APPENDICI

L’emblema del movimento

Il nome e l'emblema della Croce Rossa vennero stabiliti, come già detto in precedenza, con la Convenzione di Ginevra del 1864 per definire e riconoscere il carattere di neutralità degli ospedali, delle ambulanze e del personale sanitario. A tale scopo venne stabilita, inoltre, l'adozione di un bracciale e di una bandiera, uguali per tutti gli Stati, con una croce rossa in campo bianco come segno universale di protezione.

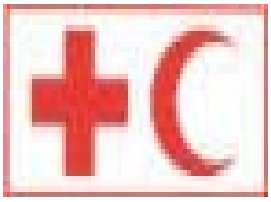
L’emblema suscitò in seguito diversi problemi, infatti nel novembre del 1876 la Turchia, in guerra da sei mesi con la Russia, dichiarò improvvisamente che l'emblema con la croce rossa contrastava con le convinzioni religiose delle sue truppe e di conseguenza adottò come segno distintivo la mezzaluna rossa in campo bianco. Tale emblema venne in seguito adottato anche da numerosi paesi arabi o di predominanza musulmana.

Nel 1923 anche la Persia adottò un terzo emblema: il leone e sole rossi su fondo bianco, nonostante nella Conferenza Diplomatica del 1946 si fosse ricordato come l'emblema della Croce Rossa fosse un segno internazionale privo di alcun significato religioso, per cui era illogico sostituirlo con emblemi nazionali che in tempo di conflitto, sono simbolo di belligeranza, la richiesta della Persia fu accettata dalla stessa Conferenza.

Infatti, l'art. 38 della *I Convenzione di Ginevra del 1949* recita:

> *"In omaggio alla Svizzera il segno araldico della croce rossa su fondo bianco, formato con l'inversione dei colori federali, è mantenuto comeemblema e segno distintivo del servizio sanitario degli eserciti. Tuttavia, per i paesi che impiegano come segno distintivo, in luogo della croce rossa, la mezzaluna rossa o il sole e leone rossi su fondo bianco, questi emblemi sono parimenti concessi nel caso della presente convenzione".*

In seguito alla caduta del regime degli Scià, con la costituzione della Repubblica Islamica dell'Iran, la Società Nazionale Iraniana decise di adottare l'emblema della mezzaluna rossa rinunciando al terzo simbolo.



La Conferenza Internazionale del 1997 ha istituito una commissione di esperti affinché risolva la confusione che la diversità dei simboli, vista anche la richiesta di

riconoscimento fatta da Israele del riconoscimento della stella di Davide rossa, potrebbe scatenare.

Le Convenzioni autorizzano il Movimento Internazionale a far uso, sia in pace che in guerra, dell'emblema della Croce Rossa su fondo bianco con il quale potranno designare quanto appartiene alle Società Nazionali: locali, vetture e personale come segno distintivo, autorizzando anche l'uso delle parole "croce rossa" e "mezzaluna rossa".

L'uso dell'emblema ha, però, un duplice aspetto: esso è usato a titolo indicativo quando, in tempo di pace, serve ad indicare le installazioni ed il personale delle Società Nazionali, deve essere, pertanto, di piccole dimensioni in modo da essere visibile solo da vicino. L'emblema usato a titolo protettivo è, invece, destinato ad essere visto dai combattenti in caso di conflitto armato e, in tal caso, sarà di grandi dimensioni e la sua utilizzazione è di competenza delle autorità preposte.

Gli organismi internazionali della Croce Rossa, così come il loro personale debitamente legittimato, sono autorizzati a servirsi dell'emblema protettore o indicativo secondo le circostanze e, in nome della Croce Rossa, in qualsiasi momento lo ritengano opportuno.

L'emblema è segno di protezione e come tale deve essere rispettato in quanto la persona o la cosa che lo porta è da considerarsi neutrale. Affinché sia salvaguardata la sua efficacia esso deve essere utilizzato senza abuso o perfidia.

DPR 27 marzo 1992

Atto di indirizzo e coordinamento alle regioni per la determinazione dei livelli di assistenza sanitaria di emergenza

(G.U. 31 marzo 1992, n.76, serie generale)

IL PRESIDENTE DELLA REPUBBLICA

Visto l'art. 4 della legge 30 dicembre 1991, n. 412, che detta norme in materia di assistenza sanitaria per l'anno 1992;

Visto il comma 1 della richiamata norma che autorizza il Governo ad emanare un atto di indirizzo e di coordinamento per la determinazione dei livelli di assistenza sanitaria da assicurare in condizioni di uniformità sul territorio nazionale sulla base dei limiti e principi di cui alle successive lettere a), b), c), d) ed e);

Vista la deliberazione del CIPE in data 3 agosto 1990 che ha disciplinato, su conforme parere della Conferenza permanente per i rapporti tra lo Stato, le regioni e le province autonome, le priorità degli interventi relativi all'emergenza-urgenza sanitaria ed al rischio anestesiologico anche utilizzando con vincolo di destinazione le risorse in conto capitale del Fondo sanitario nazionale;

Visto l'art. 22 dell'accordo collettivo nazionale per la regolamentazione dei rapporti con i medici addetti al servizio di guardia medica e di emergenza territoriale, reso esecutivo con decreto del Presidente della Repubblica 25 gennaio 1991, n. 41;

Visto il documento tecnico di intesa approvato dalla Conferenza Stato-regioni nella seduta del 14 gennaio 1992;

Visto il parere espresso dal Consiglio superiore di sanità in data 12 febbraio 1992;

Ritenuto che, nelle more della definizione degli standard organizzativi e dei costi unitari dei livelli di assistenza uniformi di cui all'art. 4 della legge 30 dicembre 1991, n. 412, la Conferenza Stato-regioni in data 7 febbraio 1992 ha definito l'intesa sul livello uniforme di assistenza del sistema dell'emergenza sanitaria;

Ritenuto che le spese in conto capitale per l'organizzazione del livello assistenziale fanno carico agli stanziamenti di cui all'art. 20 della legge 11 marzo 1988, n. 67, nonché agli

stanziamenti in conto capitale del Fondo sanitario nazionale, mentre quelle correnti fanno carico al Fondo sanitario nazionale di parte corrente di cui all'art. 51 della legge 23 dicembre 1978, ne; 833, nella misura che sarà determinata ai sensi del combinato disposto della norma di cui ai commi 1 e 16 dell'art. 4 della legge 30 dicembre 1991, n. 412;

Vista la deliberazione del Consiglio dei Ministri, adottata nella riunione del 13 marzo 1992, su proposta del Ministro della sanità, di concerto con il Ministro per le riforme istituzionali e gli affari regionali;

DECRETA:

È approvato il seguente atto di indirizzo e coordinamento delle attività delle regioni e delle province autonome di Trento e di Bolzano, in materia di emergenza sanitaria.

Art. 1 Il livello assistenziale di emergenza sanitaria

1. Ai sensi del comma 1 dell'art. 4 della legge 30 dicembre 1991, n. 412, il livello assistenziale di emergenza sanitaria da assicurare con carattere di uniformità in tutto il territorio nazionale è costituito dal complesso dei servizi e delle prestazioni di cui agli articoli successivi.

Art. 2 Il sistema di emergenza sanitaria

1. Le regioni e le province autonome di Trento e di Bolzano organizzano le attività di urgenza e di emergenza sanitaria articolate su:

a. il sistema di allarme sanitario;
b. il sistema di accettazione e di emergenza sanitaria.

Art. 3 Il sistema di allarme sanitario

1. Il sistema di allarme sanitario è assicurato dalla centrale operativa, cui fa riferimento il numero unico telefonico nazionale "118". Alla centrale operativa affluiscono tutte le richieste di intervento per emergenza sanitaria. La centrale operativa garantisce il coordinamento di tutti gli interventi nell'ambito territoriale di riferimento.

2. Le centrali operative della rete regionale devono essere compatibili tra loro e con quelle delle altre regioni e delle province autonome di Trento e di Bolzano in termini di standard telefonici di comunicazione e di servizi per consentire la gestione del traffico

interregionale. Con decreto del Ministro della sanità, di concerto con il Ministro delle poste e delle telecomunicazioni, entro sessanta giorni dalla data di pubblicazione del presente atto nella Gazzetta Ufficiale della Repubblica, sono definiti gli standard di comunicazione e di servizio.

3. L'attivazione della centrale operativa comporta il superamento degli altri numeri di emergenza sanitaria di enti, associazioni e servizi delle unità sanitarie locali nell'ambito territoriale di riferimento, anche mediante convogliamento automatico delle chiamate sulla centrale operativa del "118".

4. Le centrali operative sono organizzate, di norma su base provinciale. In ogni caso nelle aree metropolitane dove possono all'occorrenza sussistere più centrali operative, è necessario assicurare il coordinamento tra di esse.

5. Le centrali operative assicurano i radiocollegamenti con le autoambulanze e gli altri mezzi di soccorso coordinati e con i servizi sanitari del sistema di emergenza sanitaria del territorio di riferimento, su frequenze dedicate e riservate al servizio sanitario nazionale, definite con il decreto di cui al comma 2.

6. Il dimensionamento e i contenuti tecnologici delle centrali operative sono definiti sulla base del documento approvato dalla Conferenza Stato-regioni in data 14 gennaio 1992, che viene allegato al presente atto.

Art. 4 Competenze e responsabilità nelle centrali operative

1. La responsabilità medico-organizzativa della centrale operativa è attribuita nominativamente, anche a rotazione, a un medico ospedaliero con qualifica non inferiore ad aiuto corresponsabile, preferibilmente anestesista, in possesso di documentata esperienza ed operante nella medesima area dell'emergenza.

2. La centrale operativa è attiva per 24 ore al giorno e si avvale di personale infermieristico adeguatamente addestrato, nonché di competenze mediche di appoggio. Queste devono essere immediatamente consultabili e sono assicurate nominativamente anche a rotazione, da medici dipendenti con esperienza nel settore dell'urgenza ed emergenza e da medici del servizio di guardia medica di cui all'art. 22 dell'accordo collettivo nazionale per la regolamentazione dei rapporti con i medici addetti al servizio di guardia medica e di emergenza territoriale, reso esecutivo con decreto del Presidente della Repubblica 25 gennaio 1991, n. 41. La responsabilità operativa è affidata al personale infermieristico

professionale della centrale, nell'ambito dei protocolli decisi dal medico responsabile della centrale operativa.

Art. 5 Disciplina delle attività

1. Gli interventi di emergenza sono classificati con appositi codici. Il Ministro della sanità, con proprio decreto da emanarsi entro sessanta giorni dalla data di pubblicazione del presente atto nella Gazzetta Ufficia/e della Repubblica, stabilisce criteri e requisiti cui debbono attenersi le regioni e le province autonome di Trento e di Bolzano nella definizione di tale codificazione, anche ai fini delle registrazioni necessarie per documentare le attività svolte e i soggetti interessati.

2. L'attività di soccorso sanitario costituisce competenza esclusiva del Servizio sanitario nazionale. Il Governo determina gli standard tipologici e di dotazione dei mezzi di soccorso ed i requisiti professionali del personale di bordo, di intesa con la Conferenza Stato-regioni.

3. Ai fini dell'attività di cui al precedente comma, le regioni e le province autonome di Trento e di Bolzano possono avvalersi del concorso di enti e di associazioni pubbliche e private, in possesso dell'apposita autorizzazione sanitaria, sulla base di uno schema di convenzione definito dalla Conferenza Stato-regioni, su proposta del Ministro della sanità.

Art. 6 Il sistema di accettazione e di emergenza sanitaria

1. Fermo restando quanto previsto dall'art. 14 del decreto del Presidente della Repubblica 27 marzo 1969, n. 128, in materia di accettazione sanitaria, il sistema di emergenza sanitaria assicura:

a. il servizio di pronto soccorso;
b. il dipartimento di emergenza.

2. Le regioni e le province autonome di Trento e di Bolzano individuano gli ospedali sedi di pronto soccorso e di dipartimento di emergenza.

Art. 7 Le funzioni di pronto soccorso

1. L'ospedale sede di pronto soccorso deve assicurare, oltre agli interventi diagnostico-terapeutici di urgenza compatibili con le specialità di cui e dotato, almeno il primo accertamento diagnostico, clinico, strumentale e di laboratorio e gli interventi necessari alla stabilizzazione del paziente, nonché garantire il trasporto protetto.

2. La responsabilità delle attività del pronto soccorso e il collegamento con le specialità di cui è dotato l'ospedale sono attribuiti nominativamente, anche a rotazione non inferiore a sei mesi, ad un medico con qualifica non inferiore ad aiuto, con documentata esperienza nel settore.

Art. 8 Le funzioni del dipartimento di emergenza

1.il dipartimento di emergenza deve assicurare nell'arco delle 24 ore, anche attraverso le unità operative specialistiche di cui è dotato l'ospedale, oltre alle funzioni di pronto soccorso, anche:

a. interventi diagnostico-terapeutici di emergenza medici, chirurgici, ortopedici, ostetrici e pediatrici;
b. osservazione breve, assistenza cardiologica e rianimatoria.

2. Al dipartimento di emergenza sono assicurate le prestazioni analitiche, strumentali e di immunoematologia per l'arco delle 24 ore giornaliere .

3. La responsabilità delle attività del dipartimento e il coordinamento con le unità operative specialistiche di cui è dotato l'ospedale sono attribuiti nominativamente, anche a rotazione non inferiore a sei mesi, ad un primario medico, chirurgo o rianimatore, con documentata esperienza nel settore .

Art. 9 Le funzioni regionali

1. Le regioni e le province autonome di Trento e di Bolzano, anche a stralcio del Piano sanitario regionale, determinano, entro centoventi giorni, dalla data di pubblicazione del presente atto nella Gazzetta Ufficiale della Repubblica, la ristrutturazione del sistema di emergenza sanitaria, con riferimento alle indicazioni del parere tecnico fornito dal Consiglio superiore di sanità, in data 12 febbraio 1991, e determinano le attribuzioni dei responsabili dei servizi che compongono il sistema stesso.

Il provvedimento di cui al comma precedente determina altresì le modalità di accettazione dei ricoveri di elezione in relazione alla esigenza di garantire adeguate disponibilità di posti letto per l'emergenza. Con il medesimo provvedimento sono determinate le dotazioni di posti letto per l'assistenza subintensiva da attribuire alle singole unità operative

Art. 10 Prestazioni del personale infermieristico

1. Il personale infermieristico professionale, nello svolgimento del servizio di emergenza, può essere autorizzato a praticare iniezioni per via endovenosa e fleboclisi, nonché a svolgere le altre attività e manovre atte a salvaguardare le funzioni vitali, previste dai protocolli decisi dal medico responsabile del servizio.

Art. 11 Onere del trasporto di emergenza

1. Gli oneri delle prestazioni di trasporto e soccorso sono a carico del servizio sanitario nazionale solo se il trasporto è disposto dalla centrale operativa e comporta il ricovero del paziente. Detti oneri sono altresì a carico del Servizio sanitario nazionale anche in mancanza di ricovero determinata da accertamenti effettuati al pronto soccorso.

Art. 12 Attuazione

1. All'attuazione di quanto disposto dal presente atto provvedono le regioni e le province autonome.

2. Le spese in conto capitale per l'organizzazione del livello assistenziale fanno carico come priorità agli stanziamenti di cui all art. 20 della legge 11 marzo 1988, n. 67, nonché agli stanziamenti in conto capitale del Fondo sanitario nazionale, mentre quelle correnti fanno carico al Fondo sanitario nazionale di parte corrente di cui all'art. 51 della legge 23 dicembre 1978, n. 833, nella misura che sarà determinata al sensi del combinato disposto delle norme di cui ai commi 1 e 16 dell'art. 4 della legge 30 dicembre 1991, n 412.

3. Entro sei mesi dalla data di pubblicazione del presente atto nella Gazzetta Ufficiale della Repubblica, la Conferenza Stato regioni verifica le iniziative assunte, lo stato di attuazione del sistema emergenza sanitaria in ciascuna regione e provincia autonoma, nonché le risorse finanziarie impiegate. Allo scopo di attuare il sistema di emergenza sanitaria nelle regioni che non lo abbiano attuato, in tutto o in parte la Conferenza Stato-regioni approva uno schema tipo di accordo di programma, che, sottoscritto dal Ministro della sanità e dal Presidente della regione interessata, determina tempi, modi e risorse finanziarie per l'attuazione, anche avvalendosi di apposite conferenze dei servizi. L'accordo di programma può essere attivato anche prima della verifica, su richiesta della regione e provincia autonoma.

Il presente decreto sarà pubblicato nella Gazzetta Ufficiale della Repubblica italiana.

Codici di intervento

Codici di Criticità

0 : Non emergenza; situazione di intervento differibile e/o programmabile.

1 : Non emergenza; situazione differibile ma prioritaria rispetto al codice zero.

2 : Emergenza sanitaria; situazione a rischio, intervento non differibile.

3 : Emergenza assoluta; intervento prioritario.

4 : Paziente deceduto

Codici di Patologia

0 : Etilista.

1 : Trauma.

2 : Cardiocircolatoria.

3 : Respiratoria.

4 : Neurologica.

5 : Psichiatrica.

6 : Neoplastica.

7 : Intossicazione.

8 : Altra patologia.

9 : Non identificata.

Codici di Luogo

S : Strada.

P : Esercizio pubblico.

Y : Impianto sportivo.

K : Casa.

L : Lavoro.

Q : Scuola.

Z : Altro.

Mezzi di soccorso che intervengono su Torino

Le ambulanze che agiscono su Torino (dati aggiornati al 2002) sono:

	POSTAZIONE	***TIPO***	***SIGLA***	***CONVENZIONE***	***ENTE***
1.	Ospedale CTO	MSAB	055	H24	Croce Verde
2.	Croce Verde Torino	MSB	360	Estemporanea	Croce Verde
3.	Croce Verde Torino	MSA	365	H24	Croce Verde
4.	Croce Rossa Torino	MSB	370	Estemporanea	CRI
5.	Ospedale Valletta	MSB	380	H24	Croce Verde
6.	Ospedale Molinette	MSB	390	H24	CRI
7.	Ospedale Molinette	MSA	395	H24	CRI
8.	Ospedale Martini	MSB	400	H24	Croce Verde
9.	Ospedale Martini	MSA	405	H24	Croce Verde
10.	Ospedale Mauriziano	MSB	410	H24	CRI
11.	Ospedale Giovanni Bosco	MSB	420	H24	CRI
12.	Ospedale Giovanni Bosco	MSA	425	H24	CRI
13.	Ospedale Maria Vittoria	MSB	430	H24	Croce Verde
14.	Ospedale Maria Vittoria	MSA	435	H24	Croce Verde
15.	Ospedale Gradenigo	MSB	450	H24	Croce Bianca

Mezzi di soccorso che intervengono sulla provincia di Torino

Le ambulanze che agiscono sulla Provincia di Torino (dati aggiornati al 2002) sono:

	POSTAZIONE	***TIPO***	***SIGLA***	***CONVENZIONE***	***ENTE***
1.	Mathi Canadese	MSB	040	Estemporanea	CRI
2.	Lanzo	MSB	060	H24	CRI
3.	Lanzo	MSAB	065	H24	CRI
4.	Pont Canadese	MSB	080	H24	Gruppo Vol.
5.	Rivarolo	MSB	090	H24	CRI
6.	Castellamonte	MSB	100	H24	CRI
7.	Ivrea	MSB	110	H24	CRI
8.	Ivrea	MSB	112	Estemporanea	Gruppo Vol.
9.	Ivrea	MSA	115	H24	ASL 9
10.	Courgnè	MSB	120	Estemporanea	CRI
11.	Courgnè	MSA	125	H24	CRI
12.	Caluso	MSA	135	H24	Gruppo Vol.
13.	San Giorgio Canavese	MSB	140	Estemporanea	CRI
14.	Agliè	MSB	150	Estemporanea	CRI
15.	Caravino	MSB	160	Estemporanea	VASC
16.	Balme	MSB	170	Estemporanea	CRI
17.	Ala di Stura	MSB	180	Estemporanea	CRI
18.	Usseglio	MSB	190	Estemporanea	CRI
19.	Chivasso	MSA	205	H24	CRI
20.	Corio	MSB	210	H24	CRI
21.	Viù	MSB	220	H24	CRI
22.	Leinì	MSB	230	Estemporanea	CRI
23.	Ciriè	MSB	240	Estemporanea	Croce Verde
24.	Ciriè	MSAB	245	H24	Croce Verde
25.	San Francesco al campo	MSB	250	Estemporanea	CRI
26.	Venaria Reale	MSB	260	H24	Croce Verde
27.	Caselle	MSB	270	H24	Croce Verde
28.	Settimo Torinese	MSB	280	Estemporanea	CRI
29.	Settimo Torinese	MSA	285	H24	CRI

30.	Beinasco	MSB	290	H24	CRI
31.	Gassino Torinese	MSB	300	H24	CRI
32.	Volpiano	MSB	310	H24	Croce Bianca
33.	Piossasco	MSB	320	Estemporanea	CRI
34.	Nichelino	MSB	330	Estemporanea	CRI
35.	Nichelino	MSAB	335	H24	CRI
36.	Orbassano	MSB	340	Estemporanea	Croce Bianca
37.	Orbassano	MSAB	345	H24	Croce Bianca
38.	Giaveno	MSB	350	Estemporanea	CRI
39.	Giaveno	MSA	355	H24	CRI
40.	Rivalta	MSB	440	Estemporanea	ANPAS
41.	Carmagnola	MSB	460	Estemporanea	CRI
42.	Carmagnola	MSA	465	H24	CRI
43.	Moncalieri	MSB	470	Estemporanea	CRI
44.	Moncalieri	MSAB	475	H24	CRI
45.	Villadora	MSB	480	Estemporanea	CRI
46.	Avigliana	MSA	485	H24	CRI
47.	Trofarello	MSB	490	Estemporanea	CRI
48.	Carignano	MSB	500	Estemporanea	CRI
49.	Collegno	MSB	510	Estemporanea	Misericordia
50.	Susa	MSB	520	H24	CRI
51.	Susa	MSA	525	H24	CRI
52.	Bardonecchia	MSB	530	Estemporanea	CRI
53.	Poirino	MSB	540	H24	CRI
54.	Rivoli	MSB	550	Estemporanea	CRI
55.	Rivoli	MSAB	555	H24	CRI
56.	Chieri	MSB	560	Estemporanea	CRI
57.	Chieri	MSA	565	H24	CRI
58.	Santena	MSB	570	Estemporanea	CRI
59.	Vinovo	MSB	580	H24	ANPAS
60.	None	MSB	590	Estemporanea	ANPAS
61.	Pecetto Torinese	MSB	600	Estemporanea	CRI
62.	Alpignano	MSB	610	H24	ANPAS
63.	Crescentino	MSB	620	Estemporanea	CRI
64.	Lauriano Po	MSB	630	Estemporanea	CRI
65.	Verolengo	MSB	640	Estemporanea	ANPAS

66.	Chivasso	MSB	650	Estemporanea	CRI
67.	Chivasso	MSA	655	H24	CRI
68.	Vignone	MSB	660	H24	CRI
69.	Torre Pellice	MSB	670	Estemporanea	CRI
70.	Torre Pellice	MSAB	675	H24	CRI
71.	San Mauro	MSB	680	Estemporanea	ANPAS
72.	Prali	MSB	690	Estemporanea	ANPAS
73.	Strambino	MSB	700	Estemporanea	CRI
74.	Castagnole Piemonte	MSB	710	Estemporanea	CRI
75.	Castelnuovo Don Bosco	MSB	720	Estemporanea	CRI
76.	Airasca	MSB	730	Estemporanea	CRI
77.	Cumiana	MSB	740	H24	ANPAS
78.	Cavour	MSB	750	H24	ANPAS
79.	Bricherasio	MSB	760	H24	ANPAS
80.	Perosa Argentina	MSB	770	Estemporanea	ANPAS
81.	Perosa Argentina	MSAB	775	H24	ANPAS
82.	Ponte	MSB	780	H24	ANPAS
83.	Pinerolo	MSB	790	H24	ANPAS
84.	Pinerolo	MSA	795	H24	ANPAS
85.	Pragelato	MSB	800	Estemporanea	ANPAS
86.	Pragelato	MSA	805	H24	ANPAS
87.	Ulzio	MSA	815	H24	CRI
88.	Bessolo	MSB	820	Estemporanea	Croce Verde
89.	Settimio Vittone	MSB	830	Estemporanea	CRI
90.	Voliera	MSB	840	Estemporanea	ANPAS
91.	Cascine Vica	MSB	850	H24	ANPAS
92.	Druento	MSB	860	Estemporanea	CRI
93.	La Loggia	MSB	870	Estemporanea	CRI
94.	Fiano	MSB	880	Estemporanea	CRI
95.	Pino Torinese	MSB	890	Estemporanea	CRI
96.	Grugliasco	MSAB	905	H24	ANPAS
97.	San Giusto Canavese	MSB	910	Estemporanea	ANPAS
98.	Valperga	MSB	920	Estemporanea	Croce Bianca
99.	Ceresole Reale	MSB	930	Estemporanea	ANPAS

Friedrich August Von Hayek[1]

La vita e le opere

Friderich August Von Hayek nacque a Vienna l'8 maggio 1899.

I suoi studi universitari si svolsero a Vienna dove si laureò alla facoltà di legge nel novembre del 1921. Durante il periodo delle lezioni frequentò le lezioni dell'economista Friedrich von Wieser e del filosofo Otmar Spann. Nel 1923 prese una seconda laurea in scienze politiche.

Quando il periodo accademico terminò partì per l'america dove intraprese ancora un periodo di formazione frequentando le lezioni di Wesley C. Mitchell sulla storia del pensiero economico e il seminario di John B. Clark presso Columbia University di New York.

Tornato a Vienna, nel 1924 fondò con Ludwig Von Mises l'Istituto austriaco per lo studio dei cicli monetari e nel 1929 pubblicò la sua prima opera, in quello stesso anno gli venne affidata la docenza in economia politica.

Nel 1931 si recò a Londra su invito di Lionel Robbins e tenne quattro lezioni alla London School Of Economics su "*Prezzi e produzione*". Dopo questa prima esperienza a Londra vi si trasferì e ci rimase per 18 anni, proponendosi come antagonista di John Maynard Keynes.

Hayek provò da subito interesse verso K. Popper fin dalla pubblicazione della sua prima opera che prese il titolo di "*La logica della scoperta scientifica"*. Conobbe Popper personalmente e con lui e Robbins partecipò ad un seminario alla London School of Economics invitando lo stesso Popper a leggere il suo scritto *"Miseria dello storicismo"*.

Nel 1937 Hayek pubblicò il saggio "*Economics and Knowledge"*. Quando scoppiò la II guerra mondiale iniziò la stesura di un testo, ironicamente "dedicato ai miei amici socialisti", che intitolò *"The Road to Serfdom"*, tradotto e pubblicato in italiano da Rizzoli con il titolo *"La via verso la schiavitù"*. Il libro ebbe un notevole successo nonostante le critiche feroci che gli furono mosse soprattutto ad opera di Guido Marenco che afferma:

"Una volta tanto concordo con lui [dove con lui intende l'economista Mill] *perché mi pare evidente che elementi fondanti del nazismo quali l'antisemitismo e la teorizzazione*

[1] Quanto scritto in questa appendice è tratto da due siti web, la vita e le opere è tratto dal sito http://digilander.libero.it/moses/hajek1.html. Il testo è riportato quasi integralmente ma con qualche modifica.
La parte inerente alla filosofia è invece tratta dal sito http://www.filosofico.net/hayek.htm ed è riportata fedelmente

della supremazia della razza non hanno nulla di socialista e nemmeno di cristiano. Come del resto hanno poco a che fare anche con l'evoluzionismo di Herbert Spencer."

Hayek intraprese anche una strenua battaglia anche contro lo *scientismo* che considerò come la pretesa di applicare il metodo delle scienze naturali anche alla sfera delle scienze sociali rintracciando le origini di questo atteggiamento nei lavori di Comte e Saint-Simon e nel Positivismo.

Tutti questi studi andarono a finire nel volume *"L'abuso della ragione. Studi sulla controrivoluzione nella scienza"*, edito in Italia da Vallecchi nel 1987.

Nel 1945 uscì *"L'uso della conoscenza nella società"*, sulla rivista "American Economic Review".
Successivamente fondò, insieme a Von Mises, Milton Friedman e Karl Popper la *Mont Pélerin Society*, un gruppo di circa 400 membri, la cui principale attività era quella di organizzare seminari periodici "al fine di scambiarsi idee sulla natura di una società libera e sui mezzi per rafforzare la sua difesa intellettuale." Di essa von Hajek fu presidente fino al 1960, e presidente onorario fino alla morte (1992).
Nel 1949 lasciò la London School of Economics per trasferirsi negli Stati Uniti dove gli fu conferita la cattedra di scienze sociali e morali all'Università di Chicago e dove ebbe modo di lavorare a più stretto contatto con Milton Friedman e Frank Knight.

Nel 1952 uscì un suo studio di teoria psicologica che affrontava il rapporto tra mente e cervello *"The Sensory Order"* poi tradotto in italiano ed edito in Italia dalla casa editrice Rusconi nel 1990 con il titolo *"L'ordine sensoriale"*.

Nel 1954 partecipò alla stesura del volume "*Capitalism and Historians"* scritto da vari autori ed edito da Routledge and Kegan Paul a Londra (tradotto successivamente in Italia dall'editore Sansoni). Di questo volume Von Hayek scrisse un'introduzione intitolata *"Hystory and Politics"* nella quale attaccava in modo duro il comunismo. Oggetto della sua critica fu soprattutto la secondo la quale la classe operaia avrebbe peggiorato la propria condizione con il sorgere del capitalismo.

Nel 1960 uscì *"The Constitution of Liberty"* edito in Italia nel 1969 con il titolo *"La società libera"* da Vallecchi.
Nel 1962 tornò in Europa, succedendo all'amico Walter Eucken alla cattedra di Economia politica all'Università di Friburgo. È questo il periodo in cui pubblica nuovi scritti tra i quali è bene ricordare *"Il sistema concorrenziale quale strumento di conoscenza"* e *"Nuovi studi di filosofia , politica, economia e storia delle idee"*.

undici anni dopo il suo ritorno in Europa esce il primo volume della sua opera *"Law, Legislation and Liberty"* intitolato uscito in Italia con il titolo *"Regole ed ordine"* ed edito da Saggiatore nel 1986. Seguiranno gli altri due volumi editi con i titoli *"Il miraggio della giustizia sociale"* e *"Il sistema politico di un popolo libero"*.

"In questa sua grandiosa opera Hayek distingue tra ordine spontaneo ed organizzazione precostruita: mostra come il costruttivismo razionalistico distrugga l'ordine spontaneo e con ciò le difese della libertà individuale; fa vedere che l'idea di giustizia sociale è un residuo di attivismo carico di pericoli per la nostra civiltà; mette il dito sulla piaga di quell'onnipotenza dei parlamenti delle attuali democrazie, su quel potere illimitato che le ha trasformate in tirannie; e propone un ordine politico di un popolo libero".

Paradosso forse non casuale volle che nel 1974 egli ricevesse il premio Nobel per l'economia condiviso con uno dei suoi grandi "avversari", lo svedese Gunnar Myrdal, ovvero uno dei teorici dello stato sociale.
Il suo ultimo lavoro fu *"The Fatal Conceit. The Errors of Socialism"*, tradotto in italiano come *"La presunzione finale"* ed edito da Rusconi. In questo lavoro criticò il socialismo come erede della tradizione razionalistica costruttivista e denunciò il suo totale fallimento.

Friedrich August von Hajek morì a Friburgo nel marzo del 1992.

La filosofia

Friedrich August von Hayek (Vienna 1899 – Friburgo 1992) è uno dei più grandi esponenti del neoliberalismo novecentesco ed è uno dei maggiori critici dell'economia pianificata e centralista […]

La filosofia politica di Hayek è interamente costruita sull'ideale di libertà individuale e sulla stretta connessione – come ha rilevato Norberto Bobbio – tra **libertà economica** e **libertà senza altri aggettivi**. La libertà è sempre una condizione che riguarda la persona in quanto individuo, equipaggiato di una sfera privata attorno a sé che gli altri non possono valicare. La libertà è allora essenzialmente assenza di interferenza o di coercizione esterne. Quando l'uomo è costretto a seguire dei fini impostigli dagli altri e non dal proprio libero esercizio intellettuale, ecco che allora si riduce a uno stato di schiavitù. In tale prospettiva, Hayek mette in luce come anche chi vivesse negli agi e nell'opulenza (ad esempio, un cortigiano) o in mezzo a un popolo che partecipa alle scelte del proprio governo (come nei regimi democratici novecenteschi) non per questo deve credersi libero. Da ciò appare comprensibile la concezione "negativa" che Hayek ha della libertà, intesa come assenza di costrizione esterna: in ciò, egli è in perfetta sintonia più con la tradizione liberale inglese del Settecento (*in primis* con Locke) che con quella continentale europea (Kant innanzitutto). Ciò che più interessa ad Hayek è dunque la libertà concepita come protezione mediante la legge contro ogni forma di coercizione arbitraria (*freedom from*) e non come rivendicazione del diritto di ognuno di partecipare alla determinazione della forma di governo (*freedom to*). In tale impostazione, acquista grande rilievo il discorso sullo **Stato**, che deve avere essenzialmente un ruolo secondario e negativo, deve intervenire il meno possibile nell'ambito di autonomia individuale e deve garantire, grazie a leggi

generali, il pieno dispiegarsi delle libertà individuali, assicurando solide barriere a difesa dei "territori" dei singoli individui. La proprietà privata, intesa lockeanamente come diritto alla "*vita, alla libertà e ai beni*", è, di conseguenza, il fondamento di ogni civiltà evoluta. A tal proposito, Hayek scrive che essa

> "*è la sola soluzione finora scoperta dagli uomini per risolvere il problema di conciliare la libertà individuale con l'assenza di conflitti. Legge, libertà, proprietà sono una trinità inseparabile. Non vi può essere alcuna legge, nel senso di regola universale di condotta, che non determini confini di aree d'azione, stabilendo regole che permettono a ciascuno di accertare fin dove egli è libero di agire*" (*Law, Legislation and Liberty*).

Come risulta da tali affermazioni, lo Stato dev'essere esso stesso soggetto alla legge, che è l'unica garanzia della libertà individuale. In tema di legge, Hayek distingue innanzitutto la "**legge**" (avente carattere generale e universale) dalla "**legislazione**" concreta, riguardante le singole norme o i comandi che perseguono fini specifici e interessi di gruppo. A suo avviso, un esempio di legge generale è il divieto di uccidere un altro individuo, mentre il comando di non uccidere in un ben determinato caso ha a che fare con la legislazione di uno Stato. Altri esempi di leggi astratte e generali sono quelle messe in luce, a suo tempo, da Hume: la stabilità del possesso dei beni, la cessione per comune consenso, il mantener fede alle promesse. Hayek sottolinea come la legge non riguardi casi individuali, mentre la legislazione si componga di provvedimenti amministrativi voluti dalla maggioranza parlamentare per fini particolari o, più spesso ancora (soprattutto nelle democrazie moderne), per fini elettorali. È un gravissimo errore – nota Hayek – identificare la legge con la legislazione, come spesso si fa: infatti, la legislazione dipende dal governo, mentre la legge è da esso svincolata e, anzi, rappresenta la norma che ogni governo deve osservare. Sulle orme di Locke, Hayek nutre la convinzione che, dove finisce la legge, là inizia la tirannide, con l'inevitabile conseguenza che la sfera legislativa dei governi dev'essere limitata dal "***governo della legge***" (*rule of law*):

> "*L'imperio della legge […] comporta dei limiti al campo della legislazione; esso lo restringe a quel tipo di regole generali cui si tributa il nome di leggi formali ed esclude la legislazione che miri direttamente a persone determinate o che metta in grado qualcuno di usare il potere*

coercitivo dello Stato ai fini di una tale discriminazione. Esso non significa che tutto deve essere regolato dalla legge, ma significa all'opposto che il potere coercitivo dello Stato può essere usato soltanto in casi anticipatamente definiti dalla legge e in maniera tale che si possa prevedere come sarà impiegato" (*Verso la schiavitù*).

Da ciò risulta che Hayek pensa che un governo possa intervenire legittimamente nella vita dei suoi cittadini soltanto per far rispettare le norme generali, ossia le norme che servono a proteggere "*la vita, la libertà, i beni*". Lo Stato diventa coercitivo nella misura in cui interferisce in qualche modo con la libertà degli individui di perseguire i propri scopi e di realizzare i propri personali piani di vita:

"*Ciò che distingue radicalmente le condizioni di un paese libero da quelle di un paese sottoposto a un governo arbitrario è il fatto che nel primo si osserva il grande principio denominato l'imperio della legge. Spogliato da ogni tecnicismo, esso significa che il governo, in tutte le sue azioni, è vincolato da regole fisse e annunziate in anticipo, regole che danno la possibilità di prevedere con ragionevole sicurezza in qual modo l'autorità userà i suoi poteri coercitivi in determinate circostanze, e di indirizzare i propri affari individuali sulla base di tale cognizione*" (*Verso la schiavitù*).

Una delle forme più diffuse di interferenza è sicuramente la legislazione in materia di **giustizia sociale**, la quale tende a modificare la posizione economico/sociale delle persone favorendo (ad esempio attraverso la tassazione) le persone meno agiate. Su questa tematica, la posizione di Hayek è assai drastica: le persone svantaggiate (i poveri, gli ammalati, i portatori di handicap, le vedove, gli orfani, ecc) debbono essere protetti da una "rete" che assicuri loro il minimo necessario alla sopravvivenza, ma ciò deve avvenire al di fuori del libero mercato e non come intervento correttivo del mercato da parte della legislazione. Assicurare un reddito minimo a tutti è, secondo Hayek, un dovere della società libera: ma ciò deve verificarsi tramite l'assistenza e non cambiando in modo artificiale le regole del mercato. Tra i vari compiti dello Stato, spicca quello di costruire strade, fissare indici di misura, di fornire altri tipi di informazioni (attraverso mappe e cartelli stradali, ad esempio) e di controllare il controllo sulla qualità dei beni e dei servizi. Ma riguardo ad altri servizi, come ad esempio quello postale, quello dell'istruzione e delle telecomunicazioni, il

monopolio dello Stato è pernicioso oltre che inefficiente. Da questa posizione, ben emerge l'immensa **fiducia nel libero mercato** che, pur non funzionando sempre in modo perfetto, presenta benefici che superano di gran lunga gli svantaggi. Indubbiamente suggestionato dalla "*mano invisibile*" di cui parlava Adam Smith, Hayek è convinto che il mercato riesca ad armonizzare in maniera spontanea le decisioni dei produttori con la volontà e coi desideri dei consumatori, senza la mediazione del governo, e che assicuri il perseguimento dei propri scopi a tutti, sviluppando altresì quella che Hayek chiama la "**Grande Società**", cioè la moderna *società complessa*, che sfugge a ogni pianificazione centralizzata poiché si affida solo all'iniziativa individuale e al meccanismo della concorrenza. La politica, concepita come sistema decisionale e governativo, resta allora sempre e comunque un metodo imperfetto rispetto al libero mercato. Queste idee implicano ovviamente una **riconsiderazione della democrazia**, la quale agli occhi di Hayek dev'essere esplicitamente condannata quando diventa "*governo della maggioranza dotato di potere illimitato*" (*Law, Legislation and Liberty*). In una siffatta prospettiva, il filosofo viennese propone di sostituire il termine "democrazia" (che significa "potere del popolo", dal greco □□□□□□ e □□□□□) con "**demarchia**". Infatti, il verbo greco □□□□□□□ (avere potere, dominare) – nota Hayek - "*al contrario del verbo alternativo □□□□□□ (usato nei composti quali monarchia, oligarchia, ecc) sembra sottolineare la forza bruta, piuttosto che il governare secondo regole*". L'espressione "democrazia" deve essere allora sostituita da quella "demarchia":

> "*Questo sarebbe il nuovo nome di cui ha bisogno, se si vuole preservare l'ideale alla sua radice, in un'epoca in cui, dato il crescente abuso del termine democrazia per designare sistemi che tendono alla creazione di nuovi privilegi attraverso coalizioni o interessi organizzati, un numero sempre crescente di persone si allontana dal sistema prevalente […]. Se tale reazione giustificata contro l'abuso del termine non si vuole che porti a discreditare l'ideale stesso, e a far accettare alla gente disillusa forme di governo molto meno desiderabili, sembra necessario avere un nuovo termine come 'demarchia' che descriva l'antico ideale con un nome non macchiato da un lungo abuso*" (*Law, Legislation and Liberty*).

E Hayek si propone anche di indicare gli organi costituzionali che questa demarchia dovrebbe avere:

a) un'**Assemblea legislativa**, composta da uomini e donne tra i 45 e i 60 anni (ossia dalle persone più esperte) che restino in carica per quindici anni, col compito di assicurare il quadro generale delle libertà individuali, impedendo ogni forma di coercizione arbitraria sulla sfera privata;

b) un'**Assemblea governativa**, che corrisponda *lato sensu* ai parlamenti, i cui membri (suddivisi in partiti) siano eletti periodicamente col fine di occuparsi degli interessi particolari.

La fiducia smisurata di Hayek nel libero mercato si coniuga con un'incessante polemica **contro l'economia pianificata** e dirigista del comunismo e **contro l'eccesso di interventismo** del *Welfare State*. Tale polemica si sostanzia non solo di motivazioni di ordine economico o della persuasione dell'inesistenza di una mente collettiva in grado di possedere tutte le conoscenze necessarie per la regolamentazione di una società complessa, ma anche di ragioni etiche, politiche ed esistenziali:

"*Il controllo economico non è il semplice controllo di un settore della vita umana che possa essere separato dal resto; è il controllo dei mezzi per tutti i nostri fini. E chiunque abbia il controllo dei mezzi deve anche determinare quali fini debbano essere alimentati, quali valori vadano stimati […] in breve, ciò che gli uomini debbano credere e ciò per cui debbano affannarsi*" (*Verso la schiavitù*).

Deriva di qui il nesso imprescindibile fra liberalismo politico e liberismo economico: i due liberalismi sono strutturalmente uniti e ogni distinzione fra essi dev'essere – nota Hayek – respinta senza mezzi termini.

OrderDistiller.java

```
//OrderDistiller.java modified by Pietro Terna (look below at rows signed //pp)
```

```
//OrderDistiller.java modified by Marco Lamieri and Francesco Merlo (look below at rows
signed //lm)
import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;
import swarm.collections.ListImpl;

/**
 * This class is used to read data from two worksheets.<br />
 * The first one contains the list of recipes of our virtual enterprise.<br/>
 * The second one contains a sequence of orders to be launched,
 * shift by shift, in order to make the daily production activities.
 *
 * @author Cristian Barreca, Elena Bonessa, Antonella Borra. Modified by: Marco
Lamieri,
 *  Francesco Merlo. Refactored by Francesco Merlo
 *
 */
public class OrderDistiller extends OrderGenerator
{
	String
		semicolon = ";",
		gate = "#",
		layer="l",
		computation="c",
		orderSequences1 = "recipeData/orderStartingSequence.xls", //The first
worksheet of orders
		orderSequences2 = "recipeData/orderSequence.xls", //The second
worksheet of orders
		recipeFile = "recipeData/recipes.xls", //The worksheet of recipes
		currentOrderSequenceWorksheet;
	int
		orderCount = 0,
		currentLayer = 0;
	boolean
		firstTime = true,
		orderSequenceWorksheetOpen = false;
	int[]
		currentOrder;
	ExcelReader
		orderSequenceWorksheet,
		recipeWorksheet;
	AssigningTool
		assigningTool;//pt
	Recipe
		aRecipe;
	Unit
		aUnit;
	Order
		anOrder;
	ListImpl
		unitList,
		endUnitList,
		orderList,
		recipeList;
```

```
	// CONSTRUCTOR
	public OrderDistiller (Zone aZone, int msn, int msl, ListImpl ul,
		ListImpl eul, ListImpl ol, int tln, ESFrameModelSwarm mo,
AssigningTool at)
	{ //pt
		super(aZone, msn, msl, ul, eul, ol, tln, mo, at);

		unitList=ul;
		endUnitList=eul;
		orderList=ol;
		assigningTool=at; //pt
	}

	/**This is the method containing the iterator needed to launch the daily production
of recipes.<br />
	* It take a look at the orderSequence arrays to determine which recipes must be
done and how many times. <br />
	* A request for which unit can do the first production phase of each recipe will be
done to units or endUnits.
	*/
	public void distill()
	{
		if(firstTime == true)
		{
			currentOrderSequenceWorksheet = orderSequences1;

			firstTime = false;
		}
		else
			currentOrderSequenceWorksheet = orderSequences2;

		if(!orderSequenceWorksheetOpen)
		{
			orderSequenceWorksheet = new
ExcelReader(currentOrderSequenceWorksheet);
			orderSequenceWorksheetOpen = true;
		}

		checkForComment(); //see below
		//Getting the number of the shift
		int shiftNumber = orderSequenceWorksheet.getIfInteger();

		boolean isEndOfShift = false;

		while(!isEndOfShift)
		{
			if(orderSequenceWorksheet.checkForLabelCell())
			{
				String aString =
orderSequenceWorksheet.getStrValue();
				if(aString.equals(semicolon))
				{
					isEndOfShift = true;
					break;
```

```
                        }
                        else
                        {
                                orderSequenceWorksheet.goBack();
                        }
                }

                //Reading the order from the worksheet
                readOrderFrom(orderSequenceWorksheet);//see below

                //Checking if the recipe cose exists
                boolean recipeCodeFound = false;
                for (int r = 0; r < recipeList.getCount() && !recipeCodeFound;
r++)
                {
                        aRecipe = (Recipe) recipeList.atOffset(r);
                        if(aRecipe.getRecipeCode() == currentOrder[0])
                                recipeCodeFound = true;
                }

                if(recipeCodeFound == false)
                {
                        System.err.println("OrderDistiller error: There is no
Receipe with code '" + currentOrder[0] +
                                        "' check '" + recipeFile + "' and '" +
currentOrderSequenceWorksheet + "'");
                        MyExit.exit(1);
                }

                for(int q = 0; q < currentOrder[1]; q++)
                {
                        orderCount++;

                        anOrder = new Order(getZone(), orderCount,
Globals.env.getCurrentTime(),
                                aRecipe.getLength(),
aRecipe.getRecipeSteps(),eSFrameModelSwarm, endUnitList);

                        anOrder.setRecipeName(aRecipe.getRecipeName());
                        // setting the layer (from 0 to totalLayerNumber-1)
                        if(currentLayer <0 || currentLayer >=
totalLayerNumber)
                          {
                                System.out.println(
                                "A layer in the sequence is not inclued\n"+
                                "in the interval from 0 to totalLayerNumber-
1");
                                MyExit.exit(1);
                          }
                        anOrder.setOrderLayer(currentLayer);
                        // add the active orders to the general order list (they
will be
                        // eliminated when dropped in a unit, being finished);
this
```

```
                                        // list has been introduced for accounting purposes [may
be it would
                                        // be better substitute it with a get to the units to know
their
                                        // waiting lists]
                                        orderList.addLast(anOrder);
                                        // sending the order to the first production unit (we are
acting as the Front End of the ES)
                                        assigningTool.assign(anOrder);

                                        if(StartESFrame.verbose)
                                        {
                                                System.out.println("OrderDistiller: Order #" +
anOrder.getOrderNumber() +
                                                        " with Name '" +
anOrder.getRecipeName() +
                                                        "' and layerNumber '" +
anOrder.getOrderLayer() + "' is starting production");
                                        }
                                }
                        }
                        if(orderSequenceWorksheet.eof())
                                orderSequenceWorksheetOpen = false;
                }
                /**
                * This method is used to collect the names of the units and of the end units, so that
it can operate the check of
                * corrispondency between the production phases required by recipes and the
phases of production the units can do.
                */
                public void setDictionary()
                {
                        super.setDictionary();
                        //The List of Recipes
                        recipeList = new ListImpl(getZone());

                        recipeWorksheet = new ExcelReader(recipeFile);

                        while(! recipeWorksheet.eof())
                        {
                                        aRecipe = new Recipe(getZone());
                                        aRecipe.setRecipeFrom(recipeWorksheet);
                                        recipeList.addLast(aRecipe);
                        }

                        if(StartESFrame.verbose)
                        {
                                System.out.println("OrderDistiller: " + recipeList.getCount() +
"recipes readed.");
                                for (int i = 0; i < recipeList.getCount(); i++)
                                {
                                        aRecipe = (Recipe) recipeList.atOffset(i);
                                        System.out.println("Recipe #" + i + " with name '" +
aRecipe.getRecipeName() +
                                                "' and code '" + aRecipe.getRecipeCode() +"'");
```

```
                }
            }
        }

        // *****************
        // PRIVATE FUNCTIONS
        // *****************
        private int[] readOrderFrom(ExcelReader e)
        {
            orderSequenceWorksheet = e;
            currentOrder = new int[2];

            checkForComment();
            checkForLayer();
            currentOrder[0] = orderSequenceWorksheet.getIfInteger();
            orderSequenceWorksheet.getIfString();
            currentOrder[1] = orderSequenceWorksheet.getIfInteger();

            return currentOrder;
        }

        private void  checkForLayer()
        {
            if(orderSequenceWorksheet.checkForLabelCell())
            {
                String isLayer = orderSequenceWorksheet.getStrValue();

                if(isLayer.equals(layer))
                {
                    currentLayer = orderSequenceWorksheet.getIntValue();

                    if(StartESFrame.verbose)
                        System.out.println("OrderDistiller: Current
Layer is #" + currentLayer);
                }
                else
                {
                    orderSequenceWorksheet.goBack();
                }
            }
        }

        private void checkForComment()
        {
            if(orderSequenceWorksheet.checkForLabelCell())
            {
                String isGate = orderSequenceWorksheet.getStrValue();
                if(isGate.equals(gate))
                {
                    if(StartESFrame.verbose)
                    {
                        System.out.println("OrderDistiller: there is a
comment ...");
                        System.out.print("# ");
                    }
```

```
                                        while(!orderSequenceWorksheet.eol())
                                        {
                                                String comment =
orderSequenceWorksheet.getStrValue();
                                                if(StartESFrame.verbose)
                                                        System.out.print(" " + comment + "
|");
                                        }
                                        String comment =
orderSequenceWorksheet.getStrValue();
                                        if(StartESFrame.verbose)
                                                System.out.println(" " + comment + " #");

                                        //Checking for other lines of comments
                                        checkForComment();
                                }
                                else
                                {
                                        orderSequenceWorksheet.goBack();
                                }
                        }
                }

} // OrderDistiller
```

Jesframe.scm

```
(list
 (
  cons 'eSFrameObserverSwarm
      (
           make-instance 'ESFrameObserverSwarm
                        #:displayFrequency       1
                        #:verboseChoice         #f
                        #:printMatrixes         #t
                  #:checkMemorySize       #f
                  #:unitHistogramXPos     10
                        #:unitHistogramYPos    300
                        #:endUnitHistogramXPos  10
                        #:endUnitHistogramYPos 250
                        #:timeToFinish 0
                        )
      )

 (
  cons 'eSFrameModelSwarm
      (
           make-instance 'ESFrameModelSwarm
                        #:useOrderDistiller  #t
                        #:ticksInATimeUnit    1
                        #:totalUnitNumber    102
                        #:totalEndUnitNumber  0
                        #:totalLayerNumber    1
                        #:totalMemoryMatrixNumber 4
                        #:maxStepNumber      30
                        #:maxStepLength      1
                        #:sameStepLifoAssignment #t
                        #:assignEqualStepsToSameUnit #t
                        #:compareDisregardingUnits #f
                        #:maxTickInAUnit      0
                  #:useWarehouses      #f
                        #:useNewses         #f
                  #:maxInWarehouses    10
                  #:minInWarehouses     3
                        #:infDeepness        30
                  #:inventoryFinancialRate  0
                        #:inventoryEvaluationCriterion 1
                  #:revenuePerEachRecipeStep  0
                  #:revenuePerCostUnit  0
                        #:nOfNewsesToProduce  1
                        #:nOfNewsesToBeCleared 100000
                        #:nOfOrdersInNewses   1
                        #:orCriterion         1
                        #:orMemoryMatrix      2
                        #:unitCriterion       2
                  #:distillerMultiplicity 1
                              #:noAccountingInFirstTimeUnit #f
             )
      )

 )
```

; a comment ticks 3600->2880; units 84->97; maxStepUnmber 0->1

ESFrameObserverSwarm.java

// ESFrameObserverSwarm.java

```
import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;

import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;

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

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

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

import swarm.collections.ListImpl;

import java.util.*;


/**
 * The Observer contains the Model and the graphic widgets
 * @author  Pietro Terna and several of his students
 */
public class ESFrameObserverSwarm extends GUISwarmImpl {

   /** update frequency */
   public int displayFrequency;

   /** the time at which we stop the simulation (if != 0) */
   public int timeToFinish;

   /** displaying all messages on the console */
   public boolean verboseChoice;

   /** activating the print function in MemoryMatrix class */
   public boolean printMatrixes;

   /** displaying memory information on the console */
   public boolean checkMemorySize;

   /** two ActionGroup for sequence of GUI events */
   public ActionGroup displayActions1, displayActions2;

   /** the single Schedule instance */
   public Schedule displaySchedule;

   /** the Swarm we are observing */
   public ESFrameModelSwarm eSFrameModelSwarm;
```

```
/** our graphics or EZGraph output to files*/
public EZGraphImpl
        waitingListGraph=null, warehouseGraph=null,
        totalTimeLengthGraph=null, totalTimeLengthDataFile=null,
        totalDailyCostFile=null, totalCostFile=null,
    totalRevenueFile=null, dailyRevenueFile=null,
        dailySemimanufacturedOrderRevenueFile=null,
    totalInventoryFinancialCostFile=null,
        dailyStoredComponentValueFile=null,
    dailySemimanufacturedOrderCostFile=null,
        finishedOrderCostFile=null, benefitFile=null,
        benefitGraph=null, heldOrderGraph=null, heldOrderFile=null;

/** a Java graphic via ptplot class*/
public PTHistogram pTHistogram1, pTHistogram2;

/** PTHistogram positions */
public int unitHistogramXPos, unitHistogramYPos,
        endUnitHistogramXPos, endUnitHistogramYPos;

/** the address of the probe, if we need to access it; remember casting
 *  (ESFrameObserverSwarm$1$ESFrameObserverProbeMap)
 */
Object observerProbe;

/** Constructor for class */
public ESFrameObserverSwarm (Zone aZone) {

        super(aZone);

        // fill in the relevant parameters (only two, in this case).
        displayFrequency = 1;
        verboseChoice=false;
        printMatrixes=false;
        checkMemorySize=false;
        // PTHistogram positions
        unitHistogramXPos =    10;
        unitHistogramYPos =   300;
        endUnitHistogramXPos=  10;
        endUnitHistogramYPos= 250;
        timeToFinish=0;

        // Build a customized probe map using a `local' subclass
        // (a special kind of Java `inner class') of the
        // EmptyProbeMapImpl class.
        // This building operation produces a separated class file, with
        // name  ESFrameObserverSwarm$1$ESFrameObserverProbeMap.class
    class ESFrameObserverProbeMap extends EmptyProbeMapImpl {
            public VarProbe probeVariable (String name) {
                    return
                       Globals.env.probeLibrary.getProbeForVariable$inClass
                       (name, ESFrameObserverSwarm.this.getClass ());
            }
            public MessageProbe probeMessage (String name) {
```

```
                return
                   Globals.env.probeLibrary.getProbeForMessage$inClass
                   (name, ESFrameObserverSwarm.this.getClass ());
            }
            public void addVar (String name) {
                addProbe (probeVariable (name));
            }
            public void addMessage (String name) {
                addProbe (probeMessage (name));
            }

            public ESFrameObserverProbeMap (Zone _aZone, Class aClass) {
                super (_aZone, aClass);
                addVar ("displayFrequency");
                addVar ("verboseChoice");
                addVar ("printMatrixes");
                addVar ("checkMemorySize");
                addVar ("unitHistogramXPos");
                addVar ("unitHistogramYPos");
                addVar ("endUnitHistogramXPos");
                addVar ("endUnitHistogramYPos");
                addVar ("timeToFinish");
            }
         }
         // Install our custom probeMap class directly into the
         // probeLibrary
         observerProbe = new ESFrameObserverProbeMap (aZone, getClass ());
         Globals.env.probeLibrary.setProbeMap$For
            ((ESFrameObserverProbeMap) observerProbe, getClass ());
   }


   /** Create the objects used to display the model. */
   public Object buildObjects () {

         super.buildObjects ();

         // First, we create the model that we're actually observing. The
         // model is a subswarm of the observer.

         // creating the ESFrameModelSwarm via lispAppArchivers, which
         // refers to the jesframe.scm file where we can modify parameters
         // in a stable way without compiling again the code
         eSFrameModelSwarm =
            (ESFrameModelSwarm)Globals.env.lispAppArchiver.getWithZone$key(
                                                      getZone(),
"eSFrameModelSwarm");
         // instead of using the direct constructor way
         // eSFrameModelSwarm = new ESFrameModelSwarm (getZone ());

         // Now create probe objects on the model and ourselves. This gives a
         // simple user interface to let the user change parameters.

         Globals.env.createArchivedProbeDisplay (eSFrameModelSwarm,
                                                 "eSFrameModelSwarm");
```

```
Globals.env.createArchivedProbeDisplay (this, "eSFrameObserverSwarm");

// Instruct the control panel to wait for a button event: we
// halt here until someone hits a control panel button so the
// user can get a chance to fill in parameters before the
// simulation runs
getControlPanel ().setStateStopped ();

// to change the verboseChoice via probe you have to wright 'true'
// or 'false'; not simply 't' or 'f'

// OK - the user has specified all the parameters for the
// simulation.  Now we're ready to start.

// setting the 'verbose' value in StartESFrame
StartESFrame.verbose=verboseChoice;

// setting printMatrixes option
eSFrameModelSwarm.setMemoryMatrixPrintCapability(printMatrixes);

// First, let the model swarm build its objects.
eSFrameModelSwarm.buildObjects ();

// checking for logical consistency of dislayFrequency and
// ticksInTimeUnit
if (! (1==displayFrequency || 0==displayFrequency%
      eSFrameModelSwarm.getTicksInATimeUnit()) )
   {
         System.out.println("Use displayFrequency == 1 or");
         System.out.println(
         "    displayFrequency == k*ticksInATimeUnit, with k integer.");
         MyExit.exit(1);
   }

// Create the graph widget to display the waiting list content
waitingListGraph = new EZGraphImpl
   (getZone (), "Waiting lists", "Time",
    "Waiting list values (avrg, min, max)","waiting_list");

Globals.env.setWindowGeometryRecordName (waitingListGraph,
                                         "waitingListGraph");

// create the data for the waitingListGraph
waitingListGraph.createAverageSequence$withFeedFrom$andSelector
   ("avrgWaitingList", eSFrameModelSwarm.getUnitList(),
    SwarmUtils.getSelector("Unit","getWaitingListLength"));
waitingListGraph.createMinSequence$withFeedFrom$andSelector
   ("minWaitingList", eSFrameModelSwarm.getUnitList(),
    SwarmUtils.getSelector("Unit","getWaitingListLength"));

waitingListGraph.createMaxSequence$withFeedFrom$andSelector
   ("maxWaitingList", eSFrameModelSwarm.getUnitList(),
    SwarmUtils.getSelector("Unit", "getWaitingListLength"));
```

```
// Create the graph widget to display the quantities stored into
// the warehouse of each unit

if (eSFrameModelSwarm.getUseWarehouses())
   // if useWarehouses is true, draw the graph
   {
         warehouseGraph = new EZGraphImpl
            (getZone (), "Quantities in the warehouses", "Time",
             "Quantities (avrg, min, max)", "warehouse_list");

         Globals.env.setWindowGeometryRecordName
            (warehouseGraph, "warehouseGraph");

         // the data for the warehouseGraph
         warehouseGraph.createAverageSequence$withFeedFrom$andSelector
            ("avrgQ", eSFrameModelSwarm.getWarehouseList(),
             SwarmUtils.getSelector("Warehouse","getInventoryQuantity"));
         warehouseGraph.createMinSequence$withFeedFrom$andSelector
            ("minQ", eSFrameModelSwarm.getWarehouseList(),
             SwarmUtils.getSelector("Warehouse","getInventoryQuantity"));
         warehouseGraph.createMaxSequence$withFeedFrom$andSelector
            ("maxQ", eSFrameModelSwarm.getWarehouseList(),
             SwarmUtils.getSelector("Warehouse","getInventoryQuantity"));
   }

// Create the graph widget to display the total production time
// for all the finished orders divided by their total recipe lengths

totalTimeLengthGraph = new EZGraphImpl
   (getZone (), "Ratio total time / total lengths", "Time",
    "Ratio", "totalTimeLengthGraph");

Globals.env.setWindowGeometryRecordName
   (totalTimeLengthGraph, "totalTimeLengthGraph");

// the data for the totalTimeLengthGraph
totalTimeLengthGraph.createSequence$withFeedFrom$andSelector
   ("ratio", eSFrameModelSwarm,
    SwarmUtils.getSelector(eSFrameModelSwarm,"getTimeLengthRatio"));

// the same, in a file ('data.ratio')
totalTimeLengthDataFile =  new EZGraphImpl
   (getZone(), "data");
totalTimeLengthDataFile.createSequence$withFeedFrom$andSelector
   ("ratio", eSFrameModelSwarm,
    SwarmUtils.getSelector(eSFrameModelSwarm,"getTimeLengthRatio"));

// Send the amount of totalDailyCosts of Units to the file
// totalDailyCosts.txt
totalDailyCostFile =  new EZGraphImpl
  (getZone(), "Costs/totalDailyCosts");
totalDailyCostFile.createTotalSequence$withFeedFrom$andSelector
   ("txt", eSFrameModelSwarm.getUnitList(),
    SwarmUtils.getSelector("Unit","getTotalDailyCosts"));
```

```
// Send the cumulated amount of fixed and variable costs of units
// from the beginning of the simulation to the file totalCosts.txt)
totalCostFile =  new EZGraphImpl
   (getZone(), "Costs/totalCosts");
totalCostFile.createTotalSequence$withFeedFrom$andSelector
   ("txt", eSFrameModelSwarm.getUnitList(),
    SwarmUtils.getSelector("Unit","getTotalCosts"));

// Send the revenues of the virtual enterprise to the file
// totalRevenues.txt
totalRevenueFile =  new EZGraphImpl
   (getZone(), "Revenues/totalRevenues");
totalRevenueFile.createTotalSequence$withFeedFrom$andSelector
   ("txt", eSFrameModelSwarm.getUnitList(),
    SwarmUtils.getSelector("Unit","getTotalSaleRevenues"));

// Send the daily revenue from the finished orders to the file
// dailyRevenues
dailyRevenueFile =  new EZGraphImpl
   (getZone(), "Revenues/dailyRevenues");
dailyRevenueFile.createTotalSequence$withFeedFrom$andSelector
   ("txt", eSFrameModelSwarm.getUnitList(),
    SwarmUtils.getSelector("Unit","getDailySaleRevenues"));

// Send the revenues of the semimanufactured orders to the file
// dailySemimanufacturedOrderRevenues.txt)
dailySemimanufacturedOrderRevenueFile =  new EZGraphImpl
   (getZone(), "Revenues/dailySemimanufacturedOrderRevenues");
dailySemimanufacturedOrderRevenueFile.
   createTotalSequence$withFeedFrom$andSelector
   ("txt", eSFrameModelSwarm.getOrderList(),
    SwarmUtils.getSelector("Order","getOrderEvaluation"));

// Send the cumulated financial costs of the semi manufactured product
// in warehouses to the file totalInventoryFinancialCosts.txt
// we do not check if(eSFrameModelSwarm.getUseWarehouses()==true)
// producing a file filled of zeros if Warehouses are not used
   totalInventoryFinancialCostFile =  new EZGraphImpl
           (getZone(), "Costs/totalInventoryFinancialCosts");
   totalInventoryFinancialCostFile.
           createTotalSequence$withFeedFrom$andSelector
           ("txt", eSFrameModelSwarm.getUnitList(),
            SwarmUtils.getSelector("Unit","getTotalInventoryFinancialCosts"));

// Send the revenue of the stored components in each day to the
// file dailyStoredComponentValue
// we do not chesk if(eSFrameModelSwarm.getUseWarehouses()==true)
// producing a file filled of zeros if Warehouses are not used
   dailyStoredComponentValueFile =  new EZGraphImpl
           (getZone(), "Revenues/dailyStoredComponentValue");
   dailyStoredComponentValueFile.
           createTotalSequence$withFeedFrom$andSelector
           ("txt", eSFrameModelSwarm.getUnitList(),
            SwarmUtils.getSelector("Unit","getDailyStoredComponentValue"));
```

```
// Send the costs of the orders made from the beginning of the
// simulation, to the file finishedOrderCosts.txt
finishedOrderCostFile =  new EZGraphImpl
   (getZone(), "Costs/finishedOrderCosts");
finishedOrderCostFile.createTotalSequence$withFeedFrom$andSelector
   ("txt", eSFrameModelSwarm.getUnitList(),
    SwarmUtils.getSelector("Unit","getTotalFinishedOrderCosts"));

// Send the daily cost of the orders in production to the
// file dailySemimanufacturedOrderCosts.txt
dailySemimanufacturedOrderCostFile =  new EZGraphImpl
   (getZone(), "Costs/dailySemimanufacturedOrderCosts");
dailySemimanufacturedOrderCostFile.
   createTotalSequence$withFeedFrom$andSelector
   ("txt", eSFrameModelSwarm.getOrderList(),
    SwarmUtils.getSelector("Order","getOrderTotalCosts"));


// the data for the heldOrderGrapj
if(eSFrameModelSwarm.orCriterion==5){
   heldOrderGraph = new EZGraphImpl
          (getZone (), "Held orders", "Time",
           "Number", "heldOrderGraph");
   Globals.env.setWindowGeometryRecordName
          (heldOrderGraph, "heldOrderGraph");
   heldOrderGraph.createSequence$withFeedFrom$andSelector
          ("heldOrderGraph", eSFrameModelSwarm,
           SwarmUtils.getSelector(eSFrameModelSwarm,
                                  "getHeldOrderNumber"));
   // the same in a file
   heldOrderFile =  new EZGraphImpl
          (getZone(), "HeldOrders/heldOrders");
   heldOrderFile.createSequence$withFeedFrom$andSelector
          ("txt", eSFrameModelSwarm,
           SwarmUtils.getSelector(eSFrameModelSwarm,
                                  "getHeldOrderNumber"));
}

// the data for the benefitGraph
benefitGraph = new EZGraphImpl
   (getZone (), "Enterprise benefit", "Time",
    "Benefit", "benefitGraph");
Globals.env.setWindowGeometryRecordName
   (benefitGraph, "benefitGraph");
benefitGraph.createSequence$withFeedFrom$andSelector
   ("benefitGraph", eSFrameModelSwarm,
    SwarmUtils.getSelector(eSFrameModelSwarm,"getBenefit"));
// the same in a file
benefitFile =  new EZGraphImpl
   (getZone(), "Benefit/benefit");
benefitFile.createSequence$withFeedFrom$andSelector
   ("txt", eSFrameModelSwarm,
    SwarmUtils.getSelector(eSFrameModelSwarm,"getBenefit"));
```

```
        // the histogram for the waitingList of the units and for the
        // inventories in warehouses
        if(eSFrameModelSwarm.getUseWarehouses()==true){
           pTHistogram1 = new PTHistogram(unitHistogramXPos,unitHistogramYPos,
                                          "Orders in Units",
                                          "Units", "Count",
                                          eSFrameModelSwarm.totalUnitNumber,
                                          20.0,
                                          eSFrameModelSwarm.unitList,
                                          "Queues in u.",
                                          eSFrameModelSwarm.warehouseList,
                                          "Quant. in w.",
                                          eSFrameModelSwarm.
                                          procurementAssemblerList,
                                          "Q. in proc.ass.");
        }
     else pTHistogram1 =new PTHistogram(unitHistogramXPos,unitHistogramYPos,
                                          "Orders in Units",
                                          "Units", "Count",
                                          eSFrameModelSwarm.totalUnitNumber,
                                          20.0,
                                          eSFrameModelSwarm.unitList,
                                          "Queues in u.",
                                          eSFrameModelSwarm.
                                          procurementAssemblerList,
                                          "Q. in proc.ass.");

        // the histogram for the waitingList of the end units
        if ( eSFrameModelSwarm.totalEndUnitNumber != 0)
        pTHistogram2 = new PTHistogram(endUnitHistogramXPos,
                                       endUnitHistogramYPos,
                                       "Procurement (int. or ext.)",
                                       "End Units", "Count",
                                       eSFrameModelSwarm.totalEndUnitNumber,
                                       20.0,
                                       eSFrameModelSwarm.endUnitList,
                                       "q. in endU.");

        return this;
  }


/**
 * Create the actions necessary for the simulation. This is where
 * the schedule is built (but not run!)  Here we create a display
 * schedule - this is used to display the state of the world and
 * check for user input.
 */

  public Object buildActions () {
        int i;

        super.buildActions();
```

```
// First, let our model swarm build its own schedule.
eSFrameModelSwarm.buildActions();

// Create two ActionGroup for display
displayActions1 = new ActionGroupImpl (getZone());
displayActions2 = new ActionGroupImpl (getZone());

// Add the methods to the ActionGroup

// Schedule the update of the graphic widgets

if(checkMemorySize)
   displayActions1.createActionTo$message
         (this,SwarmUtils.getSelector(this, "checkMemorySize"));

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

if(eSFrameModelSwarm.getUseWarehouses())
   displayActions1.createActionTo$message(
                  warehouseGraph,
                  SwarmUtils.getSelector(warehouseGraph,"step"));

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

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

// ptplot Java histograms
displayActions1.createActionTo$message
   (pTHistogram1,
    SwarmUtils.getSelector(pTHistogram1, "addPoints"));
if(eSFrameModelSwarm.totalEndUnitNumber != 0)
displayActions1.createActionTo$message
   (pTHistogram2,
    SwarmUtils.getSelector(pTHistogram2, "addPoints"));

// EZGraph files

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

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

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

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

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

        // if(eSFrameModelSwarm.getUseWarehouses()){
        // look above for the reasons suggesting the ommission of this check
           displayActions1.createActionTo$message
                   (totalInventoryFinancialCostFile,
                    SwarmUtils.getSelector(totalInventoryFinancialCostFile, "step"));
// }

        // if(eSFrameModelSwarm.getUseWarehouses()){
        // look above for the reasons suggesting the ommission of this check
           displayActions1.createActionTo$message
                   (dailyStoredComponentValueFile,
                   SwarmUtils.getSelector(dailyStoredComponentValueFile, "step"));
        // }

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

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

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

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

        if(eSFrameModelSwarm.orCriterion==5)
           displayActions1.createActionTo$message
                   (heldOrderGraph,
                    SwarmUtils.getSelector(heldOrderGraph, "step"));

        if(eSFrameModelSwarm.orCriterion==5)
           displayActions1.createActionTo$message
                   (heldOrderFile,
                    SwarmUtils.getSelector(heldOrderFile, "step"));

        // Schedule the update of the probe displays
        displayActions2.createActionTo$message
           (Globals.env.probeDisplayManager,
            SwarmUtils.getSelector(Globals.env.probeDisplayManager,"update"));

        // Schedule an update for the whole user
        // interface code.  This is crucial: without this, no
```

```
        // graphics update and the control panel will be
        // dead. It's best to put it at the end of the display
        // schedule
        displayActions2.createActionTo$message(getActionCache(),
                        SwarmUtils.getSelector(getActionCache(),"doTkEvents"));

        // check to finish
        displayActions2.createActionTo$message(this,
                        SwarmUtils.getSelector(this,"finish"));

        // The display schedule. Note the repeat interval: display
        // is frequently the slowest part of a simulation, so redrawing less
        // frequently can be a help.
        displaySchedule = new ScheduleImpl (getZone (), displayFrequency);

        // insert ActionGroup instance on the repeating Schedule
        // instance

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

        return this;
  }

  /**
    activateIn: - activate the schedules so they're ready to run.
    The swarmContext argument has to do with what we were activated
    *in*.  Typically the ObserverSwarm is the top-level Swarm, so
    it's activated in "null". But other Swarms and Schedules and
    such will be activated inside of us.  */

  public Activity activateIn (Swarm swarmContext) {
        // First, activate ourselves (just pass along the context).
        super.activateIn (swarmContext);

        // Activate the model swarm in ourselves. The model swarm is a
        // subswarm of the observer swarm.
        eSFrameModelSwarm.activateIn (this);

        // Now activate our schedule in ourselves. This arranges for
        // the execution of the schedule we built.
        displaySchedule.activateIn (this);

        // Activate returns the swarm activity - the thing that's ready to run.
        return getActivity();
  }

  /** drop the Observer, but after having dropped all the EZGraphs files,
        to close them */
  public void drop() {
```

```
System.out.println("\nPress newly Quit to exit.");
getControlPanel ().setStateStopped ();

if(waitingListGraph!=null)
   waitingListGraph.drop();
waitingListGraph=null;

if(warehouseGraph!=null)
   warehouseGraph.drop();
warehouseGraph=null;

if(totalTimeLengthGraph!=null)
   totalTimeLengthGraph.drop();
totalTimeLengthGraph=null;

if(totalTimeLengthDataFile!=null)
   totalTimeLengthDataFile.drop();
totalTimeLengthDataFile=null;

if(totalDailyCostFile!=null)
   totalDailyCostFile.drop();
totalDailyCostFile=null;

if(totalCostFile!=null)
   totalCostFile.drop();
totalCostFile=null;

if(totalRevenueFile!=null)
   totalRevenueFile.drop();
totalRevenueFile=null;

if(dailyRevenueFile!=null)
   dailyRevenueFile.drop();
dailyRevenueFile=null;

if(dailySemimanufacturedOrderRevenueFile!=null)
   dailySemimanufacturedOrderRevenueFile.drop();
dailySemimanufacturedOrderRevenueFile=null;

if(totalInventoryFinancialCostFile!=null)
   totalInventoryFinancialCostFile.drop();
totalInventoryFinancialCostFile=null;

if(dailyStoredComponentValueFile!=null)
   dailyStoredComponentValueFile.drop();
dailyStoredComponentValueFile=null;

if(dailySemimanufacturedOrderCostFile!=null)
   dailySemimanufacturedOrderCostFile.drop();
dailySemimanufacturedOrderCostFile=null;

if(finishedOrderCostFile!=null)
   finishedOrderCostFile.drop();
finishedOrderCostFile=null;
```

```
        if(benefitFile!=null)
           benefitFile.drop();
        benefitFile=null;

        if(benefitGraph!=null)
           benefitGraph.drop();
        benefitGraph=null;

        super.drop();
    }

    /** print the memory usage information */
    public void checkMemorySize(){

        Runtime runtime = Runtime.getRuntime();
        System.out.println ("At tick: " + Globals.env.getCurrentTime() +
                            "  free mem.: " + runtime.freeMemory() +
                            "  total mem.: " + runtime.totalMemory() +
                            "  orders in wait: " +
                            ((ListImpl)eSFrameModelSwarm.getOrderList()).
                            getCount()
                            );
    }

    /** to finish automatically, if timeToFinish!=0 */
    public void finish(){
        if(timeToFinish != 0 && timeToFinish<=Globals.env.getCurrentTime())
           {
                System.out.println("jES virtually quitted at time " +
                                   Globals.env.getCurrentTime());
                getControlPanel ().setStateQuit ();;
           }
    }
}
```