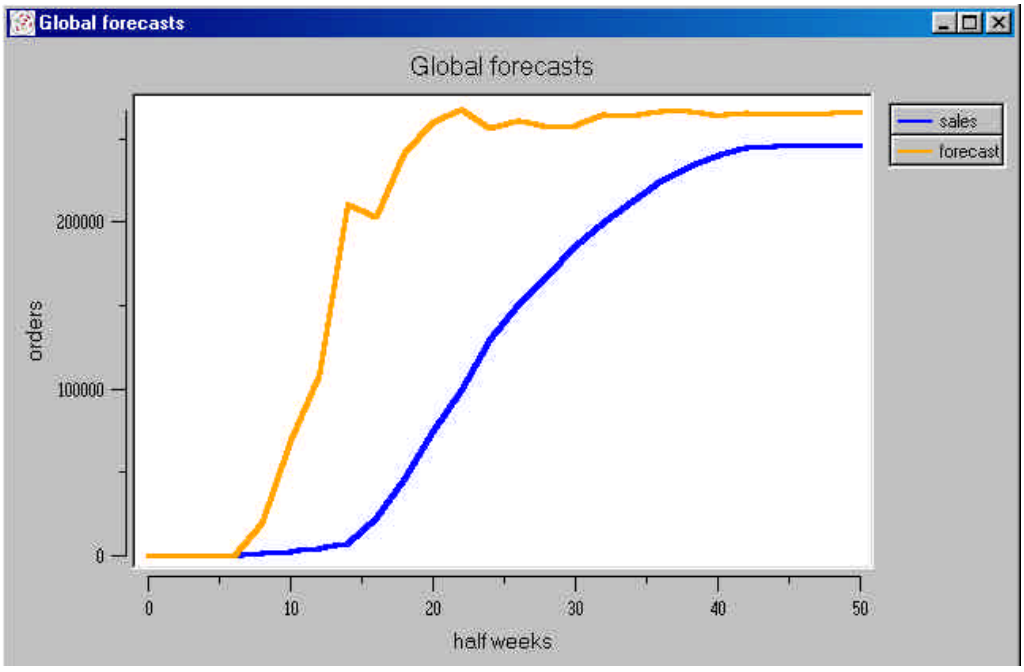

fig. 24 Previsioni aggregate del modello base sulla stagione 1998

Il grafico esprime la somma delle previsioni di tutte le linee di prodotto e fornisce dunque soltanto un'impressione visiva della settimana in cui l'ordine di grandezza della previsione raggiunge un

livello compatibile con le vendite a fine stagione. Si può osservare che la linea delle previsioni raggiunge tale livello intorno alla nona, decima settimana (equivalente al valore 18-20 nell'asse delle ascisse)[13], ovvero alla metà del mese di settembre. Per convenzione, infatti, la stagione estiva comincia il 15 luglio, una data in cui difficilmente si raccolgono ordini, al fine di poter comparare le stagioni tra loro in termini di numero di settimane.

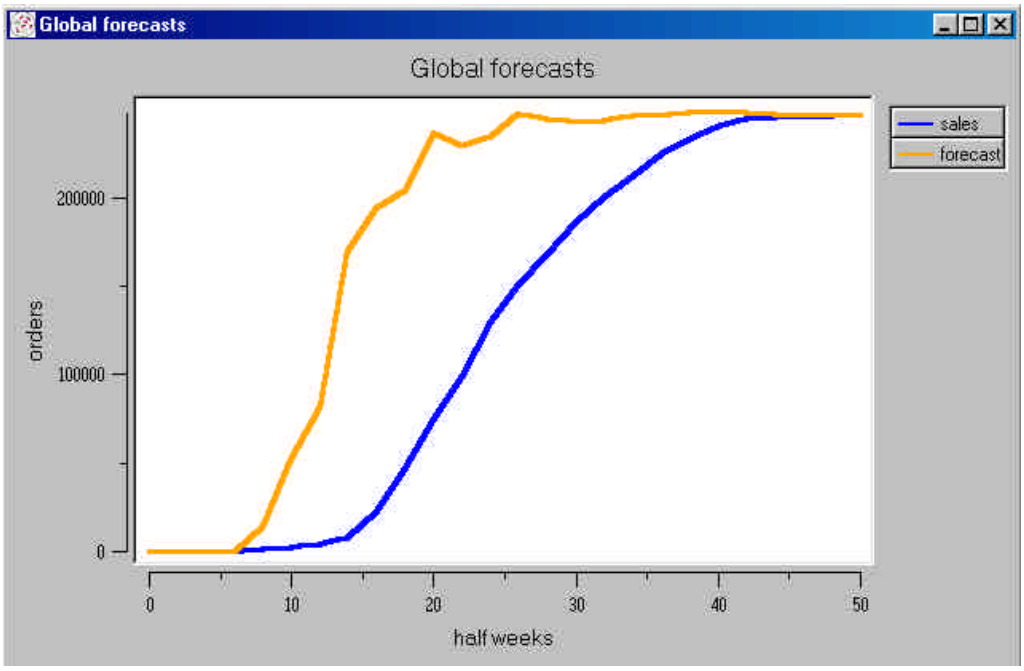

fig. 25 Previsioni aggregate del modello di simulazione sulla stagione 1998

Per svolgere una corretta valutazione dei risultati è prima necessario descrivere il metodo di previsione da usare come termine di confronto.

[13] Il grafico mostra i risultati due volte alla settimana, pertanto i valori delle ascisse (*half weeks*) corrispondono al doppio del valore relativo alla settimana.

Tabella 3 Confronto tra le tecniche di previsione (dati estate 1998)

U=Uomo D=Donna B=Bimbo DP=Dolci Pensieri

Previsioni al 30/9/1998

1998	Clienti	U classico	U moda	D classico	D moda	B classico	B moda	U slip	D slip	B slip	U slip moda	D varie	D notte	DP	DP notte
Vendite 30/12	1434	39703	9012	24963	78523	4422	3285	7377	22977	2775	6069	14473	15968	10494	5781
Vendite 30/9	681	19734	5052	11828	40936	2064	1746	3894	12953	1440	3660	8565	8974	5797	3342
Capi/cli 30/12	2,11	27,69	6,28	17,41	54,76	3,08	2,29	5,14	16,02	1,94	4,23	10,09	11,14	7,32	4,03
Capi/cli 30/9		28,98	7,42	17,37	60,11	3,03	2,56	5,72	19,02	2,11	5,37	12,58	13,18	8,51	4,91
K		**0,96**	**0,85**	**1,00**	**0,91**	**1,02**	**0,89**	**0,90**	**0,84**	**0,92**	**0,79**	**0,80**	**0,85**	**0,86**	**0,82**
Previsione Boglietti	1309,615	40359	9482	22902	72764	5314	3880	8478	23197	4075	6400	14110	16408	9359	6427
		-1,7%	**-5,2%**	**8,3%**	**7,3%**	**-20,2%**	**-18,1%**	**-14,9%**	**-1,0%**	**-46,8%**	**-5,5%**	**2,5%**	**-2,8%**	**10,8%**	**-11,2%**
Modello base		37778	9706	23451	79659	4853	3645	7355	25164	3276	7121	17353	17601	12561	6530
		4,8%	**-7,7%**	**6,1%**	**-1,4%**	**-9,7%**	**-11,0%**	**0,3%**	**-9,5%**	**-18,1%**	**-17,3%**	**-19,9%**	**-10,2%**	**-19,7%**	**-13,0%**
Reti neurali		45706	10497	25062	77066	4516	3835	6829	24499	2884	6849	17130	15294	10032	5107
		-15,1%	**-16,5%**	**-0,4%**	**1,9%**	**-2,1%**	**-16,7%**	**7,4%**	**-6,6%**	**-3,9%**	**-12,9%**	**-18,4%**	**4,2%**	**4,4%**	**11,7%**

Previsioni al 15/10/1998

Vendite 30/12	1434	39703	9012	24963	78523	4422	3285	7377	22977	2775	6069	14473	15968	10494	5781
Vendite 15/10	918	25575	6321	15640	54099	2769	2169	4869	16478	1812	4335	10813	11494	7274	4289
Capi/cli 30/12	1,56	27,69	6,28	17,41	54,76	3,08	2,29	5,14	16,02	1,94	4,23	10,09	11,14	7,32	4,03
Capi/cli 15/10		27,86	6,89	17,04	58,93	3,02	2,36	5,30	17,95	1,97	4,72	11,78	12,52	7,92	4,67
K		**0,99**	**0,91**	**1,02**	**0,93**	**1,02**	**0,97**	**0,97**	**0,89**	**0,98**	**0,90**	**0,86**	**0,89**	**0,92**	**0,86**
Previsione Boglietti		39145	9178	22534	72285	5268	3482	7953	22662	3597	5908	14122	16053	9755	6307
		1,4%	**-1,8%**	**9,7%**	**7,9%**	**-19,1%**	**-6,0%**	**-7,8%**	**1,4%**	**-29,6%**	**2,6%**	**2,4%**	**-0,5%**	**7,0%**	**-9,1%**
Modello base		37708	9658	23936	82103	4708	3366	7395	25080	3018	6600	16886	17890	12010	6470
		5,0%	**-7,2%**	**4,1%**	**-4,6%**	**-6,5%**	**-2,5%**	**-0,2%**	**-9,2%**	**-8,8%**	**-8,7%**	**-16,7%**	**-12,0%**	**-14,4%**	**-11,9%**
Reti neurali		41915	9362	24362	79305	4702	3322	6758	24214	2685	5945	16252	15520	9921	5405
		-5,6%	**-3,9%**	**2,4%**	**-1,0%**	**-6,3%**	**-1,1%**	**8,4%**	**-5,4%**	**3,2%**	**2,0%**	**-12,3%**	**2,8%**	**5,5%**	**6,5%**

Tabella 4 Confronto tra le tecniche di previsione (dati estate 1999)

U=Uomo D=Donna B=Bimbo DP=Dolci Pensieri

Previsioni al 30/9/1999

1999	Clienti	U classico	U moda	D classico	D moda	B classico	B moda	U slip	D slip	B slip	U slip moda	D varie	D notte	DP	DP notte
Vendite 30/12	1262	27540	7242	16779	84165	3267	2724	12285	68379	1536	4965	17089	13118	4283	4813
Vendite 30/9	703	16809	4737	9288	49415	2100	1647	7833	43323	558	3369	11028	8136	2237	3346
Capi/cli 30/12		21,82	5,74	13,30	66,69	2,59	2,16	9,73	54,18	1,22	3,93	13,54	10,39	3,39	3,81
Capi/cli 30/9		23,91	6,74	13,21	70,29	2,99	2,34	11,14	61,63	0,79	4,79	15,69	11,57	3,18	4,76
K		**0,91**	**0,85**	**1,01**	**0,95**	**0,87**	**0,92**	**0,87**	**0,88**	**1,53**	**0,82**	**0,86**	**0,90**	**1,07**	**0,80**
Previsione Boglietti		33818	8450	19602	94787	4499	3099	14839	76850	1075	5586	18635	14477	4050	5788
		-22,8%	**-16,7%**	**-16,8%**	**-12,6%**	**-37,7%**	**-13,8%**	**-20,8%**	**-12,4%**	**30,0%**	**-12,5%**	**-9,0%**	**-10,4%**	**5,5%**	**-20,3%**
Modello base		32863	9551	18448	96184	4887	3575	15491	89683	1090	7034	23001	16613	4868	6687
		-19,3%	**-31,9%**	**-9,9%**	**-14,3%**	**-49,6%**	**-31,2%**	**-26,1%**	**-31,2%**	**29,0%**	**-41,7%**	**-34,6%**	**-26,6%**	**-13,7%**	**-38,9%**
Reti neurali		28430	8411	15983	76660	3355	2628	12364	69007	839	5390	20587	14485	3377	5831
		-3,2%	**-16,1%**	**4,7%**	**8,9%**	**-2,7%**	**3,5%**	**-0,6%**	**-0,9%**	**45,4%**	**-8,6%**	**-20,5%**	**-10,4%**	**21,2%**	**-21,2%**

Previsioni al 15/10/1999

Vend al 30/12	1262	27540	7242	16779	84165	3267	2724	12285	68379	1536	4965	17089	13118	4283	4813
Vend al 15/10	965	22134	5928	12806	66406	2463	2019	10023	55536	804	4215	13846	10854	3140	3987
Capi/cli 30/12		21,82	5,74	13,30	66,69	2,59	2,16	9,73	54,18	1,22	3,93	13,54	10,39	3,39	3,81
Capi/cli 15/10		22,94	6,14	13,27	68,81	2,55	2,09	10,39	57,55	0,83	4,37	14,35	11,25	3,25	4,13
K		**0,95**	**0,93**	**1,00**	**0,97**	**1,01**	**1,03**	**0,94**	**0,94**	**1,46**	**0,90**	**0,94**	**0,92**	**1,04**	**0,92**
Previsione Boglietti		34361	8452	20440	96386	3933	3058	15186	77440	1231	5901	18533	15079	4530	5374
		-24,8%	**-16,7%**	**-21,8%**	**-14,5%**	**-20,4%**	**-12,3%**	**-23,6%**	**-13,3%**	**19,8%**	**-18,9%**	**-8,4%**	**-14,9%**	**-5,8%**	**-11,7%**
Modello base		32852	9433	19923	101102	4258	3411	15294	87674	1260	6779	21926	17204	5273	6218
		-19,3%	**-30,3%**	**-18,7%**	**-20,1%**	**-30,3%**	**-25,2%**	**-24,5%**	**-28,2%**	**18,0%**	**-36,5%**	**-28,3%**	**-31,1%**	**-23,1%**	**-29,2%**
Reti neurali		28958	8122	17115	84221	3216	2665	12868	71196	1003	5367	19618	15024	3909	5372
		-5,1%	**-12,2%**	**-2,0%**	**-0,1%**	**1,6%**	**2,2%**	**-4,7%**	**-4,1%**	**34,7%**	**-8,1%**	**-14,8%**	**-14,5%**	**8,7%**	**-11,6%**

9.4 Il modello di previsione attuale

È già stato accennato che il metodo attuale di previsione delle vendite è parzialmente basato su elementi soggettivi. Il parametro fondamentale è espresso dal rapporto tra numero di capi venduti e clienti contattati. Sia ${}_aS_t$ il numero di capi venduti durante la settimana *t*, nella stagione *a* e ${}_aC_t$ il numero di clienti contattati nello stesso periodo, tale coefficiente si esprime con

$$ {}_aK_t = \frac{\sum_{i=0}^{t} {}_aS_i}{\sum_{i=0}^{t} {}_aC_i} . \qquad (26) $$

La previsione è calcolata usando il coefficiente calcolato con la (26) e con un parametro C_F stimato soggettivamente dal previsore, che esprime il *target* di clienti contattati a fine stagione. Si descrive con la seguente

$$ {}_aP_t^T = C_F \cdot {}_aK_t \cdot \frac{{}_{a-1}K_T}{{}_{a-1}K_t} , \qquad (27) $$

nella quale il numero di clienti previsti a fine stagione è deciso dal previsore sulla base degli obiettivi stabiliti dal *management* aziendale e dalla rete di vendita. Tale numero non è determinabile con l'applicazione di una formula, ma è ipotizzato tenendo presente il risultato della stagione precedente e la percezione soggettiva circa le condizioni della domanda nella stagione in corso.

Al fine di fornire una formula di calcolo più oggettiva per confrontare i risultati delle previsioni ottenute con i metodi proposti nel modello, il coefficiente C_F è calcolato con la seguente

$$ C_F = \sum_{i=0}^{t} {}_aC_i \cdot \frac{\sum_{i=0}^{T} {}_{a-1}C_i}{\sum_{i=0}^{t} {}_{a-1}C_i} . \qquad (28) $$

L'analisi ed il confronto dei risultati fornisce un'impressione generalizzata per cui la previsione delle linee classiche sia abbastanza soddisfacente, mentre quelle relative alle linee moda presenti risultati non molto affidabili, a causa della maggiore variabilità della domanda ed al più ridotto numero di capi venduti, con costi per unità di prodotto molto elevati. In quest'ultimo caso l'errore di previsione ha un peso più importante sui mancati utili aziendali.

9.5 Le previsioni della stagione 1998

Le previsioni ottenute dalla prima esecuzione del modello sono abbastanza interessanti, come si può osservare in Tabella 3. Deve essere tenuto presente che nella stagione 1997 i dati relativi alla linea "Dolci Pensieri" contengono anche quelli relativi alla linea "Dolci Pensieri

Notte". La suddivisione è stata inserita nel 1998. Questo elemento comporta un confronto non ideale tra i dati relativi alle due stagioni.

I risultati ottenuti con l'applicazione del modello base sono riportati per completezza. Nella sostanza non sono interessanti, soprattutto perché molto variabili rispetto al *target*. La variabilità è sinonimo di inaffidabilità in questo contesto.

Non è tanto importante il valore assoluto della previsione, quanto la sua stabilità in termini di errore. Se la previsione garantisce di non scostarsi mai troppo dall'obiettivo, allora può essere ritenuta affidabile.

Nel caso del modello base, alcuni valori sono sorprendentemente buoni. Si veda, ad esempio, la linea "Donna Moda", in cui il metodo di riferimento si è discostato del 7,3% dalle vendite effettive contro il –1,4% del modello base. La presenza di risultati come la linea "Uomo Slip Moda" (-5,5% del metodo Boglietti contro -17,3% del modello base) rendono, però, rischioso affidarsi a questo metodo.

La differenza tra i due metodi è dovuta al fatto che il modello base non calcola, diversamente dal metodo di riferimento, un coefficiente di correzione del rapporto tra capi venduti e clienti contattati, il coefficiente *K* poc'anzi descritto. Esso è un elemento importante poiché l'ordine per cliente tende sistematicamente a diminuire al trascorrere del tempo.

Come è già stato ampiamente notato, la funzione del modello base è soltanto quella di fornire dati al modello che si occupa di riprodurre il funzionamento dell'azienda. Il confronto più interessante deve essere effettuato tra il metodo di riferimento e quello delle reti neurali.

I risultati complessivi sono difficili da interpretare, poiché circa nel 50% dei casi la funzione a rete neurale ha fornito risultati migliori in valore assoluto e nell'altro 50% risultati peggiori.

Alla luce di questo elemento non è giustificato l'uso del secondo metodo rispetto a quello attualmente usato dalla Boglietti, dato che quest'ultimo è più semplice da realizzare.

Ma se si considera il fatto che i parametri della rete neurale sono stati scelti in modo da ottimizzare la previsione delle linee moda, per i motivi già espressi in precedenza, allora i risultati possono essere interpretati in maniera diversa. Infatti, il confronto tra le linee moda è a favore delle reti neurali. Ciò è assai plausibile, poiché il basso numero di nodi nascosti fornisce alla funzione una buona capacità di generalizzazione, che nel caso di dati molto disturbati ed imprevedibili corrisponde ad una migliore capacità di approssimare i risultati.

Questo risultato è particolarmente evidente nelle previsioni realizzate nelle due settimane successive alla prima elaborazione. I dati relativi al 15 ottobre sono molto interessanti, in quanto la rete fornisce una previsione peggiore soltanto nel caso della linea "Uomo Moda" e la differenza di errore è di circa 2 punti percentuali, ovvero circa 200 capi. Tutte le altre linee moda sono previste più accuratamente rispetto al metodo di riferimento.

9.6 Le previsioni della stagione 1999

L'esecuzione del programma per l'apprendimento dei dati relativi alla stagione 1998 è stata fatta con il parametro *readWeightsFromFile* impostato a *true*. Le reti neurali, cioè, sono state addestrate sui dati della stagione '98 partendo dalla struttura usata nella stagione precedente.

Come si è detto nel capitolo 2, le funzioni a rete neurale sono tecniche di previsione *data intesive*. Forniscono buoni risultati quando la quantità di dati su cui poggiare le previsioni è notevole. Questa caratteristica emerge nel caso in esame, poiché i risultati ottenuti dopo il secondo processo di apprendimento sono decisamente più interessanti della stagione precedente.

Ciò è dovuto anche al fatto che nel confronto tra la stagione 1998 e 1999 i dati relativi alle linee moda sono più significativi. Infatti, l'azienda ha puntato sulla vendita di questi prodotti in modo più marcato proprio a partire dalla stagione '98.

Come si nota in Tabella 4 l'intera gamma di prodotti, comprese le linee classiche, è prevista dal modello con notevole precisione già alla fine del mese di settembre. Soltanto nel caso delle linee "Bimbo Slip", "Donna Varie", "Dolci Pensieri" i risultati del metodo Boglietti sono nettamente migliori, in tutte le altre il metodo basato sulle reti neurali ha fornito risultati talvolta nettamente migliori, talvolta di poco superiori. Se si considera che la sola linea "Dolci Pensieri" è una linea moda, il risultato è assolutamente rilevante.

Le previsioni realizzate al 15 ottobre confermano la maggiore capacità delle reti neurali a correggere i propri errori quando le informazioni disponibili aumentano. In questa data, infatti, seppure le tre linee appena citate siano previste meno efficacemente del metodo di riferimento, lo scostamento si riduce e tutti gli altri valori migliorano.

9.7 Le conclusioni

Ciò che emerge dall'analisi dei dati è un risultato piuttosto interessante. Il metodo delle previsioni fondato sulle funzioni a rete neurale si è dimostrato una valida alternativa ai metodi attualmente adottati dall'azienda.

Il risultato più interessante è certamente l'affidabilità delle previsioni. Nelle linee moda, l'errore di previsione delle reti neurali non supera mai in valore assoluto il 12% (linea "Uomo moda") al 15 ottobre nell'arco delle due stagioni, mentre il metodo di confronto raggiunge il valore massimo del 19% (linea "Uomo Slip Moda").

L'affidabilità è importante perché la difficoltà nell'utilizzo delle tecniche di previsione è soprattutto concentrata sul timore che i dati forniti si rivelino totalmente infondati, producendo effetti economici negativi. Come si può osservare in Tabella 5 il metodo delle reti neurali possiede una variabilità media più contenuta e ciò lo rende più affidabile rispetto al metodo di riferimento.

Il risultato non deve però essere ridotto soltanto ad un mero confronto tra le *performance* delle diverse tecniche. Il vantaggio più importante dell'intero lavoro di costruzione del modello di simulazione dell'azienda consiste nella capacità di elaborare i dati che sono giornalmente resi disponibili nel database aziendale, nella capacità di fornire le previsioni settimanalmente, così da poter apprezzare la dinamica delle congetture elaborate, di valutare come la previsione evolve man mano che sono disponibili informazioni sempre più significative.

Tabella 5 Errore medio dei due metodi di previsione

Periodo	Scostamento medio previsioni Boglietti	Scostamento medio previsioni con reti neurali
30/09/1998	11,2%	8,7%
15/10/1998	7,6%	4,7%
30/09/1999	17,2%	12%
15/10/1999	16,2%	8,9%

La previsione dei dati non può essere accolta in modo acritico da chi si occupa di decidere gli acquisti delle materie prime e dei semilavorati. Essa deve essere utilizzata conoscendone le caratteristiche: le tecniche di previsione sono strumenti che forniscono una tendenza, un'ipotesi totalmente dipendente dalle informazioni codificate nel database. La decisione deve essere sempre assunta da una o più persone che posseggono informazioni implicite e che posseggono la capacità di percepire attraverso le "sensazioni" quali scelte possano rivelarsi migliori. Spesso, il successo aziendale è legato alla capacità di rischiare, al coraggio di assumere decisioni non sempre perfettamente razionali.

Per queste ragioni lo strumento di previsione deve essere comunque sempre utilizzato come un metodo utile a confermare le scelte assunte dal management. Può fornire suggerimenti ma non può determinare la politica degli acquisti, poiché non è in grado di considerare le informazioni implicite, che solo l'elemento umano conosce.

Il modello di simulazione è certamente uno strumento flessibile, poiché dalla semplice ricostruzione dei flussi di informazione codificata realizzatisi nel passato permette con poco sforzo di introdurre molti meccanismi di analisi.

Il processo di formulazione del modello ha consentito inoltre di raggiungere due scopi. Il primo è stato il tentativo di formalizzare la conoscenza posseduta implicitamente dalle entità che formano l'azienda. Coloro che si occupano di formulare le decisioni hanno dovuto descrivere formalmente il loro modo di operare, comprendendo gli eventuali limiti del metodo e migliorandolo. Infatti, spesso non si ha la consapevolezza delle conoscenze possedute.

Il secondo scopo è stato quello di fornire un'infrastruttura informativa basata su un modello facilmente adattabile, attraverso il quale possono essere descritti i flussi delle informazioni, possono essere realizzati esperimenti relativi ad ipotesi di mutamento strutturale nelle politiche di gestione commerciale dell'azienda.

Inoltre, è stata esplorata la possibilità di sfruttare questa architettura progettuale per ottenere uno strumento di elaborazione dei dati finalizzato al supporto alle decisioni. Tutto ciò in modo dinamico, durante la riproduzione del modello di funzionamento dell'azienda. Risultato che non si può ottenere neanche con l'uso di sofisticati strumenti di analisi *on-line* delle informazioni, di cui si è ampiamente trattato nei capitoli precedenti.

I modelli di simulazione aziendale sono una tecnica di indagine piuttosto recente, che apre interessanti sviluppi nella possibilità di valutare le imprese attraverso esperimenti riprodotti da un computer. Il limite principale è dato dalla scarsa letteratura in merito all'argomento, il che implica la necessità di dover esplorare la metodologia da diversi punti di vista con l'incertezza del risultato che si può ottenere.

Nel caso specifico, anche grazie all'obiettivo particolarmente circoscritto, il risultato complessivo è stato positivo.

Bibliografia

CAPITOLO 1

- ARNTZEN, B. C., G. G. BROWN, T. P. HARRISON, and L. TRAFTON. *Global Supply Chain Management at Digital Equipment Corporation*. Interfaces, Jan.-Feb., 1995
- ARORA A., BOKHARI F., MOREL B. 1997, 1998, *Returns to specialization, transaction costs, and the dynamics of industial evolution*, Working paper Carnegie Mellon University
- BILLINGTON C., LEE H.L., 1992, *Managing Supply Chain Inventory: Pitfalls and Opportunities*, Sloan Management Review, spring 1992
- BILLINGTON C., LEE H.L., 1993, *Material Management in Decentralized Supply Chains*, Operations Research
- BREITMAN, R. L., and J. M. LUCAS. 1987. PLANETS: *A Modeling System for Business Planning*, Interfaces, 17, Jan.-Feb
- BRHUN P., SCHLUETER-LANGDON C., SHAW M.J., 1999, *Online supply chain modeling and simulation*, Santa Fe Institute, Santa Fe Working Paper
- CHANDLER A.D. 1990, *Scale and Scope: The Dynamics of Industrial Capitalism*, Harvard University Press
- COHEN, M. A. and H. L. LEE. 1988. *Strategic Analysis of Integrated Production-Distribution Systems: Models and Methods*, Operations Research
- 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
- FARRELL J., MONROE H.K., SALONER G. 1998, *The Vertical Organization of Industry: Systems Competition versus Component Competition*, Journal of Economics and Management Strategy
- 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)
- GEOFFRION, A., R. POWERS. 1993. *20 Years of strategic Distribution System Design: An Evolutionary Perspective*, Interfaces
- 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 1999.
- LIN F.-R., TAN G.W., SHAW M.J., 1996, *Multi-agent enterprise modeling*. Working Paper 96-0314, University of Illinois at Urbana-Campaign

- MERRY U., 1999, *Practically Applying the New Science to Organizations*, KE Workgroup Paper, (http://pw2.netcom.com/~nmerry/pract1.htm)
- SADEH N.M., SMITH S.F. e SWAMINATHAN J.M., 1998, *Modeling SupplyChaing Dynamices: a Multi-Agent Approach*, Decision Sciences Vol.29
- STALK G.P., EVANS E., SHULMAN L. 1992. *Competing on Capabilities: The New Rules of Corporate Strategy*. Harvard Business Review, 57-69.
- THORSTENSON A., 1999, *Expediting, Pooling, and Postponing in Supply Chain Models - A Review of Some Related Results*, Slides (http://www.himolde.no/~thorsten)

CAPITOLO 2

- AZOFF M.E., 1995, *Neural network time series forecasting for financial markets*, Wiley Finance Editon
- BELLACICCO A., LAURO N.C. (1997), *Reti neurali e statistica*, FrancoAngeli
- BELTRATTI A., MARGARITA S., TERNA P., 1996, *Neural Networks for Economic and Financial Modelling*, International Thomson Computer Press
- BOSQ D., 1998, *Nonparametric statistics for stochastic processes*. Estimation and prediction. 2nd, New York
- G. E. P. BOX, G. M. JENKINS (1976): *Time Series Analysis, Forecasting and Control*, Holden-Day, San Francisco.
- CARRELLA G., 1995, *L'officina neurale*, FrancoAngeli
- DELGADO A., PRAT A., 1997, *Modeling time series using a Hybrid System: Neural Networks and Genetic Algorithm*, (in BELLACICCO, LAURO, 1997).
- FREUND J.R., WILSON W.J., 1998, *Regression Analysis Statistical Modeling of a Response Variable*, Academic Press
- GRIGOLETTO M., VENTURA L., 1998, *Statistica per le Scienze Economiche*, Giappichelli
- MARBACH, MAZZIOTTA, RIZZI, 1991, *Le previsioni – Fondamenti logici e basi statistiche*, Etaslibri
- TERNA P., 1994, *Reti neurali artificiali e modelli con agenti adattivi*, XXXV Riunione scientifica annuale della società italiana degli economisti
- TERNA P., 2000, - comunicazione personale -
- TIROZZI B., 1995, *Modelli Matematici di Reti Neurali*, CEDAM
- WHITE H., 1988, *Economic prediction using neural networks*, IEEE San Diego
- WEIGEND A.S., HUBERMAN B.A., RUMELHART D.E., 1992, *Predicting Sunspots and Exchange Rates with Connectionist Networks*, SFI Studies in the Science of Complexity, Proc.Vol. XII

CAPITOLO 3

- 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
- CILLERS, P., 1998, *Complexity and Postmodernism: understanding complex systems*, Routledge.
- CONTE R., 1997, *Il metodo simulativo*, Istituto di Psicologia, Cnr Roma, Paper
- 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.
- EOYANG G., 1999, *Coping with Chaos: Seven Simple Tools by Glenda Holladay Eoyang*. Reviewed by Wei-Bin Zhang Institute for Future Studies, Stockholm, Sweden
- FORRESTER J.W., 1961, *Industrial Dynamics*, MIT Press
- GLEICK J., 1987, *Chaos: Making A New Science*, Viking ed.
- 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
- 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
- NONAKA I., 1994, *Come un'organizzazione crea conoscenza*, Economia&Management n.3
- 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
- TAKADA, 1994, In PEPE G., 2000.

- 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
- TESTFATION L., 1996, *How Economists Can Get Alife, to appear in The Economy as an Evolving Complex Systems*, Santa Fe Institute in the Science of Complexity. Addison-Wesley
- TESFATSION, L., 1998, *Agent-based Computational Economics* (http://www.iastate.edu/tesfatsi/ace.htm).
- THOM R., 1975, *Structural Stability and morphogenesis*, Reading: Benjamin
- WALDROP, M. MITCHELL. 1992, *Complexity: the emergence of science at the edge of order and chaos* Simon and Schuster.
- VRIEND NJ, 1999, *Was Hayek an Ace*?, University of London [Paper]

CAPITOLO 4

- AKERLOF G., 1970, The Market for Lemons: Quality Uncertainty and the Market Mechanism, The Quarterly Journal of Economics
- AKERLOF G., 1984, *An Economic Theorist's book of Tales*, Cambridge University Press
- GOLDSPINK C., 2000, Modeling social systems as complex: Towards a social simulation meta-model, JASSS
- HIEB D., 1992, *A Mathematica Primer*, Univeristy of Colorado
- MAEDER, R.E., 1993, *Object-oriented Programming*., The Mathematica Journal
- RESNICK M., 1994, Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds, MIT Press
- SCHNIDER J., WEISS J., 1992, *Tour Of Mathematica*
- VARIAN H.R., 1987, *Microeconomia*, Cafoscarina

CAPITOLO 5

- 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 1994
- GILBERT N., TERNA P., 1999, *How to build and use agent-based models in social science*, Mind & Society, 1

CAPITOLO 6

- DOUMEINGTS D., et al., 1993, *Gim (grai integrated methodology) and its evolutions – a methodology to design and specify advanced manufacturing systems.* In Yoshikawa H., Goossenaerts J., *Information Infrastructure Systems for Manufacturing*, Elservier Science B.V.
- LIN F.-R., TAN G.W., SHAW M.J., 1996, *Multi-agent enterprise modeling*. Working Paper 96-0314, University of Illinois at Urbana-Campaign
- MERTINS K., SUSSENGUTH W., JOCHEM R., 1992, *An object oriented method for integrated enterprise modeling as a basis for enterprise coordination.* In Petrie C.J., *Enterprise Integration Modeling: Proceedings of the First International Conference*, The MIT Press
- NIIIP Consortium, 1998, *NIIIP Reference Architecture,* (htttp://www.niiip.org)
- VERNADAT F., 1993, *Cimosa: enterprise modelling and enterprise integration using a process-based approach.* In Yoshikawa H., Goossenaerts J., *Information Infrastructure Systems for Manufacturing*, Elservier Science B.V.
- SALTARIN A., 2000, *BizTalk: fare EDI con XML e Schema*, Computer Programmino, Maggio, 2000SCHEER A.W., 1993, *Architecture of integrated information systems (aris).* In Yoshikawa H., Goossenaerts J., *Information Infrastructure Systems for Manufacturing*, Elservier Science B.V.
- SCHEER A.W., 1994, *Business Process Engineering: Reference Models for Industrial Enterprises.* Springer-Verlag
- WILLIAMS T.J., 1993, *The purdue enterprise reference architecture.* In Yoshikawa H., Goossenaerts J., *Information Infrastructure Systems for Manufacturing*, Elservier Science B.V.

CAPITOLO 7

- CHAUDHURI S., DAYAL U., 1996, *An Overview of Data Warehousing and OLAP Technology*, Proc. of VLDB

CAPITOLO 8

- HORSTMANN C., CORNELL G, 1999, *Core Java 1.2 - Fundamentals*, The Sun Microsystems Press
- 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*. Kluver Academic.
- TERNA P., 1999, *How to use bp-ct*, (terna@econ.unito.it).

APPENDICE A – Il codice di *Starlogo*

Codice di Starlogo (Akerloff-1.1.2.slogo)

Observer

```
; ********** Global variables of the environment
globals [
    envNumSellers envNumBuyers          ; number of agents
    envListSellers envListBuyers        ; NOT CURRENTLY USED
    envSincerityMin envSincerityMax
    envBasisPrice
    envFaithCorrection
    envMissingCorrection
    envFaithMin envFaithMax

    ;for probing: this variables are used to keep probe values generated by
    ;             updateProbes at the end of every cycle
    probeAverageTransactionPrice
    probeNumberOfTransactions
    probeAverageBuyerPrice
    probeAverageSellerPrice
    probeAverageFaith
]

; ********** Definitions
breeds [b_buyer  b_seller]

patches-own [
          ; variables used to comunicate information among agents
          priceBySeller
          qualityComunicatedBySeller
          qualityRealBySeller
          wantToExchangeByBuyer
        ]

turtles-own [
          ;general to every agent
          otherPrice
          price
          missing
          transactionDone
          transactionPrice

          ;buyer variables
          buyerFaith
          buyerQualityComunicated
          buyerQualityReal

          ;seller variables
          sellerSincerity
          sellerCurrentQuality
        ]

; ********** Setup MODEL
to setupModel

  clear-turtles
  clear-output
  setupPlot

  ; SET THE VARIABLES OF THE MODEL HERE
  set envNumSellers envNumAgentsSlider ;slider
                    ; to fix the value and disable the slider put the istruction below
                    ; set envNumSellers NUMBER_OF_AGENTS
  set envNumBuyers envNumSellers          ; the agents are the same number
                                          ; but this is not mandatory
  set envListSellers []         ; not currently used
  set envListBuyers []          ; not currently used

  set envSincerityMin 1
  set envSincerityMax 1.9
  set envBasisPrice envBasisPriceSlider                 ;slider
  set envMissingCorrection envMissingCorrectionSlider   ;slider
  set envFaithMin 0.3
  set envFaithMax 1
  set envFaithCorrection 0.02

  ; PRINT CURRENT MODEL PARAMETERS
  print "Setting up model: Akerloff v.0.97"
  print "   written by Michele Sonnessa 2000"
```

```
print ""
print "The model will run with these parameters:"
let [:st "The seller's sincerity random by {"]
set :st se :st envSincerityMin
set :st se :st ","
set :st se :st envSincerityMax
print se :st "}"
set :st "The buyer's faith random by {"
set :st se :st envFaithMin
set :st se :st ","
set :st se :st envFaithMax
print se :st "}"
print se [The buyer's faith correction is: ] envFaithCorrection
print se [The basis price is: ] envBasisPrice
print se [The missing correction is: ] envMissingCorrection
print ""
print "Good work ..."

; create the sellers, with red color
let [:temp 0]
repeat envNumSellers [
  crt 1
  ; init variables
  ask-turtle first list-of-turtles [set price 0]
  ask-turtle first list-of-turtles [set missing 0]
  ask-turtle first list-of-turtles [set transactionDone false]

  ask-turtle first list-of-turtles [set sellerCurrentQuality 1]
  let [:fth SincerityRandomizer]
  ask-turtle first list-of-turtles [set sellerSincerity :fth]

  ;internal parameters
  ask-turtle first list-of-turtles [setc red]
  ask-turtle first list-of-turtles [setx -10]
  ask-turtle first list-of-turtles [sety :temp]
  set :temp :temp + 1
  setbreed-of first list-of-turtles b_seller
  set envListSellers se envListSellers first list-of-turtles
]

; create the buyers, with blue color
let [:temp 0]
repeat envNumBuyers [
  crt 1
  ; init variables
  ask-turtle first list-of-turtles [set price 0]
  ask-turtle first list-of-turtles [set missing 0]
  ask-turtle first list-of-turtles [set otherPrice 0]
  ask-turtle first list-of-turtles [set transactionDone false]

  ask-turtle first list-of-turtles [set buyerQualityComunicated 1]
  ask-turtle first list-of-turtles [set buyerQualityReal 1]
  let [:fth FaithRandomizer]
  ask-turtle first list-of-turtles [set buyerFaith :fth]

  ;internal parameters
  ask-turtle first list-of-turtles [setc blue]
  ask-turtle first list-of-turtles [setx 10]
  ask-turtle first list-of-turtles [sety :temp]
  set :temp :temp + 1
  setbreed-of first list-of-turtles b_buyer
  set envListBuyers se envListBuyers first list-of-turtles
]

end

; ********** Setup PLOT
to setupPlot
  clearplot
  setplot-title "Average price of transactions"
  setplot-ymin envBasisPrice
  setplot-ymax envBasisPrice * 2
  setplot-xmax 300

;transaction
  pp 1
  ppreset
  ppd
  setppc yellow

;buyer
  pp 2
  ppreset
  ppd
```

```
  setppc blue

;seller
  pp 3
  ppreset
  ppd
  setppc red
end

; ********** runModel
to runModel
  ask-patches[set wantToExchangeByBuyer false]
  ask-turtles [
    agentInitCycle
    sellerThinkPrice
  ]
  ask-turtles [buyerGoToSeller]
  ask-turtles [sellerComunicatePriceQualityToPatch]
  ask-turtles [buyerGetPriceQualityFromPatch]
  ask-turtles [buyerThinkPrice]
  ask-turtles [sellerTryToExchange]
  ask-turtles [sellerHaveYouExchanged]
  ask-turtles [agentMakeCorrections]

  updateProbes
end

; ********** updateProbes
to updateProbes
  set probeAverageTransactionPrice sum-of-turtles [sellerGetTransactionPrice]
  let [:tmp sum-of-turtles [sellerTransactionDone]]
;print se (se probeAverageTransactionPrice " ") :tmp
  set probeAverageTransactionPrice probeAverageTransactionPrice / :tmp

  set probeNumberOfTransactions count-turtles-with [transactionDone]

  set probeAverageBuyerPrice sum-of-turtles [buyerGetPrice]
  set probeAverageSellerPrice sum-of-turtles [sellerGetPrice]
  set probeAverageBuyerPrice probeAverageBuyerPrice / envNumBuyers
  set probeAverageSellerPrice probeAverageSellerPrice / envNumSellers
  set probeAverageFaith sum-of-turtles [buyerGetFaith]
  set probeAverageFaith probeAverageFaith / envNumBuyers * 100
  ;set probeAverageTransactionPrice ((probeAverageBuyerPrice +
probeAverageSellerPrice) / 2)

  pp 1
  plot probeAverageTransactionPrice
  pp 2
  plot probeAverageBuyerPrice
  pp 3
  plot probeAverageSellerPrice
end

; ********** FaithRandomizer
to FaithRandomizer
  let [:tmp random (envFaithMax - envFaithMin) * 10]
  set :tmp :tmp / 10 + envFaithMin
  output :tmp
end

; ********** SincerityRandomizer
to SincerityRandomizer
  let [:tmp random (envSincerityMax - envSincerityMin) * 10]
  set :tmp :tmp / 10 + envSincerityMin
  output :tmp
end
```

Turtles

```
; ********** agentInitCycle
to agentInitCycle
  set price 0
  set transactionDone false
  set transactionPrice 0

  ifelse (breed = b_seller)
  [
    let [:rnd (random 11) / 10]
    ifelse :rnd <= 0.5
      [let [:rnd 1]]
      [let [:rnd 2]]
    set sellerCurrentQuality :rnd
  ]
```

```
  [
    set otherPrice 0
    ;set price 0
    set buyerQualityComunicated 0
    set buyerQualityReal 0
  ]
end

; ********** sellerThinkPrice
to sellerThinkPrice
  if (breed = b_seller) [
    set price (envBasisPrice * sellerCurrentQuality * sellerSincerity -
envMissingCorrection * missing)
    if price < envBasisPrice [set price envBasisPrice]
    ;print se [s ] price
  ]
end

; ********** buyerThinkPrice
to buyerThinkPrice
  if (breed = b_buyer) [
    set price (envBasisPrice * buyerQualityComunicated * buyerFaith +
envMissingCorrection * missing)
    if price < envBasisPrice [set price envBasisPrice]
    ;print se [b ] price
  ]
end

; ********** buyerGetPrice
to buyerGetPrice
  ifelse (breed = b_buyer) [
    output price
  ]
  [
    output 0
  ]
end

; ********** buyerGetFaith
to buyerGetFaith
  ifelse (breed = b_buyer) [
    output buyerFaith
  ]
  [
    output 0
  ]
end

; ********** sellerGetPrice
to sellerGetPrice
  ifelse (breed = b_seller) [
    output price
  ]
  [
    output 0
  ]
end

; ********** sellerGetTransactionPrice
to sellerGetTransactionPrice
  ifelse (breed = b_seller) [
    output transactionPrice
  ]
  [
    output 0
  ]
end

; ********** sellerTransactionDone
to sellerTransactionDone
  ifelse (breed = b_seller and transactionDone) [
    output 1
  ]
  [
    output 0
  ]
end

; ********** sellerComunicatePriceQualityToPatch
to sellerComunicatePriceQualityToPatch
  if (breed = b_seller)
  [
    let [:prc price]
    let [:qComByS (sellerCurrentQuality * sellerSincerity)]
```

```
    let [:qRByS sellerCurrentQuality]
    ask-patch-at xcor ycor [
      set priceBySeller :prc
      set qualityComunicatedBySeller :qComByS
      set qualityRealBySeller :qRByS
    ]
  ]
end

; ********** buyerGetPriceQualityFromPatch
to buyerGetPriceQualityFromPatch
  if (breed = b_buyer) [
    ask-patch-at xcor ycor [
      let [:price priceBySeller]
      let [:qComunicatedBySeller qualityComunicatedBySeller]
      let [:realQuality qualityRealBySeller]
    ]
    set otherPrice :price
    set buyerQualityComunicated :qComunicatedBySeller
    set buyerQualityReal :realQuality
  ]
end

; ********** buyerGoToSeller
to buyerGoToSeller
  if (breed = b_buyer) [
    setx -10
    sety random envNumSellers + 1
  ]
end

; ********** sellerTryToExchange
to sellerTryToExchange
  if (breed = b_seller) [
    if count-turtles-here < 2 [stop]

    grab one-of-turtles-here  ;choose one or all turtles to meet

    [
      ask-turtle partner [buyerSayIfYouWantToExchange]
    ]
  ]
end

; ********** buyerSayIfYouWantToExchange
to buyerSayIfYouWantToExchange
  ifelse (breed = b_seller)
  [
    Print "Error grabbing in proc 'buyerSayIfYouWantToExchange'."
    stopall
  ]
  [
    ifelse (buyerLikeMyPrice)
    [
      agentMakeTransaction
      ask-patch-at xcor ycor [set wantToExchangeByBuyer true]
    ]
    [
      agentDontMakeTransaction
      ask-patch-at xcor ycor [set wantToExchangeByBuyer false]
    ]
  ]
end

; ********** sellerHaveYouExchanged
to sellerHaveYouExchanged
  if (breed = b_seller)
  [
    ask-patch-at xcor ycor [let [:tmpExch wantToExchangeByBuyer]]

    ifelse :tmpExch
    [
      agentMakeTransaction
    ]
    [
      agentDontMakeTransaction
    ]
  ]
end

; ********** buyerLikeMyPrice
to buyerLikeMyPrice
  if (breed = b_seller) [print "Error grabbing in proc 'buyerLikeMyPrice '."]
  ifelse price >= otherPrice
```

```
  [                                                                    end
    output true
  ]
  [
    output false
  ]
end

; ********** agentMakeTransaction
to agentMakeTransaction
  set transactionDone true
  set transactionPrice price
  if (breed = b_buyer) [
    set transactionPrice price
    setc yellow
  ]
end

; ********** agentDontMakeTransaction
to agentDontMakeTransaction
  set transactionDone false
  set transactionPrice 0
  if (breed = b_buyer) [
    setc blue
  ]
end

; ********** agentMakeCorrections
to agentMakeCorrections
  ifelse transactionDone
  [
    set missing 0
  ]
  [
    set missing missing + 1
  ]
  if breed = b_buyer [
    if transactionDone [
      set buyerFaith buyerFaith - (buyerQualityComunicated - buyerQualityReal) *
envFaithCorrection
    ]
  ]
```

APPENDICE B – Il codice di *Mathematica*

Codice di Mathematica (Akerloff.1.0.1.nb)

```
Needs["Tools`Classes`"];
Needs["Graphics`MultipleListPlot`"];
(* Class Buyer
    Construct : minFaithRand.,  maxFaithRand., envBasisPrice,
                    envMissingCorrection,   envFaithCorrection
 *)
Class[Buyer, Object,
 {buyPrice, buySellerPrice, buyMissing, buyTransactionPrice, faith,
  qualityComunicated, qualityReal,
  buyEnvBasisPrice, buyEnvMissingCorrection, buyEnvFaithCorrection
  },
 {
  {new, (new[super];
       buyPrice = 0;
       buySellerPrice = 0;
       buyMissing = 0;
       buyTransactionPrice = 0;
       faith = Random[Real, {#1, #2}];
       qualityComunicated = 0;
       qualityReal = 0;
       buyEnvBasisPrice = #3;
       buyEnvMissingCorrection = #4;
       buyEnvFaithCorrection = #5;
       ) &},
  {buyInitCycle, (buyTransactionPrice = 0;
                  buyPrice = 0;
                  buySellerPrice = 0;
                  qualityComunicated = 0;
                  qualityReal = 0;
                   ) &},
  {buyThinkPrice, (buyPrice =
        buyEnvBasisPrice * qualityComunicated*faith +
          buyEnvMissingCorrection*buyMissing;
       If[buyPrice < buyEnvBasisPrice, buyPrice = buyEnvBasisPrice,];) &},
  {buyGetPrice, buyPrice &},
  {buyGetSellerPrice, buySellerPrice &},
  {buyGetTransactionPrice, buyTransactionPrice &},
  {buySetFromSellerPriceVirtualQRealQ, (buySellerPrice = #1;
       qualityComunicated = #2; qualityReal = #3;) &},
  {buyLikeMyPrice, (buyPrice >= buySellerPrice) &},
  {buyMakeTransaction, (buyTransactionPrice = buySellerPrice;) &},
  {buyDontMakeTransaction, (buyTransactionPrice = 0;) &},
  {buyMakeCorrections, (
       If[buyTransactionPrice == 0,
          buyMissing++,
          (buyMissing = 0;

             faith -= (qualityComunicated - qualityReal)*
                buyEnvFaithCorrection);];
       ) &}
  }
 ]
Buyer
(* Class Seller
    Construct : minSincerityRand.,
 maxSincerityRand., envBasisPrice, envMissingCorrection
 *)
Class[Seller, Object,
 {sellPrice, sellMissing, sellTransactionPrice, sincerity, currentQuality,
  sellEnvBasisPrice, sellEnvMissingCorrection},
 {
  {new, (new[super];
       sellPrice = 0;
       sellMissing = 0;
       sellTransactionPrice = 0;
       sincerity = Random[Real, {#1, #2}];
       currentQuality = 1;
       sellEnvBasisPrice = #3;
       sellEnvMissingCorrection = #4;
       ) &},
  {sellGetPrice, sellPrice &},
  {sellGetTransactionPrice, sellTransactionPrice &},
  {sellGetSincerity, sincerity &},
  {sellGetCurrentQuality, currentQuality &},
  {sellInitCycle, (sellTransactionPrice = 0;
                   sellPrice = 0;
                   currentQuality = Random[Integer, {1, 2}];
                    ) &},
```

```
    {sellThinkPrice, (sellPrice =
          sellEnvBasisPrice * currentQuality*sincerity -
            sellEnvMissingCorrection*sellMissing;

          If[sellPrice < sellEnvBasisPrice,
            sellPrice = sellEnvBasisPrice,]) &},
    {sellTellMeYourQuality, (currentQuality*sincerity) &},
    {sellMakeTransaction, (sellTransactionPrice = sellPrice;) &},
    {sellDontMakeTransaction, (sellTransactionPrice = 0;) &},
    {sellMakeCorrections, (
          If[sellTransactionPrice == 0, sellMissing++, sellMissing = 0];
          ) &}
    }
  ]
Seller
(* Function sellerMeetOneBuyer
     Params : aSeller, aBuyer
  *)
sellerMeetOneBuyer[aSeller_, aBuyer_] := Module[
      {tRealQuality, tVirtualQuality, tempPrice},
      (tRealQuality = sellGetCurrentQuality[aSeller];
        tVirtualQuality = sellTellMeYourQuality[aSeller];
        tempPrice = sellGetPrice[aSeller];
        buySetFromSellerPriceVirtualQRealQ[aBuyer, tempPrice, tVirtualQuality,
          tRealQuality];)
      ];
(* Function tryToExchange
     Params : aSeller, aBuyer
  *)
tryToExchange[aSeller_, aBuyer_] := Module[
      {}, (
          If[(buyLikeMyPrice[aBuyer]),
              (buyMakeTransaction[aBuyer];
                sellMakeTransaction[aSeller];
                ),
              (buyDontMakeTransaction[aBuyer];
                sellDontMakeTransaction[aSeller];)
              ];);
      ];
(* Function Avg
     Params : aList
  *)
AvgNoZero[aList_] := Module[
      {ss, jj}, (
        ss = 0; jj = 0;
        Table[(ss += aList[[i]]; If[aList[[i]] == 0, , jj++];), {i,
            Length[aList]}];
        If[jj == 0, jj = 0.001,];
        Return[(ss / jj)];)
      ];

(* Function setupModel
     Params :
      agentNumber, sincerityMin, sincerityMax, faithMin, faithMax, \
basisPrice,
                    missingCorrection, faithCorrection
  *)
setupModel[agentNumber_, sincerityMin_, sincerityMax_, faithMin_, faithMax_,
      basisPrice_, missingCorrection_, faithCorrection_] := Module[
      {buyersList, sellersList},
      buyersList =
        Table[new[Buyer, faithMin, faithMax, basisPrice, missingCorrection,
            faithCorrection] , {agentNumber}];
      sellersList =
        Table[new[Seller, sincerityMin, sincerityMax, basisPrice,
            missingCorrection] , {agentNumber}];
      Return[{buyersList, sellersList}];
      ];
(* Function runModel
     Params :
      agentNumber, sincerityMin, sincerityMax, faithMin, faithMax, \
basisPrice,
                    missingCorrection, faithCorrection
  *)
runModel[buyers_, sellers_, cycles_] := Module[
      {howManyAgents, tempTable, k, results}, (
        howManyAgents = Length[buyers];
        results = 0;
        For[i = 0, i < cycles, i++, (
            Table[buyInitCycle[buyers[[j]]], {j, howManyAgents}];
            Table[sellInitCycle[sellers[[j]]], {j, howManyAgents}];
            Table[sellThinkPrice[sellers[[j]]], {j, howManyAgents}];
            Table[
                          (k = Random[Integer, {1, howManyAgents}];
```

```
                    sellerMeetOneBuyer[sellers[[j]], buyers[[k]]];
                    buyThinkPrice[buyers[[k]]];
                    tryToExchange[sellers[[j]], buyers[[k]]];)
                , {j, howManyAgents}];

        Table[buyMakeCorrections[buyers[[j]]], {j, howManyAgents}];
        Table[sellMakeCorrections[sellers[[j]]], {j, howManyAgents}];

        (*Probe section*)


        tempTable =
          Table[buyGetTransactionPrice[buyers[[j]]], {j, howManyAgents}];
        (*ListPlot[tempTable, PlotJoined -> True];*)
        results = Flatten[{results, AvgNoZero[tempTable]}];
        (*Debug section*)
        (*
          Print[Table[buyGetPrice[buyers[[j]]], {j, howManyAgents}]];

          Print[Table[
            buyGetSellerPrice[buyers[[j]]], {j, howManyAgents}]];

          Print[Table[
            sellGetTransactionPrice[sellers[[j]]], {j,
              howManyAgents}]];
          *)
        )];
      ListPlot[results, PlotJoined -> True];
      Print["Overall medium price: ", IntegerPart[AvgNoZero[results]]];
      )
    ];
perfectMarket = setupModel[50, 1, 1, 1, 1, 1070, 10, 0.02];
akerloffMarket = setupModel[50, 1, 1.9, 0.3, 1, 1070, 10, 0.02];
runModel[First[perfectMarket], Last[perfectMarket], 30];
runModel[First[akerloffMarket], Last[akerloffMarket], 30];
```

APPENDICE C – Il codice di *jAkerloff*

AkerlofModelSwarm.java

```
// Java Akerlof application. Copyright © 2000 Michele Sonnessa.
// This library is distributed without any warranty; without even the
// implied warranty of merchantability or fitness for a particular
// purpose.  See file COPYING for details and terms of copying.

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.activity.ActionForEach;

import swarm.collections.List;
import swarm.collections.ListImpl;
import swarm.collections.ListShufflerImpl;

import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;

public class AkerlofModelSwarm extends SwarmImpl
{
  // simulation parameters
  public int agentNumber;
  public float envBasisPrice;
  public float envFaithCorrection;
  public int envMissingCorrection;

  public float envSincerityMin, envSincerityMax;
  public float envFaithMin, envFaithMax;

  public ActionGroup modelActions;
  public Schedule modelSchedule;

  public List buyerList;
  public List sellerList;
  private ListShufflerImpl listShuffler;

  public List getBuyerList () {  return buyerList;  }
  public List getSellerList () {  return sellerList;  }

  public AkerlofModelSwarm (Zone aZone) {
   super (aZone);

   agentNumber = 100;
   envBasisPrice = 1200;
   envFaithCorrection = (float) 0.02;
   envMissingCorrection = 50;

   envSincerityMin = (float) 1.0;
   envSincerityMax = (float) 1.5;
   envFaithMin = (float) 0.3;
   envFaithMax = (float) 1.0;

   class AkerlofModelProbeMap extends EmptyProbeMapImpl {
    private VarProbe probeVariable (String name) {
     return
      Globals.env.probeLibrary.getProbeForVariable$inClass
      (name, AkerlofModelSwarm.this.getClass ());
    }
    private MessageProbe probeMessage (String name) {
     return
      Globals.env.probeLibrary.getProbeForMessage$inClass
      (name, AkerlofModelSwarm.this.getClass ());
    }
    private void addVar (String name) {
     addProbe (probeVariable (name));
    }
    private void addMessage (String name) {
     addProbe (probeMessage (name));
    }
    public AkerlofModelProbeMap (Zone _aZone, Class aClass) {
```

```
      super (_aZone, aClass);
      addVar ("agentNumber");
      addVar ("envBasisPrice");
      addVar ("envFaithCorrection");
      addVar ("envMissingCorrection");
      addVar ("envSincerityMin");
      addVar ("envSincerityMax");
      addVar ("envFaithMin");
      addVar ("envFaithMax");
    }
  }

  Globals.env.probeLibrary.setProbeMap$For
    (new AkerlofModelProbeMap (aZone, getClass ()), getClass ());
 }

 public Object meetAgents () {
        int i;
        listShuffler.shuffleWholeList(buyerList);
        for (i = 0; i < agentNumber; i++)
                ((Seller)sellerList.atOffset(i)).setCurrentBuyer(
                        (Buyer)buyerList.atOffset(i));
        return this;
}


 public Object buildObjects ()
 {
  int i;

  super.buildObjects();
  buyerList = new ListImpl (getZone ());
  sellerList = new ListImpl (getZone ());
  listShuffler = new ListShufflerImpl(getZone());

  for (i = 0; i < agentNumber; i++)
        {
    Buyer aBuyer =  new Buyer (envFaithMin, envFaithMax,
                               envBasisPrice, envFaithCorrection,
                               envMissingCorrection);

    buyerList.addLast (aBuyer);

    Seller aSeller =  new Seller (envSincerityMin, envSincerityMax,
                                  envBasisPrice, envMissingCorrection);
    sellerList.addLast (aSeller);

  }
  return this;
}

public Object buildActions () {
  super.buildActions();
  modelActions = new ActionGroupImpl (getZone ());

  try {
    modelActions.createActionTo$message
      (this, new Selector (this.getClass (), "meetAgents", false));
  } catch (Exception e) {
    System.err.println ("Exception meetAgents: " + e.getMessage ());
  }

  try {
    ActionForEach actionForEach;
    Selector sel =
      new Selector (Class.forName ("Buyer"), "step1", false);
    actionForEach =
      modelActions.createActionForEach$message (buyerList, sel);
  }
  catch (Exception e) {
    System.err.println("Exception BuyerStep1: " + e.getMessage ());
  }

  try {
    ActionForEach actionForEach;
    Selector sel =
      new Selector (Class.forName ("Seller"), "step1", false);
    actionForEach =
      modelActions.createActionForEach$message (sellerList, sel);
  }
  catch (Exception e) {
    System.err.println("Exception SellerStep1: " + e.getMessage ());
  }

  try {
```

```
  ActionForEach actionForEach;
  Selector sel =
    new Selector (Class.forName ("Buyer"), "step2", false);
  actionForEach =
    modelActions.createActionForEach$message (buyerList, sel);
}
catch (Exception e) {
  System.err.println("Exception BuyerStep2: " + e.getMessage ());
}

try {
  ActionForEach actionForEach;
  Selector sel =
    new Selector (Class.forName ("Seller"), "step2", false);
  actionForEach =
    modelActions.createActionForEach$message (sellerList, sel);
}
catch (Exception e) {
  System.err.println("Exception SellerStep2: " + e.getMessage ());
}

try {
  ActionForEach actionForEach;
  Selector sel =
    new Selector (Class.forName ("Buyer"), "makeCorrections", false);
  actionForEach =
    modelActions.createActionForEach$message (buyerList, sel);
}
catch (Exception e) {
  System.err.println("Exception BuyerMakeCorrections: " + e.getMessage ());
}

try {
  ActionForEach actionForEach;
  Selector sel =
    new Selector (Class.forName ("Seller"), "makeCorrections", false);
  actionForEach =
    modelActions.createActionForEach$message (sellerList, sel);
}
catch (Exception e) {
  System.err.println("Exception SellerMakeCorrections: " + e.getMessage ());
}
    modelSchedule = new ScheduleImpl (getZone (), 1);
    modelSchedule.at$createAction (0, modelActions);
    return this;
  }

  public Activity activateIn (Swarm swarmContext) {
    super.activateIn (swarmContext);
    modelSchedule.activateIn (this);
    return getActivity ();
  }
}
```

AkerlofObserverSwarm.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.collections.List;

import swarm.objectbase.Swarm;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;

import swarm.analysis.EZGraph;
import swarm.analysis.EZGraphImpl;

import swarm.simtoolsgui.GUISwarm;
import swarm.simtoolsgui.GUISwarmImpl;

public class AkerlofObserverSwarm extends GUISwarmImpl {
        public int displayFrequency;
        public ActionGroup displayActions;
```

```
public Schedule displaySchedule;
public AkerlofModelSwarm akerlofModelSwarm;
public EZGraph exchangesGraph;

public AkerlofObserverSwarm (Zone aZone) {
        super(aZone);
        displayFrequency = 1;
        class ObserverProbeMap extends EmptyProbeMapImpl {
                private VarProbe probeVariable (String name) {
                        return
                  Globals.env.probeLibrary.getProbeForVariable$inClass
                          (name, AkerlofObserverSwarm.this.getClass ());
                }
                private MessageProbe probeMessage (String name) {
                        return
                Globals.env.probeLibrary.getProbeForMessage$inClass
                          (name, AkerlofObserverSwarm.this.getClass ());
                }
                private void addVar (String name) {
                        addProbe (probeVariable (name));
                }
                private void addMessage (String name) {
                        addProbe (probeMessage (name));
                }
                public ObserverProbeMap (Zone _aZone, Class aClass) {
                        super (_aZone, aClass);
                        addVar ("displayFrequency");
                        addMessage ("addAgentGraph:");
                }
        }

        Globals.env.probeLibrary.setProbeMap$For
          (new ObserverProbeMap (aZone, getClass ()), getClass ());
}

public Object _exchangesGraphDeath_ (Object caller) {
        exchangesGraph.drop ();
        exchangesGraph = null;
        return this;
}

public Object buildObjects () {
    super.buildObjects ();
    akerlofModelSwarm = new AkerlofModelSwarm (getZone ());
    Globals.env.createArchivedProbeDisplay (akerlofModelSwarm,
                                "akerlofModelSwarm");
    Globals.env.createArchivedProbeDisplay (this, "observerSwarm");
    getControlPanel ().setStateStopped ();
    akerlofModelSwarm.buildObjects ();

exchangesGraph = new EZGraphImpl
  (getZone (),
   "Transactions",
   "time", "price",
   "exchangesGraph");

        try {
          exchangesGraph.enableDestroyNotification$notificationMethod
            (this, new Selector (getClass (),
                          "_exchangesGraphDeath_",
                          false));
        } catch (Exception e) {
          System.err.println ("Exception _exchangesGraphDeath_: "
                        + e.getMessage ());
        }

        try {
exchangesGraph.createAverageSequence$withFeedFrom$andSelector
            ("transactionPrice", akerlofModelSwarm.getSellerList (),
             new Selector (Class.forName ("Seller"), "getTransactionPrice",
                        false));
        } catch (Exception e) {
          System.err.println ("Exception getTransactionPrice: "
                        + e.getMessage ());
        }

        try {
exchangesGraph.createAverageSequence$withFeedFrom$andSelector
            ("sellerPrice", akerlofModelSwarm.getSellerList (),
             new Selector (Class.forName ("Seller"), "getPrice",
                        false));
        } catch (Exception e) {
          System.err.println ("Exception getPrice: "
                        + e.getMessage ());
```

```
        }

        try {
exchangesGraph.createAverageSequence$withFeedFrom$andSelector
            ("buyerPrice", akerlofModelSwarm.getBuyerList (),
             new Selector (Class.forName ("Buyer"), "getPrice",
                           false));
        } catch (Exception e) {
          System.err.println ("Exception getPrice: "
                              + e.getMessage ());
        }

        return this;
}

public Object _update_ ()  {
        if (exchangesGraph != null)
          exchangesGraph.step ();
        return this;
}

public Object buildActions () {
        super.buildActions();
        akerlofModelSwarm.buildActions();
        displayActions = new ActionGroupImpl (getZone());

        try {
          displayActions.createActionTo$message
            (this, new Selector (getClass (), "_update_", false));

          // Schedule the update of the probe displays
          displayActions.createActionTo$message
            (Globals.env.probeDisplayManager,
             new Selector (Globals.env.probeDisplayManager.getClass (),
                           "update", true));

          displayActions.createActionTo$message
            (getActionCache (), new Selector
              (getActionCache ().getClass (), "doTkEvents", true));
        } catch (Exception e) {
          System.err.println ("Exception in setting up displayActions : "
                              + e.getMessage ());
        }

        displaySchedule = new ScheduleImpl (getZone (), displayFrequency);
        displaySchedule.at$createAction (0, displayActions);
        return this;
}

public Activity activateIn (Swarm swarmContext) {
        super.activateIn (swarmContext);
        akerlofModelSwarm.activateIn (this);
        displaySchedule.activateIn (this);
        return getActivity();
}

public Object addAgentGraph (int number) {
  if (exchangesGraph != null)
    try {
        Seller aSeller =
        (Seller)akerlofModelSwarm.getSellerList().atOffset(number);
      exchangesGraph.createSequence$withFeedFrom$andSelector
        ("Seller" + number, aSeller, new Selector (aSeller.getClass (),
                                   "getTransactionPrice", false));
    } catch (Exception e) {
      System.err.println ("Exception getTransactionPrice: " +
                          e.getMessage());
    }
  return this;
}
public void drop () {
        if (exchangesGraph != null)
          exchangesGraph.disableDestroyNotification ();
        super.drop ();
}
}
```

Buyer.java

```
import swarm.Globals;

public class Buyer {
        private float faith;
```

```
private float price;
private float qualityComunicated;
private int qualityReal;
private int missing;
private float sellerPrice;
private float transactionPrice;

private float envBasisPrice;
private float envFaithCorrection;
private int envMissingCorrection;

public Buyer (float faithMin, float faithMax,
                        float basisPrice, float faithCorrection,
                        int missingCorrection) {

        faith = (float)
        Globals.env.uniformDblRand.getDoubleWithMin$withMax(
                                        faithMin, faithMax);
        price = (float) 0.0;
        qualityComunicated = (float) 1.0;
        qualityReal = 1;
        missing = 0;
        sellerPrice = (float) 0.0;
        transactionPrice = (float) 0.0;

        envBasisPrice = basisPrice;
        envFaithCorrection = faithCorrection;
        envMissingCorrection = missingCorrection;
}

public float getFaith() { return faith; }
public float getPrice() { return price; }
public int getMissing() { return missing; }
public float getTransactionPrice() { return transactionPrice; }
public float getSellerPrice() { return sellerPrice; }

public Object thinkPrice() {
        price = envBasisPrice * qualityComunicated * faith +
                        envMissingCorrection * missing;
        if (price < envBasisPrice)
                price = envBasisPrice;
        return this;
}

public Object initCycle () {
        transactionPrice = 0;
        return this;
}

public Object makeCorrections() {
        faith -= (qualityComunicated - qualityReal) * envFaithCorrection;
        if (transactionPrice > 0)
                missing = 0;
        else
                missing++;
        return this;
}

public Object setSellerPrice$Quality$QualityReal(
                float aPrice, float aQuality, int aQReal) {
        sellerPrice = aPrice;
        qualityComunicated = aQuality;
        qualityReal = aQReal;
        return this;
}

public Object step1 () {
        initCycle();
        return this;
}

public Object step2 () {
        thinkPrice();
        return this;
}
public boolean doYouLikeMyPrice() {
        if (price >= sellerPrice)
        {
                transactionPrice = sellerPrice;
                return true;
        }
        else
                return false;
}
```

```
}
```

Seller.java

```
import swarm.Globals;

public class Seller {
        private float sincerity;
        private float price;
        private int quality;
        private int missing;
        private float transactionPrice;

        private float envBasisPrice;
        private int envMissingCorrection;

        private Buyer myBuyer;

        public Seller (float sincerityMin, float sincerityMax,
                        float basisPrice, int missingCorrection) {

                sincerity = (float)
                Globals.env.uniformDblRand.getDoubleWithMin$withMax(
                                        sincerityMin, sincerityMax);
                price = (float) 0.0;
                quality = 1;
                missing = 0;
                transactionPrice = (float) 0.0;
                envBasisPrice = basisPrice;
                envMissingCorrection = missingCorrection;
                myBuyer = null;

        }

        public float getSincerity() { return sincerity; }
        public float getPrice() { return price; }
        public int getMissing() { return missing; }
        public float getTransactionPrice() { return transactionPrice; }

        public Object initCycle () {
                quality = Globals.env.uniformIntRand.getIntegerWithMin$withMax(
                                                                1, 2);
                transactionPrice = 0;
                return this;
        }

        public Object thinkPrice() {
                price = envBasisPrice * quality * sincerity -
                                envMissingCorrection * missing;
                return this;
        }

        public Object comunicatePriceToBuyer() {
                myBuyer.setSellerPrice$Quality$QualityReal(
                        price, quality * sincerity,
                        quality);
                return this;
        }

        public Object makeCorrections() {
                if (transactionPrice > 0)
                        missing = 0;
                else
                        missing++;
                return this;
        }

        public Object setCurrentBuyer(Buyer aBuyer) {
                myBuyer = aBuyer;
                return this;
        }

}

        public Object step1 () {
                initCycle();
                thinkPrice();
                comunicatePriceToBuyer();

                return this;
        }

        public Object step2 () {
                if (myBuyer.doYouLikeMyPrice())
                        transactionPrice = price;
```

```
                return this;
        }
}
```

StartAkerlof.java

```
// Java Akerlof application. Copyright © 2000 Michele Sonnessa.
// This library is distributed without any warranty; without even the
// implied warranty of merchantability or fitness for a particular
// purpose.  See file COPYING for details and terms of copying.

import swarm.Globals;

public class StartAkerlof {
        public static void main (String[] args) {

        Globals.env.initSwarm ("jAkerlof", "0.0.1", "supersonny@libero.it", args);

                AkerlofObserverSwarm topLevelSwarm =
                        new AkerlofObserverSwarm (Globals.env.globalZone);
                        Globals.env.setWindowGeometryRecordName
                                        (topLevelSwarm, "topLevelSwarm");

                topLevelSwarm.buildObjects ();
                topLevelSwarm.buildActions ();
                topLevelSwarm.activateIn (null);
                topLevelSwarm.go ();
                topLevelSwarm.drop ();
   }
}
```

APPENDICE D – Il codice di *jBogliettiSwarm*

OBSERVER

BarChart.java

```
/**
 * @version 1.20 25 Mar 1998 - June 2000
 * @author Cay Horstmann - Michele Sonnessa
 *
 * This code is taken from Horstman's chart applet
 * in Core Java 1.2 (1999), McGraw-Hill,
 * converted and modified
 * according to freeware licence by Sonnessa (2000)
 * into an application component
 *
 * E-mail: sonnessa@tiscalinet.it
 */

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class BarChart
{

        private class ChartPanel extends JPanel
        {  public ChartPanel(double[] v, String[] n, String t)
           {  names = n;
                values = v;
                title = t;
           }

           public void paintComponent(Graphics g)
           {  super.paintComponent(g);
                if (values == null || values.length == 0) return;
                int i,j=0;
double minValue = 0;
double maxValue = 0;
for (i = 0; i < values.length; i++)
{  if (minValue > values[i]) minValue = values[i];
        if (maxValue < values[i]) maxValue = values[i];
}

Dimension d = getSize();
int clientWidth = d.width;
int clientHeight = d.height;
int barWidth = clientWidth / values.length;
Font titleFont
        = new Font("SansSerif", Font.BOLD, 20);
FontMetrics titleFontMetrics
        = g.getFontMetrics(titleFont);
Font labelFont
        = new Font("SansSerif", Font.PLAIN, 10);
FontMetrics labelFontMetrics
        = g.getFontMetrics(labelFont);
int titleWidth
        = titleFontMetrics.stringWidth(title);
int y = titleFontMetrics.getAscent();
int x = (clientWidth - titleWidth) / 2;
g.setFont(titleFont);
g.drawString(title, x, y);
int top = titleFontMetrics.getHeight();
int bottom = labelFontMetrics.getHeight();
if (maxValue == minValue) return;
double scale = (clientHeight - top - bottom)
        / (maxValue - minValue);
y = clientHeight - labelFontMetrics.getDescent();
g.setFont(labelFont);
for (i = 0; i < values.length; i++)
{  int x1 = i * barWidth + 1;
        int y1 = top;
        int height = (int)(values[i] * scale);
        if (values[i] >= 0)
                y1 += (int)((maxValue - values[i]) * scale);
        else
        {  y1 += (int)(maxValue * scale);
                height = -height;
        }
```

```
                    if (j==0)
                    {
                            g.setColor(Color.red);
                            j=1;
                    }
                    else
                    {
                            g.setColor(Color.blue);
                            j=0;
                    }
                    g.fillRect(x1, y1, barWidth - 2, height);
                    g.setColor(Color.black);
                    g.drawRect(x1, y1, barWidth - 2, height);
                    int labelWidth
                            = labelFontMetrics.stringWidth(names[i]);
                    x = i * barWidth + (barWidth - labelWidth) / 2;
                    g.drawString(names[i], x, y);
            }
    }

    private double[] values;
    private String[] names;
    private String title;

    public double[] getValues() { return values; }
    public String[] getNames() { return names; }
    public String getTitle() { return title; }

            public void setBarTitle(String newTitle) {
                    title = newTitle;
                    return;
            }
}

private class ChartFrame extends JFrame
{
            private ChartPanel cPanel;

            public ChartFrame(String windowTitle, double[] v,
                                    String[] n, String t)
            {
                    setTitle(windowTitle);
                    setSize(300, 200);
                    addWindowListener(new WindowAdapter()
                    {
                            public void windowClosing(WindowEvent e)
                            {  System.exit(0);
                            }
                     } );
                    Container contentPane = getContentPane();
                    cPanel = new ChartPanel(v, n, t);
                    contentPane.add(cPanel);
            }

            public double[] getValues() { return cPanel.getValues(); };
            public String[] getNames() { return cPanel.getNames(); };
            public String getTitle() { return cPanel.getTitle(); }

                    public void setBarTitle(String newTitle) {
                            cPanel.setBarTitle(newTitle);
                            return;
                    }
            }
            private ChartFrame cFrame;

            BarChart(String windowTitle, double[] v, String[] n, String t)
            {
                    cFrame = new ChartFrame(windowTitle,v,n,t);
                    JFrame frame = cFrame;
            frame.show();
            }

public double[] getValues() { return cFrame.getValues(); };
public String[] getNames() { return cFrame.getNames(); };
public String getTitle() { return cFrame.getTitle(); }

            public void setBarTitle(String newTitle) {
                    cFrame.setBarTitle(newTitle);
                    return;
            }

public void update() { cFrame.repaint(); return; }
```

```
  public void drop() {
        cFrame.dispose();
   cFrame = null;
        return;
  }

}
```

BogliettiObserverSwarm.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.collections.List;

import swarm.objectbase.Swarm;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;

import swarm.analysis.EZGraph;
import swarm.analysis.EZGraphImpl;

import swarm.simtoolsgui.GUISwarm;
import swarm.simtoolsgui.GUISwarmImpl;

public class BogliettiObserverSwarm extends GUISwarmImpl {
        public int displayFrequency;
        public boolean useDoubleModel;

        public ActionGroup displayActions;
        public Schedule displaySchedule;

        public BogliettiModelSwarm bogliettiModelSwarm;
public ForecastModelSwarm BPCTModelSwarm;

/** graphing widget */
public EZGraph forecastError;
public EZGraph customersBargained;
public BarChart sellerForecasts;

public int weekToStopGraphForecasting;
private int i;

/** Constructor for class */
public BogliettiObserverSwarm (Zone aZone) {
        super(aZone);

        displayFrequency = 2;
        useDoubleModel = true;
        weekToStopGraphForecasting = 24;

        class BogliettiObserverProbeMap extends EmptyProbeMapImpl {
          private VarProbe probeVariable (String name) {
            return
              Globals.env.probeLibrary.getProbeForVariable$inClass
              (name, BogliettiObserverSwarm.this.getClass ());
          }
          private MessageProbe probeMessage (String name) {
            return
              Globals.env.probeLibrary.getProbeForMessage$inClass
              (name, BogliettiObserverSwarm.this.getClass ());
          }
          private void addVar (String name) {
            addProbe (probeVariable (name));
          }
          private void addMessage (String name) {
            addProbe (probeMessage (name));
          }
          public BogliettiObserverProbeMap (Zone _aZone, Class aClass) {
            super (_aZone, aClass);
            addVar ("displayFrequency");
                addVar ("weekToStopGraphForecasting");
            addVar ("useDoubleModel");
          }
        }
```

```
            Globals.env.probeLibrary.setProbeMap$For
          (new BogliettiObserverProbeMap (aZone, getClass ()), getClass ());
}

public Object _forecastErrorDeath_ (Object caller) {
          forecastError.drop ();
          forecastError = null;
          return this;
}

public Object _customersBargainedDeath_ (Object caller) {
          customersBargained.drop ();
          customersBargained = null;
          return this;
}

public Object buildObjects () {
          super.buildObjects ();
          boglietti ModelSwarm = new Boglietti ModelSwarm (getZone ());
          BPCTModelSwarm = new ForecastModelSwarm
                                    (boglietti ModelSwarm, getZone ());

          Globals.env.createArchivedProbeDisplay (boglietti ModelSwarm,
                                    "boglietti ModelSwarm");
          Globals.env.createArchivedProbeDisplay (BPCTModelSwarm,
                                    "BPCTModelSwarm");
          Globals.env.createArchivedProbeDisplay (this,
                                    "boglietti ObserverSwarm");
          getControlPanel ().setStateStopped ();
          boglietti ModelSwarm.buildObjects ();
          BPCTModelSwarm.buildObjects();

   forecastError = new EZGraphImpl
     (getZone (),
      "Global forecasts",
      "half weeks", "orders",
      "forecastError");

   Globals.env.setWindowGeometryRecordName (
                          forecastError, "forecastError");
          try {
            forecastError.enableDestroyNotification$notificationMethod
              (this, new Selector (getClass (),
                          "_forecastErrorDeath_",
                          false));
          } catch (Exception e) {
            System.err.println ("Exception _forecastErrorDeath_: "
                          + e.getMessage ());
          }

try {
  forecastError.createSequence$withFeedFrom$andSelector
    ("sales", boglietti ModelSwarm.getTheForecaster (),
     new Selector (Class.forName ("Forecaster"), "getCurrentSum",
                false));
} catch (Exception e) {
  System.err.println ("Exception getCurrentSum: "
                + e.getMessage ());
}

try {
  forecastError.createSequence$withFeedFrom$andSelector
    ("forecast", boglietti ModelSwarm.getTheForecaster (),
     new Selector (Class.forName ("Forecaster"), "getCurrentForecast",
                false));
} catch (Exception e) {
  System.err.println ("Exception getCurrentForecast: "
                + e.getMessage ());
}


          customersBargained = new EZGraphImpl
            (getZone (),
             "# of customers bargained",
             "half weeks", "customers",
             "customersBargained");

          Globals.env.setWindowGeometryRecordName
                    (customersBargained, "customersBargained");
try {
  customersBargained.enableDestroyNotification$notificationMethod
    (this, new Selector (getClass (),
                "_customersBargainedDeath_",
                false));
```

```
    } catch (Exception e) {
      System.err.println ("Exception _customersBargainedDeath_: "
                          + e.getMessage ());
    }
    try {
      customersBargained.createSequence$withFeedFrom$andSelector
        ("error", bogliettiModelSwarm.getEnvironment(),
         new Selector (Class.forName ("Environment"),
                                "getCustomersBargained", false));
    } catch (Exception e) {
      System.err.println ("Exception getCustomersBargained: "
                          + e.getMessage ());
    }

double[] values = new double[bogliettiModelSwarm.sellersNumber * 2];
String[] labels = new String[bogliettiModelSwarm.sellersNumber * 2];

            for (i = 0; i < bogliettiModelSwarm.sellersNumber; i++)
            {
            labels[i * 2] =
                        bogliettiModelSwarm.getInterface().getSellerKey(i);
            labels[i * 2 + 1] = "-";
            }
    sellerForecasts = new BarChart("", values,
    labels,"Forecast vs. sales of single sellers.");
    return this;
}

public Object updateSellerForecasts() {
            double[] x = sellerForecasts.getValues();
            List sellersList = bogliettiModelSwarm.getSellersList();
            //ModelInterface inf = bogliettiModelSwarm.getInterface();
            int i;

            boolean updateForecasts = true;
            if (bogliettiModelSwarm.environ.getCurrentTime() >=
                                weekToStopGraphForecasting)
                        updateForecasts = false;

            for (i = 0; i < bogliettiModelSwarm.sellersNumber; i++)
            {
                        if (updateForecasts)
                                    x[i * 2] = (double)
                                    ((Seller)sellersList.atOffset(i)).getTotalForecast();
                        x[i * 2 + 1] = (double)
                                    ((Seller)sellersList.atOffset(i)).getTotalSum();
            }
            String s = new String("Forecast/sales of week # " +
                        bogliettiModelSwarm.environ.getCurrentTime());
            sellerForecasts.setBarTitle(s);
            sellerForecasts.update();
            return this;
}

public Object _update_ ()  {
            if (sellerForecasts != null)
                        updateSellerForecasts();
            if (forecastError != null)
                         forecastError.step ();
            if (customersBargained != null)
              customersBargained.step ();
            return this;
}

public Object buildActions () {
            super.buildActions();
            bogliettiModelSwarm.buildActions();
            BPCTModelSwarm.buildActions();
            displayActions = new ActionGroupImpl (getZone());

    try {
      displayActions.createActionTo$message
        (this, new Selector (getClass (), "_update_", false));
      displayActions.createActionTo$message
        (Globals.env.probeDisplayManager,
         new Selector (Globals.env.probeDisplayManager.getClass (),
                     "update", true));
      displayActions.createActionTo$message
        (getActionCache (), new Selector
          (getActionCache ().getClass (), "doTkEvents", true));
      displayActions.createActionTo$message
        (this,
         new Selector (getClass (),
                     "checkToStop", true));
```

```
    } catch (Exception e) {
      System.err.println ("Exception in setting up displayActions : "
                          + e.getMessage ());
    }
    displaySchedule = new ScheduleImpl (getZone (), displayFrequency);
    displaySchedule.at$createAction (0, displayActions);
    return this;
}

public Activity activateIn (Swarm swarmContext) {
    super.activateIn (swarmContext);
    boglietti ModelSwarm.activateIn (this);
            if (useDoubleModel)
                BPCTModelSwarm.activateIn (this);
    displaySchedule.activateIn (this);
    return getActivity();
}

public void drop () {
            if (forecastError != null)
              forecastError.disableDestroyNotification ();
            if (customersBargained != null)
              customersBargained.disableDestroyNotification ();
    if (sellerForecasts != null)
      sellerForecasts.drop ();
            super.drop ();
}

// dealing with stopping conditions
public Object checkToStop ()          {
     if (boglietti ModelSwarm.lastWeek <
            boglietti ModelSwarm.environ.getCurrentTime())  {
            System.out.println("Stopping at week number " +
                        boglietti ModelSwarm.environ.getCurrentTime() + ".");

              if (!BPCTModelSwarm.onlyVerificationRun)
                        BPCTModelSwarm.orderAgentsToSaveItsWeight();
            getControlPanel ().setStateStopped ();
  }
  return this;
}
```

```
}
```

Evnironment.java

```
import swarm.Globals;
import swarm.collections.List;

import java.util.ArrayList;
import java.io.*;

public class Environment {
            private int currentTime;
            private int customerBargained;
            private List sellersList = null;
            private int stopForecast;

            public Environment (int timeToStopForecast) {
                        currentTime = -1;
                        customerBargained = 0;
                        stopForecast = timeToStopForecast;
            }

            public Object setSellerList (List aSellersList) {
                        sellersList = aSellersList;
                        return this;
            }

            public List getSellerList () { return sellersList; }
            public int getCurrentTime () { return currentTime; }
            public int getStopForecast () { return stopForecast;}
            public boolean isNoMoreTheForecastTime() {
                        return (currentTime >= stopForecast);       }

            public boolean isTheForecastTime() {
                        return (currentTime == stopForecast);
            }

            public Object updateTime () {
                        currentTime++;
                        return this;
            }
```

```
	public Object setCustomersBargained (int howManyCustomers) {
		customerBargained = howManyCustomers;
		return this;
	}

	public int getCustomersBargained () { return customerBargained; }

	public Object sendOrders$toSellerID(ArrayList orders, long aSellerID) {
		Seller aSeller;
		aSeller = getASeller(aSellerID);
		aSeller.receiveOrdersFromACustomer(orders);
		return this;
	}

	private Seller getASeller (long aSellerID) {
		int i;

		for (i=0; i < sellersList.getCount(); i++)
		  if (((Seller)sellersList.atOffset(i)).getSellerID() == aSellerID)
				return (Seller)sellersList.atOffset(i);
		return null;
	}

}
```

MatrixSerialize.java

```
import java.io.*;
import java.util.*;
import java.text.*;

public final class MatrixSerialize {
	static final public boolean saveMatrix$withX$andY$intoFileName(
		long[][] matrix, int xDim,int yDim, String fileName) {
	BufferedWriter outFile;
		int i,j;

	try {
		outFile= new BufferedWriter(new FileWriter(fileName));
	} catch (Exception e) {
		System.err.println("Error while trying to open file " + fileName +
				":" + e.getMessage());
		return false;
	}

	try {
			String s = Integer.toString(xDim);
			outFile.write(s,0,s.length());
			outFile.newLine();
				s = Integer.toString(yDim);
				outFile.write(s,0,s.length());
				outFile.newLine();
	} catch (Exception e) {
		System.err.println("Error writing file " + fileName +
				": " + e.getMessage());
		}

	try {
		for (i=0;i<xDim;i++)
		{
			for (j=0;j<yDim;j++)
			{
				String s = Long.toString(matrix[i][j]) + " ";
				outFile.write(s,0,s.length());
			}
	outFile.newLine();
		}
		outFile.close();
	} catch (Exception e) {
		System.err.println("Error writing file " + fileName + ": " +
e.getMessage());
		}

		return true;
	}

static final public boolean saveMatrix$withX$andY$intoFileName(
	double[][] matrix, int xDim,int yDim, String fileName) {
	BufferedWriter outFile;
		int i,j;

	try {
```

```
                outFile= new BufferedWriter(new FileWriter(fileName));
        } catch (Exception e) {
                System.err.println("Error while trying to open file " + fileName +
                                        ": " + e.getMessage());
                return false;
        }

        try {
                        String s = Integer.toString(xDim);
                        outFile.write(s,0,s.length());
                        outFile.newLine();
                                s = Integer.toString(yDim);
                                outFile.write(s,0,s.length());
                                outFile.newLine();
        } catch (Exception e) {
                System.err.println("Error writing file " + fileName + ": " +
e.getMessage());
                }

        try {
                for (i=0;i<xDim;i++)
                {
                        for (j=0;j<yDim;j++)
                        {
                                String s = matrix[i][j] + " ";
                                outFile.write(s,0,s.length());
                        }
        outFile.newLine();
                }
                outFile.close();
        } catch (Exception e) {
                System.err.println("Error writing file " + fileName +
                                        ": " + e.getMessage());
                }
                return true;
        }

static public long[][] loadLongMatrixFromFileName(String fileName) {

                BufferedReader inFile;
                int xDim, yDim, i, j;
                String x;
                long[][] matrix;

                try {
                        inFile = new BufferedReader(new FileReader(fileName));

                } catch(IOException ex) {
                        System.err.println("Impossible to open " + fileName +
                                " due to: " + ex.getMessage());
                        return null;
                }

                try {
                        xDim = Integer.parseInt(inFile.readLine());
                        yDim = Integer.parseInt(inFile.readLine());
                } catch(IOException ex) {
                        System.err.println("Impossible read " + fileName +
                                        " due to: " + ex.getMessage());
                        return null;
                }

                matrix = new long[xDim][yDim];
        try {
                for (i=0;i<xDim;i++)
                {
                        x = inFile.readLine();
                        StringTokenizer t = new StringTokenizer(x, " ");

                        for (j=0;j<yDim;j++)
                                matrix[i][j] = Long.parseLong(t.nextToken());
                }
                        inFile.close();
        } catch (Exception e) {
                System.err.println("Error reading file " + fileName +
                                ": " + e.getMessage());
                }
                return matrix;
        }

static public double[][] loadDoubleMatrixFromFileName(String fileName) {

                BufferedReader inFile;
                int xDim, yDim, i, j;
```

```
		String x;
		double[][] matrix;
		try {
			inFile = new BufferedReader(new FileReader(fileName));
		} catch(IOException ex) {
			System.err.println("Impossible to open " + fileName +
				" due to: " + ex.getMessage());
			return null;
		}

		try {
			xDim = Integer.parseInt(inFile.readLine());
			yDim = Integer.parseInt(inFile.readLine());
		} catch(IOException ex) {
			System.err.println("Impossible read " + fileName +
			" due to: " + ex.getMessage());
			return null;
		}

		matrix = new double[xDim][yDim];
	try {
		for (i=0;i<xDim;i++)
		{
			x = inFile.readLine();
			StringTokenizer t = new StringTokenizer(x, " ");
			for (j=0;j<yDim;j++)
				matrix[i][j] =
					Double.parseDouble(t.nextToken());
		}
			inFile.close();
	} catch (Exception e) {
		System.err.println("Error reading file " + fileName +
				": " + e.getMessage());
		}
		return matrix;
	}

static final public boolean saveVector$withDimension$intoFileName(

	double[] matrix, int xDim, String fileName) {
		BufferedWriter outFile;
		int i;

try {
	outFile= new BufferedWriter(new FileWriter(fileName));
} catch (Exception e) {
	System.err.println("Error while trying to open file " + fileName +
				": " + e.getMessage());
	return false;
}

try {
		String s = Integer.toString(xDim);
		outFile.write(s,0,s.length());
		outFile.newLine();
} catch (Exception e) {
	System.err.println("Error writing file " + fileName +
		": " + e.getMessage());
	}

try {
	for (i=0;i<xDim;i++)
	{
		String s = matrix[i] + " ";
		outFile.write(s,0,s.length());
	}
outFile.newLine();
	outFile.close();
} catch (Exception e) {
	System.err.println("Error writing file " + fileName +
			": " + e.getMessage());
	}
	return true;
}

static final public boolean saveVector$withDimension$intoFileName(

long[] matrix, int xDim, String fileName) {
	BufferedWriter outFile;
	int i;

try {
	outFile= new BufferedWriter(new FileWriter(fileName));
} catch (Exception e) {
```

```
                System.err.println("Error while trying to open file " + fileName +
                                        ": " + e.getMessage());
                return false;
        }

        try {
                        String s = Integer.toString(xDim);
                        outFile.write(s,0,s.length());
                        outFile.newLine();
        } catch (Exception e) {
                System.err.println("Error writing file " + fileName +
                                ": " + e.getMessage());
                }
        try {
                for (i=0;i<xDim;i++)
                {
                        String s = matrix[i] + " ";
                        outFile.write(s,0,s.length());
                }
        outFile.newLine();
                outFile.close();
        } catch (Exception e) {
                System.err.println("Error writing file " + fileName +
                                ": " + e.getMessage());
                }
                return true;
        }

static public long[] loadLongVectorFromFileName(String fileName) {

                BufferedReader inFile;
                int xDim,i;
                String x;
                long[] matrix;

                try {
                        inFile = new BufferedReader(new FileReader(fileName));
                } catch(IOException ex) {
                        System.err.println("Impossible to open " + fileName +
                                                " due to: " + ex.getMessage());
                        return null;
                }
                try {
                        xDim = Integer.parseInt(inFile.readLine());
                } catch(IOException ex) {
                        System.err.println("Impossible read " + fileName +
                                                " due to: " + ex.getMessage());
                        return null;
                }
                matrix = new long[xDim];
        try {
                        x = inFile.readLine();
                        StringTokenizer t = new StringTokenizer(x, " ");
                        for (i=0;i<xDim;i++)
                                matrix[i] = Long.parseLong(t.nextToken());
                        inFile.close();
        } catch (Exception e) {
                System.err.println("Error reading file " + fileName +
                                ": " + e.getMessage());
                }
                return matrix;
        }

static public double[] loadDoubleVectorFromFileName(String fileName) {

                BufferedReader inFile;
                int xDim, i;
                String x;
                double[] matrix;

                try {
                        inFile = new BufferedReader(new FileReader(fileName));
                } catch(IOException ex) {
                        System.err.println("Impossible to open " + fileName +
                                        " due to: " + ex.getMessage());
                        return null;
                }

                try {
                        xDim = Integer.parseInt(inFile.readLine());
                } catch(IOException ex) {
                        System.err.println("Impossible read " + fileName +
                                " due to: " + ex.getMessage());
```

```
        return null;
                }
                matrix = new double[xDim];
        try {
                        x = inFile.readLine();
                        StringTokenizer t = new StringTokenizer(x, " ");

                        for (i=0;i<xDim;i++)
                                matrix[i] = Double.parseDouble(t.nextToken());

                        inFile.close();
        } catch (Exception e) {
                System.err.println("Error reading file " + fileName +
                                ": " + e.getMessage());
                }
                return matrix;
        }
}
```

StartBoglietti.java

```
// Java Boglietti application. Copyright © 2000 Michele Sonnessa.
// This library is distributed without any warranty; without even the
// implied warranty of merchantability or fitness for a particular
// purpose.  See file COPYING for details and terms of copying.

import swarm.Globals;
/**
        * @author Michele Sonnessa
        * @version 0.90.0
        */


public class StartBoglietti {
   public static void main (String[] args) {
      Globals.env.initSwarm ("Boglietti", "2.0.1", "supersonny@libero.it", args);
                BoglettiObserverSwarm topLevelSwarm =
                        new BoglettiObserverSwarm (Globals.env.globalZone);
                Globals.env.setWindowGeometryRecordName (
                                topLevelSwarm, "topLevelSwarm");
                topLevelSwarm.buildObjects ();
                topLevelSwarm.buildActions ();
                topLevelSwarm.activateIn (null);
                topLevelSwarm.go ();
                topLevelSwarm.drop ();
   }
}
```

BASE MODEL

BoglettiModelSwarm.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.activity.ActionForEach;

import swarm.collections.List;
import swarm.collections.ListImpl;

import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;

import java.io.*;
import java.util.*;
import java.lang.*;

public class BoglettiModelSwarm extends SwarmImpl
{
  // simulation parameters
  public int customersNumber;
  public int sellersNumber;
```

```
public int linesNumber;
public int colorsNumber;

public String DSN;
public String tableName;
public int lastWeek;
public int frequencyFileOutput;

//public int lastWeek;
public int year;

public boolean readFormFile;
public boolean onlyVerificationRun;

private DataInterface dataInterface;
private Forecaster forecaster;
private SellerRuleMaster ruleMaster;
private ModelInterface theInterface;
public Environment environ;

public ActionGroup modelActions, modelActions2;
public Schedule modelSchedule;

public List customersList; // list of all customers
public List sellersList; // list of all sellers

public List getCustomersList () { return customersList; }
public List getSellersList () { return sellersList; }

public Forecaster getTheForecaster () {      return forecaster; }
public Environment getEnvironment () {     return environ; }
public ModelInterface getInterface() { return theInterface; }

public Object addCustomer (Customer aCustomer) {
  customersList.addLast(aCustomer);
  return this;
}

public BogliettiModelSwarm (Zone aZone) {
  super (aZone);
  customersNumber = 0;
          sellersNumber = 0;
          linesNumber = 0;
          colorsNumber = 0;
          frequencyFileOutput = 2;

          lastWeek = 24;

          readFormFile = true;
DSN = new String("Boglietti");
          tableName = new String("StagioneEstiva");

year = 1997;
          onlyVerificationRun          = true;

class BogliettiModelProbeMap extends EmptyProbeMapImpl {
  private VarProbe probeVariable (String name) {
    return
      Globals.env.probeLibrary.getProbeForVariable$inClass
      (name, BogliettiModelSwarm.this.getClass ());
  }
  private MessageProbe probeMessage (String name) {
    return
      Globals.env.probeLibrary.getProbeForMessage$inClass
      (name, BogliettiModelSwarm.this.getClass ());
  }
  private void addVar (String name) {
    addProbe (probeVariable (name));
  }
  private void addMessage (String name) {
    addProbe (probeMessage (name));
  }
  public BogliettiModelProbeMap (Zone _aZone, Class aClass) {
    super (_aZone, aClass);
    addVar ("readFormFile");
    addVar ("year");
    addVar ("lastWeek");
    addVar ("tableName");
    addVar ("DSN");
    addVar ("colorsNumber");
    addVar ("linesNumber");
    addVar ("sellersNumber");
    addVar ("customersNumber");
        addVar ("onlyVerificationRun");
```

```
        addVar ("frequencyFileOutput");
    }
  }
  Globals.env.probeLibrary.setProbeMap$For
    (new BoglietttiModelProbeMap (aZone, getClass ()), getClass ());
        }

public Object buildObjects ()
{
                int i;
        Customer aCustomer;
        SellerDataWarehouse aDW;
        ForecasterDataWarehouse aFDW;

  super.buildObjects();
  customersList = new ListImpl (getZone ());
  sellersList = new ListImpl (getZone ());
  environ = new Environment(lastWeek);

  if (readFormFile)
        dataInterface = new FileInterface(environ, tableName, lastWeek, year);
  else
        dataInterface = new DBInterface(environ, DSN, tableName,
                                                        lastWeek, year);
  customersNumber = 0;
  sellersNumber = 0;
  ruleMaster = new SellerRuleMaster(lastWeek,onlyVerificationRun);

  if (!dataInterface.openConnection())
    return this;

  linesNumber = (int)dataInterface.getTableNumber(DataInterface.line_table);
  if (linesNumber == -1)
  {
    System.err.println("Error in connection to DataBase trying to retrieve
                                                lines number");
    return this;
  }

  colorsNumber = (int)dataInterface.getTableNumber(DataInterface.color_table);
  if (colorsNumber == -1)
  {
    System.err.println("Error in connection to DataBase trying to retrieve
                                                colors number");
    return this;
  }

  sellersNumber = (int)dataInterface.getTableNumber(DataInterface.seller_table);
  if (sellersNumber == -1)
  {
    System.err.println("Error in connection to DataBase trying to retrieve
                                                sellers number");
    return this;
  }

        for (i=0; i < sellersNumber; i++)
        {
                aDW = new SellerDataWarehouse(linesNumber, i,lastWeek);
                sellersList.addLast( new Seller(environ, i, aDW, ruleMaster));
        }

        environ.setSellerList(sellersList);


  customersNumber =
(int)dataInterface.getTableNumber(DataInterface.customer_table);
  if (customersNumber == -1)
  {
    System.err.println("Error in connection to DataBase trying to retrieve
                                                customers number");
    return this;
  }

  for (i=0; i < customersNumber; i++)
        customersList.addLast( new Customer(environ,  i, dataInterface));

/**/System.out.println("Customers loaded # " + customersNumber);
/**/System.out.println("Sellers loaded # " + sellersNumber);
/**/System.out.println("Colors loaded # " + colorsNumber);
/**/System.out.println("Lines loaded # " + linesNumber);

 theInterface = new ModelInterface(dataInterface, customersNumber,
                                        sellersNumber, colorsNumber, linesNumber);
dataInterface.setTheInterface(theInterface);
```

```
 // Create the FORECASTER
  aFDW = new ForecasterDataWarehouse(linesNumber, 1, lastWeek);
  forecaster = new Forecaster(environ, aFDW, theInterface, frequencyFileOutput);
  return this;
}

public Object buildActions () {
  super.buildActions();
  modelActions = new ActionGroupImpl (getZone ());
          modelActions2 = new ActionGroupImpl (getZone ());
  try {
          modelActions.createActionTo$message
          (environ, new Selector (environ.getClass (), "updateTime", false));
  } catch (Exception e) {
    System.err.println ("environ updateTime: " + e.getMessage ());
  }

  try {
          modelActions.createActionTo$message
          (dataInterface, new Selector (dataInterface.getClass (), "step1", false));
  } catch (Exception e) {
    System.err.println ("dataInterface step1: " + e.getMessage ());
  }

  try {
    ActionForEach actionForEach;
    Selector sel =
      new Selector (Class.forName ("Customer"), "step1", false);
    actionForEach =
      modelActions.createActionForEach$message (customersList, sel);
  }
  catch (Exception e) {
    System.err.println("Customer step1: " + e.getMessage ());
  }

  try {
    ActionForEach actionForEach;
    Selector sel =
      new Selector (Class.forName ("Seller"), "step1", false);
    actionForEach =
      modelActions.createActionForEach$message (sellersList, sel);
  }
  catch (Exception e) {
    System.err.println("Seller step1: " + e.getMessage ());
  }

  try {
    ActionForEach actionForEach;
    Selector sel =
      new Selector (Class.forName ("Seller"), "step2", false);
    actionForEach =
      modelActions.createActionForEach$message (sellersList, sel);
  }
  catch (Exception e) {
    System.err.println("Seller step2: " + e.getMessage ());
  }

  try {
    ActionForEach actionForEach;
    Selector sel =
      new Selector (Class.forName ("Seller"), "step3", false);
    actionForEach =
      modelActions2.createActionForEach$message (sellersList, sel);
  }
  catch (Exception e) {
    System.err.println("Seller step3: " + e.getMessage ());
  }

  try {
          modelActions2.createActionTo$message
          (forecaster, new Selector (forecaster.getClass (), "step1", false));
  } catch (Exception e) {
    System.err.println ("forecaster step1: " + e.getMessage ());
  }

  modelSchedule = new ScheduleImpl (getZone (), 2);
  modelSchedule.at$createAction (0, modelActions);
  modelSchedule.at$createAction (1, modelActions2);
  return this;
}


public Activity activateIn (Swarm swarmContext) {
```

```
    super.activateIn (swarmContext);
    modelSchedule.activateIn (this);
    return getActivity ();
  }
}
```

Customer.java

```
import swarm.Globals;

import swarm.collections.List;
import java.util.ArrayList;
import java.io.*;

public class Customer {

  private DataInterface dataInterface;
  private long customerID;
          public ArrayList customerOrders;
          private Environment environment;
          public Customer (Environment anEnv, long aCustomerID,
                                        DataInterface aDataInterface) {
          customerID = aCustomerID;
          environment = anEnv;
          dataInterface = aDataInterface;
          }

  public Object setOrders (ArrayList listOfOrders) {
    customerOrders = listOfOrders;
                    return this;
          }

   public ArrayList getOrders () { return customerOrders; }

  public Object step1() {
                    int i;
                    long sellerID;
          customerOrders = dataInterface.getCustomerOrders(customerID);

          if (customerOrders.size() > 0)
          {
                    sellerID = ((OrderType)customerOrders.get(0)).getSellerID();
                    environment.sendOrders$toSellerID(customerOrders, sellerID);
          }
          return this;
  }

  public double getNumberOfOrders () {
                    if (customerOrders == null)
                              return 0.0;
                    else
                              return (double)customerOrders.size();
          }

 public long getCustomerID () { return customerID;   }
}
```

DataInterface.java

```
import java.util.*;

public abstract class DataInterface {
          private String tableName;
          private int year;
          protected Environment environment;
          protected ModelInterface theInterface;
          protected ArrayList ordersOnLine;

           private int currentWeek, maxWeek;

  static public final int customer_table = 1;
  static public final int seller_table = 2;
  static public final int color_table = 3;
  static public final int line_table = 4;


  public DataInterface (Environment anEnv, String tableSource,
                                  int stopWeek, int anYear){
    tableName = tableSource;
    maxWeek = stopWeek;
    currentWeek = 0;
    year = anYear;
```

```
    environment = anEnv;
  }

  public abstract Object step1 ();
  public abstract boolean openConnection ();
  public abstract Object closeConnection ();
  protected abstract Object retrieveOrders(int curDate);
  public abstract String[] getTableName$Entries(int table_type, int entries);

  public ArrayList getCustomerOrders(long aCustomerID) {
    int i;
    long customerID = aCustomerID;
    ArrayList ordersForCustomer = new ArrayList();
    for (i = 0; i < ordersOnLine.size(); i++)
      if (((OrderType)ordersOnLine.get(i)).getCustomerID() == customerID)
        ordersForCustomer.add(ordersOnLine.get(i));
    return ordersForCustomer;
  }

public abstract long getTableNumber(int table_type);

public Object setTheInterface(ModelInterface anInterface) {
                theInterface = anInterface;
                return this;
        }

}
```

DBInterface.java

```
import java.util.*;
import java.sql.*;

public class DBInterface extends DataInterface {
  private String DSN, url;
  private Connection connectionODBC;
  private String tableName;
        private int year;
        private int currentWeek, maxWeek;

public DBInterface (Environment anEnv, String DSN_name,
                                        String tableSource,  int stopWeek, int anYear){
        super(anEnv, tableSource, stopWeek, anYear);
        DSN = DSN_name;
                url = "jdbc:odbc:" + DSN_name;
                tableName = tableSource;
                currentWeek = 0;
                maxWeek = stopWeek;
        year = anYear;
}

public Object step1 () {
        currentWeek = environment.getCurrentTime();
  if (currentWeek <= maxWeek)
    retrieveOrders(currentWeek);
  return this;
}

public boolean openConnection () {
                try {
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                } catch(java.lang.ClassNotFoundException e) {
                        System.err.print("ClassNotFoundException: ");
                        System.err.println(e.getMessage());
                        return false;
                }

                try {
                        connectionODBC = DriverManager.getConnection(
                                                url, "sa", "");
                } catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                        return false;
                }
  return true;
}

public Object closeConnection () {
        try {
                        connectionODBC.close();
  } catch(SQLException ex) {
                System.err.println("SQLException: " + ex.getMessage());
          }
```

```
  return this;
 }

 protected Object retrieveOrders(int curDate) {
  ResultSet rs;
                Statement stmt;
  OrderType anOrder;
                int cnt;

  long aCustomerID, aQuantity, aTotalPrice;
  long aWeekNumber, aSellerID;
  String aColorID, aLineID;

String query = "Select * from " + tableName + year + " WHERE " +
            "Settimana = " + currentWeek;

/**/System.out.println("---");
/**/System.out.println("Reading data of week # " + curDate);

        try {
                stmt = connectionODBC.createStatement();
                rs = stmt.executeQuery(query);

                 ordersOnLine = new ArrayList();
                        cnt=0;
                while (rs.next()) {
                        aSellerID = rs.getLong("CodiceVenditore");
            aCustomerID = rs.getLong("CodiceCliente");
            aWeekNumber = rs.getLong("Settimana");
                aColorID = rs.getString("CodiceColore");
                aLineID = rs.getString("CodiceLinea");
                aQuantity = rs.getLong("TotaleCapi");
                aTotalPrice = rs.getLong("TotaleLire");
                anOrder = new
                OrderType(theInterface.getCustomer(aCustomerID),
                                theInterface.getSeller(aSellerID), aWeekNumber,
                                        theInterface.getColor(aColorID),
                                        theInterface.getLine(aLineID),
                                        aQuantity, aTotalPrice);
            ordersOnLine.add(anOrder);
                                cnt++;
                        }
        /**/System.out.println("Records read #: " + cnt);
                  rs.close();
                  stmt.close();
                } catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                }

  return this;
}

public long getTableNumber(int table_type) {
                ResultSet rs;
                Statement stmt;
                String tName = "";
                switch (table_type)
                {
                        case DBInterface.customer_table:
                                tName = tableName + "_Clienti";
                                break;
                        case DBInterface.seller_table:
                                tName = tableName + "_Venditori";
                                break;
                        case DBInterface.color_table:
                                tName = tableName + "_Colori";
                                break;
                        case DBInterface.line_table:
                                tName = tableName + "_Linee";
                                break;
                }

                String query = "select COUNT(*) as Numero from " + tName;

                try {
                        stmt = connectionODBC.createStatement();
                        rs = stmt.executeQuery(query);
                } catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                         return -1;
                }

                try {
```

```
        if (rs.next())
                return rs.getLong("Numero");
        else
                return -1;
        } catch(SQLException ex) {
                System.err.println("SQLException: " + ex.getMessage());
        }

        return -1;
}

public String[] getTableName$Entries(int table_type, int entries) {
        ResultSet rs;
        Statement stmt;
        String tName = "";
        String[] stringList = new String[entries];

        switch (table_type)
        {
                case DBInterface.customer_table:
                        tName = tableName + "_Clienti";
                        break;
                case DBInterface.seller_table:
                        tName = tableName + "_Venditori";
                        break;
                case DBInterface.color_table:
                        tName = tableName + "_Colori";
                        break;
                case DBInterface.line_table:
                        tName = tableName + "_Linee";
                        break;
        }

        String query = "select Codice from " + tName;

        int i;
        try {
                stmt = connectionODBC.createStatement();
                 rs = stmt.executeQuery(query);
        } catch(SQLException ex) {
                System.err.println("SQLException: " + ex.getMessage());
                System.exit(0);
                return null;
        }


        try {
                for (i=0; i < entries; i++)
                        if (rs.next())
                                stringList[i] = rs.getString("Codice");
        } catch(SQLException ex) {
                System.err.println("SQLException: " + ex.getMessage());
        }
        return stringList;
        }

}
```

FileInterface.java

```
import java.util.*;
import java.io.*;

public class FileInterface extends DataInterface {
        private String tableName;
        private int year;
        private BufferedReader inFile;
        private OrderType lastOrder;
        private final String dataDelimitator = "\t";
        private int currentWeek, maxWeek;

 public FileInterface (Environment anEnv, String tableSource,
                                        int stopWeek, int anYear) {

        super(anEnv, tableSource, stopWeek, anYear);
        tableName = tableSource;
        currentWeek = 0;
  maxWeek = stopWeek;
  year = anYear;
 }

 public Object step1 () {
        currentWeek = environment.getCurrentTime();
```

```
  if (currentWeek <= maxWeek)
    retrieveOrders(currentWeek);
  return this;
 }

 public boolean openConnection () {
                try {
        inFile = new BufferedReader(new FileReader(tableName + year + ".data"));
                } catch(IOException ex) {
                        System.err.println("Impossible to open " + tableName +
                                        ".data due to: " + ex.getMessage());
                return false;
                }
  return true;
 }

 public Object closeConnection () {
                inFile = null;
        return this;
 }

 protected Object retrieveOrders(int curDate) {
                int cnt;
                String x;
   long aCustomerID, aQuantity, aTotalPrice;
   long aWeekNumber, aSellerID;
   String aColorID, aLineID;

/**/System.out.println("---");
/**/System.out.println("Reading data of week # " + curDate);

try {
        ordersOnLine = new ArrayList();
        cnt=0;
        if (lastOrder == null)
        {
                x = inFile.readLine();
/**/System.out.println(x);
                StringTokenizer t = new StringTokenizer(x, dataDelimitator);
                  aSellerID = Long.parseLong(t.nextToken());
                     aCustomerID = Long.parseLong(t.nextToken());
                aWeekNumber = Long.parseLong(t.nextToken());
                                aColorID = t.nextToken();
                                aLineID = t.nextToken();
                                aQuantity = Long.parseLong(t.nextToken());
                                aTotalPrice = Long.parseLong(t.nextToken());
        if (theInterface == null)
        {
                System.err.println("The interface cannot be null!");
                System.exit(0);
        }

        lastOrder = new
        OrderType(theInterface.getCustomer(aCustomerID),
                theInterface.getSeller(aSellerID),
                aWeekNumber, theInterface.getColor(aColorID),
                theInterface.getLine(aLineID), aQuantity, aTotalPrice);
        }
while (lastOrder.getWeekNumber() == curDate)
        {
    ordersOnLine.add(lastOrder);
        cnt++;
        x = inFile.readLine();
        StringTokenizer t = new StringTokenizer(x, dataDelimitator);
        aSellerID = Long.parseLong(t.nextToken());
        aCustomerID = Long.parseLong(t.nextToken());
        aWeekNumber = Long.parseLong(t.nextToken());
        aColorID = t.nextToken();
        aLineID = t.nextToken();
        aQuantity = Long.parseLong(t.nextToken());
        aTotalPrice = Long.parseLong(t.nextToken());

        lastOrder = new
                OrderType(theInterface.getCustomer(aCustomerID),
                        theInterface.getSeller(aSellerID), aWeekNumber,
                theInterface.getColor(aColorID),
                        theInterface.getLine(aLineID),
                        aQuantity, aTotalPrice);
                }
/**/System.out.println("Records read #: " + cnt);
        } catch(IOException ex) {
                System.err.println("IOException: " + ex.getMessage());
        }
        catch (NoSuchElementException e) {
```

```
                        System.err.println("Data not correct. Please verify.");
                }
  return this;
}

        public long getTableNumber(int table_type) {
                String tName = "";
                BufferedReader tempInFile;
                switch (table_type)
                {
                        case DataInterface.customer_table:
                                tName = tableName + "_Clienti.data";
                                break;
                        case DataInterface.seller_table:
                                tName = tableName + "_Venditori.data";
                                break;
                        case DataInterface.color_table:
                                tName = tableName + "_Colori.data";
                                break;
                        case DataInterface.line_table:
                                tName = tableName + "_Linee.data";
                                break;
                }

                try {
                tempInFile = new BufferedReader(new FileReader(tName));
                } catch(IOException ex) {
                        System.err.println("Impossible to open " + tName +
                                        " due to: " + ex.getMessage());
                        return -1;
                }

                try {
                        return Long.parseLong(tempInFile.readLine());
                } catch(IOException ex) {
                        System.err.println("IOException: " + ex.getMessage());
                }
                return -1;
        }

        public String[] getTableName$Entries(int table_type, int entries) {
                String tName = "";
                BufferedReader tempInFile;
                String[] stringList = new String[entries];

                switch (table_type)
                {
                        case DataInterface.customer_table:
                                tName = tableName + "_Clienti.data";
                                break;
                        case DataInterface.seller_table:
                                tName = tableName + "_Venditori.data";
                                break;
                        case DataInterface.color_table:
                                tName = tableName + "_Colori.data";
                                break;
                        case DataInterface.line_table:
                                tName = tableName + "_Linee.data";
                                break;
                }
                int i;
                try {
                tempInFile = new BufferedReader(new FileReader(tName));
                } catch(IOException ex) {
                        System.err.println("Impossible to open " + tName +
                                " due to: " + ex.getMessage());
                        return null;
                }

                try {
                        String x = tempInFile.readLine();
                        for (i=0; i < entries; i++)
                                stringList[i] = tempInFile.readLine();
                        } catch(IOException ex) {
                        System.err.println("IOException: " + ex.getMessage());
                }
                return stringList;
        }

}
```

Forecaster.java

```
import swarm.Globals;
import swarm.collections.List;

import java.util.ArrayList;
import java.io.*;

public class Forecaster {
        private final String _outputFileName = "forecaster.output";
        private double currentError = 0.0;
        private double currentSum = 0.0;
        private double currentForecast = 0.0;
        private int totalCustomers;
        private int frequencyInDataOutput, currentOutputCounter;
        private PrintWriter outFile;
        private ForecasterDataWarehouse myDataWarehouse;
        private Environment environment;
        private ModelInterface theInterface;

        public Forecaster (Environment anEnv, ForecasterDataWarehouse aDw,
                                        ModelInterface itf, int freqInOutput) {
                myDataWarehouse = aDw;
                environment = anEnv;
                theInterface = itf;
                totalCustomers = 0;
                currentOutputCounter = 0;
                frequencyInDataOutput = freqInOutput;
                if (frequencyInDataOutput == 0)
                        outFile = null;
                else
                        try {
                                outFile= new PrintWriter(new
                                        FileWriter(_outputFileName));
                        } catch (Exception e) {
                                System.err.println("Error in opening file " +
                                _outputFileName + ": " + e.getMessage());
                        }

                int i, nLines = myDataWarehouse.getTotalLines();
                outFile.println("File structure description");
                outFile.print("First line: #week #customers [lines:");
                for (i = 0; i < nLines; i++)
                        outFile.print(" " + theInterface.getLineKey(i));
                outFile.println("]");
                outFile.println("Second line: #week #customers [forecasts]");
        }

public Object step1 () {
                int i,j,k;
                double cnt;

                double totalForecast = 0.0;
                double totalSum = 0.0;
                totalCustomers = 0;

                if (environment.isTheForecastTime() && outFile != null)
                {
                        outFile.close();
                        return this;
                }

                if (environment.isNoMoreTheForecastTime())
                        return this;

                int nLines = myDataWarehouse.getTotalLines();
                int nWeeks = myDataWarehouse.getTotalWeeks();
                int nWeek = environment.getCurrentTime();

                long[] totalSales = myDataWarehouse.getSells();
                double[] totalForecasts = myDataWarehouse.getForecasts();
                List sellerList = environment.getSellerList();

                Seller aSeller;
                SellerDataWarehouse aSDW;
                long[][] totalSellerSales;
                double[][] forecastTotalSellerSales;

                for (j = 0; j < nLines; j++)
                {
                        totalSales[j] = 0;
                        totalForecasts[j] = 0.0;
                }
```

```
        for (i=0; i < sellerList.getCount(); i++)
        {
                aSeller = (Seller)sellerList.atOffset(i);
                aSDW = aSeller.getMyDataWarehouse();
                totalSellerSales = aSDW.getTotalSales();
                forecastTotalSellerSales =
                        aSDW.getTotalSalesForecasts();

                for (j = 0; j < nLines; j++)
                {
                        cnt = 0.0;
                        for (k = 0; k < nWeek; k++)
                                cnt += totalSellerSales[j][k];
                        totalSales[j] += cnt;
                        totalForecasts[j] +=
                                forecastTotalSellerSales[j][nWeek];
                }
                totalCustomers += aSDW.getCustomersBargained();
        }

        for (j = 0; j < nLines; j++)
        {
                totalSum += (double) totalSales[j];
                totalForecast += (double) totalForecasts[j];
        }

        currentForecast = (float) totalForecast;
        currentSum = (float) totalSum;
        currentError = (float) totalForecast - totalSum;
        environment.setCustomersBargained(totalCustomers);

        if (outFile != null)
                if (++currentOutputCounter == frequencyInDataOutput)
                {
                        currentOutputCounter = 0;
                        writeCurrentDataToFile();
                }

        return this;
}

private void writeCurrentDataToFile() {
        int j;
        int nLines = myDataWarehouse.getTotalLines();

        long[] totalSales = myDataWarehouse.getSells();
        double[] totalForecasts = myDataWarehouse.getForecasts();

        // total sales line
        outFile.print(environment.getCurrentTime() + " ");
        outFile.print(totalCustomers + " ");

        for (j = 0; j < nLines; j++)
                outFile.print(totalSales[j] + " ");
        outFile.println();

        // total forecasts line
        outFile.print(environment.getCurrentTime() + " ");
        outFile.print(totalCustomers + " ");

        for (j = 0; j < nLines; j++)
                outFile.print(((long) totalForecasts[j]) + " ");
        outFile.println();

        return;
    }

    public float getCurrentError () { return (float) currentError; }

    public float getCurrentSum () { return (float) currentSum; }
    public float getCurrentForecast () {  return (float) currentForecast; }
}
```

ForecasterDataWarehouse.java

```
import swarm.Globals;
import java.io.*;
import java.util.*;

public class ForecasterDataWarehouse {

        private int nLines;
        private int lastWeek;
```

```
        private int myForecaster;

        // OLAP containers
        private long[] totalSells;
        private double[] totalForecasts;

        // customers counter
        private int customersNumber;

/** Constructor
        */
        public ForecasterDataWarehouse(int lines,
                                        int forecasterNumber, int aLastWeek) {
                nLines = lines;
                lastWeek = aLastWeek;
                int i,j;

                customersNumber = 0;
                myForecaster = forecasterNumber;
                totalSells = new long[nLines];
                totalForecasts = new double[nLines];
        }

        public int getCustomers() { return customersNumber; }

        public Object setCustomers(int custNumber) {
                customersNumber = custNumber;
                return this;
        }

public double[] getForecasts () { return totalForecasts; }
public long[] getSells () {   return totalSells;  }
public int getTotalLines () { return nLines; }
public int getTotalWeeks () { return lastWeek;}
}
```

ModelInterface.java

```
import swarm.Globals;

import java.io.*;

import java.util.*;

public class ModelInterface {

        private Hashtable customers, sellers;
        private Hashtable colors, lines;
        private Hashtable backCustomers, backSellers;
        private Hashtable backColors, backLines;
        private DataInterface dataInterface;

        public ModelInterface (DataInterface aDataInterface, int customerNumber,
                        int sellerNumber, int colorNumber, int lineNumber) {

                dataInterface = aDataInterface;
                customers = new Hashtable(); backCustomers = new Hashtable();
                sellers = new Hashtable(); backSellers = new Hashtable();
                colors = new Hashtable(); backColors = new Hashtable();
                lines = new Hashtable(); backLines = new Hashtable();

                getElements(DataInterface.customer_table, customerNumber,
                                        customers, backCustomers);
                getElements(DataInterface.seller_table, sellerNumber,
                                        sellers, backSellers);
                getElements(DataInterface.color_table, colorNumber,
                                        colors, backColors);
                getElements(DataInterface.line_table, lineNumber,
                                        lines, backLines);
        }

        private void getElements (int table_type, long elemNumber,
                        Hashtable  theHash,  Hashtable theBackHash) {
                String[] aList;
                int i;

                aList = dataInterface.getTableName$Entries(
                                table_type, (int)elemNumber);

                for (i=0; i < elemNumber; i++)
                {
                        theHash.put(new String(aList[i]), new Integer(i));
                        theBackHash.put(new Integer(i), new String(aList[i]));
                }
```

```
        return ;
}

public int getCustomer (long elemID) {
        return ((Integer)customers.get(new
                        String("" + elemID))).intValue();
}

public int getSeller (long elemID) {
 return ((Integer)sellers.get(new String("" + elemID))).intValue();
}

public int getColor (String elemID) {
        return ((Integer)colors.get(new String(elemID))).intValue();
}

public int getLine (String elemID) {
        return ((Integer)lines.get(new String(elemID))).intValue();

}

public String getCustomerKey (int elemNumber) {
        return (String)backCustomers.get(new Integer(elemNumber));
}

public String getSellerKey (int elemNumber) {
        return (String)backSellers.get(new Integer(elemNumber));
}

public String getColorKey (int elemNumber) {
        return (String)backColors.get(new Integer(elemNumber));
}

public String getLineKey (int elemNumber) {
        return (String)backLines.get(new Integer(elemNumber));
}

public Object[] getSellerKeys() {
        Object[] s;
        s = sellers.keySet().toArray();
        return s;
}
```

```
}
```

OrderType.java

```
import swarm.Globals;
import java.io.*;

public class OrderType {

        private long customerID;
        private long sellerID;
        private long weekNumber;
        private long quantity, totalPrice;
        private long colorID, lineID;
        public OrderType (long aCustomerID, long aSellerID, long aWeekNumber,
                long aColorID, long aLineID,long aQuantity, long aTotalPrice) {

                customerID = aCustomerID;
                sellerID = aSellerID;
                weekNumber = aWeekNumber;
                colorID = aColorID;
                lineID = aLineID;
                quantity = aQuantity;
                totalPrice = aTotalPrice;
        }

        public long getCustomerID () { return customerID; }
        public Object setCustomerID (long aCustomerID) {
                customerID = aCustomerID;
                return this;
        }

        public long getSellerID () { return sellerID; }
        public Object setSellerID (long aSellerID) {
                sellerID = aSellerID;
                return this;
        }

        public long getWeekNumber () { return weekNumber; }
        public Object setWeekNumber (long aWeekNumber) {
```

```
                weekNumber = aWeekNumber;
                return this;
        }

        public long getColorID () { return colorID; }
        public Object setCodColor (long aColorID) {
                colorID = aColorID;
          return this;
        }
        public long getLineID () { return lineID; }
        public Object setLineID (long aLineID) {
                lineID = aLineID;
          return this;
        }

        public long getQuantity () { return quantity; }
        public Object setQuantity (long aQuantity) {
                quantity = aQuantity;
          return this;
        }
        public long getTotalPrice () { return totalPrice; }
        public Object setTotalPrice (long aTotalPrice) {
                totalPrice = aTotalPrice;
          return this;
        }
}
```

Seller.java

```
import swarm.Globals;
import swarm.gui.Raster;

import java.util.ArrayList;
import java.io.*;

public class Seller {

        private long sellerID;
        private SellerDataWarehouse dataWarehouse;
        private SellerRuleMaster ruleMaster;
        private int pointTime;
        private Environment environment;

public Seller (Environment anEnv, long aSellerID,
          SellerDataWarehouse aDW,  SellerRuleMaster aRM) {

                sellerID = aSellerID;
                dataWarehouse = aDW;
                ruleMaster = aRM;
                pointTime = 0;
                environment = anEnv;
                aDW.loadFromFileForecastOfSales$OfCustomers(true,true);
        }

        private Object setCurrentPointTime (int aPointTime) {
                pointTime = aPointTime;
                return this;
        }

public Object receiveOrdersFromACustomer(ArrayList orders) {
        int i;
        OrderType order;

        for (i=0; i < orders.size(); i++)
        {
                order = (OrderType) orders.get(i);
                dataWarehouse.addQuantityToSalesLine$Quantity$Week(
                        (int) order.getLineID(), order.getQuantity(),pointTime);
                dataWarehouse.addCustomerToCounter(order.getCustomerID());
                }
        return this;
  }

  public Object step1 () {
        pointTime = environment.getCurrentTime();
        dataWarehouse.stepEndWeek(pointTime);
        return this;
  }

  public Object step2 () {

        ruleMaster.applyRulesToValuesInDataWarehouseStep1(dataWarehouse,
                                                pointTime);
```

```
        return this;
 }

 public Object step3 () {

        ruleMaster.applyRulesToValuesInDataWarehouseStep2(dataWarehouse,
                                                pointTime);
        return this;
 }

public float getTotalForecast () {
                double total = 0.0;
                double[][] forecasts = dataWarehouse.getTotalSalesForecasts();

                int i, k;

                if (pointTime < dataWarehouse.getTotalWeeks())
                        k = pointTime;
                else
                        k= dataWarehouse.getTotalWeeks() - 1;

                for (i=0; i < dataWarehouse.getTotalLines(); i++)

                                total += forecasts[i][k];

                return (float) total;
        }

        public long getTotalSum () {
                long total = 0;
                long[][] sumOfSales = dataWarehouse.getTotalSales();
                int i, j, k;

                if (pointTime < dataWarehouse.getTotalWeeks())
                        k = pointTime;
                else
                        k= dataWarehouse.getTotalWeeks() - 1;

                for (i=0; i < dataWarehouse.getTotalLines(); i++)
                        for (j=0; j <= k; j++)
                                total += sumOfSales[i][j];
                return total;
        }

        public float getTotalError () {
                return getTotalForecast() - getTotalSum();
        }

        public int getHowManyCustomers () {
                return dataWarehouse.getCustomersBargained();
        }

        public SellerDataWarehouse getMyDataWarehouse () {
                return dataWarehouse;
        }

        public long getSellerID () { return sellerID;   }

}
```

SellerDataWarehouse.java

```
import swarm.Globals;
import java.io.*;
import java.util.*;

public class SellerDataWarehouse {

        private final String _salesForecastPrexif = "Sale_";
        private final String _customersForecastPrexif = "Cust_";

        private int nLines;
        private int nWeeks;

        private int mySellerNumber;

        // data containers
        private long[][] totalSales;
        private double[][] forecastTotalSales;
        private long[] customersPerWeek;
        private double[] forecastCustomersPerWeek;
```

```
        // customers counter
        private Hashtable howManyCustomers;
        private int customersBargained;

public SellerDataWarehouse(int lines,   int sellerNumber, int aLastWeek) {
                nLines = lines;
                nWeeks = aLastWeek;
                int i,j;
                customersBargained = 0;
                howManyCustomers = new Hashtable();

                mySellerNumber = sellerNumber;
                totalSales = new long[nLines][nWeeks];
                customersPerWeek = new long[nWeeks];
                forecastTotalSales = new double[nLines][nWeeks];
                forecastCustomersPerWeek = new double[nWeeks];
        }

        public Object loadFromFileForecastOfSales$OfCustomers(
                        boolean v1, boolean v2) {
                double[][] fTS;
                double[] fCPW;
                if (v1)
                {
                fTS =
                   MatrixSerialize.loadDoubleMatrixFromFileName(
                        _salesForecastPrexif + mySellerNumber + ".mtrx");
                        if (fTS != null)
                                forecastTotalSales = fTS;
                }

                if (v2)
                {
                fCPW =
                  MatrixSerialize.loadDoubleVectorFromFileName(
                        _customersForecastPrexif + mySellerNumber + ".mtrx");

                        if (fCPW != null)
                                forecastCustomersPerWeek = fCPW;
                }

                return this;
        }

        public Object saveToFileDataOfSales$OfCustomers(
                                                boolean v1, boolean v2) {

                if (v1)
                {
                        MatrixSerialize.saveMatrix$withX$andY$intoFileName(
                          totalSales, nLines, nWeeks,
                        _salesForecastPrexif + mySellerNumber + ".mtrx");
                }

        if (v2)
        {
                MatrixSerialize.saveVector$withDimension$intoFileName(
                        customersPerWeek, nWeeks,
                        _customersForecastPrexif + mySellerNumber + ".mtrx");
                }

                return this;
        }

public Object stepEndWeek(int numberOfWeek) {
                int i;
                long k = 0;

                if (numberOfWeek < nWeeks)
                {
                        for (i=0; i < numberOfWeek; i++)
                                k += customersPerWeek[i];

                customersPerWeek[numberOfWeek] = customersBargained - k;

                }

                return this;
        }

public Object addQuantityToSalesLine$Quantity$Week(int line, long quantity,
                                                        int numberOfWeek) {
                if (line >= nLines)
                {
```

```
                        System.err.println("addQuantityToSell: index out
                                                of bounds");
                        System.exit(0);
                }

                if (numberOfWeek < nWeeks)
                        totalSales[line][numberOfWeek] += quantity;

                return this;
        }

public Object addCustomerToCounter(long customer) {

        if (howManyCustomers.get(new String("" + customer)) == null)
        {
        howManyCustomers.put(new String("" + customer), new Integer(0));
        ++customersBargained;
        }
                return this;
        }

        public int getCustomersBargained() {return customersBargained;}

        public int getCustomersBargainedInWeek(int numberOfWeek) {
                return (int) customersPerWeek[numberOfWeek];
        }

        public double[][] getTotalSalesForecasts () {return forecastTotalSales; }
        public double[] getCustomersPerWeekForecast () {
                return forecastCustomersPerWeek;
        }
        public long[] getCustomersPerWeek () {return customersPerWeek;}
        public long[][] getTotalSales () {     return totalSales; }
        public int getTotalLines () { return nLines; }
        public int getTotalWeeks () { return nWeeks; }
        public int getMySellerNumber () { return mySellerNumber; }
}
```

SellerRuleMaster.java

```
import swarm.Globals;
import java.io.*;
import java.util.*;
import java.lang.Math;

public class SellerRuleMaster {

        private final double _customerProportional = 0.8;
        private int stopForecast;
        private boolean onlyVerificationRun;

        public SellerRuleMaster (int stopForecastTime,
                                 boolean dontSaveAtTheEnd) {
                stopForecast = stopForecastTime;
                onlyVerificationRun = dontSaveAtTheEnd;
        }

public Object applyRulesToValuesInDataWarehouseStep1(
                SellerDataWarehouse agentDw, int pointTime) {

                long[][] sumOfSales;
                double[][] forecasts;
                double[] forecastCustomersPerWeek;
                int nLines;
                sumOfSales = agentDw.getTotalSales();
                forecasts = agentDw.getTotalSalesForecasts();
                forecastCustomersPerWeek =
                        agentDw.getCustomersPerWeekForecast();
                nLines = agentDw.getTotalLines();

                double foreseePastCustomers = 0.0;
                double foreseeCustomers = 0.0;
                double customerCorrection;
                double result = 0.0;
                int i,j;

                if (pointTime < stopForecast)
                {
                for (j=0; j <= pointTime; j++)
                        foreseePastCustomers += forecastCustomersPerWeek[j];

                        if (foreseePastCustomers != 0)
                          customerCorrection =
```

```
                                        agentDw.getCustomersBargained()  /
                                                foreseePastCustomers;
                        else
                                customerCorrection =
                                        agentDw.getCustomersBargained();

                for (j=pointTime + 1; j < stopForecast; j++)
                        foreseeCustomers += forecastCustomersPerWeek[j];
                        if (foreseeCustomers == 0) foreseeCustomers = 1;

                        foreseeCustomers *= customerCorrection;

                        for (i=0; i < nLines; i++)
                        {
                                forecasts[i][pointTime] = 0.0;
                                for (j=0; j <= pointTime; j++)
                                forecasts[i][pointTime] += sumOfSales[i][j];

                                if (agentDw.getCustomersBargained()          != 0)
                                        result = (forecasts[i][pointTime] /

                agentDw.getCustomersBargained()) *              foreseeCustomers;

                                forecasts[i][pointTime] += result;
                        }
                }
                return this;
        }

public Object applyRulesToValuesInDataWarehouseStep2(
                        SellerDataWarehouse agentDw, int pointTime) {

                if (pointTime == stopForecast && !onlyVerificationRun)
                        agentDw.saveToFileDataOfSales$OfCustomers(true, true);
                return this;
        }
}
```

FORECAST MODEL

Agent.java

```
import swarm.Globals;
import java.io.*;

public class Agent {

    private RuleMaster ruleMaster;
    private DataWarehouse myDataWarehouse;
    private Interface myInterface;
    private int number, inputNodeNumber;
    private int hiddenNodeNumber, outputNodeNumber;
    private int patternNumberInVerificationSet;
    private boolean agentsAreDisplayingData, readWeightsFromFile;
    private float backPropagationErrorInVerificationSet;
    private float proportionalErrorInVerificationSet;
    private float backPropagationErrorInTrainingSet;
    private float proportionalErrorInTrainingSet;
    private Matrix outputVerificationMatrix;
    private Matrix targetVerificationMatrix, minmax;

    private SellerDataWarehouse sellerDataWarehouse;

    public Agent (int n, boolean rw,
                  RuleMaster rm, DataWarehouse dw,
                  Interface inf, boolean d,
                  SellerDataWarehouse sDW) {

        number = n;
        readWeightsFromFile = rw;
        ruleMaster = rm;
        myDataWarehouse = dw;
        myInterface = inf;
        agentsAreDisplayingData = d;

        sellerDataWarehouse = sDW;

        inputNodeNumber= myDataWarehouse.getInputNodeNumber();
        hiddenNodeNumber=myDataWarehouse.getHiddenNodeNumber();
        outputNodeNumber=myDataWarehouse.getOutputNodeNumber();
        patternNumberInVerificationSet=
        myDataWarehouse.getPatternNumberInVerificationSet();

        outputVerificationMatrix=
                myDataWarehouse.getOutputVerificationMatrix();
        targetVerificationMatrix=
                myDataWarehouse.getTargetVerificationMatrix();
        minmax= myDataWarehouse.getMinmax();

        if (readWeightsFromFile)
            myDataWarehouse.readWeightsFromFileOfAgentNumber(number);
    }


    public int getNumber() { return number; }
    public DataWarehouse getDataWarehouse() { return myDataWarehouse; }
    public Interface getInterface() { return myInterface; }

    public Object step1() {
        myInterface.doActionsForAgentStep1();
        if(patternNumberInVerificationSet<0)
        {
        myInterface.setCT_Inputs();
           ruleMaster.applyRulesToInputOutputValuesInDataWarehouseStepA(
                myDataWarehouse, sellerDataWarehouse);
      }
      return this;
    }

    public Object step2() {
        int i, j, pN;

        myInterface.doActionsForAgentStep2();
        if(patternNumberInVerificationSet<0)
            myInterface.setCT_Targets();
        ruleMaster.applyRulesToInputOutputValuesInDataWarehouseStepA(
            myDataWarehouse, sellerDataWarehouse);
        backPropagationErrorInVerificationSet=
                ruleMaster.getBackPropagationErrorInVerificationSet();
        proportionalErrorInVerificationSet   =
                ruleMaster.getProportionalErrorInVerificationSet();
```

```
    if(agentsAreDisplayingData)
    {

        System.out.println("Agent # " + number);
        pN=patternNumberInVerificationSet;
        if (patternNumberInVerificationSet < 0) pN*=-1; // if CT

        for (i = 0; i < pN; i++)
        for (j = 0; j < outputNodeNumber; j++)
          System.out.println("pattern # " + (i+1) + ";" + (j+1) +
                          " output " +
outputVerificationMatrix.R$C$asExternalValueUsing$withShift(
                i, j, minmax, inputNodeNumber) +
                          " target " +
targetVerificationMatrix.R$C$asExternalValueUsing$withShift(
                i, j, minmax, inputNodeNumber));

        System.out.println("Conventional back propagation error
                    in verification set " +
                                    backPropagationErrorInVerificationSet);
        System.out.println("Proportional error in verification set  "  +
                                    proportionalErrorInVerificationSet);
    }
    return this;
}

public Object step3() {

    myInterface.doActionsForAgentStep3();
    ruleMaster.applyRulesToInputOutputValuesInDataWarehouseStepB(
                myDataWarehouse, sellerDataWarehouse);

    backPropagationErrorInTrainingSet=
        ruleMaster.getBackPropagationErrorInTrainingSet();
    proportionalErrorInTrainingSet   =
        ruleMaster.getProportionalErrorInTrainingSet();
return this;
}

public float getBackPropagationErrorInVerificationSet() {
    return backPropagationErrorInVerificationSet;
}

public float getProportionalErrorInVerificationSet() {
    return proportionalErrorInVerificationSet;
}

public float getBackPropagationErrorInTrainingSet() {
    return backPropagationErrorInTrainingSet;
}

public float getProportionalErrorInTrainingSet() {
    return proportionalErrorInTrainingSet;
}

public Object save() // to save weights
{
    int i, j, pN;
    PrintWriter tempOutFile = null;
    String name;

    //wih
    name = new String("agent" + number + ".wih");

    try {
        tempOutFile= new PrintWriter(new FileWriter(name));
    } catch (Exception e) {
        System.err.println("Error in opening file " + name +
                ": " + e.getMessage());
    }

    for (i = 0; i < hiddenNodeNumber; i++)
        try {
        for (j = 0; j < inputNodeNumber + 1; j++)
            {
                Matrix k = myDataWarehouse.getWih();
                tempOutFile.print("" + k.R$C(i,j) + " ");
            }
    tempOutFile.println();
    } catch (Exception e) {
        System.err.println("Error writing file " + name +
                ": " + e.getMessage());
    }
```

```
tempOutFile.close();
tempOutFile = null;

//who
name = new String("agent" + number + ".who");

try {
tempOutFile= new PrintWriter(new FileWriter(name));
} catch (Exception e) {
System.err.println("Error in opening file " + name + ": " + e.getMessage());
}

for (i = 0; i < outputNodeNumber; i++)
    try {
    for (j = 0; j < hiddenNodeNumber + 1; j++)
    {
        Matrix k = myDataWarehouse.getWho();
            tempOutFile.print("" + k.R$C(i,j) + " ");
        }
tempOutFile.println();
} catch (Exception e) {
System.err.println("Error writing file " + name + ": " + e.getMessage());
}

tempOutFile.close();
tempOutFile = null;


//outputVerificationMatrix
name = new String("agent" + number + ".outputVerificationMatrix");

try {
    tempOutFile= new PrintWriter(new FileWriter(name));
} catch (Exception e) {
    System.err.println("Error in opening file " + name +
            ": " + e.getMessage());
}

pN = patternNumberInVerificationSet;
if (pN < 0) pN = -1 * pN;

        for (i = 0; i < pN; i++)
            try {
            for (j = 0; j < outputNodeNumber; j++)
                {
                Matrix k =
                    myDataWarehouse.getOutputVerificationMatrix();
                    tempOutFile.print("" +
                    k.R$C$asExternalValueUsing$withShift(
                            i,j,minmax,inputNodeNumber) + " ");
                }
            tempOutFile.println();
        } catch (Exception e) {
                System.err.println("Error writing file " + name + ": " +
e.getMessage());
          }

          tempOutFile.close();
          tempOutFile = null;

          return this;
    }



}
```

DataWarehouse.java

```
import swarm.Globals;
import java.io.*;
import java.util.*;

public class DataWarehouse {
    private int inputNodeNumber, inputNodeNumber1;
    private int hiddenNodeNumber, hiddenNodeNumber1, outputNodeNumber;
    private int patternNumberInVerificationSet, patternNumberInTrainingSet;
    private int epochNumberInEachTrainingCycles;
    private float weightRange, eps, alpha;
    private int usingRandomOrderInLearning;
    private int longTermLearningInCT_OnlyWithCompleteTrainingSet;
```

```
private int useOutputsAsTargetsInCT_RelearningScheme;

private Matrix outputTrainingMatrix, targetTrainingMatrix;
private Matrix dataTrainingMatrix;
private Matrix outputVerificationMatrix, targetVerificationMatrix;
private Matrix dataVerificationMatrix;
private Matrix minmax, newMinMax;
private Matrix wih, dwih, who, dwho;
private Matrix inputLayer, hiddenLayer, outputLayer, deltaOut;
private Matrix hiddenLayerTransFuncDerivatives;
private Matrix outputLayerTransFuncDerivatives;

private String trainingFileName, minmaxFileName;
private String verificationFileName, initValuesFileName;

private BufferedReader trainingSet, verificationSet;
private BufferedReader minmaxSet, initValues;

private boolean internalDataTraining;
private String creationSequence = new String("");
public DataWarehouse (boolean internalData) {
    internalDataTraining = internalData;
};

public Object setFileNames(String aVerificationFN, String aTrainingFN,
                           String aMinmaxFN, String aInitValuesFileName)
{
    trainingFileName=     aTrainingFN;
    minmaxFileName=       aMinmaxFN;
    verificationFileName= aVerificationFN;
    initValuesFileName=   aInitValuesFileName;

    creationSequence = creationSequence + "1";
    return this;
}

public Object setNeuralNetParameters(
                int anInputNodeNumber,int anHiddenNodeNumber,
            int anOutputNodeNumber,
            int aPatternNumberInVerificationSet,
            int aPatternNumberInTrainingSet,
            int           anEpochNumberInEachTrainingCycles) {
    inputNodeNumber   =anInputNodeNumber;
    inputNodeNumber1  =anInputNodeNumber + 1;
    hiddenNodeNumber  =anHiddenNodeNumber;
    hiddenNodeNumber1 =anHiddenNodeNumber + 1;
    outputNodeNumber  =anOutputNodeNumber;
    patternNumberInVerificationSet=aPatternNumberInVerificationSet;
    patternNumberInTrainingSet    =aPatternNumberInTrainingSet;
    epochNumberInEachTrainingCycles   =anEpochNumberInEachTrainingCycles;

    creationSequence = creationSequence + "2";
    return this;
}

public Object setBackPropagationParameters(
                float aWeightRange, float anEps,
                float anAlpha, int anOrderInLearning,
                int aLongTermLearningInCT,
            int aUseOutputsAsTargetsInCT) {
    weightRange = aWeightRange;
    eps         = anEps;
    alpha       = anAlpha;
    usingRandomOrderInLearning = anOrderInLearning;
    longTermLearningInCT_OnlyWithCompleteTrainingSet =
            aLongTermLearningInCT;
    useOutputsAsTargetsInCT_RelearningScheme = aUseOutputsAsTargetsInCT;

    creationSequence = creationSequence + "3";
    return this;
}

public Object createEnd () {

    int i, j, k;
    float V;
    String x="";
    StringTokenizer t= new StringTokenizer(x, " ");

    if (!creationSequence.equals("123"))
    {
        System.err.println("Datawarehouse creation sequence is wrong!");
        System.exit(0);
```

```
}
creationSequence = "OK";

k = inputNodeNumber + outputNodeNumber;
outputTrainingMatrix = new Matrix(patternNumberInTrainingSet ,
                            outputNodeNumber, 1);

targetTrainingMatrix = new Matrix(patternNumberInTrainingSet,
                                        outputNodeNumber,2);

dataTrainingMatrix = new Matrix(patternNumberInTrainingSet,
                            k,3);

outputVerificationMatrix= new Matrix(patternNumberInVerificationSet,
                                        outputNodeNumber,4);

targetVerificationMatrix= new Matrix(patternNumberInVerificationSet,
                                        outputNodeNumber,5);

dataVerificationMatrix= new Matrix(patternNumberInVerificationSet,
                                                k,6);


minmax= new Matrix(k,4,7);
newMinMax = new Matrix(k,4,7);

try {
    minmaxSet = new BufferedReader(new FileReader(minmaxFileName));
} catch (Exception e) {
    System.err.println ("Data file '" + minmaxFileName + "' does not exist.");
    System.err.println ("Exception: " + e.getMessage ());
            System.exit(0);
}

if (!internalDataTraining)
{
if (patternNumberInVerificationSet > 0)
    {
        try {
        trainingSet = new BufferedReader(new FileReader(trainingFileName));
        } catch (Exception e) {
            System.err.println ("Data file '" + trainingFileName +
                                        "' does not exist.");
            System.err.println ("Exception: " + e.getMessage ());
                                        System.exit(0);
        }

    try {
        verificationSet = new BufferedReader(new
                    FileReader(verificationFileName));
    } catch (Exception e) {
        System.err.println ("Data file '" + verificationFileName +
                                        "' does not exist.");
        System.err.println ("Exception: " + e.getMessage ());
                    System.exit(0);
    }
}

else
    try {
        initValues = new BufferedReader(new
                    FileReader(initValuesFileName));
    } catch (Exception e) {
        System.err.println ("Data file '" + initValuesFileName +
                    "' does not exist.");
        System.err.println ("Exception: " + e.getMessage ());
                    System.exit(0);
    }
}

x="";
t= new StringTokenizer(x, " ");
float f;

for (i = 0; i < k; i++)
    for (j = 0; j < 4; j++)
        try {
            if (!t.hasMoreTokens())
            {
                    x = minmaxSet.readLine();
                    t = new StringTokenizer(x, " ");
            }
            f =     Float.parseFloat(t.nextToken());
            minmax.R$C$setFrom(i,j, f);
```

```
            newMinMax.R$C$setFrom(i,j, f);

        } catch (Exception e) {
            System.err.println ("Lacking data in minmax set file." +
                                e.getMessage ());
            System.exit(0);
        }

try {
    minmaxSet.close();
} catch (Exception e) {
    System.err.println (e.getMessage ());
}
minmaxSet = null;

if (!internalDataTraining)
{
    if (patternNumberInVerificationSet > 0)
    {
        x="";
        t= new StringTokenizer(x, " ");

        for (i = 0; i < patternNumberInTrainingSet; i++)
            for (j = 0; j < k; j++)
                try {
            if (!t.hasMoreTokens())
            {
                    x = trainingSet.readLine();
                    t = new StringTokenizer(x, " ");
            }

            dataTrainingMatrix.R$C$setFrom$asExternalValueUsing$withShift(
                    i,j, Float.parseFloat(t.nextToken()), minmax, 0);

    } catch (Exception e) {
        System.err.println ("Lacking data in training set file." +
                            e.getMessage ());
        System.exit(0);
}

try {
    trainingSet.close();
} catch (Exception e) {
    System.err.println (e.getMessage ());
}
trainingSet = null;

for (i = 0; i < patternNumberInVerificationSet; i++)
    for (j = 0; j < k; j++)
        try {
            if (!t.hasMoreTokens())
            {
                    x = verificationSet.readLine();
                    t = new StringTokenizer(x, " ");
            }

        dataVerificationMatrix.R$C$setFrom$asExternalValueUsing$withShift(
                    i,j, Float.parseFloat(t.nextToken()), minmax, 0);

        } catch (Exception e) {
            System.err.println ("Lacking data in verification set file." +
                                e.getMessage ());
            System.exit(0);
        }

        try {
            verificationSet.close();
        } catch (Exception e) {
            System.err.println (e.getMessage ());
        }
        verificationSet = null;
    }
    else // CT case
    if (initValues != null)
    {
        for (j = 0; j < k; j++)
            for (i=0; i < -1 * patternNumberInTrainingSet; i++)
            try {
                    if (!t.hasMoreTokens())
                    {
                            x = initValues.readLine();
                            t = new StringTokenizer(x, " ");
                    }
```

```
            dataTrainingMatrix.R$C$setFrom$asExternalValueUsing$withShift(
                                    i,j, Float.parseFloat(t.nextToken()), minmax, 0);

                } catch (Exception e) {
                        System.err.println ("Lacking data in 'init.val' file." +
                                            e.getMessage ());
                        System.exit(0);
                }

            try {
                initValues.close();
            } catch (Exception e) {
                System.err.println (e.getMessage ());
            }
            initValues = null;
        }
    }

    wih= new Matrix(hiddenNodeNumber, inputNodeNumber1, 8);
    dwih= new Matrix(hiddenNodeNumber, inputNodeNumber1, 9);
    who= new Matrix(outputNodeNumber, hiddenNodeNumber1, 10);
    dwho= new Matrix(outputNodeNumber, hiddenNodeNumber1, 11);

    for (i = 0; i < hiddenNodeNumber; i++)
        for (j = 0; j < inputNodeNumber1; j++)
            wih.R$C$setFrom(i, j, (float)
                Globals.env.uniformDblRand.getDoubleWithMin$withMax(
                        -weightRange, weightRange));

    for (i = 0; i < outputNodeNumber; i++)
        for (j = 0; j < hiddenNodeNumber1; j++)
            who.R$C$setFrom(i, j, (float)
                Globals.env.uniformDblRand.getDoubleWithMin$withMax(
                        -weightRange, weightRange));

    inputLayer= new Matrix(inputNodeNumber1, 12);
    hiddenLayer= new Matrix(hiddenNodeNumber1, 13);
    outputLayer= new Matrix(outputNodeNumber, 14);
    deltaOut= new Matrix(outputNodeNumber, 15);
    return this;
}
```

```
public int getInputNodeNumber() {return inputNodeNumber;}
public int getHiddenNodeNumber() {return hiddenNodeNumber;}
public int getOutputNodeNumber() {return outputNodeNumber;}
public int getPatternNumberInVerificationSet() {return
                                patternNumberInVerificationSet;}
public int getPatternNumberInTrainingSet() {return
                               patternNumberInTrainingSet;}
public int getEpochNumberInEachTrainingCycles() {return
                               epochNumberInEachTrainingCycles;}

public Matrix getOutputVerificationMatrix() {return outputVerificationMatrix;}
public Matrix getTargetVerificationMatrix() {return targetVerificationMatrix;}
public Matrix getDataVerificationMatrix() {return dataVerificationMatrix;}
public Matrix getOutputTrainingMatrix() {return outputTrainingMatrix;}
public Matrix getTargetTrainingMatrix() {return targetTrainingMatrix;}
public Matrix getDataTrainingMatrix() {return dataTrainingMatrix;}
public Matrix getMinmax() {return minmax;}
public Matrix getNewMinMax() {return newMinMax;}
public String getMinMaxFileName() { return minmaxFileName; }

public Matrix getInputLayer() {return inputLayer;}
public Matrix getHiddenLayer() {return hiddenLayer;}
public Matrix getOutputLayer() {return outputLayer;}
public Matrix getDeltaOut() {return deltaOut;}

public Matrix getWih() {return wih;}
public Matrix getDwih() {return dwih;}
public Matrix getWho() {return who;}
public Matrix getDwho() {return dwho;}

public float getEps() {return eps;}
public float getAlpha() {return alpha;}
public int getRandomOrderInLearning() {return usingRandomOrderInLearning;}
public int getLongTermLearningInCT_OnlyWithCompleteTrainingSet() {return
        longTermLearningInCT_OnlyWithCompleteTrainingSet;}
public int getUseOutputsAsTargetsInCT_RelearningScheme() {return
        useOutputsAsTargetsInCT_RelearningScheme;}


public Object readWeightsFromFileOfAgentNumber(int n) {
    int i, j;
    float V;
```

```
    BufferedReader tempInFile = null;
    String name;
    String x="";
    StringTokenizer t= new StringTokenizer(x, " ");

    if (!creationSequence.equals("OK"))
    {
        System.err.println("You cannot use a datawarahouse
                                    without initalization");
        System.exit(0);
    }

    //wih
    name = new String("agent" + n + ".wih");

    try {
    tempInFile = new BufferedReader(new FileReader(name));
   } catch (Exception e) {
        System.err.println ("Data file '" + name + "' does not exist.");
        System.err.println ("Exception: " + e.getMessage ());
                System.exit(0);
   }

x="";
    t= new StringTokenizer(x, " ");

    for (i = 0; i < hiddenNodeNumber; i++)
        for (j = 0; j < inputNodeNumber + 1; j++)
        try {
            if (!t.hasMoreTokens())
            {
                x = tempInFile.readLine();
                t = new StringTokenizer(x, " ");
            }
            V = Float.parseFloat(t.nextToken());
            if (V != 0) wih.R$C$setFrom(i,j,V);
            else {
                System.out.println("Lacking data in agent" + n + ".wih file.");
                System.exit(0);
            }
        } catch (Exception e) {
            System.err.println ("Lacking data in agent" + n + ".wih file. "
                                        + e.getMessage ());
                System.exit(0);
        }

    try {
        tempInFile.close();
    } catch (Exception e) {
        System.err.println (e.getMessage ());
    }
   tempInFile = null;

    //who
    name = new String("agent" + n + ".who");

    try {
        tempInFile = new BufferedReader(new FileReader(name));
    } catch (Exception e) {
        System.err.println ("Data file '" + name + "' does not exist.");
        System.err.println ("Exception: " + e.getMessage ());
                System.exit(0);
    }

    for (i = 0; i < outputNodeNumber; i++)
        for (j = 0; j < hiddenNodeNumber + 1; j++)
        try {
            if (!t.hasMoreTokens())
            {
                x = tempInFile.readLine();
                t = new StringTokenizer(x, " ");
            }
            V = Float.parseFloat(t.nextToken());
            if (V != 0) who.R$C$setFrom(i,j,V);
            else {
                System.err.println("Lacking data in agent" + n + ".who file.");
                System.exit(0);
            }
        } catch (Exception e) {
            System.err.println ("Lacking data in agent" + n + ".who file. " +
e.getMessage ());
                System.exit(0);
        }
```

```
        try {
            tempInFile.close();
        } catch (Exception e) {
            System.err.println (e.getMessage ());
        }
        tempInFile = null;

        return this;
    }

}
```

ForecastModelSwarm.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.activity.ActionForEach;

import swarm.collections.List;
import swarm.collections.ListImpl;

import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;

import java.io.*;
import java.util.*;
import java.lang.*;
public class ForecastModelSwarm extends SwarmImpl
{
  // simulation parameters
public int agentNumber, inputNodeNumber, hiddenNodeNumber,
           outputNodeNumber;
public int patternNumberInVerificationSet, patternNumberInTrainingSet;
public int epochNumberInEachTrainingCycles, epochGroups;
public RuleMaster ruleMaster;
public RuleMaker ruleMaker;

public List agentList;
public Agent[] agentArray;

public boolean onlyVerificationRun;
public boolean internalDataGeneration;

public ActionGroup modelActions, modelActions2;
public Schedule modelSchedule;

public float weightRange, eps, alpha;
public int usingRandomOrderInLearning;
public int longTermLearningInCT_OnlyWithCompleteTrainingSet;
public int useOutputsAsTargetsInCT_RelearningScheme;

public boolean agentsAreDisplayingData, readWeightsFromFile;
private BogliettiModelSwarm baseModel;

public ForecastModelSwarm (BogliettiModelSwarm aBaseModel, Zone aZone) {

 super (aZone);
  baseModel = aBaseModel;
    agentNumber                  =  baseModel.sellersNumber;
    inputNodeNumber              =  4;
    hiddenNodeNumber             =  5;
    outputNodeNumber             =  2;
    patternNumberInVerificationSet     =
              aBaseModel.linesNumber;
    patternNumberInTrainingSet         =
              aBaseModel.linesNumber * aBaseModel.lastWeek;
    epochNumberInEachTrainingCycles    =  100;
    agentsAreDisplayingData            =  false;
    readWeightsFromFile                =  true;
    epochGroups                  =  0;

    onlyVerificationRun                                  = true;
```

```
   internalDataGeneration                                = true;

class ModelProbeMap extends EmptyProbeMapImpl {
  private VarProbe probeVariable (String name) {
    return
      Globals.env.probeLibrary.getProbeForVariable$inClass
      (name, ForecastModelSwarm.this.getClass ());
  }
  private MessageProbe probeMessage (String name) {
    return
      Globals.env.probeLibrary.getProbeForMessage$inClass
      (name, ForecastModelSwarm.this.getClass ());
  }

  private void addVar (String name) {
    addProbe (probeVariable (name));
  }
  private void addMessage (String name) {
    addProbe (probeMessage (name));
  }
  public ModelProbeMap (Zone _aZone, Class aClass) {
    super (_aZone, aClass);
    addVar ("hiddenNodeNumber");
    addVar ("epochNumberInEachTrainingCycles");
    addVar ("readWeightsFromFile");
    addVar ("onlyVerificationRun");
  }
}
Globals.env.probeLibrary.setProbeMap$For
  (new ModelProbeMap (aZone, getClass ()), getClass ());
 }

 public Object buildObjects () {
    agentNumber = baseModel.sellersNumber;

      Agent anAgent;
      DataWarehouse aDataWarehouse;
      SpecificInterface anInterface;
      List sellersList;
      sellersList = baseModel.getSellersList();
      SellerDataWarehouse aSDW;
```

```
   patternNumberInVerificationSet     =
              baseModel.linesNumber;
   patternNumberInTrainingSet         =
              baseModel.linesNumber * baseModel.lastWeek;


int i;
String trainingFileName = new String("data.training");
   String minmaxFileName = new String("minmax.data");

ruleMaker = new RuleMaker();

ruleMaster = new RuleMaster(ruleMaker, baseModel.getEnvironment(),
                                          onlyVerificationRun);


weightRange  =(float)0.3;
eps          =(float)0.6;
alpha        =(float)0.9;
usingRandomOrderInLearning =1;
longTermLearningInCT_OnlyWithCompleteTrainingSet =1;
useOutputsAsTargetsInCT_RelearningScheme =0;

agentList  = new ListImpl(getZone());

   for (i=1;i<=agentNumber;i++){

aDataWarehouse = new DataWarehouse(internalDataGeneration);
aDataWarehouse.setFileNames("",trainingFileName,minmaxFileName,"");


aDataWarehouse.setNeuralNetParameters(inputNodeNumber,
       hiddenNodeNumber,  outputNodeNumber,
       patternNumberInVerificationSet,  patternNumberInTrainingSet,
       epochNumberInEachTrainingCycles);
aDataWarehouse.setBackPropagationParameters(
           weightRange, eps, alpha,
           usingRandomOrderInLearning,
           longTermLearningInCT_OnlyWithCompleteTrainingSet,
           useOutputsAsTargetsInCT_RelearningScheme);
aDataWarehouse.createEnd();
```

```
   anInterface = new SpecificInterface(aDataWarehouse, i);
   aSDW = (SellerDataWarehouse)((Seller)sellersList.atOffset(i-1)).
                              getMyDataWarehouse();
   anAgent = new Agent(i, readWeightsFromFile, ruleMaster, aDataWarehouse,
                               anInterface, agentsAreDisplayingData, aSDW);
   agentList.addLast(anAgent);

    }
   return this;
 }

 public Object buildActions ()      {
      super.buildActions();
      modelActions = new ActionGroupImpl (getZone ());

 try {
  ActionForEach actionForEach;

  Selector sel =
    new Selector (Class.forName ("Agent"), "step2", false);

  actionForEach =
    modelActions.createActionForEach$message (agentList, sel);
}
catch (Exception e) {
  System.err.println("Exception step2: " + e.getMessage ());
}

try {
  ActionForEach actionForEach;

  Selector sel =
    new Selector (Class.forName ("Agent"), "step3", false);

  actionForEach =
    modelActions.createActionForEach$message (agentList, sel);
}
catch (Exception e) {
  System.err.println("Exception step3: " + e.getMessage ());
}

modelSchedule = new ScheduleImpl (getZone (), 2);
   modelSchedule.at$createAction (0, modelActions);
    return this;
    }

    public Activity activateIn (Swarm swarmContext) {
    super.activateIn (swarmContext);
    modelSchedule.activateIn (this);
    return getActivity ();
    }

    public int currentEpochGroups () {
        epochGroups++;
        return epochGroups;
    }

    public int getCurrentEpochGroup () { return epochGroups; }
    public int getPatternNumberInVerificationSet () { return
                    patternNumberInVerificationSet;     }
    public int getAgentNumber () {  return agentNumber;}
    public List getList () {   return agentList; }

    public Object orderAgentsToSaveItsWeight() {
        int i;

        for (i=0; i < agentNumber; i++)
            ((Agent)agentList.atOffset(i)).save();
        return this;
    }
}
```

Interface.java

```
import swarm.Globals;
import java.io.*;

public class Interface {
    protected int number, patternNumberInVerificationSet, inputNodeNumber;
    protected Matrix outputVerificationMatrix, targetVerificationMatrix;
    protected Matrix dataVerificationMatrix, dataTrainingMatrix;
    protected Matrix minmax;
    protected DataWarehouse myAgentDataWarehouse;
```

```
    protected int useEO;

    public Interface(DataWarehouse aDw, int agentNumber) {
        myAgentDataWarehouse = aDw;
        number = agentNumber;
        initialize();
        inputNodeNumber = myAgentDataWarehouse.getInputNodeNumber();
        patternNumberInVerificationSet =

    myAgentDataWarehouse.getPatternNumberInVerificationSet();
        outputVerificationMatrix =
                myAgentDataWarehouse.getOutputVerificationMatrix();
        targetVerificationMatrix =
                myAgentDataWarehouse.getTargetVerificationMatrix();
        dataVerificationMatrix   =
                myAgentDataWarehouse.getDataVerificationMatrix();
        dataTrainingMatrix       =
                myAgentDataWarehouse.getDataTrainingMatrix();
        minmax                                  =
                myAgentDataWarehouse.getMinmax();
    }

    public Object setUseEO(int eo) {
        useEO=eo;
        return this;
    }
    public int getAgentNumber () {  return number;    }

    public Object initialize () { return this;}
    public Object setCT_Inputs () {  return this;}
    public Object setCT_Targets () { return this;}
    public Object doActionsForAgentStep1 () { return this;}
    public Object doActionsForAgentStep2 () { return this;}
    public Object doActionsForAgentStep3 () { return this;}
}
```

Matrix.java

```
import swarm.Globals;
import java.io.*;

public class Matrix {
    float[] m;
    int rows, cols, max, code, firstRow;

 public Matrix(int nRows, int nCols, int nCode) {
        firstRow=0;
        rows = nRows;
        if (rows < 0) {rows *= -1; firstRow=rows;}
        cols= nCols;
        if (cols < 0) cols *= -1;
        m = new float[rows * cols];

      max = rows * cols;
      code = nCode;
 }

 public Matrix(int dimension, int nCode) {
        firstRow=0;
        rows= dimension;
        if (rows < 0){ rows *= -1; firstRow=rows;}
        m = new float[rows];
        cols=1;
        max=rows;
        code=nCode;
   }

   public Object setDimensionRows$Cols$Code(int nRows, int nCols, int nCode) {
        firstRow=0;
        rows = nRows;
        if (rows < 0) {rows *= -1; firstRow=rows;}
        cols= nCols;
        if (cols < 0) cols *= -1;
        m = new float[rows * cols];
        max = rows * cols;
        code = nCode;
        return this;
   }

   public Object setDimension$Code(int dimension, int nCode) {
        firstRow=0;
        rows= dimension;
        if (rows < 0){ rows *= -1; firstRow=rows;}
```

```
    m = new float[rows];
    cols=1;
    max=rows;
    code=nCode;
    return this;
}

public Object shift () {
  int i, j;

  firstRow--; if (firstRow < 0) firstRow = 0;
  if(rows==1) return this;

  for (i=1;i<rows;i++)
  for (j=0;j<cols;j++)
  m[(i-1)*cols+j]=m[i*cols+j];

  return this;
}

public int getFirstRow () { return firstRow; }

public float R$C(int r, int c) {
    if (r >= rows || c >= cols){
    System.out.println("Max dimension exceeded in an instance of
                            Matrix.h (code=" + code + ")");
    System.out.println("(reading at " + r + " " + c + ")");
    System.exit(0);}
    return m[r*cols+c];
}

public float P(int p) {
    if (p >= max){
    System.out.println("Max dimension exceeded in an instance of
                            Matrix.h (code=" + code + ")");
    System.out.println("(reading at " + p + ")");
    System.exit(0);}
    return m[p];
}

public Object R$C$setFrom(int r, int c, float v) {
    if (r >= rows || c >= cols){
    System.out.println("Max dimension exceeded in an instance of
                            Matrix.h (code=" + code + ")");
    System.out.println("(writing at " + r + " " + c + ")");
    System.exit(0);}
    m[r*cols+c]=v;
    return this;
}

public Object P$setFrom(int p, float v) {
    if (p >= max){
    System.out.println("Max dimension exceeded in an instance of
                            Matrix.h (code=" + code + ")");
    System.out.println("(writing at " + p + ")");
    System.exit(0);}
    m[p]=v;
    return this;
}

public Object R$C$setFromP(int r, int c, float v) {
    if (r >= rows || c >= cols){
    System.out.println("Max dimension exceeded in an instance of
                            Matrix.h (code=" + code + ")");
    System.out.println("(writing at " + r + " " + c + ")");
    System.exit(0);}
    m[r*cols+c]+=v;
    return this;
}

public Object P$setFromP(int p, float v) {
    if (p >= max){
    System.out.println("Max dimension exceeded in an instance of
                            Matrix.h (code=" + code + ")");
    System.out.println("(writing at " + p +")");
    System.exit(0);}
    m[p]+=v;
    return this;
}

public int getNumberOfElements () { return max; }
public int getNumberOfRows () { return rows; }
public int getNumberOfCols () { return cols;}
public float[] getAddress () { return m; }
```

```
public float getAddress1 () {  return m[1]; }

public float R$C$asExternalValueUsing$withShift
                        (int r, int c, Matrix mm, int sf) {
    if (r >= rows || c >= cols){
    System.out.println("Max dimension exceeded in an instance of
                        Matrix.h (code=" + code + ")");
    System.out.println("(reading to external at " + r + " " + c + ")");
    System.exit(0);}
    return
        ((m[r*cols+c]-mm.R$C(c+sf,2))/
                (mm.R$C(c+sf,3)-mm.R$C(c+sf,2)) ) *
                (mm.R$C(c+sf,1)-mm.R$C(c+sf,0))+mm.R$C(c+sf,0);
}

public Object R$C$setFrom$asExternalValueUsing$withShift
                        (int r, int c, float v,Matrix mm, int sf) {
    if (r >= rows || c >= cols){
    System.out.println("Max dimension exceeded in an instance of
                        Matrix.h (code=" + code + ")");
    System.out.println("(writing from external at " + r + " " + c + ")");
    System.exit(0);}
    m[r*cols+c]=
            ((v-mm.R$C(c+sf,0))/
                (mm.R$C(c+sf,1)-mm.R$C(c+sf,0)) ) *
                (mm.R$C(c+sf,3)-mm.R$C(c+sf,2))+mm.R$C(c+sf,2);
    return this;
}

public Object R_lastAndC$setFrom$asExternalValueUsing$withShift
                                (int c, float v, Matrix mm, int sf) {
    if (c >= cols){
    System.out.println("Max dimension exceeded in an instance of
                        Matrix.h (code=" + code + ")");
    System.out.println("(writing from external at " + (rows-1) + " " + c + ")");
    System.exit(0);}
    m[(rows-1)*cols+c]=
            ((v-mm.R$C(c+sf,0))/
                (mm.R$C(c+sf,1)-mm.R$C(c+sf,0)) ) *
                (mm.R$C(c+sf,3)-mm.R$C(c+sf,2))+mm.R$C(c+sf,2);
    return this;
}
```

```
}
```

MatrixMult.java

```
import swarm.Globals;
import java.io.*;

public final class MatrixMult {
  public MatrixMult() {  }
    static final public void mult$by$to$row1$col1row12$col2
                    (float[] m1, float[] m2, float[] m3, int r1, int c1r2, int c2) {
    int i, j, l;

      for (i=0;i<r1;i++)
        for (j=0;j<c2;j++)
        {
            m3[i*c2+j]=0;
            for (l=0;l<c1r2;l++) m3[i*c2+j] += m1[i*c1r2+l]*m2[l*c2+j];
        }

      return ;
    }

    static final public void m1$m2$to(Matrix m1, Matrix m2, Matrix m3) {
                int r1, c1, r2, c2, r3, c3, i, j, l;
      r1=m1.getNumberOfRows();
      c1=m1.getNumberOfCols();
      r2=m2.getNumberOfRows();
      c2=m2.getNumberOfCols();
      r3=m3.getNumberOfRows();
      c3=m3.getNumberOfCols();

      if (c1 != r2 || r1 != r3 || c2 != c3)
        {
        System.out.println("Error: matrix rows and cols not congruent
                                in MatrixMult.");
        System.exit(0);
        }


      for (i=0;i<r1;i++) for (j=0;j<c2;j++)
```

```
      {
        m3.R$C$setFrom(i,j,0);
        for (l=0;l<c1;l++) m3.R$C$setFromP(i,j,m1.R$C(i,l) * m2.R$C(l,j));
      }

    return ;
  }

  static final public void m1$m2$to1(Matrix m1,Matrix m2, Matrix m3)          {
      int r1, c1, r2, c2, r3, c3, i, j, l;

    r1=m1.getNumberOfRows();
    c1=m1.getNumberOfCols();
    r2=m2.getNumberOfRows();
    c2=m2.getNumberOfCols();
    r3=m3.getNumberOfRows();
    c3=m3.getNumberOfCols();


    if (c1 != r2 || r1 != r3-1 || c2 != c3)
      {
      System.out.println("Error: matrix rows and cols not congruent
                          in MatrixMult.");
      System.exit(0);
      }

    for (i=1;i<r3;i++) for (j=0;j<c2;j++)
     {
     m3.R$C$setFrom(i,j,0);
     for (l=0;l<c1;l++) m3.R$C$setFromP(i,j,m1.R$C(i-1,l) * m2.R$C(l,j));
     }

    return ;
  }
}
```

RuleMaker.java

```
import swarm.Globals;
import java.io.*;
import java.lang.Math;

public class RuleMaker {
    private DataWarehouse agentDataWarehouse;
    private Matrix dataTrainingMatrix, minmax, outputTrainingMatrix;
    private Matrix targetTrainingMatrix;
    private Matrix wih, dwih, who, dwho;
    private Matrix inputLayer, hiddenLayer, outputLayer, deltaOut;
    private int inputNodeNumber, inputNodeNumber1;
    private int hiddenNodeNumber, hiddenNodeNumber1;
    private int outputNodeNumber, patternNumberInTrainingSet;
    private int epochNumberInEachTrainingCycles;
    private int longTermLearningInCT_OnlyWithCompleteTrainingSet;
    private int useOutputsAsTargetsInCT_RelearningScheme;
    private float eps, alpha, backPropagationError, proportionalError;
    private int usingRandomOrderInLearning;
    private int[] p;
    private int currentAgentInExecution;
   private SellerDataWarehouse sellerDataWarehouse;

    public RuleMaker() {
        p = null;
        currentAgentInExecution = 0;
    }

    public Object adaptRulesToInputOutputValuesInDataWarehouse(
                    DataWarehouse agentDw, SellerDataWarehouse aSDW) {

        int i, I, j, k, iii, n, temp, row, col, colOut, initRow;
        float sum, delta, epochNumber, usingOutputsAsTargetsInCT_Relearning;

        agentDataWarehouse = agentDw;
        sellerDataWarehouse = aSDW;

        currentAgentInExecution = sellerDataWarehouse.getMySellerNumber();

        dataTrainingMatrix   = agentDataWarehouse.getDataTrainingMatrix();
        outputTrainingMatrix = agentDataWarehouse.getOutputTrainingMatrix();
        targetTrainingMatrix = agentDataWarehouse.getTargetTrainingMatrix();
        minmax=agentDataWarehouse.getMinmax();
        inputNodeNumber  = agentDataWarehouse.getInputNodeNumber();
        inputNodeNumber1 = inputNodeNumber+1;
        hiddenNodeNumber = agentDataWarehouse.getHiddenNodeNumber();
```

```
        hiddenNodeNumber1= hiddenNodeNumber+1;
        outputNodeNumber = agentDataWarehouse.getOutputNodeNumber();
        patternNumberInTrainingSet =
                              agentDataWarehouse.getPatternNumberInTrainingSet();
        epochNumberInEachTrainingCycles =
                    agentDataWarehouse.getEpochNumberInEachTrainingCycles();

        wih = agentDataWarehouse.getWih();
        dwih= agentDataWarehouse.getDwih();
        who = agentDataWarehouse.getWho();
        dwho= agentDataWarehouse.getDwho();
        inputLayer  = agentDataWarehouse.getInputLayer();
        hiddenLayer = agentDataWarehouse.getHiddenLayer();
        outputLayer = agentDataWarehouse.getOutputLayer();
        deltaOut = agentDataWarehouse.getDeltaOut();

        System.out.println("Making learning ..." + currentAgentInExecution);
        makeTargetsForLearning();

        eps=agentDataWarehouse.getEps();
        alpha=agentDataWarehouse.getAlpha();
        usingRandomOrderInLearning=
                              agentDataWarehouse.getRandomOrderInLearning();
        longTermLearningInCT_OnlyWithCompleteTrainingSet=
agentDataWarehouse.getLongTermLearningInCT_OnlyWithCompleteTrainingSet();
        useOutputsAsTargetsInCT_RelearningScheme=
    agentDataWarehouse.getUseOutputsAsTargetsInCT_RelearningScheme();

        if(patternNumberInTrainingSet>0)
            useOutputsAsTargetsInCT_RelearningScheme=0;
        if(patternNumberInTrainingSet < 0)
            patternNumberInTrainingSet *= -1;

        k=inputNodeNumber+outputNodeNumber;

        for (usingOutputsAsTargetsInCT_Relearning=0;
              usingOutputsAsTargetsInCT_Relearning <=
                 useOutputsAsTargetsInCT_RelearningScheme;
              usingOutputsAsTargetsInCT_Relearning++)
        {

        p = null;
        p = new int[patternNumberInTrainingSet];
        for (i=0;i<patternNumberInTrainingSet;i++) p[i]=i;

        for (i=0;i<patternNumberInTrainingSet;i++)
        for (j=0;j<outputNodeNumber;j++)
            targetTrainingMatrix.R$C$setFrom(i,j,
                    dataTrainingMatrix.R$C(i,inputNodeNumber+j));

        initRow= dataTrainingMatrix.getFirstRow();

        epochNumber=epochNumberInEachTrainingCycles;
        if (longTermLearningInCT_OnlyWithCompleteTrainingSet == 1 &&
           initRow != 0)
          {initRow=patternNumberInTrainingSet-1;
           epochNumber=1;
          }
          if (usingOutputsAsTargetsInCT_Relearning == 0 &&
           useOutputsAsTargetsInCT_RelearningScheme == 1)
           {initRow=patternNumberInTrainingSet-1;
           epochNumber=1;}

        for (n=0; n<epochNumber;n++)
        {
           for (i=patternNumberInTrainingSet; i>initRow+1; i--)
           {
           iii= Globals.env.uniformIntRand.getIntegerWithMin$withMax(initRow, i-1);
           temp=p[iii];
           p[iii]=p[i-1];
           p[i-1]=temp;
           }
          for (j=0;j<outputNodeNumber;j++)
          deltaOut.P$setFrom(j, (targetTrainingMatrix.R$C(i,j)-
                              outputLayer.P(j))*
                              outputLayer.P(j)*((float)1.0-outputLayer.P(j)));


          // dwho
          for (row=0;row<outputNodeNumber; row++)
          for (col=0;col<hiddenNodeNumber1;col++)
          dwho.R$C$setFrom(row, col, eps*deltaOut.P(row)* hiddenLayer.P(col)
                            + alpha*dwho.R$C(row, col));
```

```
// dwih
for (row=0;row<hiddenNodeNumber; row++)
{
     sum=0;
     for (colOut=0;colOut<outputNodeNumber;colOut++)
     sum+=deltaOut.P(colOut)*who.R$C(colOut,row+1);

     delta=hiddenLayer.P(row+1)*((float)1.0-hiddenLayer.P(row+1))*sum;

for (col=0;col<inputNodeNumber1;col++)
dwih.R$C$setFrom(row, col, eps*delta*inputLayer.P(col)
                  + alpha*dwih.R$C(row,col));
}

// who and wih correction
for (row=0;row<outputNodeNumber; row++)
for (col=0;col<hiddenNodeNumber1;col++)
who.R$C$setFromP(row, col, dwho.R$C(row, col));

for (row=0;row<hiddenNodeNumber; row++)
for (col=0;col<inputNodeNumber1; col++)
wih.R$C$setFromP(row, col, dwih.R$C(row, col));

}

if (n==0)
{
backPropagationError=0;
proportionalError=0;
 for (i=initRow;i<patternNumberInTrainingSet;i++)
 {
 for (j=0;j<outputNodeNumber;j++)
   backPropagationError+=Math.pow(outputTrainingMatrix.R$C(i,j)-
                         targetTrainingMatrix.R$C(i,j),2)/
                         (2*(patternNumberInTrainingSet-initRow));
 for (j=0;j<outputNodeNumber;j++){
    proportionalError+=
       ( Math.abs(outputTrainingMatrix.R$C(i,j)
           - targetTrainingMatrix.R$C(i,j)) /
           ((minmax.R$C(j+inputNodeNumber,3) -
             minmax.R$C(j+inputNodeNumber,2))/2.)
       ) / (outputNodeNumber*(patternNumberInTrainingSet-initRow));  }
            }
        System.out.println("Prop error: " + proportionalError);
      }
    }

      if (usingOutputsAsTargetsInCT_Relearning == 0 &&
          useOutputsAsTargetsInCT_RelearningScheme == 1)
            for (j=0;j<outputNodeNumber;j++)
               dataTrainingMatrix.R$C$setFrom(patternNumberInTrainingSet-1,
                           inputNodeNumber+j ,
             outputTrainingMatrix.R$C(patternNumberInTrainingSet-1, j));
    }
    return this;
}

public Object makeTargetsForLearning() {

    //counters
    int i, j;
    long x,y;
    int line, week;
    long customersToEnd;
    double ordersByCustomers, ordByCust2;
    long finalCustomers;

    // data of seller datawarehouse
    int nLines, nWeeks;
    long[] customersPerWeek;
    long[][] totalSales;

    Matrix nMM = agentDataWarehouse.getNewMinMax();
    float o1, o2;

    totalSales = sellerDataWarehouse.getTotalSales();
    customersPerWeek = sellerDataWarehouse.getCustomersPerWeek();

    nLines = sellerDataWarehouse.getTotalLines();
    nWeeks = sellerDataWarehouse.getTotalWeeks();

    finalCustomers = sellerDataWarehouse.getCustomersBargained();

    for (i=0; i < patternNumberInTrainingSet; i++)
```

```
{
    week = (int)
        dataTrainingMatrix.R$C$asExternalValueUsing$withShift(
                                                i, 0, minmax, 0);
    line = (int)
        dataTrainingMatrix.R$C$asExternalValueUsing$withShift(
                                                i, 1, minmax, 0);

    // calculating first target node
    x=0;
    for (j=0; j < week; j++)
        x += customersPerWeek[j];
    customersToEnd = finalCustomers - x;
    if(customersToEnd < 0)
        System.out.println("Error customersToEnd: " + customersToEnd);

    o1 = customersToEnd;
    dataTrainingMatrix.R$C$setFrom$asExternalValueUsing$withShift(
                                        i, 4, o1, minmax,0);

    targetTrainingMatrix.R$C$setFrom(i,0,
            dataTrainingMatrix.R$C(i,4));
    checkNewMinMax(nMM, inputNodeNumber, o1);

    // calculating second target node
    x=0; y=0;
    for (j = 0; j < week; j++)
        y += totalSales[line][j];
    for (j = week; j < nWeeks; j++)
        x += totalSales[line][j];

    ordersByCustomers = 0;
    ordByCust2 = 1;
    if (customersToEnd != 0)
        ordersByCustomers = x / customersToEnd;
    if (finalCustomers - customersToEnd != 0)

    ordByCust2 = y  / (finalCustomers - customersToEnd);
    if (ordByCust2 == 0) ordByCust2 = 1;
    o2 = (float) (ordersByCustomers / ordByCust2);
    if (o2 > 2.0) o2 = 2;
    dataTrainingMatrix.R$C$setFrom$asExternalValueUsing$withShift(
                                                i,5, o2, minmax,0);

            targetTrainingMatrix.R$C$setFrom(i,1,
                    dataTrainingMatrix.R$C(i,5));
            checkNewMinMax(nMM, inputNodeNumber + 1, o2);
        }

        saveTrainingMatrix();
        saveMinMaxMatrix(minmax, nMM);

        return this;
    }

    public float getBackPropagationErrorInTrainingSet ()        {
      return backPropagationError;
    }

    public float getProportionalErrorInTrainingSet ()   {
      return proportionalError;
    }

    private Object saveTrainingMatrix () {

        int i, j, pN;
        PrintWriter tempOutFile = null;
        String name;

        //wih
        name = new String("train" + currentAgentInExecution + ".matrix");

        try {
            tempOutFile= new PrintWriter(new FileWriter(name));
            } catch (Exception e) {
                System.err.println("Error in opening file " + name +
                                        ": " + e.getMessage());
            }

        try {
        for (i=0; i < patternNumberInTrainingSet; i++)
             {
                tempOutFile.print("" + (float)
dataTrainingMatrix.R$C$asExternalValueUsing$withShift(i,0,minmax,0) + " ");
```

```
                tempOutFile.print("" + (float)
dataTrainingMatrix.R$C$asExternalValueUsing$withShift(i,1,minmax,0) + " ");
                tempOutFile.print("" + (float)
dataTrainingMatrix.R$C$asExternalValueUsing$withShift(i,2,minmax,0) + " ");
                tempOutFile.print("" + (float)
dataTrainingMatrix.R$C$asExternalValueUsing$withShift(i,3,minmax,0) + " ");

                tempOutFile.print("" + (float)
dataTrainingMatrix.R$C$asExternalValueUsing$withShift(i,4,minmax,0) + " ");

                tempOutFile.println("" + (float)
dataTrainingMatrix.R$C$asExternalValueUsing$withShift(i,5,minmax,0));



            }
        } catch (Exception e) {
                System.err.println("Error writing file " + name + ": " +
                                e.getMessage());
                }

        tempOutFile.close();
        tempOutFile = null;

        return this;
    }

    private Object saveMinMaxMatrix (Matrix oldMatrix, Matrix newMatrix) {
        int i, j;
        PrintWriter tempOutFile = null;
        String name;

        boolean flag = false;
        for (i=0; i < inputNodeNumber + outputNodeNumber; i++)
            for (j=0; j < 2; j++)
                if (oldMatrix.R$C(i,j) != newMatrix.R$C(i,j))
                    flag = true;
        if (!flag)
            return this;

        //wih
        name = agentDataWarehouse.getMinMaxFileName();

        try {
            tempOutFile= new PrintWriter(new FileWriter(name));
            } catch (Exception e) {
                System.err.println("Error in opening file " + name + ": " +
                                        e.getMessage());
            }

        try {
            for (i=0; i < inputNodeNumber + outputNodeNumber; i++)
            {
                for (j=0; j < 4; j++)
                    tempOutFile.print("" + newMatrix.R$C(i,j) + " ");
                tempOutFile.println();
            }
        } catch (Exception e) {
                System.err.println("Error writing file " + name + ": " +
                                                e.getMessage());
                }

        tempOutFile.close();
        tempOutFile = null;

        return this;
    }

    private void checkNewMinMax(Matrix newMinMax, int nodeNumber, float value) {

        if (value < newMinMax.R$C(nodeNumber,0))
        {
            newMinMax.R$C$setFrom(nodeNumber,0, value);
            System.out.println("WARNING: min max matrix has changed!!");
            System.out.println(" please run learning again. Node: " +
                                        nodeNumber + " val: " + value);
        }

        if (value > newMinMax.R$C(nodeNumber,1))
        {
            newMinMax.R$C$setFrom(nodeNumber,1, value);
            System.out.println("WARNING: min max matrix has changed!!");
            System.out.println(" please run learning again. Node: " +
                                        nodeNumber + " val: " + value);
        }
```

```
        return;
    }
}
```

RuleMaster.java

```
import swarm.Globals;
import java.io.*;
import java.lang.Math;

public class RuleMaster {
    private DataWarehouse agentDataWarehouse;
    private Matrix dataVerificationMatrix, outputVerificationMatrix;
    private Matrix targetVerificationMatrix;
    private Matrix wih, dwih, who, dwho;
    private int inputNodeNumber, inputNodeNumber1;
    private int hiddenNodeNumber, hiddenNodeNumber1, outputNodeNumber;
    private int patternNumberInVerificationSet;
    private Matrix inputLayer, hiddenLayer, outputLayer, minmax;
    private float backPropagationError, proportionalError;
    private RuleMaker ruleMaker;

    private Environment environment;
    private SellerDataWarehouse sellerDataWarehouse;
    private boolean onlyVerificationRun;

    public RuleMaster(RuleMaker rm, Environment env,
                      boolean isOnlyVerificationRun) {
        ruleMaker = rm;
        environment = env;
        onlyVerificationRun = isOnlyVerificationRun;
    }

    public Object applyRulesToInputOutputValuesInDataWarehouseStepA(
                                DataWarehouse agentDw,
                                SellerDataWarehouse sDW) {

        int i, j;
        if (environment.isNoMoreTheForecastTime())
            return this;

agentDataWarehouse=agentDw;
sellerDataWarehouse=sDW;

dataVerificationMatrix   = agentDataWarehouse.getDataVerificationMatrix();
outputVerificationMatrix =
            agentDataWarehouse.getOutputVerificationMatrix();
targetVerificationMatrix = agentDataWarehouse.getTargetVerificationMatrix();
minmax=agentDataWarehouse.getMinmax();
inputNodeNumber  = agentDataWarehouse.getInputNodeNumber();
inputNodeNumber1 = inputNodeNumber+1;
hiddenNodeNumber = agentDataWarehouse.getHiddenNodeNumber();
hiddenNodeNumber1= hiddenNodeNumber+1;
outputNodeNumber = agentDataWarehouse.getOutputNodeNumber();
patternNumberInVerificationSet =
        agentDataWarehouse.getPatternNumberInVerificationSet();

fillVerificationTrainingMatrix();

if (patternNumberInVerificationSet < 0)
        patternNumberInVerificationSet *= -1;

wih = agentDataWarehouse.getWih();
dwih= agentDataWarehouse.getDwih();
who = agentDataWarehouse.getWho();
dwho= agentDataWarehouse.getDwho();
inputLayer  = agentDataWarehouse.getInputLayer();
hiddenLayer = agentDataWarehouse.getHiddenLayer();
outputLayer = agentDataWarehouse.getOutputLayer();

backPropagationError=0;
proportionalError=0;

for (i = 0; i < patternNumberInVerificationSet; i++)
{
    inputLayer.P$setFrom(0,(float)1.0); // bias artificial input
    for (j=0;j<inputNodeNumber;j++)
      inputLayer.P$setFrom(j+1, dataVerificationMatrix.R$C(i,j));

    MatrixMult.m1$m2$to1(wih ,inputLayer, hiddenLayer);
    VectorTransFunc.from1$to1(hiddenLayer, hiddenLayer);
    hiddenLayer.P$setFrom(0,(float)1.0);
```

```
        MatrixMult.m1$m2$to(who, hiddenLayer, outputLayer);
        VectorTransFunc.from$to(outputLayer, outputLayer);
        for (j=0;j<outputNodeNumber;j++)
        {
            outputVerificationMatrix.R$C$setFrom(i,j, outputLayer.P(j));
            targetVerificationMatrix.R$C$setFrom(i,j,
                    dataVerificationMatrix.R$C(i, j + inputNodeNumber));
        }
        for (j=0;j<outputNodeNumber;j++)
          backPropagationError+=Math.pow(outputVerificationMatrix.R$C(i,j)-
                                         targetVerificationMatrix.R$C(i,j),2)/
                                    (2*patternNumberInVerificationSet);

        for (j = 0; j < outputNodeNumber; j++)
        {
          proportionalError+=
            ( Math.abs(outputVerificationMatrix.R$C(i,j)
                - targetVerificationMatrix.R$C(i,j)) /
                ((minmax.R$C(j+inputNodeNumber,3) -
                  minmax.R$C(j+inputNodeNumber,2))/2.)
            ) / (outputNodeNumber*patternNumberInVerificationSet);
        }
        copyOutputVerificationToBaseModel();
    }
    return this;
}

public Object applyRulesToInputOutputValuesInDataWarehouseStepB(
                            DataWarehouse agentDw,
                            SellerDataWarehouse aSDW) {

    if (!environment.isTheForecastTime() || onlyVerificationRun)
        return this;

    agentDataWarehouse=agentDw;
    sellerDataWarehouse=aSDW;

    ruleMaker.adaptRulesToInputOutputValuesInDataWarehouse(
                                    agentDataWarehouse, sellerDataWarehouse);

    return this;
}
```

```
public float getBackPropagationErrorInVerificationSet () {
    return backPropagationError;
}

public float getProportionalErrorInVerificationSet () {
    return proportionalError;
}

public float getBackPropagationErrorInTrainingSet () {
    return ruleMaker.getBackPropagationErrorInTrainingSet();
}

public float getProportionalErrorInTrainingSet () {
    return ruleMaker.getProportionalErrorInTrainingSet();
}

public Object fillVerificationTrainingMatrix () {
    int i, j, pN, k;
    int pointTime;
    int nLines;
    long[][] totalSales;
    double totalSalesPerLine;
    float i1, i2, i3, i4;


    pointTime = environment.getCurrentTime();
    nLines = sellerDataWarehouse.getTotalLines();
    totalSales = sellerDataWarehouse.getTotalSales();
    Matrix nMM=agentDataWarehouse.getNewMinMax();

    for (j = 0; j < nLines; j++)
    {
        totalSalesPerLine = 0.0;
        for (k=0; k <= pointTime; k++)
            totalSalesPerLine += totalSales[j][k];

        i1 = (float) pointTime;
        i2 = (float) j;
        i3 = (float) sellerDataWarehouse.getCustomersBargained();
        if (sellerDataWarehouse.getCustomersBargained() != 0)
            i4 = (float) (totalSalesPerLine /
```

```
                sellerDataWarehouse.getCustomersBargained());
        else
            i4 = (float) 0.0;
        dataVerificationMatrix.R$C$setFrom$asExternalValueUsing$withShift(
                                                    j,0, i1, minmax,0);
        dataVerificationMatrix.R$C$setFrom$asExternalValueUsing$withShift(
                                                    j,1, i2, minmax,0);
        dataVerificationMatrix.R$C$setFrom$asExternalValueUsing$withShift(
                                                    j,2, i3, minmax,0);
        dataVerificationMatrix.R$C$setFrom$asExternalValueUsing$withShift(
                                                    j,3, i4, minmax,0);

        checkNewMinMax(nMM, 0, i1);
        checkNewMinMax(nMM, 1, i2);
        checkNewMinMax(nMM, 2, i3);
        checkNewMinMax(nMM, 3, i4);
    }
    return this;
}

public Object copyOutputVerificationToBaseModel() {
    int i, j, k;
    long[][] totalSales;
    Matrix dataTrainingMatrix;
    float out1, out2, tmp;
    int pointTime, nLines;
    double totalSalesPerLine;
    double[][] forecasts;

    totalSales = sellerDataWarehouse.getTotalSales();
    forecasts = sellerDataWarehouse.getTotalSalesForecasts();
    pointTime = environment.getCurrentTime();
    nLines = sellerDataWarehouse.getTotalLines();

    for (j=0; j < nLines; j++)
    {
        totalSalesPerLine = 0.0;
        for (k=0; k <= pointTime; k++)
            totalSalesPerLine += totalSales[j][k];
        out1 = outputVerificationMatrix.R$C$asExternalValueUsing$withShift(
                                    j,0,minmax,inputNodeNumber);
        out2 = outputVerificationMatrix.R$C$asExternalValueUsing$withShift(
                                    j,1,minmax,inputNodeNumber);
        tmp = 0;
        if (sellerDataWarehouse.getCustomersBargained() != 0)
            tmp = (float)(totalSalesPerLine /
                    sellerDataWarehouse.getCustomersBargained());
        forecasts[j][pointTime] = totalSalesPerLine +
                                        out1 * out2* tmp;

    }

    i = pointTime * nLines;
    dataTrainingMatrix = agentDataWarehouse.getDataTrainingMatrix();
    for (j = 0; j < nLines; j++)
    {
        dataTrainingMatrix.R$C$setFrom(j+i,0, (float)
                    dataVerificationMatrix.R$C(j,0));
        dataTrainingMatrix.R$C$setFrom(j+i,1, (float)
                    dataVerificationMatrix.R$C(j,1));
        dataTrainingMatrix.R$C$setFrom(j+i,2, (float)
                    dataVerificationMatrix.R$C(j,2));
        dataTrainingMatrix.R$C$setFrom(j+i,3, (float)
                    dataVerificationMatrix.R$C(j,3));
    }
    return this;
}

private void checkNewMinMax(Matrix newMinMax, int nodeNumber, float value) {
    if (value < newMinMax.R$C(nodeNumber,0))
    {
        newMinMax.R$C$setFrom(nodeNumber,0, value);
        System.out.println("WARNING: min max matrix has changed!!");
        System.out.println(" please run learning again. Node: " +
                                nodeNumber + " val: " + value);
    }

    if (value > newMinMax.R$C(nodeNumber,1))
    {
        newMinMax.R$C$setFrom(nodeNumber,1, value);
        System.out.println("WARNING: min max matrix has changed!!");
        System.out.println(" please run learning again. Node: " +
                                nodeNumber + " val: " + value);
    }
```

```
        return;
    }
}
```

SpecificInterface.java

```
import swarm.Globals;
import java.io.*;

public class SpecificInterface extends Interface {
  public SpecificInterface(DataWarehouse aDw, int agentNumber) {
        super(aDw,agentNumber);
  }
    public float getI1 () {
        int k = patternNumberInVerificationSet;

    return dataVerificationMatrix.R$C$asExternalValueUsing$withShift(
                                        k,0,minmax,0);
    }

    public float getI2 () {
      int k = patternNumberInVerificationSet;
      return dataVerificationMatrix.R$C$asExternalValueUsing$withShift(
                                        k,1,minmax,0);
    }

    public float getI3 () {
      int k = patternNumberInVerificationSet;
      return dataVerificationMatrix.R$C$asExternalValueUsing$withShift(
                                        k,2,minmax,0);
    }

    public float getI4 () {
      int k = patternNumberInVerificationSet;
      return dataVerificationMatrix.R$C$asExternalValueUsing$withShift(
                                        k,3,minmax,0);
    }

    public float getCustomers_out () {
      int k= patternNumberInVerificationSet;
      return outputVerificationMatrix.R$C$asExternalValueUsing$withShift(
                                        k,0,minmax,inputNodeNumber);
    }

    public float getCustomers_target () {
      int k = patternNumberInVerificationSet;
      return targetVerificationMatrix.R$C$asExternalValueUsing$withShift(
                                        k,0,minmax,inputNodeNumber);
    }

    public float getOrders_out () {
      int k= patternNumberInVerificationSet;
      return outputVerificationMatrix.R$C$asExternalValueUsing$withShift(
                                        k,1,minmax,inputNodeNumber);
    }

    public float getOrders_target () {
      int k = patternNumberInVerificationSet;
      return targetVerificationMatrix.R$C$asExternalValueUsing$withShift(
                                k,1,minmax,inputNodeNumber);
    }

    public Object doActionForAgentStep2 () { return this; }
    public Object doActionsForAgentStep3 () { return this; }
    public Object initialize () {  return this; }
    public Object setCT_Inputs () { return this;}
    public Object setCT_Targets ()  {  return this;}
}
```

VectorTransFunc.java

```
import swarm.Globals;
import java.io.*;
import java.lang.Math;

public final class VectorTransFunc {
  public VectorTransFunc() {
  }
  static final public void from$to$itemNumber(float[] v1, float[] v2, int n)          {
     int i;
       for (i=0;i<n;i++) v2[i]=(float) (1.0/(1.0 + Math.exp((double)-v1[i])));
```

```
      return ;
    }
    static final public void from$to(Matrix v1,Matrix v2)           {
        int i, n;
      n=v1.getNumberOfElements();
      for (i=0;i<n;i++) v2.P$setFrom(i, (float)(1.0/(1.0 + Math.exp(-v1.P( i)))));
      return ;
    }
    static final public void from1$to1(Matrix v1, Matrix v2)       {
        int i, n;
      n=v1.getNumberOfElements();
      for (i=1;i<n;i++) v2.P$setFrom(i, (float)(1.0/(1.0 + Math.exp(-v1.P(i)))));
      return ;
    }
}
```