Das Referenzmodell PECS

Agentenbasierte Modellierung menschlichen Handelns, Entscheidens und Verhaltens

Dissertation

zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
(Dr. rer. nat.)

Fakultät für Mathematik und Informatik
Universität Passau

- Anhang -

Dipl.-Inf. Christoph Urban

April 2004

ANHANG A QUELLCODE DES MODELLS *SOLIDARNETZWERK*

A.1 Die Strukturkomponente *Solidarnetzwerk*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell Solidarnetzwerk
#
#  Name:             Solidarnetzwerk
#
#  Art:              Strukturkomponente
#
#  Version:          1
#
#  Beschreibung:     Gesamtmodell
#
#-------------------------------------------------------------------------------

HIGH LEVEL COMPONENT Solidarnetzwerk

SUBCOMPONENTS
 Umwelt,
 ARRAY [315] Agent

COMPONENT CONNECTIONS

#-------------------------------------------------------------------------------
# Umwelt -> Agent
#-------------------------------------------------------------------------------

Umwelt.InitRow                   --> Agent{ALL}.InitRow;
Umwelt.InitCol                   --> Agent{ALL}.InitCol;
Umwelt.SatisLev{i OF 1..9}       --> Agent{j OF 1..315|IMOD(j,9)+1=i}.SatisLev;
Umwelt.R{i OF 1..9}{j OF 1..9}      --> Agent{k OF 1..315|IMOD(k,9)+1=i}.R[j];
Umwelt.Grid{i OF 1..21}{j OF 1..21} --> Agent{ALL}.Grid[i][j];
Umwelt.Classes{i OF 1..315}         --> Agent[i].InitClass;

Umwelt.INIT{i OF 1..315}            --> Agent[i].INIT;
Umwelt.MIGRAT{i OF 1..315}          --> Agent[i].MIGRAT;
Umwelt.PLAYGAME{i OF 1..315}        --> Agent[i].PLAY;
Umwelt.RESET{i OF 1..315}           --> Agent[i].RESET;

#-------------------------------------------------------------------------------
# Agent -> Umwelt
#-------------------------------------------------------------------------------

Agent{i OF 1..315}.Row               --> Umwelt.AgentPos[i][1];
Agent{i OF 1..315}.Col               --> Umwelt.AgentPos[i][2];
Agent{i OF 1..315}.Income            --> Umwelt.AgentInc[i];
Agent{i OF 1..315}.Satisfac          --> Umwelt.AgentSat[i];
Agent{ALL i}.freeValues{ALL k}{ALL l} --> Umwelt.freeValues[i][k][l];

Agent{i OF 1..315}.DONEINIT          --> Umwelt.DONEINIT[i];
Agent{i OF 1..315}.DONEMIGR          --> Umwelt.DONEMIGR[i];
Agent{i OF 1..315}.DONEPLAY          --> Umwelt.DONEPLAY[i];
Agent{i OF 1..315}.NOTMIGR           --> Umwelt.NOTMIGR[i];
Agent{i OF 1..315}.moved             --> Umwelt.AgentMov[i];
Agent{ALL i}.VALUES_READY            --> Umwelt.VALUES_OK[i];

END OF Solidarnetzwerk
```

A.2 Die Komponente *Umwelt*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell Solidarnetzwerk
#
#  Name:             Umwelt
#
#  Art:              Basiskomponente
#
#  Version:          1
#
#  Beschreibung:     Steuerung des Spielablaufes und Bereitstellung des
#                    Spielfeldes
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Umwelt

MOBILE SUBCOMPONENTS OF CLASS AgentRef

LOCAL DEFINITIONS

DIMENSIONS
 GridDim:= 21

#-------------------------------------------------------------------------------
# fuer Animation:
# CopyArray sucht aus allen 315 Arrays der Groesse GridDim x GridDim
# das des Aktiven heraus => das Array mit den aktuellen Einkommen der
# gerade freien Felder im Migrationsfenster des aktiven Spielers
#-------------------------------------------------------------------------------
PROCEDURE CopyARRAY(ARRAY [b][a][a] REAL: values_S,
                    INTEGER: active   --> ARRAY [a][a] REAL)
DECLARE
 ARRAY [a][a] help (REAL)
 BEGIN

 FOR i FROM 1 TO GridDim REPEAT
   FOR j FROM 1 TO GridDim REPEAT
     help[i][j] := values_S[active][i][j];
   END_LOOP
 END_LOOP
RETURN(ARRAY help);
END_PROC

#-------------------------------------------------------------------------------
# Berechnen der Pay-Off-Bilanzen Matrix
#-------------------------------------------------------------------------------
PROCEDURE CalcPay(INT:b, INT:e, REAL:alpha --> ARRAY [a][a] REAL)
DECLARE
 ARRAY [a][a] p (REAL),
 Gij(REAL),
 Gji(REAL),
 Lij(REAL),
 Lji(REAL)

BEGIN
 FOR i FROM 1 TO 9 REPEAT
   FOR j FROM 1 TO 9 REPEAT
     Gij := i/10 * (1 - j/10) * b;
     Gji := j/10 * (1 - i/10) * b;
     Lij := (1 - i/10) * j/10 * e;
     Lji := (1 - j/10) * i/10 * e;
```

```
      # Ueberpruefen der PD- und COOP Bedingungen
      IF ((Gij-Lij > 0) AND (Gji-Lji > 0)) AND
          (alpha >= (Lij / Gij)) AND (alpha >= (Lji/Gji))
      DO
        p[i][j] := Gij - Lij;  # Nettogewinn falls Spiel moeglich
      END
      ELSE DO
        p[i][j] :=0;
      END
      DISPLAY("%f\n", p[i][j]);
    END_LOOP
    DISPLAY("\n");
  END_LOOP
RETURN (ARRAY p);  # 9x9 Array der Ergebnisse, wenn jeder mit jedem spielt
END_PROC

#-------------------------------------------------------------------------------
# Fuer Animation: Statistikvariablen erstellen
# Gibt von 3 Arrays jeweils die Summe zurueck, Gesamteinkommen und
# Anzahl der zufriedenen Agenten und Anzahl der gerade bewegten Agenten
#-------------------------------------------------------------------------------
PROCEDURE CalcNet(ARRAY[a] REAL : agentInc ,
                  ARRAY[a] INTEGER: agentSat,
                  ARRAY[a] INTEGER: isMoved   --> REAL, INTEGER, INTEGER)
DECLARE
  income(REAL),
  happy (INTEGER),
  nMoved (INTEGER)
BEGIN
  income := 0;
  happy  := 0;
  nMoved := 0;
FOR k FROM 1 TO 315 REPEAT
  income := income + agentInc[k];
  happy  := happy  + agentSat[k];
  nMoved := nMoved + isMoved[k];
END_LOOP
RETURN (income, happy, nMoved);
END_PROC

#-------------------------------------------------------------------------------
# Fuer alle 9 Klassen:Berechnen des SatisLev Feldes = Mindesteinkommen,
# um zufrieden zu sein
# Suche zu jeder Beduerftigkeitsklasse i das j, mit dem er den hoechsten
# PayOff erreichen wuerde
# das * 4 (im besten Fall 4 "sehr gute" Nachbarn). Dies dann * minlevel
#-------------------------------------------------------------------------------
PROCEDURE CalSatis (ARRAY [a][b] REAL: r, REAL: minLevel --> ARRAY [a] REAL)
DECLARE
 ARRAY [a] rhelp (REAL),
 help(REAL)
BEGIN
 FOR i FROM 1 TO 9 REPEAT
   help := 0;
   FOR j FROM 1 TO 9 REPEAT
     IF r[i][j] > help DO
       help := r[i][j];  # hoechster PayOff, den i mit einem j machen kann
     END
   END_LOOP
   rhelp[i] := ( help * 4 ) * minLevel;
 END_LOOP

 FOR l FROM 1 TO 9 REPEAT
   DISPLAY ("%lf, ", rhelp[l]);
 END_LOOP
RETURN (ARRAY rhelp);
END_PROC
```

```
#-------------------------------------------------------------------------------
# Berechnen der 21x21 Matrix Ani, um die Agenten animieren zu koennen;
#  1 -  9 : unzufriedener Agent der Klassen 1 bis 9
# 11 - 19 : zufriedener Agent der Klassen 1 bis 9
# 10 : leeres Feld
#-------------------------------------------------------------------------------
PROCEDURE CalcAnim (ARRAY [a][b] INTEGER: agentPos,
                    ARRAY [a]    INTEGER: agentSat,
                    ARRAY [c][c] INTEGER: grid
                    --> ARRAY [c][c] INTEGER)
DECLARE
  ARRAY[c][c] help (INTEGER)
BEGIN
  FOR i FROM 1 TO 21 REPEAT
   FOR j FROM 1 TO 21  REPEAT
    IF grid [i][j] = 10 DO
       help[i][j] := 10;     # leeres Feld bleibt auch in Animation leer
    END
   END_LOOP
  END_LOOP
  FOR k FROM 1 TO 315 REPEAT # sonst je nach Besitzer den Wert setzen
    help[agentPos[k][1]][agentPos[k][2]]:=agentSat[k]*10 + IMOD(k, 9) + 1;
  END_LOOP
RETURN (ARRAY help);
END_PROC

DECLARATION OF ELEMENTS

CONSTANTS
#-------------------------------------------------------------------------------
# Wahrscheinlichkeit fuer Migrationsoption
#-------------------------------------------------------------------------------
 MigrProb (REAL)     := 0.05,

#-------------------------------------------------------------------------------
# Schwellwert fuer Zufriedenheit relativ zu Vmax
#-------------------------------------------------------------------------------
 MinLevel (REAL)     := 0.50,

#-------------------------------------------------------------------------------
# Maximal - Benefit
#-------------------------------------------------------------------------------
 B        (INTEGER)  := 4,

#-------------------------------------------------------------------------------
# Maximal - Effort
#-------------------------------------------------------------------------------
 E        (INTEGER)  := 1

STATE VARIABLES
DISCRETE
#-------------------------------------------------------------------------------
# Spielrunde (Iteration)
#-------------------------------------------------------------------------------
 Round    (INTEGER)  := 0,

#-------------------------------------------------------------------------------
# Id des aktiven Agenten
#-------------------------------------------------------------------------------
 Active   (INTEGER)  := 1,

#-------------------------------------------------------------------------------
# Aktuelles Gesamteinkommen
#-------------------------------------------------------------------------------
 NetBenef (REAL)     := 0.0,
```

```
#-------------------------------------------------------------------------------
# Anzahl der zufriedenen Agenten
#-------------------------------------------------------------------------------
 NetSatis (INTEGER)  := 0,

#-------------------------------------------------------------------------------
# Anzahl der Agenten, die umgezogen sind
#-------------------------------------------------------------------------------
 NetMoved (INTEGER)  := 0,

#-------------------------------------------------------------------------------
# Anfangsplazierung Zeile
#-------------------------------------------------------------------------------
 InitRow  (INTEGER)  := 0,

#-------------------------------------------------------------------------------
# Anfangsplazierung Spalte
#-------------------------------------------------------------------------------
 InitCol  (INTEGER)  := 0,

#-------------------------------------------------------------------------------
# alte Postion nach Umzug
#-------------------------------------------------------------------------------
 OldRow   (INTEGER)  := 0,

#-------------------------------------------------------------------------------
# alte Position nach Umzug
#-------------------------------------------------------------------------------
 OldCol   (INTEGER)  := 0,

#-------------------------------------------------------------------------------
# Payoff-Bilanz fuer die Agenten (R)
#-------------------------------------------------------------------------------
 ARRAY [9][9] R     (REAL) := 0,

#-------------------------------------------------------------------------------
# pro Klasse der Wert, ab dem Mitglied zuf.
#-------------------------------------------------------------------------------
 ARRAY [9] SatisLev (REAL) := 0,

#-------------------------------------------------------------------------------
# Spielfeld, 10 =: leer
#-------------------------------------------------------------------------------
 ARRAY [GridDim][GridDim] Grid (INTEGER) := 10,

#-------------------------------------------------------------------------------
# fuer Animation des Spielfeldes:
# Initalisiert mit 10 (leer), bekommt dann einen Wert in Abhaengigkeit
# der Agentenklasse und der Zufriedenheit
# 10 =: leer || 1-9 =: unzufriedene ||11-19 zufriedene Agenten
#-------------------------------------------------------------------------------
 ARRAY [GridDim][GridDim] Ani  (INTEGER) := 10,

#-------------------------------------------------------------------------------
# fuer Animation: Array mit den Einkommen der freien Felder in Umgebung
#-------------------------------------------------------------------------------
 ARRAY [GridDim][GridDim] ValuesOfEnvironment (REAL) := 99,

#-------------------------------------------------------------------------------
# die Klassen aller Agenten
#-------------------------------------------------------------------------------
 ARRAY [315] Classes (INTEGER) :=1,

#-------------------------------------------------------------------------------
# Das eigentliche Spiel geht los
#-------------------------------------------------------------------------------
 StartGam (LOGICAL) := FALSE
```

```
DEPENDENT VARIABLES
DISCRETE
 Alpha(REAL)

SENSOR VARIABLES
#-------------------------------------------------------------------------------
# Row, Column
#-------------------------------------------------------------------------------
 ARRAY [315][2] AgentPos (INTEGER),

#-------------------------------------------------------------------------------
# Income
#-------------------------------------------------------------------------------
 ARRAY [315]    AgentInc (REAL),

#-------------------------------------------------------------------------------
# Moode von Agent (0=:unhappy)
#-------------------------------------------------------------------------------
 ARRAY [315]    AgentSat (INTEGER),

#-------------------------------------------------------------------------------
# Agent hat sich bewegt (1) oder nicht (0)
#-------------------------------------------------------------------------------
 ARRAY [315]    AgentMov (INTEGER),

#-------------------------------------------------------------------------------
# ==> ValuesOfEnvironment
#-------------------------------------------------------------------------------
 ARRAY [315][GridDim][GridDim] freeValues (REAL)

RANDOM VARIABLES

#-------------------------------------------------------------------------------
# fuer Anfangsbelegung des Spielfeldes
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Position des Agenten
#-------------------------------------------------------------------------------
 RanRow(INTEGER): IUNIFORM(LowLimit:=1, UpLimit:=21),
 RanCol(INTEGER): IUNIFORM(LowLimit:=1, UpLimit:=21),

#-------------------------------------------------------------------------------
# Migrationswahrsch.
#-------------------------------------------------------------------------------
 RanMigr (REAL) : UNIFORM (LowLimit:=0, UpLimit:=1),

 RanPos(INTEGER): IUNIFORM(LowLimit:=1, UpLimit:=1000000)

TRANSITION INDICATORS

#-------------------------------------------------------------------------------
# Berechnung der "Satisfactionlevels"
#-------------------------------------------------------------------------------
 CALCSATI,

#-------------------------------------------------------------------------------
# Position auslosen
#-------------------------------------------------------------------------------
 SEEKPOS,

#-------------------------------------------------------------------------------
# Position frei?
#-------------------------------------------------------------------------------
 TESTFREE,
```

```
#-------------------------------------------------------------------------------
# Position fuer naechsten Agent auslosen
#-------------------------------------------------------------------------------
 NEXTPOS,

#-------------------------------------------------------------------------------
# neue Runde beginnt
#-------------------------------------------------------------------------------
 NEWROUND,

#-------------------------------------------------------------------------------
# Auslosen der Migrationsoptionen
#-------------------------------------------------------------------------------
 PLAYMIGR,

#-------------------------------------------------------------------------------
# Agent hat Migrationsoption gewonnen
#-------------------------------------------------------------------------------
 WONMIGR,

#-------------------------------------------------------------------------------
# der naechste ist dran
#-------------------------------------------------------------------------------
 NEXTMIGR,

#-------------------------------------------------------------------------------
# Ende der Spielrunde
#-------------------------------------------------------------------------------
 ENDROUND,

 Test,

#-------------------------------------------------------------------------------
# Signale an Agenten
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Initialisierungsphase
#-------------------------------------------------------------------------------
 ARRAY [315] INIT,

#-------------------------------------------------------------------------------
# Migrationsoption erhalten
#-------------------------------------------------------------------------------
 ARRAY [315] MIGRAT,

#-------------------------------------------------------------------------------
# SupportGame spielen
#-------------------------------------------------------------------------------
 ARRAY [315] PLAYGAME,

#-------------------------------------------------------------------------------
# Anzahl bewegter auf Null setzen
#-------------------------------------------------------------------------------
 ARRAY [315] RESET

SENSOR INDICATORS

#-------------------------------------------------------------------------------
# Initialisierung abgeschlossen
#-------------------------------------------------------------------------------
 ARRAY [315] DONEINIT,

#-------------------------------------------------------------------------------
# Agent ist umgezogen
#-------------------------------------------------------------------------------
 ARRAY [315] DONEMIGR,
```

```
#-------------------------------------------------------------------------------
# SupportGame zu Ende
#-------------------------------------------------------------------------------
 ARRAY [315] DONEPLAY,

#-------------------------------------------------------------------------------
# Einkommen der freien Felder in Umgebung gespeichert
#-------------------------------------------------------------------------------
 ARRAY [315] VALUES_OK,

#-------------------------------------------------------------------------------
# Agent will nicht umziehen
#-------------------------------------------------------------------------------
 ARRAY [315] NOTMIGR


LOCATIONS
#-------------------------------------------------------------------------------
# Queue, in der alle Agenten in der Spielreihenfolge abgelegt sind
#-------------------------------------------------------------------------------
 Lottery (AgentRef ORDERED BY INC Played, INC WaitPos) := 0 AgentRef

DYNAMIC BEHAVIOUR

#-------------------------------------------------------------------------------
# Wahrscheinlichkeit dafuer, dass zwei Agenten auch in der naechsten Runde
# noch Nachbarn sind
#-------------------------------------------------------------------------------
Alpha := (1-MigrProb)*(1-MigrProb);

#-------------------------------------------------------------------------------
# Berechnung der Payoffs
#-------------------------------------------------------------------------------
ON START DO
  (ARRAY R^) := CalcPay(B, E, Alpha);  # 9x9 Array,jeder mit jedem
  SIGNAL CALCSATI;  #  nur einmal am Anfang berechnen, aendert sich nicht
  SIGNAL Test;
END

#-------------------------------------------------------------------------------
# Berechnung der "Satisfactionlevels" = Einkommen, ab dem ein Agent zufrieden
# ist (mit seiner Position) [Zufriedenheit -> Agent/Emotion]
#-------------------------------------------------------------------------------
ON CALCSATI DO
  (ARRAY SatisLev^) := CalSatis(ARRAY R, MinLevel);
                       # liefert Array mit einem
                       # Wert pro Beduerftigkeitsklasse, ab dem der
                       # Spieler zufrieden ist
  SIGNAL SEEKPOS;
END

#-------------------------------------------------------------------------------
# initiale Verteilung der Agenten, Active ist mit 1 initialisiert
#-------------------------------------------------------------------------------
ON SEEKPOS DO
  InitRow^ := RanRow;
  InitCol^ := RanCol;
  SIGNAL TESTFREE;
END

#-------------------------------------------------------------------------------
# Ist der ausgeloste Platz frei ?
#-------------------------------------------------------------------------------
ON TESTFREE DO
 IF Grid[InitRow][InitCol] = 10 DO    # Feld ist noch frei, Agent wird ...
   Lottery^ : ADD 1 NEW AgentRef      # ...in Liste der Spieler aufgenommen
   CHANGING
```

```
      Id^      := Active;
      WaitPos^ := RanPos;   # Einordnung in der Schlange wird neu bestimmt
    END
    # Anfangsklasse des Agenten wird in ein Array eingetragen
    Classes[Active]^ := IMOD(Active, 9) + 1;
    # Agent wird auf Schachbrett gesetzt
    Grid[InitRow][InitCol]^ := IMOD(Active, 9) + 1;   # Zuteilen in Klassen,
         # erster kommt in Kl. 2, der naechste in 3 usw, der 9. in Kl. 1 usw
    SIGNAL INIT[Active]; # an Agent/INIT_S
 END
 ELSE DO
    # neuen Platz suchen
    SIGNAL SEEKPOS;     # neuer Aufruf des Zufallgenerators
 END
END

#-------------------------------------------------------------------------------
# Warte auf Rueckmeldung des Agenten, bevor der naechste gesetzt wird
#-------------------------------------------------------------------------------
ON DONEINIT[Active] DO
 SIGNAL NEXTPOS;
END

#-------------------------------------------------------------------------------
# Wenn alle Agenten initialisiert sind, kann das Spiel losgehen,
# sonst wird fuer den naechsten eine Position gesucht
#-------------------------------------------------------------------------------
ON NEXTPOS DO
 IF Active < 315 DO
    Active^ := Active + 1;
    SIGNAL SEEKPOS;
 END
 ELSE DO
    StartGam^ := TRUE;
    (ARRAY Ani^) := CalcAnim(ARRAY AgentPos, ARRAY AgentSat, ARRAY Grid);
 END
END

#-------------------------------------------------------------------------------
# Ab hier beginnt der eigentliche Spielablauf !
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Dieses Konstrukt synchronisiert Simulationszeit und Runde
#-------------------------------------------------------------------------------
WHENEVER (StartGam = TRUE AND T >= Round + 1) DO
  Round^ := Round + 1;
  SIGNAL NEWROUND;
END

#-------------------------------------------------------------------------------
# Beginn einer neuen Runde
#-------------------------------------------------------------------------------
ON NEWROUND DO
 Lottery^ : FROM Lottery GET AgentRef{ALL}
 CHANGING
    Played ^ := 0;
 END
 SIGNAL PLAYMIGR;
 SIGNAL RESET{ALL}; # in Physis die Anzahl der bewegten Ag.auf 0
END

#-------------------------------------------------------------------------------
# Auslosen der Migrationsoptionen
#-------------------------------------------------------------------------------
ON PLAYMIGR DO
  IF Lottery:AgentRef[1].Played = 0 DO
```

```
    Active^  := Lottery:AgentRef[1].Id;
    Lottery^ : FROM Lottery GET AgentRef[1]
    CHANGING
      Played^  := 1;
      # Positionierungskriterium fuer die naechste Runde
      WaitPos^ := RanPos;
    END
    # Auslosen der Migrationsoption
    IF RanMigr <= MigrProb DO
      SIGNAL WONMIGR;
    END
    ELSE DO
      SIGNAL NEXTMIGR;
    END
  END
  ELSE DO
    SIGNAL PLAYGAME{ALL};
  END
END

#-------------------------------------------------------------------------------
# Der naechste Agent aus der Warteschlange lost um eine Migrationsoption
#-------------------------------------------------------------------------------
ON NEXTMIGR DO
  SIGNAL PLAYMIGR;
END

#-------------------------------------------------------------------------------
# Der aktive Agent ist zufrieden und will nicht gehen, also
# ist der naechste dran
#-------------------------------------------------------------------------------
ON NOTMIGR[Active]  DO
  SIGNAL PLAYMIGR;
END

#-------------------------------------------------------------------------------
# Agent hat eine Migrationsoption gewonnen => die alten Koordinaten werden
# gespeichert, um sie spaeter gegebenenfalls auf dem Spielfeld frei zu machen
#-------------------------------------------------------------------------------
ON WONMIGR DO
  OldRow^ := AgentPos[Active][1];
  OldCol^ := AgentPos[Active][2];
  SIGNAL MIGRAT[Active];
END

#-------------------------------------------------------------------------------
# Agent hat die freien Felder der Umgebung betrachtet, fuer Animation
#-------------------------------------------------------------------------------
ON VALUES_OK[Active] DO
 (ARRAY ValuesOfEnvironment^) := CopyARRAY(ARRAY freeValues, Active);
END

#-------------------------------------------------------------------------------
# Rueckmeldung des Agenten : er ist umgezogen
#-------------------------------------------------------------------------------
ON DONEMIGR[Active] DO
   Grid[OldRow][OldCol]^ := 10;     # altes Feld wird frei
   Grid[AgentPos[Active][1]][AgentPos[Active][2]]^ := IMOD(Active, 9) + 1;


   Ani[OldRow][OldCol]^ := 10;     # altes Feld wird frei
   Ani[AgentPos[Active][1]][AgentPos[Active][2]]^:=
                          AgentSat[Active]*10+IMOD(Active,9)+1;
  #-------------------------------------------------------------------------------
  # Falls noch nicht alle Agenten in der Runde dran waren, spielt der
  # naechste um eine Migrationsoption, andernfalls beginnt das Support-Game
  #-------------------------------------------------------------------------------
```

```
  IF Lottery:AgentRef[1].Played = 0 DO
    SIGNAL NEXTMIGR;
  END
  ELSE DO
    SIGNAL PLAYGAME{ALL}; # an Agenten
  END
END

#-------------------------------------------------------------------------------
# spezielles Konstrukt zur Sunchronisation der Agenten :
# Alle melden sich zum gleichen Zeitpunkt (T +0,751) zurueck, so
# dass keine Ruecksicht auf die Takte genommen werden muss
# Deshalb genuegt es hier, auf die Rueckmeldung des Agenten Nr. 1 zu warten
#-------------------------------------------------------------------------------
ON DONEPLAY[1] DO
  #-----------------------------------------------------------------------------
  # Berechnen des NetBenefit und die Anzahl der zufriedenen Agenten
  #-----------------------------------------------------------------------------
  (NetBenef^, NetSatis^, NetMoved^):= CalcNet(ARRAY AgentInc, ARRAY AgentSat,
                                              ARRAY AgentMov);
  #-----------------------------------------------------------------------------
  # Berechnung der Matrix Ani fuer spaetere Animation, Verteilung von
  # Werten von 0 (leer), ueber 1 (Klasse 1 unzufrieden),
  # bis 19 (Klasse 9, zufrieden)
  #-----------------------------------------------------------------------------
  (ARRAY Ani^) := CalcAnim(ARRAY AgentPos, ARRAY AgentSat, ARRAY Grid);

 SIGNAL ENDROUND;
END

END OF Umwelt
```

A.3 Die Agenten

A.3.1 Die Strukturkomponente *Agent*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell Solidarnetzwerk
#
#  Name:             Agent
#
#  Art:              Strukturkomponente
#
#  Version:          1
#
#  Beschreibung:     Strukturkomponente, die die Subkomponenten
#                    eines Agenten instanziiert und miteinander
#                    verschaltet
#
#-------------------------------------------------------------------------------

HIGH LEVEL COMPONENT Agent

SUBCOMPONENTS
 Sensor,
 Wahrnehmung,
 Emotion,
 Wissen,
 Status,
 Verhalten,
 Aktor
```

```
INPUT CONNECTIONS

#-------------------------------------------------------------------------------
# Initialisierung der SensorVariablen
#-------------------------------------------------------------------------------
 InitRow     --> Sensor.InitRow_S;
 InitCol     --> Sensor.InitCol_S;
 Grid        --> Sensor.Grid_S;
 InitClass   --> Sensor.InitClass_S;
 R           --> Sensor.R_S;
 SatisLev    --> Sensor.SatisLev_S;

#-------------------------------------------------------------------------------
# Initialisierung der SensorSignale
#-------------------------------------------------------------------------------
 INIT        --> Sensor.INIT_S;
 MIGRAT      --> Sensor.MIGRAT_S;
 PLAY        --> Sensor.PLAY_S;
 RESET       --> Sensor.RESET_MOVED_S;

OUTPUT EQUIVALENCES

#-------------------------------------------------------------------------------
# Durchreichen der Variablen an die Umwelt
#-------------------------------------------------------------------------------
 Income        := Status.Income;
 Row           := Wissen.Row;
 Col           := Wissen.Col;
 Satisfac      := Emotion.Moode;
 moved         := Status.moved;
 freeValues    := Wissen.FreeValues;

#-------------------------------------------------------------------------------
# Durchreichen der Signale an die Umwelt
#-------------------------------------------------------------------------------
 DONEINIT      := Aktor.DONE_INIT;
 DONEMIGR      := Aktor.DONEMIGR;
 DONEPLAY      := Aktor.DONEPLAY;
 NOTMIGR       := Aktor.NOTMIGR;
 VALUES_READY  := Wissen.FREE_VALUES_READY;

COMPONENT CONNECTIONS

#-------------------------------------------------------------------------------
# Sensor --> Wahrnehmung
#-------------------------------------------------------------------------------
 Sensor.InitRow              --> Wahrnehmung.InitRow_S;
 Sensor.InitCol              --> Wahrnehmung.InitCol_S;
 Sensor.Grid{ALL i}{ALL j}   --> Wahrnehmung.Grid_S[i][j];
 Sensor.InitClass            --> Wahrnehmung.InitClass_S;
 Sensor.R{ALL i}             --> Wahrnehmung.R_S[i];
 Sensor.SatisLev             --> Wahrnehmung.SatisLev_S;

 Sensor.INIT                 --> Wahrnehmung.INIT_S;
 Sensor.MIGRAT               --> Wahrnehmung.MIGRAT_S;
 Sensor.PLAY                 --> Wahrnehmung.PLAY_S;
 Sensor.RESET_MOVED          --> Wahrnehmung.RESET_MOVED_S;

#-------------------------------------------------------------------------------
# Wahrmehmung --> Wissen
#-------------------------------------------------------------------------------
 Wahrnehmung.InitRow                --> Wissen.InitRow_S;
 Wahrnehmung.InitCol                --> Wissen.InitCol_S;
 Wahrnehmung.Grid{ALL i}{ALL j}     --> Wissen.Grid_S[i][j];
 Wahrnehmung.LocalWin{ALL i}{ALL j} --> Wissen.LocalWin_S[i][j];
 Wahrnehmung.InitClass              --> Wissen.InitClass_S;
 Wahrnehmung.R{ALL i}               --> Wissen.R_S[i];
```

```
 Wahrnehmung.SatisLev              --> Wissen.SatisLev_S;
 Wahrnehmung.INIT                  --> Wissen.INIT_S;
 Wahrnehmung.MIGRAT                --> Wissen.MIGRAT_S;
 Wahrnehmung.PLAY                  --> Wissen.PLAY_S;

#-------------------------------------------------------------------------------
# Wahrnehmung --> Status
#-------------------------------------------------------------------------------
 Wahrnehmung.InitClass             --> Status.Class_S;
 Wahrnehmung.INIT                  --> Status.INIT_S;
 Wahrnehmung.RESET_MOVED           --> Status.RESET_MOVED_S;

#-------------------------------------------------------------------------------
# Wissen  --> Wahrnehmung
#-------------------------------------------------------------------------------
 Wissen.Row                        --> Wahrnehmung.Row_S;
 Wissen.Col                        --> Wahrnehmung.Col_S;

#-------------------------------------------------------------------------------
# Wissen --> Status
#-------------------------------------------------------------------------------
 Wissen.Income                     --> Status.Income_S;
 Wissen.UPDATE_INCOME              --> Status.UPDATE_INCOME_S;
 Wissen.MOVED                      --> Status.MOVED_S;

#-------------------------------------------------------------------------------
# Wissen --> Emotion
#-------------------------------------------------------------------------------
 Wissen.SatisLev      --> Emotion.SatisLev;
 Wissen.Income        --> Emotion.Income;
 Wissen.ASKMOODE      --> Emotion.ASKMOODE_S;
 Wissen.ONLY_ASKMOODE --> Emotion.ONLY_ASKMOODE_S;

#-------------------------------------------------------------------------------
# Emotion --> Wissen
#-------------------------------------------------------------------------------
 Emotion.Moode            --> Wissen.Moode_S;
 Emotion.MOODE_IS_UPDATED --> Wissen.MOODE_IS_UPDATED;

#-------------------------------------------------------------------------------
# Wissen --> Verhalten
#-------------------------------------------------------------------------------
 Wissen.MIGRAT                --> Verhalten.MIGRAT_S;
 Wissen.I_STAY                --> Verhalten.I_STAY_S;
 Wissen.PLAY                  --> Verhalten.PLAY_S;
 Wissen.DONE_PLAY             --> Verhalten.DONE_PLAY_S;
 Wissen.DONE_INIT             --> Verhalten.DONE_INIT_S;
 Wissen.DONE_SET              --> Verhalten.DONE_SET_S;

#-------------------------------------------------------------------------------
# Verhalten --> Aktor
#-------------------------------------------------------------------------------
 Verhalten.DONE_INIT             --> Aktor.DONE_INIT_S;
 Verhalten.CALCULATE_INCOME      --> Aktor.CALCULATE_INCOME_S;
 Verhalten.FIND_BEST_POSITION    --> Aktor.FIND_BEST_POSITION_S;
 Verhalten.CALC_INCOME_FOR_PLAY  --> Aktor.INCOME_FOR_PLAY_S;
 Verhalten.DONE_PLAY             --> Aktor.DONEPL_S;
 Verhalten.NOTMIGR               --> Aktor.NOTMIGR_S;
 Verhalten.DONE_SET              --> Aktor.DONE_SET_S;

#-------------------------------------------------------------------------------
# Aktor --> Wissen
#-------------------------------------------------------------------------------
 Aktor.CALCULATE_INCOME     --> Wissen.CALCULATE_INCOME_S;
 Aktor.FIND_BEST_POSITION   --> Wissen.FIND_BEST_POSITION_S;
 Aktor.ONLY_INCOME          --> Wissen.ONLY_INCOME_S;
END OF Agent
```

A.3.2 Die Komponente *Sensor*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell Solidarnetzwerk
#
#  Name:             Sensor
#
#  Art:              Basiskomponente
#
#  Version:          1
#
#  Beschreibung:     Entgegennahme sensorischer Inputs aus der Umwelt
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Sensor

LOCAL DEFINITIONS

DIMENSIONS
 GridDim := 21

#-------------------------------------------------------------------------------
# kopiert das per Sensorvariable erhaltene Schachbrett in eine Zustandsvariable
#-------------------------------------------------------------------------------
PROCEDURE CopyGrid(ARRAY [a][a] INTEGER: grid_S --> ARRAY [a][a] INTEGER)
 DECLARE
 ARRAY [a][a] help (INTEGER)

 BEGIN
 FOR i FROM 1 TO GridDim     REPEAT
   FOR j FROM 1 TO GridDim           REPEAT
     help[i][j] := grid_S[i][j];
   END_LOOP
 END_LOOP

RETURN(ARRAY help);
END_PROC

#-------------------------------------------------------------------------------
# Diese Prozedur kopiert die entsprechende Zeile aus der PayOff-Matrix
# und den Wert, ab dem der Agent zufrieden ist
#-------------------------------------------------------------------------------
PROCEDURE CopyKnowS(ARRAY [a] REAL: rS, REAL: satisLeS --> ARRAY [a] REAL,REAL)
 DECLARE
 ARRAY [a] help (REAL)

 BEGIN
 FOR i FROM 1 TO 9    REPEAT
   help[i] := rS[i];
 END_LOOP

RETURN (ARRAY help, satisLeS);
END_PROC

DECLARATION OF ELEMENTS

STATE VARIABLES
#-------------------------------------------------------------------------------
# Anfangsposition
#-------------------------------------------------------------------------------
 InitRow (INTEGER)                      := 0,
 InitCol (INTEGER)                      := 0,
```

```
#-------------------------------------------------------------------------------
# Bedürftigkeitsklasse
#-------------------------------------------------------------------------------
 InitClass (INTEGER)                    := 0,

#-------------------------------------------------------------------------------
# aktueller Spielfeld-Zustand
#-------------------------------------------------------------------------------
 ARRAY [GridDim][GridDim] Grid (INTEGER) :=10,      # leeres Spielfeld

#-------------------------------------------------------------------------------
# Auszahlung bei bilateraler Kooeperation
#-------------------------------------------------------------------------------
 ARRAY [9] R (REAL)                     := 0,

#-------------------------------------------------------------------------------
# Schranke für die Berechnung der Zufriedenheit
#-------------------------------------------------------------------------------
 SatisLev (REAL)                        := 0,      # und Sensor SatisLes

 helpfree(INTEGER)                      := 0

SENSOR VARIABLES

 InitRow_S (INTEGER),
 InitCol_S (INTEGER),
 ARRAY [GridDim][GridDim] Grid_S (INTEGER),
 InitClass_S(INTEGER), # Anfangsklasse, von Umwelt ermittelt
 ARRAY [9] R_S (REAL), # kommt aus der 9x9 PayOff Matrix der Umwelt. Diese
                       # wird bei der InputConnection aufgeteilt: jedem Agenten
                       # ein 9 elt.Vektor, seiner Beduerftigkeit entsprechend

 SatisLev_S (REAL)     # kommt aus dem 9-elt Vektor aus der Umwelt, pro Agent
                       # ein Wert, seiner Beduerft. entsprechend.

TRANSITION INDICATORS
 INIT,
 MIGRAT,
 PLAY,
 RESET_MOVED           # an Wahrnehm

SENSOR INDICATORS      # kommen durch Initialsierung, aus Umwelt
 INIT_S,
 MIGRAT_S,
 PLAY_S,
 RESET_MOVED_S

DYNAMIC BEHAVIOUR

#-------------------------------------------------------------------------------
# Initialisierungsphase
#-------------------------------------------------------------------------------
ON INIT_S DO
 InitRow^              := InitRow_S;
 InitCol^              := InitCol_S;
 (ARRAY Grid^)         := CopyGrid(ARRAY Grid_S);
 InitClass^            := InitClass_S;
 (ARRAY R^, SatisLev^) := CopyKnowS(ARRAY R_S, SatisLev_S);
 SIGNAL INIT;          # initiales Kopieren der Matrizen in Wahrnehm anstossen
END

#-------------------------------------------------------------------------------
# Umwelt gibt das Signal, dass der Agent eine Migrationsoption gewonnen hat
#-------------------------------------------------------------------------------
ON MIGRAT_S DO
 (ARRAY Grid^) := CopyGrid(ARRAY Grid_S);# Agent braucht akutelles Schachbrett
 SIGNAL MIGRAT;        # ueber Wahrnehm, Wissen an Verhaltn
```

```
END

#-------------------------------------------------------------------------------
# Umwelt gibt das Signal zur Einleitung des Support Games
#-------------------------------------------------------------------------------
ON PLAY_S DO
 (ARRAY Grid^) := CopyGrid(ARRAY Grid_S);# als test
 SIGNAL PLAY;          # ueber Wahrnehm, Wissen an Verhaltn
END

#-------------------------------------------------------------------------------
# Signal von Umwelt, dass in Physis die Anzahl der bewegten Agenten auf 0
# zurueckgesetzt werden soll. Geht ueber Wahrnehm, Wissen, Verhalten, Actor
#-------------------------------------------------------------------------------
ON RESET_MOVED_S DO
 SIGNAL RESET_MOVED;
END

END OF Sensor
```

A.3.3 Die Komponente *Wahrnehmung*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell Solidarnetzwerk
#
#  Name:             Wahrnehmung
#
#  Art:              Basiskomponente
#
#  Version:          1
#
#  Beschreibung:     Weiterleitung der sensorischen Informationen
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Wahrnehmung

LOCAL DEFINITIONS

DIMENSIONS
 GridDim := 21

#-------------------------------------------------------------------------------
# berechnet Position so, dass er links reinkommt, wenn er rechts rausgeht ...
#-------------------------------------------------------------------------------
FUNCTION FW(INTEGER: Position --> INTEGER)
 BEGIN
 IF Position > GridDim DO
   RETURN (Position - GridDim);
 END
 ELSIF Position < 1 DO
   RETURN (Position + GridDim);
 END
 ELSE DO
   RETURN (Position);
 END
END_FUNC

#-------------------------------------------------------------------------------
# berechnet die Koordinaten des lokalen Migrationsfensters so, dass an allen
# Koordinaten ausserhalb die Werte 99 gespeichert sind und nur an denen des
# Migrationsfensters die richtigen Koordinaten abgespeichert sind.
# Auf diese Weise soll erkennbar sein, welche Koordinaten innerhalb des
```

```
# aktuellen Migrationsfensters liegen und welchen Wert sie haben.
#-------------------------------------------------------------------------------
PROCEDURE Look(ARRAY [a][a] INTEGER: grid,
               INTEGER: row,
               INTEGER: col ,
               INTEGER: migrWin  --> ARRAY [a][a] INTEGER)
DECLARE
 ARRAY [a][a] help (INTEGER),
 center (INTEGER),
 rowIsOk (LOGICAL),
 colIsOk(LOGICAL)
BEGIN
 center := IFLOOR(migrWin / 2); # Mittelpunkt des migration window

 #-------------------------------------------------------------------------------
 # Ueberpruefen, ob alle Zeilen zusammenhaengend sind
 #-------------------------------------------------------------------------------
 IF (row-center>=1 AND row+center<=21) DO
    rowIsOk:=TRUE;
 END
 ELSE DO
    rowIsOk:=FALSE;
 END

 #-------------------------------------------------------------------------------
 # Ueberpruefen, ob alle Spalten zusammenhaengend sind
 #-------------------------------------------------------------------------------
 IF (col-center>=1 AND col+center<=21) DO
    colIsOk:=TRUE;
 END
 ELSE DO
    colIsOk:=FALSE;
 END
#-------------------------------------------------------------------------------
# Nun werden alle vier Faelle durchgegangen
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Das komplette Migrationsfenster ist zusammenhaengend und geht nirgends
# ueber die Raender hinaus
#-------------------------------------------------------------------------------
 IF (rowIsOk AND colIsOk) DO
  FOR i FROM 1 TO GridDim     REPEAT
   FOR j FROM 1 TO GridDim           REPEAT
     IF ((i<row-center) OR (i>row+center) OR
         (j<col-center) OR (j> col+center)) DO
                           help[i][j] := 99;
    END
     ELSE DO
          help[i][j]:= grid[i][j];
     END
   END_LOOP
  END_LOOP
 END
 #-------------------------------------------------------------------------------
 # Der Zeilenbereich geht ueber die Raender hinaus. Die Spalten sind ok
 #-------------------------------------------------------------------------------
 ELSIF ((NOT rowIsOk) AND colIsOk) DO
  FOR i FROM 1 TO GridDim     REPEAT
   FOR j FROM 1 TO GridDim           REPEAT
     IF ((j<col-center) OR (j>col+center)OR
         ((i>FW(row+center)) AND (i<FW(row-center)))) DO
                           help[i][j] := 99;
     END
     ELSE DO
          help[i][j]:= grid[i][j];
     END
```

```
   END_LOOP
  END_LOOP
 END
 #-------------------------------------------------------------------------------
 # Der Spaltenbereich geht ueber die Raender hinaus. Die Zeilen sind ok
 #-------------------------------------------------------------------------------
 ELSIF (rowIsOk AND (NOT colIsOk)) DO
  FOR i FROM 1 TO GridDim     REPEAT
   FOR j FROM 1 TO GridDim           REPEAT
     IF ((i<row-center) OR (i>row+center)OR
         ((j>FW(col+center)) AND (j<FW(col-center)))) DO
                           help[i][j] := 99;
     END
     ELSE DO
          help[i][j]:= grid[i][j];
     END
   END_LOOP
  END_LOOP
 END
 #-------------------------------------------------------------------------------
 # Das Migrationsfenster geht ueber alle Raender hinaus, es besteht
 # also aus 4 einzelnen Bereichen
 #-------------------------------------------------------------------------------
 ELSE DO
  FOR i FROM 1 TO GridDim     REPEAT
   FOR j FROM 1 TO GridDim           REPEAT
     IF ( ((i>FW(row+center)) AND (i<FW(row-center))) OR
         ((j>FW(col+center)) AND (j<FW(col-center)))) DO
                           help[i][j] := 99;
     END
     ELSE DO
          help[i][j]:= grid[i][j];
     END
   END_LOOP
  END_LOOP
 END
RETURN (ARRAY help);
END_PROC

#-------------------------------------------------------------------------------
# kopiert das Schachbrett aus der Sensorvariable in eine Zustandsvariable
#-------------------------------------------------------------------------------
PROCEDURE CopyWinW(ARRAY [a][a] INTEGER: grid_S --> ARRAY [a][a] INTEGER)
DECLARE
 ARRAY [a][a] help (INTEGER)
 BEGIN
   FOR i FROM 1 TO GridDim REPEAT
     FOR j FROM 1 TO GridDim  REPEAT
       help[i][j] :=grid_S[i][j];
     END_LOOP
   END_LOOP
RETURN(ARRAY help);
END_PROC

#-------------------------------------------------------------------------------
# kopiert den Vektor aus der PayOff-Matrix und den Wert, ab dem der
# Agent zufrieden ist. Wird von der Sensorkomponente gemeldet
#-------------------------------------------------------------------------------
PROCEDURE CopyKnowW(ARRAY [a] REAL: rS, REAL: satisLeS --> ARRAY [a] REAL, REAL)
 DECLARE
 ARRAY [a] help (REAL)

 BEGIN
 FOR i FROM 1 TO 9    REPEAT
   help[i] := rS[i];
 END_LOOP
RETURN (ARRAY help, satisLeS);
```

```
END_PROC

DECLARATION OF ELEMENTS

CONSTANTS
 MigrWin (INTEGER):=11

STATE VARIABLES
#-------------------------------------------------------------------------------
# Startposition des Agenten
#-------------------------------------------------------------------------------
 InitRow (INTEGER) := 1,
 InitCol (INTEGER) := 1,

#-------------------------------------------------------------------------------
# Migrationsfenster
#-------------------------------------------------------------------------------
 ARRAY [GridDim][GridDim] LocalWin (INTEGER)  := 0,

#-------------------------------------------------------------------------------
# Spielfeld
#-------------------------------------------------------------------------------
 ARRAY [GridDim][GridDim] Grid (INTEGER)  := 0,

#-------------------------------------------------------------------------------
# Payouts bei bilateraler Kooperation
#-------------------------------------------------------------------------------
 ARRAY [9] R (REAL)  := 0,

#-------------------------------------------------------------------------------
# Schranke für Zufriedenheitsberechnung
#-------------------------------------------------------------------------------
 SatisLev (REAL)     := 0,

#-------------------------------------------------------------------------------
# Beduerftigkeitsklasse
#-------------------------------------------------------------------------------
 InitClass (INTEGER) := 1

SENSOR VARIABLES

#-------------------------------------------------------------------------------
# alles sind Inputs der Komponente Sensor
#-------------------------------------------------------------------------------
 InitRow_S (INTEGER),
 InitCol_S (INTEGER),
 ARRAY [GridDim][GridDim] Grid_S (INTEGER),
 InitClass_S (INTEGER),
 ARRAY [9] R_S (REAL),
 SatisLev_S (REAL),


#-------------------------------------------------------------------------------
# Inputs aus der Komponente Wissen
#-------------------------------------------------------------------------------
 Row_S     (INTEGER),
 Col_S     (INTEGER)

TRANSITION INDICATORS
 INIT,
 INITPOSITION,
 MIGRAT,
 PLAY,
 CALCULATE_LOCALWIN_1,
 CALCULATE_LOCALWIN_2,
 RESET_MOVED
```

```
SENSOR INDICATORS
 INIT_S,
 MIGRAT_S,
 PLAY_S,
 RESET_MOVED_S,
 INITPOSITION_S # von SensorA (INITPOS)


DYNAMIC BEHAVIOUR

#-------------------------------------------------------------------------------
# Initialisierungsphase
#-------------------------------------------------------------------------------
ON INIT_S DO
 InitRow^              := InitRow_S;
 InitCol^              := InitCol_S;
 InitClass^            := InitClass_S;
 (ARRAY R^, SatisLev^) := CopyKnowW(ARRAY R_S, SatisLev_S);
 SIGNAL INIT;
END

#-------------------------------------------------------------------------------
# Signal, dass Agent eine Migrationsoption gewonnen hat
#-------------------------------------------------------------------------------
ON MIGRAT_S DO
 (ARRAY Grid^):= CopyWinW(ARRAY Grid_S);
  SIGNAL CALCULATE_LOCALWIN_1;
END

#-------------------------------------------------------------------------------
# Signal, zur Einleitung des SupportGames
#-------------------------------------------------------------------------------
ON PLAY_S DO
 (ARRAY Grid^):= CopyWinW(ARRAY Grid_S);
 SIGNAL CALCULATE_LOCALWIN_2;
END

#-------------------------------------------------------------------------------
# Berechnung eines Sichtfensters: Was ausserhalb des aktuellen Migrations-
# fensters liegt, bekommt den Wert 99, die anderen behalten ihre Belegungen
#-------------------------------------------------------------------------------
ON CALCULATE_LOCALWIN_1 DO
  (ARRAY LocalWin^) := Look(ARRAY Grid_S, Row_S, Col_S, MigrWin);
   SIGNAL MIGRAT;
END

#-------------------------------------------------------------------------------
# Berechnung eines Sichtfensters: Was ausserhalb des aktuellen Migrations-
# fensters liegt, bekommt den Wert 99, die anderen behalten ihre Belegungen
#-------------------------------------------------------------------------------
ON CALCULATE_LOCALWIN_2 DO
  (ARRAY LocalWin^) := Look(ARRAY Grid_S, Row_S, Col_S, MigrWin);
   SIGNAL  PLAY;
END

#-------------------------------------------------------------------------------
# Signal von Umwelt, dass in Physis die Anzahl der bewegten Agenten auf 0
# zurueckgesetzt werden soll. Geht ueber Wahrnehm, Wissen, Verhalten, Actor
#-------------------------------------------------------------------------------
ON RESET_MOVED_S DO
       SIGNAL RESET_MOVED;
END

END OF Wahrnehmung
```

A.3.4 Die Komponente *Emotion*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell Solidarnetzwerk
#
#  Name:             Emotion
#
#  Art:              Basiskomponente
#
#  Version:          1
#
#  Beschreibung:     Berechnung der Zufriedenheit eines Agenten
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Emotion

DECLARATION OF ELEMENTS

STATE VARIABLES
DISCRETE
 Moode (INTEGER) := 0   # 0: nicht zufrieden
                        # 1: zufrieden mit Position

SENSOR VARIABLES
#-------------------------------------------------------------------------------
# Mindesteinkommen fuer Zufriedenheit, von Umwelt
#-------------------------------------------------------------------------------
 SatisLev (REAL),

#-------------------------------------------------------------------------------
# Einkommen , von Wissen
#-------------------------------------------------------------------------------
 Income (REAL)


TRANSITION INDICATORS
 MOODE_IS_UPDATED,
 WAIT

SENSOR INDICATORS
 ASKMOODE_S,            # von Wissen, Neuberechnung der Zufriedenheit
 ONLY_ASKMOODE_S        # von Wissen, bei PLAYGAME,

DYNAMIC BEHAVIOUR

#-------------------------------------------------------------------------------
# Wissen fordert die Komponente Emotion auf, die aktuelle
# Zufriedenheit zu berechnen.
#-------------------------------------------------------------------------------
ON ASKMOODE_S DO
 IF Income >= SatisLev DO
   Moode^ := 1;
 END
 ELSE DO
   Moode^ := 0;
 END
 SIGNAL WAIT;
END

#-------------------------------------------------------------------------------
# Verzoegerung der Zufriedenheitsmeldung an Wissen
#-------------------------------------------------------------------------------
ON WAIT DO
 SIGNAL MOODE_IS_UPDATED;
```

```
END

#-------------------------------------------------------------------------------
# Support Game:
# hier soll nur die Zufriedenheit berechnet werden, ohne
# eine Bestaetigung an Wissen, da mit dieser dann
# in Wissen weitere Berechnungen loslaufen wuerden
#-------------------------------------------------------------------------------
ON ONLY_ASKMOODE_S DO
  IF Income >= SatisLev DO
    Moode^ := 1;
  END
  ELSE DO
    Moode^ := 0;
  END
END

END OF Emotion
```

A.3.5 Die Komponente *Wissen*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell Solidarnetzwerk
#
#  Name:             Wissen
#
#  Art:              Basiskomponente
#
#  Version:          1
#
#  Beschreibung:     Speicherung und Verwaltung der Spielinformationen
#                    eines Agenten
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Wissen

LOCAL DEFINITIONS

DIMENSIONS
 GridDim := 21

#-------------------------------------------------------------------------------
# F sorgt dafuer, dass das Spielfeld ein Torus ist, d.h.
# wenn ein Agent rechts rausgeht, kommt er von links wieder rein usw.
#-------------------------------------------------------------------------------
FUNCTION FWI(INTEGER: Position --> INTEGER)
BEGIN
  IF Position > GridDim DO
    RETURN (Position - GridDim);
  END
  ELSIF Position < 1 DO
    RETURN (Position + GridDim);
  END
  ELSE DO
    RETURN (Position);
  END
END_FUNC

#-------------------------------------------------------------------------------
# Berechnung des aktuellen Einkommens:
#-------------------------------------------------------------------------------
FUNCTION CalcIncome(ARRAY [a][a] INTEGER: localWin,
```

```
                    ARRAY [b] REAL: r ,
                    INTEGER: row,
                    INTEGER: col         --> REAL)
DECLARE
 help (REAL)
BEGIN
 help := r[localWin[FWI(row-1)][col]]+
         r[localWin[FWI(row+1)][col]]+
         r[localWin[row][FWI(col+1)]]+
         r[localWin[row][FWI(col-1)]];
RETURN (help);
END_FUNC

#-------------------------------------------------------------------------------
# Bestimmung einer freien Position mit maximalen Einkommen innerhalb des
# Migration-Window.
# gibt die Position (Zeile und Spalte) inclusive PayOff zurueck.
# Ausserdem werden alle freien Felder innerhalb des Migrationsfenster mit ihren
# Einkommenswerten abgespeichert
#-------------------------------------------------------------------------------
PROCEDURE FindBestPosition
    ( ARRAY [a][a] INTEGER: localWin,
      ARRAY [a][a] INTEGER: grid,
      ARRAY [b] REAL: r,
      INTEGER: rowS,
      INTEGER: colS,
      ARRAY [a][a] REAL: val
      --> LOGICAL,INTEGER, INTEGER, REAL,  ARRAY [a][a]REAL )
DECLARE
 flag (LOGICAL),             # ob eine freie Stelle gefunden wurde
 row (INTEGER),
 col (INTEGER),
 best (REAL),
 help (REAL),
 ARRAY [a][a] freevalue (REAL)
BEGIN
 best   := -9;               # sicherheitshalber
 FOR i FROM 1 TO GridDim REPEAT
  FOR j FROM 1 TO GridDim REPEAT
   # Falls das Feld frei ist, (kann also nicht das aktuelle sein)
   # Einkommen berechnen
   IF localWin[i][j]<> 10 DO  # nicht leer oder ausserhalb Migrationfenster
     freevalue[i][j]:= 99;
   END
   ELSE DO # Feld ist frei und innerhalb des Migrationsfensters

     #-------------------------------------------------------------------------
     # Berechnung des Einkommens anhand der Werte des Schachbrettes,
     # damit auch ein Einkommen an den Raendern des Ausschnitts berechnet
     # werden kann
     #-------------------------------------------------------------------------
     help := r[grid[FWI(i-1)][j]] +
             r[grid[i][FWI(j+1)]] +
             r[grid[FWI(i+1)][j]] +
             r[grid[i][FWI(j-1)]];
     #-------------------------------------------------------------------------
     # Wert an freier Stelle fuer Animation speichern
     #-------------------------------------------------------------------------
     freevalue[i][j] := help;

     #-------------------------------------------------------------------------
     # Ist das Einkommen an gerade gepruefter Stelle (i,j) groesser
     # als an allen bisher gepruefften Positionen ?
     #-------------------------------------------------------------------------
     IF ((i<>row) OR (j<>col)) AND (help>best)      DO
       flag := TRUE;
       best := help;
```

```
       # Speichern der Koordinaten
       row  := i;
       col  := j;
     END
   END
   #-------------------------------------------------------------------------
   # Wenn keine Position frei war, werden die alten
   # Koordinaten zurueckgegeben und dies per flag markiert
   #-------------------------------------------------------------------------
   IF best = -9 DO
     flag:= FALSE;
     row := rowS;
     col := colS;
   END
  END_LOOP
 END_LOOP

 #-------------------------------------------------------------------------
 # beste Postion einschl. Ergebnis
 #-------------------------------------------------------------------------
RETURN(flag,row, col, best, ARRAY freevalue);
END_PROC

#-------------------------------------------------------------------------
# kopiert das Migrationsfenster aus der Senorvariable in eine Zustandsvariable
#-------------------------------------------------------------------------
PROCEDURE CopyWin(ARRAY [a][a] INTEGER: local_S --> ARRAY [a][a] INTEGER)
DECLARE
 ARRAY [a][a] help (INTEGER)
 BEGIN
   FOR i FROM 1 TO GridDim REPEAT
     FOR j FROM 1 TO GridDim  REPEAT
       help[i][j] := local_S[i][j];
     END_LOOP
   END_LOOP
RETURN(ARRAY help);
END_PROC

DECLARATION OF ELEMENTS

CONSTANTS
#-------------------------------------------------------------------------
# Groesse des Migrationsfensters
#-------------------------------------------------------------------------
 MigrWin (INTEGER):= 11

STATE VARIABLES
DISCRETE

#-------------------------------------------------------------------------
# Migrationsfenster
#-------------------------------------------------------------------------
 ARRAY [GridDim][GridDim] LocalWin (INTEGER)  := 0,
 ARRAY [GridDim][GridDim] FreeValues(REAL):=99,

#-------------------------------------------------------------------------
# Aktuelles Einkommen
#-------------------------------------------------------------------------
 Income (REAL)       := 0,

#-------------------------------------------------------------------------
# Einkommen an besserer Position
#-------------------------------------------------------------------------
 NewIncome (REAL)    := 0,
```

```
#-------------------------------------------------------------------------------
# Aktuelle Koordinaten
#-------------------------------------------------------------------------------
 Row (INTEGER)       := 1,
 Col (INTEGER)       := 1,

#-------------------------------------------------------------------------------
# Koordinaten einer besseren Position
#-------------------------------------------------------------------------------
 NewRow (INTEGER)    := 1,
 NewCol (INTEGER)    := 1,

#-------------------------------------------------------------------------------
# Payouts
#-------------------------------------------------------------------------------
 ARRAY [10] R (REAL) := 0,

#-------------------------------------------------------------------------------
# Schranke für die Berechnung Zufriedenheit
#-------------------------------------------------------------------------------
 SatisLev (REAL)     := 0,

#-------------------------------------------------------------------------------
# Bedürftigkeitsklasse
#-------------------------------------------------------------------------------
 Class (INTEGER)     := 1,
 freeFieldExists(LOGICAL) := TRUE


SENSOR VARIABLES

#-------------------------------------------------------------------------------
# Inputs von der Komponente Wahrnehmung
#-------------------------------------------------------------------------------
 InitRow_S (INTEGER),
 InitCol_S (INTEGER),
 ARRAY [GridDim][GridDim] Grid_S (INTEGER),
 ARRAY [GridDim][GridDim] LocalWin_S (INTEGER),
 InitClass_S (INTEGER),
 ARRAY [9] R_S (REAL),
 SatisLev_S (REAL),

#-------------------------------------------------------------------------------
# Inputs von der Komponente Emotion
#-------------------------------------------------------------------------------
 Moode_S (INTEGER)


TRANSITION INDICATORS
 MIGRAT,
 FREE_VALUES_READY,      # an Umwelt, fuer Animation
 UPDATE_CLASS_IN_STATUS, # an Status
 MOVED,                  # an Status
 UPDATE_INCOME,          # ""
 END_OF_WAIT,            # internes Signal zur Verzoegerung
 END_OF_WAIT2,           # ""
 END_OF_WAIT3,           # ""
 ASKMOODE,               # an Emotion
 ONLY_ASKMOODE,          # an Emotion, ohne weitere Berechnung in Wissen
 I_STAY,                 # an Verhaltn
 PLAY,                   # ""
 DONE_PLAY,              # ""
 DONE_INIT,               # ""
 DONE_SET                # an Verhalten

SENSOR INDICATORS
 INIT_S,
```

```
 INITPOSITION_S,
 MIGRAT_S,
 CALCULATE_INCOME_S,   # von Aktor
 MOODE_IS_UPDATED,     # von Emotion, Zufriedenheit wurde aktualisiert
 FIND_BEST_POSITION_S, # von Aktor
 ONLY_INCOME_S,        # von Aktor
 PLAY_S,
 SETINC_S              # von Verhaltn

DYNAMIC BEHAVIOUR

#-------------------------------------------------------------------------------
# Kopieren der Daten aus der Umwelt; damit alles konsistent bleibt,
# werden die Daten ueber Sensor und Wahrnehmung geschleust
#-------------------------------------------------------------------------------
ON INIT_S DO
 Row^ := InitRow_S;
 Col^ := InitCol_S;
 R{i OF 1..9}^ := R_S[i];
 R[10]^ := 0;                # Array wird auf 10 Elt erweitert, falls Feld leer
 SatisLev^  := SatisLev_S;
 Class^ := InitClass_S;      # Initial. der Klasse, aendert sich nicht mehr
 SIGNAL DONE_INIT;           # an Verhalten, der gibt das Signal direkt weiter
END

#-------------------------------------------------------------------------------
# Signal: Migrationsoption erhalten
#-------------------------------------------------------------------------------
ON MIGRAT_S DO
 (ARRAY LocalWin^) := CopyWin (ARRAY LocalWin_S);
 SIGNAL MIGRAT;
END

#-------------------------------------------------------------------------------
# Aufgrund Migrationsoption das Einkommen berechnen
#-------------------------------------------------------------------------------
ON CALCULATE_INCOME_S DO
 Income^ := CalcIncome(ARRAY LocalWin, ARRAY R,Row,Col);
 SIGNAL UPDATE_INCOME; # in Status aktuelles Einkommen speichern
 SIGNAL END_OF_WAIT; # sicherheitshalber die Berechnung von Moode verzoegern
END

#-------------------------------------------------------------------------------
# Nach Aktualisierung des Einkommens die Zufriedenheit abfragen,
# sicherheitshalber verzoegert
#-------------------------------------------------------------------------------
ON END_OF_WAIT DO
  SIGNAL ASKMOODE;
END

#-------------------------------------------------------------------------------
# das Signal zur Suche nach besten Position kommt gleichzeitig mit
# dem Signal zur Zufriedenheitsberechnung im Wissen an.
# Dementsprechend muessen die Folgeaktionen synchronisiert werden
#-------------------------------------------------------------------------------
ON FIND_BEST_POSITION_S DO
   (freeFieldExists^, NewRow^, NewCol^, NewIncome^, ARRAY FreeValues^) :=
   FindBestPosition(ARRAY LocalWin, ARRAY Grid_S, ARRAY R, Row, Col,
             ARRAY FreeValues);
END

#-------------------------------------------------------------------------------
# Zu diesem Zeitpunkt sollte Moode aktualisiert und die
# beste Position gefunden sein
#-------------------------------------------------------------------------------
ON MOODE_IS_UPDATED DO
 IF (freeFieldExists = TRUE) DO
```

```
   IF (NewIncome >= Income) OR (Moode_S = 0) DO
     Row^ := NewRow;
     Col^ := NewCol;
     SIGNAL MOVED;             # an Status zur Protokollierung des Umzugs
     SIGNAL DONE_SET;
     SIGNAL FREE_VALUES_READY; # nur wenn er wirklich umzieht
   END
   ELSE DO
    SIGNAL I_STAY;   # an Aktor, meldet an Umwelt, dass dieser Agent nicht
                     # gehen will
   END
 END
 ELSE DO # kein freies Feld in Umgebung
  SIGNAL I_STAY;
 END
END

#-------------------------------------------------------------------------------
# Signal: SupportGame
#-------------------------------------------------------------------------------
ON PLAY_S DO
  (ARRAY LocalWin^) := CopyWin (ARRAY LocalWin_S);
  SIGNAL PLAY;
END

#-------------------------------------------------------------------------------
# SupportGame: nur fuer Zufriedenheitsaktualisierung wird Einkommen bestimmt
#-------------------------------------------------------------------------------
ON ONLY_INCOME_S DO
  Income^ := CalcIncome(ARRAY LocalWin, ARRAY R, Row, Col);
  SIGNAL UPDATE_INCOME;
  SIGNAL END_OF_WAIT2; # sicherheitshalber die Berechnung von Moode verzoegern
END

#-------------------------------------------------------------------------------
# SupportGame: Zufriedenheit abfragen
#-------------------------------------------------------------------------------
ON END_OF_WAIT2 DO
 SIGNAL ONLY_ASKMOODE;
 SIGNAL END_OF_WAIT3; # Bestaetigung verzoegern
END

#-------------------------------------------------------------------------------
# ist die Zufriedenheit aktualisiert, wird Ende des SupportGames bestaetigt
#-------------------------------------------------------------------------------
ON END_OF_WAIT3 DO
 SIGNAL DONE_PLAY;
END

END OF Wissen
```

A.3.6 Die Komponente *Status*

```
#-------------------------------------------------------------------------------
#  Projekt:          Modell Solidarnetzwerk
#
#  Name:             Status
#
#  Art:              Basiskomponente
#
#  Version:          1
#
#  Beschreibung:     Berechnung von Statusinformationen des Agenten
#-------------------------------------------------------------------------------
```

```
BASIC COMPONENT Status

DECLARATION OF ELEMENTS

STATE VARIABLES
DISCRETE
 Class (INTEGER) :=0,
 Income (REAL)   :=0,
 moved (INTEGER) :=0          # ist der Agent umgezogen (1: ja, 0: nein)

SENSOR VARIABLES              # von Wissen
 Class_S (INTEGER),
 Income_S (REAL)

SENSOR INDICATORS
 INIT_S,                      # von Wahrnehmung
 MOVED_S,                     # von Wissen
 UPDATE_INCOME_S,             # von Wissen
 RESET_MOVED_S                # von Wahrnehmung

DYNAMIC BEHAVIOUR

#-------------------------------------------------------------------------------
# Signal von Wissen: Weitergabe der Klassennummer
#-------------------------------------------------------------------------------
ON INIT_S DO
  Class^   := Class_S;
END

#-------------------------------------------------------------------------------
# Signal von Wissen: Weitergabe des aktuellen Einkommens
#-------------------------------------------------------------------------------
ON UPDATE_INCOME_S DO
  Income^ := Income_S;
END

#-------------------------------------------------------------------------------
# Protokollierung der Bewegung
#-------------------------------------------------------------------------------
ON MOVED_S
DO
  moved^ := 1;
END

#-------------------------------------------------------------------------------
# Reset des Bewegungsstatus
#-------------------------------------------------------------------------------
ON RESET_MOVED_S
DO
  moved^ := 0;
END

END OF Status
```

A.3.7 Die Komponente *Verhalten*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell Solidarnetzwerk
#
#  Name:             Verhalten
#
#  Art:              Basiskomponente
#
```

```
#  Version:          1
#
#  Beschreibung:     Verhaltenssteuerung der Agenten
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Verhalten

DECLARATION OF ELEMENTS

TRANSITION INDICATORS

#-------------------------------------------------------------------------------
# Ausführungsanordnungen an die Komponente Aktor
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Initialisierung beendet
#-------------------------------------------------------------------------------
 DONE_INIT,

#-------------------------------------------------------------------------------
# Einkommen berechnen fuer moeglichen Umzug
#-------------------------------------------------------------------------------
 CALCULATE_INCOME,

#-------------------------------------------------------------------------------
# bestes Feld in Migrationsfenster suchen
#-------------------------------------------------------------------------------
 FIND_BEST_POSITION,

#-------------------------------------------------------------------------------
# SupportGame
#-------------------------------------------------------------------------------
 CALC_INCOME_FOR_PLAY,

#-------------------------------------------------------------------------------
# SupportGame beendet
#-------------------------------------------------------------------------------
 DONE_PLAY,

#-------------------------------------------------------------------------------
# Umzug abgeschloseen
#-------------------------------------------------------------------------------
 DONE_SET,

#-------------------------------------------------------------------------------
# Agent will nicht umziehen, der naechste ist dran.
#-------------------------------------------------------------------------------
 NOTMIGR

SENSOR INDICATORS
 DONE_INIT_S,
 DONE_SET_S,
 DONE_PLAY_S,
 I_STAY_S,
 PLAY_S,
 MIGRAT_S

DYNAMIC BEHAVIOUR

#-------------------------------------------------------------------------------
# Initialisierungsphase ist abgeschlossen
#-------------------------------------------------------------------------------
ON DONE_INIT_S DO
 SIGNAL DONE_INIT; # an Aktor
END
```

```
#-------------------------------------------------------------------------------
# Signal an Aktor, der soll Wissen zur Einkommensberechnung
# bzw. zur Suche nach bester Position auffordern
#-------------------------------------------------------------------------------
ON MIGRAT_S DO
 SIGNAL CALCULATE_INCOME;
 SIGNAL FIND_BEST_POSITION;
END

#-------------------------------------------------------------------------------
# Agent will nicht umziehen (ist zufrieden, oder nichts frei)
#-------------------------------------------------------------------------------
ON I_STAY_S DO
 SIGNAL NOTMIGR;
END

#-------------------------------------------------------------------------------
# Signal nach Aussen: Position wurde verändert
#-------------------------------------------------------------------------------
ON DONE_SET_S
DO
 SIGNAL DONE_SET;
END

#-------------------------------------------------------------------------------
# Signal von der Umwelt: Supportgame spielen
#-------------------------------------------------------------------------------
ON PLAY_S DO
 SIGNAL CALC_INCOME_FOR_PLAY; #an Aktor
END

#-------------------------------------------------------------------------------
# Meldung des Agenten nach aussen: SupportGame ist zu Ende
#-------------------------------------------------------------------------------
ON DONE_PLAY_S DO
 SIGNAL DONE_PLAY; # an Aktor
END

END OF Verhalten
```

A.3.8 Die Komponente *Aktor*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell Solidarnetzwerk
#
#  Name:             Aktor
#
#  Art:              Basiskomponente
#
#  Version:          1
#
#  Beschreibung:     Ausführung von Aktionen
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Aktor

DECLARATION OF ELEMENTS

STATE VARIABLES
 DoneGame (LOGICAL) := FALSE,
 TSignal (REAL)     := 9999
```

```
SENSOR VARIABLES
 NewRow_S(INTEGER),
 NewColumn_S(INTEGER)


TRANSITION INDICATORS
 DONE_INIT,
 CALCULATE_INCOME,   # an Wissen
 FIND_BEST_POSITION, # an Wissen
 DONEMIGR,
 ONLY_INCOME,        # an Wissen
 DONEPLAY,
 NOTMIGR,            # an Umwelt
 INTERN1

SENSOR INDICATORS
 CALCULATE_INCOME_S,   # an Wissen
 FIND_BEST_POSITION_S, # an Wissen
 DONE_INIT_S,                # von Verhaltn
 NOTMIGR_S,            # von Verhaltn
 INCOME_FOR_PLAY_S,    # von Verhaltn
 DONEPL_S ,                        # von Verhaltn
 DONE_SET_S                        # von Verhalten

DYNAMIC BEHAVIOUR

#-------------------------------------------------------------------------------
# Rueckmeldung nach aussen : Initialisierung beendet
#-------------------------------------------------------------------------------
ON DONE_INIT_S DO
 SIGNAL DONE_INIT;  # an Umwelt
END

#-------------------------------------------------------------------------------
# Verhalten hat Anstoss gegeben zur Strategie: Berechne Einkommen
# wird an Wissen weitergeleitet. Dort findet Berechnung statt
#-------------------------------------------------------------------------------
ON CALCULATE_INCOME_S DO
 SIGNAL CALCULATE_INCOME;
END

#-------------------------------------------------------------------------------
# Verhalten hat Anstoss gegeben zur Strategie:
# Finde die beste Position in der Umgebung.
# wird an Wissen weitergeleitet. Dort findet Berechnung statt
#-------------------------------------------------------------------------------
ON FIND_BEST_POSITION_S DO
 SIGNAL FIND_BEST_POSITION;
END

#-------------------------------------------------------------------------------
# Rueckmeldung nach aussen : Agent ist umgezogen
#-------------------------------------------------------------------------------
ON DONE_SET_S DO
 SIGNAL DONEMIGR;
END

#-------------------------------------------------------------------------------
# Rueckmeldung nach aussen : Agent wollte nicht umziehen
#-------------------------------------------------------------------------------
ON NOTMIGR_S DO
 SIGNAL NOTMIGR;
END

#-------------------------------------------------------------------------------
# Support Game: Einkommensberechnung
#-------------------------------------------------------------------------------
```

```
ON INCOME_FOR_PLAY_S DO
  SIGNAL ONLY_INCOME; # an Wissen
END

#-------------------------------------------------------------------------------
# Synchronisierung der Agenten am Rundenende
#-------------------------------------------------------------------------------
ON DONEPL_S DO
 DoneGame^ := TRUE;
 TSignal^  := T + 0.75;
END

#-------------------------------------------------------------------------------
# spaetestens nach 9999 Runden ist das Spiel zu Ende
#-------------------------------------------------------------------------------
WHENEVER (DoneGame = TRUE) AND (T > TSignal) DO
 DoneGame^ := FALSE;
 SIGNAL DONEPLAY;
END

END OF Aktor
```

A.4 Die mobile Komponente *AgentRef*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell Solidarnetzwerk
#
#  Name:             AgentRef
#
#  Art:              mobile Komponente
#
#  Version:          1
#
#  Beschreibung:     mobile Komponente, die eine Referenz auf einen
#                    Agenten enthält (wird für die Reihenfolgebestimmung
#                    in den einzelnen Spielrunden benötigt
#
#-------------------------------------------------------------------------------

MOBILE COMPONENT AgentRef

DECLARATION OF ELEMENTS

STATE VARIABLES
DISCRETE
 WaitPos (INTEGER) := 0,
 Id      (INTEGER) := 0,
 Played  (INTEGER) := 0

END OF AgentRef
```

ANHANG B QUELLCODE DES MARKTMODELLS

B.1 Die Strukturkomponente *Market*

```
#-------------------------------------------------------------------------
#
# Name:  "Market"
#
# Art:   High-Level Komponente
#
# Beschreibung:
#
#   - Die Komponente fuer das gesamte Modell. Dabei werden die entsprech-
#     Anzahlen 'NProd' bzw. 'NCons' von Producern bzw. Consumern erzeugt.
#
#
# Verbindung(en) mit anderen Komponenten: n/a
#
#-------------------------------------------------------------------------

HIGH LEVEL COMPONENT Market

DIMENSIONS

  #---------------------------------------------
  # Obergrenze fuer die Anzahl an Consumern
  #---------------------------------------------
  NMaxCons := 200,

  #---------------------------------------------
  # Obergrenze fuer die Anzahl an Producern
  #---------------------------------------------
  NMaxProd := 50,

  #---------------------------------------------
  # Anzahl an Consumern : 1 <= NCons <= NMaxCons
  #---------------------------------------------
  NCons    := 16,

  #---------------------------------------------
  # Anzahl an Producern : 1 <= NProd <= NMaxProd
  #---------------------------------------------
  NProd    := 2


SUBCOMPONENTS

  ARRAY [NMaxCons] Buyer  OF CLASS Consumer,  # Instanzen der Consumer-Klasse
  ARRAY [NMaxProd] Seller OF CLASS Producer,  # Instanzen der Producer-Klasse
  Connector,                                  # Connector
  Control,                                    # Steuerung der Marktphasen
  Statistic                                   # Statistik


COMPONENT CONNECTIONS

  #---------------------------------------------
  # Connector-Verbindungen fuer Producer
  #---------------------------------------------
  Connector.InP{ALL i}     --> Seller[i].InP;
  Connector.OutP{ALL i}    --> Seller[i].OutP;

  #---------------------------------------------
  # Connector-Verbindungen fuer Consumer
  #---------------------------------------------
  Connector.OutC{ALL i}    --> Buyer[i].OutC;
  Connector.InC{ALL i}     --> Buyer[i].InC;
```

```
  Control.MState         --> (
                              Seller{ALL}.MState,
                              Buyer{ALL}.MState,
                              Statistic.MState
                             );

  Statistic.MInfRdy      --> Seller{ALL}.MInfoReady;

  Seller{ALL i}.Quantity --> Statistic.Quantity[i];
  Seller{ALL i}.Price    --> Statistic.Price[i];


  Statistic.MShares{i OF 1..NMaxProd}   --> Seller{ALL}.MShares[i];
  Statistic.Prices{i OF 1..NMaxProd}    --> Seller{ALL}.Prices[i];



INITIALIZE

  #-------------------------------------------------
  # Initialisierungen fuer die Statisticik-Komponente
  #-------------------------------------------------
  Statistic.NProd    := NProd;

  #-------------------------------------------------
  # Initialisierungen fuer die Seller-Komponenten
  #-------------------------------------------------
  Seller{ALL i}.Id  := i;
  Seller{ALL}.NCons := NCons;
  Seller{ALL}.NProd := NProd;
  Seller{ALL i | 1 <= i AND i <= NProd}.Active := TRUE;

  #-------------------------------------------------
  # Initialisierungen fuer die Buyer-Komponenten
  #-------------------------------------------------
  Buyer{ALL i}.Id := i;
  Buyer{ALL i}.KnownProd{ALL j | 1 <= j AND j <= NProd} := TRUE;
  Buyer{ALL i | 1 <= i AND i <= NCons}.Active := TRUE;

END OF Market
```

B.2 Die *Producer*-Agenten

B.2.1 Die Strukturkomponente *Producer*

```
#-------------------------------------------------------------------------------
#
# Name:  "Producer"
#
# Art:   High-Level Komponente
#
# Beschreibung:
#
#   - Diese Komponenten fasst alle angegebenen Subkomponenten zu
#     einem Producer zusammen.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - Connector, Statistic
#
#-------------------------------------------------------------------------------
```

```
HIGH LEVEL COMPONENT Producer

SUBCOMPONENTS

  PSensor,
  PPerception,
  PCognition,
  PBehaviour,
  PActor,
  PStatus

INPUT CONNECTIONS

  Id          --> (
                   PSensor.Id,
                   PPerception.Id,
                   PCognition.Id,
                   PBehaviour.Id,
                   PActor.Id,
                   PStatus.Id
                  );
  Active      --> (
                   PSensor.Active,
                   PPerception.Active,
                   PCognition.Active,
                   PBehaviour.Active,
                   PActor.Active,
                   PStatus.Active
                  );
  OutP        --> PSensor.OutP;
  InP         --> PActor.InP;
  NCons       --> PCognition.NCons;
  MState      --> PSensor.MState;
  MInfoReady  --> PSensor.MInfoReady;
  MShares     --> PCognition.MShares;
  Prices      --> (
                   PCognition.Prices,
                   PBehaviour.Prices
                  );
  NProd       --> (
                   PCognition.NProd,
                   PBehaviour.NProd
                  );

OUTPUT EQUIVALENCES

  Quantity := PStatus.Quantity;
  Price    := PStatus.Price;
  Prot_S   := PSensor.Protocol;
  Prot_P   := PPerception.Protocol;
  Prot_C   := PCognition.Protocol;
  Prot_B   := PBehaviour.Protocol;
  Prot_A   := PActor.Protocol;
  Prot_St  := PStatus.Protocol;

COMPONENT CONNECTIONS

  PSensor.PTSensed            --> PPerception.PTSensed;
  PPerception.PTPerceived     --> PCognition.PTPerceived;
  PCognition.ActPrice         --> PBehaviour.ActPrice;
  PCognition.NewAskArrived    --> PBehaviour.NewAskArrived;
  PCognition.NewOrderArrived  --> PBehaviour.NewOrderArrived;
  PCognition.OrderReceived    --> PBehaviour.OrderReceived;
  PCognition.AskReceived      --> PBehaviour.AskReceived;
  PCognition.CalcNewPrice     --> PBehaviour.CalcNewPrice;
  PCognition.NPeriod          --> PBehaviour.NPeriod;
  PCognition.MShare           --> PBehaviour.MShare;
```

```
  PCognition.Profit           --> PBehaviour.Profit;
  PCognition.LastProfit       --> PBehaviour.LastProfit;
  PCognition.LowLimit         --> PBehaviour.LowLimit;
  PCognition.MaxCoopProfit    --> PBehaviour.MaxCoopProfit;
  PCognition.DirProfit        --> PBehaviour.DirProfit;
  PCognition.DirProfitReady   --> PBehaviour.DirProfitReady;
  PCognition.CompStrat{ALL i} --> PBehaviour.CompStrat[i];
  PCognition.ActPrice         --> PStatus.ActPrice;
  PCognition.TotalQty         --> PStatus.TotalQty;
  PBehaviour.PTSelfGenerated  --> PCognition.PTSelfGenerated;
  PBehaviour.PActionRequest   --> PActor.PActionRequest;

END OF Producer
```

B.2.2 Die Komponente *PSensor*

```
#-------------------------------------------------------------------------
#
# Name:  "PSensor" (Producer-Sensor)
#
# Art:   Basis-Komponente
#
# Beschreibung:
#
#   - Diese Komponente holt sich abholbereite Nachrichten aus dem
#     Connector und wandelt sie in 'PThoughts' (Producer thoughts) um.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - PPerception, Connector
#
#-------------------------------------------------------------------------

BASIC COMPONENT PSensor

MOBILE SUBCOMPONENTS OF CLASS Message, PThought

LOCAL DEFINITIONS

DECLARATION OF ELEMENTS

 STATE VARIABLES
  DISCRETE
  Protocol (LOGICAL) := FALSE

 SENSOR VARIABLES
  DISCRETE
  MState (LOGICAL) := FALSE,
  Active (LOGICAL) := FALSE,
  Id     (INTEGER)

 SENSOR INDICATORS
  MInfoReady

 LOCATIONS
  # Bereich fuer empfangene Messages
  MessageReceived (Message) := 0 Message,

  # Bereich fuer in producer thoughts (PThoughts) umgewandelte Messages
  PTSensed (PThought) := 0 PThought

 SENSOR LOCATIONS
  # Ausgangsbereich dieses Producers im Connector
  OutP (Message)
```

```
DYNAMIC BEHAVIOUR

IF Active DO

#-------------------------------------------------------------------------
#
# Ereignis 1 : Sobald im Ausgangsbereich Nachrichten fuer diesen Producer
#              vorliegen, werden diese nach MessageReceived transferiert.
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(OutP) > 0
DO
  MessageReceived^ : FROM OutP GET Message{ALL};

  IF Protocol
  DO
   DISPLAY("T=%.2f : Prod %d (PSensor) -> ", T, Id);
   DISPLAY("Retrieving %d message(s) from the connector\n", NUMBER(OutP));
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 2 : Empfangene Nachrichten werden in PThoughts umgewandelt
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(MessageReceived) > 0
DO
  IF MessageReceived:Message[1].Type = 'CAsk'
  DO
    PTSensed^ : ADD 1 NEW PThought
                CHANGING
                 Sender^   := MessageReceived:Message[1].Sender;
                 Type^     := 'PTAsk';
                 TStamp^   := T;
                 Deadline^ := MessageReceived:Message[1].Deadline;
                END
  END

  ELSIF MessageReceived:Message[1].Type = 'COrder'
  DO
    PTSensed^ : ADD 1 NEW PThought
                CHANGING
                 Sender^   := MessageReceived:Message[1].Sender;
                 Type^     := 'PTOrder';
                 TStamp^   := T;
                 Deadline^ := MessageReceived:Message[1].Deadline;
                 Price^    := MessageReceived:Message[1].Price;
                 Quantity^ := MessageReceived:Message[1].Quantity;
                END
  END

  ELSE
  DO
    DISPLAY("Error Prod %d (PSensor) : Sensed unknown message type!\n",Id);
  END

  MessageReceived^ : REMOVE Message[1];
END

#-------------------------------------------------------------------------
#
# Ereignis 3 : Markt ist geoeffnet worden
#
#-------------------------------------------------------------------------
```

```
ON ^MState = TRUE^
DO
  PTSensed^ : ADD 1 NEW PThought
              CHANGING
                Type^   := 'PTMOpen';
                TStamp^ := T;
              END

  IF Protocol
  DO
    DISPLAY("T=%.2f : Prod %d (PSensor) -> Sensed market begin\n",
              T, Id);
  END

END

#-------------------------------------------------------------------------
#
# Ereignis 4 : Markt ist geschlossen worden
#
#-------------------------------------------------------------------------

ON ^MState = FALSE^
DO
  PTSensed^ : ADD 1 NEW PThought
              CHANGING
                Type^   := 'PTMClose';
                TStamp^ := T;
              END

  IF Protocol
  DO
    DISPLAY("T=%.2f : Prod %d (PSensor) -> Sensed market close-down\n",
              T, Id);
  END

END

#-------------------------------------------------------------------------
#
# Ereignis 5 : Informationen ueber die vergangene Marktperiode liegen
#              nun in Statistic bereit.
#
#-------------------------------------------------------------------------

ON MInfoReady
DO
  PTSensed^ : ADD 1 NEW PThought
              CHANGING
                Type^   := 'PTInfRdy';
                TStamp^ := T;
              END
  IF Protocol
  DO
    DISPLAY("T=%.2f : Prod %d (PSensor) -> Sensed market info ",T,Id);
    DISPLAY("now available\n");
  END
END

END # IF Active

END OF PSensor
```

B.2.3 Die Komponente *PPerception*

```
#-------------------------------------------------------------------------------
#
# Name:  "PPerception" (Producer-Perceptionion)
#
# Art:   Basis-Komponente
#
# Beschreibung:
#
#   - Diese Komponente leitet 'PThoughts' aus "PSensor" unverzerrt an
#     "PCognit" weiter.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - PSensor, PCognit
#
#-------------------------------------------------------------------------------

BASIC COMPONENT PPerception

MOBILE SUBCOMPONENTS OF CLASS PThought

DECLARATION OF ELEMENTS

  STATE VARIABLES
   DISCRETE
   Protocol(LOGICAL) := FALSE

  SENSOR VARIABLES
   DISCRETE
   Active (LOGICAL) := FALSE,
   Id     (INTEGER)

  LOCATIONS
   # Producer-thoughts perceived (Wahrgenommene Perzepte eines Producers)
   PTPerceived (PThought) := 0 PThought

  SENSOR LOCATIONS
   # Producer-thoughts sensed (In PSensor registrierte PThoughts)
   PTSensed (PThought)

DYNAMIC BEHAVIOUR

IF Active DO

#-------------------------------------------------------------------------------
#
# Ereignis 1 : In PSensor generierte PThoughts werden nach PTPerceived geholt.
#
#-------------------------------------------------------------------------------

WHENEVER NUMBER(PTSensed) > 0
DO
  PTPerceived^ : FROM PTSensed GET PThought{ALL};

  IF Protocol
  DO
    DISPLAY("T=%.2f : Prod %d (PPerception) -> Perceiving %d thought(s)\n",
             T, Id, NUMBER(PTSensed));
  END
END

END # IF Active

END OF PPerception
```

B.2.4 Die Komponente *PCognition*

```
#-------------------------------------------------------------------------
#
# Name:  "PCognition" (Producer-Cognitionion)
#
# Art:   Basis-Komponente
#
# Beschreibung:
#
#   - Diese Komponente stellt das elementare Wissens des Producers dar.
#     'PThoughts' aus "PPercept" und "PBehav" werden verwertet. Die
#     Marktsteuerung ist auch bekannt und verlaeuft analog zu "CCognition".
#
#   - Weiter werden Signale 'NewAskArrived' und 'NewOrderArrived'
#     ausgeloest, damit die Verhaltenskomponente weiss, wann sie in
#     Aktion treten soll.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - PPercept, PBehav, PStatus
#
#-------------------------------------------------------------------------

BASIC COMPONENT PCognition

MOBILE SUBCOMPONENTS OF CLASS PThought, PriceEntry

LOCAL DEFINITIONS
  DIMENSIONS
   NMaxCons := 200,
   NMaxProd := 50

  VALUE SET StraType  : ('cut', 'cooplead', 'coopfoll')

  VALUE SET Direction : ('dontknow', 'better', 'worse', 'optimum')


  #-----------------------------------------------------------------
  # Eine Funktion die ueberprueft, ob die Preise aller Anbieter gleich
  # waren.
  #-----------------------------------------------------------------
  FUNCTION AllEqual (ARRAY [n] REAL : Prices, INTEGER : Num --> LOGICAL)
  DECLARE iPrice (REAL)
  BEGIN
    iPrice := Prices[1];

    FOR i FROM 1 TO Num
    REPEAT
      IF iPrice <> Prices[i] DO RETURN (FALSE); END
    END_LOOP

    RETURN (TRUE);
  END_FUNC


  #-----------------------------------------------------------------
  # Eine Funktion die ueberprueft, ob der Preis <SearchPrice> in einem
  # Eintrag <Price> einer mob. Komp. vorkommt. Falls ja, wird der Index
  # der mob. Komp. zurueckgegeben. Ansonsten ist der Rueckgabewert -1
  #-----------------------------------------------------------------
  FUNCTION FindIndex (LOCATION FOR PriceEntry : SearchLoc, REAL : SearchPrice
                      --> INTEGER)
  DECLARE length (INTEGER)
  BEGIN
    length := NUMBER(SearchLoc);
```

```
    IF length = 0 DO
      RETURN (-1); END

    ELSE
    DO
      FOR i FROM 1 TO length
      REPEAT
        IF SearchLoc:PriceEntry[i].Price = SearchPrice
          DO RETURN (i); END
      END_LOOP

      RETURN (-1);
    END
   END_FUNC


  #-------------------------------------------------------------------
  # Eine Funktion, die TRUE zurueckliefert, wenn mind. ein Eintrag im
  # Array Competitor 'coopfoll' oder 'cooplead' enthaelt.
  #-------------------------------------------------------------------
  FUNCTION FCoopExists(ARRAY [n] StraType : Competitor, INTEGER : Num
                       --> LOGICAL)
  BEGIN
    FOR i FROM 1 TO Num
    REPEAT
      IF Competitor[i] <> 'cut'
      DO
        RETURN (TRUE);
      END
    END_LOOP

    RETURN (FALSE);
  END_FUNC

  DECLARATION OF ELEMENTS
  CONSTANTS

    kLowLimit (REAL)    := 10,   # Preisuntergrenze, die nicht unterschritten
                                 # werden soll. (z.B. = kUnitCost)

    kMaxAsk (INTEGER)   := 100,  # Max. Anzahl an Preisanfragen, die gespeichert
                                 # werden koennen.

    kMaxOrder (INTEGER) := 100,  # Max. Anzahl an Bestellungen, die ge-
                                 # speichert werden.

#   kInitPrice (REAL)   := 40.0, # Startpreis fuer die 1. Marktperiode

    kUnitCost (REAL)    := 10.0, # Stueckkosten

    kLowMShare (REAL)   := 0.4,

    kHighMShare (REAL)  := 1.6



  STATE VARIABLES
    DISCRETE

    MState (LOGICAL)  := FALSE,  # Marktzustand:
                                 # TRUE  = "Markt geoeffnet"
                                 # FALSE = "Markt geschlossen"

    NPeriod (INTEGER) := 0,      # Nummer der Marktperiode
                                 # (wird fortlaufend durchnumeriert)
```

```
 ActPrice (REAL)   := 0.0,    # Aktuell festgelegter Preis

 LowLimit (REAL)   := 0.0,    # Preisuntergrenze

 # <Asked> speichert, welche Consumer eine Preisanfrage gestellt haben
 ARRAY [NMaxCons] Asked (LOGICAL) := FALSE,

 # <Ordered> speichert, welche Consumer bestellt haben
 ARRAY [NMaxCons] Ordered (LOGICAL) := FALSE,

 # Acked verwaltet die bestatigten Bestellungen der Marktperiode
 ARRAY [NMaxCons] Acked (LOGICAL) := FALSE,

 # Offered speichert die Consumer, denen eine Preisangebot gemacht wurde
 ARRAY [NMaxCons] Offered (REAL) := -1.0,

 # <TotalQty> enthaelt die in der Marktperiode abgesetzte Menge
 TotalQty (REAL) := 0.0,

 # Marktanteil
 MShare (REAL) := -1.0,

 # Preis der letzten Marktperiode
 PreviousPrice (REAL) := -1.0,

 # Gewinn der letzten Marktperiode
 Profit   (REAL) := 0.0,

 # Gewinn der vorletzten Marktperiode
 LastProfit (REAL) := 0.0,

 # Maximaler Gewinn bei Preisgleichheit
 MaxCoopProfit (REAL) := 0.0,

 # Gewinnentwicklung
 DirProfit (Direction) := 'dontknow',

 Debug (LOGICAL) := FALSE,

 # Protokollvariable, die Marktinformationen ein- und ausblendet
 Protocol (LOGICAL) := FALSE,

 CoopFound (LOGICAL) := FALSE,

 # Vermerk über die Strategien der Konkurrenten
 ARRAY [NMaxProd] CompStrat (StraType) := 'cut',

 ARRAY [NMaxProd] CopyOfPrices (REAL) := 0.0,

 ARRAY [NMaxProd] PrevPrices (REAL) := 0.0,

 ARRAY [NMaxProd] CopyOfMShares (REAL) := 0.0,

 ARRAY [NMaxProd] PrevMShares (REAL) := 0.0


SENSOR VARIABLES
 DISCRETE
 Active (LOGICAL) := FALSE,
 Id     (INTEGER),
 NProd  (INTEGER),
 NCons  (INTEGER),
 ARRAY [NMaxProd] MShares (REAL), # Marktanteile aller Produzenten
                                  # in der letzten Marktperiode

 ARRAY [NMaxProd] Prices  (REAL)  # Preise aller Produzenten aus der
                                  # letzten Marktperiode
```

```
  TRANSITION INDICATORS

   NewAskArrived,
   NewOrderArrived,
   CalcNewPrice,
   MInfoReady,
   ReviewCompStrat,
   DirProfitReady,
   DirProfitReq

  LOCATIONS

   # PThoughts aus PPercept werden in "Producer thought known" gespeichert.
   PTKnown   (PThought) := 0 PThought,

   # PThoughts, die Preisanfragen entsprochen haben, werden hier gespeichert.
   AskReceived (PThought) := 0 PThought,

   # PThoughts, die Bestellungen entsprochen haben, werden hier gesammelt.
   OrderReceived (PThought) := 0 PThought,

   # PThoughts aus PBehav werden hierhin transferiert
   PTUpdate    (PThought) := 0 PThought,

   # Enthaelt die Preis/Profit-Paare vergangener Marktperioden, in denen
   # alle Anbieter den gleichen Preis hatten
   EqualPrices (PriceEntry ORDERED BY INC Price) := 0 PriceEntry

  SENSOR LOCATIONS

   PTPerceived (PThought),    # verbunden mit Location in PPercept
   PTSelfGenerated (PThought) # verbunden mit Location in PBehav

DYNAMIC BEHAVIOUR

IF Active DO

#-------------------------------------------------------------------------------
#
# Ereignis 1 : ActPrice wird mit dem Wert aus kInitPrice vorbesetzt.
#
#-------------------------------------------------------------------------------

ON START
DO
#  ActPrice^ := kInitPrice;
  LowLimit^ := kLowLimit;
END

#-------------------------------------------------------------------------------
#
# Ereignis 2 : Ruecksetzen der Variablen fuer die neue Periode, sobald
#              <MState> von FALSE auf TRUE wechselt.
#
#-------------------------------------------------------------------------------

ON ^MState = TRUE^
DO
  # Neue Marktperiode beginnt
  NPeriod^      := NPeriod + 1;
  TotalQty^     := 0.0;
  Asked{ALL}^   := FALSE;
  Ordered{ALL}^ := FALSE;
  Acked{ALL}^   := FALSE;
  Offered{ALL}^ := -1.0;
END
```

```
#-------------------------------------------------------------------------------
#
# Ereignis 3 : Protokollausgabe
#
#-------------------------------------------------------------------------------

ON ^MState = FALSE^
DO
  IF Debug DO
  DISPLAY("T=%.2f : Prod %d (PCognition) -> O.K. so its closed. Now what?\n",
           T, Id);
  END
END

#-------------------------------------------------------------------------------
#
# Ereignis 4 : Ankommende PThoughts werden registriert und nach PTKnown
#              kopiert.
#
#-------------------------------------------------------------------------------

WHENEVER NUMBER(PTPerceived) > 0
DO
  PTKnown^ : FROM PTPerceived GET PThought{ALL};
END

#-------------------------------------------------------------------------------
#
# Ereignis 5 : Ankommende Thoughts aus PPercept werden ihrem Typ ent-
#              sprechend interpretiert. Damit wird die Wissensbasis
#              aktualisiert.
#
#-------------------------------------------------------------------------------

WHENEVER NUMBER(PTKnown) > 0
DO
  IF PTKnown:PThought[1].Type = 'PTAsk'
  DO
  #-------------------------------------------------------------
  # Preisanfragen werden nach AskReceived weitergeleitet
  #-------------------------------------------------------------
    PTKnown^ : TO AskReceived SEND PThought[1];

    SIGNAL NewAskArrived;

    IF Debug DO
      DISPLAY("T=%.2f : Prod %d (PCognition) -> Received query ",T,Id);
      DISPLAY("from Consumer %d\n", PTKnown:PThought[1].Sender);
    END
  END

  ELSIF PTKnown:PThought[1].Type = 'PTOrder'
  DO
  #-------------------------------------------------------------
  # Bestellungen werden nach OrderReceived weitergeleitet
  #-------------------------------------------------------------
    PTKnown^ : TO OrderReceived SEND PThought[1];

    SIGNAL NewOrderArrived;

    IF Debug
    DO
      DISPLAY("T=%.2f : Prod %d (PCognition) -> Received order ",T, Id);
      DISPLAY("from Consumer %d", PTKnown:PThought[1].Sender);
      DISPLAY("\t(Qty = %.2f)\n", PTKnown:PThought[1].Quantity);
   END
  END
```

```
  ELSIF PTKnown:PThought[1].Type = 'PTMOpen'
  DO
  #-------------------------------------------------------------
  # Marktbeginn wurde wahrgenommen => MState wird TRUE
  #-------------------------------------------------------------
    MState^ := TRUE;
    PTKnown^ : REMOVE PThought[1];

    IF Debug
    DO
      DISPLAY("T=%.2f : Prod %d (PCognition) -> Market is open now\n",
               T,Id);
    END
  END

  ELSIF PTKnown:PThought[1].Type = 'PTMClose'
  DO
  #-------------------------------------------------------------
  # Marktende wurde wahrgenommen => MState wird FALSE
  #-------------------------------------------------------------
    MState^ := FALSE;
    PTKnown^ : REMOVE PThought[1];

    IF Debug
    DO
      DISPLAY("T=%.2f : Prod %d (PCognition) -> Market is closed now\n",
               T,Id);
    END
  END

  ELSIF PTKnown:PThought[1].Type = 'PTInfRdy'
  DO
  #-------------------------------------------------------------
  # Statistikinformationen liegen nun in <Statistic> bereit
  #-------------------------------------------------------------
    SIGNAL MInfoReady;
    PTKnown^ : REMOVE PThought[1];

    IF Debug
    DO
      DISPLAY("T=%.2f : Prod %d (PCognition) -> Overall ",T,Id);
      DISPLAY("market information now available\n");
    END
  END


  #-------------------------------------------------------------
  # Hier besteht Raum fuer zusaetzliche PThought-Typen...
  #-------------------------------------------------------------

  ELSE
  DO
    PTKnown^ : REMOVE PThought[1];
    DISPLAY("Error Prod %d (PCognition) -> Unknown thought type!\n",Id);
  END
END

#-------------------------------------------------------------------------------
#
# Ereignis 6 : Sobald mehr als kMaxAsk Preisanfragen eingetroffen sind,
#              wird der erste PThought in AskReceived geloescht.
#
#-------------------------------------------------------------------------------

WHENEVER NUMBER(AskReceived) > kMaxAsk
DO
  AskReceived^ : REMOVE PThought[1];
```

```
  IF Debug
  DO
    DISPLAY("T=%.2f : Prod %d (PCognition) -> Cant remember more than ",
             T, Id);
    DISPLAY("%d queries\n", kMaxAsk);
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 7 : Sobald mehr als kMaxOrder Bestellungen eingetroffen sind,
#              wird der erste PThought in OrderReceived geloescht
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(OrderReceived) > kMaxOrder
DO
  OrderReceived^ : REMOVE PThought[1];

  IF Debug
  DO
    DISPLAY("T=%.2f : Prod %d (PCognition) -> Cant remember more than ",
             T, Id);
    DISPLAY("%d orders\n", kMaxOrder);
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 8 : Wenn das Signal NewAskArrived gesetzt wird, merkt sich der
#              Producer die Id des Consumers, der angefragt hat.
#
#-------------------------------------------------------------------------

ON NewAskArrived
DECLARE lastitem (INTEGER)
DO PROCEDURE
  lastitem := NUMBER(AskReceived);
END
DO TRANSITIONS
  #-------------------------------------------------------------
  # Der Eintrag des Arrays <Asked> an der Stelle mit der Id
  # des Consumers wird auf TRUE gesetzt.
  #-------------------------------------------------------------
  Asked[AskReceived:PThought[lastitem].Sender]^ := TRUE;
END

#-------------------------------------------------------------------------
#
# Ereignis 9 : Wenn das Signal <NewOrderArrived> gesetzt wird, merkt sich
#              der Producer die bestellte Menge, indem er sie zu TotalQty
#              hinzuaddiert.
#
#-------------------------------------------------------------------------

ON NewOrderArrived
DECLARE lastitem(INTEGER)
DO PROCEDURE
  lastitem := NUMBER(OrderReceived);
END
DO TRANSITIONS
  #-------------------------------------------------------------
  # Der Eintrag des Arrays <Ordered> an der Stelle mit der Id
  # des Consumers wird auf TRUE gesetzt.
  #-------------------------------------------------------------
  Ordered[OrderReceived:PThought[lastitem].Sender]^ := TRUE;
```

```
  # Addiere die bestellte Menge zur Gesamtmenge hinzu
  TotalQty^ := TotalQty + OrderReceived:PThought[lastitem].Quantity;
END

#-------------------------------------------------------------------------
#
# Ereignis 10 : In PBehav generierte PThoughts werden nach PTUpdate geholt.
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(PTSelfGenerated) > 0
DO
  PTUpdate^ : FROM PTSelfGenerated GET PThought{ALL};
END

#-------------------------------------------------------------------------
#
# Ereignis 11 : Empfangene PToughts von PBehav werden ihrem Typ entsprechend
#               verarbeitet.
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(PTUpdate) > 0
DO
  IF PTUpdate:PThought[1].Type = 'PTOffUpd'
  DO
    Offered[PTUpdate:PThought[1].Id]^ := PTUpdate:PThought[1].Price;
  END

  ELSIF PTUpdate:PThought[1].Type = 'PTAckUpd'
  DO
    Acked[PTUpdate:PThought[1].Id]^ := TRUE;
  END

  ELSIF PTUpdate:PThought[1].Type = 'PTPrcUpd'
  DO
    ActPrice^ := PTUpdate:PThought[1].Price;

    PreviousPrice^ := ActPrice;
  END

  ELSIF PTUpdate:PThought[1].Type = 'PTFindDir'
  DO
    #-----------------------------------------------------------------
    # PBehav verlangt ein update von DirProfit, da die 'cooplead'-
    # Strategie diesen Wert erfordert.
    #-----------------------------------------------------------------
    SIGNAL DirProfitReq;

    IF Protocol
    DO
      DISPLAY("T=%.2f : Prod %d (PCognition) -> ", T, Id);
      DISPLAY("PBehav requested direction of profits!\n");
    END
  END

  ELSE
  DO
    DISPLAY("T=%.2f : Error Prod %d (PCognition) -> Didnt understand ",T, Id);
    DISPLAY("PThought from PBehav!\n");
  END

  PTUpdate^ : REMOVE PThought[1];
END
```

```
#-------------------------------------------------------------------------
#
# Ereignis 12 : Das Signal <MInfoReady> wird gesetzt, wenn ein PThought
#               vom Typ 'PTInfRdy' wahrgenommen wurde.
#
#-------------------------------------------------------------------------

ON MInfoReady
DECLARE newprofit (REAL)
DO PROCEDURE
  #-----------------------------
  # Neuer Gewinn wird berechnet
  #-----------------------------
  newprofit := (ActPrice - kUnitCost) * TotalQty;
END

DO TRANSITIONS

  PrevMShares{ALL i}^ := CopyOfMShares[i];
  PrevPrices{ALL i}^ := CopyOfPrices[i];
  CopyOfMShares{ALL i}^ := MShares[i];
  CopyOfPrices{ALL i}^ := Prices[i];

  #-------------------------------------------------
  # Eigener Marktanteil wird in MShare gespeichert.
  #-------------------------------------------------
  MShare^ := MShares[Id];

  #-----------------------------------------
  # Alter Gewinn wird in LastProfit gespeichert
  #-----------------------------------------
  LastProfit^ := Profit;

  #-------------------------------
  # Neuer Gewinn wird gespeichert
  #-------------------------------
  Profit^ := newprofit;

  IF Protocol
  DO
    DISPLAY("---------------------------------------\n");
    DISPLAY("Market data (period %d): Prod %d \n", NPeriod, Id);
    DISPLAY("\tMarket share: %.1f\%\n", MShares[Id]*100);
    DISPLAY("\tPrice       : %.2f\n", ActPrice);
    DISPLAY("\tQuantity    : %.2f\n", TotalQty);
    DISPLAY("\tProfit      : %.2f\n", newprofit);
    DISPLAY("---------------------------------------\n");
  END

  IF AllEqual(ARRAY Prices, NProd)
  DO
    IF Protocol
    DO
      DISPLAY("T=%.2f : Prod %d (PCognition) -> All prices are the same! ",
               T, Id);
      DISPLAY("(price=%.2f/profit=%.2f)\n",
               ActPrice, newprofit);
    END

    IF FindIndex (LOCATION EqualPrices, ActPrice) = -1
    DO
      IF newprofit > MaxCoopProfit
      DO
        MaxCoopProfit^ := newprofit;
      END

      EqualPrices^ : ADD 1 NEW PriceEntry
```

```
                    CHANGING
                      Price^  := ActPrice;
                      Profit^ := newprofit;
                    END

      IF Protocol
      DO
        DISPLAY("T=%.2f : Prod %d (PCognition) -> Adding new price/profit ",
                 T, Id);
        DISPLAY("pair (price=%.2f/profit=%.2f)\n", ActPrice, newprofit);
      END
    END

    ELSE # FindIndex (...) <> -1
    DO
    EqualPrices:PriceEntry[FindIndex(LOCATION EqualPrices, ActPrice)].Profit^
               := newprofit;
    END
  END

  #-------------------------------------------------------------------
  # Signal zur Analyse der Strategien der Mitbewerber wird gesetzt
  #-------------------------------------------------------------------
  SIGNAL ReviewCompStrat;
END

#-------------------------------------------------------------------------------
#
# Ereignis 13 : Die Aktionen der Mitbewerber muessen analysiert werden und
#               jedem Mitbewerber muss eine der drei Strategien 'cut',
#               'coopfoll' oder 'cooplead' zugeordnet werden.
#
#-------------------------------------------------------------------------------

ON ReviewCompStrat
DECLARE
  ARRAY [NMaxProd] swstrat (LOGICAL) := FALSE,
  coopexists (LOGICAL) := FALSE,
  eqsplit (REAL)
DO PROCEDURE
  IF NPeriod > 1
  DO
    eqsplit := 1 / NProd;

    coopexists := FCoopExists(ARRAY CompStrat, NProd);

    FOR i FROM 1 TO NProd
    REPEAT
      IF i <> Id
      DO
        IF CompStrat[i] = 'cut' AND PrevPrices[i] <> LowLimit
        DO
          IF PrevMShares[i] > kHighMShare * eqsplit AND
             Prices[i] > PrevPrices[i]
          DO
            swstrat[i] := FALSE;
          END

          ELSIF PrevMShares[i] > eqsplit AND Prices[i] = PrevPrices[i]
          DO
            swstrat[i] := FALSE;
          END

          ELSIF PrevMShares[i] <= eqsplit AND Prices[i] < PrevPrices[i]
          DO
            swstrat[i] := FALSE;
          END
```

```
          ELSE
          DO
            swstrat[i] := TRUE;
          END
        END
      END
    END_LOOP
  END
END

DO TRANSITIONS

  IF Protocol
  DO
    IF swstrat[1]
     DO DISPLAY ("Prod %d : Producer %d switched\n",Id,1); END

    IF swstrat[2]
     DO DISPLAY ("Prod %d : Producer %d switched\n",Id,2); END
  END

  IF coopexists
  DO
    CompStrat{ALL i | swstrat[i] = TRUE}^ := 'coopfoll';
  END

  ELSE
  DO
    CompStrat{ALL i | swstrat[i] = TRUE}^ := 'cooplead';
  END

  #-------------------------------------------------------
  # Signal zum Berechnen eines neuen Preises wird gesetzt
  #-------------------------------------------------------
  SIGNAL CalcNewPrice;
END

#-------------------------------------------------------------------------------
#
# Ereignis 14 :
#
#-------------------------------------------------------------------------------

ON DirProfitReq
DECLARE smallind (INTEGER) := 0,
        largeind (INTEGER) := 0,
        smallval (REAL) := -1.0,
        largeval (REAL) := -1.0
DO PROCEDURE
  smallind := FindIndex (LOCATION EqualPrices, ActPrice - 1.0);
  largeind := FindIndex (LOCATION EqualPrices, ActPrice + 1.0);

  IF smallind <> -1
  DO
    smallval := EqualPrices:PriceEntry[smallind].Profit;
  END

  IF largeind <> -1
  DO
    largeval := EqualPrices:PriceEntry[largeind].Profit;
  END
END

DO TRANSITIONS
  IF smallind <> -1 AND largeind <> -1
  DO
```

```
  IF Profit > smallval AND Profit > largeval
  DO
    DirProfit^ := 'optimum';
    IF Protocol
    DO
      DISPLAY("T=%.2f : Prod %d (PCognition) -> ", T, Id);
      DISPLAY("%.2f < Profit (%.2f) > %.2f\n", smallval, Profit, largeval);
    END
  END

  ELSIF Profit > smallval AND Profit < largeval
  DO
    DirProfit^ := 'better';
    IF Protocol
    DO
      DISPLAY("T=%.2f : Prod %d (PCognition) -> ", T, Id);
      DISPLAY("%.2f < Profit (%.2f) < %.2f\n", smallval, Profit, largeval);
    END
  END

  ELSIF Profit < smallval AND Profit > largeval
  DO
    DirProfit^ := 'worse';
    IF Protocol
    DO
      DISPLAY("T=%.2f : Prod %d (PCognition) -> ", T, Id);
      DISPLAY("%.2f > Profit (%.2f) > %.2f\n", smallval, Profit, largeval);
    END
  END

 ELSE # IF Profit = smallval AND Profit = largeval
 DO
   DirProfit^ := 'dontknow';
    IF Protocol
    DO
      DISPLAY("T=%.2f : Error Prod %d (PCognition) -> ", T, Id);
      DISPLAY("Couldnt find appropriate case for direction of profits\n");
    END
 END
END

ELSIF smallind <> -1 AND largeind = -1
DO
  IF Profit >= smallval
  DO
    DirProfit^ := 'better';
    IF Protocol
    DO
      DISPLAY("T=%.2f : Prod %d (PCognition) -> ", T, Id);
      DISPLAY("%.2f <= Profit (%.2f)\n", smallval, Profit);
    END
  END

  ELSE # Profit < smallval
  DO
    DirProfit^ := 'worse';
    IF Protocol
    DO
      DISPLAY("T=%.2f : Prod %d (PCognition) -> ", T, Id);
      DISPLAY("%.2f > Profit (%.2f)\n", smallval, Profit);
    END
  END
END

ELSIF smallind = -1 AND largeind <> -1
DO
  IF Profit > largeval
```

```
    DO
      DirProfit^ := 'worse';
      IF Protocol
      DO
        DISPLAY("T=%.2f : Prod %d (PCognition) -> ", T, Id);
        DISPLAY("Profit (%.2f) > %.2f\n", Profit, largeval);
      END
    END

    ELSE # Profit <= largeval
    DO
      DirProfit^ := 'better';
      IF Protocol
      DO
        DISPLAY("T=%.2f : Prod %d (PCognition) -> ", T, Id);
        DISPLAY("Profit (%.2f) <= %.2f\n", Profit, largeval);
      END
    END
  END

  ELSIF smallind = -1 AND largeind = -1
  DO
    DirProfit^ := 'dontknow';
    IF Protocol
    DO
      DISPLAY("T=%.2f : Prod %d (PCognition) -> ", T, Id);
      DISPLAY("smallval and largeval unknown!");
    END
  END

  SIGNAL DirProfitReady;
END

END # IF Active

END OF PCognition
```

B.2.5 Die Komponente *PStatus*

```
#-------------------------------------------------------------------------------
#
# Name:  "PStatus" (Producer-Status)
#
# Art:   Basis-Komponente
#
# Beschreibung:
#
#   - Diese Komponente enthaelt Preis und umgesetzte Menge in der
#     vorangegangen Marktperiode. Sie wird von Statist(ic) ausgewertet.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - PCognit
#
#-------------------------------------------------------------------------------

BASIC COMPONENT PStatus

DECLARATION OF ELEMENTS

 STATE VARIABLES
  DISCRETE
  Protocol (LOGICAL) := FALSE
```

```
 DEPENDENT VARIABLES
  DISCRETE
  Quantity (REAL)    := 0.0,
  Price    (REAL)    := 0.0

 SENSOR VARIABLES
  DISCRETE
  TotalQty   (REAL)    := 0.0,
  ActPrice (REAL)    := 0.0,
  Active   (LOGICAL) := FALSE,
  Id       (INT)


DYNAMIC BEHAVIOUR

IF Active
DO
  #-------------------------------------------------
  # Die Werte von TotalQty und ActPrice werden nach
  # Quantity und Price kopiert.
  #-------------------------------------------------
  Quantity := TotalQty;
  Price    := ActPrice;
END

ELSE # NOT(Active)
DO
  Quantity :=  0.0;
  Price    :=  0.0;
END

END OF PStatus
```

B.2.6 Die Komponente *PBehaviour*

```
#-------------------------------------------------------------------------------
#
# Name:  "PBehaviour" (Producer-Behaviouriour)
#
# Art:   Basis-Komponente
#
# Beschreibung:
#
#   - Diese Komponente implementiert das Verhalten des Producers.
#
#   - Bei Preisanfragen werden Preisangebote verschickt und bei
#     Bestellungen werden diese sofort bestaetigt.
#
#   - Nach Ende des Marktes wird eine Preisberechnung fuer die
#     naechste Periode ausgefuehrt.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - PCognit, PActor
#
#-------------------------------------------------------------------------------

BASIC COMPONENT PBehaviour

MOBILE SUBCOMPONENTS OF CLASS PAction, PThought

LOCAL DEFINITIONS
  DIMENSIONS
  NMaxCons := 200,  # Anzahl Consumer
```

```
NMaxProd := 50    # Anzahl Producer

# Aufzaehlungstyp fuer die verschiedenen Strategien
VALUE SET StraType   : ('cut', 'cooplead', 'coopfoll')

# Aufzaehlungstyp fuer <DirProfit>
VALUE SET Direction  : ('dontknow', 'better', 'worse', 'optimum')

# Aufzaehlungstyp fuer Persoenlichkeitsmerkmale
VALUE SET PersonType : ('calm', 'nervous')

#-------------------------------------------------------------------------
# Funktion, die den aktuellen Preis auf das Niveau bis 3 Preiseinheiten
# unter dem groessten Preis anhebt. Keinesfalls wird aber der Preis
# ActPrice erniedrigt oder unter den Wert von LoLim gesetzt.
#-------------------------------------------------------------------------
FUNCTION FRaise2 (ARRAY [n] REAL : Prices, INTEGER : Num,
                  REAL : LoLim, REAL : ActPrice --> REAL)
DECLARE MaxPrc (REAL) := 0.0
BEGIN
# Find maximum price
  FOR i FROM 1 TO Num
  REPEAT
   IF Prices[i] > MaxPrc
   DO MaxPrc := Prices[i]; END
  END_LOOP

  RETURN (MAX(MAX(MaxPrc - 3, LoLim),ActPrice));
END_FUNC

#-------------------------------------------------------------------------
# Funktion, die den aktuellen Preis ActPrice bis auf 3 Preiseinheiten
# ueber dem niedrigsten Preis aller Anbieter setzt. Keinesfalls wird
# jedoch LoLim unterschritten oder der Preis erhoeht.
#-------------------------------------------------------------------------
FUNCTION FLower2 (ARRAY [n] REAL : Prices, INTEGER : Num,
                  REAL : LoLim, REAL : ActPrice --> REAL)
DECLARE MinPrc (REAL) := 0.0
BEGIN
  # Find minimum price
  FOR i FROM 1 TO Num
  REPEAT
    IF Prices[i] < MinPrc OR i=1 DO
      MinPrc := Prices[i]; END
  END_LOOP

  RETURN (MIN(MAX(MinPrc+3,LoLim), ActPrice));
END_FUNC

#-------------------------------------------------------------------------
# Funktion, die TRUE zurueckliefert, wenn mind. ein Eintrag in
# Strategy gleich 'cut' ist.
#-------------------------------------------------------------------------
FUNCTION FCutExists (ARRAY [n] StraType : Strategy, INTEGER : Num,
                     INTEGER : self --> LOGICAL)
BEGIN
  FOR i FROM 1 TO Num
  REPEAT
    IF i <> self
    DO
      IF Strategy[i] = 'cut'
      DO RETURN (TRUE); END
    END
  END_LOOP
RETURN (FALSE);
END_FUNC
```

```
  #-------------------------------------------------------------------
  # Funktion, die TRUE zuruecckliefert, wenn mind. ein Eintrag in
  # Strategy gleich 'ccoplead' ist.
  #-------------------------------------------------------------------
  FUNCTION FCoopLeadExists (ARRAY [n] StraType : Strategy, INTEGER : Num,
                            INTEGER : self --> LOGICAL)
  BEGIN
    FOR i FROM 1 TO Num
    REPEAT
      IF i <> self
      DO
        IF Strategy[i] = 'cooplead'
        DO RETURN (TRUE); END
      END
    END_LOOP
  RETURN (FALSE);
  END_FUNC

  #-------------------------------------------------------------------
  # Funktion, die den kleinsten Wert aus einem REAL-Array PriceList
  # im Indexbereich von 1 bis Num findet
  #-------------------------------------------------------------------
  FUNCTION FRetaliate (ARRAY [n] REAL : PriceList, INTEGER : Num,
                       REAL : LowerLimit --> REAL)
  DECLARE minprice (REAL) := 0.0
  BEGIN
    FOR i FROM 1 TO Num
    REPEAT
      IF PriceList[i] < minprice OR i = 1
      DO minprice := PriceList[i]; END
    END_LOOP

    RETURN (MAX(minprice,LowerLimit));
  END_FUNC

DECLARATION OF ELEMENTS

 CONSTANTS

  kcutTh  (REAL) := 0.0,    # kcutTh * InitCredit =
                            # Schwellwert zum Verlassen der 'cut'-Strategie

  kcoopTh (REAL) := -2.0,   # kcoopTh * InitCredit =
                            # Schwellert zum Erneuern der 'coop'-Strategie

  kposFac (REAL) := 0.3,    # Faktor zum Erhoehen der Glaubwuerdigkeit

  knegFac (REAL) := 1.0,    # Faktor zum Erniedrigen d. Glaubwuerdigkeit

  kretalFac (REAL) := -0.3,# kretalFac * InitCredit =
                            # Schwellwert zum Ausloesen eines Retaliate,
                            # falls die Strategie 'cooplead' ist.

  kLowMShare (REAL) := 0.4,

  kHighMShare (REAL) := 1.6

 STATE VARIABLES
 DISCRETE

  # Verfolgter Strategietyp
  Strategy (StraType) := 'cut',

  # Verbliebenes Vertrauen in die gewaehlte Strategie
  Credit   (REAL) := 0.0,

  InitCredit (REAL) := 100.0,
```

```
 Personality (PersonType) := 'calm',
 Protocol (LOGICAL) := FALSE,
 Debug    (LOGICAL) := FALSE


SENSOR VARIABLES
DISCRETE

 Active   (LOGICAL) := FALSE,
 ActPrice (REAL),              # Zuletzt gueltiger Preis der letzten
                               # Marktperiode
 Id       (INTEGER),
 NPeriod  (INTEGER),           # Nummer der Marktperiode
 MShare   (REAL)    := -1.0,   # Marktanteil der letzten Marktperiode
 NProd    (INTEGER),           # Anzahl Producer
 Profit     (REAL),  # Gewinn der vorangeganen Marktperiode (aus PCognit)
 LastProfit (REAL),  # Gewinn der vorletzten Marktperiode (aus PCognit)
 LowLimit   (REAL),  # Preisuntergrenze (aus PCognit)
 MaxCoopProfit (REAL), # Maximal bislang erreichter Gewinn bei Preis-
                       # gleichheit (aus PCognit)

 # Richtung, in der sich der sich der gemeinsame Gewinn bewegt
 DirProfit (Direction) := 'dontknow',

 # Vermutete Strategien der Konkurrenten
 ARRAY [NMaxProd] CompStrat (StraType),

 ARRAY [NMaxProd] Prices (REAL) # Preise aller Produzenten aus der vergangenen
                                # Marktperiode

TRANSITION INDICATORS

  Action,
  Keep,
  Lower1,
  Lower2,
  Raise1,
  Raise2,
  PriceJump,
  Retaliate

SENSOR INDICATORS
 # Neue Preisanfrage ist eingetroffen
 NewAskArrived,

 # Neue Bestellung ist eingetroffen
 NewOrderArrived,

 # PCognit moechte einen neuen Preis fuer die naechste Periode bestimmen
 CalcNewPrice,

 # Die von der 'cooplead'-Strategie angeforderte Information ist
 # verfuegbar
 DirProfitReady

LOCATIONS
 # Location, in der Auftraege fuer den Actor bereitliegen
 PActionRequest (PAction)  := 0 PAction,

 # Location, in der generierte PThoughts fuer Cognition bereitliegen
 PTSelfGenerated (PThought) := 0 PThought

SENSOR LOCATIONS
 # Location mit Preisanfragen aus PCognit
 AskReceived (PThought),
```

```
  # Location mit Bestellungen aus PCognit
  OrderReceived (PThought)

DYNAMIC BEHAVIOUR

IF Active DO

#-------------------------------------------------------------------------
#
# Ereignis 1 : Anfaengliche Glaubwuerdigkeit wird aus der Sensorvariable
#              InitCredit nach Credit kopiert.
#
#-------------------------------------------------------------------------

ON START
DO
  Credit^   := InitCredit;
END

#-------------------------------------------------------------------------
#
# Ereignis 2 : Abwickeln von Preisanfragen. Signal <NewAskArrived> wurde
#              in PCognit gesetzt.
#
#-------------------------------------------------------------------------

ON NewAskArrived
DECLARE lastitem (INTEGER)
DO PROCEDURE
  lastitem := NUMBER(AskReceived);
END

DO TRANSITIONS
  PActionRequest^ : ADD 1 NEW PAction
                    CHANGING
                     Receiver^ := AskReceived:PThought[lastitem].Sender;
                     Type^     := 'PAOffer';
                     TStamp^   := T;
                     Price^    := ActPrice;
                    END

  PTSelfGenerated^ : ADD 1 NEW PThought
                      CHANGING
                        Type^ := 'PTOffUpd';
                        Id^   := AskReceived:PThought[lastitem].Sender;
                      END
END

#-------------------------------------------------------------------------
#
# Ereignis 3 : Bestaetigung einer Bestellung
#
#-------------------------------------------------------------------------

ON NewOrderArrived
DECLARE lastitem (INTEGER)
DO PROCEDURE
  lastitem := NUMBER(OrderReceived);
END

DO TRANSITIONS
  PActionRequest^ : ADD 1 NEW PAction
                    CHANGING
                     Receiver^ := OrderReceived:PThought[lastitem].Sender;
                     Type^     := 'PAAck';
                     TStamp^   := T;
                     Price^    := OrderReceived:PThought[lastitem].Price;
```

```
                    Quantity^ := OrderReceived:PThought[lastitem].Quantity;
                   END

  PTSelfGenerated^ : ADD 1 NEW PThought
                    CHANGING
                      Type^     := 'PTAckUpd';
                      Id^       := OrderReceived:PThought[lastitem].Sender;
                      Price^    := OrderReceived:PThought[lastitem].Price;
                      Quantity^ := OrderReceived:PThought[lastitem].Quantity;
                     END
END

#-------------------------------------------------------------------------
#
# Ereignis 4 : Aktualisierung der ZV <Strategy> und <Credit>
#
#
#-------------------------------------------------------------------------

ON CalcNewPrice
DECLARE
  coopleadexists (LOGICAL)
DO PROCEDURE
  coopleadexists := FCoopLeadExists (ARRAY CompStrat, NProd, Id);
END

DO TRANSITIONS
IF NPeriod > 1
DO
  IF Debug
  DO
    DISPLAY("T=%.2f : Prod %d (PBehaviour) --> Reviewing strategy ",T,Id);
    DISPLAY("(old credit=%.2f, ",Credit);

    IF Strategy = 'cut'
      DO DISPLAY("strategy='cut') "); END
    ELSIF Strategy = 'coopfoll'
      DO DISPLAY("strategy='coopfoll') "); END
    ELSIF Strategy = 'cooplead'
      DO DISPLAY("strategy='cooplead') "); END
  END

  IF Strategy = 'cut'
  DO
    #-----------------------------------------------------------------------
    # Ist die Glaubwuerdigkeit fuer 'cut' noch ueberhalb des Schwellwerts?
    #-----------------------------------------------------------------------
    IF Credit > kcutTh * InitCredit
    DO
      #-------------------------------------------------------------
      # Hat sich der Gewinn gegenueber dem vorherigen Gewinn verbessert?
      #-------------------------------------------------------------
      IF Profit > LastProfit
      DO
        #---------------------------------------
        # => Strategie bleibt bei 'cut'
        # => Credit wird um einen Faktor erhoeht
        #---------------------------------------
        Credit^ := Credit + kposFac * (Profit - LastProfit);

        IF Debug
        DO
          DISPLAY("'cut'-strategy successful (new credit=%.2f)\n",
                    Credit + kposFac * (Profit - LastProfit));
        END
      END
      ELSE # Profit <= LastProfit
```

```
    DO
      #-----------------------------------------------------------------
      # Gewinn ist also im Vergleich zum vorherigen Gewinn gesunken
      # (oder gleichgeblieben)
      # => Strategie bleibt bei 'cut'
      # => Credit wird um einen Faktor erniedrigt
      #-----------------------------------------------------------------
      Credit^ := Credit + knegFac * (Profit - LastProfit);

      IF Debug
      DO
        DISPLAY("'cut'-strategy unsuccessful (new credit=%.2f)\n",
                Credit + knegFac * (Profit - LastProfit));
      END
    END
  END

  ELSE # Credit <= kcutTh * InitCredit
  DO
    #-----------------------------------------------------------------
    # Hier wird in Betracht gezogen, wie die Strategie
    # der Konkurrenten aussieht.
    #-----------------------------------------------------------------
    Credit^   := InitCredit;

    IF NOT(coopleadexists)
    DO
      Strategy^ := 'cooplead';

      IF Debug
      DO
        DISPLAY("switching to 'cooplead' (new credit = %.2f)!\n",
                InitCredit);
      END
    END

    ELSE # Es gibt bereits einen 'coop-leader'
    DO
      Strategy^ := 'coopfoll';

      IF Debug
      DO
        DISPLAY("switching to 'coopfoll' (new credit = %.2f)!\n",
                InitCredit);
      END
    END
  END
END

ELSE # Strategy = 'cooplead' OR 'coopfoll'
DO
  IF Credit > kcoopTh * InitCredit
  DO
    IF Profit > MaxCoopProfit
    DO
      Credit^ := Credit + kposFac * (Profit - MaxCoopProfit);

      IF Debug
      DO
        IF Strategy = 'cooplead'
          DO DISPLAY("'cooplead'-"); END
        ELSE
          DO DISPLAY("'coopfoll'-"); END
        DISPLAY("strategy successful (new credit=%.2f)\n",
                Credit + kposFac * (Profit - MaxCoopProfit));
      END
    END
```

```
      ELSE # Profit <= MaxCoopProfit
      DO
        Credit^ := Credit + knegFac * (Profit - MaxCoopProfit);

        IF Debug
        DO
          IF Strategy = 'cooplead'
            DO DISPLAY("'cooplead'-"); END
          ELSE
            DO DISPLAY("'coopfoll'-"); END
          DISPLAY("strategy unsuccessful (new credit=%.2f)\n",
                  Credit + knegFac * (Profit - MaxCoopProfit));
        END
      END
    END

    ELSE # Credit <= kcoopTh * InitCredit
    DO
      Credit^ := InitCredit;

      IF Debug
      DO
        DISPLAY("Resetting 'coop(foll/lead)'-strategy\n");
      END
    END
  END

END # NPeriod > 1

  #-------------------------------------------------------
  # Signalisiere die Bereitschaft, eine Aktion auszuwaehlen
  #-------------------------------------------------------
  SIGNAL Action;
END

#-------------------------------------------------------------------------------
#
# Ereignis 5 : Preisbestimmung
#
#-------------------------------------------------------------------------------

ON Action
#-------------------------------------------------------------------------------
# eqsplit ist der Marktanteil, der den Markt in gleiche Stuecke aufteilt
#-------------------------------------------------------------------------------
DECLARE
  eqsplit (REAL),
  cutthroatexists (LOGICAL)

DO PROCEDURE
  eqsplit := 1/NProd;
  cutthroatexists := FCutExists (ARRAY CompStrat, NProd, Id);
END

DO TRANSITIONS

  IF Debug
  DO
    DISPLAY("T=%.2f : Prod %d (PBehaviour) -> ", T, Id);
  END

  #-----------------------------------------------------------------------
  # Strategie fuer einen 'cut-throat'-producer (Marktanteil um jeden Preis)
  #-----------------------------------------------------------------------
  IF Strategy = 'cut'
  DO
    IF Debug
```

```
DO
  DISPLAY("'cut'-strategy => ");
END

IF ActPrice = LowLimit AND MShare > kHighMShare * eqsplit
   AND Personality = 'nervous'
DO
  SIGNAL Raise2;
  IF Debug
  DO
    DISPLAY("(rule 1) ");
    DISPLAY("Actprice reached LowLimit, market share is very high ");
    DISPLAY("and personality is 'nervous'\n");
  END
END

ELSIF ActPrice = LowLimit
DO
  SIGNAL Raise1;
  IF Debug
  DO
    DISPLAY("(rule 2) ");
    DISPLAY("Actprice reached LowLimit\n");
  END
END

ELSIF MShare > kHighMShare * eqsplit AND Personality = 'nervous'
DO
  SIGNAL Raise2;
  IF Debug
  DO
    DISPLAY("(rule 3) ");
    DISPLAY("Market share is very high and personality is 'nervous'\n");
  END
END

ELSIF MShare > kHighMShare * eqsplit AND Personality = 'calm'
DO
  SIGNAL Raise1;
  IF Debug
  DO
    DISPLAY("(rule 4) ");
    DISPLAY("Market share is very high and personality is 'calm'\n");
  END
END

ELSIF MShare > eqsplit
DO
  SIGNAL Keep;
  IF Debug
  DO
    DISPLAY("(rule 5) ");
    DISPLAY("Market share is high\n");
  END
END

ELSIF MShare = eqsplit
DO
  SIGNAL Lower1;
  IF Debug
  DO
    DISPLAY("(rule 6) ");
    DISPLAY("Market share is even\n", eqsplit);
  END
END

ELSIF MShare < kLowMShare * eqsplit
```

```
  DO
    SIGNAL Lower2;
    IF Debug
    DO
      DISPLAY("(rule 8) ");
      DISPLAY("Market share is very low\n");
    END
  END

  ELSIF MShare < eqsplit
  DO
    SIGNAL Lower1;
    IF Debug
    DO
      DISPLAY("(rule 7) ");
      DISPLAY("Market share is low\n");
    END
  END
END

#-----------------------------------------------------
# Strategie fuer einen 'cooperation-follower'-producer
#-----------------------------------------------------
ELSIF Strategy = 'coopfoll'
DO
  IF Debug
  DO
    DISPLAY("'coopfoll'-strategy => ");
  END

  IF MShare > kHighMShare * eqsplit AND Personality = 'nervous'
  DO
    SIGNAL Raise2;
    IF Debug
    DO
      DISPLAY("(rule 22) ");
      DISPLAY("Market share is very high and personality is 'nervous'\n");
    END
  END

  IF MShare > kHighMShare * eqsplit AND Personality = 'calm'
  DO
    SIGNAL Raise2;
    IF Debug
    DO
      DISPLAY("(rule 23) ");
      DISPLAY("Market share is very high and personality is 'calm'\n");
    END
  END

  ELSIF MShare > eqsplit
  DO
    SIGNAL Raise1;
    IF Debug
    DO
      DISPLAY("(rule 24) ");
      DISPLAY("Market share is high\n");
    END
  END

  ELSIF MShare = eqsplit
  DO
    SIGNAL Keep;
    IF Debug
    DO
      DISPLAY("(rule 25) ");
      DISPLAY("Market share is even\n");
```

```
    END
  END

  ELSIF MShare < kLowMShare * eqsplit
  DO
    SIGNAL Lower2;
    IF Debug
    DO
      DISPLAY("(rule 27) ");
      DISPLAY("Market share is very low\n");
    END
  END

  ELSIF MShare < eqsplit
  DO
    SIGNAL Lower1;
    IF Debug
    DO
      DISPLAY("(rule 26) ");
      DISPLAY("Market share is low\n");
    END
  END
END

#-----------------------------------------------------
# Strategie fuer einen 'cooperation-leader'-producer
#-----------------------------------------------------
ELSIF Strategy = 'cooplead'
DO
  IF Debug
  DO
    DISPLAY("'cooplead'-strategy => ");
  END

  IF ActPrice = LowLimit AND cutthroatexists AND
     Credit = InitCredit AND Personality = 'nervous'
  DO
    SIGNAL PriceJump;
    IF Debug
    DO
      DISPLAY("(rule 9) ");
      DISPLAY("Actprice reached LowLimit, 'cut' exists, credit = initial");
      DISPLAY(" and personality = 'nervous'\n");
    END
  END

  ELSIF ActPrice = LowLimit
  DO
    SIGNAL Raise1;
    IF Debug
    DO
      DISPLAY("(rule 10) ");
      DISPLAY("Actprice reached LowLimit\n");
    END
  END

  ELSIF cutthroatexists AND Credit = InitCredit AND
        Personality = 'nervous'
  DO
    SIGNAL PriceJump;
    IF Debug
    DO
      DISPLAY("(rule 11) ");
      DISPLAY("'cut' exists, credit = initial");
      DISPLAY(" and personality = 'nervous'\n");
    END
  END
```

```
ELSIF cutthroatexists AND Credit < kretalFac * InitCredit
DO
  SIGNAL Retaliate;
  IF Debug
  DO
    DISPLAY("(rule 12) ");
    DISPLAY("'cut' exists and credit below retaliate threshold\n");
  END
END

ELSIF MShare > kHighMShare * eqsplit AND Personality = 'nervous'
DO
  SIGNAL Raise2;
  IF Debug
  DO
    DISPLAY("(rule 13) ");
    DISPLAY("Market share is very high");
    DISPLAY(" and personality = 'nervous'\n");
  END
END

ELSIF MShare > kHighMShare * eqsplit AND Personality = 'calm'
DO
  SIGNAL Raise1;
  IF Debug
  DO
    DISPLAY("(rule 14) ");
    DISPLAY("Market share is very high");
    DISPLAY(" and personality = 'calm'\n");
  END
END

ELSIF MShare > eqsplit
DO
  SIGNAL Keep;
  IF Debug
  DO
    DISPLAY("(rule 15) ");
    DISPLAY("Market share is high\n");
  END
END

ELSIF MShare < kLowMShare * eqsplit
DO
  SIGNAL Lower1;
  IF Debug
  DO
    DISPLAY("(rule 21) ");
    DISPLAY("Market share is very low\n");
  END
END

ELSIF MShare < eqsplit
DO
  SIGNAL Keep;
  IF Debug
  DO
    DISPLAY("(rule 20) ");
    DISPLAY("Market share is low\n");
  END
END

ELSIF MShare = eqsplit
DO
```

```
      #-------------------------------------------------------------------
      # Hier wird Wissen von PCognit angefordert:
      # Die Richtung, in der sich die gemeinsamen Gewinne bewegen fehlt...
      #-------------------------------------------------------------------
      PTSelfGenerated^ : ADD 1 NEW PThought
                         CHANGING
                           Type^ := 'PTFindDir';
                         END
      IF Debug
      DO
        DISPLAY("Market share is even. ");
        DISPLAY("Thinking about direction of profits\n");
      END
    END

    ELSE
    DO
      DISPLAY("T=%.2f : Error Prod %d (PCognit) -> ",T, Id);
      DISPLAY("'coopleader' found no suitable action!\n");
    END
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 6 : Preisbestimmung
#
#-------------------------------------------------------------------------

ON DirProfitReady
DO
  IF Debug
  DO
    DISPLAY("T=%.2f : Prod %d (PBehaviour) -> DirProfit ready : ", T, Id);
    IF DirProfit = 'dontknow'
      DO DISPLAY("(rule 16) 'dontknow'\n"); END
    ELSIF DirProfit = 'better'
      DO DISPLAY("(rule 17) 'better'\n"); END
    ELSIF DirProfit = 'worse'
      DO DISPLAY("(rule 18) 'worse'\n"); END
    ELSIF DirProfit = 'optimum'
      DO DISPLAY("(rule 19) 'optimum'\n"); END
  END

  IF DirProfit = 'dontknow'
  DO
    SIGNAL Raise1;
  END

  ELSIF DirProfit = 'better'
  DO
    SIGNAL Raise1;
  END

  ELSIF DirProfit = 'worse'
    DO
      SIGNAL Lower1;
    END

  ELSIF DirProfit = 'optimum'
   DO
     SIGNAL Keep;
   END
END
```

```
#-------------------------------------------------------------------------------
#
# Ereignis 7 : Aktion fuer 'Keep' (Preis beibehalten)
#
#-------------------------------------------------------------------------------

ON Keep
DO
  PTSelfGenerated^ : ADD 1 NEW PThought
           CHANGING
             Type^ := 'PTPrcUpd';
             Price^ := ActPrice;
           END

  IF Protocol
  DO
    DISPLAY("T=%.2f : Prod %d (PBehaviour) -> Keeping price at same level!\n",
             T, Id);
  END
END

#-------------------------------------------------------------------------------
#
# Ereignis 8 : Aktion fuer 'Raise1' (Preis um einen Punkt hochsetzen)
#
#-------------------------------------------------------------------------------

ON Raise1
DO
  PTSelfGenerated^ : ADD 1 NEW PThought
           CHANGING
             Type^ := 'PTPrcUpd';
             Price^ := ActPrice + 1;
           END

  IF Protocol
  DO
    DISPLAY("T=%.2f : Prod %d (PBehaviour) -> Raising price 1 point!\n",
             T, Id);
  END
END

#-------------------------------------------------------------------------------
#
# Ereignis 9 : Aktion fuer 'Raise2' (Preis - wenn moeglich - bis drei Punkte
#              unter das Niveau des hoechstliegenden hochsetzen)
#
#-------------------------------------------------------------------------------

ON Raise2
DO
  PTSelfGenerated^ : ADD 1 NEW PThought
           CHANGING
             Type^ := 'PTPrcUpd';
             Price^ := FRaise2(ARRAY Prices, NProd, LowLimit, ActPrice);
           END
  IF Debug
  DO
   DISPLAY("T=%.2f : Prod %d (PBehaviour) --> Raise2 = %.2f\n",
            T, Id, FRaise2(ARRAY Prices, NProd, LowLimit, ActPrice));
  END

  IF Protocol
  DO
    DISPLAY("T=%.2f : Prod %d (PBehaviour) -> Raising price to 3 points below ",
             T, Id);
    DISPLAY("overall maximum price! (if possible)\n");
```

```
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 10 : Aktion fuer 'Lower1' (Preis um einen Punkt herabsetzen)
#
#-------------------------------------------------------------------------

ON Lower1
DO
  PTSelfGenerated^ : ADD 1 NEW PThought
           CHANGING
             Type^ := 'PTPrcUpd';
             Price^ := ActPrice - 1;
           END

  IF Protocol
  DO
    DISPLAY("T=%.2f : Prod %d (PBehaviour) -> Lowering price 1 point!\n",
             T, Id);
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 11 : Aktion fuer 'Lower2' (Preis - wenn moeglich - bis auf drei
#               Punkte ueber dem niedrigsten Preis herabsetzen)
#
#-------------------------------------------------------------------------

ON Lower2
DO
  PTSelfGenerated^ : ADD 1 NEW PThought
           CHANGING
             Type^ := 'PTPrcUpd';
             Price^ := FLower2(ARRAY Prices, NProd, LowLimit, ActPrice);
           END

  IF Debug
  DO
    DISPLAY("T=%.2f : Prod %d (PBehaviour) --> Lower2 = %.2f\n",
             T, Id, FLower2(ARRAY Prices, NProd, LowLimit,ActPrice));
  END

  IF Protocol
  DO
    DISPLAY("T=%.2f : Prod %d (PBehaviour) -> Lowering price to 3 points ",
             T, Id);
    DISPLAY("above overall minimum price! (if possible)\n");
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 12 : Aktion fuer 'PriceJump' (Preissprung gemaess FPriceJump)
#
#-------------------------------------------------------------------------

ON PriceJump
DECLARE newprice(REAL)
DO PROCEDURE
  newprice := IRND(ActPrice*(1 + ((NPeriod+1)/75)));
END

DO TRANSITIONS
```

```
  PTSelfGenerated^ : ADD 1 NEW PThought
                     CHANGING
                       Type^ := 'PTPrcUpd';
                       Price^ := newprice;
                     END

  IF Protocol
  DO
    DISPLAY("T=%.2f : Prod %d (PBehaviour) -> Price jump to %.2f\n", T, Id,
             newprice);
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 13 : Aktion fuer 'Retaliate' (Preis bis auf das Niveau des
#               niedrigsten Produzenten herabsetzen)
#
#-------------------------------------------------------------------------

ON Retaliate
DECLARE newprice (REAL)
DO PROCEDURE
  newprice := FRetaliate (ARRAY Prices, NProd, LowLimit);
END

DO TRANSITIONS

  PTSelfGenerated^ : ADD 1 NEW PThought
                     CHANGING
                       Type^ := 'PTPrcUpd';
                       Price^ := newprice;
                     END

  IF Protocol
  DO
    DISPLAY("T=%.2f : Prod %d (PBehaviour) -> Retaliate to %.2f\n", T, Id,
             newprice);
  END

END

END # IF Active

END OF PBehaviour
```

B.2.7 Die Komponente *PActor*

```
#-------------------------------------------------------------------------
#
# Name:  "PActor" (Producer-Actor)
#
# Art:   Basis-Komponente
#
# Beschreibung:
#
#   - Anstehende CAction-Anweisungen aus "PBehav" werden verarbeitet,
#     indem "Messages" vom passenden Typ erzeugt und anschliessend
#     an den Connector verschickt werden.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - PBehaviour, indirekt zu Connector
#-------------------------------------------------------------------------
```

```
BASIC COMPONENT PActor

MOBILE SUBCOMPONENTS OF CLASS Message, PAction

DECLARATION OF ELEMENTS

  STATE VARIABLES
   DISCRETE
   Protocol (LOGICAL) := FALSE

  SENSOR VARIABLES
   DISCRETE
   Active (LOGICAL) := FALSE,
   Id (INTEGER)

  LOCATIONS
   # Bereich fuer zu sendende Nachrichten
   MSend (Message) := 0 Message,

   # Bereich fuer empfangene Action requests aus PBehav
   PActionRecvd (PAction) := 0 PAction

  SENSOR LOCATIONS
   # Eingangsbereich fuer diesen Producer im Connector
   InP (Message),

   # Producer Action Request
   PActionRequest (PAction)

DYNAMIC BEHAVIOUR

IF Active DO

#-------------------------------------------------------------------------
#
# Ereignis 1 : Liegen in PBehav PAction-Anweisungen bereit, werden diese
#              nach PActionRecvd geholt.
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(PActionRequest) > 0
DO
  PActionRecvd^ : FROM PActionRequest GET PAction{ALL};
END

#-------------------------------------------------------------------------
#
# Ereignis 2 : Erzeugen von messages aufgrund von action requests
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(PActionRecvd) > 0
DECLARE receiver (INTEGER)
DO PROCEDURE
  receiver := PActionRecvd:PAction[1].Receiver;
END

DO TRANSITIONS
  IF PActionRecvd:PAction[1].Type = 'PAOffer'
  DO
    MSend^ : ADD 1 NEW Message
             CHANGING
              Agent^    := 'Prod';
              Sender^   := Id;
              Receiver^ := receiver;
              Type^     := 'POffer';
              TStamp^   := T;
```

```
              Price^    := PActionRecvd:PAction[1].Price;
             END
  IF Protocol DO
  DISPLAY("T=%.2f : Prod %d (PActor) -> 'POffer' to Consumer %d ",
           T, Id, receiver);
  DISPLAY("(Price offered=%.2f)\n",PActionRecvd:PAction[1].Price);
  END
  END

  ELSIF PActionRecvd:PAction[1].Type = 'PAAck'
  DO
    MSend^ : ADD 1 NEW Message
             CHANGING
              Agent^    := 'Prod';
              Sender^   := Id;
              Receiver^ := receiver;
              Type^     := 'PAck';
              TStamp^   := T;
              Price^    := PActionRecvd:PAction[1].Price;
              Quantity^ := PActionRecvd:PAction[1].Quantity;
             END

    IF Protocol DO
    DISPLAY("T=%.2f : Prod %d (PActor) -> 'PAck' to consumer %d ",
             T, Id, receiver);
DISPLAY("(Price acknowledged=%.2f Qty=%.2f)\n",PActionRecvd:PAction[1].Price,
           PActionRecvd:PAction[1].Quantity);
    END
  END

  PActionRecvd^ : REMOVE PAction[1];
END

#-------------------------------------------------------------------------------
#
# Ereignis 3 : Nachrichten werden an den Connector uebergeben.
#
#-------------------------------------------------------------------------------

WHENEVER NUMBER(MSend) > 0
DO
  MSend^ : TO InP SEND Message[1];
END

END # IF Active

END OF PActor
```

B.3 Die *Consumer*-Agenten

B.3.1 Die Strukturkomponente *Consumer*

```
#-------------------------------------------------------------------------------
#
# Name:  "Consumer"
#
# Art:   High-Level Komponente
#
# Beschreibung:
#
#   - Diese Komponente fasst alle angegebenen Subkomponenten zu einer
#     Komponente zusammen. Signale wie Id und Active werden
```

```
#     dabei an die Subkomponenten durchgereicht.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - Connector, Statistic
#
#-------------------------------------------------------------------------------

HIGH LEVEL COMPONENT Consumer

SUBCOMPONENTS
  CSensor,
  CPerception,
  CCognition,
  CBehaviour,
  CActor,
  CStatus

INPUT CONNECTIONS

 Id     --> (
              CSensor.Id,
              CPerception.Id,
              CCognition.Id,
              CBehaviour.Id,
              CActor.Id,
              CStatus.Id
             );

 Active --> (
              CSensor.Active,
              CPerception.Active,
              CCognition.Active,
              CBehaviour.Active,
              CActor.Active,
              CStatus.Active
             );

 OutC   --> CSensor.OutC;
 InC    --> CActor.InC;
 MState --> CSensor.MState;


OUTPUT EQUIVALENCES

  Prot_S    := CSensor.Protocol;
  Prot_P    := CPerception.Protocol;
  Prot_C    := CCognition.Protocol;
  Prot_B    := CBehaviour.Protocol;
  Prot_A    := CActor.Protocol;
  Prot_St   := CStatus.Protocol;
  KnownProd := CCognition.KnownProd;
  Pref      := CCognition.Pref;

COMPONENT CONNECTIONS

  CSensor.CTSensd             --> CPerception.CTSensd;
  CPerception.CTPercvd        --> CCognition.CTPercvd;
  CCognition.MState           --> CBehaviour.MState;
  CCognition.KnownProd{ALL i} --> CBehaviour.KnownProd[i];
  CCognition.InfCompl         --> CBehaviour.InfCompl;
  CCognition.Pref{ALL i}      --> CBehaviour.Pref[i];
  CCognition.LastPrice{ALL i} --> CBehaviour.LastPrice[i];
  CCognition.QuerySent{ALL i} --> CBehaviour.QuerySent[i];
  CCognition.Qty              --> CBehaviour.Qty;
  CCognition.QtyReady         --> CBehaviour.QtyReady;
  CCognition.AckHistory       --> CStatus.AckHistory;
```

```
  CCognition.NPeriod          --> CStatus.NPeriod;
  CBehaviour.CAReq            --> CActor.CAReq;
  CBehaviour.CTGen            --> CCognition.CTGen;

END OF Consumer
```

B.3.2 Die Komponente *CSensor*

```
#-------------------------------------------------------------------------------
#
# Name:  "CSensor" (Consumer-Sensor)
#
# Art:   Basis-Komponente
#
# Beschreibung:
#
#   - Diese Komponente holt bereitliegende Nachrichten im Connector ab
#     und wandelt sie in entsprechende 'CThoughts' (Consumer thoughts) um.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - Connect, CPercept
#
#-------------------------------------------------------------------------------

BASIC COMPONENT CSensor

MOBILE SUBCOMPONENTS OF CLASS Message, CThought

DECLARATION OF ELEMENTS

  STATE VARIABLES
   DISCRETE
    #----------------------------------------
    # DISPLAY-Anweisungen ein- und ausschalten
    #----------------------------------------
    Protocol(LOGICAL) := FALSE

  SENSOR VARIABLES
   DISCRETE
    #-------------------------------------------------------------------
    # Active entscheidet, ob ein Consumer am Marktgeschehen teilnimmt
    # oder nicht.
    #-------------------------------------------------------------------
    Active (LOGICAL) := FALSE,

    #----------------------------
    # Identifikator des Consumers
    #----------------------------
    Id (INTEGER),

    #----------------------------------------
    # Marktzustand:
    #   TRUE entspricht Transaktionsphase
    #   FALSE entspricht Preisbestimmungsphase
    #----------------------------------------
    MState (LOGICAL)

  LOCATIONS
    #----------------------------------------
    # Bereich fuer empfangene messages
    #----------------------------------------
    MRecvd (Message) := 0 Message,
```

```
    #-------------------------------------------------------
    # Bereich fuer in consumer thoughts umgewandelte messages
    #-------------------------------------------------------
    CTSensd (CThought) := 0 CThought

  SENSOR LOCATIONS
    #-----------------------------------------------
    # Ausgangsbereich dieses Consumers im Connector.
    #-----------------------------------------------
    OutC (Message)


DYNAMIC BEHAVIOUR

IF Active DO

#-------------------------------------------------------------------------
#
# Ereignis 1: Sobald im Ausgangsbereich Nachrichten fuer diesen Consumer
#             vorliegen, werden diese nach MRecvd transferiert.
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(OutC) > 0
DO
  MRecvd^ : FROM OutC GET Message{ALL};

  IF Protocol
  DO
   DISPLAY("T=%.2f : Cons %d (CSensor) -> ", T, Id);
   DISPLAY("Retrieving %d message(s) from Connector\n", NUMBER(OutC));
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 2: Empfangene Nachrichten werden nun in CThoughts umgewandelt.
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(MRecvd) > 0
DO
  #-----------------------------------------------------------------------
  # Im Falle eines Preisangebotes wird ein CThought vom Typ CTOffer erzeugt
  #-----------------------------------------------------------------------
  IF MRecvd:Message[1].Type = 'POffer'
  DO
    CTSensd^: ADD 1 NEW CThought
              CHANGING
                Sender^  := MRecvd:Message[1].Sender;
                Type^    := 'CTOffer';
                TStamp^  := MRecvd:Message[1].TStamp;
                Deadline^:= MRecvd:Message[1].Deadline;
                Price^   := MRecvd:Message[1].Price;
              END
  END
  #-----------------------------------------------------------------------
  # Im Falle einer Bestaetigung wird ein CThought vom Typ CTAck erzeugt
  #-----------------------------------------------------------------------
  ELSIF MRecvd:Message[1].Type = 'PAck'
  DO
    CTSensd^: ADD 1 NEW CThought
              CHANGING
                Sender^  := MRecvd:Message[1].Sender;
                Type^    := 'CTAck';
                TStamp^  := MRecvd:Message[1].TStamp;
                Deadline^:= MRecvd:Message[1].Deadline;
```

```
                Price^   := MRecvd:Message[1].Price;
                Quantity^:= MRecvd:Message[1].Quantity;
              END
  END

  ELSE
    DO
      DISPLAY("Error Cons %d (CSensor) -> Sensed unknown message type\n", Id);
    END

  MRecvd^ : REMOVE Message[1];
END

#-------------------------------------------------------------------------------
#
# Ereignis 3 : Markt ist geoeffnet worden
#
#-------------------------------------------------------------------------------

ON ^MState = TRUE^
DO
  CTSensd^ : ADD 1 NEW CThought
             CHANGING
               Type^   := 'CTMOpen';
               TStamp^ := T;
             END

  IF Protocol
  DO
    DISPLAY("T=%.2f : Cons %d (CSensor) -> Sensed market begin\n",
             T, Id);
  END
END

#-------------------------------------------------------------------------------
#
# Ereignis 4 : Markt ist geschlossen worden
#
#-------------------------------------------------------------------------------

ON ^MState = FALSE^
DO
  CTSensd^ : ADD 1 NEW CThought
             CHANGING
               Type^   := 'CTMClose';
               TStamp^ := T;
             END
  IF Protocol
  DO
    DISPLAY("T=%.2f : Cons %d (CSensor) -> Sensed market shutdown\n",
             T, Id);
  END
END

END # IF Active

END OF CSensor
```

B.3.3 Die Komponente *CPerception*

```
#-------------------------------------------------------------------------------
#
# Name:  "CPerception" (Consumer-Perception)
#
```

```
# Art:   Basis-Komponente
#
# Beschreibung:
#
#   - Diese Komponente leitet 'CThoughts' aus "CSensor" - im Moment -
#     unverzerrt an "CCognit" weiter.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - CSensor, CCognit
#
#-------------------------------------------------------------------------

BASIC COMPONENT CPerception

MOBILE SUBCOMPONENTS OF CLASS CThought

DECLARATION OF ELEMENTS

  STATE VARIABLES
    DISCRETE
    Protocol(LOGICAL) := FALSE

  SENSOR VARIABLES
    DISCRETE
    Active  (LOGICAL) := FALSE,
    Id (INTEGER)

  LOCATIONS
    #-----------------------------------------------------------------------
    # Bereich fuer ausgehende Perzepte. Da die thoughts bereits die
    # kodierten Informationen enthalten, werden sie unverzerrt Cognition
    # bereitgestellt.
    #-----------------------------------------------------------------------
    CTPercvd (CThought) := 0 CThought

  SENSOR LOCATIONS
    #--------------------------------------------
    # Bereich fuer in CSensor erzeugte CThoughts
    #--------------------------------------------
    CTSensd (CThought)


DYNAMIC BEHAVIOUR

IF Active DO

#-------------------------------------------------------------------------
#
# Ereignis 1 : In Sensor generierte CThoughts werden nach CTPercvd geholt.
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(CTSensd) > 0
DO
  CTPercvd^: FROM CTSensd GET CThought{ALL};

  IF Protocol
  DO
    DISPLAY("T=%.2f : Cons %d (CPerception) -> Perceiving %d thought(s)\n",
             T, Id, NUMBER(CTSensd));
  END
END

END # IF Active

END OF CPerception
```

B.3.4 Die Komponente *CCognition*

```
#-------------------------------------------------------------------------------
#
# Name: "CCognition" (Consumer-Cognition)
#
# Art:   Basis-Komponente
#
# Beschreibung:
#
#   - Diese Komponente enthaelt das elementare Wissen des Consumers,
#     das durch 'CThoughts' aus "CPercept" und "CBehav" immer wieder
#     aktualisiert wird.
#
#   - Dem Consumer ist bekannt, wann der Markt geoeffnet oder
#     geschlossen ist. In den ersten beiden Ereignissen findet die
#     Steuerung des Marktzustandes 'MState' statt.
#
#   - 'CThoughts' aus "CPercept" werden gespeichert und die empfangenen
#     Informationen werden je nach Typ interpretiert. Beispielsweise
#     wird bei jedem eingehenden Preisangebot ueberprueft, ob es das
#     letzte ausstehende Angebot war, um das Signal 'InfCompl' fuer
#     "CBehav" setzen zu koennen.
#
#   - In "CBehav" erzeugte 'CThoughts' werden analog interpretiert und
#     die lokale Datenbasis wird immer auf den letzten Stand gebracht.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - CPercept, CBehav, CStatus
#
#-------------------------------------------------------------------------------

BASIC COMPONENT CCognition

MOBILE SUBCOMPONENTS OF CLASS CThought

LOCAL DEFINITIONS
  DIMENSIONS
    #-----------------------------
    # Obergrenze: Anzahl Producer
    #-----------------------------
    NMaxProd := 50

    #-------------------------------------------------------------------------
    # Funktion zum Bestimmen der einzukaufenden Menge. Dazu wird ein Array
    # Utility von 1 bis n solange durchlaufen bis Price groesser als
    # der Eintrag an dieser Stelle wird. Die Menge ergibt sich dann
    # aus der letzten Index-position.
    # (Dies ist die gewaehlte Repraesentation einer Nachfragekurve nach
    # Caldas/Coelho).
    #-------------------------------------------------------------------------
    FUNCTION FPriceUtilRel (ARRAY [n] REAL : Utility, REAL : Price --> REAL)
    DECLARE j (REAL) := 0.0
    BEGIN
      FOR i FROM 1 TO n
      REPEAT
        IF Price <= Utility[i]
        DO
          j := j + 1.0;
        END
        ELSE
        DO
          RETURN (j);
        END
      END_LOOP
```

```
      RETURN (j);
    END_FUNC


DECLARATION OF ELEMENTS

  CONSTANTS
    #---------------------------------------------------------------------
    # Maximale Anzahl an Preisangeboten, die gespeichert werden koennen.
    #---------------------------------------------------------------------
    kMaxOff (INTEGER) := 20,

    #---------------------------------------------------------------------
    # Maximale Anzahl an Bestaetigungen, die gespeichert werden koennen.
    #---------------------------------------------------------------------
    kMaxAck (INTEGER) := 20

  STATE VARIABLES
    DISCRETE
    #-----------------------------------------------------------
    # Marktzustand (Transaktions- oder Preisbestimmungsphase)
    #-----------------------------------------------------------
    MState   (LOGICAL) := FALSE,

    #--------------------------
    # Nummer der Marktperiode
    #--------------------------
    NPeriod  (INTEGER) := 0,

    #-------------------------------------------------------------------
    # Anz. d. Prod., an die eine Preisanfrage (CAsk) geschickt wurde.
    #-------------------------------------------------------------------
    NQuerySent   (INTEGER) := 0,

    #---------------------
    # Einzukaufende Menge
    #---------------------
    Qty (REAL) := 0,

    #-----------------------------------------------------------------
    # Bekannte Producer
    # Ist der Producer i bekannt, so ist sein Eintrag im Array wahr.
    #-----------------------------------------------------------------
    ARRAY [NMaxProd] KnownProd (LOGICAL) := FALSE,

    #-----------------------------------------------------------------
    # Praeferenz gegenueber den einzelnen Producern:
    # Dieser Wert soll zwischen 0.0 und 1.0 liegen. 1.0 gibt dabei die
    # hoechstmoegliche Praeferenz an.
    #-----------------------------------------------------------------
    ARRAY [NMaxProd] Pref (REAL) := 1.0,

    #-----------------------------------------------------------------
    # Producer i wurde eine Preisanfrage gesandt (QuerySent[i] = TRUE)
    #-----------------------------------------------------------------
    ARRAY [NMaxProd] QuerySent (LOGICAL) := FALSE,

    #-------------------------------------------------------------
    # Producer i wurde eine Kauforder gesandt (OrderSent[i] = TRUE)
    #-------------------------------------------------------------
    ARRAY [NMaxProd] OrderSent (LOGICAL) := FALSE,

    #---------------------------------------------------------------------------
    # LastPrice (Last Price) enthaelt den Preis der letzten Anfrage des
    # jeweiligen Producers.
    #---------------------------------------------------------------------------
    ARRAY [NMaxProd] LastPrice (REAL) := -1,
```

```
  #-------------------------------------------------------------------------
  # Nachfragetabelle des Consumers. Werte sind in absteigender Reihen-
  # folge geordnet. (siehe Funktion FPriceUtilRel)
  #-------------------------------------------------------------------------
  ARRAY [6] Utility (REAL) := 0.0,

  Protocol (LOGICAL) := FALSE

SENSOR VARIABLES
  DISCRETE
  Active (LOGICAL) := FALSE,
  Id (INTEGER)

TRANSITION INDICATORS
  #-----------------------------------
  # Ein Preisangebot ist eingetroffen.
  #-----------------------------------
  NewOff,

  #-----------------------------------
  # Eine Bestaetigung ist eingetroffen.
  #-----------------------------------
  NewAck,

  #--------------------------------------
  # Alle Preisanfragen wurden beantwortet.
  #--------------------------------------
  InfCompl,

  #-------------------------------------------------------------------------
  # Menge wurde bestimmt (Consumer ist bereit, eine Bestellung aufzugeben.)
  #-------------------------------------------------------------------------
  QtyReady

LOCATIONS

  #----------------------------------------
  # Location fuer wahrgenommene Nachrichten
  #----------------------------------------
  CTKnown (CThought) := 0 CThought,

  #-------------------------------------------------
  # Location mit den letzten kMaxOff Preisangeboten
  #-------------------------------------------------
  OffRecvd (CThought) := 0 CThought,

  #-------------------------------------------------
  # Location mit den letzten kMaxAck Bestaetigungen
  #-------------------------------------------------
  AckRecvd (CThought) := 0 CThought,

  #----------------------------------------
  # Location fuer Informationen an CStatus
  #----------------------------------------
  AckHistory (CThought) := 0 CThought,

  #--------------------------------------
  # Location mit CThoughts aus CBehaviour
  #--------------------------------------
  CTUpd (CThought) := 0 CThought

SENSOR LOCATIONS
  #-------------------------------------------------------
  # Location mit wahrgenommenen CThoughts aus CPerception
  #-------------------------------------------------------
  CTPercvd (CThought),
```

```
    #-------------------------------------------------------
    # Location mit intern erzeugten CThoughts aus CBehaviour
    #-------------------------------------------------------
    CTGen (CThought)

DYNAMIC BEHAVIOUR

IF Active DO

#-------------------------------------------------------------------------
#
# Ereignis 1 : Zu Beginn der Transaktionsphase wird die Nummer der Markt-
#              periode hochgezaehlt und die Preise der letzten Periode
#              werden geloescht, um Platz fuer neue Werte zu schaffen.
#
#-------------------------------------------------------------------------

ON ^MState = TRUE^
DO
  NPeriod^ := NPeriod + 1;

  LastPrice{ALL}^ := -1;

  IF Protocol
  DO
    DISPLAY("T=%.2f : Cons %d (CCogn) -> I know the market is open now\n",
            T, Id);
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 2 : Am Ende der Transaktionsphase werden diverse Datenbereiche
#              zurueckgesetzt.
#
#-------------------------------------------------------------------------

ON ^MState = FALSE^
DO
  NQuerySent^ := 0;

  QuerySent{ALL}^ := FALSE;

  OrderSent{ALL}^ := FALSE;

  IF Protocol
  DO
    DISPLAY("T=%.2f : Cons %d (CCogn) -> I know the market is closed now\n",
            T, Id);
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 3 : In CPerception angekommene Thoughts werden registriert
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(CTPercvd) > 0
DO
  CTKnown^: FROM CTPercvd GET CThought{ALL};
END

#-------------------------------------------------------------------------
#
# Ereignis 4 : Aus CPercept angekommene Thoughts werden mit DISPLAY-Anwei
#              Anweisungen sichtbar gemacht und in die locations AckRecvd
```

```
#              und OffRecvd verteilt.
#
#-------------------------------------------------------------------------------

WHENEVER NUMBER(CTKnown) > 0
DO
  IF CTKnown:CThought[1].Type = 'CTOffer'
  DO
    #-----------------------------------------------------
    # Preisangebote werden nach OffRecvd weitergeleitet
    #-----------------------------------------------------
    CTKnown^: TO OffRecvd SEND CThought[1];

    SIGNAL NewOff;

    IF Protocol
    DO
      DISPLAY ("T=%.2f : Cons %d (CCogn) -> Received offer from",
                T, Id);
      DISPLAY (" producer %d ", CTKnown:CThought[1].Sender);
      DISPLAY ("(price offered=%.2f)\n", CTKnown:CThought[1].Price);
    END
  END

  ELSIF CTKnown:CThought[1].Type = 'CTAck'
  DO
    #-----------------------------------------------------------
    # Preisbestaetigungen werden nach AckRecvd weitergeleitet
    #-----------------------------------------------------------
    CTKnown^: TO AckRecvd SEND CThought[1];

    SIGNAL NewAck;

    IF Protocol
    DO
      DISPLAY ("T=%.2f : Cons %d (CCogn) -> Received acknowledge ",
                T, Id);
      DISPLAY ("from Producer %d\n", CTKnown:CThought[1].Sender);
    END
  END

  ELSIF CTKnown:CThought[1].Type = 'CTMOpen'
  DO
    #-------------------------------------------------
    # Consumer weiss nun, dass der Markt geoffnet ist
    #-------------------------------------------------
    MState^ := TRUE;
    CTKnown^ : REMOVE CThought[1];
  END

  ELSIF CTKnown:CThought[1].Type = 'CTMClose'
  DO
    #-------------------------------------------------
    # Consumer weiss nun, dass der Markt geschlossen ist
    #-------------------------------------------------
    MState^ := FALSE;
    CTKnown^ : REMOVE CThought[1];
  END

  ELSE
  DO
    DISPLAY("Error Cons %d (CCogn) -> Unknown Cthought type!\n",Id);
    CTKnown^ : REMOVE CThought[1];
  END

END
```

```
#-------------------------------------------------------------------------
#
# Ereignis 5 : Sobald sich in OffRecvd mehr als kMaxOff Angebote befinden,
#              wird der aelteste CThought geloescht.
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(OffRecvd) > kMaxOff
DO
  OffRecvd^ : REMOVE CThought[1];

  IF Protocol
  DO
   DISPLAY ("T=%.2f : Cons %d (CCogn) -> Cant remember more than ",T, Id);
   DISPLAY ("%d offers\n", kMaxOff);
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 6 : Sobald sich in AckRecvd mehr kMaxAck Bestaetigungen befinden,
#              wird der aelteste CThought in AckRecvd geloescht.
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(AckRecvd) > kMaxAck
DO
  AckRecvd^ : REMOVE CThought[1];

  IF Protocol
  DO
   DISPLAY ("T=%.2f : Cons %d (CCogn) -> Cant remember more than ",T, Id);
   DISPLAY ("%d acks\n", kMaxAck);
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 7 : Stehen in CBehaviour erzeugte CThoughts bereit, werden diese
#              in die Location CTUpd bewegt.
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(CTGen) > 0
DO
  CTUpd^ : FROM CTGen GET CThought{ALL};
END

#-------------------------------------------------------------------------
#
# Ereignis 8 : In CBehaviour erzeugte CThougths werden ihrem Typ nach
#              unterschieden, so dass die entsprechende Aktion ausgefuehrt
#              wird.
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(CTUpd) > 0
DO
  IF CTUpd:CThought[1].Type = 'CTAskUpd'
  DO
    QuerySent[CTUpd:CThought[1].Id]^ := TRUE;

    NQuerySent^ := NQuerySent + 1;

    IF Protocol
    DO
      DISPLAY("T=%.2f : Cons %d (CCogn) -> So I sent a price query to ",
```

```
              T,Id);
      DISPLAY("producer %d!?\n", CTUpd:CThought[1].Id);
    END
  END

  ELSIF CTUpd:CThought[1].Type = 'CTOrdUpd'
  DO
    OrderSent[CTUpd:CThought[1].Id]^ := TRUE;
  END

  ELSIF CTUpd:CThought[1].Type = 'CTSelPId'
  DO
    IF Protocol
    DO
      DISPLAY("T=%.2f : Cons %d (CCogn) -> ");
      DISPLAY("Price of selected producer is %.2f, ",
               T, Id, CTUpd:CThought[1].Price);
    END

    #-------------------------------------------------------------------------
    # Hat der Producer irrtuemlicherweise einen negativen Preis angeboten?
    # (sollte nicht vorkommen)
    #-------------------------------------------------------------------------
    IF LastPrice[CTUpd:CThought[1].Id] < 0
    DO
      DISPLAY("T=%.2f : Error Cons %d (CCogn) -> ", T, Id);
      DISPLAY("what shall I do with a negative price!?\n");
    END

    ELSE
    DO
      #-----------------------------------------
      # Menge wird gemaess utility-Skala gewaehlt
      #-----------------------------------------
      Qty^ := FPriceUtilRel (ARRAY Utility, LastPrice[CTUpd:CThought[1].Id]);

      IF Protocol
      DO
        DISPLAY("quantity is %.2f\n",
              FPriceUtilRel (ARRAY Utility, LastPrice[CTUpd:CThought[1].Id]));
      END

      SIGNAL QtyReady;
    END
  END

  ELSE
  DO
    DISPLAY("T=%.2f : Error Cons %d (CCogn) -> Unknown CThought-type!\n",
             T,Id);
  END

  CTUpd^ : REMOVE CThought[1];
END

#-------------------------------------------------------------------------------
#
# Ereignis 9 : Sobald das Signal NewOff gesetzt wird, wird ueberprueft,
#              ob diese Nachricht auch angefordert wurde. Ist dies die
#              letzte Preisnachricht, die noch erwartet wurde, wird ein
#              weiteres Signal InfCompl gesetzt.
#              Das letzte Angebot liegt am Ende der Location OffRecvd.
#
#-------------------------------------------------------------------------------

ON NewOff
DECLARE lastitem (INTEGER) := 0
```

```
DO PROCEDURE
  lastitem := NUMBER(OffRecvd);
END
DO TRANSITIONS
  #-------------------------------------------------------
  # Speichern des angebotenen Preises im Array LastPrice
  #-------------------------------------------------------
  LastPrice [OffRecvd:CThought[lastitem].Sender]^ :=
             OffRecvd:CThought[lastitem].Price;


  #-------------------------------------------------------------
  # Wurde diesem Producer ueberhaupt eine Preisanfrage gesandt?
  #-------------------------------------------------------------
  IF QuerySent[ OffRecvd:CThought[lastitem].Sender ] = TRUE
  DO
    #-----------------------------------------------------------
    # Dieses gerade eintreffende Preisangebot war das letzte, falls
    # NQuerySent gleich 1 ist.
    #-----------------------------------------------------------
    IF NQuerySent = 1
    DO
      #-----------------------------------------------
      # Signalisiere, dass die Preisinfo komplett ist.
      #-----------------------------------------------
      SIGNAL InfCompl;
    END

    NQuerySent^ := NQuerySent - 1;
  END
END

#-------------------------------------------------------------------------------
#
# Ereignis 10 : Wenn eine acknowledge message eingetroffen ist, wird
#               ueberprueft, ob an diesen Producer ueberhaupt eine Kauf-
#               order abgeschickt wurde.
#
#-------------------------------------------------------------------------------

ON NewAck
DECLARE lastitem (INTEGER) := 0
DO PROCEDURE
  lastitem := NUMBER(AckRecvd);
END

DO TRANSITIONS
  IF OrderSent[AckRecvd:CThought[lastitem].Sender] = TRUE
  DO
    IF Protocol DO
     DISPLAY("T=%.2f : Cons %d (CCogn) -> Producer %d ",T, Id,
              AckRecvd:CThought[lastitem].Sender);
     DISPLAY("acknowledged my order!\n");
    END

  #-----------------------------------
  # Information fuer CStatus kopieren
  #-----------------------------------
  AckHistory^ : ADD 1 NEW CThought
                CHANGING
                  Sender^    := AckRecvd:CThought[lastitem].Sender;
                  Type^      := AckRecvd:CThought[lastitem].Type;
                  TStamp^    := AckRecvd:CThought[lastitem].TStamp;
                  Deadline^  := AckRecvd:CThought[lastitem].Deadline;
                  Price^     := AckRecvd:CThought[lastitem].Price;
                  Quantity^  := AckRecvd:CThought[lastitem].Quantity;
                END
```

```
  END

  ELSE
  DO
    DISPLAY("T=%.2f : Error Cons %d (CCogn) -> Received ",T, Id);
    DISPLAY("an acknowledge from a producer I havent sent an order to!\n");
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 11 : InfCompl wurde in Ereignis 8 gesetzt -> DISPLAY
#
#-------------------------------------------------------------------------

ON InfCompl
DO
  IF Protocol
  DO
   DISPLAY("T=%.2f : Cons %d (CCogn) -> ", T, Id);
   DISPLAY("all price queries have been answered\n");
  END
END

END # IF Active

END OF CCognition
```

B.3.5 Die Komponente *CStatus*

```
#-------------------------------------------------------------------------
#
# Name:  "CStatus" (Consumer-Status)
#
# Art:   Basis-Komponente
#
# Beschreibung:
#
#   - Diese Komponente speichert von "CCognit" empfangene Transaktions-
#     informationen.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - CCognit
#
#-------------------------------------------------------------------------

BASIC COMPONENT CStatus

MOBILE SUBCOMPONENTS OF CLASS CTrans, CThought

DECLARATION OF ELEMENTS

CONSTANTS
  kMaxTran (INTEGER) := 10

STATE VARIABLES
  DISCRETE
  Protocol (LOGICAL) := FALSE

SENSOR VARIABLES
  DISCRETE
  Active  (LOGICAL) := FALSE,
  Id      (INTEGER),
```

```
  NPeriod (INTEGER)

LOCATIONS
  CTransQ (CTrans ORDERED BY INC Period) := 0 CTrans,
  CTTrash (CThought) := 0 CThought

SENSOR LOCATIONS
  AckHistory (CThought)

DYNAMIC BEHAVIOUR

IF Active DO

#-------------------------------------------------------------------------
#
# Ereignis 1 : Liegt in AckHistory min. 1 CThought (Ack) bereit, wird
#              dieser ausgewertet und die Daten der Transaktion werden in
#              einem CTrans gespeichert.
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(AckHistory) > 0
DO
  CTransQ^ : ADD 1 NEW CTrans
             CHANGING
               Period^   := NPeriod;
               Producer^ := AckHistory:CThought[1].Sender;
               Price^    := AckHistory:CThought[1].Price;
               Quantity^ := AckHistory:CThought[1].Quantity;
             END

  CTTrash^ : FROM AckHistory GET CThought[1];

  IF Protocol
  DO
    DISPLAY("T=%.2f : Cons %d (CStatus) -> Recording Transaction\n",T, Id);
    DISPLAY("\tPeriod %d : Prod = %d : Quantity = %.2f, Price = %.2f\n",
             NPeriod,
             AckHistory:CThought[1].Sender,
             AckHistory:CThought[1].Quantity,
             AckHistory:CThought[1].Price);
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 2 : Ueberschreitet Transact kMaxTran Eintraege, wird der erste
#              Eintrag geloescht, um Platz frei zu machen.
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(CTransQ) > kMaxTran
DO
  CTransQ^ : REMOVE CTrans[1];
END

#-------------------------------------------------------------------------
#
# Ereignis 3 : CThoughts in CTTrash werden geloescht
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(CTTrash) > 0
DO
  CTTrash^ : REMOVE CThought{ALL};
END
```

```
END # IF Active

END OF CStatus
```

B.3.6 Die Komponente *CBehaviour*

```
#-------------------------------------------------------------------------------
#
# Name: "CBehaviour" (Consumer-Behaviour)
#
# Art:   Basis-Komponente
#
# Beschreibung:
#
#   - Diese Komponente bestimmt das Verhalten eines Consumers.
#
#   - Zum Marktbeginn werden Preisanfragen an alle bekannten Producer
#     ausgeloest.
#
#   - Von "CCognition" erfaehrt die Komponente durch das Signal 'InfCompl',
#     dass die Preisanfragen beantwortet sind. Daraufhin wird der be-
#     vorzugte Producer ausgewaehlt.
#
#   - Wieviel nun bestellt werden soll, wird in "CCognition" abgefragt
#     und in Ereignis 5 wird schliesslich eine Bestellung an den ausge-
#     waehlten Producer ausgeloest.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - CCognition, CActor
#
#-------------------------------------------------------------------------------

BASIC COMPONENT CBehaviour

MOBILE SUBCOMPONENTS OF CLASS CAction, CThought

LOCAL DEFINITIONS
  DIMENSIONS
    #-----------------------------
    # Obergrenze: Anzahl Producer
    #-----------------------------
    NMaxProd := 50

DECLARATION OF ELEMENTS

  STATE VARIABLES
    DISCRETE
    #-------------------------------------------------------------------
    # Aktueller Producer, dem moeglicherweise eine message gesendet wird.
    # (siehe Ereignis 3)
    #-------------------------------------------------------------------
    ActProd (INTEGER) := 1,

    #-----------------------------------------------------
    # Id des ausgewaehlten Producers (Selected Producer Id)
    # (siehe Ereignisse 4 und 2)
    #-----------------------------------------------------
    SelPId (INTEGER) := -1,

    #-------------------------------------------------
    # Preis des ausgewaehlten Producers (Selected Price)
    # (siehe Ereignisse 4 und 2)
    #-------------------------------------------------
```

```
  SelPrice (REAL)  := -1.0,

  Protocol(LOGICAL) := FALSE

SENSOR VARIABLES
  DISCRETE
  Active (LOGICAL) := FALSE,
  Id     (INTEGER),
  MState (LOGICAL),   # Marktzustand
  Qty    (REAL),      # einzukaufende Menge

  # siehe CCognition fuer eine Beschreibung der folgenden Arrays
  ARRAY [NMaxProd] KnownProd (LOGICAL),
  ARRAY [NMaxProd] LastPrice (REAL),
  ARRAY [NMaxProd] Pref (REAL),
  ARRAY [NMaxProd] QuerySent (LOGICAL)

RANDOM VARIABLES
  #-------------------------------------------------------------------------
  # Zufallsvariable (in Ereignis 4) zum Auswaehlen eines Producers, falls
  # mehrere Producer die gleiche Praeferenz geniessen.
  #-------------------------------------------------------------------------

  # UpLimit = NMaxProd
  RanProd (INTEGER) : IUNIFORM (LowLimit := 1, UpLimit := 50)

TRANSITION INDICATORS
  #-------------------------------------------------------------------------
  # Ein Indikator, der in Ereignis 1 gesetzt wird sobald der Markt
  # oeffnet. Daraufhin wird in Ereignis 3 an alle bekannten Producer eine
  # Preisanfrage losgeschickt.
  #-------------------------------------------------------------------------
  SendCAsk,

  #-------------------------------------------------------------------------
  # Ein Indikator, der CCognition signalisiert, dass ein Producer aus-
  # gewaehlt wurde.
  #-------------------------------------------------------------------------
  PIdRdy

SENSOR INDICATORS
  #-------------------------------------------------------------------------
  # Wird in CCognition gesetzt, sobald alle Preisanfragen beantwortet wurden.
  #-------------------------------------------------------------------------
  InfCompl,

  #-------------------------------------------------------------------------
  # Wird in CCognition gesetzt, sobald die Menge bestimmt ist, die es ein-
  # zukaufen gilt.
  #-------------------------------------------------------------------------
  QtyReady

LOCATIONS
  #---------------------------------------
  # Location mit erzeugten Action requests
  #---------------------------------------
  CAReq (CAction) := 0 CAction,

  #-----------------------------------
  # Location mit erzeugten CThoughts
  #-----------------------------------
  CTGen (CThought) := 0 CThought
```

```
DYNAMIC BEHAVIOUR

IF Active DO

#-------------------------------------------------------------------------
#
# Ereignis 1 : Sobald der Markt beginnt -> Preisanfragen losschicken an
#              alle bekannten Produzenten.
#
#-------------------------------------------------------------------------

ON ^MState = TRUE^
DO
  SIGNAL SendCAsk;
END

#-------------------------------------------------------------------------
#
# Ereignis 2 : Sobald der Markt schliesst -> SelPId und SelPrice fuer
#              die naechste Marktperiode zuruecksetzen.
#
#-------------------------------------------------------------------------

ON ^MState = FALSE^
DO
  SelPId^ := -1;

  SelPrice^ := -1;
END

#-------------------------------------------------------------------------
#
# Ereignis 3 : Absenden der Preisanfragen an bekannte Producer.
#              Dabei haelt die ZV ActProd den aktuell betrachteten
#              Producer fest.
#
#-------------------------------------------------------------------------

ON SendCAsk
DO
  IF ActProd <= NMaxProd
  #-----------------------------------------------------------------------
  # Id des momentan betrachteteten Producers uebersteigt nicht die Grenze
  # von NMaxProd
  #-----------------------------------------------------------------------
  DO
    IF KnownProd[ActProd] = TRUE
    #---------------------------------------
    # Dieser Producer ist dem Consumer bekannt
    #---------------------------------------
    DO
      IF Protocol
      DO
        DISPLAY("T=%.2f : Cons %d (CBehaviour) -> Creating 'CAAsk' request for",
                T, Id);
        DISPLAY(" producer %d\n",ActProd);
      END

      #-------------------------------------------------------------------
      # Generiere einen Action-request, damit CActor was zu tun bekommt
      #-------------------------------------------------------------------
      CAReq^ : ADD 1 NEW CAction
               CHANGING
                 Receiver^ := ActProd;
                 Type^ := 'CAAsk';
                 TStamp^ := T;
               END
```

```
      #-------------------------------------------------------------------
      # Eine Update des Wissens geschieht, indem ein CThought vom Typ
      # 'CTAskUpd' erzeugt wird, der dann von CCognition verwertet wird.
      #-------------------------------------------------------------------
      CTGen^ : ADD 1 NEW CThought
               CHANGING
                 Type^ := 'CTAskUpd';
                 Id^   := ActProd;
               END

    END
    #-----------------------------------------------
    # Hochsetzen des momentan betrachteten Producers
    #-----------------------------------------------
    ActProd^ := ActProd + 1;

    #-------------------------------------------------
    # Loese ein erneutes Absenden einer Preisanfrage aus
    #-------------------------------------------------
    SIGNAL SendCAsk;
  END

  #---------------------------------------------------------------------
  # Ansonsten ist die Grenze ueberschritten, d.h. das komplette ARRAY
  # KnownProd wurde durchlaufen -> Ruecksetzen von ActProd fuer die
  # naechste Periode.
  #---------------------------------------------------------------------
  ELSE DO ActProd^ := 1; END
END

#-------------------------------------------------------------------------------
#
# Ereignis 4.1 : Sind alle Preisanfragen beantwortet worden, wird das Signal
#                InfCompl von CCognition gesetzt. Daraufhin wird der
#                bevorzugte Producer ermittelt.
#
#-------------------------------------------------------------------------------

ON InfCompl

DECLARE
  p (REAL) := 0.0,
  x (REAL) := 0.0,
  w (REAL) := 0.0,
  min_w (REAL)      := -1.0,
  min_PID (INTEGER) := 0,
  min_pric (REAL)   := 0.0,
  howmany (INTEGER) := 0,
  ARRAY [NMaxProd] eqminprc (INTEGER) := -1

DO PROCEDURE
  FOR i FROM 1 TO NMaxProd
  REPEAT
    #-------------------------------------------------------------
    # Nur solche Producer werden beruecksichtigt, an die auch eine
    # Preisanfrage geschickt wurde
    #-------------------------------------------------------------
    IF QuerySent[i] = TRUE
    DO
      #-------------------------------------------------------------
      # wie hoch war der letzte (aktuelle) Preis dieses Producers?
      #-------------------------------------------------------------
      p := LastPrice[i];

      #-----------------------------------------------
      # wie hoch ist meine Praeferenz ihm gegenueber?
      #-----------------------------------------------
```

```
      x := Pref[i];

      #-------------------------------------------------------------
      # anschliessend Berechnung von omega (w) fuer diesen Producer
      #-------------------------------------------------------------
      w := p / x;

      #-------------------------------------------------------------------------
      # Ist w kleiner als bisher, wird es gespeichert. Passiert auch, wenn
      # noch kein Wert gespeichert wurde (min_w hat noch seinen Initial-
      # wert).
      #-------------------------------------------------------------------------
      IF w < min_w OR min_w = -1
      DO
        #---------------------------------------
        # Omega, Id und Preis werden gespeichert
        #---------------------------------------
        min_w    := w;
        min_PID  := i;
        min_pric := p;
        howmany  := 1;
        eqminprc[howmany] := i;
      END

      ELSIF w = min_w
      DO
        howmany := howmany + 1;

        eqminprc[howmany] := i;
      END
    END
  END_LOOP

END # PROCEDURE

DO TRANSITIONS
  IF howmany = 1
  DO
    #-------------------------------------------------------------------
    # Die Id des Producers mit minimalen w wird in SelPId gespeichert
    #-------------------------------------------------------------------
    SelPId^   := min_PID;
  END

  ELSE # howmany >= 2
  DO
    #-------------------------------------------------------------------
    # Suche zufaellig einen Producer mit minimalen w aus
    # Die Ids der Producer liegen von 1..howmany im Array "eqminprc"
    #-------------------------------------------------------------------
    SelPId^ := eqminprc [IMOD(RanProd,howmany) + 1 ];

    IF Protocol
    DO
      DISPLAY("T=%.2f : Cons %d (CBehaviour) -> ", T, Id);
      DISPLAY("Choosing between %d producers.", howmany);
      DISPLAY("\n\tMy choice is producer %d\n", IMOD(RanProd, howmany)+1);
    END
  END

  SIGNAL PIdRdy;
END
```

```
#-------------------------------------------------------------------------
#
# Ereignis 4.2 : Die ID des Producers wurde ausgewaehlt.
#
#-------------------------------------------------------------------------

ON PIdRdy
DO
  IF Protocol
  DO
    DISPLAY("T=%.2f : Cons %d (CBehaviour) -> Producer %d is my choice!\n",
             T, Id, SelPId);
  END

  CTGen^ : ADD 1 NEW CThought
           CHANGING
             Type^ := 'CTSelPId';
             Id^   := SelPId;
           END
END

#-------------------------------------------------------------------------
#
# Ereignis 5 : Die zu bestellende Menge liegt nun in Qty bereit. Bleibt nur
#              noch uebrig, dem Producer mit der Id SelPId eine Kauforder zu
#              schicken und dies anschliessend CCognition mitzuteilen.
#
#-------------------------------------------------------------------------

ON QtyReady
DO
  #---------------------------------------
  # Kaufanweisung versenden, falls Qty > 0
  #---------------------------------------
  IF Qty > 0
  DO
    IF Protocol
    DO
      DISPLAY("T=%.2f : Cons %d (CBehaviour) -> 'CAOrder' request for",
               T, Id);
      DISPLAY(" producer %d (Qty = %.2f)\n", SelPId, Qty);
    END

    #-----------------------------------------------------------------
    # Generiere einen Action-request, damit CActor eine Bestellung
    # losschickt.
    #-----------------------------------------------------------------
    CAReq^ : ADD 1 NEW CAction
             CHANGING
               Receiver^ := SelPId;
               Type^     := 'CAOrder';
               TStamp^   := T;
               Price^    := LastPrice[SelPId];
               Quantity^ := Qty;
             END

    #---------------------------------
    # CCognition darueber informieren
    #---------------------------------
    CTGen^ : ADD 1 NEW CThought
             CHANGING
               Type^     := 'CTOrdUpd';
               Id^       := SelPId;
               Price^    := LastPrice[SelPId];
               Quantity^ := Qty;
             END
```

```
  END # IF Qty > 0
END

END # IF Active

END OF CBehaviour
```

B.3.7 Die Komponente *CActor*

```
#-------------------------------------------------------------------------------
#
# Name: "CActor" (Consumer-Actor)
#
# Art:   Basis-Komponente
#
# Beschreibung:
#
#   - Anstehende CAction-Anweisungen aus "CBehav" werden verarbeitet,
#     indem "Messages" vom passenden Typ erzeugt und anschliessend
#     an den Connector verschickt werden.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - CBehav, Connect
#
#-------------------------------------------------------------------------------

BASIC COMPONENT CActor

MOBILE SUBCOMPONENTS OF CLASS Message, CAction

DECLARATION OF ELEMENTS

  STATE VARIABLES
   DISCRETE
   Protocol(LOGICAL) := FALSE

  SENSOR VARIABLES
   DISCRETE
   Active  (LOGICAL) := FALSE,
   Id (INTEGER)

  LOCATIONS
  #-------------------------------------------------------
  # Bereich, in dem messages erzeugt und spaeter weiter-
  # geschickt werden
  #-------------------------------------------------------
  MSend (Message) := 0 Message,

  #-------------------------------
  # Empfangene Aktionsanweisungen
  #-------------------------------
  CARecvd (CAction) := 0 CAction

  SENSOR LOCATIONS
  #-----------------------------------------------
  # Eingangsbereich dieses Consumers im Connector
  #-----------------------------------------------
  InC (Message),

  #-----------------------------------------
  # Consumer Action requests von <CBehaviour>
  #-----------------------------------------
  CAReq (CAction)
```

```
DYNAMIC BEHAVIOUR

IF Active DO

#-------------------------------------------------------------------------------
#
# Ereignis 1 : Liegen in CBehaviour CAction-Anweisungen bereit, werden
#              diese nach CARecvd geholt.
#
#-------------------------------------------------------------------------------

WHENEVER NUMBER(CAReq) > 0
DO
  CARecvd^ : FROM CAReq GET CAction{ALL};

  IF Protocol
  DO
    DISPLAY("T=%.2f : Cons %d (CActor) -> ", T, Id);
    DISPLAY("copying action request\n");
  END
END

#-------------------------------------------------------------------------------
#
# Ereignis 2 : Erzeugen von messages aufgrund von action requests
#
#-------------------------------------------------------------------------------

WHENEVER NUMBER(CARecvd) > 0
DO
  IF CARecvd:CAction[1].Type = 'CAAsk'
  DO
    MSend^: ADD 1 NEW Message
            CHANGING
            # Header-Informationen ausfuellen
            Agent^    := 'Cons';
            Sender^   := Id;
            Receiver^ := CARecvd:CAction[1].Receiver;
            Type^     := 'CAsk';
            TStamp^   := T;
            END

    IF Protocol
    DO
      DISPLAY("T=%.2f : Cons %d (CActor) ", T, Id);
      DISPLAY("CAsk to Producer %d\n", CARecvd:CAction[1].Receiver);
    END

    CARecvd^ : REMOVE CAction[1];
  END

  ELSIF CARecvd:CAction[1].Type = 'CAOrder'
  DO
    MSend^: ADD 1 NEW Message
            CHANGING
            # Header-Informationen ausfuellen
            Agent^    := 'Cons';
            Sender^   := Id;
            Receiver^ := CARecvd:CAction[1].Receiver;
            Type^     := 'COrder';
            TStamp^   := T;

            Quantity^ := CARecvd:CAction[1].Quantity;
            Price^    := CARecvd:CAction[1].Price;
            END
    IF Protocol
    DO
```

```
      DISPLAY("T = %.2f : Cons %d (CActor) -> ", T, Id);
      DISPLAY("COrder to Producer %d\n",
              CARecvd:CAction[1].Receiver);
    END

    CARecvd^ : REMOVE CAction[1];
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 3 : Liegen messages in MSend bereit, werden diese an den
#              connector geschickt.
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(MSend) > 0
DO
  MSend^: TO InC SEND Message{ALL};
END

END # IF Active

END OF CActor
```

B.4 Die Komponente *Connector*

```
#-------------------------------------------------------------------------
#
# Name:  "Connector" (Connectoror)
#
# Art:   Basis-Komponente
#
# Beschreibung:
#
#   - Zentrale Komponente fuer den Nachrichtenaustausch zwischen
#     Producer und Consumer.
#     Die Consumer besitzen einen Identifikator von 1 bis Anzahl d.
#     Consumer. Analog verhaelt es sich bei den Producern.
#
#   - Nachrichten werden so erzeugt, dass in der betreffenden 'Receiver'-
#     variablen die Id des Empfaengers steckt.
#
#   - Kommunikation ist nur zwischen "ungleichen" Partnern moeglich,
#     d.h. ein Consumer <-> Consumer Nachrichtenaustausch ist nicht
#     vorgesehen.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - Consumer->CSensor, CActor
#   - Producer->PSensor, PActor
#
#-------------------------------------------------------------------------

BASIC COMPONENT Connector

MOBILE SUBCOMPONENTS OF CLASS Message

LOCAL DEFINITIONS
  DIMENSIONS
  #----------------------------
  # Obergrenze: Anzahl Consumer
  #----------------------------
```

```
  NMaxCons := 200,

  #-----------------------------
  # Obergrenze: Anzahl Producer
  #-----------------------------
  NMaxProd := 50

  #-------------------------------------------------------------------
  # Eine einfache Funktion, die ueberprueft, ob in einer der Locations
  # aus dem angegebenen Array eine mobile Komponente vorliegt.
  #-------------------------------------------------------------------
  FUNCTION MExists (ARRAY [n] LOCATION FOR Message: MQueue --> LOGICAL)
    DECLARE M (LOGICAL) := TRUE
  BEGIN
    FOR i FROM 1 TO n
    REPEAT
      IF NUMBER(MQueue[i])>0 DO RETURN (TRUE); END
    END_LOOP
    RETURN (FALSE);
  END_FUNC

DECLARATION OF ELEMENTS

  STATE VARIABLES
    DISCRETE
    Protocol (LOGICAL) := FALSE

  DEPENDENT VARIABLES
    DISCRETE
    NMaxTot (INTEGER)  # Anzahl Consumer + Producer

  LOCATIONS
    #---------------------------------------
    # Zentraler Bereich, in dem die messages
    # gesammelt und weitergeleitet werden.
    #---------------------------------------
    Gate (Message) := 0 Message,

    #-------------------------------
    # Ausgangsbereich fuer Producer
    #-------------------------------
    ARRAY [NMaxProd] OutP (Message) := 0 Message,

    #-------------------------------
    # Ausgangsbereich fuer Consumer
    #-------------------------------
    ARRAY [NMaxCons] OutC (Message) := 0 Message,

    #-------------------------------
    # Eingangsbereich fuer Producer
    #-------------------------------
    ARRAY [NMaxProd] InP (Message) := 0 Message,

    #-------------------------------
    # Eingangsbereich fuer Consumer
    #-------------------------------
    ARRAY [NMaxCons] InC (Message) := 0 Message

DYNAMIC BEHAVIOUR

#-------------------------
# Abhaengige Gleichungen
#-------------------------

NMaxTot := NMaxCons + NMaxProd;
```

```
#-------------------------------------------------------------------------
#
# Ereignis 1 : Messages, die im Eingangsbereich fuer Producer auftauchen
#              werden in Gate gesammelt.
#
#-------------------------------------------------------------------------

WHENEVER MExists (ARRAY LOCATION InP)
DO
 Gate^ : FROM InP{ALL} GET Message{ALL};

 IF Protocol
 DO
   DISPLAY("T=%.2f: Connectoror -> New Message from a Producer arrived\n", T);
 END
END

#-------------------------------------------------------------------------
#
# Ereignis 2 : Messages, die im Eingangsbereich fuer Consumer auftauchen
#              werden in Gate gesammelt.
#
#-------------------------------------------------------------------------

WHENEVER MExists (ARRAY LOCATION InC)
DO
 Gate^ : FROM InC{ALL} GET Message{ALL};

 IF Protocol
 DO
   DISPLAY("T=%.2f: Connectoror -> message(s) to Gate\n", T);
 END

END

#-------------------------------------------------------------------------
#
# Ereignis 3 : Sobald min. 1 message in Gate angekommen ist -> weiterschicken
#
#-------------------------------------------------------------------------

WHENEVER NUMBER(Gate) > 0
DO
  #-----------------------------------
  # Test, ob Empfaenger-ID gueltig ist
  #-----------------------------------
  IF Gate:Message[1].Receiver <> -1
  DO
    #-----------------------------------------
    # Fall 1: Message stammt von einem Producer
    #-----------------------------------------
    IF Gate:Message[1].Agent = 'Prod'
    DO
      #-----------------------------------------------------------
      # Sende die Nachricht an den Ausgangsbereich fuer Consumer
      #-----------------------------------------------------------
      Gate^ : TO OutC[Gate:Message[1].Receiver] SEND Message[1];
      IF Protocol
      DO
        DISPLAY ("T=%.2f: Connectoror -> ", T);
        DISPLAY ("Nachricht Producer %d --> Consumer %d\n",
                  Gate:Message[1].Sender, Gate:Message[1].Receiver);
      END
    END
```

```
    #-----------------------------------------
    # Fall 2: Message stammt von einem Consumer
    #-----------------------------------------
    ELSIF Gate:Message[1].Agent = 'Cons'
    DO
      #-------------------------------------------------------
      # Sende die Nachricht an den Ausgangsbereich fuer Producer
      #-------------------------------------------------------
      Gate^ : TO OutP[Gate:Message[1].Receiver] SEND Message[1];
      IF Protocol
      DO
        DISPLAY ("T=%.2f: Connectoror -> ", T);
        DISPLAY ("Nachricht Consumer %d --> Producer %d\n",
                  Gate:Message[1].Sender, Gate:Message[1].Receiver);
      END
    END

    #-----------------------------
    # Fall 3: Agent-Typ unbekannt
    #-----------------------------
    ELSE
    DO
      Gate^: REMOVE Message[1];
      DISPLAY("Error Connectoror : Agent type unknown! --> Message deleted\n");
    END
  END # Receiver <> -1
  ELSE
  DO
    Gate^: REMOVE Message[1];
    DISPLAY("Error Connectoror : Receiver-ID unknown! --> Message deleted\n");
  END
END

END OF Connector
```

B.5 Die Komponente *Control*

```
#-------------------------------------------------------------------------------
#
# Name:  "Control" (Markt-Kontroll)
#
# Art:   Basis-Komponente
#
# Beschreibung:
#
#   - Diese Komponente steuert die Marktperioden mit der ZV MState.
#
# Verbindung(en) mit anderen Komponenten:
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Control

LOCAL DEFINITIONS

DECLARATION OF ELEMENTS
  CONSTANTS
    kDistT (REAL) := 1.0

  STATE VARIABLES
    DISCRETE
    #-----------------------------------------
    # MState wird folgendermassen interpretiert:
```

```
    # MState = FALSE => Markt ist geschlossen
    # MState = TRUE  => Markt ist geoeffnet
    #-----------------------------------------
    MState  (LOGICAL) := FALSE,
    TNext   (REAL)    := 0.0,
    Protocol(LOGICAL) := FALSE

DYNAMIC BEHAVIOUR

#-------------------------------------------------------------------------
#
# Ereignis 1 : Marktzustand aendern, wenn T >= TNext
#
#-------------------------------------------------------------------------

WHENEVER T >= TNext
DO
  # Marktzustand auf den negierten Wert aendern
  MState^ := NOT(MState);

  TNext^ := T + kDistT;

  IF Protocol
  DO
    IF MState = FALSE
    DO
      DISPLAY("T=%.2f : Control -> Market is opening!\n",T);
    END
    ELSE
    DO
      DISPLAY("T=%.2f : Control -> Market is closing!\n",T);
    END
  END
END

END OF Control
```

B.6 Die Komponente *Statistic*

```
#-------------------------------------------------------------------------
#
# Name:  "Statistic" (Statistic)
#
# Art:   Basis-Komponente
#
# Beschreibung:
#
#   - Diese Komponente sammelt Informationen von allen Producern und
#     bildet daraus Groessen wie den Gesamtumsatz oder den jeweiligen
#     Marktanteil.
#
#   - Sie teilt den Producern durch das Signal 'MInfRdy' mit, wann
#     die Berechnung komplett ist.
#
# Verbindung(en) mit anderen Komponenten:
#
#   - Producer
#
#-------------------------------------------------------------------------

BASIC COMPONENT Statistic

LOCAL DEFINITIONS
```

```
  DIMENSIONS
    NMaxProd := 50

  #-------------------------------------------------------------------
  # Funktion zur Berechnung der Summe eines Arrays mit REAL-Werten.
  #-------------------------------------------------------------------
  FUNCTION Summe (ARRAY [m] REAL: x --> REAL)
  DECLARE y (REAL)
  BEGIN
    y := 0.0;
    FOR i FROM 1 TO m
    REPEAT
      y := y + x[i];
    END_LOOP
    RETURN (y);
  END_FUNC


DECLARATION OF ELEMENTS

  STATE VARIABLES
    DISCRETE
    Protocol (LOGICAL) := FALSE,

    #---------------------------------
    # Gesamtumsatz aller Produzenten
    #---------------------------------
    AllQty   (REAL)    := 0.0,


    #---------------------------------
    # Durchschnittlicher Verkaufspreis
    #---------------------------------
    AvgPrice (REAL)    := 0.0,


    #--------------------------------------
    # Marktanteile der einzelnen Producer
    #--------------------------------------
    ARRAY [NMaxProd] MShares (REAL) := 0.0,


    #-------------------------------------------------------------------
    # Preise der einzelnen Producer aus der vorangegangen Marktperiode
    #-------------------------------------------------------------------
    ARRAY [NMaxProd] Prices (REAL) := 0.0

  SENSOR VARIABLES
    DISCRETE
    #-----------------------------
    # Anzahl der aktiven Producer
    #-----------------------------
    NProd  (INTEGER),

    MState (LOGICAL),

    #-----------------------------
    # Einzelumsaetze der Producer
    #-----------------------------
    ARRAY [NMaxProd] Quantity (REAL),

    #------------------------
    # Preise der Produzenten
    #------------------------
    ARRAY [NMaxProd] Price (REAL)
```

```
  TRANSITION INDICATORS
    #--------------------------------------
    # Signal zum Berechnen der Marktanteile
    #--------------------------------------
    CalcMarketShare,

    #----------------------------------------------
    # Signal zum Berechnen des Durchschnittspreises
    #----------------------------------------------
    CalcAvgPriceP,

    #-----------------------------------------------------------------
    # Signal, das dem Producer mitteilt, dass Berechnung erfolgt ist
    #-----------------------------------------------------------------
    MInfRdy

DYNAMIC BEHAVIOUR

#-------------------------------------------------------------------------------
#
# Ereignis 1 : Nach Ende einer Marktperiode, d.h. <MState> wechselt von
#              'TRUE' nach 'FALSE', werden die Signale zur Berechnung
#              der Marktanteile und des Durchschnittspreises gesetzt.
#
#-------------------------------------------------------------------------------

ON ^MState = FALSE^
DO
  SIGNAL CalcMarketShare;

  SIGNAL CalcAvgPriceP;

  #-------------------------------------------------------------
  # Die Preise aller Produzenten werden in <Prices> gespeichert
  #-------------------------------------------------------------
  Prices{ALL i}^ := Price[i];
END

#-------------------------------------------------------------------------------
#
# Ereignis 2 : Die einzelnen Marktanteile werden bestimmt. Dazu werden
#              die Einzelumsaetze durch den Gesamtumsatz geteilt.
#              In dem Fall, dass ueberhaupt keine Umsaetze getaetigt wurden
#              (z.B. wenn alle Preise der Producer so hoch waren, dass kein
#              Consumer etwas bestellt hat), werden alle Marktanteile auf
#              0 gesetzt.
#
#-------------------------------------------------------------------------------

ON CalcMarketShare
DECLARE total (REAL)
DO PROCEDURE
  total := Summe(ARRAY Quantity);

END
DO TRANSITIONS
  AllQty^ := total;

  #-------------------------------------------------------------
  # Es kann vorkommen, dass keiner der Produzenten Ware ver-
  # kauft hat, weil jeder einzelne Verkaufspreis ueber dem
  # Limit der Konsumenten lag. => Marktanteil wird auf 0 gesetzt.
  #-------------------------------------------------------------
  IF total = 0
  DO
    MShares{ALL}^ := 0.0;
    IF Protocol DO
```

```
      DISPLAY("T=%.2f : Statistic -> Total sold is 0\n",T);
    END
  END
  ELSE
  DO
    MShares{ALL i}^ := Quantity[i] / total;
    IF Protocol DO
      DISPLAY("T=%.2f : Statistic -> Total sold is %.2f\n",
               T, total);
    END
  END
  SIGNAL MInfRdy;
END


#-------------------------------------------------------------------------
#
# Ereignis 3 : Der Durchschnittspreis wird berechnet. Gesamtpreis geteilt
#              durch Anzahl der Producer.
#
#-------------------------------------------------------------------------

ON CalcAvgPriceP
DECLARE avg_prc (REAL)
DO PROCEDURE
  avg_prc := Summe(ARRAY Price) / NProd;
END

DO TRANSITIONS
  AvgPrice^ := avg_prc;
  IF Protocol DO
    DISPLAY("T=%.2f : Statistic -> Average price is %.2f\n", T, avg_prc);
  END
END

END OF Statistic
```

B.7 Mobile Komponenten

B.7.1 Die mobile Komponente *CAction*

```
#-------------------------------------------------------------------------
#
# Name: "CAction" (Consumer-Action)
#
# Art:   Mobile Komponente
#
# Beschreibung:
#
#   - Mob. Komponente zum Uebertragen von Aktionsinformationen von
#     "CBehav" nach "CActor".
#
# Verbindung(en) mit anderen Komponenten: n/a
#
#-------------------------------------------------------------------------

MOBILE COMPONENT CAction

LOCAL DEFINITIONS

 #-------------------------------------------------------------
 # Aufzaehlungstyp fuer verschiedene Consumer-Aktionen
```

```
 #   'CAAsk'     : Consumer action Ask (Preisanfrage senden)
 #   'CAOrder'   : Consumer action Order (Bestellung absenden)
 #   'CAUnknwn'  : Consumer action unknown (unbekannte Aktion)
 #-------------------------------------------------------------

  VALUE SET CAType: ('CAAsk', 'CAOrder','CAUnknwn')

DECLARATION OF ELEMENTS

STATE VARIABLES
 DISCRETE
  #----------------------
  # Header-Informationen
  #----------------------
  Receiver (INTEGER) := -1,          # ID des Producers, den diese Aktion
                                     # betrifft
  Type     (CAType)  := 'CAUnknwn',  # Typ der Aktion
  TStamp   (REAL)    := -1.0,        # Zeitstempel
                                     # (derzeit nicht weiter verwendet)
  Deadline (REAL)    := -1.0,        # Einzuhaltende Zeitfrist
                                     # (derzeit nicht weiter verwendet)

  #---------------
  # Action-Inhalt
  #---------------
  Price    (REAL)    := -1.0,
  Quantity (REAL)    := -1.0

END OF CAction
```

B.7.2 Die mobile Komponente *CThought*

```
#-------------------------------------------------------------------------------
#
# Name:  "CThought" (Consumer-Thought)
#
# Art:   Mobile Komponente
#
# Beschreibung:
#
#   - Diese mobile Komponente transportiert alle moeglichen Daten
#     zwischen den Komponenten innerhalb eines Consumers.
#
#   - Gedanken entstehen zum einen in "CSensor", wo sie ueber "CPercept"
#     nach "CCognit" gelangen und zum anderen werden sie, durch eigene
#     Aktionen veranlasst, in "CBehav" erzeugt und wiederum von "CCognit"
#     ausgewertet.
#
# Verbindung(en) mit anderen Komponenten: n/a
#
#-------------------------------------------------------------------------------

MOBILE COMPONENT CThought

LOCAL DEFINITIONS

  #-----------------------------------------------------------------------
  # Moegliche 'CThought'-Arten:
  #
  # CTMOpen   : Markt ist geoeffnet
  # CTMClose  : Markt ist geschlossen
  # CTOffer   : Angebot ist eingetroffen
  # CTAck     : Bestaetigung der Bestellung ist angekommen
  # CTAskUpd  : Angebot wurde abgeschickt
```

```
  # CTOrdUpd  : Bestellung wurde abgeschickt
  # CTSelPId  : Producer Id wurde als Handelspartner ausgewaehlt
  # CTUnknwn  : unbekannter Typ
  #-------------------------------------------------------------------------
  VALUE SET CTType: ('CTMOpen', 'CTMClose', 'CTOffer', 'CTAck', 'CTAskUpd',
                     'CTOrdUpd', 'CTSelPId', 'CTUnknwn')

DECLARATION OF ELEMENTS

STATE VARIABLES
  DISCRETE
  #----------------------
  # Header-Informationen
  #----------------------
  Sender (INTEGER) := -1,
  Type (CTType)     := 'CTUnknwn',
  TStamp (REAL)    := -1.0,
  Deadline (REAL)  := -1.0,

  #----------------
  # Thought-Inhalt
  #----------------
  Price (REAL)     := -1.0,
  Quantity (REAL)  := -1.0,
  Id (INTEGER)     := -1

END OF CThought
```

B.7.3 Die mobile Komponente *CTrans*

```
#-------------------------------------------------------------------------------
#
# Name:  "CTrans" (Consumer-Transaction)
#
# Art:   Mobile Komponente
#
# Beschreibung:
#
#   - Mobile Komponente zum Datenaustausch zwischen "CCognit" und
#     "CStatus".
#
#
# Verbindung(en) mit anderen Komponenten: n/a
#
#-------------------------------------------------------------------------------

MOBILE COMPONENT CTrans

DECLARATION OF ELEMENTS

STATE VARIABLES
  Period   (INTEGER) := -1,
  Producer (INTEGER) := -1,
  Price    (REAL)    := -1.0,
  Quantity (REAL)    := -1.0

END OF CTrans
```

B.7.4 Die mobile Komponente *Message*

```
#-------------------------------------------------------------------------------
#
# Name:  "Message"
#
# Art:   Mobile Komponente
#
# Beschreibung:
#
#   - Eine mobile Komponente fuer den Nachrichtenaustausch zwischen
#     Consumer <-> Connector <-> Producer.
#
# Verbindung(en) mit anderen Komponenten: n/a
#
#-------------------------------------------------------------------------------

MOBILE COMPONENT Message

LOCAL DEFINITIONS
  #-----------------------------------------------------------------------------
  # Typen von Nachrichten:
  #   CAsk     : Preisanfrage (Consumer --> Producer)
  #   COrder   : Kaufauftrag (Consumer --> Producer)
  #   POffer   : Preisangebot (Producer --> Consumer)
  #   PAck     : Auftragsbestaetigung (Producer --> Consumer)
  #   MUnknown : Message-Typ unbekannt
  #-----------------------------------------------------------------------------
  VALUE SET MType  :  ('CAsk', 'COrder', 'POffer', 'PAck', 'MUnknown')

  # Typen von Absendern
  VALUE SET MFrom  :  ('Cons', 'Prod', 'Unknown')

DECLARATION OF ELEMENTS

 STATE VARIABLES
 DISCRETE
   # Header-Informationen
    Agent (MFrom)      := 'Unknown',  # Typ des Absenders
    Sender (INTEGER)   := -1,         # ID des Senders
    Receiver (INTEGER) := -1,         # ID des Empfaengers
    Type (MType)       := 'MUnknown', # Nachrichtentyp
    TStamp (REAL)      := -1,         # Zeitpunkt des Nachrichtenversands
    Deadline (REAL)    := -1,         # Optionale "Deadline"

   # Nachrichteninhalt
    Price (REAL)       := -1,         # Preis
    Quantity (REAL)    := -1          # Menge

END OF Message
```

B.7.5 Die mobile Komponente *PAction*

```
#-------------------------------------------------------------------------------
#
# Name:  "PAction" (Producer-Action)
#
# Art:   Mobile Komponente
#
# Beschreibung:
#
#   - Mob. Komponente zum Uebertragen von Aktionsinformationen von
#     "PBehav" nach "PActor".
```

```
#
# Verbindung(en) mit anderen Komponenten: n/a
#
#-------------------------------------------------------------------------

MOBILE COMPONENT PAction

LOCAL DEFINITIONS

 #---------------------------------------------------------------------
 # Aufzaehlungstyp fuer verschiedene Producer-Aktionen
 #   'PAOffer'   : Producer action Offer (Preisangebot senden)
 #   'PAAck'     : Producer action Acknowledge (Bestellung bestaetigen)
 #   'PAUnknwn'  : Producer action unknown (unbekannte Aktion)
 #---------------------------------------------------------------------

 VALUE SET PAType : ('PAOffer', 'PAAck', 'PAUnknwn')

DECLARATION OF ELEMENTS

 STATE VARIABLES
 DISCRETE
 # Header-Informationen
 Receiver (INTEGER) := -1,         # ID des Consumers, den diese Aktion
                                   # betrifft
 Type (PAType)      := 'PAUnknwn', # Typ der Aktion
 TStamp (REAL)      := -1.0,       # Zeitstempel
 Deadline (REAL)    := -1.0,       # Einzuhaltende Zeitfrist

 # Aktions-Inhalte
 Price (REAL)       := -1.0,
 Quantity (REAL)    := -1.0

END OF PAction
```

B.7.6 Die mobile Komponente *PriceEntry*

```
#-------------------------------------------------------------------------
#
# Name:  "PriceEntry"
#
# Art:   Mobile Komponente
#
# Beschreibung:
#
#   - Diese Komponente speichert einen Preiseintrag und wird bei der
#     Preisfestlegung im Rahmen der Cooperation-follower-Strategie verwendet
#
# Verbindung(en) mit anderen Komponenten: n/a
#
#-------------------------------------------------------------------------

MOBILE COMPONENT PriceEntry

DECLARATION OF ELEMENTS

  STATE VARIABLES
  DISCRETE
   Price (REAL) := -1.0,
   Profit (REAL) := -1.0

END OF PriceEntry
```

B.7.7 Die mobile Komponente *PThought*

```
#-------------------------------------------------------------------------
#
# Name:  "PThought" (Producer-Thought)
#
# Art:   Mobile Komponente
#
# Beschreibung:
#
#   - Diese mobile Komponente transportiert die Daten zwischen den
#     beteiligten Komponenten innerhalb des Producers. Beteiligte
#     Komp. sind "PSensor", "PPercept", "PCognit" und "PBehav".
#
# Verbindung(en) mit anderen Komponenten: n/a
#
#-------------------------------------------------------------------------

MOBILE COMPONENT PThought

LOCAL DEFINITIONS

 #-------------------------------------------------------------------------
 # Moegliche 'PThought'-Arten:
 #
 # PTMOpen   : Markt ist geoffnet worden
 # PTMClose  : Markt ist geschlossen worden
 # PTAsk     : Preisanfrage ist eingetroffen
 # PTOrder   : Bestellung ist eingetroffen
 # PTOffUpd  : Angebot wurde verschickt
 # PTAckUpd  : Bestaetigung wurde abgeschickt
 # PTPrcUpd  : Preis fuer naechste Periode wurde bestimmt
 # PTFindDir : Bestimme die Richtung, in der sich die gemeinsamen
 #             Gewinne bewegen
 # PTInfRdy  : Marktinformationen liegen in Statistic (<Statist>) bereit
 # PTUnknwn  : Unbekannter Typ
 #-------------------------------------------------------------------------
 VALUE SET PTType: ('PTMOpen', 'PTMClose', 'PTAsk', 'PTOrder', 'PTOffUpd',
                    'PTAckUpd','PTPrcUpd', 'PTInfRdy', 'PTFindDir',
                    'PTUnknwn')

DECLARATION OF ELEMENTS

STATE VARIABLES
 DISCRETE
  # Header-Informationen
  Sender (INTEGER) := -1,
  Type (PTType)    := 'PTUnknwn',
  TStamp (REAL)    := -1.0,
  Deadline (REAL)  := -1.0,

  # Thought-Inhalt
  Price (REAL)     := -1.0,
  Quantity (REAL)  := -1.0,
  Id (INTEGER)     := -1

END OF PThought
```

B.8 Parametrisierung des Modells für das Experiment in Abschnitt 8.3

```
SetVar /Market/Buyer[1]/CCognition/Pref[2] 0.92
SetVar /Market/Buyer[1]/CCognition/Utility[1] 66
SetVar /Market/Buyer[1]/CCognition/Utility[2] 65
SetVar /Market/Buyer[1]/CCognition/Utility[3] 63
SetVar /Market/Buyer[1]/CCognition/Utility[4] 61
SetVar /Market/Buyer[1]/CCognition/Utility[5] 59
SetVar /Market/Buyer[1]/CCognition/Utility[6] 57
SetVar /Market/Buyer[2]/CCognition/Pref[2] 0.93
SetVar /Market/Buyer[2]/CCognition/Utility[1] 64
SetVar /Market/Buyer[2]/CCognition/Utility[2] 63
SetVar /Market/Buyer[2]/CCognition/Utility[3] 61
SetVar /Market/Buyer[2]/CCognition/Utility[4] 59
SetVar /Market/Buyer[2]/CCognition/Utility[5] 57
SetVar /Market/Buyer[2]/CCognition/Utility[6] 55
SetVar /Market/Buyer[3]/CCognition/Pref[2] 0.94
SetVar /Market/Buyer[3]/CCognition/Utility[1] 62
SetVar /Market/Buyer[3]/CCognition/Utility[2] 61
SetVar /Market/Buyer[3]/CCognition/Utility[3] 59
SetVar /Market/Buyer[3]/CCognition/Utility[4] 57
SetVar /Market/Buyer[3]/CCognition/Utility[5] 55
SetVar /Market/Buyer[3]/CCognition/Utility[6] 53
SetVar /Market/Buyer[4]/CCognition/Pref[2] 0.95
SetVar /Market/Buyer[4]/CCognition/Utility[1] 61
SetVar /Market/Buyer[4]/CCognition/Utility[2] 59
SetVar /Market/Buyer[4]/CCognition/Utility[3] 57
SetVar /Market/Buyer[4]/CCognition/Utility[4] 55
SetVar /Market/Buyer[4]/CCognition/Utility[5] 53
SetVar /Market/Buyer[4]/CCognition/Utility[6] 51
SetVar /Market/Buyer[5]/CCognition/Pref[2] 0.96
SetVar /Market/Buyer[5]/CCognition/Utility[1] 60
SetVar /Market/Buyer[5]/CCognition/Utility[2] 58
SetVar /Market/Buyer[5]/CCognition/Utility[3] 56
SetVar /Market/Buyer[5]/CCognition/Utility[4] 54
SetVar /Market/Buyer[5]/CCognition/Utility[5] 52
SetVar /Market/Buyer[5]/CCognition/Utility[6] 50
SetVar /Market/Buyer[6]/CCognition/Pref[2] 0.97
SetVar /Market/Buyer[6]/CCognition/Utility[1] 59
SetVar /Market/Buyer[6]/CCognition/Utility[2] 57
SetVar /Market/Buyer[6]/CCognition/Utility[3] 55
SetVar /Market/Buyer[6]/CCognition/Utility[4] 53
SetVar /Market/Buyer[6]/CCognition/Utility[5] 51
SetVar /Market/Buyer[6]/CCognition/Utility[6] 49
SetVar /Market/Buyer[7]/CCognition/Pref[2] 0.98
SetVar /Market/Buyer[7]/CCognition/Utility[1] 58
SetVar /Market/Buyer[7]/CCognition/Utility[2] 56
SetVar /Market/Buyer[7]/CCognition/Utility[3] 54
SetVar /Market/Buyer[7]/CCognition/Utility[4] 52
SetVar /Market/Buyer[7]/CCognition/Utility[5] 50
SetVar /Market/Buyer[7]/CCognition/Utility[6] 48
SetVar /Market/Buyer[8]/CCognition/Pref[2] 0.99
SetVar /Market/Buyer[8]/CCognition/Utility[1] 56
SetVar /Market/Buyer[8]/CCognition/Utility[2] 55
SetVar /Market/Buyer[8]/CCognition/Utility[3] 53
SetVar /Market/Buyer[8]/CCognition/Utility[4] 51
SetVar /Market/Buyer[8]/CCognition/Utility[5] 49
SetVar /Market/Buyer[8]/CCognition/Utility[6] 47
SetVar /Market/Buyer[9]/CCognition/Pref[1] 0.92
SetVar /Market/Buyer[9]/CCognition/Utility[1] 66
SetVar /Market/Buyer[9]/CCognition/Utility[2] 65
SetVar /Market/Buyer[9]/CCognition/Utility[3] 63
SetVar /Market/Buyer[9]/CCognition/Utility[4] 61
```

```
SetVar /Market/Buyer[9]/CCognition/Utility[5] 59
SetVar /Market/Buyer[9]/CCognition/Utility[6] 57
SetVar /Market/Buyer[10]/CCognition/Pref[1] 0.93
SetVar /Market/Buyer[10]/CCognition/Utility[1] 64
SetVar /Market/Buyer[10]/CCognition/Utility[2] 63
SetVar /Market/Buyer[10]/CCognition/Utility[3] 61
SetVar /Market/Buyer[10]/CCognition/Utility[4] 59
SetVar /Market/Buyer[10]/CCognition/Utility[5] 57
SetVar /Market/Buyer[10]/CCognition/Utility[6] 55
SetVar /Market/Buyer[11]/CCognition/Pref[1] 0.94
SetVar /Market/Buyer[11]/CCognition/Utility[1] 62
SetVar /Market/Buyer[11]/CCognition/Utility[2] 61
SetVar /Market/Buyer[11]/CCognition/Utility[3] 59
SetVar /Market/Buyer[11]/CCognition/Utility[4] 57
SetVar /Market/Buyer[11]/CCognition/Utility[5] 55
SetVar /Market/Buyer[11]/CCognition/Utility[6] 53
SetVar /Market/Buyer[12]/CCognition/Pref[1] 0.95
SetVar /Market/Buyer[12]/CCognition/Utility[1] 61
SetVar /Market/Buyer[12]/CCognition/Utility[2] 59
SetVar /Market/Buyer[12]/CCognition/Utility[3] 57
SetVar /Market/Buyer[12]/CCognition/Utility[4] 55
SetVar /Market/Buyer[12]/CCognition/Utility[5] 53
SetVar /Market/Buyer[12]/CCognition/Utility[6] 51
SetVar /Market/Buyer[13]/CCognition/Pref[1] 0.96
SetVar /Market/Buyer[13]/CCognition/Utility[1] 60
SetVar /Market/Buyer[13]/CCognition/Utility[2] 58
SetVar /Market/Buyer[13]/CCognition/Utility[3] 56
SetVar /Market/Buyer[13]/CCognition/Utility[4] 54
SetVar /Market/Buyer[13]/CCognition/Utility[5] 52
SetVar /Market/Buyer[13]/CCognition/Utility[6] 50
SetVar /Market/Buyer[14]/CCognition/Pref[1] 0.97
SetVar /Market/Buyer[14]/CCognition/Utility[1] 59
SetVar /Market/Buyer[14]/CCognition/Utility[2] 57
SetVar /Market/Buyer[14]/CCognition/Utility[3] 55
SetVar /Market/Buyer[14]/CCognition/Utility[4] 53
SetVar /Market/Buyer[14]/CCognition/Utility[5] 51
SetVar /Market/Buyer[14]/CCognition/Utility[6] 49
SetVar /Market/Buyer[15]/CCognition/Pref[1] 0.98
SetVar /Market/Buyer[15]/CCognition/Utility[1] 58
SetVar /Market/Buyer[15]/CCognition/Utility[2] 56
SetVar /Market/Buyer[15]/CCognition/Utility[3] 54
SetVar /Market/Buyer[15]/CCognition/Utility[4] 52
SetVar /Market/Buyer[15]/CCognition/Utility[5] 50
SetVar /Market/Buyer[15]/CCognition/Utility[6] 48
SetVar /Market/Buyer[16]/CCognition/Pref[1] 0.99
SetVar /Market/Buyer[16]/CCognition/Utility[1] 56
SetVar /Market/Buyer[16]/CCognition/Utility[2] 55
SetVar /Market/Buyer[16]/CCognition/Utility[3] 53
SetVar /Market/Buyer[16]/CCognition/Utility[4] 51
SetVar /Market/Buyer[16]/CCognition/Utility[5] 49
SetVar /Market/Buyer[16]/CCognition/Utility[6] 47
SetVar /Market/Seller[1]/PBehaviour/Debug TRUE
SetVar /Market/Seller[2]/PBehaviour/Debug TRUE
SetVar /Market/Seller[1]/PBehaviour/Protocol TRUE
SetVar /Market/Seller[2]/PBehaviour/Protocol TRUE
SetVar /Market/Seller[1]/PCognition/kUnitCost 47
SetVar /Market/Seller[2]/PCognition/kUnitCost 47
SetVar /Market/Seller[1]/PCognition/Protocol TRUE
SetVar /Market/Seller[2]/PCognition/Protocol TRUE
SetVar /Market/Seller[2]/PCognition/kLowLimit 48
SetVar /Market/Seller[1]/PCognition/kLowLimit 48
SetVar /Market/Seller[1]/PBehaviour/InitCredit 150.000000
SetVar /Market/Seller[1]/PBehaviour/Personality 'calm'
SetVar /Market/Seller[2]/PBehaviour/InitCredit 100.000000
SetVar /Market/Seller[2]/PBehaviour/Personality 'calm'
SetVar /Market/Seller[1]/PCognition/ActPrice 57.000000
SetVar /Market/Seller[2]/PCognition/ActPrice 55.000000
```

ANHANG C QUELLCODE DES MODELLS *LERNGRUPPE*

C.1 Die Strukturkomponente *Lerngruppe*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    Lerngruppe
#
#  Art:     High-Level Komponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Die Komponente für das gesamte Modell. Hier wird die Anzahl der
#      Individuen bzw. Gruppenagenten eingestellt und die Agenten werden
#      initialisiert (ID, Protokoll)
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - untergeordnete Komponenten:
#       + 36 Individuen
#       + 36 Gruppenagenten
#       + Connector
#       + Statistik
#
#-------------------------------------------------------------------------------

HIGH LEVEL COMPONENT Lerngruppe

DIMENSIONS
  #-----------------------------------------------
  # Anzahl der Individuen
  #-----------------------------------------------
  AnzIndividuen := 36,

  #-----------------------------------------------
  #  Obergrenze für die Anzahl der Gruppenagenten
  #-----------------------------------------------
  AnzGruppenagenten := 36

SUBCOMPONENTS
  ARRAY [AnzIndividuen] Individuum,       # Instanzen der Klasse Individuum
  ARRAY [AnzGruppenagenten] Gruppenagent, # Instanzen der Klasse Gruppenagent
  Connector,                              # Connector
  Statistik                               # Informationssammlung/-aufbereitung


COMPONENT CONNECTIONS

  #-------------------------------------------------
  # Nachrichtenverbindung Individuum --> Gruppenagent
  #-------------------------------------------------
  Connector.EingangI{ALL i} --> Individuum[i].Ausgang;
  Connector.AusgangG{i OF 1..AnzGruppenagenten} --> Gruppenagent[i].Eingang;

  #-------------------------------------------------
  # Nachrichtenverbindung Gruppenagent --> Individuum
  #-------------------------------------------------
  Connector.EingangG{ALL i} --> Gruppenagent[i].Ausgang;
  Connector.AusgangI{i OF 1..AnzIndividuen} --> Individuum[i].Eingang;

  #-------------------------------------------------
  # Informationsbeschaffung für das Blackboard
  #-------------------------------------------------
```

```
  Individuum{i OF 1..AnzIndividuen}.Wissen --> Connector.Wissen_S[i];
  Individuum{i OF 1..AnzIndividuen}.SozKompetenz
                                        --> Connector.SozKompetenz_S[i];

  Gruppenagent{i OF 1..AnzGruppenagenten}.QualitaetG
                                    --> Connector.QualitaetG_S[i];
  Gruppenagent{i OF 1..AnzGruppenagenten}.GruppenagentAktiv
                                    --> Connector.GruppenagentAktiv_S[i];


  #-------------------------------------------------
  # Blackboard (für Individuen lesbar)
  #-------------------------------------------------
  Connector.QualitaetG{ALL i}        --> Individuum{ALL}.QualitaetG[i];
  Connector.GruppenagentAktiv{ALL i} --> Individuum{ALL}.GruppenagentAktiv[i];

  #-------------------------------------------------
  # Blackboard (für Gruppenagenten lesbar)
  #-------------------------------------------------
  Connector.Wissen{ALL i}       --> Gruppenagent{ALL}.Wissen[i];
  Connector.SozKompetenz{ALL i} --> Gruppenagent{ALL}.SozKompetenz[i];


  #---------------------------------
  # Informationen für die Statistik
  #---------------------------------

  #-------------------------------------
  # Informationen über Gruppen(agenten)
  #-------------------------------------
  Gruppenagent{ALL i}.GruppenagentAktiv --> Statistik.GruppenagentAktiv[i];
  Gruppenagent{ALL i}.GroesseG          --> Statistik.GroesseG[i];
  Gruppenagent{ALL i}.RelWisG           --> Statistik.RelWisG[i];
  Gruppenagent{ALL i}.RelSozG           --> Statistik.RelSozG[i];
  Gruppenagent{ALL i}.QualitaetG        --> Statistik.QualitaetG[i];

  #-------------------------------
  # Informationen über Individuen
  #-------------------------------
  Individuum{ALL i}.Gruppenmitglied --> Statistik.Gruppenmitglied[i];
  Individuum{ALL i}.GruppenID       --> Statistik.GruppenID[i];
  Individuum{ALL i}.Wissen          --> Statistik.Wissen[i];
  Individuum{ALL i}.SozAkt          --> Statistik.SozAkt[i];
  Individuum{ALL i}.SozKompetenz    --> Statistik.SozKompetenz[i];
  Individuum{ALL i}.QualitaetI      --> Statistik.QualitaetI[i];
  Individuum{ALL i}.TInGruppe       --> Statistik.TInGruppe[i];

  #-----------------------------------------
  # Statistik-Informationen an Gruppenagenten
  #-----------------------------------------
  Statistik.DurchschnWissen       --> Gruppenagent{ALL}.DurchschnWissen;
  Statistik.DurchschnSozKompetenz --> Gruppenagent{ALL}.DurchschnSozKompetenz;
  Statistik.RelWisGMax            --> Gruppenagent{ALL}.RelWisGMax;
  Statistik.RelSozGMax            --> Gruppenagent{ALL}.RelSozGMax;

  #--------------------------------------
  # Statistik-Informationen an Individuen
  #--------------------------------------
  Statistik.WissenMax       --> Individuum{ALL}.WissenMax;
  Statistik.SozKompetenzMax --> Individuum{ALL}.SozKompetenzMax;


INITIALIZE

  #-----------------------------------
  # Initialisierung der Individuen
  #-----------------------------------
```

```
  Individuum{ALL i}.ID      := i;
  Individuum{ALL}.Protokoll := FALSE;

  #-----------------------------------
  # Initialisierung der Gruppenagenten
  #-----------------------------------
  Gruppenagent{ALL i}.ID      := i;
  Gruppenagent{ALL}.Protokoll := FALSE;

  #-----------------------------------
  # Initialisierung der Statistik
  #-----------------------------------
  Statistik.Protokoll := FALSE;

END OF Lerngruppe
```

C.2 Die Individualagenten

C.2.1 Die Strukturkomponente *Individuum*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    Individuum
#
#  Art:     High-Level Komponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Die Komponente ist eine Klassenschablone für Individuen
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Connector an SensorI
#       + QualitaetG[]
#       + GruppenagentAktiv []
#       + Nachrichten
#
#    - Statistik an KognitionI
#       + DurchschnWissen:       Durchschnitt des Wissens aller Individuen
#       + DurchschnSozKompetenz: Durchschnitt der sozialen Zufriedenheit aller
#                                Individuen
#
#    - übergeordnete Komponente(n):
#       + Lerngruppe
#
#    - untegeordnete Komponente(n):
#       + SensorI
#       + WahrnehmungI
#       + KognitionI
#       + EmotionI
#       + StatusI
#       + VerhaltenI
#       + AktorI
#
#-------------------------------------------------------------------------------
```

```
HIGH LEVEL COMPONENT Individuum

SUBCOMPONENTS
  SensorI,
  WahrnehmungI,
  KognitionI,
  StatusI,
  EmotionI,
  VerhaltenI,
  AktorI

INPUT CONNECTIONS
  ID        --> ( SensorI.ID,
                  WahrnehmungI.ID,
                  KognitionI.ID,
                  StatusI.ID,
                  EmotionI.ID,
                  VerhaltenI.ID,
                  AktorI.ID );

  Protokoll --> ( SensorI.Protokoll,
                  WahrnehmungI.Protokoll,
                  KognitionI.Protokoll,
                  StatusI.Protokoll,
                  EmotionI.Protokoll,
                  VerhaltenI.Protokoll,
                  AktorI.Protokoll );

  #---------------------------------------
  # Nachrichten-Schnittstellen (Connector)
  #---------------------------------------
  Eingang --> SensorI.Eingang;
  Ausgang --> AktorI.Ausgang;

  #-------------------------------------------------------------
  # Informationen über Gruppenagenten vom Blackboard (Connector)
  #-------------------------------------------------------------
  QualitaetG        --> SensorI.QualitaetG_S;
  GruppenagentAktiv --> SensorI.GruppenagentAktiv_S;

  #-------------------------------------------------------------
  # Informationen über Individuen (nur fuer Statistische Zwecke)
  #-------------------------------------------------------------
  WissenMax       --> StatusI.WissenMax;
  SozKompetenzMax --> StatusI.SozKompetenzMax;

OUTPUT EQUIVALENCES
  #----------------------------------
  # Informationen über das Individuum
  #----------------------------------
  Wissen          := KognitionI.Wissen;
  SozAkt          := StatusI.SozAkt;
  SozKompetenz    := StatusI.SozKompetenz;
  Gruppenmitglied := StatusI.Gruppenmitglied;
  GruppenID       := KognitionI.GruppenID;
  QualitaetI      := StatusI.QualitaetI;
  TInGruppe       := StatusI.TInGruppe;

COMPONENT CONNECTIONS
  #-------------------------------------------------------------------------------
  # SensorI an WahrnehmungI
  #-------------------------------------------------------------------------------

  #-------------
  # Nachrichten
  #-------------
  SensorI.Ausgang --> WahrnehmungI.Eingang;
```

```
#-------------------------
# Blackboard-Informatioen
#-------------------------
SensorI.QualitaetG{ALL i}        --> WahrnehmungI.QualitaetG_S[i];
SensorI.GruppenagentAktiv{ALL i} --> WahrnehmungI.GruppenagentAktiv_S[i];

#-----------------------------------------------------------------------------
# WahrnehmungI an KognitionI
#-----------------------------------------------------------------------------

#-------------
# Nachrichten
#-------------
WahrnehmungI.Ausgang --> KognitionI.Eingang;

#-------------------------
# Blackboard-Informatioen
#-------------------------
WahrnehmungI.QualitaetG               --> KognitionI.QualitaetG;
WahrnehmungI.GruppenagentAktiv{ALL i} --> KognitionI.GruppenagentAktiv_S[i];

#-----------------------------------------------------------------------------
# WahrnehmungI an StatusI
#-----------------------------------------------------------------------------

#-------------------------
# Blackboard-Informatioen
#-------------------------
WahrnehmungI.QualitaetG --> StatusI.QualitaetG;

#-----------------------------------------------------------------------------
# KognitionI an WahrnehmungI
#-----------------------------------------------------------------------------

#----------------------
# Gruppenzugehörigkeit
#----------------------
KognitionI.GruppenID --> WahrnehmungI.GruppenID;

#-----------------------------------------------------------------------------
# KognitionI an StatusI
#-----------------------------------------------------------------------------

#-------------------------------
# Änderung der Gruppensituation
#-------------------------------
KognitionI.NeueGruppensituation --> StatusI.NeueGruppensituation;

#----------------------
# Gruppenzugehörigkeit
#----------------------
KognitionI.GruppenID --> StatusI.GruppenID;

#--------------
# Gruppengröße
#--------------
KognitionI.GroesseG --> StatusI.GroesseG;

#---------------
# Echte Gruppe?
#---------------
KognitionI.echteGruppe --> StatusI.echteGruppe;

#--------------
# Gesamtwissen
#--------------
KognitionI.Wissen --> StatusI.Wissen;
```

```
#-------------------------------------------------------------------------------
# KognitionI an VerhaltenI
#-------------------------------------------------------------------------------

#--------------------------------------
# Entscheidungsrelevante Informationen
#--------------------------------------
KognitionI.Wissen                   --> VerhaltenI.Wissen;
KognitionI.WisAkt                   --> VerhaltenI.WisAkt;
KognitionI.GroesseG                 --> VerhaltenI.GroesseG;
KognitionI.TUnechteGruppe           --> VerhaltenI.TUnechteGruppe;
KognitionI.TEintritt                --> VerhaltenI.TEintritt;
KognitionI.GruppenID                --> VerhaltenI.GruppenID;
KognitionI.GruppenagentAktiv{ALL i} --> VerhaltenI.GruppenagentAktiv[i];

#----------------------------------
# Zusage von Gruppenagent erhalten
#----------------------------------
KognitionI.ZusageErhalten --> VerhaltenI.ZusageErhalten;

#----------------------------------
# Absage von Gruppenagent erhalten
#----------------------------------
KognitionI.AbsageErhalten -->  VerhaltenI.AbsageErhalten;

#----------------------------------
# freier Gruppenangent bestimmt
#----------------------------------
KognitionI.GruppenagentBestimmt --> VerhaltenI.GruppenagentBestimmt;

#----------------------------------
# Bewerbungsliste bestimmt
#----------------------------------
KognitionI.BewerbungslisteBestimmt --> VerhaltenI.BewerbungslisteBestimmt;

#-------------------------------------------------------------------------------
# EmotionI an VerhaltenI
#-------------------------------------------------------------------------------

# vorlaeufig leer!

#-------------------------------------------------------------------------------
# StatusI an VerhaltenI
#-------------------------------------------------------------------------------

#------------------------
# Soziale Zufriedenheit
#------------------------
StatusI.SozAkt --> VerhaltenI.SozAkt;

#-------------------
# Sozialbeduerfnis
#-------------------
StatusI.SozBed --> VerhaltenI.SozBed;

#---------------------------------------------
# Soziale Anlage (Geselligkeit) des Individuums
#---------------------------------------------
StatusI.SozAnlage --> VerhaltenI.SozAnlage;

#--------------------
# Soziale Kompetenz
#--------------------
StatusI.SozKompetenz --> VerhaltenI.SozKompetenz;
```

```
  #-------------------
  # Gruppenmitglied?
  #-------------------
  StatusI.Gruppenmitglied --> VerhaltenI.Gruppenmitglied;

  #-------------------------------------------------------------------------------
  # VerhaltenI an KognitionI
  #-------------------------------------------------------------------------------

  #---------------------------------
  # Anzahl der generierten Anfragen
  #---------------------------------
  VerhaltenI.GenerierteAnfragen --> KognitionI.GenerierteAnfragen;

  #--------------------------------------
  # Bestimme ersten freien Gruppenagenten
  #--------------------------------------
  VerhaltenI.BestimmeGruppenagent --> KognitionI.BestimmeGruppenagent;

  #-------------------------------
  # Verlassen/Auflösen der Gruppe
  #-------------------------------
  VerhaltenI.GruppeVerlassen --> KognitionI.GruppeVerlassen;

  #-----------------------
  # Bewerbungsreihenfolge
  #-----------------------
  VerhaltenI.Rangfolge --> KognitionI.Rangfolge_S;

  #-------------------------------------------------------------------------------
  # VerhaltenI an AktorI
  #-------------------------------------------------------------------------------

  #------------------------
  # Ausführungsanordnungen
  #------------------------
  VerhaltenI.Ausgang --> AktorI.Eingang;

END OF Individuum
```

C.2.2 Die Komponente *SensorI*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    SensorI
#
#  Art:     Basiskomponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Schnittstelle zwischen Individuum und Aussenwelt:
#       + Aufnehmen von Nachrichten
#       + Informationen vom Blackboard
#
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Connector an SensorI
#       + Nachricht
#       + QualitaetG []
```

```
#       + GruppenagentAktiv []
#
#    - SensorI an WahrnehmungI
#       + Nachricht
#       + QualitaetG
#
#-------------------------------------------------------------------------------

BASIC COMPONENT SensorI

MOBILE SUBCOMPONENTS OF CLASS Nachricht

USE OF UNITS
  UNIT [GQ]  = BASIS
  TIMEUNIT   = [h]

LOCAL DEFINITIONS

DIMENSIONS
  #-----------------------------------------------
  #  Obergrenze für die Anzahl der Gruppenagenten
  #-----------------------------------------------
  AnzGruppenagenten := 36

VALUE SET
  #-------------------------------------------------------------------------------
  # Nachrichtentypen:
  #  - Individuum an Gruppenagent
  #     + Anfrage:       Individuum fragt Gruppenagent nach Gruppeninformation
  #     + Bewerbung:     Individuum bewirbt sich um Aufnahme in die Gruppe
  #     + Ausstieg:      Individuum verlaesst die Gruppe
  #
  # - Gruppenagent an Individuum
  #     + Auskunft:      Gruppenagent antwortet auf Anfrage des Individuums
  #     + Absage:        Individuum wird nicht aufgenommen
  #     + Zusage:        Individuum wird in die Gruppe aufgenommen
  #     + Ausschluss:    Grupppenagent schliesst Individuum aus der Gruppe aus
  #     + Update:        Aenderung der diskreten, für die Gruppenmitglieder
  #                      relevanten Informationen (GroesseG)
  #-------------------------------------------------------------------------------
  Nachrichtentyp : ('Anfrage', 'Bewerbung', 'Ausstieg',
                    'Auskunft', 'Absage', 'Zusage', 'Ausschluss', 'Update')

DECLARATION OF ELEMENTS

DEPENDENT VARIABLES
DISCRETE
  #--------------------------
  # Blackboard-Informationen
  #--------------------------
  ARRAY [AnzGruppenagenten] GruppenagentAktiv (LOGICAL) := FALSE

CONTINUOUS
  #--------------------------
  # Blackboard-Informationen
  #--------------------------
  ARRAY [AnzGruppenagenten] QualitaetG (REAL[GQ]) := 0 [GQ]

SENSOR VARIABLES
DISCRETE
  #-----------------------------------
  # Informationen vom Blackboard lesen
  #-----------------------------------
  ARRAY [AnzGruppenagenten] GruppenagentAktiv_S (LOGICAL),

  ID              (INTEGER) := 0,
  Protokoll       (LOGICAL) := TRUE
```

```
CONTINUOUS
  #-----------------------------------
  # Informationen vom Blackboard lesen
  #-----------------------------------
  ARRAY [AnzGruppenagenten] QualitaetG_S (REAL[GQ])


LOCATIONS
  #-------------------------------------------
  # Weitergabe von Nachrichten an Wahrnehmung
  #-------------------------------------------
  Ausgang (Nachricht) := 0 Nachricht

SENSOR LOCATIONS
  #-------------------------------------------
  # Eingang für Nachrichten von Gruppenagenten
  #-------------------------------------------
  Eingang (Nachricht) := 0 Nachricht

DYNAMIC BEHAVIOUR

  #-------------------------------------------------
  # Bereitstellung der Informationen vom Blackboard
  #-------------------------------------------------
  QualitaetG {i OF 1..AnzGruppenagenten} := QualitaetG_S[i];
  GruppenagentAktiv {i OF 1..AnzGruppenagenten} := GruppenagentAktiv_S[i];

  #-----------------------------------------------------------------------------
  #
  # Ereignis 1 : Weiterleitung der eingehenden Nachrichten
  #
  #-----------------------------------------------------------------------------
  WHENEVER NUMBER(Eingang) > 0 DO
    Ausgang^ : FROM Eingang GET Nachricht{ALL};
      IF Protokoll DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente SensorI, ", T, ID);
      DISPLAY("Ereignis 1\n");
      DISPLAY("Individuum hat neue Nachricht(en) empfangen\n");
    END
  END

END OF SensorI
```

C.2.3 Die Komponente *WahrnehmungI*

```
#-----------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    WahrnehmungI
#
#  Art:     Basiskomponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Filterung der Informationen vom Blackboard
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - SensorI an WahrnehmungI
#       + QualitaetG []
#       + GruppenagentAktiv []
```

```
#
#    - KognitionI an WahrnehmungI
#       + GruppenID
#
#    - WahrnehmungI an KognitionI
#       + Nachrichten
#       + Qualitaet[GruppenID]
#
#-------------------------------------------------------------------------------

BASIC COMPONENT WahrnehmungI

MOBILE SUBCOMPONENTS OF CLASS Nachricht

USE OF UNITS
  UNIT [GQ]  = BASIS
  TIMEUNIT   = [h]

LOCAL DEFINITIONS
DIMENSIONS
  #-----------------------------------------------
  #  Obergrenze für die Anzahl der Gruppenagenten
  #-----------------------------------------------
  AnzGruppenagenten := 36

VALUE SET
  #-------------------------------------------------------------------------------
  # Nachrichtentypen:
  #  - Individuum an Gruppenagent
  #     + Anfrage:       Individuum fragt Gruppenagent nach Gruppeninformation
  #     + Bewerbung:     Individuum bewirbt sich um Aufnahme in die Gruppe
  #     + Ausstieg:      Individuum verlaesst die Gruppe
  #
  # - Gruppenagent an Individuum
  #     + Auskunft:      Gruppenagent antwortet auf Anfrage des Individuums
  #     + Absage:        Individuum wird nicht aufgenommen
  #     + Zusage:        Individuum wird in die Gruppe aufgenommen
  #     + Ausschluss:    Grupppenagent schliesst Individuum aus der Gruppe aus
  #     + Update:        Aenderung der diskreten, für die Gruppenmitglieder
  #                      relevanten Informationen (GroesseG)
  #-------------------------------------------------------------------------------
  Nachrichtentyp : ('Anfrage', 'Bewerbung', 'Ausstieg',
                    'Auskunft', 'Absage', 'Zusage', 'Ausschluss', 'Update')

DECLARATION OF ELEMENTS

DEPENDENT VARIABLES
DISCRETE
  #--------------------------
  # Blackboard-Informationen
  #--------------------------
  ARRAY [AnzGruppenagenten] GruppenagentAktiv (LOGICAL)

CONTINUOUS
  #-----------------------------------
  # gefilterte Blackboard-Information
  #-----------------------------------
  QualitaetG (REAL[GQ]) := 0 [GQ]

SENSOR VARIABLES
DISCRETE
  #-----------------------------------------
  # Informationen vom Sensor/Blackboard lesen
  #-----------------------------------------
  ARRAY [AnzGruppenagenten] GruppenagentAktiv_S (LOGICAL),
```

```
  #-------------------------
  # GruppenID aus Kognition
  #-------------------------
  GruppenID   (INTEGER),

  ID          (INTEGER) := 0,
  Protokoll   (LOGICAL) := TRUE

CONTINUOUS
  #----------------------------------------
  # Informationen vom Sensor/Blackboard lesen
  #----------------------------------------
  ARRAY [AnzGruppenagenten] QualitaetG_S (REAL[GQ])

LOCATIONS
  #---------------------------------------
  # Weitergabe von Nachrichten an Kognition
  #---------------------------------------
  Ausgang (Nachricht) := 0 Nachricht

SENSOR LOCATIONS
  #-------------------------------------------------
  # Eingang für Nachrichten von SensorI/Gruppenagent
  #-------------------------------------------------
  Eingang (Nachricht) := 0 Nachricht

DYNAMIC BEHAVIOUR

  #-------------------------------------
  # Informationsfilterung/-aufbereitung
  #-------------------------------------
  IF GruppenID <> 0 DO
    QualitaetG := QualitaetG_S[GruppenID];
  END
  ELSE DO
    QualitaetG := 1 [GQ];
  END

  GruppenagentAktiv {i OF 1..AnzGruppenagenten} := GruppenagentAktiv_S[i];

  #-----------------------------------------------------------------------------
  #
  # Ereignis 1 :  Weiterleitung der eingehenden Nachricht(en)
  #
  #-----------------------------------------------------------------------------
  WHENEVER NUMBER(Eingang) > 0 DO
    Ausgang^ : FROM Eingang GET Nachricht{ALL};
  END

END OF WahrnehmungI
```

C.2.4 Die Komponente *KognitionI*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    KognitionI
#
#  Art:     Basiskomponente
#
#  Version: 0
#
#  Beschreibung:
```

```
#
#    - Verarbeiten der Informationen der Wahrnehmung/Blackboard
#    - Entwicklung des Wissens
#
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - WahrnehmungI an KognitionI
#       + Nachrichten
#       + Qualitaet [GruppenID]
#       + GruppenagentAktiv []
#
#     - VerhaltenI an KognitionI
#        + GenerierteAnfragen
#        + BestimmeGruppenagent
#        + GruppeVerlassen
#        + (Rangfolge)
#
#    - KognitionI an WahrnehmungI
#       + GruppenID
#    - KognitionI an StatusI
#       + GruppenID
#       + NeueGruppensituation
#
#    - KognitionI an VerhaltenI
#       + GroesseG
#       + TUnechteGruppe
#       + AktiveGruppenagenten
#       + ZusageErhalten
#       + AbsageErhalten
#       + GruppenagentBestimmt
#       + BewerbungslisteBestimmt
#
#-------------------------------------------------------------------------------

BASIC COMPONENT KognitionI

MOBILE SUBCOMPONENTS OF CLASS Nachricht, Gruppe

USE OF UNITS
  UNIT [WI]  = BASIS
  UNIT [IQ]  = BASIS
  UNIT [GQ]  = BASIS
  TIMEUNIT   = [h]

LOCAL DEFINITIONS

DIMENSIONS
  #-----------------------------------------------
  #  Obergrenze für die Anzahl der Gruppenagenten
  #-----------------------------------------------
  AnzGruppenagenten := 36

VALUE SET
  #-------------------------------------------------------------------------------
  # Nachrichtentypen:
  #  - Individuum an Gruppenagent
  #     + Anfrage:       Individuum fragt Gruppenagent nach Gruppeninformation
  #     + Bewerbung:     Individuum bewirbt sich um Aufnahme in die Gruppe
  #     + Ausstieg:      Individuum verlaesst die Gruppe
  #
  # - Gruppenagent an Individuum
  #     + Auskunft:      Gruppenagent antwortet auf Anfrage des Individuums
  #     + Absage:        Individuum wird nicht aufgenommen
  #     + Zusage:        Individuum wird in die Gruppe aufgenommen
  #     + Ausschluss:    Grupppenagent schliesst Individuum aus der Gruppe aus
  #     + Update:        Aenderung der diskreten, für die Gruppenmitglieder
```

```
  #                      relevanten Informationen (GroesseG)
  #-------------------------------------------------------------------------
  Nachrichtentyp : ('Anfrage', 'Bewerbung', 'Ausstieg',
                    'Auskunft', 'Absage', 'Zusage', 'Ausschluss', 'Update')

  #-------------------------------------------------------------------------
  # FreierGruppenagent()
  #
  # Typ:              Funktion
  # Aufrufparameter: ARRAY AktiveGruppenagenten
  # Rückgabewert:    INTEGER
  #
  # Beschreibung: Diese Funktion liefert die ID des ersten freien Gruppen-
  #               agenten zurück
  #-------------------------------------------------------------------------
  FUNCTION FreierGruppenagent (ARRAY [n] LOGICAL: AktiveGruppenagenten
                               --> INTEGER)
    DECLARE freieID (INTEGER) := 0
  BEGIN
    FOR i REVERSE FROM AnzGruppenagenten TO 1
    REPEAT
      IF NOT AktiveGruppenagenten[i] DO
        freieID := i;
      END
    END_LOOP
    RETURN (freieID);
  END_FUNC

  #-------------------------------------------------------------------------
  # EingegangeneAuskuenfte()
  #
  # Typ:              Funktion
  # Aufrufparameter: LOCATION Eingang
  # Rückgabewert:    INTEGER
  #
  # Beschreibung: Diese Funktion liefert die Anzahl der Nachrichten des Typs
  #               'Auskunft' in der Location Eingang zurueck
  #-------------------------------------------------------------------------
  FUNCTION EingegangeneAuskuenfte (LOCATION FOR Nachricht: Eingang --> INTEGER)
    DECLARE AnzahlAuskuenfte (INTEGER) := 0
  BEGIN
    FOR i FROM 1 TO NUMBER(Eingang)
    REPEAT
      IF Eingang:Nachricht[i].Typ = 'Auskunft' DO
        AnzahlAuskuenfte := AnzahlAuskuenfte + 1;
      END
    END_LOOP
    RETURN (AnzahlAuskuenfte);
  END_FUNC

DECLARATION OF ELEMENTS

CONSTANTS
  #-----------------------------------------------------
  # Maximalwert für in einer Lernphase erreichbares Wissen
  #-----------------------------------------------------
  WisAktMax (REAL[WI])   := 1.0 [WI],

  #-----------------------------------------------------
  # Parameter für normalen Wissenszuwachs
  #-----------------------------------------------------
  WisNormal (REAL[WI/h]) := 0.5 [WI/h],
  a (REAL[1/(WI*IQ*GQ)])   := 1 [1/(WI*IQ*GQ)],

  #-----------------------------------------------------
  # Wissen zu Begin jeder neuen Lernphase
  #-----------------------------------------------------
```

```
  Grundwissen (REAL[WI]) := 0.05 [WI]

STATE VARIABLES
DISCRETE
  #-------------------------------------------------
  # Gruppenzugehörigkeit, gruppenrelevante Attribute
  #-------------------------------------------------
  GruppenID      (INTEGER) := 0,       # Gruppenzugehoerigkeit
  GroesseG       (INTEGER) := 0,       # Gruppengroesse
  neueGLernphase (LOGICAL) := FALSE,   # Das Individuum will eine Gruppen-
                                       # Lernphase starten
  echteGruppe    (LOGICAL) := FALSE,   # Individuum ist Mitglied in einer
                                       # echten Gruppe / hat Lernpartner
  TUnechteGruppe (REAL[h]) := 0 [h],   # Zeitpunkt der Entstehung der unechten
                                       # Gruppe
  TEintritt      (REAL[h]) := 0 [h],   # Zeitpunkt des Gruppeneintritts
  AnzahlZusagen  (INTEGER) := 0,       # Bisher erhaltene Zusagen

  #---------------------------------
  # Die Auskuenfte werden bearbeitet
  #---------------------------------
  AuskuenfteBearbeiten (LOGICAL) := FALSE,

  #-------------------------------------------------
  # Die Intelligenz bestimmt die Lerngeschwindigkeit
  #-------------------------------------------------
  Intelligenz (REAL[IQ]) := 0 [IQ],

  #-------------------------------------------------------
  # Festlegung der Zufallsverteilung fuer die Intelligenz
  #-------------------------------------------------------
  AuswahlVerteilung (INTEGER) := 0,

  # Hilfsgroessen
  ARRAY [AnzGruppenagenten] RangfolgeID (INTEGER)  := 0 ,
  ARRAY [AnzGruppenagenten] RangfolgeQu (REAL[GQ]) := 0 [GQ]

CONTINUOUS
  #---------------
  # Wissen
  #---------------
  Wissen (REAL[WI]) := 0 [WI],      # Gesamtwissen des Individuums
  WisAkt (REAL[WI]) := 0 [WI]       # Wissen, das das Individuum in der
                                    # aktuellen Lernphase bereits gelernt
DEPENDENT VARIABLES
DISCRETE
  #-------------------------------------------------------------------------
  # Bitvektor für Gruppenagenten: GruppenagentAktiv[i] = TRUE bedeutet,
  # dass der Gruppenagent i aktiv ist = eine Gruppe leitet
  #-------------------------------------------------------------------------
  ARRAY [AnzGruppenagenten] GruppenagentAktiv (LOGICAL)

CONTINUOUS
  WisKap (REAL)

SENSOR VARIABLES
DISCRETE
  #-----------------------------------------------------------------------------
  # Anzahl der generieren Anfragen / AAO 'Anfrage_verschicken' (von VerhaltenI)
  #-----------------------------------------------------------------------------
  GenerierteAnfragen (INTEGER) := 0,

  #-------------------------------------------------------------------------
  # Information über existierende Gruppen (von WahrnehmungI)
  #-------------------------------------------------------------------------
  ARRAY [AnzGruppenagenten] GruppenagentAktiv_S (LOGICAL),
```

```
  ID (INTEGER)        := 0,
  Protokoll (LOGICAL) := TRUE

CONTINUOUS
  #-----------------------------------------------------
  # gefilterte Informationen von Wahrnehmung/Blackboard
  #-----------------------------------------------------
  QualitaetG (REAL[GQ])

RANDOM VARIABLES
  #-------------------------------------
  # Zufallswert für anfängliches Wissen
  #-------------------------------------
  StartWissen (REAL[WI]): UNIFORM(LowLimit := 0.1 [WI], UpLimit := 0.5 [WI]),

  #-----------------------------------------------------------------
  # Zufallswert für Intelligenz des Individuums (Gaussverteilung)
  #-----------------------------------------------------------------
  ZIntelligenzGauss (REAL[IQ]): GAUSS(LowLimit := 80 [IQ],
                                      UpLimit := 120 [IQ],
                                      Mean := 100 [IQ],
                                      Sigma := 15 [IQ]),

  #-----------------------------------------------------------------
  # Zufallswert für Intelligenz des Individuums (Gleichverteilung)
  #-----------------------------------------------------------------
  ZIntelligenzGleich (REAL[IQ]): UNIFORM(LowLimit := 80 [IQ],
                                         UpLimit := 120 [IQ])

TRANSITION INDICATORS
  #-------------------------------------------------
  # Zusage von Gruppenagent erhalten (an VerhaltenI)
  #-------------------------------------------------
  ZusageErhalten,

  #-------------------------------------------------
  # Absage von Gruppenagent erhalten (an VerhaltenI)
  #-------------------------------------------------
  AbsageErhalten,

  #-------------------------------------------------
  # freier Gruppenangent bestimmt (an VerhaltenI)
  #-------------------------------------------------
  GruppenagentBestimmt,

  #-------------------------------------------------
  # Änderung der Gruppensituation (an StatusI)
  #-------------------------------------------------
  NeueGruppensituation,

  #-------------------------------------------------------------------------
  # Bewerbungsliste bestimmt = Gruppen nach Eignung geordnet (an VerhaltenI)
  #-------------------------------------------------------------------------
  BewerbungslisteBestimmt,
    INTERN1,
    INTERN2

SENSOR INDICATORS
  #-------------------------------------------------------
  # Bestimme ersten freien Gruppenagenten (von Verhalten)
  #-------------------------------------------------------
  BestimmeGruppenagent,

  #-------------------------------------------------------
  # Gruppe wurde verlassen/aufgelöst (von Verhalten)
  #-------------------------------------------------------
  GruppeVerlassen
```

```
LOCATIONS
  #-----------------------------------------------
  # Gruppen nach potentieller Qualitaet geordnet
  #-----------------------------------------------
  Rangfolge (Gruppe ORDERED BY DEC Qualitaet)  := 0 Gruppe,

  #----------------------------
  # Vernichten der Nachrichten
  #----------------------------
  Muelleimer (Nachricht) := 0 Nachricht

SENSOR LOCATIONS
  #-------------------------------------------------------
  # Eingang für Nachrichten von Wahrnehmung/Gruppenagent
  #-------------------------------------------------------
  Eingang (Nachricht) := 0 Nachricht,

  #-------------------------------------------------------------
  # Gruppen nach potentieller Qualitaet geordnet (in Verhalten)
  #-------------------------------------------------------------
  Rangfolge_S (Gruppe)  := 0 Gruppe

DYNAMIC BEHAVIOUR

  #-------------------------------------------------------------------------
  # Wissenserwerb
  #-------------------------------------------------------------------------

  WisKap := (WisAktMax - WisAkt) / WisAktMax;

  DIFFERENTIAL EQUATION
    Wissen' := a * WisKap * WisNormal * (Intelligenz/100) * QualitaetG
               * WisAkt;
    WisAkt' := a * WisKap * WisNormal * (Intelligenz/100) * QualitaetG
               * WisAkt;
  END

  #-----------------------------------------------------------------------------
  # Information aus der Komponente WahrnehmungI; auch für VerhaltenI benötigt
  #-----------------------------------------------------------------------------
  GruppenagentAktiv{i OF 1..AnzGruppenagenten} := GruppenagentAktiv_S[i];


  #-----------------------------------------------------------------------------
  #
  # Ereignis 1 : Zufaellige Initialisierung von Wissen, WisAkt und der
  #              Intelligenz des Individuums
  #
  #-----------------------------------------------------------------------------
  ON START DO
    Wissen^      := StartWissen;
    WisAkt^      := StartWissen;
    #-------------------------------------------------
    # Auswahl der Zufallsverteilung fuer die Intelligenz
    #-------------------------------------------------
    IF AuswahlVerteilung = 0 DO
      Intelligenz^ := ZIntelligenzGauss;
      IF Protokoll DO
        DISPLAY("T = %5.2f, Individuum %2d, Komponente KognitionI, ", T, ID);
        DISPLAY("Ereignis 1\n");
        DISPLAY("Initialisierung Wissen, WisAkt mit %4.2f\n", StartWissen);
        DISPLAY("Initialisierung der Intelligenz mit %4.2f\n",
                ZIntelligenzGauss);
      END
    END
    ELSIF AuswahlVerteilung = 1 DO
      Intelligenz^ := ZIntelligenzGleich;
```

```
    IF Protokoll DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente KognitionI, ", T, ID);
      DISPLAY("Ereignis 1\n");
      DISPLAY("Initialisierung Wissen, WisAkt mit %4.2f\n", StartWissen);
      DISPLAY("Initialisierung der Intelligenz mit %4.2f\n",
               ZIntelligenzGleich);
    END
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 2 : Neue Nachricht(en) eingetroffen
#
#-------------------------------------------------------------------------
WHENEVER NUMBER(Eingang) > 0 DO
  IF Eingang:Nachricht[1].Typ = 'Auskunft' AND NOT AuskuenfteBearbeiten DO
    #---------------------------------------------------------------
    # Es sind Auskünfte von einem oder mehreren Gruppenagenten angekommen
    # ==> Bewerbungsliste muss bestimmt werden
    #---------------------------------------------------------------
    AuskuenfteBearbeiten^ := TRUE;
    IF Protokoll DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente KognitionI, ", T, ID);
      DISPLAY("Ereignis 2\n");
      DISPLAY("Auskunft von Gruppenagent %2d\n", Eingang:Nachricht[1].Sender);
    END
  END
  ELSIF Eingang:Nachricht[1].Typ = 'Zusage' DO
    #-----------------------------------------
    # Das Individuum wird Mitglied einer Gruppe
    #-----------------------------------------
    TEintritt^      := T;
    neueGLernphase^ := TRUE;
    GruppenID^      := Eingang:Nachricht[1].Sender;
    GroesseG^       := Eingang:Nachricht[1].GroesseG;
    SIGNAL NeueGruppensituation;                          # an StatusI
    SIGNAL ZusageErhalten;                                # an VerhaltenI
    AnzahlZusagen^  := AnzahlZusagen + 1;
    Muelleimer^ : FROM Eingang GET Nachricht[1];
    IF Protokoll DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente KognitionI, ", T, ID);
      DISPLAY("Ereignis 2\n");
      DISPLAY("Zusage von Gruppenagent %2d\n", Eingang:Nachricht[1].Sender);
    END
  END
  ELSIF Eingang:Nachricht[1].Typ = 'Absage' DO
    #---------------------------------------------
    # Die Bewerbung des Individuums wird abgelehnt
    #---------------------------------------------
    SIGNAL AbsageErhalten;
    Muelleimer^ : FROM Eingang GET Nachricht[1];
    IF Protokoll DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente KognitionI, ", T, ID);
      DISPLAY("Ereignis 2\n");
      DISPLAY("Absage von Gruppenagent %2d\n", Eingang:Nachricht[1].Sender);
    END
  END
  ELSIF Eingang:Nachricht[1].Typ = 'Ausschluss' DO
    #-------------------------------------------------
    # Das Individuum wird aus der Gruppe ausgeschlossen
    #-------------------------------------------------
    GruppenID^ := 0;
    GroesseG^  := 0;
    SIGNAL NeueGruppensituation;                          # an StatusI
    Muelleimer^ : FROM Eingang GET Nachricht[1];
    IF Protokoll DO
```

```
      DISPLAY("T = %5.2f, Individuum %2d, Komponente KognitionI, ", T, ID);
      DISPLAY("Ereignis 2\n");
      DISPLAY("Ausschluss aus der Gruppe erfolgt !! \n");
    END
  END
  ELSIF Eingang:Nachricht[1].Typ = 'Update' DO
    #-------------------------------------------------------------------------
    # Der Gruppenagent informiert seine Mitglieder über eine Änderung in
    # der Gruppe. Es ist ein Mitglied hinzugekommen oder ausgetreten,
    # dadurch ändern sich Größe und frühester Auflösungszeitpunkt der
    # Gruppe
    #-------------------------------------------------------------------------
    GroesseG^ := Eingang:Nachricht[1].GroesseG;
    Muelleimer^ : FROM Eingang GET Nachricht[1];
    IF Protokoll DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente KognitionI, ", T, ID);
      DISPLAY("Ereignis 2\n");
      DISPLAY("Update von Gruppenagent %d \n",
               Eingang:Nachricht[1].Sender);
    END
  END
END

#-------------------------------------------------------------------------------
#
# Ereignis 3 : Starten einer neuen Einzel-Lernphase
#
#-------------------------------------------------------------------------------
ON ^GruppenID = 0 ^ DO
  WisAkt^ := Grundwissen;
  IF Protokoll DO
    DISPLAY("T = %5.2f, Individuum %2d, Komponente KognitionI, ", T, ID);
    DISPLAY("Ereignis 3\n");
    DISPLAY("Starten einer neuen Einzel-Lernphase\n");
  END
END

#-------------------------------------------------------------------------------
#
# Ereignis 4 : Starten einer neuen Gruppen-Lernphase
#
#-------------------------------------------------------------------------------
ON ^echteGruppe AND neueGLernphase^ DO
  WisAkt^ := Grundwissen;
  neueGLernphase^ := FALSE;
  IF Protokoll DO
    DISPLAY("T = %5.2f, Individuum %2d, Komponente KognitionI, ", T, ID);
    DISPLAY("Ereignis 4\n");
    DISPLAY("Starten einer neuen Gruppen-Lernphase\n");
  END
END

#-------------------------------------------------------------------------------
#
# Ereignis 5 : Freien Gruppenagent suchen (zur Gruppengründung)
#
#-------------------------------------------------------------------------------
ON BestimmeGruppenagent DO
  GruppenID^ := FreierGruppenagent(ARRAY GruppenagentAktiv);
  SIGNAL GruppenagentBestimmt;
  IF Protokoll DO
    DISPLAY("T = %5.2f, Individuum %2d, Komponente KognitionI, ", T, ID);
    DISPLAY("Ereignis 5\n");
    DISPLAY("Auftrag freien Gruppenagent suchen (Gruppengründung)\n");
  END
END
```

```
#-------------------------------------------------------------------------
# Ereignis 6 : Aufstellen einer Bewerbungsliste: Die Liste steht dann im
#              Verhalten zur Verfügung
#              Die Erstellung der Liste kann erst beginnen, wenn die
#              Auskuenfte von allen aktiven Gruppenagenten vorliegen
#              Aus technischen Gründen in 3 Schritten:
#               1. Hilfsarrays aufbauen (Auswerten der Auskünfte)
#               2. Informationen aus den Hilsarrays an mobile Komponeten
#                  übergeben
#               3. Mobile Komponenten an die Location Rangfolge in der
#                  Komponente VerhaltenI übergeben
#-------------------------------------------------------------------------
ON ^AuskuenfteBearbeiten AND
    EingegangeneAuskuenfte(LOCATION Eingang) = GenerierteAnfragen^ DECLARE
  ARRAY [AnzGruppenagenten] HilfeID (INTEGER)  := 0,
  ARRAY [AnzGruppenagenten] HilfeQu (REAL[GQ]) := 0 [GQ],
  AktPosition                       (INTEGER)  := 0
DO PROCEDURE
  #-----------------------------------------------------------------------
  # Die Informationen die in den Nachrichten des Typs 'Anfrage' vom
  # Gruppenagent/den aktiven Gruppenagenten übermittlet werden, müssen
  # für die weitere Bearbeitung extrahiert werden
  #-----------------------------------------------------------------------
  FOR i FROM 1 TO NUMBER(Eingang)
  REPEAT
    IF Eingang:Nachricht[i].Typ = 'Auskunft' DO
      HilfeID[Eingang:Nachricht[i].Sender] := Eingang:Nachricht[i].Sender;
      HilfeQu[Eingang:Nachricht[i].Sender] :=
                                  Eingang:Nachricht[i].QualitaetG;
    END
  END_LOOP
END
DO TRANSITIONS
  Rangfolge^ : ADD AnzGruppenagenten NEW Gruppe;
  RangfolgeID{i OF 1..AnzGruppenagenten}^ := HilfeID[i];
  RangfolgeQu{i OF 1..AnzGruppenagenten}^ := HilfeQu[i];
  #-----------------------------------------------
  # Alle verarbeiteten Anfragen in den Muelleimer
  #-----------------------------------------------
  Muelleimer^ : FROM Eingang GET Nachricht{ALL i |
                    Eingang:Nachricht[i].Typ = 'Auskunft'};
  SIGNAL INTERN1;
  IF Protokoll DO
    DISPLAY("T = %5.2f, Individuum %2d, Komponente KognitionI, ", T, ID);
    DISPLAY("Ereignis 6 (1)\n");
    DISPLAY("Bestimme Bewerbungsliste\n");
  END
END

ON INTERN1 DO
  Rangfolge:Gruppe{i OF 1..AnzGruppenagenten}.ID^        := RangfolgeID[i];
  Rangfolge:Gruppe{j OF 1..AnzGruppenagenten}.Qualitaet^ := RangfolgeQu[j];
  SIGNAL INTERN2;
END

ON INTERN2 DO
  Rangfolge^ : TO Rangfolge_S SEND Gruppe{ALL i |
                                      Rangfolge:Gruppe[i].ID <> 0};
  Rangfolge^ : REMOVE Gruppe{ALL i | Rangfolge:Gruppe[i].ID = 0};
  SIGNAL BewerbungslisteBestimmt;
  AuskuenfteBearbeiten^ := FALSE;
  IF Protokoll DO
    DISPLAY("T = %5.2f, Individuum %2d, Komponente KognitionI, ", T, ID);
    DISPLAY("Ereignis 6 (2)\n");
    DISPLAY("Bewerbungsliste erstellt\n");
  END
END
```

```
  #-------------------------------------------------------------------------
  #
  # Ereignis 7 : Individuum ist Mitglied in einer echten Gruppe geworden /
  #              hat Lernpartner
  #
  #-------------------------------------------------------------------------
  ON ^GroesseG >= 2^ DO
    echteGruppe^ := TRUE;
    IF Protokoll DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente KognitionI, ", T, ID);
      DISPLAY("Ereignis 7\n");
      DISPLAY("Mitglied hat Lernpartner\n");
    END
  END

  #-------------------------------------------------------------------------
  #
  # Ereignis 8 : Individuum ist (wieder) allein / hat keine Lernpartner (mehr)
  #
  #-------------------------------------------------------------------------
  ON ^GroesseG = 0^ OR ^GroesseG = 1^ DO
    echteGruppe^ := FALSE;
    IF GroesseG = 0 DO
      IF Protokoll DO
        DISPLAY("T = %5.2f, Individuum %2d, Komponente KognitionI, ", T, ID);
        DISPLAY("Ereignis 8\n");
        DISPLAY("Individuum ist wieder allein/ kein Gruppenmitglied mehr\n");
      END
    END
    ELSE DO
      TUnechteGruppe^ := T;
      IF Protokoll DO
        DISPLAY("T = %5.2f, Individuum %2d, Komponente KognitionI, ", T, ID);
        DISPLAY("Ereignis 8\n");
        DISPLAY("Individuum ist einziges Mitglied einer (unechten) Gruppe; ");
        DISPLAY("TUnechteGruppe = %f\n", T);
      END
    END
  END

  #-------------------------------------------------------------------------
  #
  # Ereignis 9 : Gruppe verlassen/aufgelöst
  #
  #-------------------------------------------------------------------------
  ON GruppeVerlassen DO
    GruppenID^ := 0;
    GroesseG^  := 0;
    SIGNAL NeueGruppensituation;
  END

  #-------------------------------------------------------------------------
  #
  # Ereignis 10 : Nachrichten vernichten
  #
  #-------------------------------------------------------------------------
  WHENEVER NUMBER(Muelleimer) > 0 DO
    Muelleimer^ : REMOVE Nachricht{ALL};
  END

END OF KognitionI
```

C.2.5 Die Komponente *StatusI*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    StatusI
#
#  Art:     Basiskomponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Status des Individuums:
#       + Gruppenmitglied oder allein
#       + soziale Kompetenz
#       + "Qualitaet" des Individuums
#
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - WahrnehmungI an StatusI
#       + QualitaetG
#
#    - KognitionI an StatusI
#       + GruppenID
#       + GroesseG
#       + echteGruppe
#       + Wissen
#       + NeueGruppensituation
#
#    - Statistik an StatusI
#       + WissenMax
#       + SozKompetenzMax
#
#    - StatusI an VerhaltenI
#       + SozAkt: aktuelle soziale Zufriedenheit
#       + SozBed: soziales Bedürfnis
#       + SozAnlage: soziale Anlage (Geselligkeit)
#       + SozKompetenz: soziale Kompetenz des Individuums (für Bewerbung)
#       + Gruppenmitglied
#
#-------------------------------------------------------------------------------

BASIC COMPONENT StatusI

USE OF UNITS
  UNIT [SoZ] = BASIS
  UNIT [GQ]  = BASIS
  UNIT [SK]   = BASIS
  UNIT [WI]   = BASIS
  UNIT [SQ]   = BASIS
  UNIT [IndQ] = BASIS
  TIMEUNIT    = [h]


DECLARATION OF ELEMENTS

CONSTANTS
  #--------------------------------------
  # Obergrenze für soziale Zufriedenheit
  #--------------------------------------
  SozAktMax (REAL[SoZ]) := 1 [SoZ],
```

```
  #-------------------------------------------------------------
  # Parameter für normale Ab-/Zunahme der sozialen Zufriedenheit
  #-------------------------------------------------------------
  SozNormal (REAL[SoZ/h]) := 0.5 [SoZ/h],

  b (REAL[1/(SoZ*SQ)])    := 1.0 [1/(SoZ*SQ)],

  #-----------------------------------------------------------
  # Parameter fuer normale Entwicklung der sozialen Kompetenz
  #-----------------------------------------------------------
  KompNormal (REAL[SK/h]) := 0.4 [SK/h],
  d          (REAL[1/SQ]) := 1.0 [1/SQ]

STATE VARIABLES
DISCRETE
  #-------------------------------------------------------------------------
  # Soziale Anlage des Individuums; beeinflußt Geschwindigkeit der Zu-
  # und Abnahme der sozialen Zufriedenheit
  #-------------------------------------------------------------------------
  SozAnlage (REAL[SQ]) := 0 [SQ],


  #-----------------------------------------------------------
  # Festlegung der Zufallsverteilung fuer die soziale Anlage
  #-----------------------------------------------------------
  AuswahlVerteilung (INTEGER) := 0,

  #-------------------------------------------------------------------
  # Das Individuum ist Mitglied einer Gruppe (es kann auch das einzige
  # Gruppenmitglied sein = unechte Gruppe)
  #-------------------------------------------------------------------
  Gruppenmitglied (LOGICAL) := FALSE,

  #-----------------------------------------------------------------------------
  # Gewichtungsfaktor für den Einfluss des Wissens auf die Qualitaet/ soziale
  # Position des Individuums
  #-----------------------------------------------------------------------------
  i1 (REAL) := 1,

  #-----------------------------------------------------------------------------
  # Gewichtungsfaktor für den Einfluss der sozialen Kompetenz auf die
  # Qualitaet/ soziale Position des Individuums
  #-----------------------------------------------------------------------------
  i2 (REAL) := 1

CONTINUOUS
  #-----------------------------------
  # Soziale Zufriedenheit
  #-----------------------------------
  SozAkt (REAL[SoZ])    := 0 [SoZ],

  #-----------------------------------
  # Soziale Kompetenz des Individuums
  #-----------------------------------
  SozKompetenz (REAL[SK]) := 0 [SK],

  #-----------------------------------------------------
  # Zeit, die das Individuum in einer echten Gruppe war
  #-----------------------------------------------------
  TInGruppe (REAL[h]) := 0 [h]

DEPENDENT VARIABLES
DISCRETE
  #-----------------------------------
  # Normierung der Gewichtungsfaktoren
  #-----------------------------------
  i (REAL[IndQ]) := 0 [IndQ]
```

```
CONTINUOUS
  #-------------------------------------------------------------------------
  # Einflussgroesse zur Beschränkung des Wachstums der sozialen Zufriedenheit
  #-------------------------------------------------------------------------
  SozKap (REAL),

  #-----------------------------------------------------
  # Sozialbeduerfnis (Beduerfnis nach sozialen Kontakten)
  #-----------------------------------------------------
  SozBed (REAL[SoZ]),

  #-------------------------------------------------------------------------
  # Qualitaet des Individuums: bestimmt in Relation zu den anderen Individuen
  # soziale Position des Individuums innerhalb des Gesamtsystems
  #-------------------------------------------------------------------------
  QualitaetI (REAL[IndQ]) := 0 [IndQ]

SENSOR VARIABLES
DISCRETE


  #----------------------------
  # GruppenID (von KognitionI)
  #----------------------------
  GruppenID (INTEGER),

  #-------------------------------------
  # Gruppengröße (von Kognition)
  #-------------------------------------
  GroesseG  (INTEGER),

  #-----------------------------
  # echteGruppe (von KognitionI)
  #-----------------------------
  echteGruppe (LOGICAL),

  ID        (INTEGER) := 0,
  Protokoll (LOGICAL) := TRUE

CONTINUOUS
  #-------------------------------------------------
  # Informationen von WahrnehmungI/Blackboard lesen
  #-------------------------------------------------
  QualitaetG (REAL[GQ]),

  #-------------------------
  # Wissen (von KognitionI)
  #-------------------------
  Wissen (REAL[WI]),

  #-----------------------------------------
  # Statistische Informationen (von Statistik)
  #-----------------------------------------
  WissenMax (REAL[WI]),
  SozKompetenzMax (REAL[SK])

RANDOM VARIABLES
  #-----------------------------------------------------
  # Zufallswert für anfängliche soziale Zufriedenheit
  #-----------------------------------------------------
  StartSozAkt (REAL[SoZ]):UNIFORM(LowLimit := 0.1 [SoZ], UpLimit := 0.9 [SoZ]),

  #-------------------------------------------------
  # Zufallswert für anfängliche Sozialkompetenz
  #-------------------------------------------------
  StartSozKompetenz (REAL[SK]): UNIFORM(LowLimit := 0.1 [SK],
                                        UpLimit  := 0.3 [SK]),
```

```
  #-------------------------------------------------------------------
  # Zufallswert für soziale Anlage des Individuums (Gaussverteilung)
  #-------------------------------------------------------------------
  ZSozAnlageGauss (REAL[SQ]): GAUSS(LowLimit := 80 [SQ],
                                    UpLimit := 120 [SQ],
                                    Mean := 100 [SQ],
                                    Sigma := 15 [SQ]),

  #-------------------------------------------------------------------
  # Zufallswert für soziale Anlage des Individuums (Gleichverteilung)
  #-------------------------------------------------------------------
  ZSozAnlageGleich (REAL[SQ]): UNIFORM(LowLimit := 80 [SQ],
                                       UpLimit := 120 [SQ])

SENSOR INDICATORS
  #-------------------------------------------------
  # Änderung der Gruppensituation (von KognitionI)
  #-------------------------------------------------
  NeueGruppensituation

DYNAMIC BEHAVIOUR

  #-------------------------------------------------------------------------------
  # Entwicklung der sozialen Kompetenz: Wenn das Individuum Mitglied einer
  # echten Gruppe ist, sammelt es durch die Interaktion mit den anderen
  # Gruppenmitglieder soziale Erfahrung und gewinnt so an sozialer Kompetenz
  #-------------------------------------------------------------------------------
  IF echteGruppe DO
    DIFFERENTIAL EQUATION

      # Erhoehung der sozialen Kompetenz
      SozKompetenz' := d * KompNormal * SozAnlage/100;

      # Erhoehung der Zeit, die das Individuum in einer echten Gruppe war
      TInGruppe' := 1;
    END
  END
  ELSE DO
    DIFFERENTIAL EQUATION
      SozKompetenz' := 0 [SK/h];
      TInGruppe' := 0;
    END
  END

  #-----------------------------------------------------------
  # Entwicklung der sozialen Zufriedenheit
  #-----------------------------------------------------------

  SozKap := (SozAktMax - SozAkt) / SozAktMax;

  IF GroesseG >= 2 DO
    #-----------------------------------------------------
    # Das Individuum ist Mitglied in einer echten Gruppe
    #-----------------------------------------------------
    DIFFERENTIAL EQUATION
      SozAkt' := b * SozKap * SozNormal * (SozAnlage/100)
                 * QualitaetG * 1 [1/GQ] * SozAkt;
    END
  END
  ELSE DO
    #-----------------------------------------------------
    # Das Individuum ist allein
    #-----------------------------------------------------
    DIFFERENTIAL EQUATION
      SozAkt' := -b * SozKap * SozNormal * (SozAnlage/100) * SozAkt;
    END
  END
```

```
#-------------------------------------
# Bestimmung des Sozialbeduerfnisses
#-------------------------------------
SozBed := SozAktMax - SozAkt;

#-----------------------------------------
# Bestimmung der Qualitaet des Individuums
#-----------------------------------------
i := 1 [IndQ] / (i1  +  i2 );

QualitaetI := i * (  i1 * (Wissen / WissenMax)
                   + i2 * (SozKompetenz / SozKompetenzMax));

#-------------------------------------------------------------------------
#
# Ereignis 1 : Zufällige Initialisierung der sozialen Zufriedenheit, der
#              sozialen Kompetenz und der sozialen Anlage
#
#-------------------------------------------------------------------------
ON START DO
  #---------------------------------------------------------------
  # Zufällige Initialisierung von sozialer Kompetenz und Zufriedenheit
  # (für einen sauberen Simulationsstart)
  # Es würde reichen SozKompetenz mit > 0 zu initialisieren; durch
  # den Parameter UpLimit kann man einstellen, wieviel Sozialkompetenz
  # die Individuen schon vor Beginn dieses Experiments gesammelt haben
  # (im Laufe ihres bisherigen Lebens)
  #---------------------------------------------------------------
  SozAkt^       := StartSozAkt;
  SozKompetenz^ := StartSozKompetenz;

  #-------------------------------------------------
  # Auswahl der Zufallsverteilung fuer die soziale Anlage
  #-------------------------------------------------
  IF AuswahlVerteilung = 0 DO
    SozAnlage^ := ZSozAnlageGauss;
    IF Protokoll DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente StatusI, ", T, ID);
      DISPLAY("Ereignis 1\n");
      DISPLAY("Initialisierung soziale Zufriedenheit mit %4.2f\n",
               StartSozAkt);
      DISPLAY("Initialisierung soziale Anlage mit %4.2f\n",ZSozAnlageGauss);
    END
  END
  ELSIF AuswahlVerteilung = 1 DO
    SozAnlage^ := ZSozAnlageGleich;
    IF Protokoll DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente StatusI, ", T, ID);
      DISPLAY("Ereignis 1\n");
      DISPLAY("Initialisierung soziale Zufriedenheit mit %4.2f\n",
               StartSozAkt);
      DISPLAY("Initialisierung soziale Anlage mit %4.2f\n",ZSozAnlageGleich);
    END
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 2 : Änderung der Gruppensituation des Individuums
#
#-------------------------------------------------------------------------
ON NeueGruppensituation DO
  IF GruppenID <> 0 DO
    #----------------------------
    # Zusage erhalten = Aufnahme
    #----------------------------
    Gruppenmitglied^ := TRUE;
```

```
      IF Protokoll DO
        DISPLAY("T = %5.2f, Individuum %2d, Komponente StatusI, ", T, ID);
        DISPLAY("Ereignis 3\n");
        DISPLAY("Individuum ist Gruppenmitglied geworden \n");
      END
    END
    ELSE DO
      #-----------------------------------------
      # Ausschluss oder Gruppe verlassen/aufgelöst
      #-----------------------------------------
      Gruppenmitglied^ := FALSE;
      IF Protokoll DO
        DISPLAY("T = %5.2f, Individuum %2d, Komponente StatusI,  ", T, ID);
        DISPLAY("Ereignis 2\n");
        DISPLAY("Individuum ist wieder allein\n");
      END
    END
  END

  #-------------------------------------------------------------------------
  #
  # Ereignis 3 : Da die Entwicklung der sozialen Zufriedenheit logistisch
  #              modelliert wurde, muss sichergestellt werden, das die Variable
  #              SozAkt nicht zu nahe bei einem ihrere Extremwerte ist, wenn
  #              sich die Gruppensituation ändert.
  #              Wird das Individuum erst wieder Mitglied einer Gruppe, wenn
  #              SozAkt sehr nahe bei Null ist, würde die Zunahme anfangs
  #              extrem langsam sein; ein völlig zufriedenes Individuum
  #              (SozAkt = 1) koennte nie wieder unglücklicher werden.
  #              Diese zwei (extrem seltenen) Sonderfaelle werden durch dieses
  #              Ereignis abgesichert, indem ggf. SozAkt auf einem Wert in dem
  #              gewuenschten Bereich (0.05 .. 0.95) gesetzt wird.
  #
  #-------------------------------------------------------------------------
  ON ^GroesseG >= 2^ OR ^GroesseG < 2^ DO
    IF GroesseG >= 2 DO
      SozAkt^ := MAX (0.01 [SoZ], SozAkt);
    END
    ELSE DO
      SozAkt^ := MIN (0.99 [SoZ], SozAkt);
    END
    IF Protokoll AND (SozAkt < 0.01 [SoZ] OR SozAkt > 0.99 [SoZ]) DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente StatusI, ", T, ID);
      DISPLAY("Ereignis 3\n");
      DISPLAY("Sonderfall: SozAkt von %3.2f in Bereich 0.01 .. 0.99 [SoZ]",
               SozAkt);
      DISPLAY(" geholt\n");
    END
  END

END OF StatusI
```

C.2.6 Die Komponente *EmotionI*

```
#-------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    EmotionI
#
#  Art:     Basiskomponente
#
#  Version: 0
#
```

```
#  Beschreibung:
#
#    - Die Komponente ist vorlaeufig leer!
#
#-------------------------------------------------------------------------------

BASIC COMPONENT EmotionI
USE OF UNITS
  TIMEUNIT   = [h]

DECLARATION OF ELEMENTS

SENSOR VARIABLES
DISCRETE

  ID        (INTEGER) := 0,
  Protokoll (LOGICAL) := TRUE
END OF EmotionI
```

C.2.7 Die Komponente *VerhaltenI*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    VerhaltenI
#
#  Art:     Basiskomponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Beschreibung/ Steuerung des Verhalten des Individuums (Regelbasis)
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - KognitionI an VerhaltenI
#       + Wissen
#       + WisAkt
#       + GroesseG
#       + QualitaetG
#       + TUnechteGruppe
#       + TEintritt
#       + GruppenagentAktiv []
#       + Rangfolge(Gruppe)
#       + ZusageErhalten
#       + AbsageErhalten
#       + GruppenagentBestimmt
#       + BewerbungslisteBestimmt
#
#    - StatusI an VerhaltenI
#       + SozKompetenz
#       + SozAkt
#       + SozBed
#       + SozAnlage
#       + Gruppenmitglied
#
#    - Verhalten an Kognition
#       + GenerierteAnfragen
#       + BestimmeGruppenagent
#       + GruppeVerlassen
#
#-------------------------------------------------------------------------------
```

```
BASIC COMPONENT VerhaltenI
MOBILE SUBCOMPONENTS OF CLASS Gruppe,
                             AusfuehrungsAOI

USE OF UNITS
  UNIT [WI]  = BASIS
  UNIT [SoZ] = BASIS
  UNIT [SK]  = BASIS
  UNIT [SQ]  = BASIS
  TIMEUNIT   = [h]

LOCAL DEFINITIONS

DIMENSIONS
  #-----------------------------------------------
  #  Obergrenze für die Anzahl der Gruppenagenten
  #-----------------------------------------------
  AnzGruppenagenten := 36

VALUE SET
  #-------------------------------------------------------------------------
  # Ausführungsanordnungs-Typen:
  # - Anfrage_verschicken:   Mitteilung an Gruppenagenten schicken (Anfrage)
  # - Bewerbung_verschicken: Mitteilung an Gruppenagent schicken (Bewerbung)
  # - Gruppe_verlassen:      Mitteilung an Gruppenagent schicken (Ausstieg)
  #-------------------------------------------------------------------------
  AAOTyp : ('Anfrage_verschicken', 'Bewerbung_verschicken', 'Gruppe_verlassen')

  #-------------------------------------------------------------------------
  # keineGruppe()
  #
  # Typ:             Funktion
  # Aufrufparameter: ARRAY GruppenagentAktiv
  # Rückgabewert:    LOGICAL
  #
  # Beschreibung: Diese Funktion liefert den Wert TRUE zurück, falls es noch
  #               keine Gruppe und damit auch keinen aktiven Gruppenagenten
  #               gibt
  #-------------------------------------------------------------------------
  FUNCTION keineGruppe (ARRAY [n] LOGICAL: GruppenagentAktiv
                                --> LOGICAL)
    DECLARE keineGruppen (LOGICAL) := TRUE
  BEGIN
    FOR i FROM 1 TO AnzGruppenagenten
    REPEAT
      IF GruppenagentAktiv[i] DO
        keineGruppen := FALSE;
      END
    END_LOOP
    RETURN (keineGruppen);
  END_FUNC

DECLARATION OF ELEMENTS
CONSTANTS
  #---------------------------------------------------------------
  # Zeit, die das Individuum eine unechte Gruppe aufrecht erhaelt
  #---------------------------------------------------------------
  TGruppe (REAL[h]) := 2 [h],

  #-------------------------------------------------------
  # Schwellwert fuer das Beenden einer Einzel-Lernphase
  #-------------------------------------------------------
  LimitAllein (REAL) := 1.5,

  #-------------------------------------------------------
  # Schwellwert fuer das Beenden einer Gruppen-Lernphase
  #-------------------------------------------------------
```

```
  LimitGruppe (REAL) := 1.5

STATE VARIABLES
DISCRETE
  #-------------------------------------------------------------
  # Falls Bewerben = TRUE kann eine Bewerbung verschickt werden
  #-------------------------------------------------------------
  Bewerben  (LOGICAL) := FALSE,

  #-------------------------------------------------------------
  # Falls Anfragen = TRUE können Ausfuehrungsanordnungen vom Typ
  # 'Anfrage_verschicken' erzeugt werden
  #-------------------------------------------------------------
  Anfragen (LOGICAL) := FALSE,

  #-------------------------------------------------------------
  # ID des Gruppenagent, an den gerade eine Anfrage erzeugt wird
  #-------------------------------------------------------------
  AktuellerGruppenagent (INTEGER) := 0,

  #-------------------------------------------------------------
  # Anzahl der generieren Anfragen / AAO 'Anfrage_verschicken'
  #-------------------------------------------------------------
  GenerierteAnfragen (INTEGER) := 0,

  #-------------------------------------------------------------------
  # Gewichtungsfaktor fuer den Einfluss des kognitiven Aspekts auf die
  # Entscheidungen bzgl. Wechsel der Lernphase
  #-------------------------------------------------------------------
  c1 (REAL) := 2,

  #-------------------------------------------------------------------
  # Gewichtungsfaktor fuer den Einfluss des sozialen Aspekts auf die
  # Entscheidungen bzgl. Wechsel der Lernphase
  #-------------------------------------------------------------------
  c2 (REAL) := 1

DEPENDENT VARIABLES
DISCRETE
  #-------------------------------------------------------------------------
  # Einfluss des Fortschritts in der Einzel-Lernphase auf die Entschei-
  # dung, das Alleinsein beenden zu wollen
  #-------------------------------------------------------------------------
  KogAllein (REAL[1/WI]) := 0 [1/WI],

  #-------------------------------------------------------------------------
  # Einfluss des Sozialbedürfnisses auf die Entscheidung, das Alleinsein
  # beenden zu wollen
  #-------------------------------------------------------------------------
  SozAllein (REAL[1/SoZ]) := 0 [1/SoZ],

  #-------------------------------------------------------------------------
  # Einfluss des Fortschritts in der Gruppen-Lernphase auf die Entschei-
  # dung, wieder allein lernen zu wollen
  #-------------------------------------------------------------------------
  KogGruppe (REAL[1/WI]) := 0 [1/WI],

  #-------------------------------------------------------------------------
  # Einfluss der sozialen Zufriedenheit auf die Entscheidung, die Gruppe
  # verlassen zu wollen
  #-------------------------------------------------------------------------
  SozGruppe (REAL[1/SoZ]) := 0 [1/SoZ],

  #-----------------------------------
  # Normierung der Gewichtungsfaktoren
  #-----------------------------------
  cAllein (REAL) := 0,
```

```
  cGruppe (REAL) := 0

SENSOR VARIABLES
DISCRETE
  #------------------------------
  # Informationen von KognitionI
  #------------------------------
  GroesseG       (INTEGER),
  TUnechteGruppe (REAL[h]),
  TEintritt      (REAL[h]),
  ARRAY [AnzGruppenagenten] GruppenagentAktiv (LOGICAL),

  #-------------------------------------
  # ID des ersten freien Gruppenagenten
  #-------------------------------------
  GruppenID (INTEGER),

  #------------------------------
  # Informationen von StatusI
  #------------------------------
  Gruppenmitglied (LOGICAL),
  SozAnlage (REAL[SQ]),

  ID        (INTEGER) := 0,
  Protokoll (LOGICAL) := TRUE

CONTINUOUS
  #------------------------------
  # Informationen von KognitionI
  #------------------------------
  Wissen    (REAL[WI]),
  WisAkt (REAL[WI]),

  #------------------------------
  # Informationen von StatusI
  #------------------------------
  SozAkt (REAL[SoZ]),
  SozBed (REAL[SoZ]),
  SozKompetenz (REAL[SK])

TRANSITION INDICATORS
  #-------------------------------------------------------
  # Bestimme ersten freien Gruppenagenten (an KognitionI)
  #-------------------------------------------------------
  BestimmeGruppenagent,

  #-------------------------------------------------------
  # Gruppe wurde verlassen/aufgelöst (an KognitionI)
  #-------------------------------------------------------
  GruppeVerlassen

SENSOR INDICATORS
  #-----------------------------------------------
  # Zusage von Gruppenagent erhalten (von KognitionI)
  #-----------------------------------------------
  ZusageErhalten,

  #-----------------------------------------------
  # Absage von Gruppenagent erhalten (von KognitionI)
  #-----------------------------------------------
  AbsageErhalten,

  #-----------------------------------------------
  # freier Gruppenangent bestimmt (von KognitionI)
  #-----------------------------------------------
  GruppenagentBestimmt,
```

```
  #-------------------------------------------------------------------------
  # Bewerbungsliste bestimmt = Gruppen nach Eignung geordnet (von KognitionI)
  #-------------------------------------------------------------------------
  BewerbungslisteBestimmt

LOCATIONS
  #-------------------------------------------------------------------------
  # Gruppen nach potentieller Qualitaet geordnet (wird in KognitionI gefüllt)
  #-------------------------------------------------------------------------
  Rangfolge (Gruppe)  := 0 Gruppe,

  #---------------------------------------------
  # Generieren/Ablage von Ausführungsanordnungen
  #---------------------------------------------
  Ausgang (AusfuehrungsAOI) := 0 AusfuehrungsAOI


DYNAMIC BEHAVIOUR

  #-------------------------------------------------------------------------
  # Einfluss des Fortschritts in der Einzel-Lernphase auf die Entschei-
  # dung, das Alleinsein beenden zu wollen
  #-------------------------------------------------------------------------
  KogAllein := c1 * 1 [1/WI];

  #-------------------------------------------------------------------------
  # Einfluss des Sozialbedürfnisses auf die Entscheidung, das Alleinsein
  # beenden zu wollen
  #-------------------------------------------------------------------------
  SozAllein := c2 * 1 [1/SoZ];

  #-------------------------------------------------------------------------
  # Einfluss des Fortschritts in der Gruppen-Lernphase auf die Entschei-
  # dung, wieder allein lernen zu wollen
  #-------------------------------------------------------------------------
  KogGruppe := 1 / c1 * 1 [1/WI];

  #-------------------------------------------------------------------------
  # Einfluss der sozialen Zufriedenheit auf die Entscheidung, die Gruppe
  # verlassen zu wollen
  #-------------------------------------------------------------------------
  SozGruppe := 1 / c2 * 1 [1/SoZ];

  #--------------------------------------
  # Normierung der Gewichtungsfaktoren
  #--------------------------------------
  cAllein := 2 / (c1     + c2);
  cGruppe := 2 / (1 / c1 + 1 / c2);

  #-------------------------------------------------------------------------------
  #
  # Ereignis 1 : Das Individuum will Mitglied in einer Gruppe werden, da es
  #              sich ziemlich einsam fühlt und alleine nicht mehr viel lernen
  #              kann.
  #
  #-------------------------------------------------------------------------------
  ON ^cAllein * (KogAllein * WisAkt + SozAllein * SozBed) > LimitAllein
      AND NOT Gruppenmitglied^ DO
    IF keineGruppe(ARRAY GruppenagentAktiv) DO
      #-----------------------------------------------------------------
      # Es existiert noch keine Gruppe, daher keine Anfragen möglich
      # ==> Sofort Bewerbungsvorgang (impliziert Gruppengründung) starten
      #-----------------------------------------------------------------
      Bewerben^ := TRUE;
      IF Protokoll DO
        DISPLAY("T = %5.2f, Individuum %2d, Komponente VerhaltenI, ", T, ID);
        DISPLAY("Ereignis 1\n");
```

```
      DISPLAY("Bewerben / !! ERSTE !! Gruppe gründen!\n");
    END
  END
  ELSE DO
    #--------------------------------------
    # Mindestens ein Gruppenagent ist aktiv
    #--------------------------------------
    Anfragen^              := TRUE;
    AktuellerGruppenagent^ := 1;
    GenerierteAnfragen^    := 0;
    IF Protokoll DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente VerhaltenI, ", T, ID);
      DISPLAY("Ereignis 1\n");
      DISPLAY("Individuum will Mitglied einer Gruppe werden\n");
    END
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 2 : Erzeugen der Ausführungsanordnungen 'Anfrage_verschicken'
#
#              Um zu entscheiden, welcher Gruppe es sich anschließen möchte,
#              benötigt das Individuum Informationen über die vorhandenen
#              Gruppen. Dazu schickt es an jeden aktiven Gruppenagenten
#              eine Anfrage. Um die Auswertung der Auskünfte der Gruppen-
#              agenten in der Kognition zu erleichtern, sollen alle Auskünfte
#              zum gleichen Takt vorliegen. Deshalb müssen die Anfragen auch
#              alle Gruppenagenten zum gleichen Takt erreichen.
#
#-------------------------------------------------------------------------
WHENEVER Anfragen AND AktuellerGruppenagent <= AnzGruppenagenten DO
  IF GruppenagentAktiv[AktuellerGruppenagent] DO
    #-------------------
    # Anfragen erzeugen
    #-------------------
    Ausgang^ : ADD 1 NEW AusfuehrungsAOI
                 CHANGING
                   Typ^ := 'Anfrage_verschicken';
                   ID^  := AktuellerGruppenagent;
                 END
    #------------------
    # Anfragen zaehlen
    #------------------
    GenerierteAnfragen^ := GenerierteAnfragen + 1;
  END
  AktuellerGruppenagent^ := AktuellerGruppenagent + 1;

  #-----------------------------------------------------------------------
  # Falls alle Ausführungsanordnungen 'Anfrage_verschicken' erzeugt wurden
  #-----------------------------------------------------------------------
  IF AktuellerGruppenagent = AnzGruppenagenten DO
    Anfragen^ := FALSE;
    IF Protokoll DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente VerhaltenI, ", T, ID);
      DISPLAY("Ereignis 2\n");
      DISPLAY("Alle AAO Anfragen erzeugt\n");
    END
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 3 : In der Kognition wurde die Bewerbungsliste erstellt. In der
#              Location Rangfolge stehen jetzt alle existierenden Gruppen
#              sortiert nach dem potentiellen Nutzen für das Individuum.
#              Das Individuum kann sich jetzt bewerben.
```

```
#
#-------------------------------------------------------------------------
ON BewerbungslisteBestimmt DO
  Bewerben^ := TRUE;
  IF Protokoll DO
    DISPLAY("T = %5.2f, Individuum %2d, Komponente VerhaltenI, ", T, ID);
    DISPLAY("Ereignis 3\n");
    DISPLAY("Bewerbungsliste bestimmt\n");
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 4 : Das Individuum bewirbt sich bei der vorteilhaftesten Gruppe
#
#-------------------------------------------------------------------------
WHENEVER Bewerben AND NUMBER(Rangfolge) > 0 DO
  Ausgang^ : ADD 1 NEW AusfuehrungsAOI
               CHANGING
                 Typ^          := 'Bewerbung_verschicken';
                 ID^           := Rangfolge:Gruppe[1].ID;
                 Wissen^       := Wissen;
                 SozKompetenz^ := SozKompetenz;
               END
  #-------------------------------------------------------------
  # Bewerbung läuft, vorläufig keine neue Bewerbung verschicken
  #-------------------------------------------------------------
  Bewerben^ := FALSE;
  IF Protokoll DO
    DISPLAY("T = %5.2f, Individuum %2d, Komponente VerhaltenI, ", T, ID);
    DISPLAY("Ereignis 4\n");
    DISPLAY("Bewerben bei Gruppe %2d \n", Rangfolge:Gruppe[1].ID);
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 5 : Die Bewerbung des Individuum wurde angenommen, es wird
#              Mitglied in dieser Gruppe
#
#-------------------------------------------------------------------------
ON ZusageErhalten DO
  #-------------------------
  # Bewerbungsliste löschen
  #-------------------------
  Rangfolge^ : REMOVE Gruppe{ALL};
  IF Protokoll DO
    DISPLAY("T = %5.2f, Individuum %2d, Komponente VerhaltenI, ", T, ID);
    DISPLAY("Ereignis 5\n");
    IF NUMBER(Rangfolge) > 0 DO
      DISPLAY("Zusage von Gruppe %2d (1)\n", Rangfolge:Gruppe[1].ID);
    END
    ELSE DO
      DISPLAY("Zusage von Gruppe %2d (2)\n", GruppenID);
    END
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 6 : Die Bewerbung des Individuum wurde abgelehnt, es muss sich bei
#              der naechstgeignetsten Gruppe bewerben
#
#-------------------------------------------------------------------------
ON AbsageErhalten DO
```

```
  #--------------
  # Neu bewerben
  #--------------
  Bewerben^ := TRUE;
  IF NUMBER(Rangfolge) = 0 DO
    # Das Individuum wurde von einem zum Bewerbungszeitpunkt inaktiven
    # Gruppenagent abgelehnt (nur wenn sich mehrere bewerben: sehr selten!)
    IF Protokoll DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente VerhaltenI, ", T, ID);
      DISPLAY("Ereignis 6\n");
      DISPLAY("Absage von Gruppe %2d \n", GruppenID);
    END
  END
  ELSE DO
    # Standard: Absage von einem aktiven Gruppenagenten
    #----------------------------------
    # Gruppe aus Bewerbungsliste löschen
    #----------------------------------
    Rangfolge^ : REMOVE Gruppe[1];
    IF Protokoll DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente VerhaltenI, ", T, ID);
      DISPLAY("Ereignis 6\n");
      DISPLAY("Absage von Gruppe %2d \n", Rangfolge:Gruppe[1].ID);
    END
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 7 : Das Individuum wurde von keiner Gruppe aufgenommen oder es
#              existiert keine Gruppe
#
#-------------------------------------------------------------------------
ON ^Bewerben AND NUMBER(Rangfolge) = 0^ DO
  Bewerben^ := FALSE;
  #-----------------------------------------------------------------
  # Der erste freie Gruppenagent muss bestimmt werden (Kognition), um
  # ihn mit der Leitung und Verwaltung der Gruppen beauftragen zu können
  #-----------------------------------------------------------------
  SIGNAL BestimmeGruppenagent;
  IF Protokoll DO
    DISPLAY("T = %5.2f, Individuum %2d, Komponente VerhaltenI, ", T, ID);
    DISPLAY("Ereignis 7\n");
    DISPLAY("Auftrag freien Gruppenagent bestimmen (an KognitionI)\n");
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 8 : Das Individuum gründet eine neue Gruppe
#              Durch die Bewerbung bei einem freien Gruppenagent wird dieser
#              aktiviert und übernimmt die Leitung der neuen Gruppe
#
#-------------------------------------------------------------------------
ON GruppenagentBestimmt DO
  Ausgang^ : ADD 1 NEW AusfuehrungsAOI
               CHANGING
                 Typ^          := 'Bewerbung_verschicken';
                 ID^           := GruppenID;
                 Wissen^       := Wissen;
                 SozKompetenz^ := SozKompetenz;
               END
  IF Protokoll DO
    DISPLAY("T = %5.2f, Individuum %2d, Komponente VerhaltenI, ", T, ID);
    DISPLAY("Ereignis 8\n");
    DISPLAY("AAO Bewerbung_verschicken/Gruppe %2d gründen!\n", GruppenID);
  END
```

```
  END


  #-------------------------------------------------------------------------
  #
  # Ereignis 9 : Das Individuum verläßt die Gruppe, da es genug Sozialkontakt
  #              gehabt hat und in der Gruppe nicht mehr viel lernen kann oder
  #              weil sich eine TGruppe Zeiteinheiten kein anderes Individuum
  #              der unechten Gruppe angeschlossen hat.
  #
  #-------------------------------------------------------------------------
  ON ^cGruppe * (KogGruppe * WisAkt + SozGruppe * SozAkt) > LimitGruppe
      AND Gruppenmitglied^ OR
     ^T > TUnechteGruppe + TGruppe AND T > TEintritt + TGruppe AND
      GroesseG = 1^ DO
    Ausgang^ : ADD 1 NEW AusfuehrungsAOI
                  CHANGING
                    Typ^ := 'Gruppe_verlassen';
                    ID^  := GruppenID;
                  END
    SIGNAL GruppeVerlassen;         # an KognitionI
    IF Protokoll DO
      IF GroesseG <> 1 DO
        DISPLAY("T = %5.2f, Individuum %2d, Komponente VerhaltenI, ", T, ID);
        DISPLAY("Ereignis 9\n");
        DISPLAY("AAO Gruppe_verlassen \n");
      END
      ELSE DO
        DISPLAY("T = %5.2f, Individuum %2d, Komponente VerhaltenI, ", T, ID);
        DISPLAY("Ereignis 9\n");
        DISPLAY("AAO Gruppe_verlassen/Auflösen der Gruppe %2d !\n", GruppenID);
      END
    END
  END

END OF VerhaltenI
```

C.2.8 Die Komponente *AktorI*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    AktorI
#
#  Art:     Basiskomponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Umsetzen von Ausführungsanordnungen in Aktionen
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - VerhaltenI an AktorI
#        + AusfuehrungsAOI
#
#-------------------------------------------------------------------------------

BASIC COMPONENT AktorI

MOBILE SUBCOMPONENTS OF CLASS Nachricht,
                              AusfuehrungsAOI
```

```
USE OF UNITS
  TIMEUNIT = [h]
LOCAL DEFINITIONS

DIMENSIONS
  #------------------------
  # Anzahl der Individuen
  #------------------------
  AnzGruppenagenten := 36

VALUE SET
  #-------------------------------------------------------------------------
  # Nachrichtentypen:
  #  - Individuum an Gruppenagent
  #     + Anfrage:       Individuum fragt Gruppenagent nach Gruppeninformation
  #     + Bewerbung:     Individuum bewirbt sich um Aufnahme in die Gruppe
  #     + Ausstieg:      Individuum verlaesst die Gruppe
  #
  # - Gruppenagent an Individuum
  #     + Auskunft:      Gruppenagent antwortet auf Anfrage des Individuums
  #     + Absage:        Individuum wird nicht aufgenommen
  #     + Zusage:        Individuum wird in die Gruppe aufgenommen
  #     + Ausschluss:    Grupppenagent schliesst Individuum aus der Gruppe aus
  #     + Update:        Aenderung der diskreten, für die Gruppenmitglieder
  #                      relevanten Informationen (GroesseG)
  #-------------------------------------------------------------------------
  Nachrichtentyp : ('Anfrage', 'Bewerbung', 'Ausstieg',
                    'Auskunft', 'Absage', 'Zusage', 'Ausschluss', 'Update')

VALUE SET
  #-------------------------------------------------------------------------
  # Ausführungsanordnungs-Typen:
  # - Anfrage_verschicken:   Mitteilung an Gruppenagenten schicken (Anfrage)
  # - Bewerbung_verschicken: Mitteilung an Gruppenagent schicken (Bewerbung)
  # - Gruppe_verlassen:      Mitteilung an Gruppenagent schicken (Ausstieg)
  #-------------------------------------------------------------------------
  AAOTyp : ('Anfrage_verschicken', 'Bewerbung_verschicken', 'Gruppe_verlassen')

DECLARATION OF ELEMENTS

STATE VARIABLES
DISCRETE
  #---------------------------------------------------------------------
  # Wenn Anfragen = TRUE, dann ist/sind Ausführungsanordnungen des Typs
  # 'Anfrage_verschicken' eingetroffen, die gesammelt werden müssen
  #---------------------------------------------------------------------
  Anfragen (LOGICAL) := FALSE,

  #---------------------------------------------------------------------
  # Die Variable ANfragenzaehler sorgt dafür, daß mit dem Verschicken
  # der Nachrichten 'Anfrage' solange gewartet wird, bis alle
  # Ausführungsanordnungen des Typs 'Anfrage_verschicken' eingetroffen
  # sind und die entsprechende Nachricht generiert wurde
  #---------------------------------------------------------------------
  Anfragenzaehler (INTEGER) := 0

SENSOR VARIABLES
DISCRETE
  ID        (INTEGER) := 0,
  Protokoll (LOGICAL) := TRUE

LOCATIONS
  #--------------------------------------
  # Vernichten der Ausführungsanordnungen
  #--------------------------------------
  Muelleimer (AusfuehrungsAOI) := 0 AusfuehrungsAOI,
```

```
  #-------------------------------------------
  # Generieren einer Nachricht an Gruppenagenten
  #-------------------------------------------
  NeueNachricht (Nachricht) := 0 Nachricht,

  #-------------------------------------------------------------------------
  # Generieren von Nachrichten, die gleichzeitig verschickt werden sollen
  #-------------------------------------------------------------------------
  NeueNachrichten (Nachricht) := 0 Nachricht

SENSOR LOCATIONS
  #-------------------------------------
  # Empfangen von Ausführungsanordnungen
  #-------------------------------------
  Eingang (AusfuehrungsAOI) := 0 AusfuehrungsAOI,

  #-----------------------------------------------------------------
  # Eingangsbereich für Nachrichten dieses Individuums im Connector
  #-----------------------------------------------------------------
  Ausgang (Nachricht) := 0 Nachricht

DYNAMIC BEHAVIOUR

  #-----------------------------------------------------------------------------
  #
  # Ereignis 1 : Eine Ausführungsanordnung trifft ein
  #
  #-----------------------------------------------------------------------------
  WHENEVER NUMBER(Eingang) > 0 DO
    IF Eingang:AusfuehrungsAOI[1].Typ = 'Anfrage_verschicken' DO
      #-------------------------------------------------------------------
      # Nachricht 'Anfrage' generieren
      # Im Gegensatz zu den anderen Ausführungsanordnungstypen müssen in
      # diesem Fall mehrere Nachrichten zum selben Takt verschickt werden.
      # Deshalb werden die sequentiell erzeugten Nachrichten 'Anfrage'
      # zwischengespeichert und dann gleichzeitig weitergeleitet.
      #-------------------------------------------------------------------
      NeueNachrichten^ : ADD 1 NEW Nachricht
                           CHANGING
                             Sender^  := ID;
                             Empfaenger^ := Eingang:AusfuehrungsAOI[1].ID;
                             Typ^        := 'Anfrage';
                           END
      Anfragen^ := TRUE;
      #-------------------------------------------
      # Ausfuehrungsanordnung in den Mülleimer legen
      #-------------------------------------------
      Muelleimer^ : FROM Eingang GET AusfuehrungsAOI[1];
      IF Protokoll DO
        DISPLAY("T = %5.2f, Individuum %2d, Komponente AktorI, ", T, ID);
        DISPLAY("Ereignis 1\n");
        DISPLAY("Nachricht 'Anfrage' an Gruppenagent %d generiert\n",
                Eingang:AusfuehrungsAOI[1].ID);
      END
    END
    ELSIF Eingang:AusfuehrungsAOI[1].Typ = 'Bewerbung_verschicken' DO
      #-------------------------------------------------------------------
      # Nachricht 'Bewerbung' generieren
      #-------------------------------------------------------------------
      NeueNachricht^ : ADD 1 NEW Nachricht
                         CHANGING
                           Sender^       := ID;
                           Empfaenger^   := Eingang:AusfuehrungsAOI[1].ID;
                           Typ^          := 'Bewerbung';
                           Wissen^       := Eingang:AusfuehrungsAOI[1].Wissen;
                           SozKompetenz^ := Eingang:
                                            AusfuehrungsAOI[1].SozKompetenz;
```

```
                    END
    #-------------------------------------------
    # Ausfuehrungsanordnung in den Mülleimer legen
    #-------------------------------------------
    Muelleimer^ : FROM Eingang GET AusfuehrungsAOI[1];
    IF Protokoll DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente AktorI, ", T, ID);
      DISPLAY("Ereignis 1\n");
      DISPLAY("Nachricht 'Bewerbung' an Gruppenagent %d generiert\n",
              Eingang:AusfuehrungsAOI[1].ID);
    END
  END
  ELSIF Eingang:AusfuehrungsAOI[1].Typ = 'Gruppe_verlassen' DO
    #-------------------------------------------------------------------------
    # Nachricht 'Ausstieg' generieren
    #-------------------------------------------------------------------------
    NeueNachricht^ : ADD 1 NEW Nachricht
                     CHANGING
                       Sender^     := ID;
                       Empfaenger^ := Eingang:AusfuehrungsAOI[1].ID;
                       Typ^        := 'Ausstieg';
                     END
    #-------------------------------------------
    # Ausfuehrungsanordnung in den Mülleimer legen
    #-------------------------------------------
    Muelleimer^ : FROM Eingang GET AusfuehrungsAOI[1];
    IF Protokoll DO
      DISPLAY("T = %5.2f, Individuum %2d, Komponente AktorI, ", T, ID);
      DISPLAY("Ereignis 1\n");
      DISPLAY("Nachricht 'Ausstieg' generiert\n");
    END
  END
END

#-----------------------------------------------------------------------------
#
# Ereignis 2 : Weiterleiten der generierten Nachricht an den Connector
#
#-----------------------------------------------------------------------------
WHENEVER NUMBER(NeueNachricht) > 0 DO
  NeueNachricht^ : TO Ausgang SEND Nachricht[1];
END

#-----------------------------------------------------------------------------
#
# Ereignis 3 : "Mitzählen" ob noch Ausführungsanordnungen des Typs
#              'Anfrage_verschicken' eintreffen können
#
#-----------------------------------------------------------------------------
WHENEVER Anfragen AND Anfragenzaehler <= AnzGruppenagenten DO
  Anfragenzaehler^ := Anfragenzaehler + 1;
END

#-----------------------------------------------------------------------------
#
# Ereignis 4 : Wenn alle Nachrichten des Typs 'Anfrage' generiert sind,
#              werden sie gleichzeitig weitergeleitet.
#
#-----------------------------------------------------------------------------
ON ^Anfragen AND Anfragenzaehler = AnzGruppenagenten + 1^ DO
  NeueNachrichten^ : TO Ausgang SEND Nachricht{ALL};
  Anfragen^        := FALSE;
  Anfragenzaehler^ := 1;
END
```

```
  #-------------------------------------------------------------------------------
  #
  # Ereignis 5 : Vernichten der abgearbeiteten Ausführungsanordnungen
  #
  #-------------------------------------------------------------------------------
  WHENEVER NUMBER(Muelleimer) > 0 DO
    Muelleimer^ : REMOVE AusfuehrungsAOI{ALL};
  END

END OF AktorI
```

C.3 Die Gruppenagenten

C.3.1 Die Strukturkomponente *Gruppenagent*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    Gruppenagent
#
#  Art:     High-Level Komponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Die Komponente ist eine Klassenschablone für Grupppenagenten
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Statistik an KognitionG
#       + DurchschnWissen:       Durchschnitt des Wissens aller Individuen
#       + DurchschnSozKompetenz: Durchschnitt der sozialen Zufriedenheit aller
#                                Individuen
#       + RelWisGMax
#       + RelSozGMax
#
#    - KognitionG an Statistik
#       + GruppenagentAktiv
#       + RelWisG
#       + RelSozG
#       + QualitaetG
#       + GroesseG
#
#    - übergeordnete Komponente(n):
#       + Lerngruppe
#
#    - untegeordnete Komponente(n):
#       + SensorG
#       + WahrnehmungG
#       + KognitionG
#       + EmotionG
#       + StatusG
#       + VerhaltenG
#       + AktorG
#
#-------------------------------------------------------------------------------

HIGH LEVEL COMPONENT Gruppenagent

SUBCOMPONENTS
```

```
  SensorG,
  WahrnehmungG,
  KognitionG,
  EmotionG,
  StatusG,
  VerhaltenG,
  AktorG

INPUT CONNECTIONS
  ID        --> ( SensorG.ID,
                  WahrnehmungG.ID,
                  KognitionG.ID,
                  StatusG.ID,
                  EmotionG.ID,
                  VerhaltenG.ID,
                  AktorG.ID );

  Protokoll --> ( SensorG.Protokoll,
                  WahrnehmungG.Protokoll,
                  KognitionG.Protokoll,
                  StatusG.Protokoll,
                  EmotionG.Protokoll,
                  VerhaltenG.Protokoll,
                  AktorG.Protokoll );

  #---------------------------------------
  # Nachrichten-Schnittstellen (Connector)
  #---------------------------------------
  Eingang --> SensorG.Eingang;
  Ausgang --> AktorG.Ausgang;

  #-----------------------------------------------------------
  # Informationen über Individuen vom Blackboard (Connector)
  #-----------------------------------------------------------
  Wissen       --> SensorG.Wissen_S;
  SozKompetenz --> SensorG.SozKompetenz_S;

  #-------------------------------------------------------------
  # Informationen über Individuen aus der Komponente Statistik
  #-------------------------------------------------------------
  DurchschnWissen       --> KognitionG.DurchschnWissen;
  DurchschnSozKompetenz --> KognitionG.DurchschnSozKompetenz;

  #-----------------------------------------------------------
  # Informationen über Gruppen aus der Komponetne Statistik
  #-----------------------------------------------------------
  RelWisGMax --> KognitionG.RelWisGMax;
  RelSozGMax --> KognitionG.RelSozGMax;

OUTPUT EQUIVALENCES
  #---------------------------------------
  # Informationen über den Gruppenagenten
  #---------------------------------------
  GruppenagentAktiv := StatusG.GruppenagentAktiv;
  GroesseG          := KognitionG.GroesseG;
  RelWisG           := KognitionG.RelWisG;
  RelSozG           := KognitionG.RelSozG;
  QualitaetG        := KognitionG.QualitaetG;

COMPONENT CONNECTIONS
  #-------------------------------------------------------------------------------
  # SensorG an WahrnehmungG
  #-------------------------------------------------------------------------------

  #-------------
  # Nachrichten
  #-------------
```

```
SensorG.Ausgang -->  WahrnehmungG.Eingang;

#-------------------------
# Blackboard-Informatioen
#-------------------------
SensorG.Wissen{ALL i}       --> WahrnehmungG.Wissen_S[i];
SensorG.SozKompetenz{ALL i} --> WahrnehmungG.SozKompetenz_S[i];

#-----------------------------------------------------------------------------
# WahrnehmungG an KognitionG
#-----------------------------------------------------------------------------


#-------------
# Nachrichten
#-------------
WahrnehmungG.Ausgang --> KognitionG.Eingang;

#-------------------------
# Blackboard-Informatioen
#-------------------------
WahrnehmungG.Wissen{ALL i}       --> KognitionG.Wissen[i];
WahrnehmungG.SozKompetenz{ALL i} --> KognitionG.SozKompetenz[i];

#-----------------------------------------------------------------------------
# KognitionG an StatusG
#-----------------------------------------------------------------------------

#--------------
# Gruppengröße
#--------------
KognitionG.GroesseG --> StatusG.GroesseG;

#-----------------------------------------------------------------------------
# KognitionG an VerhaltenG
#-----------------------------------------------------------------------------

#-----------------------------------------
# Informationen für Ausführungsanordnungen
#-----------------------------------------
KognitionG.GroesseG        --> VerhaltenG.GroesseG;
KognitionG.Mitglied{ALL i} --> VerhaltenG.Mitglied[i];
KognitionG.BewerberID      --> VerhaltenG.BewerberID;

#----------------------
# Qualitaet der Gruppe
#----------------------
KognitionG.QualitaetG --> VerhaltenG.QualitaetG;

#-------------------------------------------------------------------
# potentielle Qualitaet der Gruppe nach Änderung der Gruppengröße
#-------------------------------------------------------------------
KognitionG.QualitaetGNeu --> VerhaltenG.QualitaetGNeu;

#--------------------------------------
# Kandidat für Ausschluß aus der Gruppe
#--------------------------------------
KognitionG.Ausschlusskandidat --> VerhaltenG.Ausschlusskandidat;

#---------------------------------
# Anfrage von Individuum erhalten
#---------------------------------
KognitionG.AnfrageErhalten --> VerhaltenG.AnfrageErhalten;

#----------------------------------
# Bewerbung von Individuum erhalten
#----------------------------------
```

```
  KognitionG.BewerbungErhalten --> VerhaltenG.BewerbungErhalten;

  #----------------------------------
  # Ausschlußkandidat bestimmt
  #----------------------------------
  KognitionG.AusschlusskandidatBestimmt
             --> VerhaltenG.AusschlusskandidatBestimmt;

  #-------------------------------------------------------------------------
  # Neue Gruppensituation durch Aufnahme eines Bewerbers, Ausschluß
  # eines Mitglieds oder Ausstieg eines Mitglieds
  #-------------------------------------------------------------------------
  KognitionG.NeueGruppensituation --> VerhaltenG.NeueGruppensituation;

  #-------------------------------------------------------------------------------
  # VerhaltenG an KognitionG
  #-------------------------------------------------------------------------------

  #-----------------------------------------------
  # Die Beurteilung des Bewerbers ist abgeschlossen
  #-----------------------------------------------
  VerhaltenG.BewerberBeurteilt --> KognitionG.BewerberBeurteilt;

  #--------------------
  # Bewerber aufnehmen
  #--------------------
  VerhaltenG.BewerberAufnehmen --> KognitionG.BewerberAufnehmen;

  #-----------------------------
  # Ausschlußkandidat bestimmen
  #-----------------------------
  VerhaltenG.BestimmeAusschlusskandidat
             --> KognitionG.BestimmeAusschlusskandidat;

  #------------------------
  # Mitglied ausschließen
  #------------------------
  VerhaltenG.MitgliedAusschliessen --> KognitionG.MitgliedAusschliessen;

  #-------------------------------------------------------------------------------
  # VerhaltenG an AktorG
  #-------------------------------------------------------------------------------

  #-------------------------
  # Ausführungsanordnungen
  #-------------------------
  VerhaltenG.Ausgang --> AktorG.Eingang;

END OF Gruppenagent
```

C.3.2 Die Komponente *SensorG*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    SensorG
#
#  Art:     Basiskomponente
#
#  Version: 0
#
#  Beschreibung:
#
```

```
#    - Schnittstelle zwischen Gruppenagent und Umwelt:
#       + Aufnehmen von Nachrichten
#       + Informationen vom Blackboard
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Connector an SensorG
#       + Nachricht
#       + Wissen[]
#       + SozKompetenz[]
#
#    - SensorG an WahrnehmungG
#       + Nachricht
#       + Wissen[]
#       + SozKompetenz[]
#
#-------------------------------------------------------------------------------

BASIC COMPONENT SensorG

MOBILE SUBCOMPONENTS OF CLASS Nachricht

USE OF UNITS
  UNIT [WI]  = BASIS
  UNIT [SK]  = BASIS
  TIMEUNIT   = [h]

LOCAL DEFINITIONS

DIMENSIONS
  #---------------------------------------------
  # Anzahl der Individuen
  #---------------------------------------------
  AnzIndividuen := 36

VALUE SET
  #-------------------------------------------------------------------------------
  # Nachrichtentypen:
  #  - Individuum an Gruppenagent
  #     + Anfrage:       Individuum fragt Gruppenagent nach Gruppeninformation
  #     + Bewerbung:     Individuum bewirbt sich um Aufnahme in die Gruppe
  #     + Ausstieg:      Individuum verlaesst die Gruppe
  #
  # - Gruppenagent an Individuum
  #     + Auskunft:      Gruppenagent antwortet auf Anfrage des Individuums
  #     + Absage:        Individuum wird nicht aufgenommen
  #     + Zusage:        Individuum wird in die Gruppe aufgenommen
  #     + Ausschluss:    Grupppenagent schliesst Individuum aus der Gruppe aus
  #     + Update:        Aenderung der diskreten, für die Gruppenmitglieder
  #                      relevanten Informationen (GroesseG)
  #-------------------------------------------------------------------------------
  Nachrichtentyp : ('Anfrage', 'Bewerbung', 'Ausstieg',
                    'Auskunft', 'Absage', 'Zusage', 'Ausschluss', 'Update')

DECLARATION OF ELEMENTS

DEPENDENT VARIABLES
CONTINUOUS
  #-------------------------------------
  # Blackboard
  #-------------------------------------
  ARRAY [AnzIndividuen] Wissen       (REAL[WI])  := 0 [WI],
  ARRAY [AnzIndividuen] SozKompetenz (REAL[SK])  := 0 [SK]

SENSOR VARIABLES
DISCRETE
  ID              (INTEGER) := 0,
```

```
  Protokoll       (LOGICAL) := TRUE

CONTINUOUS
  #-----------------------------------
  # Informationen vom Blackboard lesen
  #-----------------------------------
  ARRAY [AnzIndividuen] Wissen_S (REAL[WI]),
  ARRAY [AnzIndividuen] SozKompetenz_S (REAL[SK])

LOCATIONS
  #-------------------------------------------
  # Weitergabe von Nachrichten an Wahrnehmung
  #-------------------------------------------
  Ausgang (Nachricht) := 0 Nachricht

SENSOR LOCATIONS
  #-------------------------------------------
  # Eingang für Nachrichten von Individuen
  #-------------------------------------------
  Eingang (Nachricht) := 0 Nachricht

DYNAMIC BEHAVIOUR

  #-------------------------------------------------
  # Bereitstellung der Informationen vom Blackboard
  #-------------------------------------------------
  Wissen       {i OF 1..AnzIndividuen} := Wissen_S[i];
  SozKompetenz {i OF 1..AnzIndividuen} := SozKompetenz_S[i];

  #-------------------------------------------------------------------------------
  #
  # Ereignis 1 : Weiterleitung der eingehenden Nachrichten
  #
  #-------------------------------------------------------------------------------
  WHENEVER NUMBER(Eingang) > 0 DO
    Ausgang^ : FROM Eingang GET Nachricht[1];
      IF Protokoll DO
      DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente SensorG, ", T, ID);
      DISPLAY("Ereignis 1\n");
      DISPLAY("Gruppenagent hat neue Nachricht empfangen\n");
    END
  END

END OF SensorG
```

C.3.3 Die Komponente *WahrnehmungG*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    WahrnehmungG
#
#  Art:     Basiskomponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Filterung der Informationen vom Blackboard
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - SensorG an WahnehmungG
```

```
#       + Nachricht
#       + Wissen[]
#       + SozKompetenz[]
#
#    - WahrnehmungG an KognitionG
#       + Nachricht
#       + Wissen der Gruppenmitglieder
#       + SozKompetenz der Gruppenmitglieder
#
#-------------------------------------------------------------------------------

BASIC COMPONENT WahrnehmungG

MOBILE SUBCOMPONENTS OF CLASS Nachricht

USE OF UNITS
  UNIT [WI]  = BASIS
  UNIT [SK] = BASIS
  TIMEUNIT   = [h]

LOCAL DEFINITIONS

DIMENSIONS
  #---------------------------------------------
  # Anzahl der Individuen
  #---------------------------------------------
  AnzIndividuen := 36

VALUE SET
  #-------------------------------------------------------------------------------
  # Nachrichtentypen:
  #  - Individuum an Gruppenagent
  #     + Anfrage:       Individuum fragt Gruppenagent nach Gruppeninformation
  #     + Bewerbung:     Individuum bewirbt sich um Aufnahme in die Gruppe
  #     + Ausstieg:      Individuum verlaesst die Gruppe
  #
  # - Gruppenagent an Individuum
  #     + Auskunft:      Gruppenagent antwortet auf Anfrage des Individuums
  #     + Absage:        Individuum wird nicht aufgenommen
  #     + Zusage:        Individuum wird in die Gruppe aufgenommen
  #     + Ausschluss:    Grupppenagent schliesst Individuum aus der Gruppe aus
  #     + Update:        Aenderung der diskreten, für die Gruppenmitglieder
  #                      relevanten Informationen (GroesseG)
  #-------------------------------------------------------------------------------
  Nachrichtentyp : ('Anfrage', 'Bewerbung', 'Ausstieg',
                    'Auskunft', 'Absage', 'Zusage', 'Ausschluss', 'Update')

DECLARATION OF ELEMENTS

DEPENDENT VARIABLES
CONTINUOUS
  #-------------------------------------
  # gefilterte Blackboard-Informationen
  #-------------------------------------
  ARRAY [AnzIndividuen] Wissen       (REAL[WI]) := 0 [WI],
  ARRAY [AnzIndividuen] SozKompetenz (REAL[SK]) := 0 [SK]

SENSOR VARIABLES
DISCRETE
  ID              (INTEGER) := 0,
  Protokoll       (LOGICAL) := TRUE

CONTINUOUS
  #-----------------------------------------
  # Informationen vom SensorG/Blackboard lesen
  #-----------------------------------------
  ARRAY [AnzIndividuen] Wissen_S       (REAL[WI]),
```

```
  ARRAY [AnzIndividuen] SozKompetenz_S (REAL[SK])

LOCATIONS
  #-------------------------------------------
  # Weitergabe von Nachrichten an KognitionG
  #-------------------------------------------
  Ausgang (Nachricht) := 0 Nachricht

SENSOR LOCATIONS
  #-------------------------------------------
  # Eingang für Nachrichten von Individuen
  #-------------------------------------------
  Eingang (Nachricht) := 0 Nachricht

DYNAMIC BEHAVIOUR

  #---------------------------------------------------
  # Bereitstellen/Filtern der Blackboard-Informationen
  #---------------------------------------------------
  Wissen       {i OF 1..AnzIndividuen} := Wissen_S[i];
  SozKompetenz {i OF 1..AnzIndividuen} := SozKompetenz_S[i];

  #-------------------------------------------------------------------------------
  #
  # Ereignis 1 : Weiterleitung der eingehenden Nachrichten
  #
  #-------------------------------------------------------------------------------
  WHENEVER NUMBER(Eingang) > 0 DO
    Ausgang^ : FROM Eingang GET Nachricht[1];
  END

END OF WahrnehmungG
```

C.3.4 Die Komponente *KognitionG*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    KognitionG
#
#  Art:     Basiskomponente
#
#  Version: 0
#  Beschreibung:
#
#    - Verarbeiten der Informationen der Wahrnehmung/Blackboard
#    - Berechnen der Größen
#       + RelWisG, RelWisGNorm
#       + RelSozG, RelSozGNorm
#       + QualitaetG
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - WahrnehmungG an KognitionG
#       + Nachricht
#       + Wissen[]
#       + SozKompetenz[]
#
#    - VerhaltenG an KognitionG
#       + BewerberBeurteilt
#       + BewerberAufnehmen
#       + AusschlusskandidatBestimmen
#       + MitgliedAusschliessen
```

```
#
#    - Statistik an KognitionG
#       + DurchschnWissen
#       + DurchschnSozKompetenz
#       + RelWisGMax
#       + RelSozGMax
#
#    - KognitionG an StatusG
#       + GroesseG
#
#    - KognitionG an VerhaltenG
#       + SummeWissenG
#       + SummeSozAktG
#       + GroesseG
#       + BewerberID
#       + QualitaetG
#       + QualitaetGNeu
#       + Ausschlusskandidat
#       + AnfrageErhalten
#       + BewerbungErhalten
#       + AusschlusskandidatBestimmt
#       + NeueGruppensituation
#
#-------------------------------------------------------------------------------

BASIC COMPONENT KognitionG

MOBILE SUBCOMPONENTS OF CLASS Nachricht

USE OF UNITS
  UNIT [WI] = BASIS
  UNIT [SK] = BASIS
  UNIT [IQ] = BASIS
  UNIT [GQ] = BASIS
  TIMEUNIT  = [h]

LOCAL DEFINITIONS

DIMENSIONS
  #---------------------------------------------
  # Anzahl der Individuen
  #---------------------------------------------
  AnzIndividuen := 36

VALUE SET
  #-------------------------------------------------------------------------------
  # Nachrichtentypen:
  #  - Individuum an Gruppenagent
  #     + Anfrage:       Individuum fragt Gruppenagent nach Gruppeninformation
  #     + Bewerbung:     Individuum bewirbt sich um Aufnahme in die Gruppe
  #     + Ausstieg:      Individuum verlaesst die Gruppe
  #
  # - Gruppenagent an Individuum
  #     + Auskunft:      Gruppenagent antwortet auf Anfrage des Individuums
  #     + Absage:        Individuum wird nicht aufgenommen
  #     + Zusage:        Individuum wird in die Gruppe aufgenommen
  #     + Ausschluss:    Grupppenagent schliesst Individuum aus der Gruppe aus
  #     + Update:        Aenderung der diskreten, für die Gruppenmitglieder
  #                      relevanten Informationen (GroesseG)
  #-------------------------------------------------------------------------------
  Nachrichtentyp : ('Anfrage', 'Bewerbung', 'Ausstieg',
                    'Auskunft', 'Absage', 'Zusage', 'Ausschluss', 'Update')

  #-------------------------------------------------------------------------------
  # QualGroesse()
  #
  # Typ:             Tabellenfunktion
```

```
# Aufrufparameter: GroesseG (INTEGER)
# Rückgabewert:    REAL
#
# Beschreibung: Diese Funktion liefert zu einer gegebenen Gruppengroesse
#               den Einfluß dieser Groesse auf die Qualität der Gruppe
#-------------------------------------------------------------------------
TABULAR FUNCTION QualGroesse (REAL --> REAL) CONTINUOUS
  BY LINEAR INTERPOLATION ON
  ( 0,   1,   2,    3,  4,  5,    6  ) -->
  ( 0, 0.45, 0.7, 0.85, 1, 0.9, 0.65 )

#-------------------------------------------------------------------------
# BestimmeGroesseG()
#
# Typ:             Funktion
# Aufrufparameter: ARRAY LOGICAL Mitglied
# Rückgabewert:    INTEGER
#
# Beschreibung: Diese Funktion bestimmt die Anzahl der Gruppenmitglieder
#-------------------------------------------------------------------------
FUNCTION BestimmeGroesseG (ARRAY [n] LOGICAL: Mitglied --> INTEGER)
DECLARE
  GroesseG (INTEGER)  := 0
BEGIN
  FOR i FROM 1 TO AnzIndividuen
  REPEAT
    IF Mitglied[i] DO
      GroesseG := GroesseG + 1;
    END
  END_LOOP
  RETURN(GroesseG);
END_FUNC

#-------------------------------------------------------------------------
# WissenGruppe()
#
# Typ:             Funktion
# Aufrufparameter: ARRAY Wissen, ARRAY Mitglied
# Rückgabewert:    REAL[WI]
#
# Beschreibung: Diese Funktion liefert die Summe des Wissens der
#               Gruppenmitglieder zurück
#-------------------------------------------------------------------------
FUNCTION WissenGruppe (ARRAY [n] REAL[WI]: Wissen,
                       ARRAY [n] LOGICAL:  Mitglied --> REAL[WI])
DECLARE
  SummeWissen (REAL[WI])  := 0 [WI]
BEGIN
  FOR i FROM 1 TO AnzIndividuen
  REPEAT
    IF Mitglied[i] DO
      SummeWissen   := SummeWissen + Wissen[i];
    END
  END_LOOP
  RETURN (SummeWissen);
END_FUNC

#-------------------------------------------------------------------------
# SozKompetenzGruppe()
#
# Typ:             Funktion
# Aufrufparameter: ARRAY SozKompetenz, ARRAY Mitglied
# Rückgabewert:    REAL[SK]
#
# Beschreibung: Diese Funktion liefert die Summe der sozialen Kompetenzen
#               der Gruppenmitglieder zurück
#-------------------------------------------------------------------------
```

```
  FUNCTION SozKompetenzGruppe (ARRAY [n] REAL[SK]: SozKompetenz,
                               ARRAY [n] LOGICAL:   Mitglied --> REAL[SK])
  DECLARE
    SummeSozKompetenz (REAL[SK])  := 0 [SK]
  BEGIN
    FOR i FROM 1 TO AnzIndividuen
    REPEAT
      IF Mitglied[i] DO
        SummeSozKompetenz := SummeSozKompetenz + SozKompetenz[i];
      END
    END_LOOP
    RETURN (SummeSozKompetenz);
  END_FUNC

  #-------------------------------------------------------------------------
  # BestimmeQualitaetG()
  #
  # Typ:              Funktion
  # Aufrufparameter: REAL RelWisGN,
  #                   REAL RelSozGN,
  #                   REAL QualGroesse
  #                   REAL e, f1, f2, f3
  # Rückgabewert:     REAL[GQ]
  #
  # Beschreibung: Diese Funktion bestimmt die Qualitaet der Gruppe
  #-------------------------------------------------------------------------
  FUNCTION BestimmeQualitaetG (REAL: RelWisGN,
                               REAL: RelSozGN,
                               REAL: QualGroesseG,
                               REAL[GQ]: e, REAL: f1, REAL: f2, REAL: f3
                               --> REAL[GQ])
  DECLARE
    f (REAL[GQ])          := 0 [GQ],
    Qualitaet(REAL[GQ])   := 0 [GQ]
  BEGIN
    f := 2 [GQ] / (f1 + f2 + f3);
    Qualitaet := e + f * (f1 * RelWisGN + f2 * RelSozGN + f3 * QualGroesseG);
    RETURN (Qualitaet);
  END_FUNC


DECLARATION OF ELEMENTS

CONSTANTS
  #-------------------------
  # Maximale Gruppengroesse
  #-------------------------
  GroesseGMax (INTEGER) := 6

STATE VARIABLES
DISCRETE
  #-----------------------------
  # Basisqualitaet einer Gruppe
  #-----------------------------
  e (REAL[GQ]) := 1.0 [GQ],

  #---------------------------------------------------------------------
  # Gewicht des Wissens der Gruppe fuer die spezifische Gruppenqualitaet
  #---------------------------------------------------------------------
  f1 (REAL) := 2.0,

  #---------------------------------------------------------------------
  # Gewicht der sozialen Kompetenz der Gruppe fuer die spezifische
  # Gruppenqualitaet
  #---------------------------------------------------------------------
  f2 (REAL) := 1.0,
```

```
  #-------------------------------------------------------------------
  # Gewicht der Groesse der Gruppe auf die spezifische Gruppenqualitaet
  #-------------------------------------------------------------------
  f3 (REAL) := 1.0,

  #-------------------------------------------------------------------------------
  # Bitvektor für Gruppenmitglieder: Mitglied[i] = TRUE bedeutet, daß das
  # Individuum i Mitglied in der von diesem Gruppenagenten verwalteten Gruppe
  # ist
  #-------------------------------------------------------------------------------
  ARRAY [AnzIndividuen] Mitglied (LOGICAL) := FALSE,

  #-----------------------------------------------------------------
  # potentielle Qualitaet der Gruppe nach Änderung der Gruppengröße
  #-----------------------------------------------------------------
  QualitaetGNeu (REAL[GQ]) := 0 [GQ],

  #--------------------------------------------------
  # ID des Bewerbers oder des anfragenden Individuums
  #--------------------------------------------------
  BewerberID (INTEGER) := 0,

  #---------------------------------
  # Bewerber wird gerade beurteilt
  #---------------------------------
  BewerberBeurteilen (LOGICAL) := FALSE,

  #---------------------------------------
  # Kandidat für Ausschluss aus der Gruppe
  #---------------------------------------
  Ausschlusskandidat (INTEGER) := 0,

  #-------------------------------------
  # Anzahl der bisherigen Ausschluesse
  #-------------------------------------
  AnzahlAusschluesse (INTEGER) := 0

DEPENDENT VARIABLES
DISCRETE

  #--------------
  # Gruppengröße
  #--------------
  GroesseG (INTEGER) := 0

CONTINUOUS

  #----------------------------------------
  # Summe des Wissens der Gruppenmitglieder
  #----------------------------------------
  SummeWissenG (REAL[WI]) := 0 [WI],

  #-----------------------------------------------------
  # Summe der sozialen Kompetenz der Gruppenmitglieder
  #-----------------------------------------------------
  SummeSozKompetenzG (REAL[SK]) := 0 [SK],

  #-----------------------------
  # Relatives Wissen der Gruppe
  #-----------------------------
  RelWisG (REAL) := 0 ,

  #---------------------------------------
  # Normiertes relatives Wissen der Gruppe
  #---------------------------------------
  RelWisGN (REAL) := 0 ,
```

```
  #--------------------------------------
  # Relative soziale Kompetenz der Gruppe
  #--------------------------------------
  RelSozG (REAL) := 0,

  #-----------------------------------------------
  # Normierte relative soziale Kompetenz der Gruppe
  #-----------------------------------------------
  RelSozGN (REAL) := 0,

  #---------------------
  # Qualität der Gruppe
  #---------------------
  QualitaetG (REAL[GQ]) := 0 [GQ]

SENSOR VARIABLES
DISCRETE
  ID (INTEGER)        := 0,
  Protokoll (LOGICAL) := TRUE

CONTINUOUS
  #-------------------------------------------------------------
  # gefilterte Informationen von Wahrnehmung/Blackboard lesen
  #-------------------------------------------------------------
  ARRAY [AnzIndividuen] Wissen       (REAL[WI]),
  ARRAY [AnzIndividuen] SozKompetenz (REAL[SK]),

  #-------------------------------------------
  # Informationen aus der Komponente Statistik
  #-------------------------------------------
  DurchschnWissen       (REAL[WI]),
  DurchschnSozKompetenz (REAL[SK]),
  RelWisGMax            (REAL),
  RelSozGMax            (REAL)

TRANSITION INDICATORS
  #---------------------------------
  # Anfrage von Individuum erhalten
  #---------------------------------
  AnfrageErhalten,

  #---------------------------------
  # Bewerbung von Individuum erhalten
  #---------------------------------
  BewerbungErhalten,

  #---------------------------------
  # Ausschlußkandidat bestimmt
  #---------------------------------
  AusschlusskandidatBestimmt,

  #-------------------------------------------------------------------------
  # Neue Gruppensituation durch Aufnahme eines Bewerbers, Ausschluss
  # eines Mitglieds oder freiwilligen Ausstieg eines Mitglieds
  #-------------------------------------------------------------------------
  NeueGruppensituation

SENSOR INDICATORS
  #-------------------------------------
  # Bewerber beurteilt (von VerhaltenG)
  #-------------------------------------
  BewerberBeurteilt,

  #----------------------------------------
  # Bewerber aufnehmen (von VerhaltenG)
  #----------------------------------------
  BewerberAufnehmen,
```

```
  #-----------------------------------------------
  # Ausschlußkandidat bestimmen (von VerhaltenG)
  #-----------------------------------------------
  BestimmeAusschlusskandidat,

  #---------------------------------------
  # Mitglied ausschließen (von VerhaltenG)
  #---------------------------------------
  MitgliedAusschliessen

LOCATIONS
  Muelleimer (Nachricht) := 0 Nachricht

SENSOR LOCATIONS
  #-------------------------------------------------------
  # Eingang für Nachrichten von WahrnehmungG/Individuum
  #-------------------------------------------------------
  Eingang (Nachricht) := 0 Nachricht

DYNAMIC BEHAVIOUR
  #-----------------------------------
  # Bestimmung der abhängigen Variablen
  #-----------------------------------
  SummeWissenG       := WissenGruppe(ARRAY Wissen, ARRAY Mitglied);
  SummeSozKompetenzG := SozKompetenzGruppe(ARRAY SozKompetenz, ARRAY Mitglied);
  RelWisG            := (SummeWissenG / GroesseG) / DurchschnWissen;
  RelWisGN           := RelWisG / RelWisGMax;
  RelSozG            := (SummeSozKompetenzG / GroesseG) / DurchschnSozKompetenz;
  RelSozGN           := RelSozG / RelSozGMax;
  GroesseG           := BestimmeGroesseG(ARRAY Mitglied);
  IF GroesseG > 1 DO
    QualitaetG := BestimmeQualitaetG(RelWisGN, RelSozGN,
                                     QualGroesse(GroesseG), e, f1, f2, f3);
  END
  ELSIF GroesseG = 1 DO
    QualitaetG := 1 [GQ];
  END
  ELSE DO
    QualitaetG := 0 [GQ];
  END

  #-------------------------------------------------------------------------
  #
  # Ereignis 1 : Neue Nachricht eingetroffen
  #
  #-------------------------------------------------------------------------
  WHENEVER NUMBER(Eingang) > 0 AND NOT BewerberBeurteilen DO
    IF Eingang:Nachricht[1].Typ = 'Anfrage' DO
      #-----------------------------------------------------------------------
      # Individuum will Informationen über die Gruppe
      #-----------------------------------------------------------------------
      BewerberID^ := Eingang:Nachricht[1].Sender;
      SIGNAL AnfrageErhalten;
      Muelleimer^ : FROM Eingang GET Nachricht[1];
      IF Protokoll DO
        DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente KognitionG, ", T, ID);
        DISPLAY("Ereignis 1\n");
        DISPLAY("Anfrage von Individuum %2d\n", Eingang:Nachricht[1].Sender);
      END
    END
    ELSIF Eingang:Nachricht[1].Typ = 'Bewerbung' DO
      #-----------------------------------------------------------------------
      # Individuums bewirbt sich um Aufnahme in die Gruppe
      #-----------------------------------------------------------------------
      BewerberID^         := Eingang:Nachricht[1].Sender;
      BewerberBeurteilen^ := TRUE;
```

```
    #------------------------------------------------------------------
    # Bestimmen der Gruppenqualität, die sich durch die Aufnahme des
    # Bewerbers ergeben würde. Dies wird nur geprüft, wenn die Gruppe
    # noch nicht "voll" ist
    #------------------------------------------------------------------
    IF GroesseG < GroesseGMax DO
      #----------------------------------------------------------------------
      # Bei der Bestimmung der Gruppenqualitaet muss unbedingt beachtet
      # werden, ob schon Gruppen existieren!
      # Falls noch keine Gruppen existieren, funktioniert die Normierung
      # nicht, da RelWisGMax = RelSozGMax = 0 !
      # Dies kann nur bei einer Gruppengründung der Fall sein.
      #----------------------------------------------------------------------
      IF GroesseG = 0 DO
        #----------------------------------------------------------------------
        # Es existiert evtl. keine Gruppe, anfängliche Gruppenqualitaet ist
        # (da unechte Gruppe) in jedem Fall 1 [GQ]
        #----------------------------------------------------------------------
        QualitaetGNeu^ := 1 [GQ];
      END
      ELSE DO
        #----------------------------------------------------------------------
        # Normalfall: Es existiert mindestens eine Gruppe (zumindest diese!)
        # RelSozGMax, RelWisGMax sind > Null
        #----------------------------------------------------------------------
        QualitaetGNeu^ :=
          BestimmeQualitaetG( ((SummeWissenG + Eingang:Nachricht[1].Wissen) /
                              (GroesseG + 1) / DurchschnWissen) / RelWisGMax,
                              ((SummeSozKompetenzG
                               + Eingang:Nachricht[1].SozKompetenz) /
                               (GroesseG + 1) / DurchschnSozKompetenz
                               / RelSozGMax),
                               QualGroesse(GroesseG + 1),
                               e, f1, f2, f3);
      END
    END
    ELSE DO
      QualitaetGNeu^ := 0 [GQ];
    END
    SIGNAL   BewerbungErhalten;      # an VerhaltenG
    IF Protokoll DO
      DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente KognitionG, ", T, ID);
      DISPLAY("Ereignis 1\n");
      DISPLAY("Bewerbung von Individuum %2d\n", Eingang:Nachricht[1].Sender);
    END
  END
  ELSIF Eingang:Nachricht[1].Typ = 'Ausstieg' DO
    #----------------------------------------------------------------------
    # Individuum verläßt die Gruppe: Bitvektor Mitglied[] aktualisieren
    #----------------------------------------------------------------------
    Mitglied[Eingang:Nachricht[1].Sender]^ := FALSE;
    SIGNAL NeueGruppensituation;     # an VerhaltenG
    Muelleimer^ : FROM Eingang GET Nachricht[1];
    IF Protokoll DO
      DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente KognitionG, ", T, ID);
      DISPLAY("Ereignis 1\n");
      DISPLAY("Individuum %2d verläßt die Gruppe\n",
               Eingang:Nachricht[1].Sender);
    END
  END
END

#------------------------------------------------------------------------
#
# Ereignis 2 : Die Beurteilung des Bewerbers ist abgeschlossen
#
#------------------------------------------------------------------------
```

```
ON BewerberBeurteilt DO
  BewerberBeurteilen^ := FALSE;
  Muelleimer^ : FROM Eingang GET Nachricht[1];
END

#-------------------------------------------------------------------------
#
# Ereignis 3 : Bewerber aufnehmen
#
#-------------------------------------------------------------------------
ON BewerberAufnehmen DO
  #-----------------------------------
  # Bitvektor Mitglied[] aktualisieren
  #-----------------------------------
  Mitglied[BewerberID]^ := TRUE;
  SIGNAL NeueGruppensituation;       # an VerhaltenG

  #-------------------------------------------------------------
  # Die erste Bewerbung an einen freien Gruppenagent bewirkt
  # die Gründung dieser Gruppe; der Gruppenagent wird aktiviert
  #-------------------------------------------------------------
  IF Protokoll DO
    IF GroesseG = 0 DO
      DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente KognitionG, ", T, ID);
      DISPLAY("Ereignis 3\n");
      DISPLAY("Gruppenagent wird durch die Bewerbung von Individuum %2d ",
               BewerberID);
      DISPLAY ("aktiviert = Gruppengründung\n");
    END
    ELSE DO
      DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente KognitionG, ", T, ID);
      DISPLAY("Ereignis 3\n");
      DISPLAY("Individuum %2d in die Gruppe aufgenommen\n", BewerberID);
    END
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 4 : Ausschlußkandidat (= schlechtestes Gruppenmitglied) bestimmen
#
#-------------------------------------------------------------------------
ON BestimmeAusschlusskandidat DECLARE
  vorlaeufigerAusschlusskandidat   (INTEGER)  := 0,
  QualitaetGOhneAusschlusskandidat (REAL[GQ]) := 0 [GQ],
  HilfeQual                        (REAL[GQ]) := 0 [GQ]
DO PROCEDURE
  FOR i FROM 1 TO AnzIndividuen
  REPEAT
    IF Mitglied[i] DO
      HilfeQual :=
        BestimmeQualitaetG( ((SummeWissenG - Wissen[i]) / (GroesseG - 1)
                             / DurchschnWissen) / RelWisGMax,
                            ((SummeSozKompetenzG - SozKompetenz[i]) /
                             (GroesseG - 1) / DurchschnSozKompetenz)
                             / RelSozGMax,
                            QualGroesse(GroesseG - 1),
                            e, f1, f2, f3);

      IF HilfeQual > QualitaetGOhneAusschlusskandidat DO
        #-----------------------------------------------------
        # Dieses Mitglied drückt die Qualitaet stärker als
        # alle anderen bisher getesteten Mitglieder
        #-----------------------------------------------------
        vorlaeufigerAusschlusskandidat   := i;
        QualitaetGOhneAusschlusskandidat := HilfeQual;
      END
```

```
      END
    END_LOOP
  END
  DO TRANSITIONS
    Ausschlusskandidat^ := vorlaeufigerAusschlusskandidat;
    QualitaetGNeu^      := QualitaetGOhneAusschlusskandidat;
    SIGNAL AusschlusskandidatBestimmt; # an VerhaltenG
    IF Protokoll DO
      DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente KognitionG, ", T, ID);
      DISPLAY("Ereignis 4\n");
      DISPLAY("Individuum %2d ist Ausschlusskandidat\n",
               vorlaeufigerAusschlusskandidat);
    END
  END

  #-------------------------------------------------------------------------
  #
  # Ereignis 5 : Mitglied ausschliessen
  #
  #-------------------------------------------------------------------------
  ON MitgliedAusschliessen DO
    #-----------------------------------
    # Bitvektor Mitglied[] aktualisieren
    #-----------------------------------
    Mitglied[Ausschlusskandidat]^ := FALSE;
    AnzahlAusschluesse^ := AnzahlAusschluesse + 1;
    SIGNAL NeueGruppensituation;       # an VerhaltenG
    IF Protokoll DO
      DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente KognitionG, ", T, ID);
      DISPLAY("Ereignis 5\n");
      DISPLAY("Individuum %2d ausgeschlossen\n", Ausschlusskandidat);
    END
  END

  #-------------------------------------------------------------------------
  #
  # Ereignis 6 : Nachrichten vernichten
  #
  #-------------------------------------------------------------------------
  WHENEVER NUMBER(Muelleimer) > 0 DO
    Muelleimer^ : REMOVE Nachricht{ALL};
  END

END OF KognitionG
```

C.3.5 Die Komponente *EmotionG*

```
#-------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    EmotionG
#
#  Art:     Basiskomponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Die Komponente ist vorlaeufig leer!
#
#-------------------------------------------------------------------------

BASIC COMPONENT EmotionG
```

```
USE OF UNITS
  TIMEUNIT = [h]

DECLARATION OF ELEMENTS
SENSOR VARIABLES
DISCRETE
  ID              (INTEGER) := 0,
  Protokoll       (LOGICAL) := TRUE

END OF EmotionG
```

C.3.6 Die Komponente *StatusG*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    StatusG
#
#  Art:     Basiskomponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Status des Gruppenagenten
#       + GruppenagentAktiv: Gruppenagent leitet eine Gruppe
#
#
#  Verbindung(en) mit anderen Komponenten:
#
#    -  KognitionG an StatusG
#        + GroesseG
#
#-------------------------------------------------------------------------------

BASIC COMPONENT StatusG

USE OF UNITS
  TIMEUNIT = [h]


DECLARATION OF ELEMENTS

STATE VARIABLES
DISCRETE
  #---------------------------------------------
  # Gruppenagent ist aktiv = leitet eine Gruppe
  #---------------------------------------------
  GruppenagentAktiv (LOGICAL) := FALSE

SENSOR VARIABLES
DISCRETE
  #-------------------------------
  # Gruppengröße (von KognitionG)
  #-------------------------------
  GroesseG (INTEGER),

  ID        (INTEGER) := 0,
  Protokoll (LOGICAL) := TRUE

DYNAMIC BEHAVIOUR
```

```
  #--------------------------------------------------------------------------
  #
  # Ereignis 1 : Gruppenagent ist aktiv geworden
  #
  #--------------------------------------------------------------------------
  ON ^GroesseG >= 1^ DO
    GruppenagentAktiv^ := TRUE;
    IF Protokoll DO
      DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente StatusG, ", T, ID);
      DISPLAY("Ereignis 1\n");
      DISPLAY("Gruppenagent ist aktiv geworden\n");
    END
  END

  #--------------------------------------------------------------------------
  #
  # Ereignis 2 : Gruppenagent ist nicht mehr aktiv = Gruppe hat sich aufgelöst
  #
  #--------------------------------------------------------------------------
  ON ^GroesseG = 0^ DO
    GruppenagentAktiv^ := FALSE;
    IF Protokoll DO
      DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente StatusG, ", T, ID);
      DISPLAY("Ereignis 2\n");
      DISPLAY("Gruppenagent ist nicht mehr aktiv\n");
    END
  END

END OF StatusG
```

C.3.7 Die Komponente *VerhaltenG*

```
#--------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    VerhaltenG
#
#  Art:     Basiskomponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Beschreibung/Steuerung des Verhalten des Gruppenagenten (Regelbasis)
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - KognitionG an VerhaltenG
#       + GroesseG
#       + Mitglied
#       + BewerberID
#       + QualitaetG
#       + QualitaetGNeu
#       + Ausschlusskandidat
#       + AnfrageErhalten
#       + BewerbungErhalten
#       + AusschlusskandidatBestimmt
#       + NeueGruppensituation
#
#    - VerhaltenG an KognitionG
#       + BewerberBeurteilt
#       + BewerberAufnehmen
#       + AusschlusskandidatBestimmen
```

```
#       + MitgliedAusschliessen
#
#    - VerhaltenG an AktorG
#       + AusfuehrungsAOG
#
#-------------------------------------------------------------------------------

BASIC COMPONENT VerhaltenG

MOBILE SUBCOMPONENTS OF CLASS AusfuehrungsAOG

USE OF UNITS
  UNIT [WI] = BASIS
  UNIT [SK] = BASIS
  UNIT [IQ] = BASIS
  UNIT [GQ] = BASIS
  TIMEUNIT  = [h]

LOCAL DEFINITIONS

DIMENSIONS
  #---------------------------------------------
  # Anzahl der Individuen
  #---------------------------------------------
  AnzIndividuen := 36

VALUE SET
  #-------------------------------------------------------------------------------
  # Ausführungsanordnungs-Typen:
  # - Auskunft_verschicken:     Mitteilung an Individuum schicken (Auskunft)
  # - Zusage_verschicken:       Mitteilung an Individuum schicken (Zusage)
  # - Absage_verschicken:       Mitteilung an Individuum schicken (Absage)
  # - Mitglied_eliminieren:     Mitteilung an Individuum schicken (Ausschluss)
  # - Update_verschicken:       Mitteilung an Individuum schicken (Update)
  #-------------------------------------------------------------------------------
  AAOTyp : ('Auskunft_verschicken', 'Zusage_verschicken', 'Absage_verschicken',
            'Mitglied_eliminieren', 'Update_verschicken')

DECLARATION OF ELEMENTS

CONSTANTS
  #-------------------------------------------------------------------------------
  # Fensterbreite für Aufnahme eines Bewerbers oder Ausschluß eines Mitglieds
  #-------------------------------------------------------------------------------
  QualitaetGMinFaktor (REAL) := 0.95,
  #-------------------------------------------------
  # Zeitraster für Überwachung der Gruppenqualität
  #-------------------------------------------------
  TRaster (REAL[h]) := 0.1 [h],

  #-------------------------
  # Maximale Gruppengroesse
  #-------------------------
  GroesseGMax (INTEGER) := 6

STATE VARIABLES
DISCRETE
  #------------------------------
  # Qualitätsstandard der Gruppe
  #------------------------------
  QualitaetGMin (REAL[GQ]) := 0 [GQ],

  #-----------------------------------------------------
  # Zeitpunkt für die nächste Kontrolle der Gruppenqualität
  #-----------------------------------------------------
  TKontrolle (REAL[h]) := 0 [h],
```

```
  #-------------------------------------------------------------------
  # Falls Update = TRUE werden Ausführungsanordnungen vom Typ
  # 'Update_verschicken' erzeugt
  #-------------------------------------------------------------------
  Update (LOGICAL) := FALSE,

  #-----------------------------------------------------
  # Laufvariable für Generierung der Ausführungsanordnungen
  # 'Update_verschicken'
  #-----------------------------------------------------
  UpdateID (INTEGER) := 0,

  #-----------------------------------
  # Anzahl der bisherigen Zusagen
  #-----------------------------------
  AnzahlZusagen (INTEGER) := 0,

  #-----------------------------------
  # Anzahl der bisherigen Absagen
  #-----------------------------------
  AnzahlAbsagen (INTEGER) := 0

SENSOR VARIABLES
DISCRETE
  #-------------------------------
  # Gruppengröße (von KognitionG)
  #-------------------------------
  GroesseG (INTEGER) := 0,

  #-------------------------------------------------------------------------------
  # Bitvektor für Gruppenmitglieder: Mitglied[i] = TRUE bedeutet, daß das
  # Individuum i Mitglied in der von diesem Gruppenagenten verwalteten Gruppe
  # ist (von KognitionG)
  #-------------------------------------------------------------------------------
  ARRAY [AnzIndividuen] Mitglied (LOGICAL) := FALSE,

  #-----------------------------------------------------------------
  # ID des Bewerbers oder des anfragenden Individuums (von KognitionG)
  #-----------------------------------------------------------------
  BewerberID (INTEGER) := 0,

  #-----------------------------------------------------------------
  # potentielle Qualitaet der Gruppe nach Änderung der Gruppengröße
  # (von KognitionG)
  #-----------------------------------------------------------------
  QualitaetGNeu (REAL[GQ]) := 0 [GQ],

  #-----------------------------------------------------
  # Kandidat für Ausschluß aus der Gruppe (von KognitionG)
  #-----------------------------------------------------
  Ausschlusskandidat (INTEGER) := 0,

  ID (INTEGER)        := 0,
  Protokoll (LOGICAL) := TRUE

CONTINUOUS
  #-------------------------------------
  # Qualität der Gruppe (von KognitionG)
  #-------------------------------------
  QualitaetG (REAL[GQ]) := 0 [GQ]

TRANSITION INDICATORS
  #-----------------------------------------------------------------
  # Die Beurteilung des Bewerbers ist abgeschlossen (an KognitionG)
  #-----------------------------------------------------------------
  BewerberBeurteilt,
```

```
  #-----------------------------------
  # Bewerber aufnehmen (an KognitionG)
  #-----------------------------------
  BewerberAufnehmen,

  #-------------------------------------------
  # Ausschlußkandidat bestimmen (an KognitionG)
  #-------------------------------------------
  BestimmeAusschlusskandidat,

  #---------------------------------------
  # Mitglied ausschließen (an KognitionG)
  #---------------------------------------
  MitgliedAusschliessen

SENSOR INDICATORS
  #-------------------------------------------------
  # Anfrage von Individuum erhalten (von KognitionG)
  #-------------------------------------------------
  AnfrageErhalten,

  #---------------------------------------------------
  # Bewerbung von Individuum erhalten (von KognitionG)
  #---------------------------------------------------
  BewerbungErhalten,

  #-------------------------------------------
  # Ausschlußkandidat bestimmt (von KognitionG)
  #-------------------------------------------
  AusschlusskandidatBestimmt,

  #-------------------------------------------------------------------
  # Neue Gruppensituation durch Aufnahme eines Bewerbers oder Ausschluß
  # eines Mitglieds (von KognitionG)
  #-------------------------------------------------------------------
  NeueGruppensituation

LOCATIONS
  #-----------------------------------------
  # Generieren/Ablage von Ausführungsanordnungen
  #-----------------------------------------
  Ausgang (AusfuehrungsAOG) := 0 AusfuehrungsAOG,

  #-------------------------------------------------------------------------
  # Puffer für Ausführungsanordnungen, die gleichzeitig verschickt werden
  # sollen und sich überschneiden können ('Update_verschicken' ausgelöst
  # durch Aufnahme, Ausschluß oder Ausstieg eines Mitglieds)
  #-------------------------------------------------------------------------
  Puffer (AusfuehrungsAOG) := 0 AusfuehrungsAOG,

  #------------------
  # Überschneidungen
  #------------------
  Ueberschneidungen (AusfuehrungsAOG) := 0 AusfuehrungsAOG

DYNAMIC BEHAVIOUR

  #-----------------------------------------------------------------------------
  #
  # Ereignis 1 : Anfrage von Individuum eingetroffen
  #
  #-----------------------------------------------------------------------------
  ON AnfrageErhalten DO
    Ausgang^ : ADD 1 NEW AusfuehrungsAOG
                 CHANGING
                   Typ^        := 'Auskunft_verschicken';
                   ID^         := BewerberID;
```

```
                QualitaetG^ := QualitaetG;
              END
  IF Protokoll DO
    DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente VerhaltenG, ", T, ID);
    DISPLAY("Ereignis 1\n");
    DISPLAY("AAO Auskunft für Individuum  %2d generiert\n", BewerberID);
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 2 : Bewerbung von Individuum eingetroffen
#
#-------------------------------------------------------------------------
ON BewerbungErhalten DO
  IF (QualitaetGNeu > QualitaetGMin OR QualitaetGNeu > QualitaetG)
     AND GroesseG < GroesseGMax DO
    #-----------------------------------------------------------
    # Der Bewerber erfüllt die Qualitaetsanforderungen der Gruppe
    # oder kann die Qualität zumindest erhöhen
    #-----------------------------------------------------------
    Ausgang^ : ADD 1 NEW AusfuehrungsAOG
                CHANGING
                  Typ^      := 'Zusage_verschicken';
                  ID^       := BewerberID;
                  GroesseG^ := GroesseG + 1;
                END

    SIGNAL BewerberAufnehmen;        # an KognitionG

    AnzahlZusagen^ := AnzahlZusagen + 1;

    IF QualitaetGNeu > QualitaetGMin DO
      IF Protokoll DO
        DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente VerhaltenG, ",T,ID);
        DISPLAY("Ereignis 2\n");
        DISPLAY("AAO Zusage für Individuum  %2d generiert", BewerberID);
        DISPLAY("(Standard)\n");
      END
    END
    ELSE DO
      IF Protokoll DO
        DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente VerhaltenG, ",T,ID);
        DISPLAY("Ereignis 2\n");
        DISPLAY("AAO Zusage für Individuum  %2d generiert", BewerberID);
        DISPLAY("(Verbesserung)\n");
      END
    END
  END
  ELSE DO
    #-------------------------------------------------------------------
    # Der Bewerber erfüllt die Qualitaetsanforderungen der Gruppe nicht
    #-------------------------------------------------------------------
    Ausgang^ : ADD 1 NEW AusfuehrungsAOG
                CHANGING
                  Typ^      := 'Absage_verschicken';
                  ID^       := BewerberID;
                END
    AnzahlAbsagen^ := AnzahlAbsagen + 1;
    IF Protokoll DO
      IF GroesseG < 6 DO
        DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente ", T, ID);
        DISPLAY("VerhaltenG, Ereignis 2\n");
        DISPLAY("AAO Absage für Individuum %2d generiert (1)\n", BewerberID);
      END
      ELSE DO
        DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente ", T, ID);
```

```
        DISPLAY("VerhaltenG, Ereignis 2\n");
        DISPLAY("AAO Absage für Individuum %2d generiert (2)\n", BewerberID);
      END
    END
  END
  #-------------------------------------------------
  # Die Beurteilung des Bewerbers ist abgeschlossen
  #-------------------------------------------------
  SIGNAL BewerberBeurteilt;          # an KognitionG
END

#-------------------------------------------------------------------------------
#
# Ereignis 3 : Die Gruppe hat sich verändert, da ein Bewerber aufgenommen,
#              ein Mitglied ausgeschlossen oder ein Mitglied die
#              die Gruppe verlassen hat. der Qualitätsstandard wird neu
#              festgelegt.
#              ==> Alle Gruppenmitglieder müssen über die neue Situation
#                  (GroesseG) informiert werden
#
#-------------------------------------------------------------------------------
ON NeueGruppensituation DO
  IF GroesseG > 0 DO

    #-----------------------------
    # Qualitaetsstandard anpassen
    #-----------------------------
    QualitaetGMin^ := QualitaetGMinFaktor * QualitaetG;

    #-----------------------------------------
    # Falls die Gruppe noch Mitglieder hat ...
    #-----------------------------------------
    IF Update DO
      #-----------------------------------------------------
      # Überschneidung! Es werden gerade Updates erzeugt!
      #-----------------------------------------------------
      Ueberschneidungen^ : ADD 1 NEW AusfuehrungsAOG;
      IF Protokoll DO
        DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente VerhaltenG", T, ID);
        DISPLAY(", Ereignis 3\n");
        DISPLAY("Überschneidung von Updates erkannt!\n");
      END
    END
    ELSE DO
      Update^   := TRUE;
      UpdateID^ := 1;
      IF Protokoll DO
        DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente Verh", T, ID);
        DISPLAY("altenG, Ereignis 3\n");
        DISPLAY("Neue Gruppensituation ==> Updates generieren!\n");
      END
    END
  END
  ELSE DO
    #-----------------------------------
    # Die Gruppe hat sich aufgelöst
    #-----------------------------------
    # Qualitatsstandard auf Null setzen
    #-----------------------------------
    QualitaetGMin^ := 0 [GQ];
  END
END
```

```
#-------------------------------------------------------------------------
#
# Ereignis 4 : Überschneidung von Ereignissen, die ein Update an die
#              Gruppenmitglieder erforderlich machen
#
#-------------------------------------------------------------------------
WHENEVER NUMBER(Ueberschneidungen) > 0 DO
  #-----------------------------------------------------------------
  # Löschen der veralteten Ausführungsanordnungen 'Update_verschicken'
  #-----------------------------------------------------------------
  Puffer^   : REMOVE AusfuehrungsAOG{ALL};
  UpdateID^ := 1;
  Ueberschneidungen^ : REMOVE AusfuehrungsAOG[1];
END

#-------------------------------------------------------------------------
#
# Ereignis 5 : Erzeugen der Ausführungsanordnungen 'Update_verschicken'
#
#-------------------------------------------------------------------------
WHENEVER Update AND NUMBER(Ueberschneidungen) = 0 DO
  IF UpdateID <= AnzIndividuen DO
    IF Mitglied[UpdateID] DO
      Puffer^ : ADD 1 NEW AusfuehrungsAOG
                  CHANGING
                    Typ^      := 'Update_verschicken';
                    ID^       := UpdateID;
                    GroesseG^ := GroesseG;
                  END
      END
    UpdateID^ := UpdateID + 1;
  END
  ELSE DO
    #-----------------------------------------------------------------
    # Ausführungsanordnung 'Update_verschicken' für alle Gruppenmitglieder
    # generiert
    #-----------------------------------------------------------------
    Update^   := FALSE;
    Puffer^   : TO Ausgang SEND AusfuehrungsAOG {ALL};
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 6 : Überwachung der Gruppenqualität
#
#-------------------------------------------------------------------------
WHENEVER T > TKontrolle AND GroesseG > 2 DO
  TKontrolle^ := T + TRaster;
  IF QualitaetG < QualitaetGMin DO
    SIGNAL BestimmeAusschlusskandidat;    # an KognitionG
    IF Protokoll DO
      DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente VerhaltenG, ", T, ID);
      DISPLAY("Ereignis 6\n");
      DISPLAY("Qualitätsstandard unterschritten!\n");
    END
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 7 : Überprüfen, ob der Ausschluss des schlechtesten Mitglieds
#              die Qualität der Gruppe wieder auf das vereinbarte Mindest-
#              niveau hebt und in diesem Fall den Ausschlusskandidat aus der
#              Gruppe ausschliessen
#
#-------------------------------------------------------------------------
```

```
  ON AusschlusskandidatBestimmt DO
    IF QualitaetGNeu > QualitaetGMin DO
      #-------------------------------------------------------------
      # Der Ausschluss würde die Qualität der Gruppe wieder auf das
      # vereinbarte Mindestniveau heben
      #-------------------------------------------------------------
      Ausgang^ : ADD 1 NEW AusfuehrungsAOG
                   CHANGING
                     Typ^ := 'Mitglied_eliminieren';
                     ID^  := Ausschlusskandidat;
                   END
      SIGNAL MitgliedAusschliessen;    # an KognitionG
      IF Protokoll DO
        DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente VerhaltenG, ", T, ID);
        DISPLAY("Ereignis 7\n");
        DISPLAY("AAO Mitglied_eliminieren für Individuum  %2d generiert\n",
                 Ausschlusskandidat);
      END
    END
  END

END OF VerhaltenG
```

C.3.8 Die Komponente *AktorG*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    AktorG
#
#  Art:     Basiskomponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Umsetzen von Ausführungsanordnungen in Aktionen/Nachrichten
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - VerhaltenG an AktorG
#        + AusfuehrungsAOG
#
#-------------------------------------------------------------------------------

BASIC COMPONENT AktorG

MOBILE SUBCOMPONENTS OF CLASS Nachricht,
                              AusfuehrungsAOG

USE OF UNITS
  UNIT [WI]  = BASIS
  UNIT [SoZ] = BASIS
  TIMEUNIT   = [h]

LOCAL DEFINITIONS

VALUE SET
  #-----------------------------------------------------------------------------
  # Nachrichtentypen:
  #  - Individuum an Gruppenagent
  #     + Anfrage:       Individuum fragt Gruppenagent nach Gruppeninformation
  #     + Bewerbung:     Individuum bewirbt sich um Aufnahme in die Gruppe
```

```
  #     + Ausstieg:      Individuum verlaesst die Gruppe
  #
  # - Gruppenagent an Individuum
  #     + Auskunft:      Gruppenagent antwortet auf Anfrage des Individuums
  #     + Absage:        Individuum wird nicht aufgenommen
  #     + Zusage:        Individuum wird in die Gruppe aufgenommen
  #     + Ausschluss:    Grupppenagent schliesst Individuum aus der Gruppe aus
  #     + Update:        Aenderung der diskreten, für die Gruppenmitglieder
  #                      relevanten Informationen (GroesseG)
  #-------------------------------------------------------------------------
  Nachrichtentyp : ('Anfrage', 'Bewerbung', 'Ausstieg',
                    'Auskunft', 'Absage', 'Zusage', 'Ausschluss', 'Update')

VALUE SET
  #-------------------------------------------------------------------------
  # Ausführungsanordnungs-Typen:
  # - Auskunft_verschicken:     Mitteilung an Individuum schicken (Auskunft)
  # - Zusage_verschicken:       Mitteilung an Individuum schicken (Zusage)
  # - Absage_verschicken:       Mitteilung an Individuum schicken (Absage)
  # - Mitglied_eliminieren:     Mitteilung an Individuum schicken (Ausschluss)
  # - Update_verschicken:       Mitteilung an Individuum schicken (Update)
  #-------------------------------------------------------------------------
  AAOTyp : ('Auskunft_verschicken', 'Zusage_verschicken', 'Absage_verschicken',
            'Mitglied_eliminieren', 'Update_verschicken')

DECLARATION OF ELEMENTS

STATE VARIABLES
DISCRETE
  #-------------------------------------------------------------------------
  # Wenn Updates = TRUE, dann sind Ausführungsanordnungen des Typs
  # 'Update_verschicken' eingetroffen
  #-------------------------------------------------------------------------
  Updates (LOGICAL) := FALSE

SENSOR VARIABLES
DISCRETE
  ID        (INTEGER) := 0,
  Protokoll (LOGICAL) := TRUE

LOCATIONS
  #---------------------------------------
  # Vernichten der Ausführungsanordnungen
  #---------------------------------------
  Muelleimer (AusfuehrungsAOG) := 0 AusfuehrungsAOG,

  #-----------------------------------------------
  # Generieren einer Nachricht an ein Individuum
  #-----------------------------------------------
  NeueNachricht (Nachricht) := 0 Nachricht,

  #-----------------------------------------------------------------------
  # Generieren von Nachrichten, die gleichzeitig an mehrere Individuen
  # verschickt werden sollen
  #-----------------------------------------------------------------------
  NeueNachrichten (Nachricht) := 0 Nachricht

SENSOR LOCATIONS
  #---------------------------------------
  # Empfangen von Ausführungsanordnungen
  #---------------------------------------
  Eingang (AusfuehrungsAOG) := 0 AusfuehrungsAOG,

  #-----------------------------------------------------------------------
  # Eingangsbereich für Nachrichten dieses Gruppenagenten im Connector
  #-----------------------------------------------------------------------
  Ausgang (Nachricht) := 0 Nachricht
```

```
DYNAMIC BEHAVIOUR

  #-------------------------------------------------------------------------
  #
  # Ereignis 1 : Eine Ausführungsanordnung trifft ein
  #
  #-------------------------------------------------------------------------
  WHENEVER NUMBER(Eingang) > 0 DO
    IF Eingang:AusfuehrungsAOG[1].Typ = 'Auskunft_verschicken' DO
      #---------------------------------
      # Nachricht 'Auskunft' generieren
      #---------------------------------
      NeueNachricht^ : ADD 1 NEW Nachricht
                       CHANGING
                         Sender^     := ID;
                         Empfaenger^ := Eingang:AusfuehrungsAOG[1].ID;
                         Typ^        := 'Auskunft';
                         QualitaetG^ := Eingang:AusfuehrungsAOG[1].QualitaetG;
                         GroesseG^   := Eingang:AusfuehrungsAOG[1].GroesseG;
                       END
      #-----------------------------------------------
      # Ausfuehrungsanordnung in den Mülleimer legen
      #-----------------------------------------------
      Muelleimer^ : FROM Eingang GET AusfuehrungsAOG[1];
      IF Protokoll DO
        DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente AktorG, ", T, ID);
        DISPLAY("Ereignis 1\n");
        DISPLAY("Nachricht 'Auskunft' für Individuum %2d generiert\n",
                Eingang:AusfuehrungsAOG[1].ID);
      END
    END
    ELSIF Eingang:AusfuehrungsAOG[1].Typ = 'Zusage_verschicken' DO
      #-------------------------------------------------------------------
      # Nachricht 'Zusage' generieren
      #-------------------------------------------------------------------
      NeueNachricht^ : ADD 1 NEW Nachricht
                         CHANGING
                           Sender^     := ID;
                           Empfaenger^ := Eingang:AusfuehrungsAOG[1].ID;
                           Typ^        := 'Zusage';
                           GroesseG^   := Eingang:AusfuehrungsAOG[1].GroesseG;
                         END
      #-----------------------------------------------
      # Ausfuehrungsanordnung in den Mülleimer legen
      #-----------------------------------------------
      Muelleimer^ : FROM Eingang GET AusfuehrungsAOG[1];
      IF Protokoll DO
        DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente AktorG, ", T, ID);
        DISPLAY("Ereignis 1\n");
        DISPLAY("Nachricht 'Zusage' für Individuum %2d generiert\n",
                Eingang:AusfuehrungsAOG[1].ID);
      END
    END
    ELSIF Eingang:AusfuehrungsAOG[1].Typ = 'Absage_verschicken' DO
      #-------------------------------------------------------------------
      # Nachricht 'Absage' generieren
      #-------------------------------------------------------------------
      NeueNachricht^ : ADD 1 NEW Nachricht
                         CHANGING
                           Sender^     := ID;
                           Empfaenger^ := Eingang:AusfuehrungsAOG[1].ID;
                           Typ^        := 'Absage';
                         END
      #-----------------------------------------------
      # Ausfuehrungsanordnung in den Mülleimer legen
      #-----------------------------------------------
      Muelleimer^ : FROM Eingang GET AusfuehrungsAOG[1];
```

```
    IF Protokoll DO
      DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente AktorG, ", T, ID);
      DISPLAY("Ereignis 1\n");
      DISPLAY("Nachricht 'Absage' für Individuum %2d generiert\n",
               Eingang:AusfuehrungsAOG[1].ID);
    END
  END
  ELSIF Eingang:AusfuehrungsAOG[1].Typ = 'Mitglied_eliminieren' DO
    #-------------------------------------------------------------------
    # Nachricht 'Ausschluss' generieren
    #-------------------------------------------------------------------
    NeueNachricht^ : ADD 1 NEW Nachricht
                     CHANGING
                       Sender^     := ID;
                       Empfaenger^ := Eingang:AusfuehrungsAOG[1].ID;
                       Typ^        := 'Ausschluss';
                     END
    #---------------------------------------------
    # Ausfuehrungsanordnung in den Mülleimer legen
    #---------------------------------------------
    Muelleimer^ : FROM Eingang GET AusfuehrungsAOG[1];
    IF Protokoll DO
      DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente AktorG, ", T, ID);
      DISPLAY("Ereignis 1\n");
      DISPLAY("Nachricht 'Ausschluss' für Individuum %2d generiert\n",
               Eingang:AusfuehrungsAOG[1].ID);
    END
  END
  ELSIF Eingang:AusfuehrungsAOG[1].Typ = 'Update_verschicken' DO
    #-------------------------------------------------------------------------
    # Nachricht 'Update' generieren
    # Alle Nachrichten dieses Typs sollen alle Gruppenmitglieder zum gleichen
    # Takt erreichen. Sie werden sequentiell erzeugt und dann gleichzeitig
    # in den Connector geleitet.
    #-------------------------------------------------------------------------
    NeueNachrichten^ : ADD 1 NEW Nachricht
                       CHANGING
                         Sender^     := ID;
                         Empfaenger^ := Eingang:AusfuehrungsAOG[1].ID;
                         Typ^        := 'Update';
                         GroesseG^   := Eingang:AusfuehrungsAOG[1].GroesseG;
                       END
    Updates^ := TRUE;
    #---------------------------------------------
    # Ausfuehrungsanordnung in den Mülleimer legen
    #---------------------------------------------
    Muelleimer^ : FROM Eingang GET AusfuehrungsAOG[1];
    IF Protokoll DO
      DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente AktorG, ", T, ID);
      DISPLAY("Ereignis 1\n");
      DISPLAY("Nachricht 'Update' für Individuum %2d generiert\n",
               Eingang:AusfuehrungsAOG[1].ID);
    END
  END
END

#-------------------------------------------------------------------------
#
# Ereignis 2 : Weiterleiten der generierten Nachricht an den Connector
#
#-------------------------------------------------------------------------
WHENEVER NUMBER(NeueNachricht) > 0 DO
  NeueNachricht^ : TO Ausgang SEND Nachricht[1];
END
```

```
  #-------------------------------------------------------------------------------
  #
  # Ereignis 3 : Wenn alle Nachrichten des Typs 'Update' generiert sind,
  #              werden sie gleichzeitig weitergeleitet.
  #
  #-------------------------------------------------------------------------------
  ON ^Updates AND NUMBER(Eingang) = 0^ DO
    NeueNachrichten^ : TO Ausgang SEND Nachricht{ALL i |
                       NeueNachrichten:Nachricht[i].Typ = 'Update'};
    Updates^ := FALSE;
    IF Protokoll DO
      DISPLAY("T = %5.2f, Gruppenagent %2d, Komponente AktorG, ", T, ID);
        DISPLAY("Ereignis 3\n");
      DISPLAY(">>>Alle Nachrichten 'Update' gleichzeitig weitergeleitet\n");
      END
  END

  #-------------------------------------------------------------------------------
  #
  # Ereignis 4 : Vernichten der abgearbeiteten Ausführungsanordnungen
  #
  #-------------------------------------------------------------------------------
  WHENEVER NUMBER(Muelleimer) > 0 DO
    Muelleimer^ : REMOVE AusfuehrungsAOG{ALL};
  END

END OF AktorG
```

C.4 Die Komponente *Connector*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    Connector
#
#  Art:     Basiskomponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Zentrale Komponente für den Nachrichtenaustausch zwischen Individuen
#      und Grupppenagenten und Bereitstellung von Informationen (Blackboard)
#
#    - Informationen für Individuen (über Gruppenagenten):
#       + QualitaetG []
#       + GruppenagentAktiv []
#
#    - Informationen für Gruppenagenten (über Individuen):
#       + Wissen []
#       + SozKompetenz
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Connector    --> Individuen, Gruppenagent
#    - Individuen   --> Connector
#    - Gruppenagent --> Connecor
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Connector
```

```
MOBILE SUBCOMPONENTS OF CLASS Nachricht

USE OF UNITS
  UNIT [WI]  = BASIS
  UNIT [SK]  = BASIS
  UNIT [GQ]  = BASIS
  TIMEUNIT   = [h]

LOCAL DEFINITIONS

DIMENSIONS
  #-----------------------------------------------
  # Anzahl der Individuen
  #-----------------------------------------------
  AnzIndividuen := 36,

  #-----------------------------------------------
  #  Obergrenze für die Anzahl der Gruppenagenten
  #-----------------------------------------------
  AnzGruppenagenten := 36

VALUE SET
  #-------------------------------------------------------------------------------
  # Nachrichtentypen:
  #  - Individuum an Gruppenagent
  #     + Anfrage:       Individuum fragt Gruppenagent nach Gruppeninformation
  #     + Bewerbung:     Individuum bewirbt sich um Aufnahme in die Gruppe
  #     + Ausstieg:      Individuum verlaesst die Gruppe
  #
  # - Gruppenagent an Individuum
  #     + Auskunft:      Gruppenagent antwortet auf Anfrage des Individuums
  #     + Absage:        Individuum wird nicht aufgenommen
  #     + Zusage:        Individuum wird in die Gruppe aufgenommen
  #     + Ausschluss:    Grupppenagent schliesst Individuum aus der Gruppe aus
  #     + Update:        Aenderung der diskreten, für die Gruppenmitglieder
  #                      relevanten Informationen (GroesseG)
  #-------------------------------------------------------------------------------
  Nachrichtentyp : ('Anfrage', 'Bewerbung', 'Ausstieg',
                    'Auskunft', 'Absage', 'Zusage', 'Ausschluss', 'Update')

  #-------------------------------------------------------------------------------
  # NeueNachricht()
  #
  # Typ:             Funktion
  # Aufrufparameter: ARRAY LOCATION (Nachricht)
  # Rückgabewert:    LOGICAL
  #
  # Beschreibung: Diese Funktion liefert den Wert TRUE zurueck, falls sich in
  #               einer der uebergebenen Loactions eine Nachricht befindet
  #-------------------------------------------------------------------------------
  FUNCTION NeueNachricht(ARRAY [n] LOCATION FOR Nachricht: Eingang --> LOGICAL)
    DECLARE dummy (LOGICAL) := TRUE
  BEGIN
    FOR i FROM 1 TO n
    REPEAT
      IF NUMBER(Eingang[i]) > 0 DO
        RETURN (TRUE);
      END
    END_LOOP
    RETURN (FALSE);
  END_FUNC

DECLARATION OF ELEMENTS

DEPENDENT VARIABLES
DISCRETE
```

```
  #-----------------------------------
  # Blackboard (für Individuen lesbar)
  #-----------------------------------
  ARRAY [AnzGruppenagenten] GruppenagentAktiv (LOGICAL) := FALSE

CONTINUOUS
  #-----------------------------------
  # Blackboard (für Individuen lesbar)
  #-----------------------------------
  ARRAY [AnzGruppenagenten] QualitaetG (REAL[GQ]) := 0 [GQ],

  #---------------------------------------
  # Blackboard (für Gruppenagenten lesbar)
  #---------------------------------------
  ARRAY [AnzIndividuen] Wissen       (REAL[WI]),
  ARRAY [AnzIndividuen] SozKompetenz (REAL[SK])

SENSOR VARIABLES
DISCRETE
  #-----------------------------------------
  # "Informationsbeschaffung" für Blackboard
  #-----------------------------------------
  ARRAY [AnzGruppenagenten] GruppenagentAktiv_S (LOGICAL)

CONTINUOUS
  #-----------------------------------------
  # "Informationsbeschaffung" für Blackboard
  #-----------------------------------------
  ARRAY [AnzGruppenagenten] QualitaetG_S (REAL[GQ]),

  ARRAY [AnzIndividuen] Wissen_S       (REAL[WI]),
  ARRAY [AnzIndividuen] SozKompetenz_S (REAL[SK])

LOCATIONS
  #-----------------------------------------------------
  # Eingangsbereich für Nachrichten von Individuen
  #-----------------------------------------------------
  ARRAY [AnzIndividuen] EingangI (Nachricht) := 0 Nachricht,

  #-----------------------------------------------------
  # Eingangsbereich für Nachrichten von Gruppenagenten
  #-----------------------------------------------------
  ARRAY [AnzGruppenagenten] EingangG (Nachricht) := 0 Nachricht,

  #-----------------------------------------------------
  # Ausgangsbereich für Nachrichten an Individuen
  #-----------------------------------------------------
  ARRAY [AnzIndividuen] AusgangI (Nachricht) := 0 Nachricht,

  #-----------------------------------------------------
  # Ausgangsbereich für Nachrichten an Gruppenagenten
  #-----------------------------------------------------
  ARRAY [AnzGruppenagenten] AusgangG (Nachricht) := 0 Nachricht,

  #-----------------------------------------------------------
  # Zentrale Bearbeitungsstation für eingehende Nachrichten
  #-----------------------------------------------------------
  Bearbeitung (Nachricht) := 0 Nachricht

DYNAMIC BEHAVIOUR

  #-------------------------------------------------
  # Bereitstellen der Informationen auf dem Blackboard
  #-------------------------------------------------
  QualitaetG {i OF 1..AnzGruppenagenten} := QualitaetG_S[i];
  GruppenagentAktiv {i OF 1..AnzGruppenagenten} := GruppenagentAktiv_S[i];
```

```
Wissen       {i OF 1..AnzIndividuen} := Wissen_S[i];
SozKompetenz {i OF 1..AnzIndividuen} := SozKompetenz_S[i];

#-------------------------------------------------------------------------
# Bearbeitung der eingehenden Nachrichten
#-------------------------------------------------------------------------

#-------------------------------------------------------------------------
#
# Ereignis 1 : Nachrichten aus den Eingangsbereichen zur Bearbeitungsstation
#              schicken
#
#-------------------------------------------------------------------------
WHENEVER NeueNachricht(ARRAY LOCATION EingangI) DO
  Bearbeitung^ : FROM EingangI{ALL} GET Nachricht{ALL};
END

WHENEVER NeueNachricht(ARRAY LOCATION EingangG) DO
  Bearbeitung^ : FROM EingangG{ALL} GET Nachricht{ALL};
END

#-------------------------------------------------------------------------
#
# Ereignis 2 : Nachrichten weiterleiten. Nachrichten gleichen Typs sollen
#              immer gleichzeitig (gleicher Takt!) weitergeleitet werden.
#
#-------------------------------------------------------------------------
WHENEVER NUMBER(Bearbeitung) > 0 DO

  #-----------------------------------------------
  # Nachrichten von Individuum an Gruppenagent(en)
  #-----------------------------------------------

  IF Bearbeitung:Nachricht[1].Typ = 'Anfrage' DO
    #-----------------------------------------------------------------------
    # Anfragen des Individuum an alle aktiven Gruppenagenten gleichzeitig
    #-----------------------------------------------------------------------
    AusgangG{ALL i}^ : FROM Bearbeitung GET Nachricht {ALL j |
                         Bearbeitung:Nachricht[j].Empfaenger = i AND
                         Bearbeitung:Nachricht[j].Typ = 'Anfrage'};
  END
  ELSIF Bearbeitung:Nachricht[1].Typ = 'Bewerbung' DO
    #---------------------------------------
    # Bewerbung an Gruppenagent weiterleiten
    #---------------------------------------
    Bearbeitung^ : TO AusgangG[Bearbeitung:Nachricht[1].Empfaenger]
                   SEND Nachricht[1];
  END
  ELSIF Bearbeitung:Nachricht[1].Typ = 'Ausstieg' DO


    #---------------------------------------
    # Ausstieg an Gruppenagent weiterleiten
    #---------------------------------------
    Bearbeitung^ : TO AusgangG[Bearbeitung:Nachricht[1].Empfaenger]
                   SEND Nachricht[1];
  END

  #-----------------------------------------------
  # Nachrichten von Gruppenagent an Individuum(en)
  #-----------------------------------------------

  ELSIF Bearbeitung:Nachricht[1].Typ = 'Auskunft' DO
    #-------------------------------------------------------------
    # Auskunft aller Gruppenagenten gleichzeitig an das Individuum
    #-------------------------------------------------------------
    Bearbeitung^ : TO AusgangI[Bearbeitung:Nachricht[1].Empfaenger]
```

```
                    SEND Nachricht{ALL j |
                      Bearbeitung:Nachricht[j].Empfaenger =
                          Bearbeitung:Nachricht[1].Empfaenger AND
                      Bearbeitung:Nachricht[j].Typ = 'Auskunft' };
    END
    ELSIF Bearbeitung:Nachricht[1].Typ = 'Absage' DO
      #-----------------------------------
      # Absage an Individuum weiterleiten
      #-----------------------------------
      Bearbeitung^ : TO AusgangI[Bearbeitung:Nachricht[1].Empfaenger]
                     SEND Nachricht[1];
    END
    ELSIF Bearbeitung:Nachricht[1].Typ = 'Zusage' DO
      #-----------------------------------
      # Zusage an Individuum weiterleiten
      #-----------------------------------
      Bearbeitung^ : TO AusgangI[Bearbeitung:Nachricht[1].Empfaenger]
                     SEND Nachricht[1];
    END
    ELSIF Bearbeitung:Nachricht[1].Typ = 'Ausschluss' DO
      #--------------------------------------
      # Ausschluss an Individuum weiterleiten
      #--------------------------------------
      Bearbeitung^ : TO AusgangI[Bearbeitung:Nachricht[1].Empfaenger]
                     SEND Nachricht[1];
    END
    ELSIF Bearbeitung:Nachricht[1].Typ = 'Update' DO
      #--------------------------------------
      # Update an alle Individuen gleichzeitig
      #--------------------------------------
      AusgangI{ALL i}^ : FROM Bearbeitung GET Nachricht {ALL j |
                           Bearbeitung:Nachricht[j].Empfaenger = i AND
                           Bearbeitung:Nachricht[j].Typ = 'Update'};
    END
  END
END OF Connector
```

C.5 Die Komponente *Statistik*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    Statistik
#
#  Art:     Basiskomponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Zentrale Komponente für die Informationssammlung und Aufbereitung
#      der für die Animation relevanten Informationen
#
#    - Informationen über Gruppen(agenten):
#       + GruppenagentAktiv
#       + RelWisG
#       + RelSozG
#       + QualitaetG
#       + GroesseG
#
#    - Informationen über Individuen:
#       + Wissen
```

```
#       + SozAkt
#       + SozKompetenz
#       + Gruppenmitglied
#       + GruppenID
#       + QualitaetI
#
#       + DurchschnWissen
#       + DurchschnSozKompetenz
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Individuen an Statistik
#       + Wissen
#       + SozAkt
#       + SozKompetenz
#       + Gruppenmitglied
#       + GruppenID
#       + QualitaetI
#
#    - Gruppenagent an Statistik
#       + GruppenagentAktiv
#       + RelWisG
#       + RelSozG
#       + QualitaetG
#       + GroesseG
#
#    - Statistik an Gruppenagent
#       + DurchschnWissen
#       + DurchschnSozKompetenz
#       + RelWissenGMax
#       + RelSozGMax
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Statistik

MOBILE SUBCOMPONENTS OF CLASS Indivi,
                              Gruppe

USE OF UNITS
  UNIT [WI]   = BASIS
  UNIT [SoZ]  = BASIS
  UNIT [SK]   = BASIS
  UNIT [GQ]   = BASIS
  UNIT [IndQ] = BASIS
  TIMEUNIT    = [h]

LOCAL DEFINITIONS

DIMENSIONS

  #---------------------------------------------
  # Anzahl der Individuen
  #---------------------------------------------
  AnzIndividuen := 36,

  #---------------------------------------------
  #  Obergrenze für die Anzahl der Gruppenagenten
  #---------------------------------------------
  AnzGruppenagenten := 36

  #-------------------------------------------------------------------------------
  # BestimmeAnzahlGruppen()
  #
  # Typ:             Funktion
  # Aufrufparameter: ARRAY LOGICAL GruppenagentAktiv
  # Rückgabewert:    INTEGER
```

```
#
# Beschreibung: Diese Funktion bestimmt die Anzahl der Gruppen
#-------------------------------------------------------------------------
FUNCTION BestimmeAnzahlGruppen (ARRAY [n] LOGICAL: GruppenagentAktiv
                                --> INTEGER)
DECLARE
  AnzahlGruppen (INTEGER)  := 0
BEGIN
  FOR i FROM 1 TO AnzGruppenagenten
  REPEAT
    IF GruppenagentAktiv[i] DO
      AnzahlGruppen := AnzahlGruppen + 1;
    END
  END_LOOP
  RETURN(AnzahlGruppen);
END_FUNC

#-------------------------------------------------------------------------
# BestimmeSozPosition()
#
# Typ:             Funktion
# Aufrufparameter: INTEGER ID, ARRAY QualitaetI
# Rückgabewert:    INTEGER
#
# Beschreibung: Diese Funktion bestimmt für dieses Individuum die soziale
#               Position, die es aufgrund seiner Qualitaet im Gesamtsystem
#               einnimmt
#-------------------------------------------------------------------------
FUNCTION BestimmeSozPosition (INTEGER: ID,
                              ARRAY [n] REAL[IndQ]: Qualitaet  --> INTEGER)
DECLARE
  Position (INTEGER)  := 1
BEGIN
  FOR i FROM 1 TO AnzIndividuen
  REPEAT
    IF Qualitaet[i] > Qualitaet[ID] DO
      Position := Position + 1;
    END
  END_LOOP
  RETURN (Position);
END_FUNC

#-------------------------------------------------------------------------
# BestimmeGruppenposition()
#
# Typ:             Funktion
# Aufrufparameter: INTEGER ID, LOGICAL GruppenagentAktiv, ARRAY QualitaetG
# Rückgabewert:    INTEGER
#
# Beschreibung: Diese Funktion bestimmt für diese Gruppe die Position, die
#               sie aufgrund ihrer Qualitaet im Gesamtsystem einnimmt
#-------------------------------------------------------------------------
FUNCTION BestimmeGruppenposition (INTEGER: ID, LOGICAL: GruppenagentAktiv,
                                  ARRAY [n] REAL[GQ]: Qualitaet  --> INTEGER)
DECLARE
  Position (INTEGER)  := 1
BEGIN
  IF GruppenagentAktiv DO
    #-------------------------
    # Die Gruppe exisitert ...
    #-------------------------
    FOR i FROM 1 TO AnzGruppenagenten
      REPEAT
        IF Qualitaet[i] > Qualitaet[ID] DO
        Position := Position + 1;
      END
    END_LOOP
```

```
    RETURN (Position);
  END
  ELSE DO
    RETURN(0);
  END
END_FUNC

#-------------------------------------------------------------------------
# WissenIndividuen()
#
# Typ:              Funktion
# Aufrufparameter: ARRAY Wissen
# Rückgabewert:    REAL[WI]
#
# Beschreibung: Diese Funktion liefert das durchschnittliche Wissen aller
#               Individuen zurück
#-------------------------------------------------------------------------
FUNCTION WissenIndividuen (ARRAY [n] REAL[WI]: Wissen --> REAL[WI])
DECLARE
  SummeWissen (REAL[WI])  := 0 [WI]
BEGIN
  FOR i FROM 1 TO AnzIndividuen
  REPEAT
    SummeWissen   := SummeWissen + Wissen[i];
  END_LOOP
  RETURN (SummeWissen / AnzIndividuen);
END_FUNC

#-------------------------------------------------------------------------
# SozKompetenzIndividuen()
#
# Typ:              Funktion
# Aufrufparameter: ARRAY SozKompetenz
# Rückgabewert:    REAL[SK]
#
# Beschreibung: Diese Funktion liefert die durchschnittliche soziale
#               Kompetenz aller Inividuen zurück
#-------------------------------------------------------------------------
FUNCTION SozKompetenzIndividuen (ARRAY [n] REAL[SK]: SozKompetenz
                            --> REAL[SK])
DECLARE
  SummeSozKompetenz (REAL[SK])  := 0 [SK]
BEGIN
  FOR i FROM 1 TO AnzIndividuen
  REPEAT
    SummeSozKompetenz   := SummeSozKompetenz + SozKompetenz[i];
  END_LOOP
  RETURN (SummeSozKompetenz / AnzIndividuen);
END_FUNC


#-------------------------------------------------------------------------
# BestimmeDurchschnSozAkt()
#
# Typ:              Funktion
# Aufrufparameter: ARRAY SozAkt
# Rückgabewert:    REAL[SoZ]
#
# Beschreibung: Diese Funktion liefert die durchschnittliche soziale
#               Zufriedenheit aller Inividuen zurück
#-------------------------------------------------------------------------
FUNCTION BestimmeDurchschnSozAkt (ARRAY [n] REAL[SoZ]: SozAkt --> REAL[SoZ])
DECLARE
  SummeSozAkt (REAL[SoZ])  := 0 [SoZ]
BEGIN
  FOR i FROM 1 TO AnzIndividuen
  REPEAT
```

```
    SummeSozAkt   := SummeSozAkt + SozAkt[i];
  END_LOOP
  RETURN (SummeSozAkt / AnzIndividuen);
END_FUNC

#-------------------------------------------------------------------------
# BestimmeRelWisGMax()
#
# Typ:              Funktion
# Aufrufparameter: ARRAY RelWissenG
# Rückgabewert:    REAL
#
# Beschreibung: Diese Funktion liefert den höchsten Wert RelWissenG aller
#               Gruppen(agenten)
#-------------------------------------------------------------------------
FUNCTION BestimmeRelWisGMax (ARRAY [n] REAL: RelWisG --> REAL)
DECLARE
  RelWisGMax (REAL)  := 0
BEGIN
  FOR i FROM 1 TO AnzGruppenagenten
  REPEAT
    IF RelWisG[i] > RelWisGMax DO
      RelWisGMax := RelWisG[i];
    END
  END_LOOP
  RETURN (RelWisGMax);
END_FUNC

#-------------------------------------------------------------------------
# BestimmeRelSozGMax()
#
# Typ:              Funktion
# Aufrufparameter: ARRAY RelSozG
# Rückgabewert:    REAL
#
# Beschreibung: Diese Funktion liefert den höchsten Wert RelSozG aller
#               Gruppen(agenten)
#-------------------------------------------------------------------------
FUNCTION BestimmeRelSozGMax (ARRAY [n] REAL: RelSozG --> REAL)
DECLARE
  RelSozGMax (REAL)  := 0
BEGIN
  FOR i FROM 1 TO AnzGruppenagenten
  REPEAT
    IF RelSozG[i] > RelSozGMax DO
      RelSozGMax := RelSozG[i];
    END
  END_LOOP
  RETURN (RelSozGMax);
END_FUNC

#-------------------------------------------------------------------------
# IndividuenInGruppe()
#
# Typ:              Funktion
# Aufrufparameter: ARRAY GroesseG
# Rückgabewert:    INTEGER
#
# Beschreibung: Diese Funktion liefert die Anzahl der Individuen, die in
#               einer echten Gruppe (mindestens zwei Mitglieder) sind
#-------------------------------------------------------------------------
FUNCTION IndividuenInGruppe (ARRAY [n] INTEGER: GroesseG --> INTEGER)
DECLARE
  AnzahlIndividuenInGruppe (INTEGER)  := 0
BEGIN
  FOR i FROM 1 TO AnzGruppenagenten
  REPEAT
```

```
    IF GroesseG[i] >= 2 DO
      AnzahlIndividuenInGruppe := AnzahlIndividuenInGruppe + GroesseG[i];
    END
  END_LOOP
  RETURN (AnzahlIndividuenInGruppe);
END_FUNC

#-------------------------------------------------------------------------
# BestimmeDurchnZeitInGruppe()
#
# Typ:             Funktion
# Aufrufparameter: ARRAY TInGruppe
# Rückgabewert:    REAL[h]
#
# Beschreibung: Diese Funktion liefert die Zeit, die die Individuen in
#               einer echten Gruppe (mindestens zwei Mitglieder) waren
#-------------------------------------------------------------------------
FUNCTION BestimmeDurchschnZeitInGruppe (ARRAY [n] REAL[h]: TInGruppe
                                        --> REAL[h])
DECLARE
  GesamtZeitInGruppe (REAL[h])  := 0 [h]
BEGIN
  FOR i FROM 1 TO AnzIndividuen
  REPEAT
    GesamtZeitInGruppe := GesamtZeitInGruppe + TInGruppe[i];
  END_LOOP
  RETURN (GesamtZeitInGruppe/AnzIndividuen);
END_FUNC

#-------------------------------------------------------------------------
# BestimmeDurchschnGruppengroesse()
#
# Typ:             Funktion
# Aufrufparameter: ARRAY GroesseG
# Rückgabewert:    REAL
#
# Beschreibung: Diese Funktion liefert die durchschnittliche Gruppengroesse
#               zurück
#-------------------------------------------------------------------------
FUNCTION BestimmeDurchschnGruppengroesse (ARRAY [n] INTEGER: GroesseG
                                          --> REAL)
DECLARE
  AnzahlIndividuenInGruppe (INTEGER)  := 0 ,
  AnzahlGruppen (INTEGER)
BEGIN
  FOR i FROM 1 TO AnzGruppenagenten
  REPEAT
    IF GroesseG[i] >= 1 DO
      AnzahlIndividuenInGruppe := AnzahlIndividuenInGruppe + GroesseG[i];
      AnzahlGruppen            := AnzahlGruppen + 1;
    END
  END_LOOP
  RETURN (AnzahlIndividuenInGruppe/AnzahlGruppen);
END_FUNC

#-------------------------------------------------------------------------
# BestimmeWissenMax()
#
# Typ:             Funktion
# Aufrufparameter: ARRAY Wissen
# Rückgabewert:    REAL[WI]
#
# Beschreibung: Diese Funktion liefert den zur Zeit hoechsten Wert an Wissen
#               (eines Individuums) zurück
#-------------------------------------------------------------------------
FUNCTION BestimmeWissenMax (ARRAY [n] REAL[WI]: Wissen --> REAL[WI])
DECLARE
```

```
    WissenMax (REAL[WI]) := 0 [WI]
  BEGIN
    FOR i FROM 1 TO AnzIndividuen
    REPEAT
      IF Wissen[i] > WissenMax DO
        WissenMax := Wissen[i];
      END
    END_LOOP
    RETURN (WissenMax);
  END_FUNC

  #-------------------------------------------------------------------------
  # BestimmeSozKompetenzMax()
  #
  # Typ:              Funktion
  # Aufrufparameter: ARRAY SozKompetenz
  # Rückgabewert:    REAL[SK]
  #
  # Beschreibung: Diese Funktion liefert den zur Zeit hoechsten Wert
  #               Sozialkompetenz (eines Individuums) zurück
  #-------------------------------------------------------------------------
  FUNCTION BestimmeSozKompetenzMax (ARRAY [n] REAL[SK]: SozKompetenz
                                    --> REAL[SK])
  DECLARE
    SozKompetenzMax (REAL[SK]) := 0 [SK]
  BEGIN
    FOR i FROM 1 TO AnzIndividuen
    REPEAT
      IF SozKompetenz[i] > SozKompetenzMax DO
        SozKompetenzMax := SozKompetenz[i];
      END
    END_LOOP
    RETURN (SozKompetenzMax);
  END_FUNC

DECLARATION OF ELEMENTS

CONSTANTS
  #-------------------------------------------------
  # Zeitraster für die Berechnung der Gruppenpositionen
  #-------------------------------------------------
  TRaster (REAL[h]) := 0.1 [h]

STATE VARIABLES
  #-----------------------------------------------
  # Die zehn besten Individuen (ID)
  #-----------------------------------------------
  ARRAY [10] Top10I (INTEGER) := 0,

  #-----------------------------------------
  # Die zehn besten Gruppen (ID)
  #-----------------------------------------
  ARRAY [10] Top10G (INTEGER) := 0,

  #---------------------------------------
  # Zeitpunkt für Aktualisierung der Top10
  #---------------------------------------
  TTop10Aktualisieren (REAL[h]) := 0.1 [h]

DEPENDENT VARIABLES
DISCRETE
  #---------------------------------------------------------
  # Anzahl der Individuen, die in einer echten Gruppe sind
  #---------------------------------------------------------
  AnzahlIndividuenInGruppe (INTEGER),
```

```
  #-------------------------------------------------------------
  # Anzahl der Individuen, die nicht in einer echten Gruppe sind
  #-------------------------------------------------------------
  AnzahlIndividuenAllein (INTEGER),

  #--------------------
  # Anzahl der Gruppen
  #--------------------
  AnzahlGruppen (INTEGER) := 0,

  #---------------------------
  # Positionen der Individuen
  #---------------------------
  ARRAY [AnzIndividuen] SozPosition (INTEGER) := 0,

  #------------------------
  # Positionen der Gruppen
  #------------------------
  ARRAY [AnzGruppenagenten] Gruppenposition (INTEGER) := 0,

  #--------------------------------
  # Durchschnittliche Gruppengröße
  #--------------------------------
  DurchschnGruppengroesse (REAL),

  #----------------------
  # Maximum Gesamtwissen
  #----------------------
  WissenMax (REAL[WI]),

  #-------------------------
  # Maximum Sozialkompetenz
  #-------------------------
  SozKompetenzMax (REAL[SK])

CONTINUOUS
  #-------------------------------
  # Informationen über Individuen
  #-------------------------------
  DurchschnWissen       (REAL[WI]),
  DurchschnSozKompetenz (REAL[SK]),
  DurchschnSozAkt       (REAL[SoZ]),

  #-----------------------------------------------------------
  # Zeit, die die Individuen durchschnittlich allein waren
  #-----------------------------------------------------------
  DurchschnZeitAllein (REAL[h]) := 0 [h],

  #-------------------------------------------------------------------------
  # Zeit, die die Individuen durchschnittlich in einer echten Gruppe waren
  #-------------------------------------------------------------------------
  DurchschnZeitInGruppe (REAL[h]) := 0 [h],
  #-------------------------------------
  # Informationen über Gruppen(agenten)
  #-------------------------------------
  RelWisGMax (REAL),
  RelSozGMax (REAL)

SENSOR VARIABLES
DISCRETE
  #-------------------------------------
  # Informationen über Gruppen(agenten)
  #-------------------------------------
  ARRAY [AnzGruppenagenten] GruppenagentAktiv (LOGICAL),
  ARRAY [AnzGruppenagenten] GroesseG          (INTEGER),
```

```
  #-------------------------------
  # Informationen über Individuen
  #-------------------------------
  ARRAY [AnzIndividuen] Gruppenmitglied (LOGICAL),
  ARRAY [AnzIndividuen] GruppenID       (INTEGER),

  Protokoll (LOGICAL)

CONTINUOUS
  #-----------------------------------
  # Informationen über Gruppen(agenten)
  #-----------------------------------
  ARRAY [AnzGruppenagenten] RelWisG    (REAL),
  ARRAY [AnzGruppenagenten] RelSozG    (REAL),
  ARRAY [AnzGruppenagenten] QualitaetG (REAL[GQ]),

  #-------------------------------
  # Informationen über Individuen
  #-------------------------------
  ARRAY [AnzIndividuen] Wissen       (REAL[WI]),
  ARRAY [AnzIndividuen] SozAkt       (REAL[SoZ]),
  ARRAY [AnzIndividuen] SozKompetenz (REAL[SK]),
  ARRAY [AnzIndividuen] QualitaetI   (REAL[IndQ]),
  ARRAY [AnzIndividuen] TInGruppe    (REAL[h])

TRANSITION INDICATORS
  #-------------------------------
  # Aktualisiern der Bestenlisten
  #-------------------------------
  Top10Aktualisieren

LOCATIONS
  RangfolgeI (Indivi ORDERED BY DEC Qualitaet) := 36 Indivi,
  RangfolgeG (Gruppe ORDERED BY DEC Qualitaet) :=  0 Gruppe

DYNAMIC BEHAVIOUR

  #--------------------
  # Anzahl der Gruppen
  #--------------------
  AnzahlGruppen := BestimmeAnzahlGruppen(ARRAY GruppenagentAktiv);

  #---------------------------
  # Positionen der Individuen
  #---------------------------
  SozPosition{ALL i} := BestimmeSozPosition(i, ARRAY QualitaetI);

  #------------------------
  # Positionen der Gruppen
  #------------------------
  Gruppenposition{ALL i} := BestimmeGruppenposition(i, GruppenagentAktiv[i],
                                                    ARRAY QualitaetG);

  #-----------------------------------------
  # Durchschnittliches Wissen aller Individuen
  #-----------------------------------------
  DurchschnWissen := WissenIndividuen(ARRAY Wissen);

  #-------------------------------------------------------
  # Durchschnittliche soziale Kompetenz aller Individuen
  #-------------------------------------------------------
  DurchschnSozKompetenz := SozKompetenzIndividuen(ARRAY SozKompetenz);

  #---------------------------------------------------------
  # Durchschnittliche soziale Zufriedenheit aller Individuen
  #---------------------------------------------------------
  DurchschnSozAkt := BestimmeDurchschnSozAkt(ARRAY SozAkt);
```

```
#-----------------------------------------------------------
# Anzahl der Individuen, die in einer !echten! Gruppe sind
#-----------------------------------------------------------
AnzahlIndividuenInGruppe := IndividuenInGruppe(ARRAY GroesseG);

#-----------------------------------------------------------
# Anzahl der Individuen, die allein sind
#-----------------------------------------------------------
AnzahlIndividuenAllein := AnzIndividuen - AnzahlIndividuenInGruppe;

#-------------------------------------------------------------------------
# Zeit, die die Individuen durchschnittlich in einer echten Gruppe waren
#-------------------------------------------------------------------------
DurchschnZeitInGruppe := BestimmeDurchschnZeitInGruppe(ARRAY TInGruppe);

#-----------------------------------------------------------
# Zeit, die die Individuen durchschnittlich allein waren
#-----------------------------------------------------------
DurchschnZeitAllein := T - DurchschnZeitInGruppe;

#-------------------------------------------------------------------
# Durchschnittliche Gruppengroesse (auch Gruppen mit einem Mitglied
# werden berücksichtigt)
#-------------------------------------------------------------------
DurchschnGruppengroesse := BestimmeDurchschnGruppengroesse(ARRAY GroesseG);

#-----------------------------------------
# Maximum von RelWisG aller Gruppen(agenten)
#-----------------------------------------
RelWisGMax := BestimmeRelWisGMax(ARRAY RelWisG);

#-----------------------------------------
# Maximum von RelSozG aller Gruppen(agenten)
#-----------------------------------------
RelSozGMax := BestimmeRelSozGMax(ARRAY RelSozG);

#-------------------------------------
# Maximum von Wissen aller Individuen
#-------------------------------------
WissenMax := BestimmeWissenMax(ARRAY Wissen);

#-----------------------------------------
# Maximum von SozKompetenz aller Individuen
#-----------------------------------------
SozKompetenzMax := BestimmeSozKompetenzMax(ARRAY SozKompetenz);

ON START DO
  RangfolgeI^ : FROM RangfolgeI GET Indivi{ALL i}
                CHANGING
                  ID^ := i;
                END
END
#-------------------------------------------------------------------------------
#
# Ereignis 1 : Eine neue Gruppe ist hinzugekommen
#
#-------------------------------------------------------------------------------
ON ^AnzahlGruppen > NUMBER(RangfolgeG)^ DO
  RangfolgeG^ : ADD 1 NEW Gruppe
                CHANGING
                  ID^ := AnzahlGruppen;
                END
  IF Protokoll DO
    DISPLAY("T = %5.2f, Statistik, Ereignis 1\n", T);
    DISPLAY("Eine neue Gruppe ist hinzugekommen\n");
  END
END
```

```
#------------------------------------------------------------------------------
#
# Ereignis 2 : Bestimmen der Bestenliste anstoßen
#
#------------------------------------------------------------------------------

WHENEVER T >= TTop10Aktualisieren AND AnzahlGruppen = NUMBER(RangfolgeG) DO
  TTop10Aktualisieren^ := T + TRaster;

  RangfolgeI^ : FROM RangfolgeI GET Indivi{ALL i}
                CHANGING
                  Qualitaet^ := QualitaetI[RangfolgeI:Indivi[i].ID];
                END

  RangfolgeG^ : FROM RangfolgeG GET Gruppe{ALL i}
                CHANGING
                  Qualitaet^ := QualitaetG[RangfolgeG:Gruppe[i].ID];
                END
  SIGNAL Top10Aktualisieren;
END

#------------------------------------------------------------------------------
#
# Ereignis 3 : Aktualisieren der Bestenlisten
#              Die Variablen Top10I[], Top10G[] sind nur für die Animation
#              nötig und stellen eigentlich keine neue Information dar.
#              (Die Information steckt ja schon in den mobilen Komponenten
#              bzw. den Locations; auf diese kann man jedoch vorläufig nicht
#              in der Animation zugreifen)
#
#------------------------------------------------------------------------------
ON Top10Aktualisieren DO
  Top10I{ALL i | i <= 10}^            := RangfolgeI:Indivi[i].ID;
  Top10G{ALL i | i <= AnzahlGruppen}^ := RangfolgeG:Gruppe[i].ID;
  Top10G{ALL i | i > AnzahlGruppen}^  := 0;
END

#------------------------------------------------------------------------------
#
# Ereignis 4 : Eine Gruppe hat sich aufgelöst
#
#------------------------------------------------------------------------------
ON ^NUMBER(RangfolgeG) > AnzahlGruppen^ DO
  RangfolgeG^ : REMOVE Gruppe{ALL i |
                              QualitaetG[RangfolgeG:Gruppe[i].ID] <= 0 [GQ]};
  IF Protokoll DO
    DISPLAY("T = %5.2f, Statistik, Ereignis 4\n", T);
    DISPLAY("Entfernen einer (aufgelösten) Gruppen aus der Rangfolge\n");
  END
END

ON ^T >= 500 [h]^ DECLARE
  WissenMin (REAL[WI])       := 1000000 [WI],
  SozKompetenzMin (REAL[SK]) := 1000000 [SK]
DO PROCEDURE
  FOR i FROM 1 TO AnzIndividuen
  REPEAT
    IF Wissen[i] < WissenMin DO
      WissenMin := Wissen[i];
    END
    IF SozKompetenz[i] < SozKompetenzMin DO
      SozKompetenzMin := SozKompetenz[i];
    END
   DISPLAY("Individuum %d: SozPosition = %d\n", i, SozPosition[i]);
  END_LOOP
END
DO TRANSITIONS
```

```
    DISPLAY("WissenMin: %f, WissenMax: %f, DurchschnWissen: %f\n",
             WissenMin, WissenMax, DurchschnWissen);
    DISPLAY("SozKompetenzMin: %f, SozKompetenzMax: %f, AvgSozKompetenz: %f\n",
             SozKompetenzMin, SozKompetenzMax, DurchschnSozKompetenz);
  END

END OF Statistik
```

C.6 Mobile Komponenten

C.6.1 Die mobile Komponente *AusfuehrungsAOI*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    AusfuehrungsAOI
#
#  Art:     Mobile Komponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Mobile Komponente zur Weitergabe einer geplanten Aktion vom Verhalten
#      an den Aktor (Ausfuehrungsanordnung) beim Individuum
#
#    - Eine Ausführungsanordnung setzt sich zusammen aus Typ und Parameter
#
#
#    - Verbindung(en) mit anderen Komponenten: -
#
#-------------------------------------------------------------------------------

MOBILE COMPONENT AusfuehrungsAOI

USE OF UNITS
  UNIT [WI] =  BASIS
  UNIT [SK] = BASIS

LOCAL DEFINITIONS

VALUE SET
  #-----------------------------------------------------------------------------
  # Ausführungsanordnungs-Typen:
  # - Anfrage_verschicken:   Mitteilung an Gruppenagenten schicken (Anfrage)
  # - Bewerbung_verschicken: Mitteilung an Gruppenagent schicken (Bewerbung)
  # - Gruppe_verlassen:      Mitteilung an Gruppenagent schicken (Ausstieg)
  #-----------------------------------------------------------------------------
  AAOTyp : ('Anfrage_verschicken', 'Bewerbung_verschicken', 'Gruppe_verlassen')

DECLARATION OF ELEMENTS

STATE VARIABLES
DISCRETE
  #-------------------------------------------
  # Typ der Ausfuehrungsanordnung (Individuum)
  #-------------------------------------------
  Typ (AAOTyp) := 'Bewerbung_verschicken',
```

```
  #-----------
  # Parameter
  #-----------
  ID           (INTEGER)  := 0,
  Wissen       (REAL[WI]) := 0 [WI],
  SozKompetenz (REAL[SK]) := 0 [SK]

END OF AusfuehrungsAOI
```

C.6.2 Die mobile Komponente *AusfuehrungsAOG*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    AusfuehrungsAOG
#
#  Art:     Mobile Komponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Mobile Komponente zur Weitergabe einer geplanten Aktion vom Verhalten
#      an den Aktor (Ausfuehrungsanordnung) beim Gruppenagent
#
#    - Eine Ausführungsanordnung setzt sich zusammen aus Typ und Parametern
#
#    - Verbindung(en) mit anderen Komponenten: -
#
#-------------------------------------------------------------------------------

MOBILE COMPONENT AusfuehrungsAOG

USE OF UNITS
  UNIT [GQ]  = BASIS

LOCAL DEFINITIONS

VALUE SET
  #-----------------------------------------------------------------------------
  # Ausführungsanordnungs-Typen:
  # - Auskunft_verschicken:     Mitteilung an Individuum schicken (Auskunft)
  # - Zusage_verschicken:       Mitteilung an Individuum schicken (Zusage)
  # - Absage_verschicken:       Mitteilung an Individuum schicken (Absage)
  # - Mitglied_eliminieren:     Mitteilung an Individuum schicken (Ausschluss)
  # - Update_verschicken:       Mitteilung an Individuum schicken (Update)
  #-----------------------------------------------------------------------------
  AAOTyp : ('Auskunft_verschicken', 'Zusage_verschicken', 'Absage_verschicken',
            'Mitglied_eliminieren', 'Update_verschicken')
DECLARATION OF ELEMENTS

STATE VARIABLES
DISCRETE
  #--------------------------------------------
  # Typ der Ausführungsanordnung (Gruppenagent)
  #--------------------------------------------
  Typ (AAOTyp) := 'Zusage_verschicken',

  #-----------
  # Parameter
  #-----------
```

```
  #--------------------
  # ID des Empfaengers
  #--------------------
  ID         (INTEGER)  := 0,
  QualitaetG (REAL[GQ]) := 0 [GQ],
  GroesseG   (INTEGER)  := 0

END OF AusfuehrungsAOG
```

C.6.3 Die mobile Komponente *Nachricht*

```
#-------------------------------------------------------------------------------
#
#  Projekt: Modell Lerngruppe
#
#  Name:    Nachricht
#
#  Art:     Mobile Komponente
#
#  Version: 0
#
#  Beschreibung:
#
#    - Mobile Komponente zur Kommunikation zwischen Individuen und Gruppen-
#      agenten
#
#    - Eine Nachricht besteht aus einem Kopf- und einem Datenteil
#       + Nachrichtenkopf
#         - Sender:       ID des Absenders
#         - Empfaenger:   ID des Empfaengers
#         - Typ:          Typ der Nachricht (Bewerbung, Absage, ...)
#       + Datenteil
#         - Wissen:       Wissen des Individuums (Bewerbung)
#         - SozKompetenz: Soziale Kompetenz des Individuums (Bewerbung)
#         - QualitaetG:   Qualitaet der Gruppe
#         - GroesseG:     Groesse der Gruppe (Auskunft, Update)
#
#
#  Verbindung(en) mit anderen Komponenten: -
#
#  Verwendet in den Komponenten:
#    - Connector
#    - SensorI, WahrnehmungI, KognitionI, AktorI
#    - SensorG, WahrnehmungG, KognitionG, AktorG
#
#-------------------------------------------------------------------------------

MOBILE COMPONENT Nachricht

USE OF UNITS
  UNIT [WI]  = BASIS
  UNIT [SK] = BASIS
  UNIT [GQ]  = BASIS

LOCAL DEFINITIONS

VALUE SET
  #-----------------------------------------------------------------------------
  # Nachrichtentypen:
  #  - Individuum an Gruppenagent
  #     + Anfrage:       Individuum fragt Gruppenagent nach Gruppeninformation
  #     + Bewerbung:     Individuum bewirbt sich um Aufnahme in die Gruppe
  #     + Ausstieg:      Individuum verlaesst die Gruppe
  #
```

```
  # - Gruppenagent an Individuum
  #     + Auskunft:      Gruppenagent antwortet auf Anfrage des Individuums
  #     + Absage:        Individuum wird nicht aufgenommen
  #     + Zusage:        Individuum wird in die Gruppe aufgenommen
  #     + Ausschluss:    Grupppenagent schliesst Individuum aus der Gruppe aus
  #     + Update:        Aenderung der diskreten, für die Gruppenmitglieder
  #                      relevanten Informationen (GroesseG)
  #-------------------------------------------------------------------------
  Nachrichtentyp : ('Anfrage', 'Bewerbung', 'Ausstieg',
                    'Auskunft', 'Absage', 'Zusage', 'Ausschluss', 'Update')

DECLARATION OF ELEMENTS

STATE VARIABLES
DISCRETE
  #-----------------
  # Nachrichtenkopf
  #-----------------
  Sender     (INTEGER) := 0,
  Empfaenger (INTEGER) := 0,
  Typ        (Nachrichtentyp) := 'Bewerbung',

  #-----------------
  # Datenteil
  #-----------------
  Wissen       (REAL[WI]) := 0 [WI],
  SozKompetenz (REAL[SK]) := 0 [SK],

  QualitaetG   (REAL[GQ]) := 0 [GQ],
  GroesseG     (INTEGER)  := 0

END OF Nachricht
```

ANHANG D QUELLCODE DES MODELLS *ADAM*

D.1 Die Strukturkomponente *WorldOfAdam*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell ADAM
#
#  Name:             WorldOfAdam
#
#  Art:              Hoehere Komponente
#
#  Version:          1
#
#  Beschreibung:     Modellierung der gesamten Welt des Modells ADAM.
#
#                    Die Welt von Adam besteht aus der Basiskomponente
#                    Environment und der hoeheren Komponente Adam, in der der
#                    Agent Adam selbst modelliert ist.
#
#  Verbindung(en) mit anderen Komponenten:
#
#                    World of Adam ist die hoechste Komponente des Modells.
#                    Sie schafft nur Verbindungen zwischen Subkomponenten.
#
#-------------------------------------------------------------------------------

HIGH LEVEL COMPONENT WorldOfAdam

SUBCOMPONENTS
        Environment,
        Adam

COMPONENT CONNECTIONS

        Environment.InFalle      --> Adam.InFalle;
        Environment.Feldart {ALL i}{ALL j}
                                 --> Adam.Feldart[i][j];
        Environment.Nahrungsmenge {ALL i}{ALL j}
                                 --> Adam.Nahrungsmenge[i][j];
        Environment.xAdam        --> Adam.xAdam;
        Environment.yAdam        --> Adam.yAdam;
        Adam.DatAnENV            --> Environment.DatVonACT;

        Environment.Protokoll   --> Adam.Protokoll;

#-------------------------------------------------------------------------------
# Erzeugen eines Standard-Szenarios:
#-------------------------------------------------------------------------------
INITIALIZE

   #----------------------------------------------------------------------------
   # Bei der Initialisierung wird Adam auf das Feld (1, 1) gesetzt.
   #----------------------------------------------------------------------------
        Environment.xAdam  := 1;
        Environment.yAdam  := 1;

   #----------------------------------------------------------------------------
   # Generieren von Nahrungsfeldern:
   #----------------------------------------------------------------------------
        Environment.Feldart[9][10]       := 'Nahrung';
        Environment.Nahrungsmenge[9][10] := 20.0 [EnE];
        Environment.Feldart[5][5]        := 'Nahrung';
        Environment.Nahrungsmenge[5][5]  := 20.0 [EnE];
        Environment.Feldart[5][12]       := 'Nahrung';
        Environment.Nahrungsmenge[5][12] := 20.0 [EnE];
        Environment.Feldart[1][7]        := 'Nahrung';
```

```
        Environment.Nahrungsmenge[1][7]  := 20.0 [EnE];
        Environment.Feldart[8][4]        := 'Nahrung';
        Environment.Nahrungsmenge[8][4]  := 20.0 [EnE];
        Environment.Feldart[11][3]       := 'Nahrung';
        Environment.Nahrungsmenge[11][3] := 20.0 [EnE];
        Environment.Feldart[3][3]        := 'Nahrung';
        Environment.Nahrungsmenge[3][3]  := 20.0 [EnE];

   #--------------------------------------------------------------------------
   # Generieren von Gefahrenfeldern:
   #--------------------------------------------------------------------------
        Environment.Feldart[6][6]        := 'Gefahr';
        Environment.Feldart[2][5]        := 'Gefahr';
        Environment.Feldart[9][1]        := 'Gefahr';
        Environment.Feldart[7][11]       := 'Gefahr';
        Environment.Feldart[1][8]        := 'Gefahr';
        Environment.Feldart[3][9]        := 'Gefahr';

END OF WorldOfAdam
```

D.2 Die Komponente *Environment*

```
#------------------------------------------------------------------------------
#
#  Projekt:          Modell ADAM
#
#  Name:             Environment
#
#  Art:              Basiskomponente
#
#  Version:          1
#
#  Beschreibung:     Modellierung der Umwelt von Adam
#
#                    In der Komponente Environment werden alle Variablen
#                    verwaltet, die den Zustand der Umwelt von Adam
#                    beschreiben.
#
#                    Dynamik der Umwelt:
#
#                    - staendige Zunahme der Nahrung auf den Nahrungsfeldern
#
#                    - Veraenderung der Position von Adam durch die Aktionen
#                      'Gehen' und 'Befreien'
#
#                    - Abnahme der Nahrungsmenge auf einem Nahrungsfeld nach
#                      der Aktion 'Essen'
#
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Adam (Actor) an Environment
#       + DatVonACT
#
#    - Environment an Adam (Sensor)
#       + xAdam
#       + yAdam
#       + InFalle
#       + Nahrungsmenge
#       + Feldart
#
#------------------------------------------------------------------------------
```

```
BASIC COMPONENT Environment

MOBILE SUBCOMPONENTS OF CLASS DatenTransfer

USE OF UNITS

   #-------------------------------------------------------------------------
   # Energie-Einheit: Auch die Nahrungsmenge auf den Nahrungsfeldern wird mit
   # Energie-Einheiten gemessen.
   #-------------------------------------------------------------------------
   UNIT [EnE]  = BASIS

   #-------------------------------------------------------------------------
   # Als Zeiteinheit wird eine "Stunde" festgelegt.
   #-------------------------------------------------------------------------
   TIMEUNIT    = [h]

LOCAL DEFINITIONS

 DIMENSIONS

   #-------------------------------------------------------------------------
   # Groesse des quadratischen Spielfeldes in x- und y-Richtung: XYMAX
   #-------------------------------------------------------------------------
   XYMAX := 12

   #-------------------------------------------------------------------------
   # Die moeglichen Arten der Felder des Spielfeldes:
   #  - Neutral:     bekanntes neutrales Feld
   #  - Nahrung:     bekanntes Nahrungsfeld
   #  - Gefahr:      bekanntes Gefahrenfeld
   #  - Unbekannt:   unbekanntes Feld
   #-------------------------------------------------------------------------
   VALUE SET FELDARTEN : ('Neutral', 'Nahrung', 'Gefahr', 'Unbekannt')

DECLARATION OF ELEMENTS

 CONSTANTS

   #-------------------------------------------------------------------------
   # Faktor fuer das logistische Wachstum der Nahrungsmenge auf den
   # Nahrungsfeldern
   #-------------------------------------------------------------------------
   NahrungsFaktor               (REAL [1/h])   :=   0.15 [1/h] ,

   #-------------------------------------------------------------------------
   # Minimum an Nahrung, das immer auf einer Nahrungsstelle bleibt, auch nach
   # dem Essen.
   # Muss kleiner sein als die Wahrnehmungsschranke fuer Nahrung in der
   # Perception!!! (Sonst funktioniert der ganze Lernvorgang nicht richtig)
   #-------------------------------------------------------------------------
   NahrungsMin                  (REAL [EnE])   :=   0.05 [EnE] ,

   #-------------------------------------------------------------------------
   # Maximale Nahrungsmenge, die auf einem Nahrungsfeld erreicht werden kann.
   #-------------------------------------------------------------------------
   NahrungsMax                  (REAL [EnE])   := 100.0 [EnE]

 STATE VARIABLES

  DISCRETE

   #-------------------------------------------------------------------------
   # Feldart von jedem Feld
   #-------------------------------------------------------------------------
   ARRAY [XYMAX][XYMAX] Feldart  (FELDARTEN)    := 'Neutral',
```

```
   #-------------------------------------------------------------------------
   # x-Koordinate des Feldes, auf dem sich Adam gerade befindet.
   #-------------------------------------------------------------------------
   xAdam                         (INTEGER)      := 5,

   #-------------------------------------------------------------------------
   # y-Koordinate des Feldes, auf dem sich Adam gerade befindet.
   #-------------------------------------------------------------------------
   yAdam                         (INTEGER)      := 5,

   #-------------------------------------------------------------------------
   # gesetzt (TRUE), falls sich Adam gerade in einer Falle befindet.
   # Es wird also unterschieden, ob sich Adam auf einem Gefahrenfeld innerhalb
   # oder ausserhalb eines Gefahrenfeldes befindet.
   #-------------------------------------------------------------------------
   InFalle                       (LOGICAL)      := FALSE,

   #-------------------------------------------------------------------------
   # Protokoll = TRUE: Textausgabe am Bildschirm aktiviert
   #-------------------------------------------------------------------------
   Protokoll                     (LOGICAL)      := FALSE

  CONTINUOUS

   #-------------------------------------------------------------------------
   # Nahrungsmenge der einzelnen Felder
   # (ist natuerlich nur auf Nahrungsfeldern > 0)
   #-------------------------------------------------------------------------
   ARRAY [XYMAX][XYMAX] Nahrungsmenge (REAL [EnE]) := 0.0 [EnE]

 TRANSITION INDICATOR

   #-------------------------------------------------------------------------
   # Triggert das Vernichten einer Nachricht auf der Location DatVonACT
   #-------------------------------------------------------------------------
   Vernichten

 LOCATION

   #-------------------------------------------------------------------------
   # Hilfs-Location: benoetigt fuer das Vernichten einer Nachricht von
   # der Location DatVonACT
   #-------------------------------------------------------------------------
   HilfsLoc                      (DatenTransfer)   := 0 DatenTransfer

 SENSOR LOCATION

   #-------------------------------------------------------------------------
   # Hier kommen die Nachrichten von Adam (genauer: von der Komponente Actor)
   # an
   #-------------------------------------------------------------------------
   DatVonACT                     (DatenTransfer)   := 0 DatenTransfer

DYNAMIC BEHAVIOUR

   #-------------------------------------------------------------------------
   #
   # Differentialgleichung 1: "Nachwachsen von Nahrung"
   #
   #     Die Nahrung waechst auf allen Nahrungsstellen gemaess der logistischen
   #     Wachstumskurve nach (wenn auf einem Feld gar keine Nahrung ist,
   #     waechst auch nichts nach, z.B. auf neutralen Feldern oder
   #     Gefahrenstellen):
   #
   #-------------------------------------------------------------------------

   DIFFERENTIAL EQUATION
```

```
   Nahrungsmenge {ALL i}{ALL j}' := NahrungsFaktor * Nahrungsmenge [i][j] *
                        (NahrungsMax - Nahrungsmenge[i][j]) / NahrungsMax;
END

#-------------------------------------------------------------------------
#
# Ereignis 1: "Adam fuehrt eine externe Aktion aus."
#
#     Nach der Ausfuehrung einer externen Aktion durch Adam zieht die
#     Komponente Environment die Konsequenzen und veraendert die Umwelt
#     entsprechend der Aktion.
#
#     - Aktion 'Gehen': Veraendern der Position von Adam. Falls Adam auf
#                       eine Gefahrenstelle geht, setzen der Variable
#                       "InFalle"
#     - Aktion 'Befreien': Setzen der Variable "InFalle" auf FALSE
#     - Aktion 'Essen': Setzen der Nahrungsmenge des entsprechenden Feldes
#                       auf NahrungsMin
#
#-------------------------------------------------------------------------

WHENEVER (NUMBER (DatVonACT) > 0 )
DO

   #----------------------------------------------------------------------
   # Die ausgefuehrte Aktion ist 'Gehen'
   #----------------------------------------------------------------------
   IF (DatVonACT:DatenTransfer[1].Aktion = 'Gehen')
   DO

      #-------------------------------------------------------------------
      # Die Koordinaten von Adams Standpunkt veraendern sich entsprechend.
      #
      # Achtung: Hier findet keine Ueberpruefung statt, ob die Aenderung
      # rechtens ist. D. h., es waere hier ohne Fehlermeldung moeglich
      # (bei inkorrekter Modellierung z. B. der Komponente Behaviour),
      # dass Adam in einem Schritt eine unerlaubte Bewegung macht (z. B.
      # auf ein mehrere Schritte entferntes Feld oder auf ein Nachbarfeld,
      # obwohl er sich in der Falle befindet).
      #-------------------------------------------------------------------
      xAdam^ := DatVonACT:DatenTransfer[1].x;
      yAdam^ := DatVonACT:DatenTransfer[1].y;

      IF Protokoll
      DO
         DISPLAY ("\n T= %f: ENV: Gehen (%u,%u)\n", T,
                  DatVonACT:DatenTransfer[1].x,
                  DatVonACT:DatenTransfer[1].y);
      END

      #-------------------------------------------------------------------
      # Falls sich Adam auf eine Gefahrenstelle begeben hat , faellt er
      # immer in die Grube.
      #-------------------------------------------------------------------
      IF (Feldart[DatVonACT:DatenTransfer[1].x]
                 [DatVonACT:DatenTransfer[1].y] = 'Gefahr')
      DO

         #----------------------------------------------------------------
         # Adam faellt gerade in eine Fallgrube
         #----------------------------------------------------------------
         InFalle^ := TRUE;

         IF Protokoll
         DO
            DISPLAY ("\n T= %f: ENV: InFalle := TRUE\n", T);
         END
```

```
      END
   END

   #-------------------------------------------------------------------------
   # Die ausgefuehrte Aktion ist 'Befreien'
   #-------------------------------------------------------------------------
   ELSIF (DatVonACT:DatenTransfer[1].Aktion = 'Befreien')
   DO

      #-------------------------------------------------------------------
      # Adam befindet sich nach der Aktion 'Befreien' nicht mehr in der
      # Fallgrube.
      #
      # Achtung: Hier findet keine Ueberpruefung statt, ob sich Adam gerade
      # ueberhaupt in einer Falle befindet.
      #-------------------------------------------------------------------
      InFalle^ := FALSE;

      IF Protokoll
      DO
         DISPLAY ("\n T= %f: ENV: Befreien (%u,%u) ; InFalle := FALSE\n",
                  T, xAdam, yAdam);
      END

   END

   #-------------------------------------------------------------------------
   # Die ausgefuehrte Aktion ist 'Essen'
   #-------------------------------------------------------------------------
   ELSIF (DatVonACT:DatenTransfer[1].Aktion = 'Essen')
   DO

      #-------------------------------------------------------------------
      # Die Nahrungsmenge wird auf einen sehr kleinen Wert "NahrungsMin"
      # gesetzt.
      # (Nicht auf 0, da sich sonst die Nahrung nicht mehr vermehren
      # wuerde.)
      #
      # Achtung: Hier findet keine Ueberpruefung statt, ob sich Adam gerade
      # ueberhaut auf einem Nahrungsfeld befindet. Falls dies (z. B. wegen
      # fehlerhafter Komponente Behaviour) nicht der Fall ist, wuerde durch
      # die folgende Zeile ein neutrales Feld auf einmal Nahrung beinhalten.
      #-------------------------------------------------------------------
      Nahrungsmenge[xAdam][yAdam]^ :=  NahrungsMin;

      IF Protokoll
      DO
         DISPLAY ("\n T= %f: ENV: Essen (%u,%u)\n",
                  T, xAdam, yAdam);
      END

   END

   #-------------------------------------------------------------------------
   # Verschieben der mobilen Komponente DatenTransfer[1] von der SENSOR
   # Location "DatVonACT" auf die Location "HilfsLoc", damit sie
   # geloescht werden kann.
   #-------------------------------------------------------------------------
   HilfsLoc^: FROM DatVonACT GET DatenTransfer[1];

   #-------------------------------------------------------------------------
   # Signal zum Vernichten der mobilen Komponente geben.
   #-------------------------------------------------------------------------
   SIGNAL Vernichten;

END
```

```
   #-------------------------------------------------------------------------
   #
   # Ereignis 2: "Vernichten der Nachricht vom Actor"
   #
   #     Die bereits auf der Location "HilfsLoc" befindliche Nachricht von
   #     der Komponente Actor wird vernichtet.
   #
   #-------------------------------------------------------------------------

   ON Vernichten DO
      HilfsLoc^: REMOVE DatenTransfer{ALL};
   END

END OF Environment
```

D.3 Der Agent *Adam*

D.3.1 Die Strukturkomponente *Adam*

```
#-------------------------------------------------------------------------
#
#  Projekt:          Modell Adam
#
#  Name:             Adam
#
#  Art:              Hoehere Komponente
#
#  Version:          1
#
#  Beschreibung:     Modellierung des Agenten Adam.
#
#                    Die Ablaeufe werden in den Subkomponenten modelliert.
#                    In der hoeheren Komponente Adam werden nur die
#                    Verbindungen der Subkomponenten geschaffen.
#
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Adam (von der Komponente Actor) an WorldOfAdam (an die Komponente
#      Environment):
#       + DatAnENV
#
#    - WorldOfAdam (von der Komponente Environment) an Adam (an die Komponente
#      Sensor):
#       + InFalle
#       + Feldart
#       + Nahrungsmenge
#       + xAdam
#       + yAdam
#
#-------------------------------------------------------------------------


HIGH LEVEL COMPONENT Adam


#-------------------------------------------------------------------------
# Der Agent Adam besteht (gemaess dem Referenzmodell PECS) aus den folgenden
# Komponenten:
# Zu beachten ist, dass die PECS-Komponente "Status" nicht verwendet wird, da
# Adam alleine in seiner Welt lebt.
#-------------------------------------------------------------------------
```

```
SUBCOMPONENTS
        Physis,
        Emotion,
        Cognition,
        Behaviour,
        Actor,
        Sensor,
        Perception


#-------------------------------------------------------------------------------
# Die folgenden Variablen aus der Komponente Environment werden von der eine
# Hierarchiestufe hoeher angesiedelten hoeheren Komponente WorldOfAdam
# bereitgestellt.
#-------------------------------------------------------------------------------
INPUT CONNECTIONS
        InFalle                 --> Sensor.InFalle;
        Feldart                 --> Sensor.Feldart;
        Nahrungsmenge           --> Sensor.Nahrungsmenge;
        xAdam                   --> Sensor.xAdam;
        yAdam                   --> Sensor.yAdam;
        Protokoll               --> (Sensor.Protokoll,
                                     Perception.Protokoll,
                                     Cognition.Protokoll,
                                     Behaviour.Protokoll,
                                     Actor.Protokoll,
                                     Emotion.Protokoll,
                                     Physis.Protokoll);


#-------------------------------------------------------------------------------
# Von Adam zu WorldOfAdam muss nur eine Verbindung geschaffen werden.
# Dies ist die Location, auf der von der Komponente Actor Auftraege an die
# Komponente Environment geschickt werden.
#-------------------------------------------------------------------------------
OUTPUT EQUIVALENCES
        DatAnENV    := Actor.DatAnENV;


#-------------------------------------------------------------------------------
# Die folgenden Verbindungen zwischen den Subkomponenten werden benoetigt.
#-------------------------------------------------------------------------------
COMPONENT CONNECTIONS
        Cognition.TokenAnBEH    --> Behaviour.TokenVonCOG;
        Behaviour.TokenAnACT    --> Actor.TokenVonBEH;
        Actor.TokenAnCOG        --> Cognition.TokenVonACT;
        Actor.TokenAnSEN        --> Sensor.TokenVonACT;
        Sensor.TokenAnPER       --> Perception.TokenVonSEN;
        Perception.TokenAnCOG   --> Cognition.TokenVonPER;

        Cognition.Handlungsplan --> Behaviour.Handlungsplan;
        Behaviour.Aktionsfolge  --> Actor.Aktionsfolge;
        Actor.DatAnCOG          --> Cognition.DatVonACT;
        Actor.DatAnSEN          --> Sensor.DatVonACT;
        Sensor.DatAnPER         --> Perception.DatVonSEN;
        Perception.DatAnCOG     --> Cognition.DatVonPER;

        Emotion.Angst           --> Behaviour.Angst;
        Physis.MStHunger        --> Behaviour.MStHunger;
        Physis.FehlendeEnergie  --> Behaviour.FehlendeEnergie;
        Physis.Energie_essen    --> Actor.Energie_essen;
        Actor.Zustand           --> Physis.Zustand;
        Cognition.InFalle       --> Emotion.InFalle;
        Cognition.GefahrenstelleEntdeckt
                                --> Behaviour.GefahrenstelleEntdeckt;
        Cognition.InFalle       --> Behaviour.InFalle;
        Cognition.Feldart {ALL i}{ALL j}
```

```
                                --> Behaviour.Feldart[i][j];
        Cognition.Nahrungsmenge {ALL i}{ALL j}
                                --> Behaviour.Nahrungsmenge[i][j];
        Cognition.xAdam         --> Behaviour.xAdam;
        Cognition.yAdam         --> Behaviour.yAdam;
        Cognition.MStWissenserwerb --> Behaviour.MStWissenserwerb;

        Behaviour.HLMotiv       --> Cognition.HLMotiv;

END OF Adam
```

D.3.2 Die Komponente *Sensor*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell ADAM
#
#  Name:             Sensor
#
#  Art:              Basiskomponente
#
#  Version:          1
#
#  Beschreibung:     Aufnahme von Umwelt-Informationen (aus der Komponente
#                    Environment) mittels der Aktionen "Pruefen" und
#                    "Explorieren".
#
#                    Die Komponente Actor schickt den Auftrag zur Ausfuehrung
#                    einer dieser Aktionen. Die Komponente Sensor hat Zugriff
#                    auf die relevanten Informationen der Komponente
#                    Environment und kann so die Auftraege bearbeiten und die
#                    gewonnene Information an die Komponente Perception
#                    weitergeben.
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Actor an Sensor
#       + DatVonACT
#       + TokenVonACT
#
#    - Adam (von Environment) an Sensor
#       + Feldart
#       + Nahrungsmenge
#       + xAdam
#       + yAdam
#       + InFalle
#
#    - Sensor an Perception
#       + DatAnPER
#       + TokenAnPER
#
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Sensor

MOBILE SUBCOMPONENTS OF CLASS DatenTransfer, Token

USE OF UNITS

   #-----------------------------------------------------------------------------
   # Energie-Einheit
   #-----------------------------------------------------------------------------
```

```
   UNIT [EnE] = BASIS

   #-------------------------------------------------------------------------
   # Als Zeiteinheit wird eine "Stunde" festgelegt.
   #-------------------------------------------------------------------------
   TIMEUNIT = [h]

LOCAL DEFINITIONS

 DIMENSIONS

   #-------------------------------------------------------------------------
   # Groesse des quadratischen Spielfeldes in x- und y-Richtung: XYMAX
   #-------------------------------------------------------------------------
   XYMAX                      := 12

   #-------------------------------------------------------------------------
   # Die moeglichen Arten der Felder des Spielfeldes:
   #  - Neutral:     bekanntes neutrales Feld
   #  - Nahrung:     bekanntes Nahrungsfeld
   #  - Gefahr:      bekanntes Gefahrenfeld
   #  - Unbekannt:   unbekanntes Feld
   #-------------------------------------------------------------------------
   VALUE SET FELDARTEN : ('Neutral', 'Nahrung', 'Gefahr', 'Unbekannt')

DECLARATION OF ELEMENTS

 SENSOR VARIABLES

  DISCRETE

   #-------------------------------------------------------------------------
   # Aus Environment: Feldart von jedem Feld
   #-------------------------------------------------------------------------
   ARRAY [XYMAX][XYMAX] Feldart  (FELDARTEN) := 'Neutral' ,

   #-------------------------------------------------------------------------
   # Aus Environment: x-Koordinate des Feldes, auf dem sich Adam gerade
   # befindet
   #-------------------------------------------------------------------------
   xAdam                         (INTEGER)   := 5 ,

   #-------------------------------------------------------------------------
   # Aus Environment: y-Koordinate des Feldes, auf dem sich Adam gerade
   # befindet
   #-------------------------------------------------------------------------
   yAdam                         (INTEGER)   := 5 ,

   #-------------------------------------------------------------------------
   # Aus Environment: gesetzt (TRUE), falls sich Adam gerade in einer Falle
   # befindet.
   # Es wird also unterschieden, ob sich Adam auf einem Gefahrenfeld innerhalb
   # oder ausserhalb der Fallgrube befindet.
   #-------------------------------------------------------------------------
   InFalle                       (LOGICAL)   := FALSE ,

   #-------------------------------------------------------------------------
   # Protokoll = TRUE: Textausgabe am Bildschirm aktiviert
   #-------------------------------------------------------------------------
   Protokoll                     (LOGICAL)   := FALSE

  CONTINUOUS

   #-------------------------------------------------------------------------
   # Aus Environment: Nahrungsmenge der einzelnen Felder
   # (ist natuerlich nur auf Nahrungsfeldern > 0)
   #-------------------------------------------------------------------------
```

```
  ARRAY [XYMAX][XYMAX] Nahrungsmenge (REAL [EnE]) :=  0.0 [EnE]

TRANSITION INDICATORS

  #------------------------------------------------------------------------
  # Indikator:   Falls gesetzt, wird das Weitergeben des Tokens an die
  #              Perception in die Wege geleitet
  #------------------------------------------------------------------------
  TokenAnPerception

LOCATIONS
  #------------------------------------------------------------------------
  # Location, auf der die Nachrichten an die Komponente Perception abgelegt
  # werden (ist dort SENSOR LOCATION)
  #------------------------------------------------------------------------
  DatAnPER    (DatenTransfer)   := 0 DatenTransfer ,

  #------------------------------------------------------------------------
  # Location, auf der das Token fuer die Komponente Perception abgelegt
  # wird (ist dort SENSOR LOCATION)
  #------------------------------------------------------------------------
  TokenAnPER  (Token)           := 0 Token

SENSOR LOCATIONS
  #------------------------------------------------------------------------
  # Location, auf der die Nachrichten von der Komponente Actor empfangen
  # werden
  #------------------------------------------------------------------------
  DatVonACT   (DatenTransfer)   := 0 DatenTransfer ,

  #------------------------------------------------------------------------
  # Location, auf der das Token von der Komponente Perception ankommt
  #------------------------------------------------------------------------
  TokenVonACT (Token)           := 0 Token

DYNAMIC BEHAVIOUR

  #------------------------------------------------------------------------
  #
  # Ereignis 1: "Bearbeiten einer Nachricht der Komponente Actor"
  #
  #     Ausfuehren einer der Aktionen "Explorieren" bzw. "Pruefen"
  #
  #------------------------------------------------------------------------

  WHENEVER ((NUMBER (DatVonACT) > 0 ) AND (NUMBER (TokenVonACT) > 0))
  DO

     #---------------------------------------------------------------------
     # Die auszufuehrende Aktion ist "Explorieren"
     #---------------------------------------------------------------------
     IF (DatVonACT:DatenTransfer[1].Aktion = 'Explorieren')
     DO
        #------------------------------------------------------------------
        # Die mobile Komponente DatenTransfer wird vom Sensor an die
        # Perception weitergereicht.
        # Der Sensor nimmt dabei folgende Aenderungen vor:
        # - Koordinaten:       aendern sich auf die Koordinaten des Feldes,
        #                      auf dem sich Adam gerade befindet
        # - InFalle:           Falls sich Adam gerade innerhalb einer
        #                      Fallgrube befindet, wird dies hier vermerkt.
        # - Nahrungsmenge:     Nahrungsmenge des aktuellen Feldes
        # - Aktion:            keine Aenderung (bleibt 'Explorieren')
        # - Feldart:           aendert sich auf die Feldart des Feldes, auf
        #                      dem sich Adam gerade befindet
        #------------------------------------------------------------------
        DatAnPER^: FROM DatVonACT GET DatenTransfer[1]
```

```
      CHANGING
         x^             := xAdam;
         y^             := yAdam;
         InFalle^       := InFalle;
         Nahrungsmenge^ := Nahrungsmenge [xAdam] [yAdam];
         Feldart^       := Feldart[xAdam][yAdam];
      END

   IF Protokoll
   DO
      DISPLAY ("\n T= %f: SEN: Exploriere (%u,%u): ",
                T, xAdam, yAdam);
      IF (Feldart[xAdam][yAdam] = 'Nahrung')
      DO
         DISPLAY ("Nahrungsfeld ; Nahrungsmenge = %f \n",
                  Nahrungsmenge [xAdam] [yAdam]);
      END
      ELSIF (Feldart[xAdam][yAdam] = 'Gefahr')
      DO
         DISPLAY (" Gefahrenfeld ");
         IF (InFalle = TRUE)
         DO
            DISPLAY ("\n------------------------------\n");
            DISPLAY (" Adam ist innerhalb der Falle!\n");
            DISPLAY ("------------------------------\n");
         END
         ELSE
         DO
            DISPLAY ("; Adam ist ausserhalb der Fallgrube.\n");
         END
      END
      ELSIF (Feldart[xAdam][yAdam] = 'Neutral')
      DO
         DISPLAY (" neutrales Feld \n");
      END
   END
END

#-------------------------------------------------------------------------
# Die auszufuehrende Aktion ist "Pruefen"
#-------------------------------------------------------------------------
ELSIF (DatVonACT:DatenTransfer[1].Aktion = 'Pruefen')
DO

   #----------------------------------------------------------------------
   # Die mobile Komponente DatenTransfer wird vom Sensor an die
   # Perception weitergereicht.
   # Der Sensor nimmt dabei folgende Aenderungen vor:
   # - Koordinaten:       keine Aenderung
   # - InFalle:           keine Aenderung (bleibt 'FALSE')
   # - Nahrungsmenge:     keine Aenderung (bleibt '0')
   # - Aktion:            keine Aenderung (bleibt 'Pruefen')
   # - Feldart:           aendert sich auf 'Gefahr' oder 'Unbekannt'
   #----------------------------------------------------------------------

   #----------------------------------------------------------------------
   # Das gepruefte Feld entpuppt sich als Gefahrenstelle
   #----------------------------------------------------------------------
   IF (Feldart[DatVonACT:DatenTransfer[1].x]
              [DatVonACT:DatenTransfer[1].y] = 'Gefahr')
   DO
      DatAnPER^: FROM DatVonACT GET DatenTransfer[1]
      CHANGING
         #-------------------------------------------------------------
         # Feldart: Gefahrenfeld
         #-------------------------------------------------------------
         Feldart^ := 'Gefahr';
```

```
            END
         END

         #------------------------------------------------------------------
         # Das gepruefte Feld ist keine Gefahrenstelle
         #------------------------------------------------------------------
         ELSE
         DO
            DatAnPER^: FROM DatVonACT GET DatenTransfer[1]
            CHANGING
               #------------------------------------------------------------
               # Feldart: Unbekannt (das Pruefen hat nichts ergeben)
               #------------------------------------------------------------
               Feldart^ := 'Unbekannt';
            END
         END

         IF Protokoll
         DO
            DISPLAY ("\n T= %f: SEN: Pruefe (%u,%u): ", T,
                     DatVonACT:DatenTransfer[1].x,
                     DatVonACT:DatenTransfer[1].y);

            IF (Feldart[DatVonACT:DatenTransfer[1].x]
                       [DatVonACT:DatenTransfer[1].y] = 'Gefahr')
            DO
               DISPLAY (" GEFAHRENFELD ENTDECKT!\n");
            END
         END
      END

      #---------------------------------------------------------------------
      # Das Weitergeben des Tokens an die Perception wird in die Wege
      # geleitet
      #---------------------------------------------------------------------
      SIGNAL TokenAnPerception;

   END

   #------------------------------------------------------------------------
   #
   # Ereignis 2: "Weitergeben des Tokens an die Komponente Perception"
   #
   #------------------------------------------------------------------------

   ON TokenAnPerception
   DO
      #---------------------------------------------------------------------
      # Das Token wird auf die fuer die Komponente Perception bestimmte
      # Location gelegt.
      #---------------------------------------------------------------------
      TokenAnPER^: FROM TokenVonACT GET Token[1];
   END

END OF Sensor
```

D.3.3 Die Komponente *Perception*

```
#------------------------------------------------------------------------------
#
#  Projekt:          Modell ADAM
#
#  Name:             Perception
#
```

```
#  Art:              Basiskomponente
#
#  Version:          1
#
#  Beschreibung:     Modellierung der (selektiven) Wahrnehmung von Adam
#
#                    Von der Komponente Sensor kommende Informationen werden
#                    durch einen Filterungsprozess gegebenenfalls veraendert
#                    und an die Komponente Cognition weitergeleitet. Dieser
#                    Wahrnehmungsfilter (bei der Veraenderung) beinhaltet im
#                    Modell ADAM nur eine Schranke fuer die Wahrnehmung von
#                    Nahrung auf einem Nahrungsfeld: Wenn die Nahrungsmenge
#                    auf einem gerade explorierten Nahrungsfeld geringer ist
#                    als die Schranke "NahrungWahrnehmungMin", wird an die
#                    Komponente Cognition die Information weitergegeben, dass
#                    ein Feld mit der Nahrungsmenge 0 exploriert wurde.
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Sensor an Perception
#       + DatVonSEN
#       + TokenVonSEN
#
#    - Perception an Cognition
#       + DatAnCOG
#       + TokenAnCOG
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Perception

MOBILE SUBCOMPONENTS OF CLASS DatenTransfer, Token

USE OF UNITS

   #-----------------------------------------------------------------------------
   # Energie-Einheit
   #-----------------------------------------------------------------------------
   UNIT [EnE] = BASIS

   #-----------------------------------------------------------------------------
   # Als Zeiteinheit wird eine "Stunde" festgelegt.
   #-----------------------------------------------------------------------------
   TIMEUNIT = [h]

DECLARATION OF ELEMENTS

 CONSTANT

   #-----------------------------------------------------------------------------
   # Schranke fuer die selektive Wahrnehmung:
   # Falls die Nahrungsmenge auf einem gerade explorierten Nahrungsfeld kleiner
   # ist als der folgende Wert, wird an die Kognition anstatt des Wertes der
   # wirklichen Nahrungsmenge der Wert 0 uebermittelt.
   # Zu beachten ist, dass der Wert fuer NahrungWahrnehmungMin groesser sein
   # sollte als der Wert der Konstante NahrungMin in der Komponente
   # Environment, auf den die Nahrungsmenge eines Feldes nach dem Essen
   # gesetzt wird.
   #-----------------------------------------------------------------------------
   NahrungWahrnehmungMin   (REAL [EnE])      := 10.0 [EnE]

 SENSOR VARIABLE
  DISCRETE

   #-----------------------------------------------------------------------------
   # Protokoll = TRUE: Textausgabe am Bildschirm aktiviert
   #-----------------------------------------------------------------------------
```

```
  Protokoll               (LOGICAL)         := FALSE

 TRANSITION INDICATOR

  #-------------------------------------------------------------------------
  # Vor dem Weitergeben des Tokens an die Kognition wird dieser Indikator
  # gesetzt
  #-------------------------------------------------------------------------
  TokenAnCognition

 LOCATIONS

  #-------------------------------------------------------------------------
  # Auf diese Location werden Nachrichten an die Kognition gelegt
  #-------------------------------------------------------------------------
  DatAnCOG                (DatenTransfer)   := 0 DatenTransfer ,

  #-------------------------------------------------------------------------
  # Auf diese Location wird das Token fuer die Kognition gelegt
  #-------------------------------------------------------------------------
  TokenAnCOG              (Token)           := 0 Token

 SENSOR LOCATIONS

  #-------------------------------------------------------------------------
  # Auf dieser Location werden Nachrichten von dem Sensor empfangen
  #-------------------------------------------------------------------------
  DatVonSEN               (DatenTransfer)   := 0 DatenTransfer ,

  #-------------------------------------------------------------------------
  # Auf dieser Location wird das Token vom Sensor empfangen
  #-------------------------------------------------------------------------
  TokenVonSEN             (Token)           := 0 Token

DYNAMIC BEHAVIOUR

  #-------------------------------------------------------------------------
  #
  # Ereignis 1 : "Bearbeiten einer Nachricht vom Sensor"
  #
  #     Immer, wenn auf den beiden Locations, die mit der Komponente
  #     Sensor verbunden sind, mobile Komponenten vorhanden sind, wird
  #     die Komponente Perception aktiv und behandelt die Nachricht
  #     vom Sensor.
  #
  #-------------------------------------------------------------------------

  WHENEVER ((NUMBER (DatVonSEN) > 0 ) AND (NUMBER (TokenVonSEN) > 0))
  DO

     #-------------------------------------------------------------------------
     # Falls die vom Sensor ausgefuehrte Aktion "Explorieren" war und
     # die in der Nachricht vom Sensor angegebene Nahrungsmenge
     # kleiner ist als die Konstante NahrungWahrnehmungMin, so wird
     # in der Nachricht an die Cognition die Nahrungsmenge auf 0 gesetzt.
     #-------------------------------------------------------------------------
     IF ((DatVonSEN:DatenTransfer[1].Aktion = 'Explorieren') AND
         (DatVonSEN:DatenTransfer[1].Nahrungsmenge < NahrungWahrnehmungMin))
     DO
        DatAnCOG^: FROM DatVonSEN GET DatenTransfer[1]
        CHANGING
           Nahrungsmenge^ := 0 [EnE];
        END

        IF Protokoll
        DO
           IF (DatVonSEN:DatenTransfer[1].Nahrungsmenge > 0 [EnE])
```

```
            DO
               DISPLAY ("\n************************************************");
               DISPLAY ("\n T= %f: PER:", T);
               DISPLAY (" Nahrungsmenge (%f) auf Feld" ,
                           DatVonSEN:DatenTransfer[1].Nahrungsmenge);
               DISPLAY (" (%u,%u) unterhalb Wahrnehmungsschwelle! \n",
                  DatVonSEN:DatenTransfer[1].x, DatVonSEN:DatenTransfer[1].y );
               DISPLAY ("************************************************\n");
            END
         END
      END

      #-------------------------------------------------------------------------
      # Falls die vom Sensor ausgefuehrte Aktion nicht "Explorieren" war
      # bzw. die in der Nachricht vom Sensor angegebene Nahrungsmenge
      # nicht kleiner ist als die Konstante NahrungWahrnehmungMin, so
      # wird in der Nachricht an die Komponente Cognition die Nahrungsmenge auf
      # dem vom Sensor ermittelten Wert belassen.
      #-------------------------------------------------------------------------
      ELSE
      DO
         DatAnCOG^: FROM DatVonSEN GET DatenTransfer[1];
      END

      #-------------------------------------------------------------------------
      # Die Nachricht fuer die Cognition ist generiert. Nun kann das
      # Weitergeben des Tokens an die Cognition angestossen werden.
      #-------------------------------------------------------------------------
      SIGNAL TokenAnCognition;

   END

   #----------------------------------------------------------------------------
   #
   # Ereignis 2 : "Weitergeben des Tokens"
   #
   #     Wenn die Perception fertig ist (d.h. sie hat eine Nachricht
   #     fuer die Cognition generiert und auf der Location DatAnCOG
   #     abgelegt), gibt sie das Token an die Cognition weiter (d.h. sie
   #     verschiebt das Token von der Location TokenVonSEN auf die
   #     Location TokenAnCOG).
   #
   #----------------------------------------------------------------------------
   ON TokenAnCognition
   DO
      TokenAnCOG^: FROM TokenVonSEN GET Token[1];
   END

END OF Perception
```

D.3.4 Die Komponente *Physis*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell ADAM
#
#  Name:             Physis
#
#  Art:              Basiskomponente
#
#  Version:          1
#
#  Beschreibung:     Modellierung der physischen Belange von Adam:
#
```

```
#                     - Veraenderung des Energiehaushaltes von Adam.
#                       Falls die Energie von Adam dabei auf 0 faellt, so
#                       stirbt er.
#                       Ein Anstieg der Energie ueber eine obere Schranke
#                       wird verhindert.
#
#                     - Berechnung des Nahrungsbeduerfnisses
#
#                     - Ermitteln der Motivstaerke des Hunger-Motivs
#
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Actor an Physis
#       + Zustand
#
#    - Physis an Actor
#       + Energie_essen
#
#    - Physis an Behaviour
#       + MStHunger
#       + FehlendeEnergie
#
#-------------------------------------------------------------------------------

BASIC COMPONENT  Physis

USE OF UNITS

   #-----------------------------------------------------------------------------
   # Energie-Einheit
   #-----------------------------------------------------------------------------
   UNIT [EnE]  = BASIS

   #-----------------------------------------------------------------------------
   # Nahrungsbeduerfnis-Einheit
   #-----------------------------------------------------------------------------
   UNIT [NBE] = BASIS

   #-----------------------------------------------------------------------------
   # Motivstaerken-Einheit
   #-----------------------------------------------------------------------------
   UNIT [MStE] = BASIS

   #-----------------------------------------------------------------------------
   # Als Zeiteinheit wird eine "Stunde" festgelegt.
   #-----------------------------------------------------------------------------
   TIMEUNIT    = [h]

LOCAL DEFINITIONS


   #-----------------------------------------------------------------------------
   # Menge der moeglichen Zustaende von Adam
   #-----------------------------------------------------------------------------
   VALUE SET ZUSTAENDE : ('Planend', 'Explorierend', 'Pruefend',
                          'Gehend', 'Befreiend', 'Essend')

DECLARATION OF ELEMENTS

 CONSTANTS

   #-----------------------------------------------------------------------------
   # Faktor fuer die Abhaengigkeit der Motivstaerke des Motivs Hunger vom
   # Nahrungsbeduerfnis
   #-----------------------------------------------------------------------------
   HungerFaktor         (REAL [MStE])  := 3.0 [MStE] ,
```

```
  #-------------------------------------------------------------------------
  # Faktor fuer die Intensitaet der Abnahme der Energie in den Zustaenden
  # "Planend", "Explorierend", "Pruefend", "Gehend", "Befreiend"
  #-------------------------------------------------------------------------
  EFaktor              (REAL [1/h])   :=   4.0 [1/h],

  #-------------------------------------------------------------------------
  # spezieller Energieverbrauch im Zustand "Planend"
  #-------------------------------------------------------------------------
  Energie_planen       (REAL [EnE])   :=  -0.2 [EnE] ,

  #-------------------------------------------------------------------------
  # spezieller Energieverbrauch im Zustand "Explorierend"
  #-------------------------------------------------------------------------
  Energie_explorieren  (REAL [EnE])   :=  -0.3 [EnE] ,

  #-------------------------------------------------------------------------
  # spezieller Energieverbrauch im Zustand "Pruefend"
  #-------------------------------------------------------------------------
  Energie_pruefen      (REAL [EnE])   :=  -1.0 [EnE] ,

  #-------------------------------------------------------------------------
  # spezieller Energieverbrauch im Zustand "Gehend"
  #-------------------------------------------------------------------------
  Energie_gehen        (REAL [EnE])   :=  -0.8 [EnE] ,

  #-------------------------------------------------------------------------
  # spezieller Energieverbrauch im Zustand "Befreiend"
  #-------------------------------------------------------------------------
  Energie_befreien     (REAL [EnE])   :=  -4.0 [EnE] ,

  #-------------------------------------------------------------------------
  # Energiezugewinn pro Zeiteinheit im Zustand "Essend"
  # Auch benutzt im Actor; Je hoeher dieser Wert hier ist, desto kuerzer ist
  # die Zeit, die im Actor fuer die Aktion "Essen" verbraucht wird.
  #-------------------------------------------------------------------------
  Energie_essen        (REAL [EnE])   :=  20.0 [EnE] ,

  #-------------------------------------------------------------------------
  #  Wenn die Energie kleiner oder gleich dieser Schranke ist, stirbt Adam
  #-------------------------------------------------------------------------
  EnergieMin           (REAL [EnE])   :=   0.0 [EnE] ,

  #-------------------------------------------------------------------------
  # maximale Menge an Energie, die Adam erreichen kann.
  #-------------------------------------------------------------------------
  EnergieMax           (REAL [EnE])   := 100.0 [EnE] ,

  #-------------------------------------------------------------------------
  # Schranke fuer das Nahrungsbeduerfnis, ab der das Motiv Hunger aktiviert
  # wird. Stellt sicher, dass Adam, wenn er sich satt gegessen hat, einige
  # Zeit lang nicht mehr hungrig ist.
  #-------------------------------------------------------------------------
  BedNahrungSchranke   (REAL [NBE])   := 0.2 [NBE]

STATE VARIABLES

 CONTINUOUS
  #-------------------------------------------------------------------------
  # aktueller Energiebestand von Adam
  #-------------------------------------------------------------------------
  Energie              (REAL [EnE])   :=  100.0 [EnE]

DEPENDENT VARIABLES

 CONTINUOUS
```

```
   #-------------------------------------------------------------------------
   # Motivstaerke des Motivs Hunger
   #-------------------------------------------------------------------------
   MStHunger            (REAL [MStE])  := 0.0 [MStE] ,

   #-------------------------------------------------------------------------
   # Beduerfnis nach Nahrung
   #-------------------------------------------------------------------------
   BedNahrung           (REAL [NBE])   := 0.0 [NBE] ,

   #-------------------------------------------------------------------------
   # EnergieMax - Energie, wird nur von der Komponente Behaviour benoetigt, um
   # dort feststellen zu koennen, wieviel Energie Adam beim Essen maximal zu
   # sich nimmt.
   #-------------------------------------------------------------------------
   FehlendeEnergie      (REAL [EnE])   := 0.0 [EnE]

 SENSOR VARIABLES

  DISCRETE

   #-------------------------------------------------------------------------
   # aktueller Zustand von Adam
   #-------------------------------------------------------------------------
   Zustand              (ZUSTAENDE)    := 'Explorierend',

   #-------------------------------------------------------------------------
   # Protokoll = TRUE: Textausgabe am Bildschirm aktiviert.
   #-------------------------------------------------------------------------
   Protokoll            (LOGICAL)      := FALSE

DYNAMIC BEHAVIOUR

   #-------------------------------------------------------------------------
   #
   # algebraische Gleichung 1 : "Berechnung des Nahrungsbeduerfnisses"
   #
   #     Die folgende Formel basiert auf einem inversen Zusammenhang zwischen
   #     der Energie und dem Nahrungsbeduerfnis.
   #
   #     Der Faktor 1.0 [NBE] ist notwendig, damit die rechte Seite
   #     der Gleichung die Einheit [NBE] (Nahrungsbeduerfnis-Einheit)
   #     hat.
   #     Der Summand 1.0 [EnE] soll verhindern, dass der Nenner 0 wird.
   #
   #-------------------------------------------------------------------------
   BedNahrung := 1.0 [NBE] * (EnergieMax - Energie) / (Energie + 1.0 [EnE]) ;

   #-------------------------------------------------------------------------
   #
   # algebraische Gleichung 2: "Berechnung der Motivstaerke des Motivs Hunger"
   #
   #     Wenn das Nahrungsbeduerfnis unter einer gewissen Schranke
   #     "BedNahrungSchranke" ist, hat Adam keinen Hunger.
   #     Ansonsten verhaelt sich der Hunger logarithmisch zum
   #     Nahrungsbeduerfnis.
   #
   #     Der Summand 1.0 innerhalb der Logarithumsfunktion stellt sicher,
   #     dass das Ergebnis des Logarithmus nie negativ sein kann, da
   #     "BedNahrung" immer groesser oder gleich 0 ist.
   #
   #     Der Faktor 1.0 [1/NBE] innerhalb der Logarithmusfunktion ist
   #     notwendig, da die Standardfunktion LOG10(x) keine Einheiten-
   #     behafteten Argumente akzeptiert.
   #
   #-------------------------------------------------------------------------
   IF (BedNahrung < BedNahrungSchranke)
```

```
DO
   MStHunger := 0.0 [MStE];
END

ELSE
DO
   MStHunger  := HungerFaktor * LOG10( 1.0 [1/NBE] * BedNahrung + 1.0 );
END

#------------------------------------------------------------------------
#
# algebraische Gleichung 3: "Berechnung von 'FehlendeEnergie'"
#
#     Die "FehlendeEnergie" ergibt sich aus der Differenz aus dem
#     Maximum, das an Energie erreicht werden kann, "EnergieMax" und
#     der aktuellen Energie.
#     "FehlendeEnergie" wird nur von der Komponente Behaviour
#     benoetigt, um bei der Aktion "Essen" die Menge bestimmen zu
#     koennen, die gegessen werden soll.
#
#------------------------------------------------------------------------
FehlendeEnergie := EnergieMax - Energie;

#------------------------------------------------------------------------
#
# Differentialgleichung 1: "Veraenderung der Energie"
#
#     Die Aenderung der Energie ist abhaengig von dem Zustand, in dem sich
#     Adam befindet. Fuer jeden Zustand gibt es eine eigene Differential-
#     gleichung, die die Energieabnahme oder -zunahme(nur im Zustand
#     "Essend" festlegt.
#
#     Die Differentialgleichungen im einzelnen:
#     - 1a: "Energieverbrauch im Zustand 'Planend'"
#     - 1b: "Energieverbrauch im Zustand 'Explorierend'"
#     - 1c: "Energieverbrauch im Zustand 'Pruefend'"
#     - 1d: "Energieverbrauch im Zustand 'Gehend'"
#     - 1e: "Energieverbrauch im Zustand 'Befreiend'"
#     - 1f: "Energieverbrauch im Zustand 'Essend'"
#
#------------------------------------------------------------------------

#------------------------------------------------------------------------
#
# Differentialgleichung 1a: "Energieverbrauch im Zustand 'Planend'"
#
#     Die Aenderung der Energie von ADAM ergibt sich aus der
#     Multiplikation der Faktoren "Energie_planen" und "EFaktor".
#
#------------------------------------------------------------------------
IF (Zustand = 'Planend')
DO
   DIFFERENTIAL EQUATION
     Energie' := Energie_planen * EFaktor;
   END
END

#------------------------------------------------------------------------
#
# Differentialgleichung 1b: "Energieverbrauch im Zustand 'Explorierend'"
#
#     Die Aenderung der Energie von ADAM ergibt sich aus der
#     Multiplikation der Faktoren "Energie_explorieren" und "EFaktor".
#
#------------------------------------------------------------------------
ELSIF (Zustand = 'Explorierend')
DO
```

```
   DIFFERENTIAL EQUATION
     Energie' := Energie_explorieren * EFaktor;
   END
END

#-------------------------------------------------------------------------
#
# Differentialgleichung 1c: "Energieverbrauch im Zustand 'Pruefend'"
#
#     Die Aenderung der Energie von ADAM ergibt sich aus der
#     Multiplikation der Faktoren "Energie_pruefen" und "EFaktor".
#
#-------------------------------------------------------------------------
ELSIF (Zustand = 'Pruefend')
DO
   DIFFERENTIAL EQUATION
     Energie' := Energie_pruefen * EFaktor;
   END
END

#-------------------------------------------------------------------------
#
# Differentialgleichung 1d: "Energieverbrauch im Zustand 'Gehend'"
#
#     Die Aenderung der Energie von ADAM ergibt sich aus der
#     Multiplikation der Faktoren "Energie_gehen" und "EFaktor".
#
#-------------------------------------------------------------------------
ELSIF (Zustand = 'Gehend')
DO
   DIFFERENTIAL EQUATION
     Energie' := Energie_gehen * EFaktor;
   END
END

#-------------------------------------------------------------------------
#
# Differentialgleichung 1e: "Energieverbrauch im Zustand 'Befreiend'"
#
#     Die Aenderung der Energie von ADAM ergibt sich aus der
#     Multiplikation der Faktoren "Energie_befreien" und "EFaktor".
#
#-------------------------------------------------------------------------
ELSIF (Zustand = 'Befreiend')
DO
   DIFFERENTIAL EQUATION
     Energie' := Energie_befreien * EFaktor;
   END
END

#-------------------------------------------------------------------------
# Differentialgleichung 1f: "Energieverbrauch im Zustand 'Essend'"
#
#     Es wird hoechstens so viel Energie aufgenommen, wie der
#     Nahrungsstelle entnommen wird.
#     Im Zustand 'Essend' wird "Energie_essen" nicht mit "EFaktor"
#     multipliziert, da sonst die Energieaufnahme nicht mit dem
#     von der Nahrungsstelle entnommenen Wert uebereinstimmen
#     wuerde.
#
#     "Energie_essen" ist die Menge an Energie, die Adam im Zustand "Essend"
#     pro Zeiteinheit aufnimmt.
#
#     Der Faktor 1.0 [1/h] ist wegen der Einheiten-Ueberpruefung von
#     Simplex notwendig.
#-------------------------------------------------------------------------
```

```
   ELSIF (Zustand = 'Essend')
   DO
      DIFFERENTIAL EQUATION
        Energie' := 1.0 [1/h] * Energie_essen;
      END
   END

   #-------------------------------------------------------------------------
   #
   # Ereinis 1 : "Virtueller Tod von Adam"
   #
   #     Wenn die Energie auf oder unter den Wert EnergieMin faellt,
   #     stirbt Adam: Ausgabe an den Benutzer und Stop der Simulation.
   #     Das "WHENEVER" wird anstelle eines "ON" verwendet, damit es nicht
   #     moeglich ist, den Simulationslauf fortzusetzen.
   #
   #-------------------------------------------------------------------------

   WHENEVER (Energie <= EnergieMin)
   DO

      #----------------------------------------------------------------------
      # Die Energie soll nicht kleiner als EnergieMin werden. Deshalb setzen
      # auf EnergieMin.
      #----------------------------------------------------------------------
      Energie^ := EnergieMin;

      IF Protokoll
      DO
         DISPLAY ("\n\n++++++++++++++++++++++++++++++++++++++++++++++++++");
         DISPLAY ("\n             R. I. P.  \n");
         DISPLAY ("\n\n T= %f  ADAM verhungert qualvoll. \n",T);
         DISPLAY ("++++++++++++++++++++++++++++++++++++++++++++++++++\n\n");
      END

      #----------------------------------------------------------------------
      # Anhalten des Simulationslaufes
      #----------------------------------------------------------------------
      SIGNAL STOP;

   END

   #-------------------------------------------------------------------------
   #
   # Ereignis 2 : "Obere Schranke fuer die Energie"
   #
   #     Die Energiemenge kann einen gewissen Wert nicht uebersteigen,
   #     selbst, wenn Adam noch mehr essen wuerde.
   #
   #     (Eigentlich nicht benoetigt, da darauf geachtet wird, dass
   #     Adam nicht zuviel isst. Schadet aber auch nicht.)
   #
   #-------------------------------------------------------------------------

   ON  ^Energie > EnergieMax^
   DO
       Energie^ := EnergieMax;
   END

END OF  Physis
```

D.3.5 Die Komponente *Emotion*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell ADAM
#
#  Name:             Emotion
#
#  Art:              Basiskomponente
#
#  Version:          1
#
#  Beschreibung:     Verwaltung der emotionalen Zustandsvariable "Angst"
#
#                    Dynamik:
#
#                    - Die Angst steigt an, wenn Adam in der Cognition
#                      realisiert, dass er gerade in eine Falle geraten ist.
#
#                    - Das Absinken der Angst geschieht kontinuierlich
#                      gemaess der negativen Exponentialverteilung.
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Cognition an Emotion
#       + InFalle
#
#    - Emotion an Behaviour
#       + Angst
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Emotion

USE OF UNITS

   #-------------------------------------------------------------------------------
   # Motivstaerken-Einheit
   #-------------------------------------------------------------------------------
   UNIT [MStE] = BASIS

   #-------------------------------------------------------------------------------
   # Als Zeiteinheit wird eine "Stunde" festgelegt.
   #-------------------------------------------------------------------------------
   TIMEUNIT = [h]

DECLARATION OF ELEMENTS

 CONSTANTS


   #-------------------------------------------------------------------------------
   # Faktor fuer die Abnahme der Angst pro Zeiteinheit
   #-------------------------------------------------------------------------------
   AngstAbnahme   (REAL [1/h])      :=  -0.05 [1/h] ,

   #-------------------------------------------------------------------------------
   # Wert der Zunahme der Angst beim kognitiven Realisieren des Betretens
   # einer Falle
   #-------------------------------------------------------------------------------
   AngstZunahme   (REAL [MStE])     :=  60.0 [MStE] ,

   #-------------------------------------------------------------------------------
   # Maximalwert fuer die Angst, der nicht ueberschritten werden darf
   #-------------------------------------------------------------------------------
   AngstMax       (REAL [MStE])     := 100.0 [MStE]
```

```
 STATE VARIABLES
  CONTINUOUS

    #-------------------------------------------------------------------------
    # Wert der Emotion Angst in Motivstaerke-Einheiten
    #-------------------------------------------------------------------------
    Angst          (REAL [MStE])     :=  50.0 [MStE]

 SENSOR VARIABLES
  DISCRETE

    #-------------------------------------------------------------------------
    # InFalle ist TRUE, wenn die Kognition die Information hat, dass Adam
    # gerade in einer Falle ist.
    #-------------------------------------------------------------------------
    InFalle        (LOGICAL)         := FALSE ,

    #-------------------------------------------------------------------------
    # Protokoll = TRUE: Textausgabe am Bildschirm aktiviert.
    #-------------------------------------------------------------------------
    Protokoll      (LOGICAL)         := FALSE

DYNAMIC BEHAVIOUR

    #-------------------------------------------------------------------------
    #
    # Differentialgleichung 1 : "Abnahme der Angst"
    #
    #     Die Angst nimmt kontinuierlich gemaess der negativen
    #     Exponentialverteilung ab. Der Faktor AngstAbnahme
    #     ist verantwortlich fuer die Geschwindigkeit der Abnahme.
    #
    #-------------------------------------------------------------------------
    DIFFERENTIAL EQUATIONS
       Angst' :=  AngstAbnahme * Angst;
    END

    #-------------------------------------------------------------------------
    #
    # Ereignis 1 : "Zunahme der Angst"
    #
    #     Immer, wenn Adam kognitiv realisiert, dass er gerade in eine Falle
    #     geraten ist, d.h. wenn die Variable InFalle in der Komponente
    #     Cognition von FALSE auf TRUE wechselt, steigt die Angst sprunghaft
    #     an.
    #
    #-------------------------------------------------------------------------
    ON ^InFalle = TRUE^
    DO
       #----------------------------------------------------------------------
       #         Die Angst wird um den Wert AngstZunahme erhoeht.
       #         Der Maximalwert der Angst, AngstMax, darf dabei nicht
       #         ueberschritten werden.
       #----------------------------------------------------------------------
       Angst^ := MIN(AngstMax, Angst + AngstZunahme);

       IF Protokoll
       DO
          DISPLAY ("\n\n***********************************************\n");
          DISPLAY (" T= %f: EMO: Angstanstieg von %f auf %f!!! \n", T,
                    Angst, MIN(AngstMax, Angst + AngstZunahme));
          DISPLAY ("***********************************************\n");
       END
    END

END OF Emotion
```

D.3.6 Die Komponente *Cognition*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell ADAM
#
#  Name:             Cognition
#
#  Art:              Basiskomponente
#
#  Version:          1
#
#  Beschreibung:     Diese Komponente hat vielfaeltige Aufgaben:
#
#        - BedWissen berechnen
#        - Motivstaerke fuer das Motiv Wissenserwerb ermitteln
#        - Modellierung des Vergessens
#        - Lernen (Lernschranken, neue Felder)
#        - Kompetenz berechnen
#        - Handlungsstrategie auswaehlen
#        - Zielfeld bestimmen
#        - Die Route zum Zielfeld planen
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Perception an Cognition
#       + DatVonPER
#       + TokenVonPER
#
#    - Actor an Cognition
#       + DatVonACT
#       + TokenVonACT
#
#    - Behaviour an Cognition
#       + HLMotiv
#
#    - Cognition an Emotion
#       + InFalle
#
#    - Cognition an Behaviour
#        + Handlungsplan
#        + TokenVonCOG
#        + GefahrenstelleEntdeckt
#        + InFalle
#        + Feldart
#        + Nahrungsmenge
#        + xAdam
#        + yAdam
#        + MStWissenserwerb
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Cognition

MOBILE SUBCOMPONENTS OF CLASS DatenTransfer, Token, GedaechtnisEintrag

USE OF UNITS

   #-----------------------------------------------------------------------------
   # Energie-Einheit
   #-----------------------------------------------------------------------------
   UNIT [EnE]  = BASIS

   #-----------------------------------------------------------------------------
   # Motivstaerken-Einheit
   #-----------------------------------------------------------------------------
```

```
   UNIT [MStE] = BASIS

   #-------------------------------------------------------------------------
   # Wissensstand-Einheit
   #-------------------------------------------------------------------------
   UNIT [WiSE] = BASIS

   #-------------------------------------------------------------------------
   # Einheit fuer das Beduerfnis nach Wissen
   #-------------------------------------------------------------------------
   UNIT [WiBE] = BASIS

   #-------------------------------------------------------------------------
   # Kompetenz-Einheit
   #-------------------------------------------------------------------------
   UNIT [KoE]  = BASIS

   #-------------------------------------------------------------------------
   # Als Zeiteinheit wird eine Stunde festgelegt
   #-------------------------------------------------------------------------
   TIMEUNIT    = [h]

LOCAL DEFINITIONS

 DIMENSIONS

   #-------------------------------------------------------------------------
   # Groesse des quadratischen Spielfeldes in x- und y-Richtung: XYMAX
   #-------------------------------------------------------------------------
   XYMAX := 12

   #-------------------------------------------------------------------------
   # Die moeglichen Arten der Felder des Spielfeldes:
   #  - Neutral:     bekanntes neutrales Feld
   #  - Nahrung:     bekanntes Nahrungsfeld
   #  - Gefahr:      bekanntes Gefahrenfeld
   #  - Unbekannt:   unbekanntes Feld
   #-------------------------------------------------------------------------
   VALUE SET FELDARTEN : ('Neutral', 'Nahrung', 'Gefahr', 'Unbekannt')

   #-------------------------------------------------------------------------
   # Die Menge der moeglichen handlungsleitenden Motive:
   #  - Hunger
   #  - Wissenserwerb
   #-------------------------------------------------------------------------
   VALUE SET MOTIVE: ('Hunger', 'Wissenserwerb')

DECLARATION OF SUBFUNCTIONS

   #-------------------------------------------------------------------------
   # Die folgende Funktion ermittelt fuer die Handlungsstrategie 'Erkunden'
   # ein Zielfeld und gibt die Koordinaten dieses Zielfeldes zurueck.
   #-------------------------------------------------------------------------
   F_StrategieErkunden(        REAL: Auswahl,
                               ARRAY [m][n] FELDARTEN: Feldart,
                               INTEGER: Maximum,
                               INTEGER: xAdam,
                               INTEGER: yAdam,
                               LOGICAL: Protokoll
                               -->
                               INTEGER, INTEGER),

   #-------------------------------------------------------------------------
   # Die folgende Funktion ermittelt fuer die Handlungsstrategie
   # 'Nahrungssuche' ein Zielfeld und gibt die Koordinaten dieses
   # Zielfeldes zurueck.
   #-------------------------------------------------------------------------
```

```
   F_StrategieNahrungssuche(   REAL: Auswahl,
                               INTEGER: Maximum,
                               INTEGER: xAdam,
                               INTEGER: yAdam,
                               ARRAY [m][n] REAL [EnE]: Nahrungsmenge,
                               ARRAY [m][n] REAL [h]: TLeerGegessen,
                               REAL [h]: DeltaSicherNein,
                               LOGICAL: Protokoll
                               -->
                               INTEGER, INTEGER),

   #-------------------------------------------------------------------------
   # Die folgende Funktion ermittelt im Rahmen der Wegeplanung die
   # Koordinaten des naechsten Teilzieles fuer den Handlungsplan
   #-------------------------------------------------------------------------
   F_Wegeplanung(              ARRAY [m][n] FELDARTEN: Feldart,
                               INTEGER: Maximum,
                               INTEGER: xAktuell,
                               INTEGER: yAktuell,
                               INTEGER: xZiel,
                               INTEGER: yZiel
                               -->
                               INTEGER, INTEGER)

DECLARATION OF ELEMENTS
 CONSTANTS

   #-------------------------------------------------------------------------
   # Zeitspanne, die ein Feld nach jedem "Explorieren" im Gedaechtnis behalten
   # wird. Falls ein Feld, das noch im Gedaechtnis ist, erneut "exploriert"
   # wird, wird es um DeltaVergessen laenger im Gedaechtnis behalten.
   #-------------------------------------------------------------------------
   DeltaVergessen          (REAL [h])        :=  50.0 [h] ,

   #-------------------------------------------------------------------------
   # Wenn Adam sich auf Nahrungssuche befindet (also "Hunger" das handlungs-
   # leitende Motiv ist), erhoeht sich seine Kompetenz um den folgenden Wert,
   # wenn er auf dem anvisierten Zielfeld dann wirklich Nahrung vorfindet.
   #-------------------------------------------------------------------------
   KompetenzPlus           (REAL [KoE])      :=  10.0 [KoE] ,

   #-------------------------------------------------------------------------
   # Wenn Adam sich auf Nahrungssuche befindet (also "Hunger" das handlungs-
   # leitende Motiv ist), erniedrigt sich seine Kompetenz um den folgenden
   # Wert, wenn er auf dem anvisierten Zielfeld keine Nahrung vorfindet.
   #-------------------------------------------------------------------------
   KompetenzMinus          (REAL [KoE])      :=  20.0 [KoE] ,

   #-------------------------------------------------------------------------
   # Minimum fuer Kompetenz: Unter diesen Wert kann die Kompetenz nie fallen.
   #-------------------------------------------------------------------------
   KompetenzMin            (REAL [KoE])      :=   0.0 [KoE] ,

   #-------------------------------------------------------------------------
   # Maximum fuer Kompetenz: Ueber diesen Wert kann die Kompetenz nie steigen.
   #-------------------------------------------------------------------------
   KompetenzMax            (REAL [KoE])      := 100.0 [KoE] ,

   #-------------------------------------------------------------------------
   # Falls die Kompetenz geringer ist als der folgende Wert, so wird, wenn sich
   # Adam auf Nahrungssuche befindet, die pessimistische Variante gewaehlt.
   # (D.h., er begibt sich, falls moeglich, auf ein Nahrungsfeld, auf dem sich
   # sicher Nahrung befindet.)
   # Ansonsten (bei Kompetenz groesser oder gleich der KompetenzSchranke)
   # bezieht Adam auch unsichere Nahrungsfelder, auf denen sich Nahrung
   # befinden koennte (was er nicht sicher weiss), mit in seine Auswahl ein.
   #-------------------------------------------------------------------------
```

```
  KompetenzSchranke       (REAL [KoE])      :=  50.0 [KoE] ,

  #-------------------------------------------------------------------------
  # Faktor, der in die Berechnung der Motivstaerke des Motivs Wisssenserwerb
  # eingeht.
  #-------------------------------------------------------------------------
  WissensFaktor           (REAL [MStE])     :=   1.0 [MStE]

STATE VARIABLES

 DISCRETE

  #-------------------------------------------------------------------------
  # Wissensstand: Anzahl der Adam momentan bekannten Felder
  #-------------------------------------------------------------------------
  Wissensstand            (INTEGER [WiSE])   := 0 [WiSE] ,

  #-------------------------------------------------------------------------
  # Feldart von jedem Feld
  #-------------------------------------------------------------------------
  ARRAY [XYMAX][XYMAX] Feldart (FELDARTEN)  := 'Unbekannt' ,

  #-------------------------------------------------------------------------
  #  Nahrungsmenge von jedem Feld (nur interessant fuer Nahrungsfelder)
  #-------------------------------------------------------------------------
  ARRAY [XYMAX][XYMAX] Nahrungsmenge (REAL [EnE]) := 0 [EnE] ,

  #-------------------------------------------------------------------------
  # x-Koordinate des Feldes, von dem Adam denkt, dass er sich gerade darauf
  # befindet.
  #-------------------------------------------------------------------------
  xAdam                   (INTEGER)         := 5 ,

  #-------------------------------------------------------------------------
  # y-Koordinate des Feldes, von dem Adam denkt, dass er sich gerade darauf
  # befindet.
  #-------------------------------------------------------------------------
  yAdam                   (INTEGER)         := 5 ,

  #-------------------------------------------------------------------------
  # gesetzt (TRUE), falls Adams interne Repraesentation besagt, dass er sich
  # in einer Falle befindet.
  #-------------------------------------------------------------------------
  InFalle                 (LOGICAL)         := FALSE ,


  #-------------------------------------------------------------------------
  # Kompetenz
  #-------------------------------------------------------------------------
  Kompetenz               (REAL [KoE])      := 0 [KoE] ,

  #-------------------------------------------------------------------------
  # KompetenzModus = TRUE: Kompetenzberechnung aktiv
  #-------------------------------------------------------------------------
  KompetenzModus          (LOGICAL)         :=FALSE ,

  #-------------------------------------------------------------------------
  # x-Koordinate des Zielfeldes (des letzten Feldes im Handlungsplan)
  #-------------------------------------------------------------------------
  xZiel                   (INTEGER)         := 0 ,

  #-------------------------------------------------------------------------
  # y-Koordinate des Zielfeldes (des letzten Feldes im Handlungsplan)
  #-------------------------------------------------------------------------
  yZiel                   (INTEGER)         := 0 ,
```

```
  #-------------------------------------------------------------------------
  # ist gesetzt, wenn beim Neuplanen ein neues Teilziel des Handlungsplanes
  # erzeugt werden soll.
  #-------------------------------------------------------------------------
  TeilzielErzeugen        (LOGICAL)         := FALSE ,

  #-------------------------------------------------------------------------
  # untere Lernschranke: bis zu dieser Zeitspanne ist auf einem Nahrungsfeld
  # seit dem letzten Essen sicher noch keine Nahrung nachgewachsen.
  #-------------------------------------------------------------------------
  DeltaSicherNein         (REAL [h])        :=    0.0 [h] ,

  #-------------------------------------------------------------------------
  # obere Lernschranke: ab dieser Zeitspanne ist auf einem Nahrungsfeld seit
  # dem letzten Essen sicher schon Nahrung nachgewachsen.
  #-------------------------------------------------------------------------
  DeltaSicherJa           (REAL [h])        := 10000.0 [h] ,

  #-------------------------------------------------------------------------
  # Zeitpunkt des letzten Leer-Essens fuer Nahrungsfelder
  #-------------------------------------------------------------------------
  ARRAY [XYMAX][XYMAX] TLeerGegessen (REAL [h]) := 0.0 [h] ,

  #-------------------------------------------------------------------------
  # nur benoetigt fuer Animation: Codierung der HandlungsStrategie:
  # 0: Erkunden
  # 1: Nahrungssuche (optimistisch)
  # 2: Nahrungssuche (pessimistisch)
  #-------------------------------------------------------------------------
  Ani_HandlungsStrategie  (INTEGER)         := 0 ,

  #-------------------------------------------------------------------------
  # nur benoetigt fuer Animation:
  # Hier sind die Felder, die momentan im Handlungsplan stehen mit 1 markiert.
  # Alle anderen Felder haben den Wert 0.
  # (Dadurch, dass die Bestandteile des Handlungsplans auch in diesem Array
  # markiert sind, braucht die Animationskontrolle nicht auf die Location
  # Handlungsplan zugreifen.)
  #-------------------------------------------------------------------------
  ARRAY [XYMAX][XYMAX] Ani_TeilZiel (INTEGER) := 0

DEPENDENT VARIABLES

 CONTINUOUS

  #-------------------------------------------------------------------------
  # Je kleiner das Wissen von Adam ueber seine Umwelt ist, desto groesser
  # ist sein Beduerfnis nach Wissen. Das Wissensbeduerfnis beeinflusst die
  # Motivstaerke des Motivs Wissenserwerb.
  #-------------------------------------------------------------------------
  BedWissen       (REAL [WiBE])             := 0.0 [WiBE] ,

  #-------------------------------------------------------------------------
  # Das Motiv Wissenserwerb konkurriert mit dem Motiv Hunger (siehe Physis)
  # um die Rolle des handlungsleitenden Motivs.
  #-------------------------------------------------------------------------
  MStWissenserwerb     (REAL [MStE])        := 0.0 [MStE]

SENSOR VARIABLES

 DISCRETE

  #-------------------------------------------------------------------------
  # Das handlungsleitende Motiv (aus der Komponente Behaviour)
  #-------------------------------------------------------------------------
  HLMotiv              (MOTIVE)             := 'Hunger' ,
```

```
  #-------------------------------------------------------------------------
  # Protokoll = TRUE: Textausgabe am Bildschirm aktiviert.
  #-------------------------------------------------------------------------
  Protokoll            (LOGICAL)            := FALSE

RANDOM VARIABLES

  #-------------------------------------------------------------------------
  # gleichverteilte Zufallszahl fuer die Auswahl eines der gleich weit
  # entfernten (Ziel-)Felder
  #-------------------------------------------------------------------------
  Auswahl              (REAL) : UNIFORM (UpLimit:=1, LowLimit:=0)

TRANSITION INDICATORS

  #-------------------------------------------------------------------------
  # Wenn die Komponente Cognition fertig ist, wird mit diesem Indikator ein
  # Ereignis angestossen, das das Token an die Komponente Behaviour
  # weitergibt.
  #-------------------------------------------------------------------------
  TokenAnBehaviour,

  #-------------------------------------------------------------------------
  # Indikator fuer das Behaviour, damit es nach dem Entdecken einer
  # Gefahrenstelle (beim Pruefen) ein neues "Planen" anregt.
  #-------------------------------------------------------------------------
  GefahrenstelleEntdeckt,

  #-------------------------------------------------------------------------
  # Handlungsstrategie "Erkunden"
  #-------------------------------------------------------------------------
  Strat_Erkunden,

  #-------------------------------------------------------------------------
  # Handlungsstrategie "Nahrungssuche": - optimistische Variante
  #-------------------------------------------------------------------------
  Strat_Nahrungssuche_Opt,

  #-------------------------------------------------------------------------
  # Handlungsstrategie "Nahrungssuche": - pessimistische Variante
  #-------------------------------------------------------------------------
  Strat_Nahrungssuche_Pes

LOCATIONS

  #-------------------------------------------------------------------------
  # Handlungsplan: hier stehen die zum Zielfeld fuehrenden Teilziele, also
  # der geplante Weg zum Ziel. Die zur Erreichung des jeweiligen Teilziels
  # erforderlichen Einzelaktionen werden spaeter von der Komponente
  # Behaviour generiert.
  #-------------------------------------------------------------------------
  Handlungsplan  (DatenTransfer)   := 0 DatenTransfer ,

  #-------------------------------------------------------------------------
  # Location, auf der das Token an das Behaviour weitergegeben wird, wenn die
  # Komponente Cognition fertig ist.
  #-------------------------------------------------------------------------
  TokenAnBEH     (Token)           := 0 Token ,

  #-------------------------------------------------------------------------
  # Gedaechtnis ist eine Warteschlange, in der alle bekannten Felder, geordnet
  # nach der Vergessenszeit TVergessen, gespeichert sind.
  #-------------------------------------------------------------------------
  Gedaechtnis    (GedaechtnisEintrag ORDERED BY INC TVergessen) := 0
                                                    GedaechtnisEintrag ,
```

```
   #-------------------------------------------------------------------------
   # HilfsLoc braucht man nur zum Loeschen der von DatVonPER bzw.
   # DatVonACT verschobenen mobilen Komponenten.
   #-------------------------------------------------------------------------
   HilfsLoc       (DatenTransfer)   := 0 DatenTransfer

 SENSOR LOCATIONS

   #-------------------------------------------------------------------------
   # von Perception: Hier kommen Nachrichten von der Komponente Perception an
   #-------------------------------------------------------------------------
   DatVonPER      (DatenTransfer)   := 0 DatenTransfer ,

   #-------------------------------------------------------------------------
   # von Perception: Hier kommt das Token von der Komponente Actor an
   #-------------------------------------------------------------------------
   TokenVonPER    (Token)           := 0 Token ,

   #-------------------------------------------------------------------------
   # von Actor: Hier kommen Nachrichten von der Komponente Actor an
   #-------------------------------------------------------------------------
   DatVonACT      (DatenTransfer)   := 0 DatenTransfer ,

   #-------------------------------------------------------------------------
   # von Actor: Hier kommt das Token von der Komponente Actor an
   #-------------------------------------------------------------------------
   TokenVonACT    (Token)           := 0 Token

DYNAMIC BEHAVIOUR

   #-------------------------------------------------------------------------
   #
   # Ereignis 0: "Vermeiden einer Division durch 0"
   #
   #     Hier wird nur zu Simulationsbeginn sichergestellt, dass die
   #     Spielfeldgroesse XYMAX auf einen korrekten Wert eingestellt ist.
   #
   #-------------------------------------------------------------------------
   ON START
   DO
      #----------------------------------------------------------------------
      # Sicherheitshalber wird der Wert von XYMAX ueberprueft, damit
      # in der algebraischen Gleichung 1 kein Nenner 0 werden kann.
      #----------------------------------------------------------------------
      IF (XYMAX <= 0)
      DO
         DISPLAY("\n\n COG: Fehler bei der Einstellung des Parameters fuer ");
         DISPLAY("die Spielfeldgroesse: XYMAX <= 0 !!\n\n");

         #-------------------------------------------------------------------
         # Beenden der Simulation
         #-------------------------------------------------------------------
         SIGNAL STOP;
      END
   END

   #-------------------------------------------------------------------------
   #
   # algebraische Gleichung 1: "Berechnung des Beduerfnisses nach Wissen"
   #
   #     Die Variable "Wissensstand" beinhaltet die Anzahl der Adam bekannten
   #     Felder. Somit basiert folgende normierte Formel auf der einfacheren
   #     Beziehung:
   #
   #     BedWissen := "Anzahl unbekannter Felder" /
   #                  "Anzahl bekannter Felder"
   #
```

```
#     Die Normierung wird vorgenommen, damit das Wissensbeduerfnis mit dem
#     Nahrungsbeduerfnis (vgl. Physis) direkt vergleichbar bleibt, auch
#     wenn sich die Groesse des Spielfeldes aendert.
#     Sonst waere das Wissensbeduerfnis bei einem groesseren Spielfeld und
#     gleich vielen bekannten Feldern groesser!
#
#     Im Nenner wird 1.0 dazu addiert, damit er nie Null werden kann.
#
#     Der Faktor 1.0 [WiBE] ist notwendig, damit die rechte Seite der
#     Gleichung die Einheit [WiBE] hat.
#
#     Die Faktoren 1.0 [WiSE] sind wegen der Kommensurabilitaet der
#     Einheiten notwendig (dadurch kuerzt sich [WiSE] weg.).
#
#-------------------------------------------------------------------------

BedWissen := 1.0 [WiBE] *
             ((XYMAX*XYMAX) * 1.0 [WiSE] - Wissensstand) /
             (Wissensstand + 1.0 [WiSE]);

#-------------------------------------------------------------------------
#
# algebraische Gleichung 2: "Berechnung der Motivstaerke des Motivs
#                            Wissenserwerb"
#
#     Es wird hier angenommen, dass sich die Motivstaerke des Motivs
#     Wissenserwerb logarithmisch aus der Groesse des Beduerfnisses nach
#     Wissen ableiten laesst.
#     "WissensFaktor" ist dabei ein Faktor fuer die Gewichtung des Motivs
#     Wissenserwerb.
#
#     Die Zahl 1 wird im Argumentbereich des Logarithmus zu
#     "BedWissen" addiert, damit das Ergebnis des Logarithmus nie
#     negativ sein kann (BedWissen ist immer groesser oder gleich 0).
#
#     Der Faktor 1.0 [1/WiBE] ist notwendig, da die Standardfunktion
#     LOG10 kein einheitenbehaftetes Argument akzeptiert.
#
#-------------------------------------------------------------------------

MStWissenserwerb := WissensFaktor *
                    LOG10 (1.0 [1/WiBE] * BedWissen + 1.0);

#-------------------------------------------------------------------------
#
# Ereignis 1: "Vergessen"
#
#     Immer, wenn die Systemzeit mindestens so groß wird, wie der Zeitpunkt,
#     an dem ein Feld vergessen wird, wird die Feldart des entsprechenden
#     Feldes auf 'Unbekannt' gesetzt und der Eintrag im Gedaechtnis
#     geloescht. Auch alle anderen Informationen werden geloescht:
#     Die Nahrungsmenge und TLeerGegessen des entsprechenden Feldes werden
#     auf 0 gesetzt.
#
#-------------------------------------------------------------------------

IF (NUMBER (Gedaechtnis) > 0)
DO
   WHENEVER (T >= 1.0 [h] * Gedaechtnis:GedaechtnisEintrag[1].TVergessen)
   DO
      Feldart[Gedaechtnis:GedaechtnisEintrag[1].x]
             [Gedaechtnis:GedaechtnisEintrag[1].y]^ := 'Unbekannt';
      Nahrungsmenge[Gedaechtnis:GedaechtnisEintrag[1].x]
             [Gedaechtnis:GedaechtnisEintrag[1].y]^ := 0 [EnE];
      TLeerGegessen[Gedaechtnis:GedaechtnisEintrag[1].x]
             [Gedaechtnis:GedaechtnisEintrag[1].y]^ := 0 [h];
```

```
      Gedaechtnis^: REMOVE GedaechtnisEintrag[1];
      Wissensstand^ := Wissensstand - 1 [WiSE];

      IF Protokoll
      DO
         DISPLAY ("\n T= %f: COG: Feld (%u,%u) wird vergessen. \n",
                  T, Gedaechtnis:GedaechtnisEintrag[1].x,
                  Gedaechtnis:GedaechtnisEintrag[1].y);
      END
   END
END

#-------------------------------------------------------------------------
#  Ereignis 2: "Nachricht von der Komponente Perception"
#     Die letzte Aktion, die Adam ausgefuehrt hat, war also entweder
#     - pruefen oder
#     - explorieren
#
#     Wenn die Cognition eine Nachricht von der Perception bekommt, darf
#     sowohl die Warteschlange fuer eingehende Nachrichten von der
#     Perception nicht leer sein, als auch die Location auf der das Token
#     von der Perception kommt.
#-------------------------------------------------------------------------

ON ^((NUMBER (DatVonPER) > 0) AND (NUMBER (TokenVonPER) > 0 ))^

DECLARE
   #-------------------------------------------------------------------------
   # Dieser prozedurale Teil wird benoetigt, damit es keinen Segmentation
   # Violation Fehler gibt. Dieser Fehler tritt sonst an folgender
   # Stelle auf (wohl ein Simplex-Fehler)!:
   # Die letzte Aktion war explorieren und das Feld ist bereits bekannt.
   # In den Zeilen, in der die Location Gedaechtnis neu geordnet wird,
   # tritt sonst der Fehler auf.
   # Der schoene Nebeneffekt im Verwenden der Hilfvariablen ist ein
   # uebersichtlicherer Programm-Code.
   #-------------------------------------------------------------------------

   #-------------------------------------------------------------------------
   # in xPER wird die x-Koordinate des in der von der Perception kommenden
   # Nachricht behandelten Feldes gespeichert
   #-------------------------------------------------------------------------
   xPER (INTEGER) := 0,

   #-------------------------------------------------------------------------
   # in yPER wird die y-Koordinate des in der von der Perception kommenden
   # Nachricht behandelten Feldes gespeichert
   #-------------------------------------------------------------------------
   yPER (INTEGER) := 0

DO PROCEDURE

   #-------------------------------------------------------------------------
   # Abspeichern der Koordinaten des gepruefteten oder explorierten Feldes
   #-------------------------------------------------------------------------
   xPER := DatVonPER:DatenTransfer[1].x;
   yPER := DatVonPER:DatenTransfer[1].y;

END

DO TRANSITIONS

   #-------------------------------------------------------------------------
   # Die letzte von Adam durchgefuehrte Aktion, deren Ergebnis nun der
   # Cognition gemeldet wird, war "Pruefen".
   #-------------------------------------------------------------------------
   IF (DatVonPER:DatenTransfer[1].Aktion = 'Pruefen')
```

```
DO

   #------------------------------------------------------------------------
   # Beim Pruefen wurde ein Gefahrenfeld entdeckt.
   #------------------------------------------------------------------------
   IF (DatVonPER:DatenTransfer[1].Feldart = 'Gefahr')
   DO

      #------------------------------------------------------------------------
      # Abspeichern der Art des Feldes im Gedaechtnis
      #------------------------------------------------------------------------
      Feldart[xPER][yPER]^ := 'Gefahr';

      #------------------------------------------------------------------------
      # Die Variable, in der die Anzahl der bekannten Felder gespeichert
      # wird, muss erhoeht werden, da das Feld nun als Gefahrenfeld
      # bekannt ist!
      #------------------------------------------------------------------------
      Wissensstand^ := Wissensstand + 1 [WiSE];

      #------------------------------------------------------------------------
      # Der Zeitpunkt, an dem das Feld vergessen werden wird, wird
      # festgelegt.
      # Da das Feld geprueft wurde, war es zuvor nicht im Gedaechtnis!
      # Also muss nicht additiv aufgefrischt werden! Es kann einfach ein
      # Element auf der Location Gedaechtnis kreiert werden.
      #------------------------------------------------------------------------
      Gedaechtnis^: ADD 1 NEW GedaechtnisEintrag
         CHANGING
            x^ := xPER;
            y^ := yPER;
            TVergessen^ := 1.0 [1/h] * (T + DeltaVergessen);
         END

      IF Protokoll
      DO
         DISPLAY ("\n T= %f: COG: Feld (%u,%u) wird gemerkt."
                  , T, xPER, yPER);
      END

      #------------------------------------------------------------------------
      # Der Komponente Behaviour wird signalisiert, dass beim Pruefen
      # eines Feldes eine Gefahrenstelle entdeckt wurde:
      #------------------------------------------------------------------------
      SIGNAL GefahrenstelleEntdeckt;

   END

   #------------------------------------------------------------------------
   # Wenn beim Pruefen kein Gefahrenfeld entdeckt wird, wird
   # ganz normal fortgefahren, ohne etwas im Gedaechtnis zu
   # speichern.
   #------------------------------------------------------------------------
END

#------------------------------------------------------------------------
# Die letzte von Adam durchgefuehrte Aktion, deren Ergebnis nun der
# Cognition gemeldet wird, war "Explorieren".
#------------------------------------------------------------------------
ELSIF (DatVonPER:DatenTransfer[1].Aktion = 'Explorieren')
DO

   #------------------------------------------------------------------------
   # Falls die Koordinaten des gerade explorierten Feldes mit den
   # Koordinaten des gerade aktuellen Teilzieles (aus dem Handlungsplan)
   # uebereinstimmen, so wurde die mit diesem Teilziel direkt
   # korrespondierende Aktionsfolge bereits abgearbeitet. Das Teilziel
```

```
# kann jetzt somit geloescht werden.
#-----------------------------------------------------------------------
IF (NUMBER(Handlungsplan) > 0)
DO
   IF ((Handlungsplan:DatenTransfer[1].x = xPER) AND
       (Handlungsplan:DatenTransfer[1].y = yPER))
   DO
      Handlungsplan^: REMOVE DatenTransfer[1];

      IF Protokoll
      DO
         DISPLAY ("\n T= %f: COG: Das Teilziel (%u,%u) wird aus"
               , T, xPER, yPER);
         DISPLAY (" dem Handlungsplan geloescht.");
      END
   END
END

#-----------------------------------------------------------------------
# Abspeichern, ob sich Adam gerade in einer Fallgrube befindet.
#-----------------------------------------------------------------------
InFalle^ := DatVonPER:DatenTransfer[1].InFalle;

#-----------------------------------------------------------------------
# Abspeichern der aktuellen Koordinaten von Adam
#-----------------------------------------------------------------------
xAdam^ := xPER;
yAdam^ := yPER;

#-----------------------------------------------------------------------
# Abspeichern der Feldart des gerade explorierten Feldes.
#-----------------------------------------------------------------------
Feldart[xPER][yPER]^ := DatVonPER:DatenTransfer[1].Feldart;

#-----------------------------------------------------------------------
# Abspeichern der explorierten Nahrungsmenge des Feldes.
# Dies muesste nur bei Nahrungsfeldern gemacht werden. Der Einfachheit
# halber wird auf eine zusaetzliche Region verzichtet und fuer alle
# Feldarten die Nahrungmenge gespeichert. Diese ist bei Nicht-
# Nahrungsfeldern natuerlich 0.
#-----------------------------------------------------------------------
Nahrungsmenge[xPER][yPER]^:=DatVonPER:DatenTransfer[1].Nahrungsmenge;

#-----------------------------------------------------------------------
# Falls das Feld vor dem Explorieren noch nicht bekannt war (D.h., die
# momentan noch in der Cognition gespeicherte Feldart ist 'Unbekannt')
#-----------------------------------------------------------------------
IF (Feldart[xPER][yPER] = 'Unbekannt')
DO

   #--------------------------------------------------------------------
   # Die Variable, in der die Anzahl der bekannten Felder gespeichert
   # wird, muss erhoeht werden, da das explorierte Feld nun bekannt
   # ist!
   #--------------------------------------------------------------------
   Wissensstand^ := Wissensstand + 1 [WiSE];

   #--------------------------------------------------------------------
   # Der Zeitpunkt, an dem das Feld vergessen werden wird, wird
   # festgelegt.
   # Da das Feld noch nicht bekannt ist, war es zuvor nicht im
   # Gedaechtnis!
   # Also muss nicht additiv aufgefrischt werden!
   #--------------------------------------------------------------------
   Gedaechtnis^: ADD 1 NEW GedaechtnisEintrag
      CHANGING
         x^ := xPER;
```

```
            y^ := yPER;
            TVergessen^ := 1.0 [1/h] * (T + DeltaVergessen);
         END

      IF Protokoll
      DO
         DISPLAY ("\n T= %f: COG: Feld (%u,%u) wird gemerkt."
                  , T, xPER, yPER);
#         DISPLAY (" TVergessen := %f", T + DeltaVergessen);
      END

   END

   #-------------------------------------------------------------------
   # Falls das Feld vor dem Explorieren bereits bekannt war (D.h., die
   # momentan noch in der Cognition gespeicherte Feldart ist nicht
   # 'Unbekannt')
   #-------------------------------------------------------------------
   ELSE
   DO

      #----------------------------------------------------------------
      # Der Zeitpunkt, an dem das Feld vergessen werden wird, wird
      # festgelegt.
      # Da das Feld bereits bekannt ist, ist es bereits im
      # Gedaechtnis!
      # Also muss additiv aufgefrischt werden!
      # Die Gedaechtnis-Warteschlange wird entsprechend aktualisiert.
      # Dazu wird folgender "Trick" verwendet: Alle mobilen Komponenten,
      # die sich auf der Location Gedaechtnis befinden, werden von
      # dieser Location auf die gleiche Location geschoben.
      # Dadurch wird erreicht, dass alle mobilen Komponenten wieder
      # in der richtigen Reihenfolge (geordnet nach TVergessen) auf der
      # Location sind.
      # Hier tritt ein Segmentation Violation Error auf (zumindest
      # in der Version 0.19 von Simplex III), falls man an Stelle von
      # xPER und yPER DatVonPER:DatenTransfer[1].x bzw.
      # DatVonPER:DatenTransfer[1].y schreibt.
      #----------------------------------------------------------------
      Gedaechtnis^: FROM Gedaechtnis GET GedaechtnisEintrag
                    {ALL i|((Gedaechtnis:GedaechtnisEintrag[i].x = xPER)
                           AND
                            (Gedaechtnis:GedaechtnisEintrag[i].y = yPER))}
                    CHANGING
                       TVergessen^ :=
                           (Gedaechtnis:GedaechtnisEintrag[i].TVergessen
                            + 1.0 [1/h] * DeltaVergessen);
                    END
      IF Protokoll
      DO
         DISPLAY ("\n T= %f: COG: Feld (%u,%u) wird aufgefrischt! "
                  , T, xPER, yPER);

         #-------------------------------------------------------------
         # Da auf das ansonsten ueberfluessige Array TVergessen
         # verzichtet wird, kann hier nicht der Zeitpunkt des
         # Vergessens ausgegeben werden. Der "alte" Programmtext folgt:
         # DISPLAY ("    TVergessen: %f \n",
         #           TVergessen[xPER][yPER] + DeltaVergessen);
         #-------------------------------------------------------------

      END

      #----------------------------------------------------------------
      # Das bereits bekannte, explorierte Feld ist eine Nahrungsstelle
      #----------------------------------------------------------------
      IF (DatVonPER:DatenTransfer[1].Feldart = 'Nahrung')
```

```
DO

   #-------------------------------------------------------------
   # Es befindet sich Nahrung auf dem bekannten Nahrungsfeld
   #-------------------------------------------------------------
   IF (DatVonPER:DatenTransfer[1].Nahrungsmenge > 0 [EnE])
   DO

      #-------------------------------------------------------------
      # Aktualisieren der oberen Lernschranke "DeltaSicherJa".
      #
      # Wichtig: Der Zeitpunkt TLeerGegessen muss groesser 0 sein,
      # sonst kann zu Simulationsbeginn die Schranke falsch gesetzt
      # werden, da am Anfang TLeerGegessen ueberall auf 0 gesetzt
      # ist und die Nahrungsstellen voll sein koennen.
      #-------------------------------------------------------------
      IF (( TLeerGegessen[xPER][yPER] > 0 [h]) AND
          (T < (TLeerGegessen[xPER][yPER] + DeltaSicherJa)))
      DO

         #-------------------------------------------------------
         # Wenn das Intervall zwischen dem Leeressen einer
         # bestimmten Nahrungsstelle und dem Auffinden von Nahrung
         # auf diesem Feld (also dem festgestellten Nachwachsen
         # von Nahrung) kleiner ist als DeltaSicherJa wird diese
         # Schranke auf das Intervall gesetzt (verbessert).
         #-------------------------------------------------------
         DeltaSicherJa^ := T - TLeerGegessen[xPER][yPER];

         IF Protokoll
         DO
            DISPLAY ("\n T= %f: COG: DeltaSicherJa aktualisiert:"
                     , T);
            DISPLAY (" von %f auf %f ", DeltaSicherJa,
                     T - TLeerGegessen[xPER][yPER]);
         END
      END
   END # Nahrungsmenge > 0

   #-------------------------------------------------------------
   # Es befindet sich keine Nahrung auf dem bekannten Nahrungsfeld
   #-------------------------------------------------------------
   ELSE
   DO

      #-------------------------------------------------------------
      # Falls das Feld gerade leer gegessen wurde, ist die momentan
      # in der Cognition abgespeicherte Nahrungsmenge noch > 0
      #-------------------------------------------------------------
      IF (Nahrungsmenge[xPER][yPER] > 0 [EnE])
      DO

         #-------------------------------------------------------
         # Der Zeitpunkt des letzten Leer-Essens des aktuellen
         # Feldes wird aktualisiert.
         #-------------------------------------------------------
         TLeerGegessen[xPER][yPER]^ := T;

         IF Protokoll
         DO
            DISPLAY ("\n T= %f: COG: TLeerGegessen ",T);
            DISPLAY ("[%u][%u] := %f \n",
                     xPER,
                     yPER, T);
         END
      END
```

```
         #-------------------------------------------------------------
         # Das Feld wurde nicht gerade leer gegessen
         #-------------------------------------------------------------
         ELSE
         DO

            #----------------------------------------------------------
            # Aktualisieren der unteren Lernschranke
            # "DeltaSicherNein".
            #
            # Wichtig: Der Zeitpunkt TLeerGegessen muss groesser 0
            # sein, sonst kann zu Simulationsbeginn die Schranke
            # falsch gesetzt werden, da am Anfang TLeerGegessen
            # ueberall auf 0 gesetzt ist und die Nahrungsstellen
            # voll sein koennen.
            #----------------------------------------------------------
            IF (( TLeerGegessen[xPER][yPER] > 0 [h]) AND
                (T > (TLeerGegessen[xPER][yPER] + DeltaSicherNein)))
            DO

               #-------------------------------------------------------
               # Wenn das Intervall zwischen dem Leeressen einer
               # bestimmten Nahrungsstelle und dem Nicht-Finden von
               # Nahrung auf diesem Feld (also dem festgestellten
               # Nachwachsen von Nahrung) groesser ist als
               # DeltaSicherNein wird diese Schranke auf das Intervall
               # gesetzt (verbessert).
               #-------------------------------------------------------
               DeltaSicherNein^ := T - TLeerGegessen[xPER][yPER];

               IF Protokoll
               DO
                  DISPLAY ("\n T= %f: COG: DeltaSicherNein ",T);
                  DISPLAY ("aktualisiert: von %f auf %f ",
                           DeltaSicherNein, T - TLeerGegessen
                           [xPER][yPER]);
               END
            END
         END
      END # Nahrungsmenge = 0
   END # Feldart: Nahrung
END # Feldart: nicht 'Unbekannt'

#-----------------------------------------------------------------------
# "Aktualisieren der Kompetenz"
#
# Falls der KompetenzModus aktiviert ist und Adam gerade das
# Zielfeld exploriert hat, so wird sich die Kompetenz
#  - erhoehen, falls sich Nahrung auf dem Zielfeld befindet
#  - erniedrigen, falls keine Nahrung auf dem Zielfeld ist.
# Dabei ist zu beachten, dass der erlaubte Bereich fuer die
# Kompetenz nicht verlassen wird.
#
# Der KompetenzModus wird nach getaner Arbeit zurueckgesetzt.
#
# Achtung: Die Kompetenz wird auch erniedrigt, wenn das Zielfeld
# kein Nahrungsfeld ist!! Wenn also Adam hungrig ist, aber kein
# Nahrungsfeld kennt, leidet seine Kompetenz auch darunter, wenn
# er auf unbekannten Feldern keinen Erfolg hat und keine Nahrung
# vorfindet.
#-----------------------------------------------------------------------
IF ((KompetenzModus = TRUE) AND (xZiel = xPER) AND (yZiel = yPER))
DO

   #--------------------------------------------------------------------
   # Es befindet sich Nahrung auf dem Zielfeld: Kompetenzerhoehung
   #--------------------------------------------------------------------
```

```
            IF (DatVonPER:DatenTransfer[1].Nahrungsmenge > 0 [EnE])
            DO

               #------------------------------------------------------------------
               # Die Kompetenz muss immer <= KompetenzMax bleiben:
               #------------------------------------------------------------------
               Kompetenz^ := MIN(KompetenzMax, Kompetenz + KompetenzPlus);

               IF Protokoll
               DO
                  DISPLAY ("\n T= %f: COG: Kompetenzerhoehung", T);
                  DISPLAY (" auf %f\n",
                     MIN(KompetenzMax, Kompetenz + KompetenzPlus));
                  DISPLAY ("Kompetenz-Modus deaktiviert!");
               END
            END

            #------------------------------------------------------------------
            # Es befindet sich bei gesetztem Kompetenzmodus keine Nahrung
            # auf dem Zielfeld: Kompetenzerniedrigung
            #------------------------------------------------------------------
            ELSE
            DO
               #------------------------------------------------------------------
               # Die Kompetenz muss immer >= KompetenzMin bleiben:
               #------------------------------------------------------------------
               Kompetenz^ := MAX(KompetenzMin, Kompetenz - KompetenzMinus);

               IF Protokoll DO
                  DISPLAY ("\n T= %f: COG: Kompetenzerniedrigung",T);
                  DISPLAY (" auf %f\n",
                     MAX(KompetenzMin, Kompetenz - KompetenzMinus));
                  DISPLAY ("Kompetenz-Modus deaktiviert!");
               END
            END

            #------------------------------------------------------------------
            # Die Kompetenz wurde gerade verändert (falls sie nicht
            # bereits auf dem Minimal- oder Maximalwert war), somit ist die
            # Phase der Kompetenzberechnung (KompetenzModus = TRUE) vorbei.
            #------------------------------------------------------------------
            KompetenzModus^ := FALSE;

         END
      END # Aktion: explorieren

      #---------------------------------------------------------------------
      # Nach erfolgter Bearbeitung der Nachricht von der Perzeption ist die
      # Komponente Cognition fertig. Das Token wird an die Komponente
      # Behaviour weitergegeben.
      #---------------------------------------------------------------------
      SIGNAL TokenAnBehaviour;

      #---------------------------------------------------------------------
      # Zum Vernichten muss die Nachricht von der SENSOR Location DatVonPER auf
      # die "normale" Location HilfsLoc verschoben werden. Dort wird sie dann
      # vernichtet.
      #---------------------------------------------------------------------
      HilfsLoc^: FROM DatVonPER GET DatenTransfer[1];

END

#------------------------------------------------------------------------
#
#  Ereignis 3: "Nachricht von der Komponente Actor"
#
#     Die Aktion, die Adam gerade ausfuehrt, ist also "Planen".
```

```
#     In diesem Ereignis wird:
#
#        - der bestehende Handlungsplan geloescht
#        - eine Handlungsstrategie ausgewaehlt
#
#     Wenn die Cognition eine Nachricht vom Actor bekommt, darf
#     sowohl die Warteschlange fuer eingehende Nachrichten vom
#     Actor nicht leer sein, als auch die Location auf der das Token
#     vom Actor kommt.
#
#-------------------------------------------------------------------------

ON ^((NUMBER (DatVonACT) > 0) AND (NUMBER (TokenVonACT) > 0 ))^
DO

   #-------------------------------------------------------------------------
   # Falls in der Nachricht von der Komponente Actor die Aktion nicht
   # "Planen" ist, so ist im Programm ein Fehler! (s. u.)
   #-------------------------------------------------------------------------
   IF (DatVonACT:DatenTransfer[1].Aktion = 'Planen')
   DO
      #-------------------------------------------------------------------------
      # Loeschen des Handlungsplanes
      #-------------------------------------------------------------------------
      Handlungsplan^: REMOVE DatenTransfer{ALL};

      IF Protokoll
      DO
         DISPLAY ("\n T= %f: COG: Planen",T);
         IF (NUMBER(Handlungsplan) > 0)
         DO
            DISPLAY (" ; %u noch bestehende Teilziele ",
               NUMBER(Handlungsplan));
            DISPLAY ("des Handlungsplanes geloescht! ");
         END
      END

      #-------------------------------------------------------------------------
      # Auswahl einer Handlungsstrategie:
      # Die Auswahl der Handlungsstrategie ist primaer abhaengig vom
      # handlungsleitenden Motiv.
      #-------------------------------------------------------------------------

      IF (HLMotiv = 'Wissenserwerb')
      DO
         #-------------------------------------------------------------------------
         # Beim Handlungsleitenden Motiv 'Wissenserwerb' wird immer die
         # Handlungsstrategie 'Erkunden' gewaehlt.
         #-------------------------------------------------------------------------

         #-------------------------------------------------------------------------
         # Das Ausfuehren/Behandeln der Handlungsstrategie anstossen.
         #-------------------------------------------------------------------------
         SIGNAL Strat_Erkunden;

         #-------------------------------------------------------------------------
         # Fuer Kompetenzberechnung:
         # Der Kompetenzmodus wird auf FALSE gesetzt, da eine Aenderung der
         # Kompetenz nur stattfinden kann, wenn ADAM Hunger hat.
         #-------------------------------------------------------------------------
         KompetenzModus^ := FALSE;

         #-------------------------------------------------------------------------
         # Nur fuer Animation: Codierung der aktuellen Handlungsstrategie.
         #-------------------------------------------------------------------------
         Ani_HandlungsStrategie^ := 0;
```

```
   IF Protokoll
   DO
      DISPLAY ("\n T= %f: COG: Handlungsstrategie bestimmt: ", T);
      DISPLAY ("Erkunden ");

      IF (KompetenzModus = TRUE)
      DO
         #-----------------------------------------------------------
         # Falls der Kompetenzmodus gerade noch aktiv war, wurde er nun
         # deaktiviert. (Ansonsten war er sowieso nicht aktiv, konnte
         # also auch nicht deaktiviert werden.)
         #-----------------------------------------------------------
         DISPLAY ("; Kompetenz-Modus deaktiviert! ");
      END
   END

END

ELSIF (HLMotiv = 'Hunger')
DO
   #--------------------------------------------------------------------
   # Beim Handlungsleitenden Motiv 'Hunger' wird eine der beiden
   # Varianten der Handlungsstrategie 'Nahrungssuche' gewaehlt:
   #  - Nahrungssuche (optimistisch)
   #  - Nahrungssuche (pessimistisch)
   #--------------------------------------------------------------------

   #--------------------------------------------------------------------
   # Fuer Kompetenzberechnung:
   # Da die Handlungsstrategie "Nahrungssuche" ist, wird der
   # Kompetenzmodus gesetzt.
   # Dies ist also unabhaengig davon, ob ueberhaupt ein Nahrungsfeld
   # bekannt ist!!
   #--------------------------------------------------------------------
   KompetenzModus^ := TRUE;

   IF Protokoll
   DO
      IF (KompetenzModus = TRUE)
      DO
         #-----------------------------------------------------------
         # Falls der Kompetenzmodus gerade noch aktiv war, wurde er nun
         # deaktiviert. (Ansonsten war er sowieso nicht aktiv, konnte
         # also auch nicht deaktiviert werden.)
         #-----------------------------------------------------------
         DISPLAY ("\n COG: Planen: Kompetenz-Modus bleibt aktiviert.");
      END
      ELSE
      DO
         DISPLAY ("\n COG: Planen: Kompetenz-Modus aktiviert! ");
      END
   END

   IF (Kompetenz >= KompetenzSchranke)
   DO
      #-----------------------------------------------------------------
      # Da die Kompetenz hoch ist: optimistische Variante der
      # Strategie "Nahrungssuche"
      #-----------------------------------------------------------------

      #-----------------------------------------------------------------
      # Das Ausfuehren/Behandeln der Handlungsstrategie anstossen.
      #-----------------------------------------------------------------
      SIGNAL Strat_Nahrungssuche_Opt;
```

```
            #-------------------------------------------------------------
            # Nur fuer Animation:
            # Codierung der Handlungsstrategie
            #-------------------------------------------------------------
            Ani_HandlungsStrategie^ := 1;

            IF Protokoll
            DO
               DISPLAY ("\n T= %f: COG: Handlungsstrategie bestimmt: ",T);
               DISPLAY ("Nahrungssuche (optimistisch) ");
               DISPLAY ("\n   Kompetenz = %f ; KompetenzSchranke = %f ",
                  Kompetenz, KompetenzSchranke);
            END
         END

         ELSE
         DO
            #-------------------------------------------------------------
            # Da die Kompetenz niedrig ist: pessimistische Variante der
            # Strategie "Nahrungssuche"
            #-------------------------------------------------------------

            #-------------------------------------------------------------
            # Das Ausfuehren/Behandeln der Handlungsstrategie anstossen.
            #-------------------------------------------------------------
            SIGNAL Strat_Nahrungssuche_Pes;

            #-------------------------------------------------------------
            # Nur fuer Animation:
            # Codierung der Handlungsstrategie
            #-------------------------------------------------------------
            Ani_HandlungsStrategie^ := 2;

            IF Protokoll
            DO
               DISPLAY ("\n T= %f: COG: Handlungsstrategie bestimmt: ",T);
               DISPLAY ("Nahrungssuche (pessimistisch) ");
               DISPLAY ("\n   Kompetenz = %f ; KompetenzSchranke = %f ",
                  Kompetenz, KompetenzSchranke);
            END
         END
      END
   END

   #-----------------------------------------------------------------------
   # Nun wird das Vernichten der Nachricht in die Wege geleitet:
   # Zum Vernichten muss die Nachricht von der SENSOR Location DatVonACT auf
   # die "normale" Location HilfsLoc verschoben werden. Dort wird sie dann
   # vernichtet.
   #-----------------------------------------------------------------------
   HilfsLoc^: FROM DatVonACT GET DatenTransfer[1];

END

#--------------------------------------------------------------------------
#
# Ereignis 4:  "Bestimmen eines Zielfeldes gemaess der Handlungsstrategie"
#
#     Diese Ereignis teilt sich auf in drei Unterereignisse, die jeweils
#     versuchen, ein von der entsprechenden Handlungsstrategie gewuenschtes
#     Zielfeld zu ermitteln.
#
#     - "Handlungsstrategie 'Erkunden'"
#     - "Handlungsstrategie 'Nahrungssuche (optimistische Variante)'"
#     - "Handlungsstrategie 'Nahrungssuche (pessimistische Variante)'"
#
#--------------------------------------------------------------------------
```

```
#-------------------------------------------------------------------------
#
# Ereignis 4a: "Handlungsstrategie 'Erkunden'"
#
#     Diese Handlungsstrategie bestimmt ein unbekanntes Feld zum Zielfeld.
#
#     Mit Hilfe der Funktionskomponente F_StrategieErkunden wird eines der
#     naechstgelegenen unbekannten Felder zufaellig als Zielfeld
#     ausgewaehlt.
#
#-------------------------------------------------------------------------

ON Strat_Erkunden
DECLARE

   #-----------------------------------------------------------------------
   # Hilfsvariable zum Speichern der x-Koordinate des ermittelten
   # Zielfeldes
   #-----------------------------------------------------------------------
   Hilf_xZiel     (INTEGER) := 0,

   #-----------------------------------------------------------------------
   # Hilfsvariable zum Speichern der y-Koordinate des ermittelten
   # Zielfeldes
   #-----------------------------------------------------------------------
   Hilf_yZiel     (INTEGER) := 0

DO PROCEDURE

   #-----------------------------------------------------------------------
   # Aufruf der Funktion "F_StrategieErkunden": Diese gibt die Koordinaten
   # des Zielfeldes zurueck.
   #-----------------------------------------------------------------------
   (Hilf_xZiel, Hilf_yZiel) := F_StrategieErkunden(Auswahl,
                                                   ARRAY Feldart,
                                                   XYMAX,
                                                   xAdam,
                                                   yAdam,
                                                   Protokoll);

END # PROCEDURE

DO TRANSITIONS
   #-----------------------------------------------------------------------
   # Falls der Rueckgabewert von F_StrategieErkunden innerhalb des normalen
   # (erlaubten) Bereichs ist, so werden die Zielkoordinaten uebernommen.
   #-----------------------------------------------------------------------
   IF ((Hilf_xZiel > 0) AND (Hilf_yZiel > 0))
   DO
      IF Protokoll
      DO
         DISPLAY ("\n T= %f: COG: ", T);
         DISPLAY ("Zielfeld bestimmt: (%u,%u) ",
            Hilf_xZiel, Hilf_yZiel);
      END

      xZiel^ := Hilf_xZiel;
      yZiel^ := Hilf_yZiel;

      #--------------------------------------------------------------------
      # Es kann nun zum Wegeplanungsalgorithmus uebergegangen werden.
      # Dies geschieht durch das Setzen von TeilzielErzeugen.
      #--------------------------------------------------------------------
      TeilzielErzeugen^:= TRUE;
   END
```

```
   #------------------------------------------------------------------------
   # Falls der Rueckgabewert von F_StrategieErkunden negativ ist,
   # wird eine Fehlermeldung ausgegeben und der Simulationslauf abgebrochen.
   #------------------------------------------------------------------------
   ELSE
   DO
      IF Protokoll
      DO
         DISPLAY ("\n T= %f: COG: (FEHLER) Zielfeld ermitteln: Alle", T);
         DISPLAY (" Felder sind bekannt!\n Der Simulationslauf wird ");
         DISPLAY ("angehalten.");
      END

      SIGNAL STOP;
   END
END

#------------------------------------------------------------------------
#
# Ereignis 4b: "Handlungsstrategie: 'Nahrungssuche (optimistische
#              Variante)'"
#
#     Mit Hilfe der Funktionskomponente F_StrategieNahrungssuche wird
#     das naechstgelegene Nahrungsfeld, auf dem Nahrung sein
#     koennte (d.h. es wurde laenger als DeltaSicherNein nicht mehr
#     darauf gegessen), zum Zielfeld ausgewaehlt.
#     Falls mehrere gleich weit entfernte Nahrungsfelder in Frage kommen,
#     wird eines von diesen zufaellig bestimmt.
#
#     (Ein Nahrungsfeld, auf dem sicher Nahrung ist, wird einem
#     gleich weit entfernten Nahrungsfeld, auf dem Nahrung sein koennte,
#     nicht vorgezogen!! Beide haben die gleiche Chance, ausgewaehlt zu
#     werden.)
#
#     Falls kein Nahrungsfeld bekannt ist, auf dem schon wieder Nahrung
#     nachgewachsen sein koennte, so wird zu der Strategie "Erkunden"
#     uebergegangen (in der Hoffnung, zufaellig auf ein Nahrungsfeld
#     zu stossen).
#
#------------------------------------------------------------------------

ON Strat_Nahrungssuche_Opt
DECLARE

   #------------------------------------------------------------------------
   # Hilfsvariable zum Speichern der x-Koordinate des ermittelten
   # Zielfeldes
   #------------------------------------------------------------------------
   Hilf_xZiel     (INTEGER) := 0,

   #------------------------------------------------------------------------
   # Hilfsvariable zum Speichern der y-Koordinate des ermittelten
   # Zielfeldes
   #------------------------------------------------------------------------
   Hilf_yZiel     (INTEGER) := 0

DO PROCEDURE

   #------------------------------------------------------------------------
   # Aufruf der Funktion "F_StrategieNahrungssuche": Diese gibt die
   # Koordinaten des Zielfeldes zurueck.
   # Hier wird als vorletzter Uebergabeparameter DeltaSicherNein uebergeben
   # (im Gegensatz zur pessimistischen Variante)
   #------------------------------------------------------------------------
   (Hilf_xZiel, Hilf_yZiel) := F_StrategieNahrungssuche(Auswahl,
                       XYMAX,
                       xAdam,
```

```
                        yAdam,
                        ARRAY Nahrungsmenge,
                        ARRAY TLeerGegessen,
                        DeltaSicherNein,
                        Protokoll);

END # PROCEDURE

DO TRANSITIONS

   #------------------------------------------------------------------------
   # ACHTUNG:
   #
   # Falls sich Adam gerade auf einem Feld mit vorhandener Nahrung befindet,
   # waehlt er nicht dieses Feld als Zielfeld aus, sondern das von dem
   # Algorithmus vorgeschlagene, weiter entfernte. Er wird allerdings auf
   # seinem Feld trotzdem essen, allerdings ohne Einfluß auf seine
   # Kompetenz.
   #
   # Man koennte das ganze hier als Sonderfall behandeln und auch an dieser
   # Stelle die Kompetenz erhoehen. Darauf wurde aus Gruenden der
   # Einfachheit verzichtet (Kompetenzerhoehung nur an einer Stelle).
   #------------------------------------------------------------------------

   #------------------------------------------------------------------------
   # Falls der Rueckgabewert von F_StrategieNahrungssuche innerhalb
   # des normalen (erlaubten) Bereichs ist, so werden die Zielkoordinaten
   # uebernommen.
   #------------------------------------------------------------------------
   IF ((Hilf_xZiel > 0) AND (Hilf_yZiel > 0))
   DO
      IF Protokoll
      DO
         DISPLAY ("\n T= %f: COG: ", T);
         DISPLAY ("Zielfeld bestimmt:");
         DISPLAY (" (%u,%u) ", Hilf_xZiel, Hilf_yZiel);
      END

      xZiel^ := Hilf_xZiel;
      yZiel^ := Hilf_yZiel;

      #---------------------------------------------------------------------
      # Es kann nun zum Wegeplanungsalgorithmus uebergegangen werden.
      # Dies geschieht durch das Setzen von TeilzielErzeugen.
      #---------------------------------------------------------------------
      TeilzielErzeugen^:= TRUE;

   END

   #------------------------------------------------------------------------
   # Falls der Wert der Ausgabeparameter von F_StrategieNahrungssuche
   # negativ ist, wird zur Strategie "Erkunden" uebergegangen.
   #------------------------------------------------------------------------
   ELSE
   DO

      #---------------------------------------------------------------------
      # Kein Nahrungsfeld ist bekannt, auf dem sich Nahrung befinden
      # koennte. Deshalb wird Adam sein Glueck auf einem unbekannten Feld
      # versuchen (in der Hoffnung, dort Nahrung zu finden).
      #---------------------------------------------------------------------
      SIGNAL Strat_Erkunden;

      IF Protokoll
      DO
         DISPLAY ("\n T= %f: COG: ", T);
         DISPLAY ("Kein unsicheres oder sicheres ");
```

```
         DISPLAY ("Nahrungsfeld gefunden. Suche nun unbekanntes Feld.");
      END
   END
END

#-------------------------------------------------------------------------
#
# Ereignis 4c: "Handlungsstrategie: 'Nahrungssuche (pessimistische
#              Variante)'"
#
#     Mit Hilfe der Funktionskomponente F_StrategieNahrungssuche wird das
#     naechstgelegene Nahrungsfeld, auf dem sicher Nahrung ist, als
#     Zielfeld ausgewaehlt.
#     Falls mehrere gleich weit entfernte Nahrungsfelder in Frage kommen,
#     wird eines von diesen zufaellig bestimmt.
#
#     Falls kein Nahrungsfeld bekannt ist, auf dem Nahrung ist, so wird
#     zu der optimistischen Variante der Strategie "Nahrungssuche"
#     uebergegangen.
#
#-------------------------------------------------------------------------

ON Strat_Nahrungssuche_Pes
DECLARE

   #-------------------------------------------------------------------------
   # Hilfsvariable zum Speichern der x-Koordinate des ermittelten
   # Zielfeldes
   #-------------------------------------------------------------------------
   Hilf_xZiel     (INTEGER) := 0,

   #-------------------------------------------------------------------------
   # Hilfsvariable zum Speichern der y-Koordinate des ermittelten
   # Zielfeldes
   #-------------------------------------------------------------------------
   Hilf_yZiel     (INTEGER) := 0

DO PROCEDURE

   #-------------------------------------------------------------------------
   # Aufruf der Funktion "F_StrategieNahrungssuche": Diese gibt die
   # Koordinaten des Zielfeldes zurueck.
   # Hier wird als vorletzter Uebergabeparameter DeltaSicherJa uebergeben
   # (im Gegensatz zur pessimistischen Variante)
   #-------------------------------------------------------------------------
   (Hilf_xZiel, Hilf_yZiel) := F_StrategieNahrungssuche(Auswahl,
                       XYMAX,
                       xAdam,
                       yAdam,
                       ARRAY Nahrungsmenge,
                       ARRAY TLeerGegessen,
                       DeltaSicherJa,
                       Protokoll);

END # PROCEDURE

DO TRANSITIONS

   #-------------------------------------------------------------------------
   # ACHTUNG:
   #
   # Falls sich Adam gerade auf einem Feld mit vorhandener Nahrung befindet,
   # waehlt er nicht dieses Feld als Zielfeld aus, sondern das von dem
   # Algorithmus vorgeschlagene, weiter entfernte. Er wird allerdings auf
   # seinem Feld trotzdem essen, allerdings ohne Einfluß auf seine
   # Kompetenz.
   #
```

```
   # Man koennte das ganze hier als Sonderfall behandeln und auch an dieser
   # Stelle die Kompetenz erhoehen. Darauf wurde aus Gruenden der
   # Einfachheit verzichtet (Kompetenzerhoehung nur an einer Stelle).
   #-------------------------------------------------------------------------

   #-------------------------------------------------------------------------
   # Falls der Rueckgabewert von F_StrategieNahrungssuche innerhalb
   # des normalen (erlaubten) Bereichs ist, so werden die Zielkoordinaten
   # uebernommen.
   #-------------------------------------------------------------------------
   IF ((Hilf_xZiel > 0) AND (Hilf_yZiel > 0))
   DO
      IF Protokoll
      DO
         DISPLAY ("\n T= %f: COG: ", T);
         DISPLAY ("Zielfeld bestimmt:");
         DISPLAY (" (%u,%u)", Hilf_xZiel, Hilf_yZiel);
      END

      xZiel^ := Hilf_xZiel;
      yZiel^ := Hilf_yZiel;

      #-------------------------------------------------------------------------
      # Es kann nun zum Wegeplanungsalgorithmus uebergegangen werden.
      # Dies geschieht durch das Setzen von TeilzielErzeugen.
      #-------------------------------------------------------------------------
      TeilzielErzeugen^:= TRUE;

   END

   #-------------------------------------------------------------------------
   # Falls der Wert der Ausgabeparameter von F_StrategieNahrungssuche
   # "0" ist, wird eine Fehlermeldung ausgegeben und der Simulationslauf
   # abgebrochen.
   #-------------------------------------------------------------------------
   ELSE
   DO

      #-------------------------------------------------------------------------
      # Kein Nahrungsfeld ist bekannt, auf dem sich Nahrung befinden
      # koennte. Deshalb wird Adam sein Glueck auf einem unbekannten Feld
      # versuchen (in der Hoffnung, dort Nahrung zu finden).
      #-------------------------------------------------------------------------
      SIGNAL Strat_Nahrungssuche_Opt;

      IF Protokoll
      DO
         DISPLAY ("\n T= %f: COG: ", T);
         DISPLAY ("Kein sicheres Nahrungsfeld");
         DISPLAY (" gefunden! Suche nun unsicheres Nahrungsfeld.");
      END
   END
END

#-------------------------------------------------------------------------
#
# Ereignis 5: "Wegeplanung"
#
#     Immer, wenn das Zielfeld bereits berechnet wurde, aber der Weg
#     noch nicht fertig geplant ist, schaltet dieses Ereignis
#     (dann ist TeilzielErzeugen = TRUE) und ermittelt einen weiteren
#     Teilschritt (Teilziel), d.h. das naechste Feld des Handlungsplanes.
#
#-------------------------------------------------------------------------

WHENEVER (TeilzielErzeugen = TRUE)
DECLARE
```

```
   #------------------------------------------------------------------------
   # (hier zu ermittelnde) x-Koordinate des naechsten Feldes im Wegeplan
   #------------------------------------------------------------------------
   xNext (INTEGER),

   #------------------------------------------------------------------------
   # (hier zu ermittelnde) y-Koordinate des naechsten Feldes im Wegeplan
   #------------------------------------------------------------------------
   yNext (INTEGER),

   #------------------------------------------------------------------------
   # x-Koordinate des bisherigen Endpunktes der begonnenen Wegeplanung.
   # Falls die Wegeplanung noch nicht begonnen wurde (d.h. der
   # Handlungsplan ist noch leer), steht in xAktuell die
   # x-Koordinate von Adams aktueller Position.
   #------------------------------------------------------------------------
   xAktuell (INTEGER),

   #------------------------------------------------------------------------
   # y-Koordinate des bisherigen Endpunktes der begonnenen Wegeplanung.
   # Falls die Wegeplanung noch nicht begonnen wurde (d.h. der
   # Handlungsplan ist noch leer), steht in yAktuell die
   # y-Koordinate von Adams aktueller Position.
   #------------------------------------------------------------------------
   yAktuell (INTEGER)

DO PROCEDURE

   #------------------------------------------------------------------------
   # Der Wegeplanungsalgorithmus setzt bei den Koordinaten des bisherigen
   # Endpunktes des bereits geplanten Weges auf.
   # Wenn der Handlungsplan noch leer ist, wird bei der Wegeplanung mit den
   # Koordinaten von Adams aktueller Position angefangen.
   #------------------------------------------------------------------------
   IF (NUMBER(Handlungsplan) > 0)
   DO
      xAktuell := Handlungsplan:DatenTransfer[NUMBER(Handlungsplan)].x;
      yAktuell := Handlungsplan:DatenTransfer[NUMBER(Handlungsplan)].y;
   END
   ELSE
   DO
      xAktuell := xAdam;
      yAktuell := yAdam;
   END

   #------------------------------------------------------------------------
   # Aufruf der Funktion "F_Wegeplanung": Diese gibt die Koordinaten des
   # naechsten Teilzieles zurueck.
   #------------------------------------------------------------------------
   (xNext, yNext) := F_Wegeplanung(
                                   ARRAY Feldart,
                                   XYMAX,
                                   xAktuell,
                                   yAktuell,
                                   xZiel,
                                   yZiel);

END # PROCEDURE

DO TRANSITIONS

   #------------------------------------------------------------------------
   # Neues Teilziel im Handlungsplan erzeugen
   #------------------------------------------------------------------------
   Handlungsplan^: ADD 1 NEW DatenTransfer
      CHANGING
         x^    := xNext;
```

```
         y^    := yNext;
      END

   IF Protokoll
   DO
      DISPLAY ("\n T= %f: COG: %u. Teilziel des Handlungsplanes: ", T,
         NUMBER(Handlungsplan) + 1);
      DISPLAY ("(%u,%u) ", xNext, yNext);
   END

   #-------------------------------------------------------------------------
   # Falls das Ziel schon erreicht ist:
   #-------------------------------------------------------------------------
   IF ((xNext = xZiel) AND (yNext = yZiel))
   DO
      IF Protokoll
      DO
         DISPLAY ("\n T= %f: COG: Handlungsplan fertig erstellt.", T);
      END

      #-------------------------------------------------------------------------
      # Die Wegeplanung ist am Ziel! Kein Neues Teilziel mehr.
      #-------------------------------------------------------------------------
      TeilzielErzeugen^ := FALSE;

      #-------------------------------------------------------------------------
      # Der Handlungsplan ist fertig erstellt. Nun darf die Komponente
      # Behaviour etwas damit machen.
      #-------------------------------------------------------------------------
      SIGNAL TokenAnBehaviour;
   END
END

#-------------------------------------------------------------------------------
#
# Ereignis 6: "Weitergeben des Tokens an die Komponente Behaviour"
#
#     Wenn die Cognition mit ihren Aktionen fertig ist, gibt sie das Token
#     an die Komponente Behaviour weiter.
#
#     Die mobilen Komponenten auf der Location HilfsLoc werden geloescht.
#
#     Dabei wird fuer die Animation in das Array Ani_TeilZiel fuer jedes
#     Feld vermerkt, ob es gerade im Handlungsplan enthalten ist. Dadurch
#     wird es vermieden, in der Animation auf mobile Komponenten zugreifen
#     zu muessen.
#
#-------------------------------------------------------------------------------

ON TokenAnBehaviour

DECLARE

   #-------------------------------------------------------------------------
   # Hilfsarray, da im prozeduralen Teil keinen globalen Variablen Werte
   # zugewiesen werden duerfen. Initialisierung mit 0 (bedeutet "Feld ist
   # kein Teilziel").
   #-------------------------------------------------------------------------
   ARRAY [XYMAX][XYMAX] Hilf_Ani_TeilZiel(INTEGER)         := 0

DO PROCEDURE

   #-------------------------------------------------------------------------
   # Nur fuer Animation:
   # Es wird festgestellt, welche Felder im Handlungsplan stehen, um diese
   # auch graphisch darstellen zu koennen.
   #-------------------------------------------------------------------------
```

```
      FOR i FROM 1 TO NUMBER(Handlungsplan)
      REPEAT

         #-------------------------------------------------------------------
         # Allen Feldern, die im Handlungsplan stehen, wird in
         # Hilf_Ani_TeilZiel 1 zugewiesen (bedeutet: "Feld ist Teilziel")
         #-------------------------------------------------------------------
         Hilf_Ani_TeilZiel[Handlungsplan:DatenTransfer[i].x]
                          [Handlungsplan:DatenTransfer[i].y] :=1;

      END_LOOP

   END # PROCEDURE

   DO TRANSITIONS

      #-------------------------------------------------------------------
      # Nur fuer Animation:
      # Die festgestellten Teilziele werden von den Hilfsvariablen uebernommen.
      #-------------------------------------------------------------------
      Ani_TeilZiel{ALL i}{ALL j}^ := Hilf_Ani_TeilZiel[i][j];

      #-------------------------------------------------------------------
      # HilfsLoc ist nur zum Loeschen der von den SENSOR LOCATIONS DatVonPER
      # bzw. DatVonACT verschobenen mobilen Komponenten da. Hier werden "alle"
      # (kann nur eine sein) sich darauf befindenden mobilen Komponenten
      # geloescht.
      #-------------------------------------------------------------------
      HilfsLoc^: REMOVE DatenTransfer{ALL};

      #-------------------------------------------------------------------
      # Falls das Token auf der Location TokenVonACT liegt, wird es von dort
      # auf die Location TokenAnBEH verschoben (fuer die Komponente Behaviour).
      #-------------------------------------------------------------------
      IF (NUMBER (TokenVonACT) > 0)
      DO
         TokenAnBEH^: FROM TokenVonACT GET Token[1];
      END


      #-------------------------------------------------------------------
      # Falls das Token auf der Location TokenVonPER liegt, wird es von dort
      # auf die Location TokenAnBEH verschoben (fuer die Komponente Behaviour).
      #-------------------------------------------------------------------
      ELSIF (NUMBER (TokenVonPER) > 0)
      DO
         TokenAnBEH^: FROM TokenVonPER GET Token[1];
      END

      ELSE
      DO
         DISPLAY("\n T = %f : COG: Fehler in der Komponente Cognition:",T);
         DISPLAY(" Das Token ist nicht da. Der Simulationslauf wird");
         DISPLAY(" angehalten.\n\n");

         #-------------------------------------------------------------------
         # Anhalten des Simulationslaufes
         #-------------------------------------------------------------------
         SIGNAL STOP;
      END
   END

END OF Cognition
```

D.3.7 Die Komponente *Behaviour*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell ADAM
#
#  Name:             Behaviour
#
#  Art:              Basiskomponente
#
#  Version:          0 (mit Einzelereignissen)
#
#  Beschreibung:     Die Komponente Behaviour hat folgende Aufgaben:
#
#        - Motivselektion: Bestimmen des "Handlungsleitenden Motivs"
#        - Erstellen einer Aktionsfolge: Dies geschieht gemaess einem
#          festen Satz von Regeln. In der Komponente Behaviour wird also
#          das reaktive Verhalten von Adam modelliert.
#        - Initialisierung des Modells: Zu Simulationsbeginn startet Adam
#          in seiner unbekannten Umwelt mit der Aktion "Explorieren"
#
#  Verbindung(en) mit anderen Komponenten:
#
#     - Cognition an Behaviour
#        + Handlungsplan
#        + TokenVonCOG
#        + GefahrenstelleEntdeckt
#        + InFalle
#        + Feldart
#        + Nahrungsmenge
#        + xAdam
#        + yAdam
#        + MStWissenserwerb
#
#     - Emotion an Behaviour
#        + Angst
#
#     - Physis an Behaviour
#        + MStHunger
#        + FehlendeEnergie
#     - Behaviour an Actor
#        + Aktionsfolge
#        + TokenAnACT
#
#     - Behaviour an Cognition
#        + HLMotiv
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Behaviour

MOBILE SUBCOMPONENTS OF CLASS DatenTransfer, Token

USE OF UNITS

   #-----------------------------------------------------------------------------
   # Energie-Einheit
   #-----------------------------------------------------------------------------
   UNIT [EnE]  = BASIS

   #-----------------------------------------------------------------------------
   # Motivstaerken-Einheit
   #-----------------------------------------------------------------------------
   UNIT [MStE] = BASIS
```

```
   #-------------------------------------------------------------------------
   # Als Zeiteinheit wird eine "Stunde" festgelegt.
   #-------------------------------------------------------------------------
   TIMEUNIT    = [h]

LOCAL DEFINITIONS

 DIMENSIONS

   #-------------------------------------------------------------------------
   # Groesse des quadratischen Spielfeldes in x- und y-Richtung: XYMAX
   #-------------------------------------------------------------------------
   XYMAX                     := 12

   #-------------------------------------------------------------------------
   # Die moeglichen Arten der Felder des Spielfeldes:
   #  - Neutral:     bekanntes neutrales Feld
   #  - Nahrung:     bekanntes Nahrungsfeld
   #  - Gefahr:      bekanntes Gefahrenfeld
   #  - Unbekannt:   unbekanntes Feld
   #-------------------------------------------------------------------------
   VALUE SET FELDARTEN :      ('Neutral', 'Nahrung', 'Gefahr', 'Unbekannt')

   #-------------------------------------------------------------------------
   # Moeglichkeiten fuer das handlungsleitende Motiv:
   #  -  Hunger
   #  -  Wissenserwerb
   #-------------------------------------------------------------------------
   VALUE SET MOTIVE :         ('Hunger', 'Wissenserwerb')

DECLARATION OF ELEMENTS

 CONSTANTS

   #-------------------------------------------------------------------------
   # Wenn die Angst (aus der Komponente Emotion) groesser als diese Schranke
   # ist, so wird ein unbekanntes Feld vor dem Betreten "geprueft".
   #-------------------------------------------------------------------------
   AngstSchranke  (REAL [MStE])  := 50.0 [MStE]

 STATE VARIABLES

  DISCRETE

   #-------------------------------------------------------------------------
   # Handlungsleitendes Motiv: (auch benoetigt von der Komponente Kognition)
   #-------------------------------------------------------------------------
   HLMotiv        (MOTIVE)       := 'Wissenserwerb' ,

   #-------------------------------------------------------------------------
   # FalleEntdeckt wird auf TRUE gesetzt, falls gerade beim Pruefen eine Falle
   # entdeckt worden ist und dies von der Komponente Cognition durch das
   # Setzen des Indikators GefahrenstelleEntdeckt signalisiert wurde.
   # Wenn daraufhin ein neues "Planen" (und damit verbunden ein Loeschen
   # der aktuellen Aktionsfolge und des aktuellen Handlungsplanes) angestossen
   # wird, wird "FalleEntdeckt" wieder auf FALSE gesetzt.
   #-------------------------------------------------------------------------
   FalleEntdeckt  (LOGICAL)      := FALSE ,

   #-------------------------------------------------------------------------
   # Immer, wenn durch die Motivselektion festgestellt wird, dass ein Wechsel
   # des handlungsleitenden Motivs vollzogen werden muss, wird die logische
   # Variable "HLM_Wechselt" auf TRUE gesetzt.
   # Sie wird erst wieder auf FALSE zurueckgesetzt, wenn das handlungsleitende
   # Motiv wirklich geaendert wird.
   #-------------------------------------------------------------------------
   HLM_Wechselt   (LOGICAL)      := FALSE ,
```

```
  #-------------------------------------------------------------------------
  # Immer, wenn die Auswahl einer neuen Aktionsfolge beendet ist, wird
  # AF_Erstellt auf TRUE gesetzt, damit nicht sofort wieder ein anderes
  # Ereignis zur Erstellung einer Aktionsfolge schalten kann.
  # AF_Erstellt wird bei dem Weitergeben des Tokens an den Actor wieder auf
  # FALSE zurueckgesetzt.
  #-------------------------------------------------------------------------
  AF_Erstellt    (LOGICAL)      := FALSE

SENSOR VARIABLES

 DISCRETE

  #-------------------------------------------------------------------------
  # von Cognition: Feldart von jedem Feld
  #-------------------------------------------------------------------------
  ARRAY [XYMAX][XYMAX] Feldart(FELDARTEN) := 'Unbekannt' ,

  #-------------------------------------------------------------------------
  # von Cognition: x-Koordinate des Feldes, von dem Adam denkt, dass er sich
  # gerade darauf befindet.
  #-------------------------------------------------------------------------
  xAdam       (INTEGER)            := 0 ,

  #-------------------------------------------------------------------------
  # von Cognition: y-Koordinate des Feldes, von dem Adam denkt, dass er sich
  # gerade darauf befindet.
  #-------------------------------------------------------------------------
  yAdam       (INTEGER)            := 0 ,

  #-------------------------------------------------------------------------
  # von Cognition: gesetzt (TRUE), falls Adam gerade denkt, dass er sich in
  # einer Falle befindet.
  #-------------------------------------------------------------------------
  InFalle     (LOGICAL)            := FALSE ,

  #-------------------------------------------------------------------------
  # Protokoll = TRUE: Textausgabe am Bildschirm aktiviert.
  #-------------------------------------------------------------------------
  Protokoll   (LOGICAL)            := FALSE

 CONTINUOUS

  #-------------------------------------------------------------------------
  # von Cognition: Nahrungsmenge von jedem Feld
  #-------------------------------------------------------------------------
   ARRAY [XYMAX][XYMAX] Nahrungsmenge (REAL [EnE]) :=  0.0 [EnE] ,

  #-------------------------------------------------------------------------
  # von Cognition: die Motivstaerke des Motivs "Wissenserwerb"
  #-------------------------------------------------------------------------
  MStWissenserwerb  (REAL [MStE])  :=   0.0 [MStE] ,

  #-------------------------------------------------------------------------
  # von Physis: die Motivstaerke des Motivs "Hunger"
  #-------------------------------------------------------------------------
  MStHunger         (REAL [MStE])  :=   0.0 [MStE] ,

  #-------------------------------------------------------------------------
  # von Physis: die Differenz aus der maximal erreichbaren Energie und
  # der aktuell vorhandenen Energie.
  #-------------------------------------------------------------------------
  FehlendeEnergie   (REAL [EnE])   :=   0.0 [EnE] ,

  #-------------------------------------------------------------------------
  # von Emotion: die aktuelle Intensitaet der Emotion "Angst"
  #-------------------------------------------------------------------------
```

```
  Angst             (REAL [MStE])  :=  50.0 [MStE]

TRANSITION INDICATORS

  #-------------------------------------------------------------------------
  # Triggert das Ereignis, durch das das Token an die Komponente Actor
  # weitergegeben wird.
  #-------------------------------------------------------------------------
  TokenAnActor,

  #-------------------------------------------------------------------------
  # Hilfs-Indikator: wird benoetigt, da beim Generieren der Aktionsfolge
  # "Planen" zuerst alle auf der Lokation "Aktionsfolge" befindlichen mobilen
  # Komponenten geloescht werden muessen bevor "Planen" hineingeschrieben
  # wird. Um hier kein Problem mit der Reihenfolgenunabhaengigkeit der
  # Ereignisse zu bekommen, wird dieser Indikator zwischen das Loeschen und
  # Neugenerieren geschaltet.
  #-------------------------------------------------------------------------
  NeuPlanen,

  #-------------------------------------------------------------------------
  # Anstossen der Generierung der entsprechenden Aktionsfolge:
  # Dabei bedeutet:
  # Pl:       "Planen"
  # BeEx:     "Befreien - Explorieren"
  # EsEx:     "Essen - Explorieren"
  # PrGeEx:   "Pruefen - Gehen - Explorieren"
  # GeEx:     "Gehen - Explorieren"
  #-------------------------------------------------------------------------
  AF_Pl,
  AF_BeEx,
  AF_EsEx,
  AF_PrGeEx,
  AF_GeEx


SENSOR INDICATORS

  #-------------------------------------------------------------------------
  # Wennn beim "Pruefen" eine Gefahrenstelle entdeckt wird, signalisiert die
  # Cognition dies mit diesem Indikator. Im Verhalten wird dann die
  # Aktionsfolge "Planen" generiert.
  #-------------------------------------------------------------------------
  GefahrenstelleEntdeckt

LOCATIONS

  #-------------------------------------------------------------------------
  # Aktionsfolge: Die Komponente Behaviour generiert die Einzel-Aktionen
  # der Aktionsfolge.
  #-------------------------------------------------------------------------
  Aktionsfolge   (DatenTransfer)   := 0 DatenTransfer ,

  #-------------------------------------------------------------------------
  # Hier wird das Synchronisationstoken (fuer den Aktor) abgelegt, wenn
  # die Komponente Behaviour fertig ist.
  #-------------------------------------------------------------------------
  TokenAnACT     (Token)           := 0 Token

SENSOR LOCATIONS

  #-------------------------------------------------------------------------
  # Der Handlungsplan von Adam (aus der Komponente Cognition).
  # Hier stehen die Teilziele, zu denen die Komponente Behaviour jeweils eine
  # Aktionsfolge generiert.
  #-------------------------------------------------------------------------
  Handlungsplan  (DatenTransfer)   := 0 DatenTransfer ,
```

```
   #-------------------------------------------------------------------------
   # Location fuer das Synchronisationstoken. Die Komponente Behaviour ist erst
   # an der Reihe, wenn die Komponente Cognition hier das Token ablegt.
   #-------------------------------------------------------------------------
   TokenVonCOG    (Token)           := 0 Token

DYNAMIC BEHAVIOUR

   #-------------------------------------------------------------------------
   #
   # Ereignis 1 : "Initialisierung"
   #
   #     Zu Beginn der Simulation startet Adam mit
   #     der Aktionsfolge "Explorieren". Das Token fuer die
   #     Synchronisation der Komponenten wird auf der fuer
   #     die Komponente Actor bestimmten Location erzeugt.
   #
   #-------------------------------------------------------------------------
   ON START
   DO
      #----------------------------------------------------------------------
      # Erzeugen des Tokens
      #----------------------------------------------------------------------
      TokenAnACT^  : ADD 1 NEW Token;

      #----------------------------------------------------------------------
      # Erzeugen einer mobilen Komponente mit dem Inhalt "Explorieren" auf
      # der Location "Aktionsfolge"
      #----------------------------------------------------------------------
      Aktionsfolge^: ADD 1 NEW DatenTransfer
         CHANGING
            Aktion^ := 'Explorieren';
         END
   END

   #-------------------------------------------------------------------------
   #
   # Ereignis 2 : "Motivselektion"
   #
   #     Dieses Ereignis untergliedert sich in folgende Ereignisse:
   #
   #     2a: "Wechsel des handlungsleitenden Motivs von 'Wissenserwerb'
   #         auf 'Hunger'"
   #     2b: "Wechsel des handlungsleitenden Motivs von 'Hunger' auf
   #         'Wissenserwerb'"
   #
   #-------------------------------------------------------------------------

   #-------------------------------------------------------------------------
   #
   # Ereignis 2a : "Motivselektion: Wechsel von 'Wissenserwerb' auf 'Hunger'"
   #
   #     Wenn die Intensitaet des Motivs "Hunger" groesser
   #     (oder gleich gross) wird als die Intensitaet des
   #     Motivs Wissenserwerb, so wird Hunger zum "Handlungsleitenden
   #     Motiv".
   #     Die Variable HLM_Wechselt wird gesetzt.
   #     Das Aendern der Variable HLMotiv erfolgt allerdings erst,
   #     wenn die Aktionsfolge "Planen" erzeugt wird.
   #
   #-------------------------------------------------------------------------
   ON ^(MStHunger >= MStWissenserwerb) AND (HLMotiv = 'Wissenserwerb')^
   DO
      HLM_Wechselt^  := TRUE;

      IF Protokoll
      DO
```

```
      DISPLAY ("\n************************************** \n");
      DISPLAY (" T= %f: BEH: Motivwechsel auf 'Hunger'", T);
      DISPLAY (" steht kurz bevor.\n");
      DISPLAY ("************************************** \n");
   END
END

#-------------------------------------------------------------------------
#
# Ereignis 2b : "Motivselektion: Wechsel von 'Hunger' auf 'Wissenswerwerb'"
#
#     Das Motiv "Wissenserwerb" wird nur dann handlungsleitend, wenn die
#     Motivstaerke des Motivs "Hunger" auf den Minimalwert 0 faellt.
#     Damit wird der Beduerfnishierarchie von Maslow Rechnung getragen.
#     Der erste Teil der Ereignisbedingung ist nur dann wichtig, wenn beide
#     Motivstaerken 0 sind. Dann bleibt Hunger handlungsleitend.
#     Die Variable HLM_Wechselt wird gesetzt.
#     Das Aendern der Variable HLMotiv erfolgt allerdings erst,
#     wenn die Aktionsfolge Planen erzeugt wird.
#
#-------------------------------------------------------------------------
ON ^(MStWissenserwerb > MStHunger) AND (MStHunger <= 0 [MStE])
   AND (HLMotiv = 'Hunger')^
DO
   HLM_Wechselt^  := TRUE;

   IF Protokoll
   DO
      DISPLAY ("\n************************************** \n");
      DISPLAY (" T= %f: BEH: Motivwechsel auf 'Wissenserwerb'", T);
      DISPLAY (" steht kurz bevor.\n");
      DISPLAY ("************************************** \n");
   END
END

#-------------------------------------------------------------------------
#
# Ereignis 3 : "Setzen der logischen Variable 'FalleEntdeckt'"
#
#     Immer, wenn das Signal "GefahrenstelleEntdeckt" von der
#     Komponente Cognition kommt, wird eine in Ereignis 4
#     benoetigte logische Hilfsvariable "FalleEntdeckt" gesetzt.
#     Dies hat das Erstellen einer Aktionsfolge "Planen" zur Folge (vgl.
#     Ereignis 4a, dort wird FalleEntdeckt auch wieder zurueckgesetzt).
#
#-------------------------------------------------------------------------
ON GefahrenstelleEntdeckt
DO
   FalleEntdeckt^ := TRUE;
END

#-------------------------------------------------------------------------
# Hilfs-Ereignis: Protokoll-Ausgaben
#-------------------------------------------------------------------------
ON ^(NUMBER (TokenVonCOG) > 0 )^
DO

   #----------------------------------------------------------------------
   # Ausgabe aller Bedingungen
   #----------------------------------------------------------------------
   IF Protokoll
   DO
      DISPLAY ("\n\n T= %f: BEH-V0: Auswahl einer Aktionsfolge:", T);
      DISPLAY ("\n      NUMBER(TokenVonCOG)>0 : TRUE");
      IF (AF_Erstellt = TRUE)
      DO
         DISPLAY ("\n      AF_Erstellt : TRUE");
```

```
END
ELSE
DO
   DISPLAY ("\n      AF_Erstellt : FALSE");
END
IF (FalleEntdeckt = TRUE)
DO
   DISPLAY ("\n      FalleEntdeckt : TRUE");
END
ELSE
DO
   DISPLAY ("\n      FalleEntdeckt : FALSE");
END
IF (InFalle = TRUE)
DO
   DISPLAY ("\n      InFalle : TRUE");
END
ELSE
DO
   DISPLAY ("\n      InFalle : FALSE");
END
IF (HLM_Wechselt = TRUE)
DO
   DISPLAY ("\n      HLM_Wechselt : TRUE");
END
ELSE
DO
   DISPLAY ("\n      HLM_Wechselt : FALSE");
END
DISPLAY ("\n      NUMBER(Aktionsfolge) = %d", NUMBER(Aktionsfolge));
IF (HLMotiv = 'Hunger')
DO
   DISPLAY ("\n      HLMotiv = 'Hunger'");
END
ELSE
DO
   DISPLAY ("\n      HLMotiv = 'Wissenserwerb'");
END
DISPLAY ("\n      Nahrungsmenge = %f", Nahrungsmenge[xAdam][yAdam]);
DISPLAY ("\n      NUMBER(Handlungsplan) = %d", NUMBER(Handlungsplan));
IF (NUMBER(Handlungsplan) > 0)
DO
   DISPLAY ("\n      Naechstes Teilziel: (%d,%d)",
               Handlungsplan:DatenTransfer[1].x,
               Handlungsplan:DatenTransfer[1].y);
   IF (Feldart [Handlungsplan:DatenTransfer[1].x]
               [Handlungsplan:DatenTransfer[1].y] = 'Unbekannt')
   DO
      DISPLAY (" ; Feldart = 'Unbekannt' ");
   END
   ELSIF (Feldart [Handlungsplan:DatenTransfer[1].x]
                  [Handlungsplan:DatenTransfer[1].y] = 'Neutral')
   DO
      DISPLAY (" ; Feldart = 'Neutral' ");
   END
   ELSIF (Feldart [Handlungsplan:DatenTransfer[1].x]
                  [Handlungsplan:DatenTransfer[1].y] = 'Nahrung')
   DO
      DISPLAY (" ; Feldart = 'Nahrung' ");
   END
   ELSIF (Feldart [Handlungsplan:DatenTransfer[1].x]
                  [Handlungsplan:DatenTransfer[1].y] = 'Gefahr')
   DO
      DISPLAY (" ; Feldart = 'Gefahr' ");
   END
END
DISPLAY ("\n      Angst = %f ; AngstSchranke = %f",
```

```
                Angst, AngstSchranke);
   END
END

#------------------------------------------------------------------------
#
# Ereignis 4 : "Auswahl einer Aktionsfolge"
#
#     Immer, wenn das Token von der Komponente Cognition kommt,
#     wird entweder eine neue Aktionsfolge ausgewaehlt (und danach
#     erstellt) oder die alte Aktionsfolge unveraendert gelassen.
#
#     Dieses Ereignis teilt sich in folgende Unterereignisse auf:
#
#     4a: "Auswahl der Aktionsfolge 'Planen'"
#     4b: "Auswahl der Aktionsfolge 'Befreien - Explorieren'"
#     4c: "Auswahl der Aktionsfolge 'Essen - Explorieren'"
#     4d: "Auswahl der Aktionsfolge 'Pruefen - Gehen - Explorieren'"
#     4e: "Auswahl der Aktionsfolge 'Gehen - Explorieren'"
#     4f: "Eine bereits bestehende Aktionsfolge unveraendert lassen"
#
#------------------------------------------------------------------------

#------------------------------------------------------------------------
#
# Ereignis 4a: "Auswahl der Aktionsfolge 'Planen'"
#
#    Dafuer gibt es 3 prinzipielle Moeglichkeiten
#       - Beim "Pruefen" eines unbekannten Feldes wurde eine Falle
#         entdeckt (dann ist FalleEntdeckt = TRUE)
#       - Das Handlungleitende Motiv hat sich geaendert
#         (HLM_Wechselt = TRUE)
#       - Die Aktionsfolge und der Handlungsplan sind beide leer und die
#         Bedingung fuer das "Essen" ist nicht erfuellt.
#------------------------------------------------------------------------
ON ^(NUMBER (TokenVonCOG) > 0)
    AND
    (AF_Erstellt = FALSE)
    AND
    ( (FalleEntdeckt = TRUE)
      OR
      ( (InFalle = FALSE)
        AND
        ( (HLM_Wechselt = TRUE)
          OR
          ( (HLM_Wechselt = FALSE)
            AND
            (NUMBER (Aktionsfolge) = 0)
            AND
            NOT( (HLMotiv = 'Hunger')
                 AND
                 (Nahrungsmenge[xAdam][yAdam] > 0 [EnE])
               )
            AND
            (NUMBER (Handlungsplan) = 0)
          )
        )
      )
    )
   ^
DO

   #------------------------------------------------------------------------
   # Zuruecksetzen von "FalleEntdeckt", falls notwendig
   #------------------------------------------------------------------------
   IF (FalleEntdeckt = TRUE)
   DO
```

```
      FalleEntdeckt^ := FALSE;
   END

   #------------------------------------------------------------------------
   # Falls das "Handlungsleitende Motiv" gewechselt hat, wird die
   # entsprechende Aenderung vorgenommen und die Variable
   # "HLM_Wechselt" zurueckgesetzt.
   #------------------------------------------------------------------------
   IF (HLM_Wechselt = TRUE)
   DO
      #---------------------------------------------------------------------
      # Damit erst dann ein Motiv zum
      # "handlungsleitenden Motiv" wird, wenn deshalb die Aktionsfolge
      # "Planen" erzeugt wird, wird HLMotiv erst hier gesetzt und nicht
      # schon im obigen Ereignis 2a, in dem HLM_Wechselt gesetzt wird.
      #---------------------------------------------------------------------
      IF ((MStWissenserwerb > MStHunger) AND (HLMotiv = 'Hunger'))
      DO
         HLMotiv^       := 'Wissenserwerb';

         IF Protokoll
         DO
            DISPLAY ("\n\n****************************************** \n");
            DISPLAY ("****************************************** \n");
            DISPLAY (" T= %f: BEH: HLM wechselt auf 'Wissenserwerb'.\n", T);
            DISPLAY ("****************************************** \n");
            DISPLAY ("****************************************** \n");
         END
      END
      ELSE
      DO
         HLMotiv^       := 'Hunger';

         IF Protokoll
         DO
            DISPLAY ("\n\n****************************************** \n");
            DISPLAY ("****************************************** \n");
            DISPLAY (" T= %f: BEH: HLM wechselt auf 'Hunger'.\n", T);
            DISPLAY ("****************************************** \n");
            DISPLAY ("****************************************** \n");
         END
      END

      #---------------------------------------------------------------------
      # Zuruecksetzen von "HLM_Wechselt"
      #---------------------------------------------------------------------
      HLM_Wechselt^     := FALSE;
   END

   #------------------------------------------------------------------------
   # Anstossen des Generierens der Aktionsfolge "Planen"
   #------------------------------------------------------------------------
   SIGNAL AF_Pl;

   #------------------------------------------------------------------------
   # Setzen von "AF_Erstellt", damit kein anderes Ereignis mehr schaltet.
   #------------------------------------------------------------------------
   AF_Erstellt^ := TRUE;
END

#---------------------------------------------------------------------------
#
# Ereignis 4b: "Auswahl der Aktionsfolge 'Befreien - Explorieren'"
#
#---------------------------------------------------------------------------
ON ^(NUMBER (TokenVonCOG) > 0 ) AND (AF_Erstellt = FALSE)
    AND (InFalle = TRUE)^
```

```
DO
   #-------------------------------------------------------------------------
   # Anstossen des Generierens der Aktionsfolge "Befreien - Explorieren"
   #-------------------------------------------------------------------------
   SIGNAL AF_BeEx;

   #-------------------------------------------------------------------------
   # Setzen von "AF_Erstellt", damit kein anderes Ereignis mehr schaltet.
   #-------------------------------------------------------------------------
   AF_Erstellt^ := TRUE;
END

#-------------------------------------------------------------------------
#
# Ereignis 4c: "Auswahl der Aktionsfolge 'Essen - Explorieren'"
#
#-------------------------------------------------------------------------
ON ^(NUMBER (TokenVonCOG) > 0 )
    AND (AF_Erstellt = FALSE)
    AND (FalleEntdeckt = FALSE)
    AND (InFalle = FALSE)
    AND (HLM_Wechselt = FALSE)
    AND (NUMBER (Aktionsfolge) = 0)
    AND (HLMotiv = 'Hunger')
    AND (Nahrungsmenge[xAdam][yAdam] > 0 [EnE])^
DO
   #-------------------------------------------------------------------------
   # Anstossen des Generierens der Aktionsfolge
   #-------------------------------------------------------------------------
   SIGNAL AF_EsEx;

   #-------------------------------------------------------------------------
   # Setzen von "AF_Erstellt", damit kein anderes Ereignis mehr schaltet.
   #-------------------------------------------------------------------------
   AF_Erstellt^ := TRUE;
END

#-------------------------------------------------------------------------
#
# Ereignis 4d: "Auswahl der Aktionsfolge 'Pruefen - Gehen - Explorieren'"
#
#-------------------------------------------------------------------------
ON ^(NUMBER (TokenVonCOG) > 0 )
    AND (AF_Erstellt = FALSE)
    AND (FalleEntdeckt = FALSE)
    AND (InFalle = FALSE)
    AND (HLM_Wechselt = FALSE)
    AND (NUMBER (Aktionsfolge) = 0)
    AND NOT((HLMotiv = 'Hunger') AND
            (Nahrungsmenge[xAdam][yAdam] > 0 [EnE]))
    AND (NUMBER (Handlungsplan) > 0)
    AND (Feldart[Handlungsplan:DatenTransfer[1].x]
                [Handlungsplan:DatenTransfer[1].y] = 'Unbekannt')
    AND (Angst > AngstSchranke)^
DO
   #-------------------------------------------------------------------------
   # Anstossen des Generierens der Aktionsfolge
   #-------------------------------------------------------------------------
   SIGNAL AF_PrGeEx;

   #-------------------------------------------------------------------------
   # Setzen von "AF_Erstellt", damit kein anderes Ereignis mehr schaltet.
   #-------------------------------------------------------------------------
   AF_Erstellt^ := TRUE;
END
```

```
#-------------------------------------------------------------------------
#
# Ereignis 4e: "Auswahl der Aktionsfolge 'Gehen - Explorieren'"
#
#-------------------------------------------------------------------------
ON ^(NUMBER (TokenVonCOG) > 0 )
    AND (AF_Erstellt = FALSE)
    AND (FalleEntdeckt = FALSE)
    AND (InFalle = FALSE)
    AND (HLM_Wechselt = FALSE)
    AND (NUMBER (Aktionsfolge) = 0)
    AND NOT((HLMotiv = 'Hunger') AND
            (Nahrungsmenge[xAdam][yAdam] > 0 [EnE]))
    AND (NUMBER (Handlungsplan) > 0)
    AND NOT((Feldart[Handlungsplan:DatenTransfer[1].x]
                [Handlungsplan:DatenTransfer[1].y] = 'Unbekannt')
            AND (Angst > AngstSchranke))^
DO
   #-----------------------------------------------------------------------
   # Anstossen des Generierens der Aktionsfolge
   #-----------------------------------------------------------------------
   SIGNAL AF_GeEx;

   #-----------------------------------------------------------------------
   # Setzen von "AF_Erstellt", damit kein anderes Ereignis mehr schaltet.
   #-----------------------------------------------------------------------
   AF_Erstellt^ := TRUE;
END

#-------------------------------------------------------------------------
#
# Ereignis 4f: "Eine bereits bestehende Aktionsfolge unveraendert lassen"
#
#-------------------------------------------------------------------------
ON ^(NUMBER (TokenVonCOG) > 0 )
    AND (AF_Erstellt = FALSE)
    AND (FalleEntdeckt = FALSE)
    AND (InFalle = FALSE)
    AND (HLM_Wechselt = FALSE)
    AND (NUMBER (Aktionsfolge) > 0)^
DO
   #-----------------------------------------------------------------------
   # Das Weitergeben des Tokens an die Komponente Actor in die Wege leiten
   #-----------------------------------------------------------------------
   SIGNAL TokenAnActor;

   #-----------------------------------------------------------------------
   # Setzen von "AF_Erstellt", damit kein anderes Ereignis mehr schaltet.
   #-----------------------------------------------------------------------
   AF_Erstellt^ := TRUE;

   IF Protokoll
   DO
      DISPLAY("\n T= %f: BEH: Aktionsfolge bleibt unveraendert.\n", T);
   END
END

#-------------------------------------------------------------------------
#
# Ereignis 5: "Erzeugen einer neuen Aktionsfolge"
#
#     Dieses Ereignis teilt sich in folgende Unterereignisse auf:
#
#     5a: "Erzeugen der Aktionsfolge 'Planen'"
#     5b: "Erzeugen der Aktionsfolge 'Befreien - Explorieren'"
#     5c: "Erzeugen der Aktionsfolge 'Essen - Explorieren'"
#     5d: "Erzeugen der Aktionsfolge 'Pruefen - Gehen - Explorieren'"
```

```
#     5e: "Erzeugen der Aktionsfolge 'Gehen - Explorieren'"
#
# Diese Ereignisse waeren nicht unbedingt als Einzelereignisse noetig. Sie
# koennten auch im obigen Ereignis 4 mitbehandelt werden.
# Dadurch verliert man allerdings einiges an Uebersichtlichkeit.
#
#-------------------------------------------------------------------------

#-------------------------------------------------------------------------
#
# Ereignis 5a: "Erzeugen der Aktionsfolge 'Planen':
#
#    Dieses Ereignis teilt sich in folgende zwei Unterereignisse auf:
#
#    5a-1: Zunaechst wird die zuvor gueltige Aktionsfolge geloescht.
#    5a-2: Danach wird die Generierung der neuen Aktionsfolge angestossen.
#
#-------------------------------------------------------------------------

#-------------------------------------------------------------------------
#
# Ereignis 5a-1: "Loeschen der aktuellen Aktionsfolge"
#
#-------------------------------------------------------------------------
ON AF_Pl
DO
   #----------------------------------------------------------------------
   # Loeschen aller auf der Location "Aktionsfolge" befindlichen mobilen
   # Komponenten.
   #----------------------------------------------------------------------
   Aktionsfolge^: REMOVE DatenTransfer{ALL};

   #----------------------------------------------------------------------
   # Anstossen der wirklichen Generierung der neuen Aktionsfolge "Planen".
   # Dies wird hier nicht sofort gemacht, damit sichergestellt ist, dass
   # die Aktionsfolge auch wirklich leer ist, bevor der neue Inhalt
   # hineingeschrieben wird.
   #----------------------------------------------------------------------
   SIGNAL NeuPlanen;
END

#-------------------------------------------------------------------------
#
# Ereignis 5a-2: "Generieren der Aktionsfolge 'Planen'"
#
#-------------------------------------------------------------------------
ON NeuPlanen
DO
   #----------------------------------------------------------------------
   # Erzeugen der Ausfuehrungsanordnung fuer die Aktion 'Planen'
   #----------------------------------------------------------------------
   Aktionsfolge^: ADD 1 NEW DatenTransfer
      CHANGING
         Aktion^     := 'Planen';
      END

   #----------------------------------------------------------------------
   # Das Weitergeben des Tokens an die Komponente Actor in die Wege leiten
   #----------------------------------------------------------------------
   SIGNAL TokenAnActor;

   IF Protokoll
   DO
      DISPLAY("\n T= %f: BEH: Aktionsfolge erstellt: ", T);
      DISPLAY("'Planen' \n");
   END
END
```

```
#-------------------------------------------------------------------------
#
# Ereignis 5b: "Erzeugen der Aktionsfolge 'Befreien - Explorieren':
#
#-------------------------------------------------------------------------
ON AF_BeEx
DO
   IF (NUMBER (Aktionsfolge) > 0)
   DO
      DISPLAY("\n T = %f : Fehler in der ",T);
      DISPLAY("Komponente Behaviour:");
      DISPLAY(" Aktionsfolge vor 'Befreien' nicht leer.\n\n");
      SIGNAL STOP;
   END

   #-------------------------------------------------------------------------
   # Erzeugen der Ausfuehrungsanordnung fuer die Aktion 'Befreien'
   #-------------------------------------------------------------------------
   Aktionsfolge^: ADD 1 NEW DatenTransfer
      CHANGING
         Aktion^  := 'Befreien';
      END

   #-------------------------------------------------------------------------
   # Erzeugen der Ausfuehrungsanordnung fuer die Aktion 'Explorieren'
   #-------------------------------------------------------------------------
   Aktionsfolge^: ADD 1 NEW DatenTransfer
      CHANGING
         Aktion^  := 'Explorieren';
      END

   #-------------------------------------------------------------------------
   # Das Weitergeben des Tokens an die Komponente Actor in die Wege leiten
   #-------------------------------------------------------------------------
   SIGNAL TokenAnActor;

   IF Protokoll
   DO
      DISPLAY("\n T= %f: BEH: Aktionsfolge erstellt: ", T);
      DISPLAY("'Befreien-Explorieren' \n");
   END
END

#-------------------------------------------------------------------------
#
# Ereignis 5c: "Erzeugen der Aktionsfolge 'Essen - Explorieren':
#
#-------------------------------------------------------------------------
ON AF_EsEx
DO
   IF (NUMBER (Aktionsfolge) > 0)
   DO
      DISPLAY("\n T = %f : Fehler in der ",T);
      DISPLAY("Komponente Verhalten:");
      DISPLAY(" Aktionsfolge vor Essen nicht leer.\n\n");
      SIGNAL STOP;
   END

   #-------------------------------------------------------------------------
   # Erzeugen der Ausfuehrungsanordnung fuer die Aktion 'Essen'
   #-------------------------------------------------------------------------
   Aktionsfolge^: ADD 1 NEW DatenTransfer
      CHANGING
         Aktion^  := 'Essen';
         #-------------------------------------------------------------------
         # Die wirklich gegessene Nahrungsmenge, und damit auch die Dauer
         # des Essens, ergibt sich aus dem Minimum der folgenden Werte:
```

```
         # - zum optimalen Energiepegel fehlende Menge an Energie
         # - auf dem Nahrungsfeld vorhandene Menge an Nahrung
         #
         # Anmerkung: Hier geschieht ein kleiner Fehler, den man leicht
         # beheben kann. Adam kann naemlich die gesamte Nahrungsmenge des
         # Nahrungsfeldes essen, obwohl in der Komponente Environment die
         # Nahrungsmenge nach dem Essen nicht auf 0 sondern auf NahrungsMIN
         # gesetzt wird!!
         #-------------------------------------------------------------------
         Nahrungsmenge^ := MIN (FehlendeEnergie,
                                Nahrungsmenge[xAdam][yAdam]);
      END

   #-----------------------------------------------------------------------
   # Erzeugen der Ausfuehrungsanordnung fuer die Aktion 'Explorieren'
   #-----------------------------------------------------------------------
   Aktionsfolge^: ADD 1 NEW DatenTransfer
      CHANGING
         Aktion^  := 'Explorieren';
      END

   #-----------------------------------------------------------------------
   # Das Weitergeben des Tokens an die Komponente Actor in die Wege leiten
   #-----------------------------------------------------------------------
   SIGNAL TokenAnActor;

   IF Protokoll
   DO
      DISPLAY("\n T= %f: BEH: Aktionsfolge erstellt: ", T);
      DISPLAY("'Essen-Explorieren' \n");
   END
END

#-----------------------------------------------------------------------
#
# Ereignis 5d: "Erzeugen der Aktionsfolge 'Pruefen - Gehen - Explorieren':
#
#-----------------------------------------------------------------------
ON AF_PrGeEx
DO
   IF (NUMBER (Aktionsfolge) > 0)
   DO
      DISPLAY("\n T = %f : Fehler in der ",T);
      DISPLAY("Komponente Behaviour:");
      DISPLAY(" Aktionsfolge vor PrGeEx nicht leer.\n\n");
      SIGNAL STOP;
   END

   #-----------------------------------------------------------------------
   # Erzeugen der Ausfuehrungsanordnung fuer die Aktion 'Pruefen'
   #-----------------------------------------------------------------------
   Aktionsfolge^: ADD 1 NEW DatenTransfer
      CHANGING
         Aktion^  := 'Pruefen';
         x^       := Handlungsplan:DatenTransfer[1].x;
         y^       := Handlungsplan:DatenTransfer[1].y;
      END

   #-----------------------------------------------------------------------
   # Erzeugen der Ausfuehrungsanordnung fuer die Aktion 'Gehen'
   #-----------------------------------------------------------------------
   Aktionsfolge^: ADD 1 NEW DatenTransfer
      CHANGING
         Aktion^  := 'Gehen';
         x^       := Handlungsplan:DatenTransfer[1].x;
         y^       := Handlungsplan:DatenTransfer[1].y;
      END
```

```
   #------------------------------------------------------------------------
   # Erzeugen der Ausfuehrungsanordnung fuer die Aktion 'Explorieren'
   #------------------------------------------------------------------------
   Aktionsfolge^: ADD 1 NEW DatenTransfer
      CHANGING
         Aktion^  := 'Explorieren';
      END

   #------------------------------------------------------------------------
   # Das Weitergeben des Tokens an die Komponente Actor in die Wege leiten
   #------------------------------------------------------------------------
   SIGNAL TokenAnActor;

   IF Protokoll
   DO
      DISPLAY("\n T= %f: BEH: Aktionsfolge erstellt: ", T);
      DISPLAY("'Pruefen - Gehen - Explorieren' \n");
   END

END

#------------------------------------------------------------------------
#
# Ereignis 5e: "Erzeugen der Aktionsfolge 'Gehen - Explorieren':
#
#------------------------------------------------------------------------
ON AF_GeEx
DO
   IF (NUMBER (Aktionsfolge) > 0)
   DO
      DISPLAY("\n T = %f : Fehler in der ",T);
      DISPLAY("Komponente Behaviour:");
      DISPLAY(" Aktionsfolge vor GeEx nicht leer.\n\n");
      SIGNAL STOP;
   END

   #------------------------------------------------------------------------
   # Erzeugen der Ausfuehrungsanordnung fuer die Aktion 'Gehen'
   #------------------------------------------------------------------------
   Aktionsfolge^: ADD 1 NEW DatenTransfer
      CHANGING
         Aktion^  := 'Gehen';
         x^       := Handlungsplan:DatenTransfer[1].x;
         y^       := Handlungsplan:DatenTransfer[1].y;
      END

   #------------------------------------------------------------------------
   # Erzeugen der Ausfuehrungsanordnung fuer die Aktion 'Explorieren'
   #------------------------------------------------------------------------
   Aktionsfolge^: ADD 1 NEW DatenTransfer
      CHANGING
         Aktion^  := 'Explorieren';
      END

   #------------------------------------------------------------------------
   # Das Weitergeben des Tokens an die Komponente Actor in die Wege leiten
   #------------------------------------------------------------------------
   SIGNAL TokenAnActor;

   IF Protokoll
   DO
      DISPLAY("\n T= %f: BEH: Aktionsfolge erstellt: ", T);
      DISPLAY("'Gehen-Explorieren' \n");
   END
END
```

```
   #-------------------------------------------------------------------------
   #
   # Ereignis 6:  "Weitergeben des Tokens an die Komponente Actor"
   #
   #     Wenn die Komponente Behaviour eine neue Aktionsfolge erzeugt
   #     hat bzw. entschieden hat, dass eine bestehende Aktionsfolge
   #     weiterbenutzt wird, erhaelt die Komponente Actor das Token.
   #
   #-------------------------------------------------------------------------
   ON TokenAnActor
   DO
      IF (NUMBER (TokenVonCOG) > 0)
      DO
         #----------------------------------------------------------------------
         # Verschieben des Tokens von der Location "TokenVonCOG" auf die
         # Location "TokenAnACT"
         #----------------------------------------------------------------------
         TokenAnACT^  : FROM TokenVonCOG GET Token[1];

         #----------------------------------------------------------------------
         # AF_Erstellt muss wieder auf FALSE zurueckgesetzt werden, da sonst,
         # wenn das Token wieder kommt, keines der Ereignisse zur Auswahl
         # einer Aktionsfolge schalten kann.
         #----------------------------------------------------------------------
         AF_Erstellt^ := FALSE;
      END

      ELSE
      DO
         DISPLAY("\n T = %f : Fehler in der Komponente Behaviour:",T);
         DISPLAY(" Token soll an Actor weitergegeben werden und ist");
         DISPLAY(" nicht da.\n\n");
         SIGNAL STOP;
      END
   END

END OF Behaviour
```

D.3.8 Die Komponente *Actor*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell ADAM
#
#  Name:             Actor
#
#  Art:              Basiskomponente
#
#  Version:          1
#
#  Beschreibung:     Ausfuehren der zeitverbrauchenden Aktionen von Adam
#                    entsprechend den in der Aktionsfolge (in der Komponente
#                    Behaviour) stehenden Ausfuehrungsanordnungen.
#                    Der "Zustand" von Adam wird entsprechend der Aktion
#                    gesetzt.
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Behaviour an Actor
#       + Aktionsfolge
#       + TokenVonBEH
#
#    - Physis an Actor
#       + Energie_essen
```

```
#
#    - Actor an Physis
#       + Zustand
#
#    - Actor an Cognition
#       + DatAnCOG
#       + TokenAnCOG
#
#    - Actor an Sensor
#       + DatAnSEN
#       + TokenAnSEN
#
#    - Actor an Adam (von dort an Environment)
#       + DatAnENV
#
#-------------------------------------------------------------------------------

BASIC COMPONENT Actor

MOBILE SUBCOMPONENTS OF CLASS DatenTransfer, Token

USE OF UNITS

   #-----------------------------------------------------------------------------
   # Energie-Einheit
   #-----------------------------------------------------------------------------
   UNIT [EnE] = BASIS

   #-----------------------------------------------------------------------------
   # Als Zeiteinheit wird eine "Stunde" festgelegt.
   #-----------------------------------------------------------------------------
   TIMEUNIT   = [h]

LOCAL DEFINITIONS

   #-----------------------------------------------------------------------------
   # Adam kann sich, abhaengig davon, welche Aktion er gerade ausfuehrt, in
   # einem der folgenden Zustaende befinden:
   #-----------------------------------------------------------------------------
   VALUE SET ZUSTAENDE : ('Planend', 'Explorierend', 'Pruefend',
                          'Gehend', 'Befreiend', 'Essend')

DECLARATION OF ELEMENTS

 CONSTANTS

   #-----------------------------------------------------------------------------
   # konstanter Zeitverbrauch fuer die Aktion 'Planen'.
   #-----------------------------------------------------------------------------
   TPlanen           (REAL [h])      :=  0.2 [h] ,

   #-----------------------------------------------------------------------------
   # konstanter Zeitverbrauch fuer die Aktion 'Explorieren'.
   #-----------------------------------------------------------------------------
   TExplorieren      (REAL [h])      :=  0.1 [h] ,

   #-----------------------------------------------------------------------------
   # konstanter Zeitverbrauch fuer die Aktion 'Pruefen'.
   #-----------------------------------------------------------------------------
   TPruefen          (REAL [h])      :=  0.2 [h] ,

   #-----------------------------------------------------------------------------
   # konstanter Zeitverbrauch fuer die Aktion 'Gehen'.
   #-----------------------------------------------------------------------------
   TGehen            (REAL [h])      :=  0.2 [h] ,
```

```
  #-------------------------------------------------------------------------
  # konstanter Zeitverbrauch fuer die Aktion 'Befreien'.
  #-------------------------------------------------------------------------
  TBefreien         (REAL [h])      :=  2.0 [h]

STATE VARIABLES

 DISCRETE

  #-------------------------------------------------------------------------
  # Zustand von Adam: Der Zustand ist abhaengig von der Aktion, die er gerade
  # ausfuehrt.
  #-------------------------------------------------------------------------
  Zustand           (ZUSTAENDE)    := 'Explorierend' ,

  #-------------------------------------------------------------------------
  # Hilfsvariable: Wird benoetigt, damit das Ereignis 1 nicht staendig
  # "schaltet".
  #-------------------------------------------------------------------------
  NaechsteAktion    (LOGICAL)      := TRUE ,

  #-------------------------------------------------------------------------
  # Codierung der Aktionsfolge fuer die Animation:
  # 0: Explorieren, 1: Gehen - Explorieren, 2: Pruefen - Gehen - Explorieren,
  # 3: Planen, 4: Klettern - Explorieren, 5: Essen - Explorieren
  #-------------------------------------------------------------------------
  Ani_Aktionsfolge  (INTEGER)      := 0

 CONTINUOUS

  #-------------------------------------------------------------------------
  # In TNext steht immer der Zeitpunkt, an dem die jeweils gerade
  # ausgefuehrte Aktion beendet ist. (Die fuer diese Aktion benoetigte Zeit
  # laeuft zum Zeitpunkt TNext ab.)
  #-------------------------------------------------------------------------
  TNext             (REAL [h])     := 0.0 [h]

SENSOR VARIABLES

 DISCRETE

  #-------------------------------------------------------------------------
  # Aus der Physis: Energie, die im Zustand 'Essend' pro Zeiteinheit
  # zugefuehrt wird.
  #-------------------------------------------------------------------------
  Energie_essen     (REAL [EnE])   := 20.0 [EnE] ,

  #-------------------------------------------------------------------------
  # Protokoll = TRUE: Textausgabe am Bildschirm aktiviert.
  #-------------------------------------------------------------------------
  Protokoll         (LOGICAL)      := FALSE

TRANSITION INDICATORS

  #-------------------------------------------------------------------------
  # Indikator, der anzeigt, dass das Token an die Komponente Sensor
  # weitergegeben werden soll.
  #-------------------------------------------------------------------------
  TokenAnSensor ,

  #-------------------------------------------------------------------------
  # Indikator, der anzeigt, dass das Token an die Komponente Cognition
  # weitergegeben werden soll.
  #-------------------------------------------------------------------------
  TokenAnCognition

LOCATIONS
```

```
   #-------------------------------------------------------------------------
   # Auf dieser Location werden Daten, die fuer die Komponente Sensor bestimmt
   # sind, abgelegt.
   #-------------------------------------------------------------------------
   DatAnSEN       (DatenTransfer)   := 0 DatenTransfer ,

   #-------------------------------------------------------------------------
   # Auf dieser Location werden Daten, die fuer die Komponente Environment
   # bestimmt sind, abgelegt.
   #-------------------------------------------------------------------------
   DatAnENV       (DatenTransfer)   := 0 DatenTransfer ,

   #-------------------------------------------------------------------------
   # Auf dieser Location werden Daten, die fuer die Komponente Cognition
   # bestimmt sind, abgelegt.
   #-------------------------------------------------------------------------
   DatAnCOG       (DatenTransfer)   := 0 DatenTransfer ,

   #-------------------------------------------------------------------------
   # Hier wird das Token abgelegt, wenn der Sensor eine Aktion ausfuehren soll.
   #-------------------------------------------------------------------------
   TokenAnSEN     (Token)           := 0 Token ,

   #-------------------------------------------------------------------------
   # Hier wird das Token abgelegt, wenn die Komponente Cognition eine Aktion
   # ausfuehren soll.
   #-------------------------------------------------------------------------
   TokenAnCOG     (Token)           := 0 Token

 SENSOR LOCATIONS

   #-------------------------------------------------------------------------
   # Aus der Komponente Behaviour: Die Aktionsfolge von Adam
   #-------------------------------------------------------------------------
   Aktionsfolge   (DatenTransfer)   := 0 DatenTransfer ,

   #-------------------------------------------------------------------------
   # Auf dieser Location erwartet die Komponente Actor das Token.
   #-------------------------------------------------------------------------
   TokenVonBEH    (Token)           := 0 Token
DYNAMIC BEHAVIOUR

   #-------------------------------------------------------------------------
   #
   # Ereignis 1: "Bearbeiten der ersten Ausfuehrungsanordnung der Aktionsfolge"
   #
   #     Immer, wenn das Token da ist (die Aktionsfolge also nicht leer
   #     ist) und NaechsteAktion gesetzt ist (also die Ausfuehrung einer
   #     anderen Aktion abgeschlossen ist), wird die erste, in der
   #     Aktionsfolge stehende Aktion ausgefuehrt.
   #     NaechsteAktion wird dabei also benoetigt, damit das Ereignis nicht
   #     staendig schaltet, sondern nur, wenn das Token gerade von der
   #     der Komponente Behaviour gekommen ist oder eine Aktion
   #     vom Actor gerade ausgefuehrt worden ist und, ohne dass das Token
   #     an eine andere Komponente weitergegeben wird, die naechste
   #     Aktion der Aktionsfolge ausgefuehrt werden soll.
   #
   #-------------------------------------------------------------------------

   WHENEVER ((NUMBER (TokenVonBEH) > 0 ) AND (NaechsteAktion = TRUE))
   DO

      #----------------------------------------------------------------------
      # Sicherheitsabfrage: Wenn das Token da ist, muss auch auf der Location
      # Aktionsfolge mindestens eine mobile Komponente liegen.
      # Falls nicht: Simulationsende mit Fehlermeldung.
      #----------------------------------------------------------------------
```

```
IF (NUMBER (Aktionsfolge) <= 0)
DO
   DISPLAY (" T = %f: Programmfehler!! In der Komponente Actor", T);
   DISPLAY (" ist das Token da, aber die Aktionsfolge ist leer.");
   SIGNAL STOP;
END

#-------------------------------------------------------------------------
# Nur fuer Animation: Codieren der Aktionsfolge in Ani_Aktionsfolge, so
# dass das Animationsprogramm die Aktionsfolge anzeigen kann, ohne die
# mobilen Komponenten, die sich auf der Location Aktionsfolge befinden,
# untersuchen zu muessen.
# Begleitend findet eine Fehlerueberpruefung statt.
#-------------------------------------------------------------------------
ELSIF (NUMBER (Aktionsfolge) = 1)
DO
   IF (Aktionsfolge:DatenTransfer[1].Aktion = 'Explorieren')
   DO
      Ani_Aktionsfolge^ := 0;
   END
   ELSIF (Aktionsfolge:DatenTransfer[1].Aktion = 'Planen')
   DO
      Ani_Aktionsfolge^ := 3;
   END
   ELSE
   DO
      DISPLAY (" T = %f: Programmfehler!! In der Komponente Actor", T);
      DISPLAY (" hat die Aktionsfolge nur ein Element und ist nicht");
      DISPLAY (" 'Planen' oder 'Explorieren'.");
      SIGNAL STOP;
   END
END
ELSIF (NUMBER (Aktionsfolge) = 2)
DO
   IF ((Aktionsfolge:DatenTransfer[1].Aktion = 'Gehen') AND
       (Aktionsfolge:DatenTransfer[2].Aktion = 'Explorieren'))
   DO
      Ani_Aktionsfolge^ := 1;
   END
   ELSIF ((Aktionsfolge:DatenTransfer[1].Aktion = 'Befreien') AND
          (Aktionsfolge:DatenTransfer[2].Aktion = 'Explorieren'))
   DO
      Ani_Aktionsfolge^ := 4;
   END
   ELSIF ((Aktionsfolge:DatenTransfer[1].Aktion = 'Essen') AND
          (Aktionsfolge:DatenTransfer[2].Aktion = 'Explorieren'))
   DO
      Ani_Aktionsfolge^ := 5;
   END
   ELSE
   DO
      DISPLAY (" T = %f: Programmfehler!! In der Komponente Actor", T);
      DISPLAY (" hat die Aktionsfolge zwei Elemente und ist nicht");
      DISPLAY (" 'Gehen - Explorieren', 'Befreien - Explorieren'");
      DISPLAY (" oder 'Essen - Explorieren'.");
      SIGNAL STOP;
   END
END
ELSIF (NUMBER (Aktionsfolge) = 3)
DO
   IF ((Aktionsfolge:DatenTransfer[1].Aktion = 'Pruefen') AND
       (Aktionsfolge:DatenTransfer[2].Aktion = 'Gehen') AND
       (Aktionsfolge:DatenTransfer[3].Aktion = 'Explorieren'))
   DO
      Ani_Aktionsfolge^ := 2;
   END
   ELSE
```

```
   DO
      DISPLAY (" T = %f: Programmfehler!! In der Komponente Actor", T);
      DISPLAY (" hat die Aktionsfolge drei Elemente und ist nicht");
      DISPLAY (" 'Pruefen - Gehen - Explorieren'.");
      SIGNAL STOP;
   END
END
ELSE
DO
   DISPLAY (" T = %f: Programmfehler!! In der Komponente Actor", T);
   DISPLAY (" hat die Aktionsfolge mehr als drei Elemente.");
   SIGNAL STOP;
END

#-------------------------------------------------------------------------
# Die als naechstes auszufuehrende Aktion ist "Planen"
#-------------------------------------------------------------------------
IF (Aktionsfolge:DatenTransfer[1].Aktion = 'Planen')
DO

   #-------------------------------------------------------------------
   # Der Zustand von Adam wechselt entsprechend
   #-------------------------------------------------------------------
   Zustand^ := 'Planend';

   #-------------------------------------------------------------------
   # Der Zeitverbrauch der Aktion wird festgelegt.
   #-------------------------------------------------------------------
   TNext^   := T + TPlanen;

   IF Protokoll
   DO
      DISPLAY ("\n T= %f: ACT: Zustand := 'Planend' \n", T);
   END
END

#-------------------------------------------------------------------------
# Die als naechstes auszufuehrende Aktion ist "Pruefen"
#-------------------------------------------------------------------------
ELSIF (Aktionsfolge:DatenTransfer[1].Aktion = 'Pruefen')
DO

   #-------------------------------------------------------------------
   # Der Zustand von Adam wechselt entsprechend
   #-------------------------------------------------------------------
   Zustand^ := 'Pruefend';

   #-------------------------------------------------------------------
   # Der Zeitverbrauch der Aktion wird festgelegt.
   #-------------------------------------------------------------------
   TNext^   := T + TPruefen;

   IF Protokoll
   DO
      DISPLAY ("\n T= %f: ACTOR: Zustand := 'Pruefend' \n", T);
   END
END

#-------------------------------------------------------------------------
# Die als naechstes auszufuehrende Aktion ist "Explorieren"
#-------------------------------------------------------------------------
ELSIF (Aktionsfolge:DatenTransfer[1].Aktion = 'Explorieren')
DO

   #-------------------------------------------------------------------
   # Der Zustand von Adam wechselt entsprechend
   #-------------------------------------------------------------------
```

```
   Zustand^ := 'Explorierend';

   #-------------------------------------------------------------------------
   # Der Zeitverbrauch der Aktion wird festgelegt.
   #-------------------------------------------------------------------------
   TNext^   := T + TExplorieren;

   IF Protokoll
   DO
      DISPLAY ("\n T= %f: ACT: Zustand := 'Explorierend'\n"
               , T);
   END
END

#-------------------------------------------------------------------------
# Die als naechstes auszufuehrende Aktion ist "Gehen"
#-------------------------------------------------------------------------
ELSIF (Aktionsfolge:DatenTransfer[1].Aktion = 'Gehen')
DO

   #-------------------------------------------------------------------------
   # Der Zustand von Adam wechselt entsprechend
   #-------------------------------------------------------------------------
   Zustand^ := 'Gehend';

   #-------------------------------------------------------------------------
   # Der Zeitverbrauch der Aktion wird festgelegt.
   #-------------------------------------------------------------------------
   TNext^   := T + TGehen;

   IF Protokoll
   DO
      DISPLAY ("\n T= %f: ACT: Zustand := 'Gehend' \n", T);
   END
END

#-------------------------------------------------------------------------
# Die als naechstes auszufuehrende Aktion ist "Befreien"
#-------------------------------------------------------------------------
ELSIF (Aktionsfolge:DatenTransfer[1].Aktion = 'Befreien')
DO

   #-------------------------------------------------------------------------
   # Der Zustand von Adam wechselt entsprechend
   #-------------------------------------------------------------------------
   Zustand^ := 'Befreiend';

   #-------------------------------------------------------------------------
   # Der Zeitverbrauch der Aktion wird festgelegt.
   #-------------------------------------------------------------------------
   TNext^   := T + TBefreien;

   IF Protokoll
   DO
      DISPLAY ("\n T= %f: ACT: Zustand := 'Befreiend' \n", T);
   END
END

#-------------------------------------------------------------------------
# Die als naechstes auszufuehrende Aktion ist "Essen"
#-------------------------------------------------------------------------
ELSIF (Aktionsfolge:DatenTransfer[1].Aktion = 'Essen')
DO

   #-------------------------------------------------------------------------
   # Der Zustand von ADAM wechselt entsprechend
   #-------------------------------------------------------------------------
```

```
      Zustand^ := 'Essend';

      #------------------------------------------------------------------------
      # Der Zeitverbrauch der Aktion wird festgelegt.
      # Der Zeitverbrauch fuer das 'Essen' ist abhaengig davon, um wieviel
      # die Energie pro Zeiteinheit im Zustand "Essend" in der Komponente
      # Physis zunimmt (Diese Menge an Nahrungseinheiten ist Energie_essen,
      # vgl. Physis!)!!
      # Also werden zum Essen (Nahrungsmenge/Energie_essen) Zeiteinheiten
      # benoetigt.
      # Der Zeitverbrauch der Aktion 'Essen' ist demzufolge abhaengig von
      # der Nahrungsmenge, die gegessen wird.
      # Umso groesser Energie_essen ist, desto schneller ist der Essvorgang
      # abgeschlossen.
      # Der Faktor 1.0 [h] wird wegen der Einheitenueberpruefung von
      # Simplex benoetigt.
      #------------------------------------------------------------------------
      TNext^ := T + 1.0 [h] *
              (Aktionsfolge:DatenTransfer[1].Nahrungsmenge / Energie_essen);
      IF Protokoll
      DO
         DISPLAY ("\n T= %f: ACT: Zustand := 'Essend'\n", T);
      END
   END

   #---------------------------------------------------------------------------
   # Verhindern, dass eine weitere Ausfuehrungsanordnung aus der
   # Aktionsfolge bearbeitet wird, bevor die jetzt ausgewaehlte Aktion
   # komplett ausgefuehrt worden ist.
   #---------------------------------------------------------------------------
   NaechsteAktion^ := FALSE;

END

#------------------------------------------------------------------------------
#
# Ereignis 2: "Zeitbedarf einer Aktion ist verstrichen"
#
#     Erst nach dem Vestreichen der von einer Aktion benoetigten
#     Zeit wird die Aktion wirklich ausgefuehrt. Dazu wird an die
#     ausfuehrende Komponente ein entsprechender Auftrag gesendet.
#
#------------------------------------------------------------------------------

ON ^T >= TNext^
DO

   #---------------------------------------------------------------------------
   # Die auszufuehrende Aktion ist "Planen":
   # Nachricht an die Komponente Cognition
   #---------------------------------------------------------------------------
   IF (Zustand = 'Planend')
   DO
      #------------------------------------------------------------------------
      # Die mobile Komponente wird aus der Aktionsfolge entfernt auf die
      # fuer die Komponente Cognition bestimmte Location gelegt.
      #------------------------------------------------------------------------
      DatAnCOG^: FROM Aktionsfolge GET DatenTransfer[1];

      #------------------------------------------------------------------------
      # Das Weitergeben des Tokens an die Komponente Cognition wird in die
      # Wege geleitet.
      #------------------------------------------------------------------------
      SIGNAL TokenAnCognition;
   END
```

```
   #------------------------------------------------------------------------
   # Die auszufuehrende Aktion ist "Explorieren" oder "Pruefen":
   # Nachricht an die Komponente Sensor
   #------------------------------------------------------------------------
   ELSIF ((Zustand = 'Explorierend') OR (Zustand = 'Pruefend'))
   DO
      #---------------------------------------------------------------------
      # Die mobile Komponente wird aus der Aktionsfolge entfernt auf die
      # fuer den Sensor bestimmte Location gelegt.
      #---------------------------------------------------------------------
      DatAnSEN^: FROM Aktionsfolge GET DatenTransfer[1];

      #---------------------------------------------------------------------
      # Das Weitergeben des Tokens an die Komponente Sensor wird in die
      # Wege geleitet.
      #---------------------------------------------------------------------
      SIGNAL TokenAnSensor;
   END

   #------------------------------------------------------------------------
   # Die auszufuehrende Aktion ist eine externe Aktion, also "Gehen"
   # "Befreien" oder "Essen":
   # Nachricht an die Komponente Environment
   #------------------------------------------------------------------------
   ELSIF ((Zustand = 'Gehend') OR (Zustand = 'Befreiend')
          OR (Zustand = 'Essend'))
   DO
      #---------------------------------------------------------------------
      # Die mobile Komponente wird aus der Aktionsfolge entfernt und auf
      # die fuer die Komponente Environment bestimmte Location gelegt.
      #---------------------------------------------------------------------
      DatAnENV^: FROM Aktionsfolge GET DatenTransfer[1];

      #---------------------------------------------------------------------
      # Da es sich um eine externe Aktion handelt:
      # Die Komponente Actor bleibt weiter am Ball. Das Token wird also
      # nicht weitergegeben. NaechsteAktion wechselt auf 'TRUE', damit
      # Ereignis 1 wieder schalten kann und so die naechste Aktion der
      # Aktionsfolge bearbeitet wird.
      #---------------------------------------------------------------------
      NaechsteAktion^ := TRUE;
   END

END

#------------------------------------------------------------------------
#
# Ereignis 3: "Weitergeben des Tokens"
#
#     Hier gibt es zwei Varianten:
#
#     - Ereignis 3a: "Token an Komponente 'Cognition' weitergeben"
#       Dies geschieht bei der Aktion "Planen".
#
#     - Ereignis 3b: "Token an Komponente 'Sensor' weitergeben"
#       Wird bei allen anderen internen Aktionen gemacht.
#
#------------------------------------------------------------------------

#------------------------------------------------------------------------
#
# Ereignis 3a: "Token an Komponente 'Cognition' weitergeben"
#
#------------------------------------------------------------------------
ON TokenAnCognition
DO
```

```
      #------------------------------------------------------------------------
      # Das Token wird auf die fuer die Komponente Cognition bestimmte
      # Location geschoben.
      #------------------------------------------------------------------------
      TokenAnCOG^: FROM TokenVonBEH GET Token[1];

      #------------------------------------------------------------------------
      # Nun muss NaechsteAktion wiede auf 'TRUE' gesetzt werden, damit, wenn
      # die Komponente Actor das Token spaeter zurueckbekommt, das Ereignis 1
      # wieder schalten kann.
      #------------------------------------------------------------------------
      NaechsteAktion^ := TRUE;
   END

   #---------------------------------------------------------------------------
   #
   # Ereignis 3b: "Token an Komponente 'Sensor' weitergeben"
   #
   #---------------------------------------------------------------------------
   ON TokenAnSensor
   DO

      #------------------------------------------------------------------------
      # Das Token wird auf die fuer die Komponente Sensor bestimmte
      # Location geschoben.
      #------------------------------------------------------------------------
      TokenAnSEN^: FROM TokenVonBEH GET Token[1];

      #------------------------------------------------------------------------
      # Nun muss NaechsteAktion wiede auf 'TRUE' gesetzt werden, damit, wenn
      # die Komponente Actor das Token spaeter zurueckbekommt, das Ereignis 1
      # wieder schalten kann.
      #------------------------------------------------------------------------
      NaechsteAktion^ := TRUE;
   END
END OF Actor
```

D.4 Mobile Komponenten

D.4.1 Die mobile Komponente *DatenTransfer*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell Adam
#
#  Name:             DatenTransfer
#
#  Art:              mobile Komponente
#
#  Version:          1
#
#  Beschreibung:     universelle mobile Komponente fuer den Datenaustausch
#                    zwischen verschiedenen Komponenten.
#
#                    Je nach Verwendung der Komponente, werden einige
#                    Variablen benoetigt und einige nicht.
#
#                    Beispiel: Fuer den Datenaustausch zwischen Behaviour und
#                    Actor werden die Eintraege (x, y, Nahrungsmenge, Aktion)
#                    benoetigt.
#
#-------------------------------------------------------------------------------
```

```
MOBILE COMPONENT DatenTransfer

USE OF UNITS

   #-------------------------------------------------------------------------
   # Energie-Einheit
   #-------------------------------------------------------------------------
   UNIT [EnE] = BASIS

LOCAL DEFINITIONS

   #-------------------------------------------------------------------------
   # Definition der Wertemenge "FELDARTEN": Folgende Feldarten gibt es:
   #     - 'Neutral':   Neutrales Feld
   #     - 'Nahrung':   Nahrungsfeld
   #     - 'Gefahr':    Gefahrenfeld
   #     - 'Unbekannt': Unbekanntes Feld
   #-------------------------------------------------------------------------
   VALUE SET FELDARTEN: ('Neutral', 'Nahrung', 'Gefahr', 'Unbekannt')

  #--------------------------------------------------------------------------
   # Definition der Wertemenge "AKTIONEN": Die moeglichen Aktionen, die ADAM
   # ausfuehren kann.
   #-------------------------------------------------------------------------
   VALUE SET AKTIONEN: ('Planen', 'Explorieren', 'Pruefen',
                        'Gehen', 'Befreien', 'Essen')

DECLARATION OF ELEMENTS

 STATE VARIABLES

  DISCRETE

   #-------------------------------------------------------------------------
   # x-Koordinate des Feldes
   #-------------------------------------------------------------------------
   x          (INTEGER)        := 0 ,

   #-------------------------------------------------------------------------
   # y-Koordinate des Feldes
   #-------------------------------------------------------------------------
   y          (INTEGER)        := 0 ,

   #-------------------------------------------------------------------------
   # TRUE, falls Adam auf einem Gefahrenfeld innerhalb der Fallgrube ist
   #-------------------------------------------------------------------------
   InFalle    (LOGICAL)        := FALSE ,

   #-------------------------------------------------------------------------
   # Feldart
   #-------------------------------------------------------------------------
   Feldart    (FELDARTEN)      := 'Neutral',

   #-------------------------------------------------------------------------
   # Nahrungsmenge
   #-------------------------------------------------------------------------
   Nahrungsmenge (REAL [EnE]) := 0 [EnE] ,

   #-------------------------------------------------------------------------
   # Aktion
   #-------------------------------------------------------------------------
   Aktion     (AKTIONEN)       := 'Explorieren'

END OF DatenTransfer
```

D.4.2 Die mobile Komponente *GedaechtnisEintrag*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell ADAM
#
#  Name:             GedaechtnisEintrag
#
#  Art:              mobile Komponente
#
#  Version:          1
#
#  Beschreibung:     Eintrag auf der Location Gedaechtnis in der
#                    Komponente Cognition
#
#  Bemerkung:
#     Die Repraesentation der Felder in der Form mobiler Komponenten ist
#     notwendig, damit alle im Gedaechtnis befindlichen Felder
#     zum richtigen Zeitpunkt wieder vergessen werden.
#     Die mobilen Komponenten vom Typ GedaechtnisEintrag befinden sich stets
#     nach aufsteigendem TVergessen geordnet auf der Location Gedaechtnis.
#
#-------------------------------------------------------------------------------

MOBILE COMPONENT GedaechtnisEintrag

DECLARATION OF ELEMENTS

 STATE VARIABLES

  DISCRETE

   #-----------------------------------------------------------------------------
   # x - Koordinate des Feldes
   #-----------------------------------------------------------------------------
   x           (INTEGER)   := 0 ,

   #-----------------------------------------------------------------------------
   # y - Koordinate des Feldes
   #-----------------------------------------------------------------------------
    y          (INTEGER)   := 0 ,

   #-----------------------------------------------------------------------------
   # Zeitpunkt, an dem das Feld vergessen werden wird (falls es bis dahin
   # nicht mehr aufgefrischt wird).
   #-----------------------------------------------------------------------------
   TVergessen  (REAL)      := 0

END OF GedaechtnisEintrag
```

D.4.3 Die mobile Komponente *Token*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell ADAM
#
#  Name:             Token
#
#  Art:              mobile Komponente
#
#  Version:          1
#
#  Beschreibung:     mobile Komponente zur Synchronisation kritischer
```

```
#                   Bereiche in den Komponenten.
#
#     Im Modell Adam existiert nur eine mobile Komponente vom Typ Token!
#     Nur diejenige Komponente darf Aktionen eines kritischen
#     Bereichs ausfuehren, die gerade das Token hat. Damit laesst sich
#     eine korrekte Sequentialisierung der Ablaeufe sicherstellen.
#
#-------------------------------------------------------------------------------

MOBILE COMPONENT Token

DECLARATION OF ELEMENTS

END OF Token
```

D.5 Funktionskomponenten

D.5.1 Die Funktionskomponente *F_StrategieErkunden*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell ADAM
#
#  Name:             F_StrategieErkunden
#
#  Art:              Funktionskomponente
#
#  Version:          1
#
#  Beschreibung:     Ermitteln eines Zielfeldes gemaess der Handlungsstrategie
#                    "Erkunden"
#
#     Hier wird zufaellig eines der naechstgelegenen unbekannten Felder
#     als Zielfeld ausgewaehlt.
#     Dazu wird ein Algorithmus verwendet, der alle Felder mit der gleichen
#     Distanz "ringfoermig" durchsucht, beginnend bei Distanz=1. Also wird
#     Adams aktuelles Feld dabei nicht beruecksichtigt.
#     Die Distanz wird immer weiter erhoeht, bis mindestens ein unbekanntes
#     Feld gefunden ist.
#     Falls mehrere gleich weit entfernt unbekannte Felder ermittelt werden,
#     wird eines der Felder zufaellig als Zielfeld ausgewaehlt.
#     Falls kein unbekanntes Feld gefunden werden kann, gibt diese Funktion
#     in beiden Rueckgabeparametern den Wert "-1" zurueck.
#
#     Einschraenkung: Die aktuelle Implementierung dieses Algorithmus ist
#                     folgendermassen limitiert:
#                     Die Anzahl der gleich weit entfernten naechstgelegenen
#                     unbekannten Felder darf 1000 nicht uebersteigen. (Um mit
#                     dieser Grenze in Konflikt zu geraten, muesste das Spielfeld
#                     und die Gedaechtniskapazitaet von Adam sehr gross werden.)
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Cognition an F_StrategieErkunden
#       + Auswahl
#       + Feldart
#       + Maximum
#       + xAdam
#       + yAdam
#       + Protokoll
#
#    - F_StrategieErkunden an Cognition
```

```
#       + xZiel
#       + yZiel
#
#-------------------------------------------------------------------------------

FUNCTION F_StrategieErkunden

LOCAL DEFINITIONS

   #-------------------------------------------------------------------------------
   # Die moeglichen Arten der Felder des Spielfeldes:
   #  - Neutral:     bekanntes neutrales Feld
   #  - Nahrung:     bekanntes Nahrungsfeld
   #  - Gefahr:      bekanntes Gefahrenfeld
   #  - Unbekannt:   unbekanntes Feld
   #-------------------------------------------------------------------------------
   VALUE SET FELDARTEN : ('Neutral', 'Nahrung', 'Gefahr', 'Unbekannt')

DECLARATION OF ELEMENTS

 INPUT PARAMETERS

   #-------------------------------------------------------------------------------
   # gleichverteilte Zufallszahl zwischen 0 und 1 (exklusiv)
   # Diese Zufallszahl wird bei der Auswahl eines Zielfeldes benoetigt, falls
   # mehrere Kandidaten gleich weit entfernt sind.
   #-------------------------------------------------------------------------------
   Auswahl (REAL),

   #-------------------------------------------------------------------------------
   # Feldart von jedem Feld des Spielfeldes
   #-------------------------------------------------------------------------------
   ARRAY [m][n] Feldart (FELDARTEN),

   #-------------------------------------------------------------------------------
   # Seitenlaenge des Spielfeld-Quadrates
   #-------------------------------------------------------------------------------
   Maximum (INT),
   #-------------------------------------------------------------------------------
   # x-Koordinate von Adams Position
   #-------------------------------------------------------------------------------
   xAdam (INT),

   #-------------------------------------------------------------------------------
   # y-Koordinate von Adams Position
   #-------------------------------------------------------------------------------
   yAdam (INT),

   #-------------------------------------------------------------------------------
   # Bildschirmausgabe aktiviert/deaktiviert
   #-------------------------------------------------------------------------------
   Protokoll (LOGICAL)

 OUTPUT PARAMETERS

   #-------------------------------------------------------------------------------
   # x-Koordinate des ermittelten Zielfeldes
   #-------------------------------------------------------------------------------
   xZiel (INT),

   #-------------------------------------------------------------------------------
   # y-Koordinate des ermittelten Zielfeldes
   #-------------------------------------------------------------------------------
   yZiel (INT)

 LOCAL VARIABLES
```

```
   #-------------------------------------------------------------------------
   # Anzahl der ermittelten, gleich weit entfernten, Kandidaten fuer das
   # Zielfeld
   #-------------------------------------------------------------------------
   Anzahl (INTEGER)             := 0,

   #-------------------------------------------------------------------------
   # x-Koordinaten der Zielfeld-Kandidaten, der potentiellen Zielfelder
   #-------------------------------------------------------------------------
   ARRAY [1000] x_PotZielfeld (INTEGER) := 0,

   #-------------------------------------------------------------------------
   # y-Koordinaten der Zielfeld-Kandidaten, der potentiellen Zielfelder
   #-------------------------------------------------------------------------
   ARRAY [1000] y_PotZielfeld (INTEGER) := 0,

   #-------------------------------------------------------------------------
   # wird gesetzt, wenn mindestens ein potentielles Zielfeld gefunden worden
   # ist.
   #-------------------------------------------------------------------------
   FeldGefunden   (LOGICAL) := FALSE

BEGIN

   #-------------------------------------------------------------------------
   # mit der folgenden geschachtelten Schleifenstruktur wird erreicht,
   # dass die Felder mit steigendem Abstand zum aktuellen Feld von Adam
   # "ringfoermig" durchsucht werden.
   #-------------------------------------------------------------------------

      #-------------------------------------------------------------------------
      # Schleife 1 (aeussere Schleife): "Erhoehen der Distanz vom Startfeld"
      #
      # Die minimale Distanz ist 1, die maximale ist Maximum-1
      # (z.B. von links unten nach rechts oben)
      #-------------------------------------------------------------------------
      FOR d FROM 1 TO (Maximum - 1)
      REPEAT
         #-------------------------------------------------------------------------
         # Schleife 2a (erste innere Schleife): "Untersuchen der oberen und
         #                                      unteren Seite des Rings"
         #-------------------------------------------------------------------------
         FOR i FROM (xAdam - d) TO (xAdam + d)
         REPEAT

            #-------------------------------------------------------------------
            # Untersuchen eines der oberen Felder des Ringes (i, yAdam+d).
            # Dabei wird sicher gestellt, dass nicht auf ein Feld, das gar nicht
            # existiert, zugegriffen wird.
            #-------------------------------------------------------------------
            IF ((i >= 1) AND (i <= Maximum) AND (yAdam + d >= 1)
                         AND (yAdam + d <= Maximum))
            DO

               #----------------------------------------------------------------
               # Die unbekannten Felder sind die potentiellen Zielfelder
               #----------------------------------------------------------------
               IF (Feldart[i][yAdam + d] = 'Unbekannt')
               DO

                  #-------------------------------------------------------------
                  # Vermeiden einer Ueberschreitung der Array-Groesse:
                  #-------------------------------------------------------------
                  IF (Anzahl = 1000)
                  DO
                     DISPLAY ("\n T= %f: FEHLER: F_StrategieErkunden:", T);
                     DISPLAY (" Maximale Menge an Zielfeldern erreicht:");
```

```
            DISPLAY ("\n Keine weiteren Felder werden in die ");
            DISPLAY ("Auswahl aufgenommen!!\n");

            #-------------------------------------------------------
            # Verlassen der aeusseren Schleife
            #-------------------------------------------------------
            EXIT;
         END

         #-------------------------------------------------------
         # Ein (weiteres) potentielles Zielfeld wurde gefunden
         #-------------------------------------------------------
         Anzahl := Anzahl + 1;
         FeldGefunden := TRUE;

         #-------------------------------------------------------
         # Merken der Koordinaten des Feldes
         #-------------------------------------------------------
         x_PotZielfeld[Anzahl] := i;
         y_PotZielfeld[Anzahl] := yAdam + d;

         IF Protokoll
         DO
            DISPLAY ("\n T= %f: COG: ", T);
            DISPLAY ("%u. potentielles Zielfeld", Anzahl);
            DISPLAY (" ermittelt: (%u,%u)",
                     x_PotZielfeld[Anzahl], y_PotZielfeld[Anzahl]);
         END
      END
   END

   #-----------------------------------------------------------------
   # Untersuchen eines der unteren Felder des Ringes (i, yAdam-d).
   # Dabei wird sicher gestellt, dass nicht auf ein Feld, das gar
   # nicht existiert, zugegriffen wird.
   #-----------------------------------------------------------------
   IF ((i >= 1) AND (i <= Maximum) AND (yAdam - d >=1)
               AND (yAdam - d <= Maximum))
   DO

      #-------------------------------------------------------------
      # Die unbekannten Felder sind die potentiellen Zielfelder
      #-------------------------------------------------------------
      IF (Feldart[i][yAdam - d] = 'Unbekannt')
      DO

         #----------------------------------------------------------
         # Vermeiden einer Ueberschreitung der Array-Groesse:
         #----------------------------------------------------------
         IF (Anzahl = 1000)
         DO
            DISPLAY ("\n T= %f: FEHLER: F_StrategieErkunden:", T);
            DISPLAY (" Maximale Menge an Zielfeldern erreicht:");
            DISPLAY ("\n Keine weiteren Felder werden in die ");
            DISPLAY ("Auswahl aufgenommen!!\n");

            #----------------------------------------------------------
            # Verlassen der aeusseren Schleife
            #----------------------------------------------------------
            EXIT;
         END

         #----------------------------------------------------------
         # Ein (weiteres) potentielles Zielfeld wurde gefunden
         #----------------------------------------------------------
         Anzahl := Anzahl + 1;
         FeldGefunden := TRUE;
```

```
         #-------------------------------------------------------------
         # Merken der Koordinaten des Feldes
         #-------------------------------------------------------------
         x_PotZielfeld[Anzahl] := i;
         y_PotZielfeld[Anzahl] := yAdam - d;

         IF Protokoll
         DO
            DISPLAY ("\n T= %f: COG: ",T);
            DISPLAY ("%u. potentielles Zielfeld", Anzahl);
            DISPLAY (" ermittelt: (%u,%u)",
                     x_PotZielfeld[Anzahl], y_PotZielfeld[Anzahl]);
         END
      END
   END
 END_LOOP

#---------------------------------------------------------------------
 #  Schleife 2b (zweite innere Schleife): "Untersuchen der rechten und
 #                                        linken Seite des Rings"
 # Dies geschieht jeweils ohne das oberste und unterste Feld, die
 # schon in der vorigen Schleife behandelt wurden.
 #---------------------------------------------------------------------
 FOR i FROM (yAdam - d + 1) TO (yAdam + d - 1)
 REPEAT

    #------------------------------------------------------------------
    # Untersuchen eines der rechten Felder des Ringes (xAdam+d, i).
    # Dabei wird sichergestellt, dass nicht auf ein Feld, das nicht
    # existiert, zugegriffen wird.
    #------------------------------------------------------------------
    IF ((i >= 1) AND (i <= Maximum) AND (xAdam + d >= 1)
                 AND (xAdam + d <= Maximum))
    DO

       #---------------------------------------------------------------
       # Die unbekannten Felder sind die potentiellen Zielfelder
       #---------------------------------------------------------------
       IF (Feldart[xAdam + d][i] = 'Unbekannt')
       DO

          #------------------------------------------------------------
          # Vermeiden einer Ueberschreitung der Array-Groesse:
          #------------------------------------------------------------
          IF (Anzahl = 1000)
          DO
             DISPLAY ("\n T= %f: FEHLER: F_StrategieErkunden:", T);
             DISPLAY (" Maximale Menge an Zielfeldern erreicht:");
             DISPLAY ("\n Keine weiteren Felder werden in die ");
             DISPLAY ("Auswahl aufgenommen!!\n");

             #---------------------------------------------------------
             # Verlassen der aeusseren Schleife
             #---------------------------------------------------------
             EXIT;
          END

          #------------------------------------------------------------
          # Ein (weiteres) potentielles Zielfeld wurde gefunden
          #------------------------------------------------------------
          Anzahl := Anzahl + 1;
          FeldGefunden := TRUE;

          #------------------------------------------------------------
          # Merken der Koordinaten des Feldes
          #------------------------------------------------------------
```

```
            x_PotZielfeld[Anzahl] := xAdam + d;
            y_PotZielfeld[Anzahl] := i;

            IF Protokoll
            DO
               DISPLAY ("\n T= %f: COG: ",T);
               DISPLAY ("%u. potentielles Zielfeld", Anzahl);
               DISPLAY (" ermittelt: (%u,%u)",
                        x_PotZielfeld[Anzahl], y_PotZielfeld[Anzahl]);
            END
         END
      END

      #-------------------------------------------------------------------
      # Untersuchen eines der linken Felder des Ringes (xAdam-d, i).
      # Dabei wird sichergestellt, dass nicht auf ein Feld, das nicht
      # existiert, zugegriffen wird.
      #-------------------------------------------------------------------
      IF ((i >= 1) AND (i <= Maximum) AND (xAdam - d >=1)
                   AND (xAdam - d <= Maximum))
      DO

         #----------------------------------------------------------------
         # Die unbekannten Felder sind die potentiellen Zielfelder
         #----------------------------------------------------------------
         IF (Feldart[xAdam - d][i] = 'Unbekannt')
         DO

            #-------------------------------------------------------------
            # Vermeiden einer Ueberschreitung der Array-Groesse:
            #-------------------------------------------------------------
            IF (Anzahl = 1000)
            DO
               DISPLAY ("\n T= %f: FEHLER: F_StrategieErkunden:", T);
               DISPLAY (" Maximale Menge an Zielfeldern erreicht:");
               DISPLAY ("\n Keine weiteren Felder werden in die ");
               DISPLAY ("Auswahl aufgenommen!!\n");

               #----------------------------------------------------------
               # Verlassen der aeusseren Schleife
               #----------------------------------------------------------
               EXIT;
            END

            #-------------------------------------------------------------
            # Ein (weiteres) potentielles Zielfeld wurde gefunden
            #-------------------------------------------------------------
            Anzahl := Anzahl + 1;
            FeldGefunden := TRUE;

            #-------------------------------------------------------------
            # Merken der Koordinaten des Feldes
            #-------------------------------------------------------------
            x_PotZielfeld[Anzahl] := xAdam - d;
            y_PotZielfeld[Anzahl] := i;

            IF Protokoll
            DO
               DISPLAY ("\n T= %f: COG: ",T);
               DISPLAY ("%u. potentielles Zielfeld", Anzahl);
               DISPLAY (" ermittelt: (%u,%u)",
                        x_PotZielfeld[Anzahl], y_PotZielfeld[Anzahl]);
            END
         END
      END
   END_LOOP
```

```
         #------------------------------------------------------------------------
         # Abbruch der Suche bei Erfolg:
         # Wenn mindestens ein Feld gefunden wurde, so wird, nachdem
         # auch alle anderen Felder mit der gleichen Distanz wie dieses
         # Feld ermittelt sind, der Suchvorgang beendet.
         #------------------------------------------------------------------------
         IF (FeldGefunden = TRUE)
         DO
            EXIT;
         END

      END_LOOP

      #---------------------------------------------------------------------------
      # Bedingte Anweisung 1: "Potentielles Zielfeld gefunden?"
      #---------------------------------------------------------------------------

      #---------------------------------------------------------------------------
      # Bedingte Anweisung 1a: "Die Menge der potentiellen Zielfelder ist nicht
      #                         leer"
      #---------------------------------------------------------------------------
      IF (Anzahl > 0)
      DO
         #------------------------------------------------------------------------
         # Aus der Menge der potentiellen Zielfelder wird zufaellig
         # eines als Zielfeld ausgewaehlt.
         #------------------------------------------------------------------------
         xZiel := x_PotZielfeld[ITRUNC((Anzahl*Auswahl)+1)];
         yZiel := y_PotZielfeld[ITRUNC((Anzahl*Auswahl)+1)];

      END

      #---------------------------------------------------------------------------
      # Bedingte Anweisung 1b: "Die Menge der potentiellen Zielfelder ist
      #                         leer"
      # D. h. es existiert kein einziges unbekanntes Feld in Adams Welt.
      # Adam kennt bereits seine gesamte Umwelt.
      # Dies ist bisher nicht vorgesehen, da fuer diesen Fall z.B. eine
      # zusaetzliche Aktion "Ruhen" eingefuehrt werden muesste, damit definiert
      # ist, was Adam nun macht.
      #---------------------------------------------------------------------------
      ELSE
      DO
         #------------------------------------------------------------------------
         # mit dem Rueckgabewert "-1" wird der Cognition signalisiert, dass
         # kein Zielfeld gefunden werden konnte.
         #------------------------------------------------------------------------
         xZiel := -1;
         yZiel := -1;
      END

   #------------------------------------------------------------------------------
   # Die Funktionskomponente hat ihre Aufgabe erledigt (D. h. in die Variablen
   # zur Rueckgabe "xZiel" und "yZiel" stehen die Koordinaten des ermittelten
   # Zielfeldes bzw., falls kein Zielfeld gefunden wurde, andere Werte).
   # Also gehts zurueck zur Komponente Cognition.
   #------------------------------------------------------------------------------
   RETURN

END OF F_StrategieErkunden
```

D.5.2 Die Funktionskomponente *F_StrategieNahrungssuche*

```
#-------------------------------------------------------------------------------
#
#  Projekt:          Modell ADAM
#
#  Name:             F_StrategieNahrungssuche
#
#  Art:              Funktionskomponente
#
#  Version:          1
#
#  Beschreibung:     Ermitteln eines Zielfeldes gemaess der
#                    Handlungsstrategie "Nahrungssuche"
#
#     2 Varianten:
#        - optimistische Variante
#        - pessimistische Variante
#
#     Die beiden Varianten unterscheiden sich nur durch die verwendete
#     Lernschranke:
#     In der optimistischen Variante werden alle
#     Nahrungsfelder, auf denen seit mindestens DeltaSicherNein
#     Zeiteinheiten nicht mehr gegessen wurde beruecksichtigt.
#     (Ganz genau genommen duerfte seit mehr als DeltaSicherNein Zeiteinheiten
#     auf dem Feld nicht mehr gegessen worden sein. Dies wird an dieser Stelle
#     jedoch vernachlaessigt.)
#     Dagegen werden in der pessimistischen Variante nur die Nahrungsfelder
#     betrachtet, auf denen seit mindestens DeltaSicherJa Zeiteinheiten
#     nicht mehr gegessen wurde.
#     Diese Lernschranke wird in "Delta" aus der Komponente Cognition
#     uebergeben.
#
#     Es wird das naechstgelegene Nahrungsfeld, auf dem bekanntermassen
#     Nahrung ist, bzw. auf dem seit dem Zeitraum "Delta" nicht mehr
#     gegessen wurde als Zielfeld ausgewaehlt.
#     Das Feld, auf dem sich Adam gerade befindet, wird dabei nicht
#     beruecksichtigt.
#     Falls mehrere gleich weit entfernte Nahrungsfelder in Frage kommen,
#     wird eines von diesen zufaellig bestimmt.
#     Falls kein Nahrungsfeld bekannt ist, das den Anforderungen entspricht,
#     so wird als Rueckgabewert "0" in die beiden Ausgabeparameter
#     geschrieben. Die Komponente Cognition wird dann zu einer anderen
#     Strategie uebergehen.
#
#     Einschraenkung: Die aktuelle Implementierung dieses Algorithmus ist
#                     folgendermassen limitiert:
#                     Die Anzahl der gleich weit entfernten naechstgelegenen
#                     unbekannten Felder darf 1000 nicht uebersteigen. (Um mit
#                     dieser Grenze in Konflikt zu geraten, muesste das Spielfeld
#                     und die Gedaechtniskapazitaet von Adam sehr gross werden.)
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Cognition an F_StrategieNahrungssuche
#       + Auswahl
#       + Maximum
#       + xAdam
#       + yAdam
#       + Nahrungsmenge
#       + TLeerGegessen
#       + Delta
#       + Protokoll
#
#    - F_StrategieNahrungssuche an Cognition
#       + xZiel
```

```
#       + yZiel
#
#-------------------------------------------------------------------------------

FUNCTION F_StrategieNahrungssuche

DECLARATION OF ELEMENTS

 INPUT PARAMETERS

   #-----------------------------------------------------------------------------
   # gleichverteilte Zufallszahl zwischen 0 und 1 (exklusiv)
   # Diese Zufallszahl wird bei der Auswahl eines Zielfeldes benoetigt, falls
   # mehrere Kandidaten gleich weit entfernt sind.
   #-----------------------------------------------------------------------------
   Auswahl (REAL),

  #------------------------------------------------------------------------------
   # Spielfeldgroesse in x- und y-Richtung
   #-----------------------------------------------------------------------------
   Maximum (INT),

   #-----------------------------------------------------------------------------
   # x-Koordinate von Adams Position
   #-----------------------------------------------------------------------------
   xAdam (INT),

   #-----------------------------------------------------------------------------
   # y-Koordinate von Adams Position
   #-----------------------------------------------------------------------------
   yAdam (INT),

   #-----------------------------------------------------------------------------
   # bekannte Nahrungsmenge auf allen Feldern
   #-----------------------------------------------------------------------------
   ARRAY [m][n] Nahrungsmenge (REAL),

   #-----------------------------------------------------------------------------
   # Zeitpunkte, an denen auf den jeweiligen Feldern zuletzt gegessen wurde
   #-----------------------------------------------------------------------------
   ARRAY [m][n] TLeerGegessen (REAL),

   #-----------------------------------------------------------------------------
   # uebergebene Lernschranke (entweder DeltaSicherJa oder DeltaSicherNein)
   #-----------------------------------------------------------------------------
   Delta (REAL),

   #-----------------------------------------------------------------------------
   # Bildschirmausgabe aktiviert/deaktiviert
   #-----------------------------------------------------------------------------
   Protokoll (LOGICAL)

 OUTPUT PARAMETERS

   #-----------------------------------------------------------------------------
   # x-Koordinate des bestimmten Zielfeldes
   #-----------------------------------------------------------------------------
   xZiel (INT),

   #-----------------------------------------------------------------------------
   # y-Koordinate des bestimmten Zielfeldes
   #-----------------------------------------------------------------------------
   yZiel (INT)

 LOCAL VARIABLES
```

```
   #-------------------------------------------------------------------------
   # Anzahl der ermittelten, gleich weit entfernten, Kandidaten fuer das
   # Zielfeld
   #-------------------------------------------------------------------------
   Anzahl (INTEGER)             := 0,

   #-------------------------------------------------------------------------
   # x-Koordinaten der Zielfeld-Kandidaten, der potentiellen Zielfelder
   #-------------------------------------------------------------------------
   ARRAY [1000] x_PotZielfeld (INTEGER) := 0,

   #-------------------------------------------------------------------------
   # y-Koordinaten der Zielfeld-Kandidaten, der potentiellen Zielfelder
   #-------------------------------------------------------------------------
   ARRAY [1000] y_PotZielfeld (INTEGER) := 0,

   #-------------------------------------------------------------------------
   # wird gesetzt, wenn mindestens ein potentielles Zielfeld gefunden worden
   # ist.
   #-------------------------------------------------------------------------
   FeldGefunden   (LOGICAL) := FALSE

BEGIN

   #-------------------------------------------------------------------------
   # mit der folgenden geschachtelten Schleifenstruktur wird erreicht,
   # dass die Felder mit steigendem Abstand zum aktuellen Feld von Adam
   # "ringfoermig" durchsucht werden.
   #-------------------------------------------------------------------------

      #----------------------------------------------------------------------
      # Schleife 1 (aeussere Schleife): "Erhoehen der Distanz vom Startfeld"
      #
      # Die minimale Distanz ist 1 (also wird Adams aktuelles Feld nicht in
      # Betracht gezogen! Sonst staende hier 0.), die maximale ist Maximum-1
      # (z.B. von links unten nach rechts oben)
      #----------------------------------------------------------------------
      FOR d FROM 1 TO (Maximum - 1)
      REPEAT

         #-------------------------------------------------------------------
         # Schleife 2a (erste innere Schleife): "Untersuchen der oberen und
         #                                      unteren Seite des Rings"
         #-------------------------------------------------------------------
         FOR i FROM (xAdam - d) TO (xAdam + d)
         REPEAT

            #----------------------------------------------------------------
            # Untersuchen eines der oberen Felder des Ringes (i, yAdam+d).
            # Dabei wird sicher gestellt, dass nicht auf ein Feld, das gar nicht
            # existiert, zugegriffen wird.
            #----------------------------------------------------------------
            IF ((i >= 1) AND (i <= Maximum) AND (yAdam + d >= 1)
                         AND (yAdam + d <= Maximum))
            DO

               #-------------------------------------------------------------
               # (Nahrungs-)Felder, auf denen entweder sicher Nahrung ist oder
               # auf denen seit Delta Zeiteinheiten nicht mehr gegessen wurde,
               # sind die potentiellen Zielfelder.
               # (TLeerGegessen > 0) stellt dabei sicher, daß darauf ueberhaupt
               # schon mal gegessen wurde.)
               #-------------------------------------------------------------
               IF ((Nahrungsmenge[i][yAdam + d] > 0 ) OR
                   ((TLeerGegessen[i][yAdam + d] > 0) AND
                    (T >= (TLeerGegessen[i][yAdam + d] + Delta))))
               DO
```

```
      #-------------------------------------------------------------
      # Vermeiden einer Ueberschreitung der Array-Groesse:
      #-------------------------------------------------------------
      IF (Anzahl = 1000)
      DO
         DISPLAY ("\n T= %f: FEHLER: F_StrategieNahrungssuche:"
                  , T);
         DISPLAY (" Maximale Menge an Zielfeldern erreicht:");
         DISPLAY ("\n Keine weiteren Felder werden in die ");
         DISPLAY ("Auswahl aufgenommen!!\n");

         #-------------------------------------------------------------
         # Verlassen der aeusseren Schleife (Schleife 1)
         #-------------------------------------------------------------
         EXIT;
      END

      #-------------------------------------------------------------
      # Ein (weiteres) potentielles Zielfeld wurde gefunden
      #-------------------------------------------------------------
      Anzahl := Anzahl + 1;
      FeldGefunden := TRUE;

      #-------------------------------------------------------------
      # Merken der Koordinaten des Feldes
      #-------------------------------------------------------------
      x_PotZielfeld[Anzahl] := i;
      y_PotZielfeld[Anzahl] := yAdam + d;

      IF Protokoll
      DO
         DISPLAY ("\n T= %f: COG: ", T);
         DISPLAY ("%u. potentielles", Anzahl);
         DISPLAY (" Zielfeld ermittelt: (%u,%u)",
                  x_PotZielfeld[Anzahl], y_PotZielfeld[Anzahl]);
      END
   END
END

#---------------------------------------------------------------------
# Untersuchen eines der unteren Felder des Ringes (i, yAdam-d).
# Dabei wird sicher gestellt, dass nicht auf ein Feld, das gar
# nicht existiert, zugegriffen wird.
#---------------------------------------------------------------------
IF ((i >= 1) AND (i <= Maximum) AND (yAdam - d >=1)
            AND (yAdam - d <=Maximum))
DO

   #-------------------------------------------------------------
   # (Nahrungs-)Felder, auf denen entweder sicher Nahrung ist oder
   # auf denen seit Delta Zeiteinheiten nicht mehr gegessen wurde,
   # sind die potentiellen Zielfelder.
   # (TLeerGegessen > 0) stellt dabei sicher, daß darauf ueberhaupt
   # schon mal gegessen wurde.)
   #-------------------------------------------------------------
   IF ((Nahrungsmenge[i][yAdam - d] > 0 ) OR
       ((TLeerGegessen[i][yAdam - d] > 0) AND
        (T >= (TLeerGegessen[i][yAdam - d] + Delta))))
   DO

      #-------------------------------------------------------------
      # Vermeiden einer Ueberschreitung der Array-Groesse:
      #-------------------------------------------------------------
      IF (Anzahl = 1000)
      DO
         DISPLAY ("\n T= %f: FEHLER: F_StrategieNahrungssuche:"
                  , T);
```

```
            DISPLAY (" Maximale Menge an Zielfeldern erreicht:");
            DISPLAY ("\n Keine weiteren Felder werden in die ");
            DISPLAY ("Auswahl aufgenommen!!\n");

            #-------------------------------------------------------------
            # Verlassen der aeusseren Schleife
            #-------------------------------------------------------------
            EXIT;
         END

         #-------------------------------------------------------------
         # Ein (weiteres) potentielles Zielfeld wurde gefunden
         #-------------------------------------------------------------
         Anzahl := Anzahl + 1;
         FeldGefunden := TRUE;

         #-------------------------------------------------------------
         # Merken der Koordinaten des Feldes
         #-------------------------------------------------------------
         x_PotZielfeld[Anzahl] := i;
         y_PotZielfeld[Anzahl] := yAdam - d;

         IF Protokoll
         DO
            DISPLAY ("\n T= %f: COG: ", T);
            DISPLAY ("%u. potentielles", Anzahl);
            DISPLAY (" Zielfeld ermittelt: (%u,%u)",
                     x_PotZielfeld[Anzahl], y_PotZielfeld[Anzahl]);
         END
      END
   END
END_LOOP

#-----------------------------------------------------------------------
#  Schleife 2b (zweite innere Schleife): "Untersuchen der rechten und
#                                        linken Seite des Rings"
# Dies geschieht jeweils ohne das oberste und unterste Feld, die
# schon in der vorigen Schleife behandelt wurden.
#-----------------------------------------------------------------------
FOR i FROM (yAdam - d + 1) TO (yAdam + d - 1)
REPEAT

   #--------------------------------------------------------------------
   # Untersuchen eines der rechten Felder des Ringes (xAdam+d, i).
   # Dabei wird sichergestellt, dass nicht auf ein Feld, das nicht
   # existiert, zugegriffen wird.
   #--------------------------------------------------------------------
   IF ((i >= 1) AND (i <= Maximum) AND (xAdam + d >= 1)
               AND (xAdam + d <= Maximum))
   DO

      #-----------------------------------------------------------------
      # (Nahrungs-)Felder, auf denen entweder sicher Nahrung ist oder
      # auf denen seit Delta Zeiteinheiten nicht mehr gegessen wurde,
      # sind die potentiellen Zielfelder.
      # (TLeerGegessen > 0) stellt dabei sicher, daß darauf ueberhaupt
      # schon mal gegessen wurde.)
      #-----------------------------------------------------------------
      IF ((Nahrungsmenge[xAdam + d][i] > 0 ) OR
          ((TLeerGegessen[xAdam + d][i] > 0) AND
           (T >= (TLeerGegessen[xAdam + d][i] + Delta))))
      DO

         #-------------------------------------------------------------
         # Vermeiden einer Ueberschreitung der Array-Groesse:
         #-------------------------------------------------------------
         IF (Anzahl = 1000)
```

```
      DO
         DISPLAY ("\n T= %f: FEHLER: F_StrategieNahrungssuche:"
                  , T);
         DISPLAY (" Maximale Menge an Zielfeldern erreicht:");
         DISPLAY ("\n Keine weiteren Felder werden in die ");
         DISPLAY ("Auswahl aufgenommen!!\n");

         #-------------------------------------------------------------
         # Verlassen der aeusseren Schleife
         #-------------------------------------------------------------
         EXIT;
      END

      #-------------------------------------------------------------
      # Ein (weiteres) potentielles Zielfeld wurde gefunden
      #-------------------------------------------------------------
      Anzahl := Anzahl + 1;
      FeldGefunden := TRUE;

      #-------------------------------------------------------------
      # Merken der Koordinaten des Feldes
      #-------------------------------------------------------------
      x_PotZielfeld[Anzahl] := xAdam + d;
      y_PotZielfeld[Anzahl] := i;

      IF Protokoll
      DO
         DISPLAY ("\n T= %f: COG: ", T);
         DISPLAY ("%u. potentielles", Anzahl);
         DISPLAY (" Zielfeld ermittelt: (%u,%u)",
                  x_PotZielfeld[Anzahl], y_PotZielfeld[Anzahl]);
      END
   END
END

#-----------------------------------------------------------------------
# Untersuchen eines der linken Felder des Ringes (xAdam-d, i).
# Dabei wird sichergestellt, dass nicht auf ein Feld, das nicht
# existiert, zugegriffen wird.
#-----------------------------------------------------------------------
IF ((i >= 1) AND (i <= Maximum) AND (xAdam - d >=1)
             AND (xAdam - d <=Maximum))
DO

   #-------------------------------------------------------------------
   # (Nahrungs-)Felder, auf denen entweder sicher Nahrung ist oder
   # auf denen seit Delta Zeiteinheiten nicht mehr gegessen wurde,
   # sind die potentiellen Zielfelder.
   # (TLeerGegessen > 0) stellt dabei sicher, daß darauf ueberhaupt
   # schon mal gegessen wurde.)
   #-------------------------------------------------------------------
   IF ((Nahrungsmenge[xAdam - d][i] > 0 ) OR
       ((TLeerGegessen[xAdam - d][i] > 0) AND
        (T >= (TLeerGegessen[xAdam - d][i] + Delta))))
   DO

      #-------------------------------------------------------------
      # Vermeiden einer Ueberschreitung der Array-Groesse:
      #-------------------------------------------------------------
      IF (Anzahl = 1000)
      DO
         DISPLAY ("\n T= %f: FEHLER: F_StrategieNahrungssuche:"
                  , T);
         DISPLAY (" Maximale Menge an Zielfeldern erreicht:");
         DISPLAY ("\n Keine weiteren Felder werden in die ");
         DISPLAY ("Auswahl aufgenommen!!\n");
```

```
                  #-------------------------------------------------------
                  # Verlassen der aeusseren Schleife
                  #-------------------------------------------------------
                  EXIT;
               END

               #-------------------------------------------------------
               # Ein (weiteres) potentielles Zielfeld wurde gefunden
               #-------------------------------------------------------
               Anzahl := Anzahl + 1;
               FeldGefunden := TRUE;

               #-------------------------------------------------------
               # Merken der Koordinaten des Feldes
               #-------------------------------------------------------
               x_PotZielfeld[Anzahl] := xAdam - d;
               y_PotZielfeld[Anzahl] := i;

               IF Protokoll
               DO
                  DISPLAY ("\n T= %f: COG: ", T);
                  DISPLAY ("%u. potentielles", Anzahl);
                  DISPLAY (" Zielfeld ermittelt: (%u,%u)",
                           x_PotZielfeld[Anzahl], y_PotZielfeld[Anzahl]);
               END
            END
         END
      END_LOOP

      #-------------------------------------------------------------------
      # Abbruch der Suche bei Erfolg:
      # Wenn mindestens ein Feld gefunden wurde, so wird, nachdem
      # auch alle anderen Felder mit der gleichen Distanz wie dieses
      # Feld ermittelt sind, der Suchvorgang beendet.
      #-------------------------------------------------------------------
      IF (FeldGefunden = TRUE)
      DO
         EXIT;
      END

   END_LOOP

   #----------------------------------------------------------------------
   # Bedingte Anweisung 1: "Potentielles Zielfeld gefunden?"
   #----------------------------------------------------------------------

   #----------------------------------------------------------------------
   # Bedingte Anweisung 1a: "Die Menge der potentiellen Zielfelder ist nicht
   #                         leer"
   #----------------------------------------------------------------------
   IF (Anzahl >0)
   DO
      #-------------------------------------------------------------------
      # Aus der Menge der potentiellen Zielfelder wird zufaellig
      # eines als Zielfeld ausgewaehlt.
      #-------------------------------------------------------------------
      xZiel := x_PotZielfeld[ITRUNC((Anzahl*Auswahl)+1)];
      yZiel := y_PotZielfeld[ITRUNC((Anzahl*Auswahl)+1)];

   END

   #----------------------------------------------------------------------
   # Bedingte Anweisung 1b: "Die Menge der potentiellen Zielfelder ist
   #                         leer"
   # D. h. es existiert kein einziges Nahrungsfeld in Adams interner
   # Repraesentation, das die geforderte Bedingung erfuellt (Auf dem Feld
   # wurde zuletzt Nahrung exploriert bzw. es wurde seit Delta Zeiteinheiten
```

```
      # nicht mehr darauf gegessen.
      #--------------------------------------------------------------------------
      ELSE
      DO
         #--------------------------------------------------------------------------
         # mit Rueckgabewert "0" wird der Cognition signalisiert, dass kein
         # Zielfeld gefunden werden konnte.
         #--------------------------------------------------------------------------
         xZiel := 0;
         yZiel := 0;
      END

   #--------------------------------------------------------------------------
   # Funktionskomponente hat ihre Aufgabe erledigt (D. h. in den Variablen fuer
   # die Rueckgabe "xZiel" und "yZiel" stehen die Koordinaten des ermittelten
   # Zielfeldes bzw., falls kein Zielfeld gefunden wurde, andere
   # Werte).
   # Also gehts zurueck zur Komponente Cognition.
   #--------------------------------------------------------------------------
   RETURN

END OF F_StrategieNahrungssuche
```

D.5.3 Die Funktionskomponente *F_Wegeplanung*

```
#------------------------------------------------------------------------------
#
#  Projekt:          Modell ADAM
#
#  Name:             F_Wegeplanung
#
#  Art:              Funktionskomponente
#
#  Version:          1
#
#  Beschreibung:     Ermitteln des naechsten Teilzieles im Handlungsplan
#
#        Der hier vorgestellte Algorithmus ist sehr einfach. Er kann
#        leicht durch einen ausgefeilteren ersetzt werden. Die Aussagekraft
#        des Modells erhoeht sich jedoch dadurch wohl nicht.
#
#        Der Algorithmus funktioniert grob in zwei Schritten:
#           1. Zunaechst werden die vorlaeufigen Koordinaten des naechsten
#              Teilzieles ermittelt. Dabei wird einfach ein bestimmtes der
#              moeglichen naechsten Felder ausgewaehlt, das auf einer von der
#              Distanz her optimalen Route liegt.
#              Es wird hier also die Feldart des Feldes noch nicht beachtet,
#              sondern nur die Position.
#
#           2. Falls dieses ausgewaehlte Feld die Feldart Gefahrenfeld hat,
#              wird es wieder verworfen und moeglichst geschickt umgangen.
#              Ein unbekanntes Feld wird also nicht vermieden, wenn es ein
#              gleich guenstiges bekanntes Feld gaebe.
#
#     Einschraenkungen:
#        Der hier verwendete Wegeplanungsalgorithmus setzt voraus, dass
#        Gefahrenstellen nicht direkt waagrecht oder senkrecht nebeneinander
#        liegen.
#
#  Verbindung(en) mit anderen Komponenten:
#
#    - Cognition an F_Wegeplanung
#       + Feldart
#       + Maximum
```

```
#       + xAktuell
#       + yAktuell
#       + xZiel
#       + yZiel
#
#    - F_Wegeplanung an Cognition
#       + xNext
#       + yNext
#
#-------------------------------------------------------------------------------

FUNCTION F_Wegeplanung

LOCAL DEFINITIONS

   #-------------------------------------------------------------------------------
   # Die moeglichen Arten der Felder des Spielfeldes:
   #  - Neutral:     bekanntes neutrales Feld
   #  - Nahrung:     bekanntes Nahrungsfeld
   #  - Gefahr:      bekanntes Gefahrenfeld
   #  - Unbekannt:   unbekanntes Feld
   #-------------------------------------------------------------------------------
   VALUE SET FELDARTEN : ('Neutral', 'Nahrung', 'Gefahr', 'Unbekannt')

DECLARATION OF ELEMENTS

 INPUT PARAMETERS

   #-------------------------------------------------------------------------------
   # Feldart fuer jedes Feld des Spielfeldes
   #-------------------------------------------------------------------------------
   ARRAY [m][n] Feldart (FELDARTEN),

   #-------------------------------------------------------------------------------
   # Spielfeldgroesse des quadratischen Spielfeldes in x- und y-Richtung
   #-------------------------------------------------------------------------------
   Maximum (INT),

   #-------------------------------------------------------------------------------
   # x-Koordinate des Startpunktes fuer den jetzigen Schritt der Wegeplanung
   #-------------------------------------------------------------------------------
   xAktuell (INT),
   #-------------------------------------------------------------------------------
   # y-Koordinate des Startpunktes fuer den jetzigen Schritt der Wegeplanung
   #-------------------------------------------------------------------------------
   yAktuell (INT),

   #-------------------------------------------------------------------------------
   # x-Koordinate des Zielpunktes der Wegeplanung
   #-------------------------------------------------------------------------------
   xZiel (INT),

   #-------------------------------------------------------------------------------
   # y-Koordinate des Zielpunktes der Wegeplanung
   #-------------------------------------------------------------------------------
   yZiel (INT)

 OUTPUT PARAMETERS

   #-------------------------------------------------------------------------------
   # x-Koordinate des ermittelten naechsten Feldes im "Wegeplan"
   #-------------------------------------------------------------------------------
   xNext (INT),

   #-------------------------------------------------------------------------------
   # y-Koordinate des ermittelten naechsten Feldes im "Wegeplan"
   #-------------------------------------------------------------------------------
```

```
   yNext (INT)

 LOCAL VARIABLES

   #-------------------------------------------------------------------------
   # Der unten folgende Wegeplanungsalgorithmus funktioniert auch fuer
   # rechteckige Welten (nicht nur fuer quadratische), deren Koordinaten
   # nicht bei 1 beginnen. Um diese Allgemeinheit zu erhalten werden hier
   # einige lokale Variablen eingefuehrt.
   #-------------------------------------------------------------------------

   #-------------------------------------------------------------------------
   # Die x-Koordinaten des Spielfeldes beginnen bei 1
   #-------------------------------------------------------------------------
   XMIN (INT) := 1,

   #-------------------------------------------------------------------------
   # groesster Wert der x-Koordinate
   # XMAX wird unten auf Maximum gesetzt
   #-------------------------------------------------------------------------
   XMAX (INT) := 12,

   #-------------------------------------------------------------------------
   # Die y-Koordinaten des Spielfeldes beginnen bei 1
   #-------------------------------------------------------------------------
   YMIN (INT) := 1,

   #-------------------------------------------------------------------------
   # groesster Wert der y-Koordinate
   # YMAX wird unten auf Maximum gesetzt
   #-------------------------------------------------------------------------
   YMAX (INT) := 12

BEGIN

   #-------------------------------------------------------------------------
   # Setzen von XMAX auf Maximum
   #-------------------------------------------------------------------------
   XMAX := Maximum;

   #-------------------------------------------------------------------------
   # Setzen von YMAX auf Maximum
   #-------------------------------------------------------------------------
   YMAX := Maximum;

   #-------------------------------------------------------------------------
   # Vorlaeufiges Bestimmen der x-Koordinate des naechsten Feldes
   #-------------------------------------------------------------------------
   IF (xZiel > xAktuell)
   DO
      xNext := xAktuell + 1;
   END
   ELSIF (xZiel = xAktuell)
   DO
      xNext := xAktuell;
   END
   ELSE
   DO
      xNext := xAktuell - 1;
   END

   #-------------------------------------------------------------------------
   # Vorlaeufiges Bestimmen der y-Koordinate des naechsten Feldes
   #-------------------------------------------------------------------------
   IF (yZiel > yAktuell)
   DO
      yNext := yAktuell + 1;
```

```
END
ELSIF (yZiel = yAktuell)
DO
   yNext := yAktuell;
END
ELSE
DO
   yNext := yAktuell - 1;
END

#-------------------------------------------------------------------------
# Falls es sich um ein Gefahrenfeld handelt: Abaendern der oben
# bestimmten Koordinaten, so dass kein Umweg gemacht wird.
#-------------------------------------------------------------------------
IF (Feldart[xNext][yNext] = 'Gefahr')
DO
   #-------------------------------------------------------------------------
   # Falls der Abstand zum Zielfeld in x-Richtung groesser ist als in
   # y-Richtung: Aendern der y-Koordinate
   #-------------------------------------------------------------------------
   IF (IABS(xZiel - xAktuell) > IABS(yZiel - yAktuell))
   DO
      #-------------------------------------------------------------------------
      # Das Zielfeld befindet sich auf der gleichen Zeile wie das aktuelle
      # Feld.
      #-------------------------------------------------------------------------
      IF (yZiel = yAktuell)
      DO
         IF (yZiel = YMIN)
         DO
            #-------------------------------------------------------------------
            # Sued-Rand des Spielfeldes
            #-------------------------------------------------------------------
            yNext := yAktuell + 1;
         END
         ELSIF (yZiel = YMAX)
         DO
            #-------------------------------------------------------------------
            # Nord-Rand
            #-------------------------------------------------------------------
            yNext := yAktuell - 1;
         END
         ELSE
         DO
            #-------------------------------------------------------------------
            # falls nicht am Rand: immer "noerdlich" vorbei
            #-------------------------------------------------------------------
            yNext := yAktuell + 1;
         END
      END

      #-------------------------------------------------------------------------
      # Das Zielfeld befindet sich nicht auf der gleichen Reihe wie das
      # aktuelle Feld.
      #-------------------------------------------------------------------------
      ELSE
      DO
         #-------------------------------------------------------------------
         # Die y-Koordinate des naechsten Feldes wird geaendert. Anstelle
         # des vorher bestimmten Wertes wird sie auf yAktuell gesetzt.
         #-------------------------------------------------------------------
         yNext := yAktuell;
      END
   END
```

```
      #------------------------------------------------------------------------
      # Falls der Abstand zum Zielfeld in x-Richtung kleiner oder gleich ist
      # als in y-Richtung: Aendern der x-Koordinate.
      #------------------------------------------------------------------------
      ELSE
      DO
         #---------------------------------------------------------------------
         # Das Zielfeld befindet sich auf der gleichen Spalte wie das aktuelle
         # Feld.
         #---------------------------------------------------------------------
         IF (xZiel = xAktuell)
         DO
            IF (xZiel = XMIN)
            DO
               #------------------------------------------------------------
               # West-Rand des Spielfeldes
               #------------------------------------------------------------
               xNext := xAktuell + 1;
            END
            ELSIF (xZiel = XMAX)
            DO
               #------------------------------------------------------------
               # Ost-Rand
               #------------------------------------------------------------
               xNext := xAktuell - 1;
            END
            ELSE
            DO
               #------------------------------------------------------------
               # falls nicht am Rand: immer "oestlich" vorbei
               #------------------------------------------------------------
               xNext := xAktuell + 1;
            END
         END

         #---------------------------------------------------------------------
         # Das Zielfeld befindet sich nicht auf der gleichen Zeile wie das
         # aktuelle Feld: Ausweichen in y-Richtung, d. h. die x-Koordinate
         # bleibt die gleiche wie beim aktuellen Feld.
         #---------------------------------------------------------------------
         ELSE
         DO
            xNext := xAktuell;
         END
      END
   END

   #------------------------------------------------------------------------
   # Funktionskomponente hat ihre Aufgabe erledigt (D. h. in die Variablen
   # zur Rueckgabe "xZiel" und "yZiel" die Koordinaten des ermittelten
   # Teilzieles geschrieben):
   # Also gehts zurueck zur Komponente Cognition
   #------------------------------------------------------------------------
   RETURN

END OF F_Wegeplanung
```