

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Conception et Implémentation d'une Application Distribuée dans le Domaine de la Productique Annexes

Degroot, Michel; Delcourt, Gilles

Award date:
1994

Awarding institution:
Universite de Namur

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
Rue Grandgagnage, 21
B-5000 NAMUR

**Conception et Implémentation
d'une Application Distribuée
dans le Domaine de la Productique**

ANNEXES

**Michel Degroot
Gilles Delcourt**

FM B 16/1994/9/2^②

*nous n'avons jamais
reçu le vol. I*

Promoteur : Eric Dubois

Mémoire présenté en vue
de l'obtention du grade de
Licencié et Maître en Informatique

Année académique 1993-1994

Annexes I: Spécification des besoins	1
1) Version 1 : Le Moniteur voit tout - Les Machines montrent tout	2
a) Les types de données utilisées.....	2
b) Les agents en détail.....	3
L'agent Machine_Agv	3
L'agent Machine_Bridage.....	4
L'agent Machine_Robot	5
L'agent Machine_Tour.....	6
L'agent Machine-Fraiseuse	7
L'agent Moniteur_AGV.....	8
L'agent Moniteur_Bridage.....	9
L'agent Moniteur_Robot.....	10
L'agent Moniteur_Tour.....	11
L'agent Moniteur_Fraiseuse	12
L'agent Manager	13
2) Version 2 : Introduction d'un filtre pour le Manager.....	14
2.1) L'"intelligence" au niveau du moniteur.....	14
L'agent Moniteur_Agv.....	14
L'agent Moniteur_Bridage.....	15
L'agent Moniteur_Robot.....	16
L'agent Moniteur_Tour.....	17
L'agent Moniteur_Fraiseuse	17
2.2) L'"intelligence" au niveau des machines.....	18
L'agent Machine_Agv	18
L'agent Machine-Bridage	19
L'agent Machine-Robot	20
L'agent Machine-Tour	21
L'agent Machine-Fraiseuse	21
3) Version 3 : Actions de communication	22
3.1) Définition des nouveaux types de données.....	22
3.2) Description du système	23
L'agent Machine_Agv	23
L'agent Machine-Bridage	24
L'agent Machine-Robot	25
L'agent Machine-Tour	26
L'agent Machine-Fraiseuse	27
L'agent Moniteur	29
4) Version 4 : Les Problèmes de cohérence	33
4.1) Les machines envoient des informations incohérentes.....	33
a) Les nouveaux types de données utilisées	33
b) Définition des agents	33
L'agent Moniteur	33
4.2) Les machines n'envoient plus d'informations	39
L'agent Machine_Agv	39
L'agent Machine-Bridage	40
L'agent Machine-Robot	40
L'agent Machine-Tour	41

L'agent Machine-Fraiseuse	42
L'agent Moniteur	43
5) Version 5 : Le Technicien	49
a) Définition des nouveaux types de données utilisées.....	49
b) Description des agents	49
L'agent Technicien.....	50
L'agent Moniteur	51
L'agent Machine_Agv	59
L'agent Machine-Bridage	60
L'agent Machine-Robot	61
L'agent Machine-Tour	62
L'agent Machine-Fraiseuse	63
6) Version 6 : Plus de communication directe entre le moniteur et les machines.....	64
a) Définitions des nouveaux types de données.....	64
b) Description des agents	65
L'agent Machine_Agv	65
L'agent Machine-Bridage	66
L'agent Machine-Robot	67
L'agent Machine-Tour	68
L'agent Machine-Fraiseuse	69
L'agent Technicien.....	70
L'agent Dispatcheur	71
L'agent Moniteur	73
Annexes II: Programmation C	80
1) Le routeur et ses procédures	81
1.1) Protocol description file du ROUTEUR	81
1.2) Les procédures du ROUTEUR	81
2) Les serveurs et leurs procédures.....	84
2.1) Protocol Description File du serveur	84
2.2) Les procédures du serveur	84
2.2.1) Initialisation du site local au serveur	84
2.2.2) Procédure principale du serveur	85
3) Les procédures des Clients	86
3.1) Le client du routeur	86
3.2) Le client du serveur	87
4) Procédures diverses	89
4.1) La manipulation des listes	89
4.2) La manipulation du flag.....	90
4.3) Conversion d'une zone au format RPC vers le format OBLOG	90
Annexes III: Spécifications OBLOG	92

Annexes I: Spécification des besoins (Requirements Engineering)

1) Version 1 : Le Moniteur voit tout - Les Machines montrent tout

a) Les types de données utilisées

EQUIPEMENT :

CP : [n°_id : { CLAMP, MILL, TURN, ROBOT },
Status : { IDLE, BUSY, FAULT }]

POSXYZ :

CP : [X : INTEGER,
Y : INTEGER,
Z : INTEGER]

POSXY :

CP : [X : INTEGER,
Y : INTEGER]

PGM :

CP : [N°Prog : INTEGER,
Prog_type : { MPF, SPF, TOA, RPA, NONE },
NomProg : STRING]

VEHICULE :

CP : [Status : { IDLE, BUSY, FAULT },
Id : STRING]

PALLET_POST : { A, B }

DOCK_STATE : { CONTINUE ROTATION, STOPPED ROTATION, UNKNOWN STATE, LOADING, STOPPED LOADING, UNLOADING, STOPPED UNLOADING, STOPPED }

STATION : { CLAMP, MILL, TURN }

DISTANCE : INTEGER

SPEED : { SLOW, FAST, ZERO, UNKNOWN }

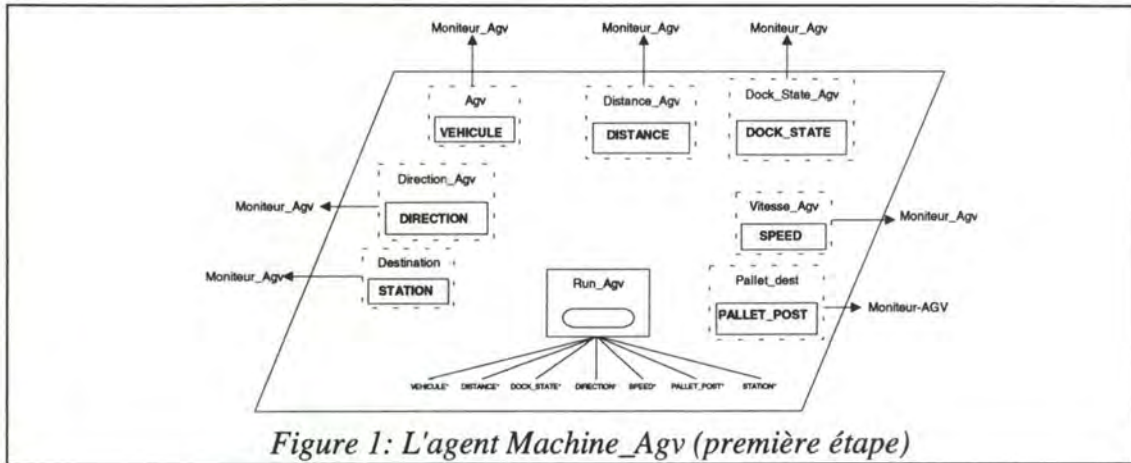
DIRECTION : { +1, -1 }

BRID_STATE : { LOCKED, UNLOCKED, PALLET PRESENT, NO PALLET PRESENT, UNKNOWN STATE }

VITESSE : INTEGER

b) Les agents en détail

L'agent Machine_Agv



- Contraintes de Base

Etat initial

/

Composants dérivés

/

- Contraintes locales

Comportement de l'état

/

Effets des actions

Run_Agv(a,b,c,d,e,f,g): Agv = a
 Distance_Agv = b
 Dock_State_Agv = c
 Direction_Agv = d
 Vitesse_Agv = e
 Pallet_dest = f
 Destination = g

* L'action Run_Agv met à jour les paramètres de la machine

Causalité

/

Capacité

/

- Contraintes de coopération

Perception des actions

/

Perception de l'état

/

Informations sur les actions

/

Informations sur l'état

K(Agv.Moniteur_Agv / TRUE)
 K(Distance_Agv.Moniteur_Agv / TRUE)
 K(Dock_State_Agv.Moniteur_Agv / TRUE)
 K(Vitesse_Agv.Moniteur_Agv / TRUE)
 K(Pallet_dest.Moniteur_Agv / TRUE)

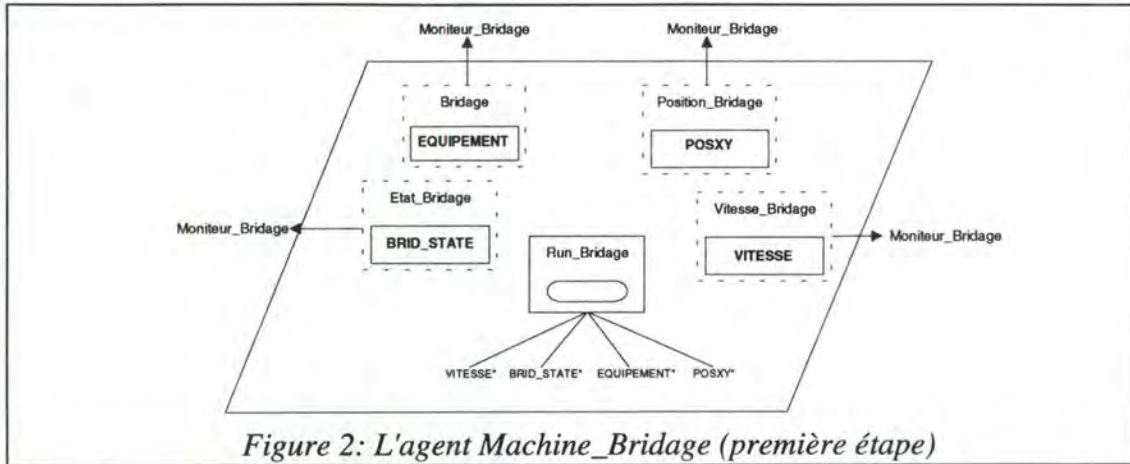
K(Direction_Agv.Moniteur_Agv / TRUE)

K(Destination.Moniteur_Agv / TRUE)

* *On est dans le cas de l'hypothèse d'omniscience, c'est à dire, la*

* *machine montre tout au monitoring.*

L'agent Machine_Bridge



• Contraintes de Base

Etat initial

/

Composants dérivés

/

• Contraintes locales

Comportement de l'état

/

Effets des actions

Run_Bridge(a,b,c,d): Vitesse_Bridge = a

Etat_Bridge = b

Bridge = c

Position_Bridge = d

* *L'action Run_Bridge met à jour les paramètres de la machine*

Causalité

/

Capacité

/

• Contraintes de coopération

Perception des actions

/

Perception de l'état

/

Informations sur les actions

/

Informations sur l'état

K(Bridge.Moniteur_Bridge / TRUE)

K(Position_Bridge.Moniteur_Bridge / TRUE)

K(Etat_Bridge.Moniteur_Bridge / TRUE)

K(Vitesse_Bridge.Moniteur_Bridge / TRUE)

* *Nous sommes est dans le cas de l'hypothèse d'omniscience, c'est à dire,*

** la machine montre tout au monitoring.*

L'agent Machine_Robot

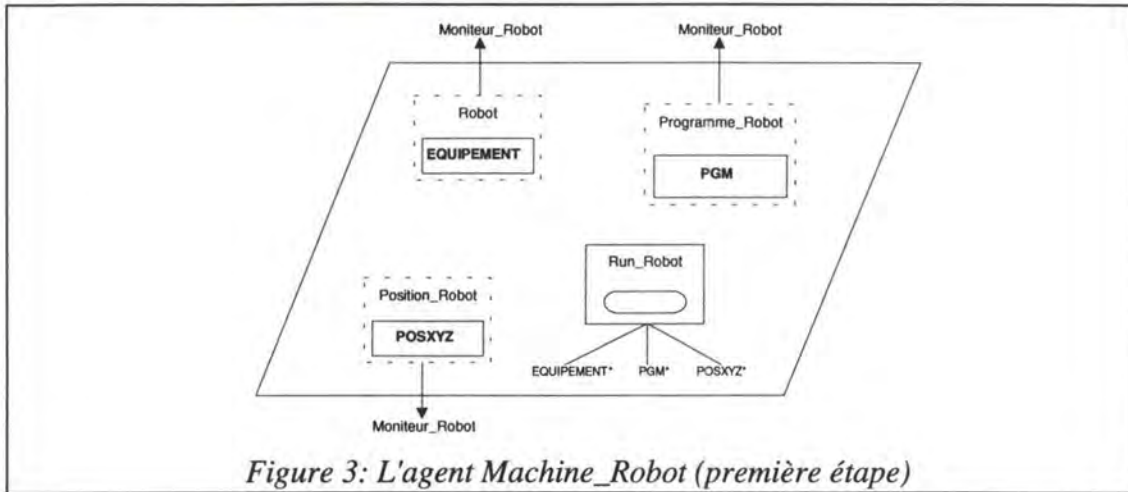


Figure 3: L'agent Machine_Robot (première étape)

- **Contraintes de Base**

- Etat initial

- /

- Composants dérivés

- /

- **Contraintes locales**

- Comportement de l'état

- /

- Effets des actions

Run_Robot(a,b,c): Robot = a
 Programme_Robot = b
 Position_Robot = c

** L'action Run_Robot met à jour les paramètres de la machine*

- Causalité

- /

- Capacité

- /

- **Contraintes de coopération**

- Perception des actions

- /

- Perception de l'état

- /

- Informations sur les actions

- /

- Informations sur l'état

K(Robot.Moniteur_Robot / TRUE)
 K(Programme_Robot.Moniteur_Robot / TRUE)
 K(Position_Robot.Moniteur_Robot / TRUE)

** On est dans le cas de l'hypothèse d'omniscience, c'est à dire, la
 * machine montre tout au monitoring.*

L'agent Machine_Tour

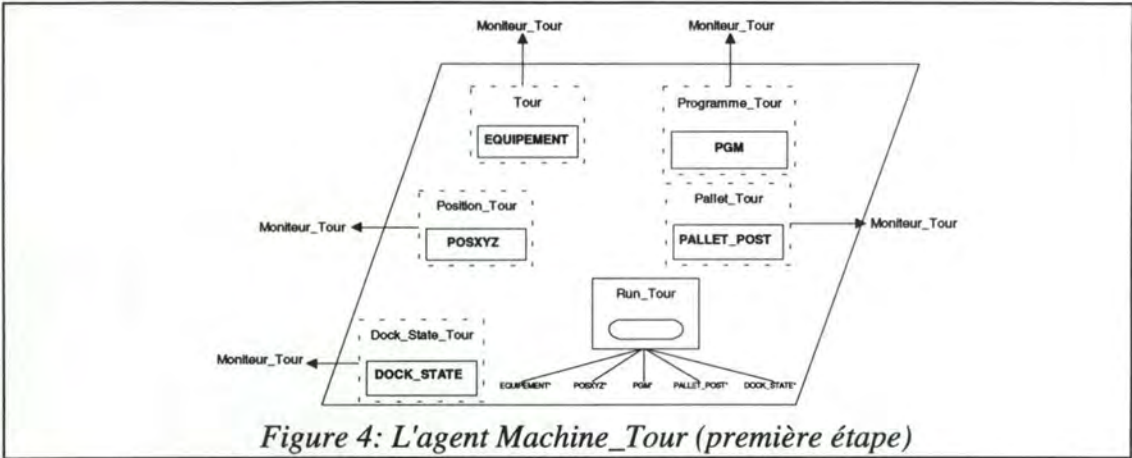


Figure 4: L'agent Machine_Tour (première étape)

• Contraintes de Base

Etat initial

/

Composants dérivés

/

• Contraintes locales

Comportement de l'état

/

Effets des actions

Run_Tour(a,b,c,d,e): Tour = a
 Position_Tour = b
 Programme_Tour = c
 Pallet_Tour = d
 Dock_State_Tour = e

* L'action Run_Tour met à jour les paramètres de la machine

Causalité

/

Capacité

/

• Contraintes de coopération

Perception des actions

/

Perception de l'état

/

Informations sur les actions

/

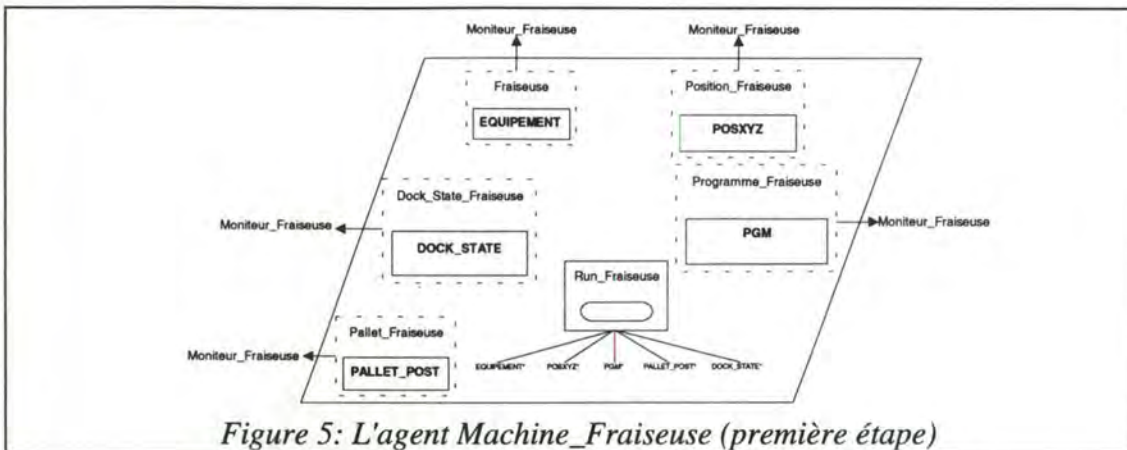
Informations sur l'état

K(Tour.Moniteur_Tour / TRUE)
K(Programme_Tour.Moniteur_Tour / TRUE)
K(Pallet_Tour.Moniteur_Tour / TRUE)
K(Position_Tour.Moniteur_Tour / TRUE)
K(Dock_State_Tour.Moniteur_Tour / TRUE)

* On est dans le cas de l'hypothèse d'omniscience, c'est à dire, la

* machine montre tout au monitoring.

L'agent Machine-Fraiseuse



- **Contraintes de Base**

- Etat initial

- /

- Composants dérivés

- /

- **Contraintes locales**

- Comportement de l'état

- /

- Effets des actions

Run_Fraiseuse(a,b,c,d,e): Fraiseuse = a
 Position_Fraiseuse = b
 Programme_Fraiseuse = c
 Pallet_Fraiseuse = d
 Dock_State_Fraiseuse = e

* L'action Run_Fraiseuse met à jour les paramètres de la machine

- Causalité

- /

- Capacité

- /

- **Contraintes de coopération**

- Perception des actions

- /

- Perception de l'état

- /

- Informations sur les actions

- /

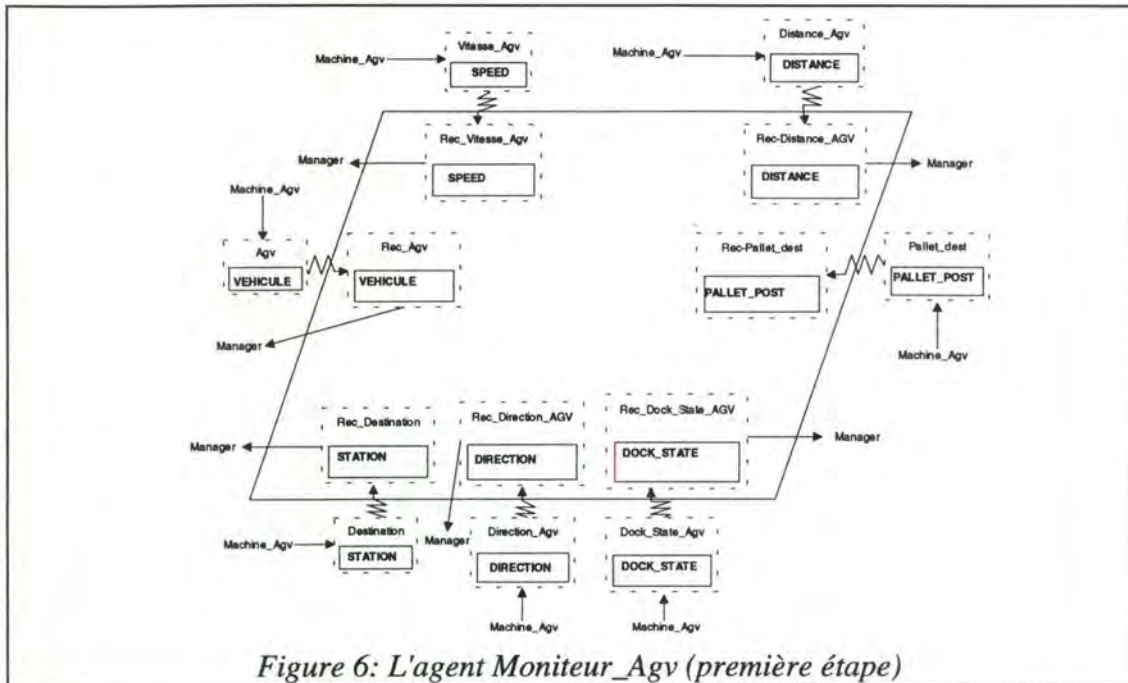
- Informations sur l'état

K(Fraiseuse.Moniteur_Fraiseuse / TRUE)
 K(Position_Fraiseuse.Moniteur_Fraiseuse / TRUE)
 K(Programme_Fraiseuse.Moniteur_Fraiseuse / TRUE)
 K(Pallet_Fraiseuse.Moniteur_Fraiseuse / TRUE)
 K(Dock_State_Fraiseuse.Moniteur_Fraiseuse / TRUE)

* Nous sommes dans le cas de l'hypothèse d'omniscience, c'est à dire, la

* machine montre tout au monitoring.

L'agent Moniteur AGV



- **Contraintes de Base**

- Etat initial

/

- Composants dérivés

Agv : Rec_Agv = Agv

Distance_Agv : Rec_Distance_Agv = Distance_Agv

Dock_State_Agv : Rec_Dock_State_Agv = Dock_State_Agv

Direction_Agv : Rec_Direction_Agv = Direction_Agv

Destination_Agv : Rec_Destination_Agv = Destination_Agv

Vitesse_Agv : Rec_Vitesse_Agv = Vitesse_Agv

Pallet_dest : Rec_Pallet_dest = Pallet_dest

- **Contraintes locales**

- Comportement de l'état

/

- Effets des actions

/

- Causalité

/

- Capacité

/

- **Contraintes de coopération**

- Perception des actions

/

- Perception de l'état

K(Machine_Agv.Agv / TRUE)

K(Machine_Agv.Distance_Agv / TRUE)

K(Machine_Agv.Dock_State_Agv / TRUE)

K(Machine_Agv.Vitesse_Agv / TRUE)

K(Machine_Agv.Pallet_dest / TRUE)

K(Machine_Agv.Direction_Agv / TRUE)

K(Machine_Agv.Destination / TRUE)

* *L'agent tient compte de ce qu'il perçoit*

Informations sur les actions

/

Informations sur l'état

K(Rec_Agv.Manager / TRUE)
 K(Rec_Distance_Agv.Manager / TRUE)
 K(Rec_Dock_State_Agv.Manager / TRUE)
 K(Rec_Vitesse_Agv.Manager / TRUE)
 K(Rec_Pallet_dest.Manager / TRUE)
 K(Rec_Direction_Agv.Manager / TRUE)
 K(Rec_Destination.Manager / TRUE)

* *L'agent de monitoring montre tout au manager*

L'agent Moniteur Bridge

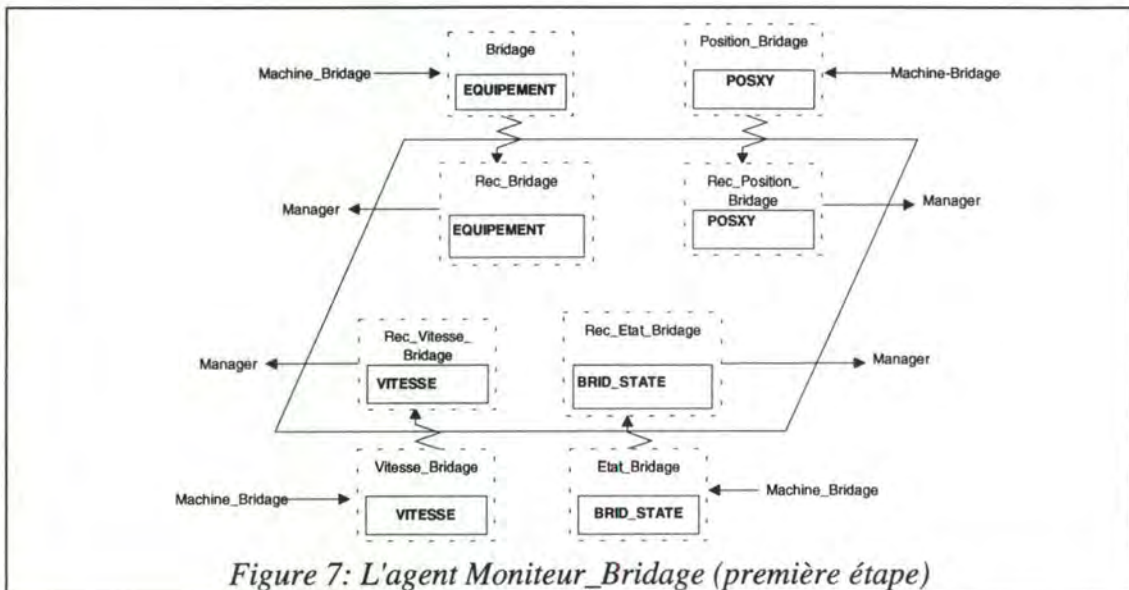


Figure 7: L'agent Moniteur_Bridge (première étape)

• **Contraintes de Base**

Etat initial

/

Composants dérivés

Bridge : Rec_Bridge = Bridge
 Position_Bridge : Rec_Position_Bridge = Position_Bridge
 Etat_Bridge : Rec_Etat_Bridge = Etat_Bridge
 Vitesse_Bridge : Rec_Vitesse_Bridge = Vitesse_Bridge

• **Contraintes locales**

Comportement de l'état

/

Effets des actions

/

Causalité

/

Capacité

/

• **Contraintes de coopération**

Perception des actions

/

Perception de l'état

K(Machine_Bridge.Bridge / TRUE)
 K(Machine_Bridge.Position_Bridge / TRUE)
 K(Machine_Bridge.Etat_Bridge / TRUE)
 K(Machine_Bridge.Vitesse_Bridge / TRUE)

** L'agent tient compte de ce qu'il perçoit*

Informations sur les actions

/

Informations sur l'état

K(Rec_Bridge.Manager / TRUE)
 K(Rec_Position_Bridge.Manager / TRUE)
 K(Rec_Etat_Bridge.Manager / TRUE)
 K(Rec_Vitesse_Bridge.Manager / TRUE)

** L'agent de monitoring montre tout au manager*

L'agent Moniteur Robot

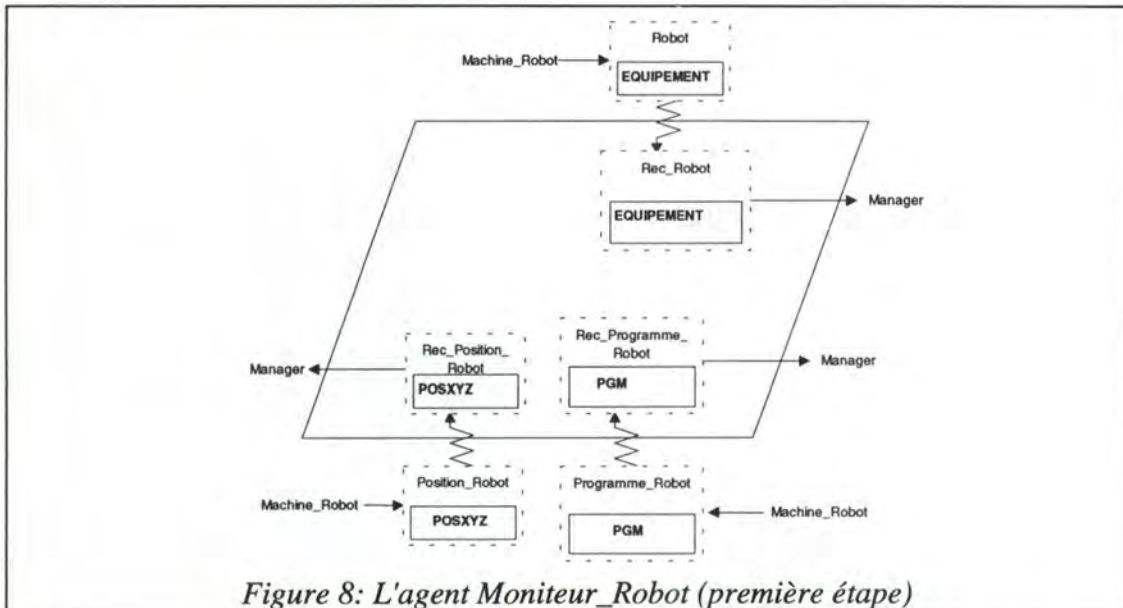


Figure 8: L'agent Moniteur_Robot (première étape)

• **Contraintes de Base**

Etat initial

/

Composants dérivés

Robot : Rec_Robot = Robot
 Position_Robot : Rec_Position_Robot = Position_Robot
 Programme_Robot : Rec_Programme_Robot = Programme_Robot

• **Contraintes locales**

Comportement de l'état

/

Effets des actions

/

Causalité

/

Capacité

/

- Contraintes de coopération

- Perception des actions

/

- Perception de l'état

K(Machine_Robot.Robot / TRUE)

K(Machine_Robot.Programme_Robot / TRUE)

K(Machine_Robot.Position_Robot / TRUE)

* L'agent tient compte de ce qu'il perçoit

- Informations sur les actions

/

- Informations sur l'état

K(Rec_Robot.Manager / TRUE)

K(Rec_Programme_Robot.Manager / TRUE)

K(Rec_Position_Robot.Manager / TRUE)

* L'agent de monitoring montre tout au manager

L'agent Moniteur_Tour

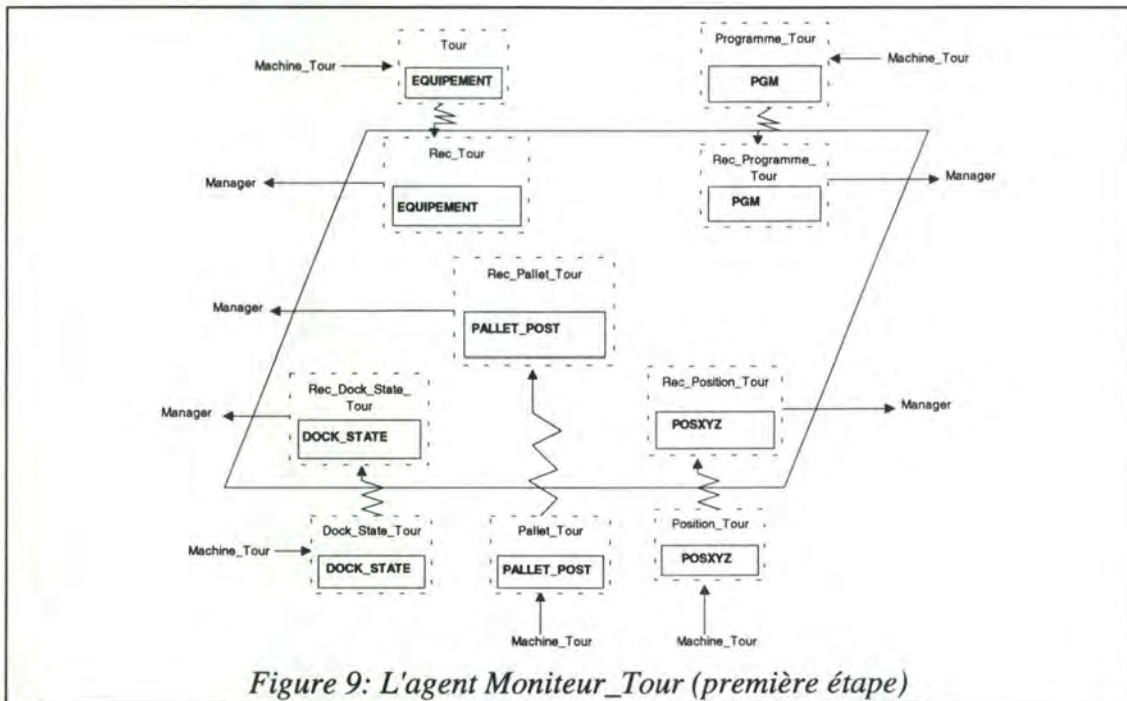


Figure 9: L'agent Moniteur_Tour (première étape)

- Contraintes de Base

- Etat initial

/

- Composants dérivés

Tour : Rec_Tour = Tour

Programme_Tour : Rec_Programme_Tour = Programme_Tour

Dock_State_Tour : Rec_Dock_State_Tour = Dock_State_Tour

Position_Tour : Rec_Position_Tour = Position_Tour

Pallet_Tour : Rec_Pallet_Tour = Pallet_Tour

- Contraintes locales

- Comportement de l'état

/

- Effets des actions

/

Causalité

/

Capacité

/

• **Contraintes de coopération**

Perception des actions

/

Perception de l'état

K(Machine_Tour.Tour / TRUE)
 K(Machine_Tour.Programme_Tour / TRUE)
 K(Machine_Tour.Pallet_Tour / TRUE)
 K(Machine_Tour.Position_Tour / TRUE)
 K(Machine_Tour.Dock_State_Tour / TRUE)

* *L'agent tient compte de ce qu'il perçoit*

Informations sur les actions

/

Informations sur l'état

K(Tour.Manager / TRUE)
 K(Programme_Tour.Manager / TRUE)
 K(Pallet_Tour.Manager / TRUE)
 K(Position_Tour.Manager / TRUE)
 K(Dock_State_Tour.Manager / TRUE)

* *L'agent de monitoring montre tout au manager*

L'agent Moniteur Fraiseuse

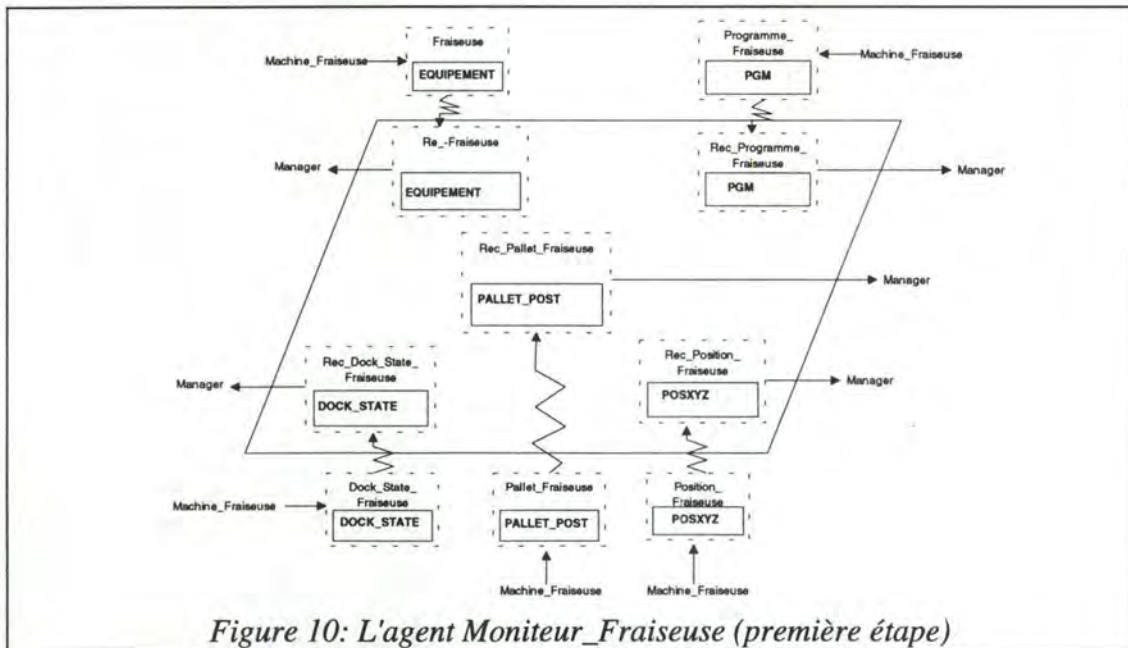


Figure 10: L'agent Moniteur_Fraiseuse (première étape)

• **Contraintes de Base**

Etat initial

/

Composants dérivés

Fraiseuse : Rec_Fraiseuse = Fraiseuse
 Programme_Fraiseuse : Rec_Programme_Fraiseuse = Programme_Fraiseuse
 Dock_State_Fraiseuse : Rec_Dock_State_Fraiseuse = Dock_State_Fraiseuse
 Position_Fraiseuse : Rec_Position_Fraiseuse = Position_Fraiseuse
 Pallet_Fraiseuse : Rec_Pallet_Fraiseuse = Pallet_Fraiseuse

• Contraintes locales

Comportement de l'état

/

Effets des actions

/

Causalité

/

Capacité

/

• Contraintes de coopération

Perception des actions

/

Perception de l'état

K(Machine_Fraiseuse.Fraiseuse / TRUE)

K(Machine_Fraiseuse.Position_Fraiseuse / TRUE)

K(Machine_Fraiseuse.Programme_Fraiseuse / TRUE)

K(Machine_Fraiseuse.Pallet_Fraiseuse / TRUE)

K(Machine_Fraiseuse.Dock_State_Fraiseuse / TRUE)

* *L'agent tient compte de ce qu'il perçoit*

Informations sur les actions

/

Informations sur l'état

K(Fraiseuse.Manager / TRUE)

K(Position_Fraiseuse.Manager / TRUE)

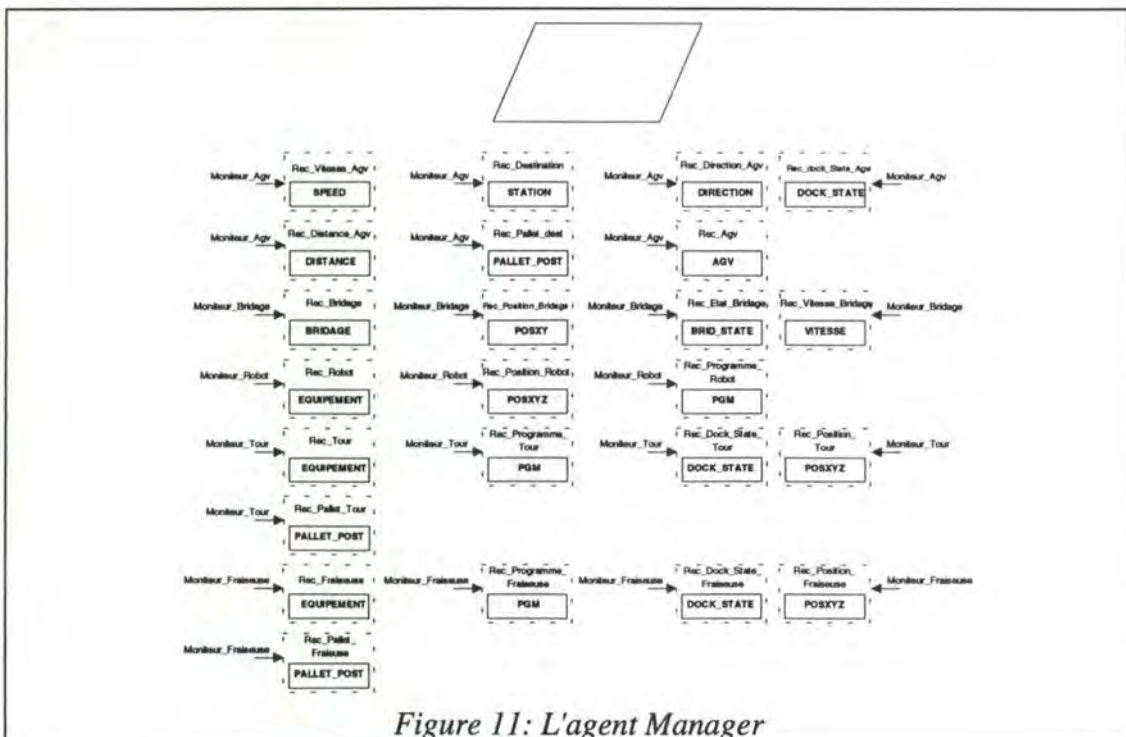
K(Programme_Fraiseuse.Manager / TRUE)

K(Pallet_Fraiseuse.Manager / TRUE)

K(Dock_State_Fraiseuse.Manager / TRUE)

* *L'agent de monitoring montre tout au manager*

L'agent Manager



- Contraintes de Base

- Etat initial

- /

- Composants dérivés

- /

- Contraintes locales

- Comportement de l'état

- /

- Effets des actions

- /

- Causalité

- /

- Capacité

- /

- Contraintes de coopération

- Perception des actions

- /

- Perception de l'état

- /

- Informations sur les actions

- /

- Informations sur l'état

- /

2) Version 2 : Introduction d'un filtre pour le Manager

2.1) L'"intelligence" au niveau du moniteur

L'agent Moniteur_Agv

- Contraintes de Base

- Etat initial

- /

- Composants dérivés

Agv : Rec_Agv = Agv

Distance_Agv : Rec_Distance_Agv = Distance_Agv

Dock_State_Agv : Rec_Dock_State_Agv = Dock_State_Agv

Direction_Agv : Rec_Direction_Agv = Direction_Agv

Destination_Agv : Rec_Destination_Agv = Destination_Agv

Vitesse_Agv : Rec_Vitesse_Agv = Vitesse_Agv

Pallet_dest : Rec_Pallet_dest = Pallet_dest,

Rec_Distance_Agv = UNDEF $\equiv$ Status(Rec_Agv) = FAULT

Rec_Dock_State_Agv = UNDEF $\equiv$ Status(Rec_Agv) = FAULT

Rec_Direction_Agv = UNDEF $\equiv$ Status(Rec_Agv) = FAULT

Rec_Destination_Agv = UNDEF $\equiv$ Status(Rec_Agv) = FAULT

Rec_Vitesse_Agv = UNDEF $\equiv$ Status(Rec_Agv) = FAULT

Rec_Pallet_dest = UNDEF $\equiv$ Status(Rec_Agv) = FAULT

Rec_Distance_Agv = UNDEF $\equiv$ Status(Rec_Agv) = IDLE

Rec_Direction_Agv = UNDEF $\equiv$ Status(Rec_Agv) = IDLE

Rec_Destination_Agv = UNDEF $\equiv$ Status(Rec_Agv) = IDLE

Rec_Pallet_dest = UNDEF $\equiv$ Status(Rec_Agv) = IDLE

Rec_Pallet_dest = UNDEF $\equiv$ Rec_Destination_Agv = CLAMP

** Toutes les données non pertinentes sont mises à UNDEF*

• **Contraintes locales**

Comportement de l'état

/

Effets des actions

/

Causalité

/

Capacité

/

• **Contraintes de coopération**

Perception des actions

/

Perception de l'état

K(Machine_Agv.Agv / TRUE)

K(Machine_Agv.Distance_Agv / TRUE)

K(Machine_Agv.Dock_State_Agv / TRUE)

K(Machine_Agv.Vitesse_Agv / TRUE)

K(Machine_Agv.Pallet_dest / TRUE)

K(Machine_Agv.Direction_Agv / TRUE)

K(Machine_Agv.Destination / TRUE)

** L'agent tient compte de ce qu'il perçoit*

Informations sur les actions

/

Informations sur l'état

K(Rec_Agv.Manager / TRUE)

K(Rec_Distance_Agv.Manager / TRUE)

K(Rec_Dock_State_Agv.Manager / TRUE)

K(Rec_Vitesse_Agv.Manager / TRUE)

K(Rec_Pallet_dest.Manager / TRUE)

K(Rec_Direction_Agv.Manager / TRUE)

K(Rec_Destination.Manager / TRUE)

** L'agent de monitoring montre tout au manager*

L'agent Moniteur Bridage

• **Contraintes de Base**

Etat initial

/

Composants dérivés

Bridage : Rec_Bridage = Bridage

Position_Bridage : Rec_Position_Bridage = Position_Bridage

Etat_Bridage : Rec_Etat_Bridage = Etat_Bridage

Vitesse_Bridage : Rec_Vitesse_Bridage = Vitesse_Bridage

Rec_Position_Bridage = UNDEF $\equiv$ Status(Rec_Bridage) = FAULT

Rec_Etat_Bridage = UNDEF $\equiv$ Status(Rec_Bridage) = FAULT

Rec_Vitesse_Bridage = UNDEF $\equiv$ Status(Rec_Bridage) = FAULT

** Toutes les données non pertinentes sont mises à UNDEF*

• **Contraintes locales**

Comportement de l'état

/

Effets des actions

/

Causalité

/

Capacité

/

• **Contraintes de coopération**

Perception des actions

/

Perception de l'état

K(Machine_Bridge.Bridge / TRUE)

K(Machine_Bridge.Position_Bridge / TRUE)

K(Machine_Bridge.Etat_Bridge / TRUE)

K(Machine_Bridge.Vitesse_Bridge / TRUE)

** L'agent tient compte de ce qu'il perçoit*

Informations sur les actions

/

Informations sur l'état

K(Rec_Bridge.Manager / TRUE)

K(Rec_Position_Bridge.Manager / TRUE)

K(Rec_Etat_Bridge.Manager / TRUE)

K(Rec_Vitesse_Bridge.Manager / TRUE)

** L'agent de monitoring montre tout au manager*

L'agent Moniteur Robot

• **Contraintes de Base**

Etat initial

/

Composants dérivés

Robot : Rec_Robot = Robot

Position_Robot : Rec_Position_Robot = Position_Robot

Programme_Robot : Rec_Programme_Robot = Programme_Robot

Rec_Position_Robot = UNDEF $\equiv$ Status(Rec_Robot) = FAULT

Rec_Programme_Robot = UNDEF $\equiv$ Status(Rec_Robot) = FAULT

Rec_Programme_Robot = UNDEF $\equiv$ Status(Rec_Robot) = IDLE

** Toutes les données non pertinentes sont mises à UNDEF*

• **Contraintes locales**

Comportement de l'état

/

Effets des actions

/

Causalité

/

Capacité

/

• **Contraintes de coopération**

Perception des actions

/

Perception de l'état

K(Machine_Robot.Robot / TRUE)

K(Machine_Robot.Programme_Robot / TRUE)

K(Machine_Robot.Position_Robot / TRUE)

** L'agent tient compte de ce qu'il perçoit*

Informations sur les actions

/

Informations sur l'état

K(Rec_Robot.Manager / TRUE)

K(Rec_Programme_Robot.Manager / TRUE)

K(Rec_Position_Robot.Manager / TRUE)

** L'agent de monitoring montre tout au manager*

L'agent Moniteur Tour

• Contraintes de Base

Etat initial

/

Composants dérivés

Tour : Rec_Tour = Tour

Programme_Tour : Rec_Programme_Tour = Programme_Tour

Dock_State_Tour : Rec_Dock_State_Tour = Dock_State_Tour

Position_Tour : Rec_Position_Tour = Position_Tour

Pallet_Tour : Rec_Pallet_Tour = Pallet_Tour

Rec_Programme_Tour = UNDEF $\equiv$ Status(Rec_Tour) = FAULT

Rec_Dock_State_Tour = UNDEF $\equiv$ Status(Rec_Tour) = FAULT

Rec_Position_Tour = UNDEF $\equiv$ Status(Rec_Tour) = FAULT

Rec_Pallet_Tour = UNDEF $\equiv$ Status(Rec_Tour) = FAULT

Rec_Programme_Tour = UNDEF $\equiv$ Status(Rec_Tour) = IDLE

** Toutes les données non pertinentes sont mises à UNDEF*

• Contraintes locales

Comportement de l'état

/

Effets des actions

/

Causalité

/

Capacité

/

• Contraintes de coopération

Perception des actions

/

Perception de l'état

K(Machine_Tour.Tour / TRUE)

K(Machine_Tour.Programme_Tour / TRUE)

K(Machine_Tour.Pallet_Tour / TRUE)

K(Machine_Tour.Position_Tour / TRUE)

K(Machine_Tour.Dock_State_Tour / TRUE)

** L'agent tient compte de ce qu'il perçoit*

Informations sur les actions

/

Informations sur l'état

K(Tour.Manager / TRUE)

K(Programme_Tour.Manager / TRUE)

K(Pallet_Tour.Manager / TRUE)

K(Position_Tour.Manager / TRUE)

K(Dock_State_Tour.Manager / TRUE)

** L'agent de monitoring montre tout au manager*

L'agent Moniteur Fraiseuse

• Contraintes de Base

Etat initial

/

Composants dérivés

Fraiseuse : Rec_Fraiseuse = Fraiseuse
 Programme_Fraiseuse : Rec_Programme_Fraiseuse = Programme_Fraiseuse
 Dock_State_Fraiseuse : Rec_Dock_State_Fraiseuse = Dock_State_Fraiseuse
 Position_Fraiseuse : Rec_Position_Fraiseuse = Position_Fraiseuse
 Pallet_Fraiseuse : Rec_Pallet_Fraiseuse = Pallet_Fraiseuse
 Rec_Programme_Fraiseuse = UNDEF $\equiv$ Status(Rec_Fraiseuse) = FAULT
 Rec_Dock_State_Fraiseuse = UNDEF $\equiv$ Status(Rec_Fraiseuse) = FAULT
 Rec_Position_Fraiseuse = UNDEF $\equiv$ Status(Rec_Fraiseuse) = FAULT
 Rec_Pallet_Fraiseuse = UNDEF $\equiv$ Status(Rec_Fraiseuse) = FAULT
 Rec_Programme_Fraiseuse = UNDEF $\equiv$ Status(Rec_Fraiseuse) = IDLE

** Toutes les données non pertinentes sont mises à UNDEF*

• **Contraintes locales**

Comportement de l'état

/

Effets des actions

/

Causalité

/

Capacité

/

• **Contraintes de coopération**

Perception des actions

/

Perception de l'état

K(Machine_Fraiseuse.Fraiseuse / TRUE)
 K(Machine_Fraiseuse.Position_Fraiseuse / TRUE)
 K(Machine_Fraiseuse.Programme_Fraiseuse / TRUE)
 K(Machine_Fraiseuse.Pallet_Fraiseuse / TRUE)
 K(Machine_Fraiseuse.Dock_State_Fraiseuse / TRUE)

** L'agent tient compte de ce qu'il perçoit*

Informations sur les actions

/

Informations sur l'état

K(Fraiseuse.Manager / TRUE)
 K(Position_Fraiseuse.Manager / TRUE)
 K(Programme_Fraiseuse.Manager / TRUE)
 K(Pallet_Fraiseuse.Manager / TRUE)
 K(Dock_State_Fraiseuse.Manager / TRUE)

** L'agent de monitoring montre tout au manager*

2.2) L'"intelligence" au niveau des machines

L'agent Machine Agv

• **Contraintes de Base**

Etat initial

/

Composants dérivés

/

• **Contraintes locales**

Comportement de l'état

/

Effets des actions

Run_Agv(a,b,c,d,e,f,g): Agv = a
 Distance_Agv = b
 Dock_State_Agv = c
 Direction_Agv = d
 Vitesse_Agv = e
 Pallet_dest = f
 Destination = g

** L'action Run_Agv met à jour les paramètres de la machine*

Causalité

/

Capacité

/

• Contraintes de coopération

Perception des actions

/

Perception de l'état

/

Informations sur les actions

/

Informations sur l'état

K(Agv.Moniteur_Agv / TRUE)

** Agv est toujours visible*

I(Vitesse_Agv.Moniteur_Agv / Status(Agv) = FAULT)

I(Dock_State_Agv.Moniteur_Agv / Status(Agv) = FAULT)

I(Direction_Agv.Moniteur_Agv / Status(Agv) = FAULT)

I(Destination.Moniteur_Agv / Status(Agv) = FAULT)

I(Pallet_dest.Moniteur_Agv / Status(Agv) = FAULT)

I(Distance_Agv.Moniteur_Agv / Status(Agv) = FAULT)

** Si Status(Agv) = FAULT, on ne montre rien d'autre que Agv.*

K(Vitesse_Agv.Moniteur_Agv / Status(Agv) = IDLE)

K(Dock_State_Agv.Moniteur_Agv / Status(Agv) = IDLE)

** Si Status(Agv) = IDLE, on rend visible en plus de Agv, Vitesse_Agv et*

** Dock_state.*

K(Vitesse_Agv.Moniteur_Agv / Status(Agv) = BUSY)

K(Dock_State_Agv.Moniteur_Agv / Status(Agv) = BUSY)

XX(Direction_Agv.Moniteur_Agv / Status(Agv) = BUSY)

XX(Destination.Moniteur_Agv / Status(Agv) = BUSY)

XX(Pallet_dest.Moniteur_Agv / (Status(Agv) = BUSY) $\wedge$ (Destination $\neq$ CLAMP))

XX(Distance_Agv.Moniteur_Agv / Status(Agv) = BUSY)

** Si Status(Agv) = BUSY, on rend tout visible sauf Pallet_dest si*

** Destination = CLAMP.*

L'agent Machine-Bridage

• Contraintes de Base

Etat initial

/

Composants dérivés

/

• Contraintes locales

Comportement de l'état

/

Effets des actions

Run_Bridage(a,b,c,d): Vitesse_Bridage = a
 Etat_Bridage = b
 Bridage = c
 Position_Bridage = d

** L'action Run_Bridage met à jour les paramètres de la machine*

Causalité

/

Capacité

/

• Contraintes de coopération

Perception des actions

/

Perception de l'état

/

Informations sur les actions

/

Informations sur l'état

XK(Bridage.Moniteur_Bridage / TRUE)

** Bridage est toujours visible*

XK(Bridage.Moniteur_Bridage / Status(Bridage) ≠ FAULT)

XK(Position_Bridage.Moniteur_Bridage / Status(Bridage) ≠ FAULT)

XK(Etat_Bridage.Moniteur_Bridage / Status(Bridage) ≠ FAULT)

XK(Vitesse_Bridage.Moniteur_Bridage / Status(Bridage) ≠ FAULT)

** Si le statut du bridage est différent de FAULT, alors on doit tout montrer*

L'agent Machine-Robot

• Contraintes de Base

Etat initial

/

Composants dérivés

/

• Contraintes locales

Comportement de l'état

/

Effets des actions

Run_Robot(a,b,c): Robot = a
 Programme_Robot = b
 Position_Robot = c

** L'action Run_Robot met à jour les paramètres de la machine*

Causalité

/

Capacité

/

• Contraintes de coopération

Perception des actions

/

Perception de l'état

/

Informations sur les actions

/

Informations sur l'état

K(Robot.Moniteur_Robot / TRUE)

* *Robot est toujours visible*

XK(Programme_Robot.Moniteur_Robot / Status(Robot) = BUSY)

* *Le robot ne montre le programme que si son statut est BUSY (cela*

signifie qu'il l'exécute).

XK(Position_Robot.Moniteur_Robot / Status(Robot) ≠ FAULT)

* *Le robot ne montre sa position que si son statut est autre que FAULT*

L'agent Machine-Tour

• **Contraintes de Base**

Etat initial

/

Composants dérivés

/

• **Contraintes locales**

Comportement de l'état

/

Effets des actions

Run_Tour(a,b,c,d,e): Tour = a
 Position_Tour = b
 Programme_Tour = c
 Pallet_Tour = d
 Dock_State_Tour = e

* *L'action Run_Tour met à jour les paramètres de la machine*

Causalité

/

Capacité

/

• **Contraintes de coopération**

Perception des actions

/

Perception de l'état

/

Informations sur les actions

/

Informations sur l'état

K(Tour.Moniteur_Tour / TRUE)

* *Tour est toujours visible.*

XK(Programme_Tour.Moniteur_Tour / Status(Tour) = BUSY)

* *Le tour ne montre le programme que si son statut est BUSY (cela*

signifie qu'il l'exécute).

XK(Dock_State_Tour.Moniteur_Tour / Status(Tour) ≠ FAULT)

XK(Pallet_Tour.Moniteur_Tour / Status(Tour) ≠ FAULT)

XK(Position_Tour.Moniteur_Tour / Status(Tour) ≠ FAULT)

* *Le tour ne montre Dock_State, Pallet_Tour, Position_Tour que si son*

statut est différent de FAULT.

L'agent Machine-Fraiseuse

- Contraintes de Base

- Etat initial

- /

- Composants dérivés

- /

- Contraintes locales

- Comportement de l'état

- /

- Effets des actions

Run_Fraiseuse(a,b,c,d,e): Fraiseuse = a
 Position_Fraiseuse = b
 Programme_Fraiseuse = c
 Pallet_Fraiseuse = d
 Dock_State_Fraiseuse = e

* *L'action Run_Fraiseuse met à jour les paramètres de la machine*

- Causalité

- /

- Capacité

- /

- Contraintes de coopération

- Perception des actions

- /

- Perception de l'état

- /

- Informations sur les actions

- /

- Informations sur l'état

K(Fraiseuse.Moniteur_Fraiseuse / TRUE)

* *Fraiseuse est toujours visible.*

XK(Programme_Fraiseuse.Moniteur_Fraiseuse / Status(Fraiseuse) = BUSY)

* *Le tour ne montre le programme que si son statut est BUSY (cela signifie qu'il l'exécute).*

XK(Dock_State_Fraiseuse.Moniteur_Fraiseuse / Status(Fraiseuse) ≠ FAULT)

XK(Pallet_Fraiseuse.Moniteur_Fraiseuse / Status(Fraiseuse) ≠ FAULT)

XK(Position_Fraiseuse.Moniteur_Fraiseuse / Status(Fraiseuse) ≠ FAULT)

* *La fraiseuse ne montre Dock_State, Pallet_Tour, Position_Tour que si son statut est différent de FAULT.*

3) Version 3 : Actions de communication

3.1) Définition des nouveaux types de données

INFO_TOUR:

CP: [PGM: *PGM*,
 Pallet_Post: *PALLET_POST*,
 Dock_State: *DOCK_STATE*,
 POSXYZ: *POSHXYZ*]

INFO_FRAISE:

CP: [PGM: **PGM**,
 Pallet_Post: **PALLET_POST**,
 Dock_State: **DOCK_STATE**,
 POSXYZ: **POSHYZ**]

INFO_Agv:

CP: [Station: **STATION**,
 Pallet_Post: **PALLET_POST**,
 Dock_State: **DOCK_STATE**,
 Distance: **DISTANCE**,
 Speed: **SPEED**,
 Direction: **DIRECTION**]

INFO_BRIDAGE:

CP: [Brid_State: **BRID_STATE**,
 Vitesse: **VITESSE**,
 Posxy: **POSHY**]

INFO_ROBOT:

CP: [PGM: **PGM**,
 POSXYZ: **POSHYZ**]

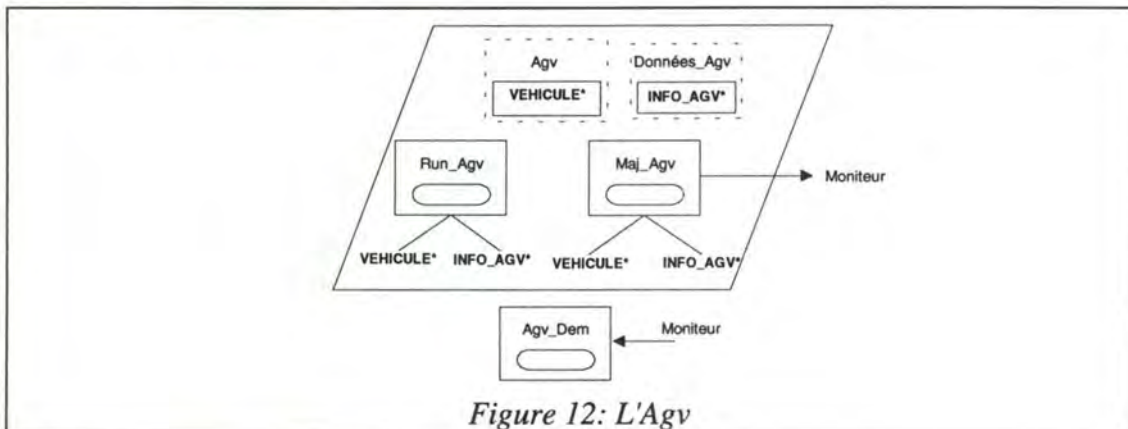
3.2) Description du système**L'agent Machine Agv**

Figure 12: L'Agv

• **Contraintes de Base****Etat initial**

/

Composants dérivés

/

• **Contraintes locales****Comportement de l'état**

/

Effets des actions

Run_Agv(a,b): Agv = a
 Données_Agv = b

* L'action Run met à jour les paramètres de la machine.

Causalité

Moniteur.Agv_Dem $\Diamond \leq 1s \rightarrow$ Maj_Agv(vehicule,infos) with ((equip = Agv)
 $\wedge$ (infos = Données_Agv))

- * L'action Maj_Agv() a lieu avec les données correspondant
- * à celles de la machine concernée.

Capacité

F (Run_Agv(.,.) / Status(Agv) = FAULT)

- * L'action run_Agv ne peut avoir lieu si le statut de la machine
- * est à FAULT.

• **Contraintes de coopération**

Perception des actions

K (Moniteur.Agv_Dem() / TRUE)

- * La demande d'information venant du moniteur doit être prise
- * en compte par la machine.

Perception de l'état

/

Informations sur les actions

K (Maj_Agv(vehicule,infos).Moniteur / TRUE)

- * Lorsque la machine répond à une demande d'information, le moniteur
- * en est informé.

Informations sur l'état

/

L'agent Machine-Bridge

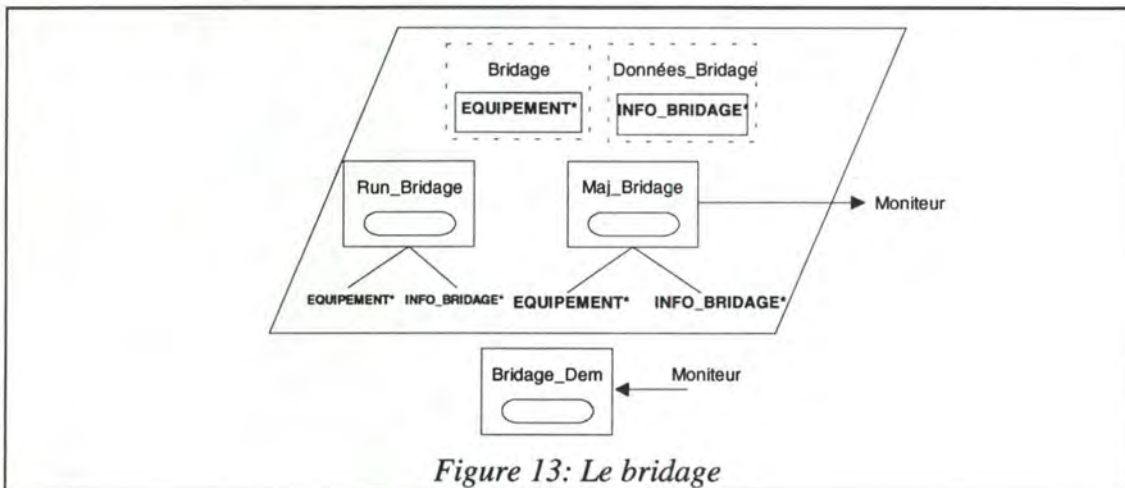


Figure 13: Le bridge

• **Contraintes de Base**

Etat initial

/

Composants dérivés

/

• **Contraintes locales**

Comportement de l'état

/

Effets des actions

Run_Bridge(a,b): Bridge = a
Données_Bridge = b

* *L'action Run met à jour les paramètres de la machine.*

Causalité

Moniteur.Bridge_Dem $\hat{Q} \leq 1s \rightarrow$ Maj_Bridge(equip,infos) with ((equip = Bridge)
 $\wedge$ (infos = Données_Bridge))

* *L'action Maj_Bridge() a lieu avec les données correspondant*
* *à celles de la machine concernée.*

Capacité

F (Run_Bridge(.,.) / Status(Bridge) = FAULT)

* *L'action run_Agv ne peut avoir lieu si le statut de la machine*
* *est à fault.*

- **Contraintes de coopération**

Perception des actions

K (Moniteur.Bridge_Dem() / TRUE)

* *La demande d'information venant du moniteur doit être prise*
* *en compte par la machine.*

Perception de l'état

/

Informations sur les actions

K (Maj_Bridge(equip,infos).Moniteur / TRUE)

* *Lorsque la machine répond à une demande d'information, le moniteur*
* *en est informé.*

Informations sur l'état

/

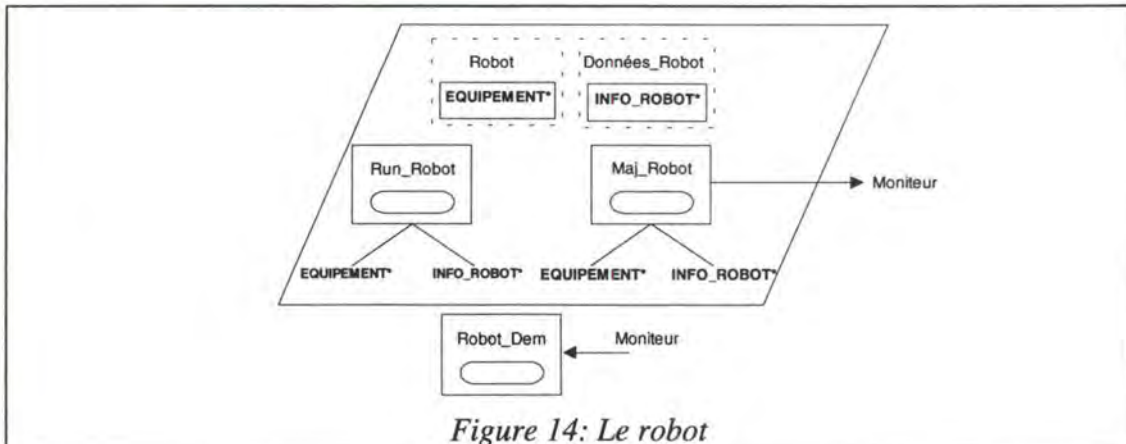
L'agent Machine-Robot

Figure 14: Le robot

- **Contraintes de Base**

Etat initial

/

Composants dérivés

/

- **Contraintes locales**

Comportement de l'état

/

Effets des actions

Run_Robot(a,b): Robot = a
Données_Robot = b

- * *L'action Run met à jour les paramètres de la machine.*

Causalité

Moniteur.Robot_Dem $\hat{0} \leq 1s \rightarrow$ Maj_Robot(equip,infos) with ((equip = Robot)
 $\wedge$ (infos = Données_Robot))

- * *L'action Maj_Robot() a lieu avec les données correspondant*
- * *à celles de la machine concernée.*

Capacité

F (Run_Robot(.,.) / Status(Robot) = FAULT)

- * *L'action run_Robot ne peut avoir lieu si le statut de la*
- * *machine est à fault.*

- **Contraintes de coopération**

Perception des actions

K (Moniteur.Robot_Dem() / TRUE)

- * *La demande d'information venant du moniteur doit être prise*
- * *en compte par la machine.*

Perception de l'état

/

Informations sur les actions

K (Maj_Robot(equip,infos).Moniteur / TRUE)

- * *Lorsque la machine répond à une demande d'information, le moniteur*
- * *en est informé.*

Informations sur l'état

/

L'agent Machine-Tour

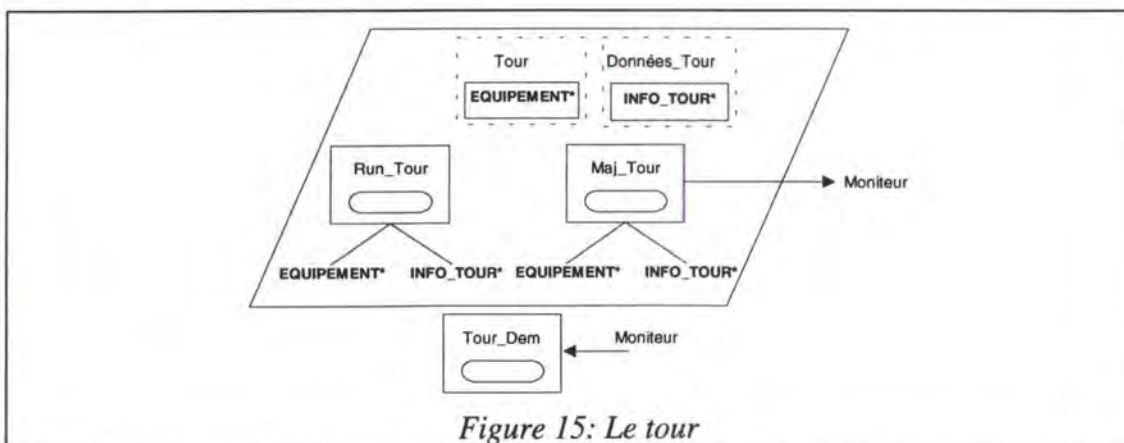


Figure 15: Le tour

- **Contraintes de Base**

Etat initial

/

Composants dérivés

/

• Contraintes locales

Comportement de l'état

/

Effets des actions

```
Run_Tour(a,b):  Tour = a
                Données_Tour = b
```

* *L'action Run met à jour les paramètres de la machine.*

Causalité

$$\text{Moniteur.Tour_Dem} \stackrel{\diamond \leq 1s}{\rightarrow} \text{Maj_Tour}(\text{equip}, \text{infos}) \text{ with } ((\text{equip} = \text{Tour}) \wedge (\text{infos} = \text{Données_Tour}))$$

* L'action Maj_Tour() a lieu avec les données correspondant

* à celles de la machine concernée.

Capacité

$$F(\text{Run_Tour}(_, _) / \text{Status}(\text{Tour}) = \text{FAULT})$$

* *L'action run_Tour ne peut avoir lieu si le statut de la machine*

* *est à fault.*

• Contraintes de coopération

Perception des actions

K (Moniteur.Tour_Dem() / TRUE)

* *La demande d'information venant du moniteur doit être prise*

* *en compte par la machine.*

Perception de l'état

/

Informations sur les actions

K (Maj_Tour(equip,infos).Moniteur / TRUE)

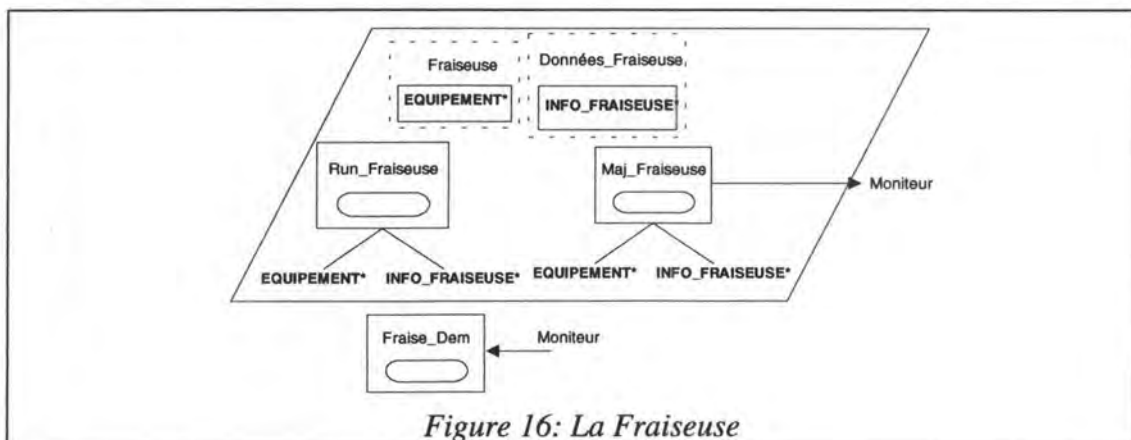
* Lorsque la machine répond à une demande d'information, le moniteur

* *en est informé.*

Informations sur l'état

/

L'agent Machine-Fraiseuse



• Contraintes de Base

Etat initial

1

Composants dérivés

1

• Contraintes locales

Comportement de l'état

/

Effets des actions

Run_Fraiseuse(a,b): Fraiseuse = a

Données_Fraiseuse = b

* *L'action Run met à jour les paramètres de la machine.*

Causalité

Moniteur.Fraise_Dem $\stackrel{\Delta \leq 1s}{\rightarrow}$ Maj_Fraise(equip,infos) with ((equip = Fraiseuse)

$\wedge$ (infos = Données_Fraiseuse))

* *L'action Maj_Fraise() a lieu avec les données correspondant*

* *à celles de la machine concernée.*

Capacité

F (Run_Fraiseuse(_,_) / Status(Fraiseuse) = FAULT)

* *L'action run_Fraiseuse ne peut avoir lieu si le statut de la*

* *machine est à fault.*

• Contraintes de coopération

Perception des actions

K (Moniteur.Fraiseuse_Dem() / TRUE)

* *La demande d'information venant du moniteur doit être prise*

* *en compte par la machine.*

Perception de l'état

/

Informations sur les actions

K (Maj_Fraiseuse(equip,infos).Moniteur / TRUE)

* *Lorsque la machine répond à une demande d'information, le moniteur*

* *en est informé.*

Informations sur l'état

/

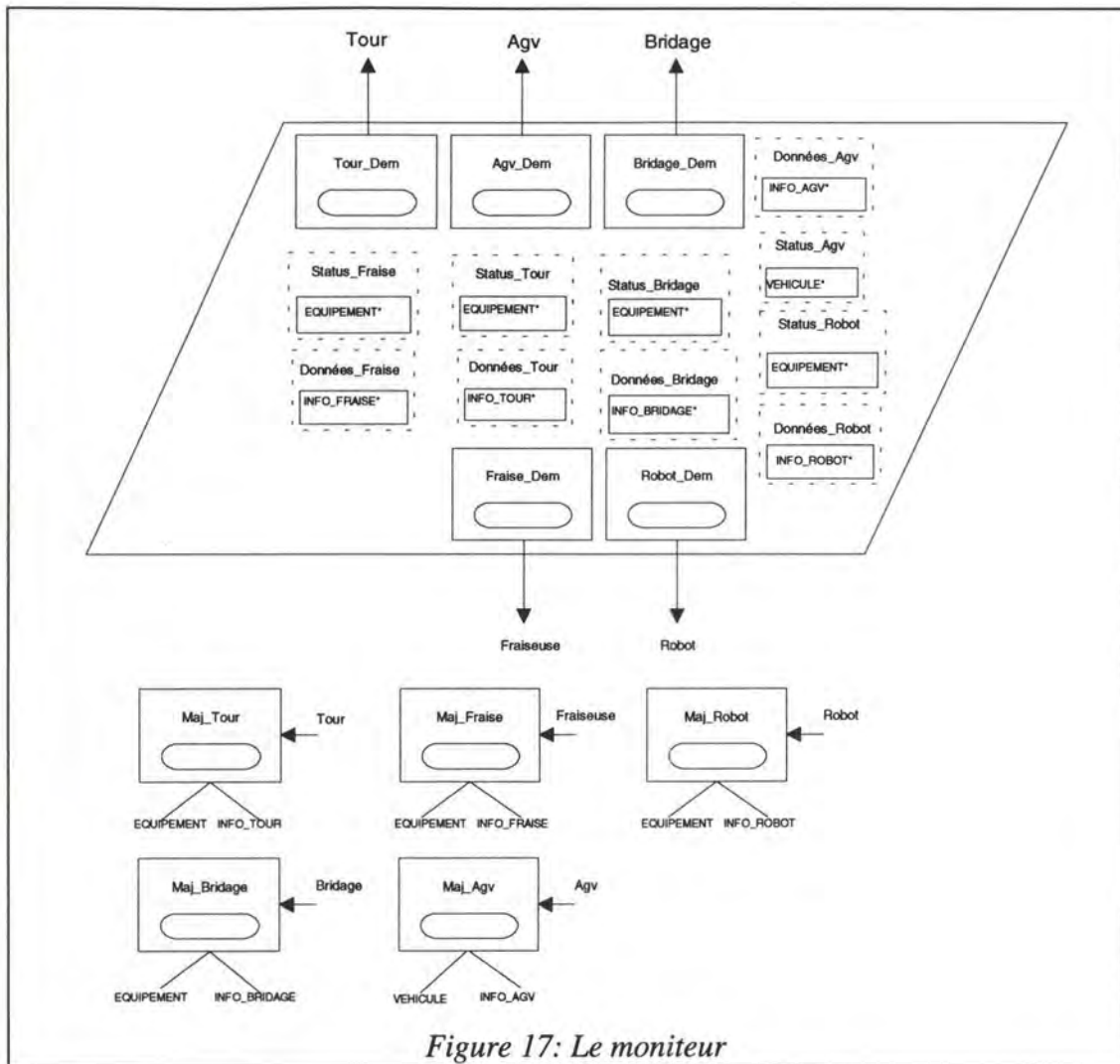
L'agent Moniteur

Figure 17: Le moniteur

• Contraintes de Base

Etat initial

Status_Fraise = UNDEF
 Status_Robot = UNDEF
 Status_Tour = UNDEF
 Status_Agv = UNDEF
 Status_Bridage = UNDEF
 Données_Fraise = UNDEF
 Données_Robot = UNDEF
 Données_Tour = UNDEF
 Données_Agv = UNDEF
 Données_Bridage = UNDEF

- * A l'état initial, les différentes données représentant les machines sont
- * à UNDEF.

Composants dérivés

/

• Contraintes locales

Comportement de l'état

/

Effets des actions

Tour.Maj_Tour(equip,infos) with Status(equip) = IDLE:

Status(Status_Tour) = Status(equip)
 Dock_State(données_Tour) = Dock_State(infos)
 Pallet_Post (données_Tour) = UNDEF
 POSXYZ(données_Tour) = UNDEF
 PGM(données_Tour) = UNDEF

- * *Si le status du tour pour le monitoring est IDLE, alors toutes*
- * *les données autres que le DOCK_STATE et le status sont non*
- * *pertinentes.*

Tour.Maj_Tour(equip, infos) with Status(equip) = FAULT:

Status(Status_Tour) = Status(equip)
 Dock_State(données_Tour) = UNDEF
 Pallet_Post (données_Tour) = UNDEF
 POSXYZ(données_Tour) = UNDEF
 PGM(données_Tour) = UNDEF

- * *Si le status du tour pour le monitoring est FAULT, alors aucune*
- * *donnée concernant le tour n'est valide du point de vue du monitoring.*

Tour.Maj_Tour(equip,infos) with Status(equip) = BUSY:

Status(Status_Tour) = Status(equip)
 Dock_State(données_Tour) = Dock_State(infos)
 Pallet_Post (données_Tour) = Pallet_Post (infos)
 POSXYZ(données_Tour) = POSXYZ(infos)
 PGM(données_Tour) = PGM(infos)

- * *Si le status du tour pour le monitoring est BUSY, alors toutes*
- * *les données sont pertinentes.*

Fraiseuse.Maj_Fraise(equip, infos) with Status(equip) = IDLE:

Status(Status_Fraise) = Status(equip)
 Dock_State(données_Fraise) = Dock_State(infos)
 Pallet_Post (données_Fraise) = UNDEF
 POSXYZ(données_Fraise) = UNDEF
 PGM(données_Fraise) = UNDEF

- * *Si le status de la fraiseuse pour le monitoring est IDLE, alors toutes*
- * *les données autres que le DOCK_STATE et le status sont non*
- * *pertinentes.*

Fraiseuse.Maj_Fraise(equip, infos) with Status(equip) = FAULT:

Status(Status_Fraise) = Status(equip)
 Dock_State(données_Fraise) = UNDEF
 Pallet_Post (données_Fraise) = UNDEF
 POSXYZ(données_Fraise) = UNDEF
 PGM(données_Fraise) = UNDEF

- * *Si le status de la fraiseuse pour le monitoring est FAULT, alors aucune*
- * *donnée concernant le tour n'est valide du point de vue du monitoring.*

Fraiseuse.Maj_Fraise(equip,infos) with Status(equip) = BUSY:

Status(Status_Fraise) = Status(equip)
 Dock_State(données_Fraise) = Dock_State(infos)
 Pallet_Post (données_Fraise) = Pallet_Post (infos)
 POSXYZ(données_Fraise) = POSXYZ(infos)
 PGM(données_Fraise) = PGM(infos)

- * *Si le status de la fraiseuse pour le monitoring est BUSY, alors toutes*
- * *les données sont pertinentes.*

Robot.Maj_Robot(equip,infos) with Status(equip) = FAULT:

Status(Status_Fraise) = Status(equip)

Données_Robot = UNDEF

- * *Si le status du robot est FAULT, alors aucune donnée le concernant*
- * *n'est valide du point de vue du monitoring.*

Robot.Maj_Robot(equip, infos) with Status(equip) ≠ FAULT:

Status(Status_Fraise) = Status(equip)

Données_Robot = infos

- * *Si le status du robot est différent de FAULT alors toutes les données*
- * *sont valides du point de vue du monitoring.*

Agv.Maj_Agv(vehic, infos) with Status(vehic) = FAULT

Status(Status_Agv) = Status(vehic)

Données_Agv = UNDEF

- * *Si le status de l'Agv est FAULT, alors aucune donnée le concernant*
- * *n'est valide du point de vue du monitoring.*

Agv.Maj_Agv(vehic, infos) with Station(infos) ≠ CLAMP

Status(Status_Agv) = Status(vehic)

Données_Agv = infos

- * *Si la station est différente de CLAMP, alors toutes les données*
- * *concernant l'Agv sont valide du point de vue du monitoring.*

Agv.Maj_Agv(vehic, infos) with Station(infos) = CLAMP

Status(Status_Agv) = Status(vehic)

Station(Données_Agv) = Station(infos)

Pallet_Post(Données_Agv) = UNDEF

Dock_State(Données_Agv) = Dock_State(infos)

Distance(Données_Agv) = Distance(infos)

Speed(Données_Agv) = Speed(infos)

Direction(Données_Agv) = Direction(infos)

- * *Si la STATION est CLAMP, alors toutes les données de l'Agv sont*
- * *pertinentes pour le monitoring sauf PALLET_POST car le*
- * *bridage n'a qu'un seul support palette.*

Bridage.Maj_Bridage(equip, infos) with Status(equip) = FAULT:

Status(Status_Bridage) = Status(equip)

Données_Bridage = UNDEF

- * *Si le statut du bridage est à FAULT, alors aucune autre données que le*
- * *statut n'est pertinente.*

Bridage.Maj_Bridage(equip, infos) with Status(equip) ≠ FAULT:

Status(Status_Bridage) = Status(equip)

Données_Bridage = infos

- * *Si le statut du bridage est différent de FAULT, alors toutes les données*
- * *sont pertinentes.*

Causalité

Tour.Maj_Tour $\overset{\diamond}{\leq} 5s \rightarrow$ Tour_Dem

Agv.Maj_Agv $\overset{\diamond}{\leq} 5s \rightarrow$ Agv_Dem

Fraise.Maj_Fraise $\overset{\diamond}{\leq} 5s \rightarrow$ Fraise_Dem

Robot.Maj_Robot $\overset{\diamond}{\leq} 5s \rightarrow$ Robot_Dem

Bridage.Maj_Bridage $\overset{\diamond}{\leq} 5s \rightarrow$ Bridage_Dem

- * *Lorsqu'une action de mise à jour des valeurs d'une machine a lieu,*
- * *dans les 5 secondes qui suivent, le monitoring doit faire une nouvelle*
- * *demande d'information sur la machine dont l'action de mise à jour a eu*

- * *lieu. Les 5 secondes représentent un délais suffisant pour que le*
- * *manager humain ait le temps de voir les valeurs.*

Capacité

O(Fraise_Dem / Status_Fraise = UNDEF)

- * *Si Status_Fraise est à UNDEF, alors une*
- * *action Fraise_Dem est générée(cela signifie qu'on a aucune données*
- * *sur cette machine et qu'il faut donc en demander).*

O(Robot_Dem / Status_Robot = UNDEF)

- * *Si Status_Robot est à UNDEF, alors une*
- * *action Robot_Dem est générée(cela signifie qu'on a aucune données*
- * *sur cette machine et qu'il faut donc en demander).*

O(Tour_Dem / Status_Tour = UNDEF)

- * *Si Status_Tour est à UNDEF, alors une*
- * *action Tour_Dem est générée(cela signifie qu'on a aucune données*
- * *sur cette machine et qu'il faut donc en demander).*

O(Agv_Dem / Status_Agv = UNDEF)

- * *Si Status_Agv est à UNDEF, alors une*
- * *action Agv_Dem est générée(cela signifie qu'on a aucune données*
- * *sur cette machine et qu'il faut donc en demander).*

O(Bridage_Dem / Status_Bridage = UNDEF)

- * *Si Status_Bridage est à UNDEF, alors une*
- * *action Bridage_Dem est générée(cela signifie qu'on a aucune données*
- * *sur cette machine et qu'il faut donc en demander).*

• **Contraintes de coopération**

Perception des actions

K (Tour.Maj_Tour(equip, infos) / TRUE)

K (Fraiseuse.Maj_Fraise(equip, infos) / TRUE)

K (Bridage.Maj_Bridage(equip, infos) / TRUE)

K (Robot.Maj_Robot(equip, infos) / TRUE)

K (Agv.Maj_Agv(vehicule, infos) / TRUE)

- * *Lorsqu'une réponse, à une demande d'information du moniteur, arrive*
- * *d'une machine, elle doit toujours être prise en compte.*

Perception de l'état

/

Informations sur les actions

K (Tour_Dem().Tour / TRUE)

K (Fraise_Dem().Fraiseuse / TRUE)

K (Robot_Dem().Robot / TRUE)

K (Agv_Dem().Agv / TRUE)

K (Bridage_Dem().Bridage / TRUE)

- * *Lorsqu'une action de demande d'info est générée par le moniteur, elle*
- * *est rendue visible à la machine concernée.*

Informations sur l'état

K(Manager.Status_Fraise / TRUE)

K(Manager.Status_Agv / TRUE)

K(Manager.Status_Bridage / TRUE)

K(Manager.Status_Tour / TRUE)

K(Manager.Status_Robot / TRUE)

K(Manager.Données_Fraise / TRUE)
K(Manager.Données_Agv / TRUE)
K(Manager.Données_Bridage / TRUE)
K(Manager.Données_Tour / TRUE)
K(Manager.Données_Robot / TRUE)

*** Le moniteur laisse toujours voir toutes ses données au manager**

4) Version 4 : Les Problèmes de cohérence

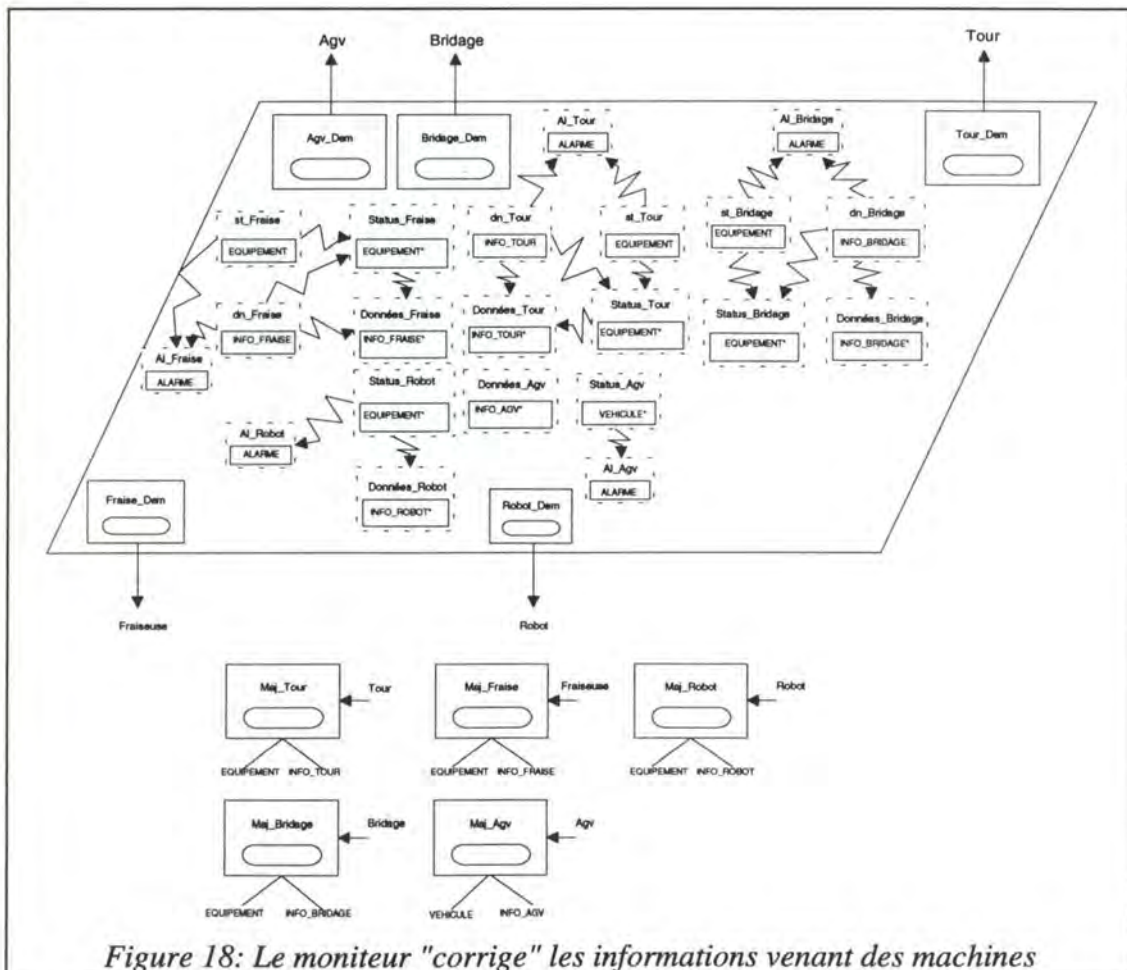
4.1) Les machines envoient des informations incohérentes

a) Les nouveaux¹ types de données utilisées

ALARME: Boolean

b) Définition des agents

L'agent Moniteur



¹pour la définition des autres types, se reporter au point "a) Les types de données utilisées" du chapitre "3.1 Version 1: Le Moniteur voit tout - Les Machines montrent tout"

• **Contraintes de Base**

Etat initial

Status_Fraise = UNDEF
 Status_Robot = UNDEF
 Status_Tour = UNDEF
 Status_Agv = UNDEF
 Status_Bridage = UNDEF
 Données_Fraise = UNDEF
 Données_Robot = UNDEF
 Données_Tour = UNDEF
 Données_Agv = UNDEF
 Données_Bridage = UNDEF

- * *A l'état initial, les différentes données représentant les machines sont*
- * *à UNDEF.*

Composants dérivés

Status_Fraise = IDLE $\equiv$ (st_Fraise = IDLE) $\wedge$ (Dock_State(dn_Fraise) = STOPPED)

- * *Si le status de la machine est IDLE et le Dock_State est STOPPED,*
- * *alors le status IDLE est valide pour le monitoring.*

Status_Fraise = FAULT $\equiv$ (Dock_State(dn_Fraise) = UNKNOWN)

$\vee ((\text{st_Fraise} = \text{IDLE}) \wedge (\text{Dock_State}(\text{dn_Fraise}) \neq \text{STOPPED}))$
 $\vee ((\text{st_Fraise} = \text{BUSY}) \wedge ((\text{Dock_State}(\text{dn_Fraise}) = \text{STOPPED})$
 $\vee (\text{Dock_State}(\text{dn_Fraise}) = \text{UNKNOWN}))$

- * *Si le Dock_State de la machine est UNKNOWN, ou que le statut de la*
- * *machine est IDLE avec un Dock_State différent de STOPPED, ou que le*
- * *statut de la machine est BUSY avec un Doc_State à STOPPED ou*
- * *UNKNOWN, alors le seul status valable pour le monitoring est FAULT*

Status_Fraise = BUSY $\equiv$ (st_Fraise = BUSY)

$\wedge (\text{Dock_State}(\text{dn_Fraise}) \neq \text{STOPPED})$
 $\wedge (\text{Dock_State}(\text{dn_Fraise}) \neq \text{UNKNOWN})$

- * *Si le Dock_State de la machine est différent de STOPPED et de*
- * *UNKNOWN et le status est BUSY alors il est correct et le status pour*
- * *le monitoring est aussi BUSY.*

Dock_State(Données_Fraise) = STOPPED $\equiv$ (Status_Fraise = IDLE)

- * *Si le status de la fraiseuse pour le monitoring est IDLE, alors le*
- * *Dock_State est STOPPED.*

PGM(Données_Fraise) = UNDEF $\equiv$ (Status_Fraise = IDLE)

Pallet_Post(Données_Fraise) = UNDEF $\equiv$ (Status_Fraise = IDLE)

POSXYZ(Données_Fraise) = UNDEF $\equiv$ (Status_Fraise = IDLE)

- * *Si le status de la fraiseuse pour le monitoring est IDLE, alors toutes*
- * *les données autres que le Dock_State et le status sont non*
- * *pertinentes.*

Dock_State(Données_Fraise) = Dock_State(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)

PGM(Données_Fraise) = PGM(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)

Pallet_Post(Données_Fraise) = Pallet_Post(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)

POSXYZ(Données_Fraise) = POSXYZ(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)

- * *Si le status de la fraiseuse pour le monitoring est BUSY, alors toutes*
- * *les données sont pertinentes.*

Données_Fraise = UNDEF $\equiv$ (Status_Fraise = FAULT)

- * *Si le status de la fraiseuse pour le monitoring est FAULT, alors aucune*
- * *donnée concernant le tour n'est valide du point de vue du monitoring.*

Status_Tour = IDLE $\equiv$ (st_Tour = IDLE) $\wedge$ (Dock_State(dn_Tour) = STOPPED)

- * *Si le status de la machine est IDLE et le Dock_State est STOPPED,*
- * *alors le status IDLE est valide pour le monitoring.*

Status_Tour = FAULT $\equiv$ (Dock_State(dn_Tour) = UNKNOWN)

$\vee ((\text{st_Fraise} = \text{IDLE}) \wedge (\text{Dock_State}(\text{dn_Tour}) \neq \text{STOPPED}))$
 $\vee ((\text{st_Fraise} = \text{BUSY}) \wedge ((\text{Dock_State}(\text{dn_Tour}) = \text{STOPPED})$
 $\vee (\text{Dock_State}(\text{dn_Tour}) = \text{UNKNOWN}))$

- * *Si le Dock_State de la machine est UNKNOWN, ou que le statut de la*
- * *machine est IDLE avec un Dock_State différent de STOPPED, ou que le*
- * *statut de la machine est BUSY avec un Doc_State à STOPPED ou*
- * *UNKNOWN, alors le seul status valable pour le monitoring est FAULT*

Status_Tour = BUSY $\equiv$ (st_Tour = BUSY)

$\wedge (\text{Dock_State}(\text{dn_Tour}) \neq \text{STOPPED})$
 $\wedge (\text{Dock_State}(\text{dn_Tour}) \neq \text{UNKNOWN})$

- * *Si le Dock_State de la machine est différent de STOPPED et de*
- * *UNKNOWN et le status est BUSY alors il est correct et le status pour*
- * *le monitoring est aussi BUSY.*

Dock_State(Données_Tour) = STOPPED $\equiv$ (Status_Tour = IDLE)

- * *Si le status du tour pour le monitoring est IDLE, alors le*
- * *Dock_State est STOPPED.*

PGM(Données_Tour) = UNDEF $\equiv$ (Status_Tour = IDLE)

Pallet_Post(Données_Tour) = UNDEF $\equiv$ (Status_Tour = IDLE)

POSXYZ(Données_Tour) = UNDEF $\equiv$ (Status_Tour = IDLE)

- * *Si le status du tour pour le monitoring est IDLE, alors toutes*
- * *les données autres que le Dock_State et le status sont non*
- * *pertinentes.*

Dock_State(Données_Tour) = Dock_State(dn_Tour) $\equiv$ (Status_Tour = BUSY)

PGM(Données_Tour) = PGM(dn_Tour) $\equiv$ (Status_Tour = BUSY)

Pallet_Post(Données_Tour) = Pallet_Post(dn_Tour) $\equiv$ (Status_Tour = BUSY)

POSXYZ(Données_Tour) = POSXYZ(dn_Tour) $\equiv$ (Status_Tour = BUSY)

- * *Si le status du tour pour le monitoring est BUSY, alors toutes*
- * *les données sont pertinentes.*

Données_Tour = UNDEF $\equiv$ (Status_Tour = FAULT)

- * *Si le status du tour pour le monitoring est FAULT, alors aucune donnée*
- * *concernant le tour n'est valide du point de vue du monitoring.*

Status_Bridage = BUSY $\equiv$ (vitesse(dn_Bridage) $\neq$ 0)

- * *Si la vitesse de bridage est différente de 0, alors le status doit être*
- * *BUSY pour le monitoring.*

Status_Bridage = st_Bridage $\equiv$ (vitesse(dn_Bridage) = 0)

- * *Si la vitesse de bridage vaut 0, alors le status du bridage correspond*
- * *à ce qu'il doit être pour le monitoring.*

Données_Bridage = dn_Bridage $\equiv$ (Status_Bridage $\neq$ FAULT)

Données_Bridage = UNDEF $\equiv$ (Status_Bridage = FAULT)

- * *Les autres données que le statut ne sont pertinentes que si le statut*
- * *n'est pas à FAULT.*

AI_Fraise = TRUE $\equiv$ ((st_Fraise = FAULT))

$\vee((\text{st_Fraise} = \text{IDLE}) \wedge (\text{Dock_State}(\text{dn_Fraise}) \neq \text{STOPPED}))$
 $\vee((\text{st_Fraise} \neq \text{FAULT}) \wedge (\text{Dock_State}(\text{dn_Fraise}) = \text{UNKNOWN}))$
 $\vee((\text{st_Fraise} \neq \text{BUSY}) \wedge ((\text{Dock_State}(\text{dn_Fraise}) = \text{UNKNOWN})$
 $\vee(\text{Dock_State}(\text{dn_Fraise}) = \text{STOPPED})))$

- * *L'alarme de la fraiseuse est à TRUE (on a des données incohérentes ou*
- * *FAULT)*
- * *si soit - le statut est IDLE et le Dock_State est différent de STOPPED*
- * *- le statut est différent de FAULT bien que le Dock_State soit*
- * *UNKNOWN*
- * *- le statut n'est pas BUSY bien que le Dock_State doit STOPPED*
- * *ou UNKNOWN*

AI_Fraise = FALSE $\equiv$ NOT (((st_Fraise = FAULT)

$\vee((\text{st_Fraise} = \text{IDLE}) \wedge (\text{Dock_State}(\text{dn_Fraise}) \neq \text{STOPPED}))$
 $\vee((\text{st_Fraise} \neq \text{FAULT}) \wedge (\text{Dock_State}(\text{dn_Fraise}) = \text{UNKNOWN}))$
 $\vee((\text{st_Fraise} \neq \text{BUSY}) \wedge ((\text{Dock_State}(\text{dn_Fraise}) = \text{UNKNOWN})$
 $\vee(\text{Dock_State}(\text{dn_Fraise}) = \text{STOPPED})))$

- * *L'alarme de la fraiseuse est à FALSE (on a des données cohérentes)*
- * *dans tous les autres cas.*

AI_Tour = TRUE $\equiv$ ((st_Tour = FAULT))

$\vee((\text{st_Tour} = \text{IDLE}) \wedge (\text{Dock_State}(\text{dn_Tour}) \neq \text{STOPPED}))$
 $\vee((\text{st_Fraise} \neq \text{FAULT}) \wedge (\text{Dock_State}(\text{dn_Tour}) = \text{UNKNOWN}))$
 $\vee((\text{st_Fraise} \neq \text{BUSY}) \wedge ((\text{Dock_State}(\text{dn_Tour}) = \text{UNKNOWN})$
 $\vee(\text{Dock_State}(\text{dn_Tour}) = \text{STOPPED})))$

- * *L'alarme du tour est à TRUE (on a des données incohérentes ou FAULT)*
- * *si soit - le statut est à FAULT*
- * *- le statut est IDLE et le Dock_State est différent de STOPPED*
- * *- le statut est différent de FAULT bien que le Dock_State soit*
- * *UNKNOWN*
- * *- le statut n'est pas BUSY bien que le Dock_State doit STOPPED*
- * *ou UNKNOWN*

AI_Tour = TRUE $\equiv$ NOT((st_Tour = FAULT)

$\vee((\text{st_Tour} = \text{IDLE}) \wedge (\text{Dock_State}(\text{dn_Tour}) \neq \text{STOPPED}))$
 $\vee((\text{st_Fraise} \neq \text{FAULT}) \wedge (\text{Dock_State}(\text{dn_Tour}) = \text{UNKNOWN}))$
 $\vee((\text{st_Fraise} \neq \text{BUSY}) \wedge ((\text{Dock_State}(\text{dn_Tour}) = \text{UNKNOWN})$
 $\vee(\text{Dock_State}(\text{dn_Tour}) = \text{STOPPED})))$

- * *L'alarme du tour est à FALSE (on a des données cohérentes) dans tous*
- * *les autres cas.*

AI_Robot = TRUE $\equiv$ ((Status_Robot = FAULT))

AI_Robot = FALSE $\equiv$ NOT((Status_Robot = FAULT))

AI_Agv = TRUE $\equiv$ ((Status_Agv = FAULT))

AI_Agv = FALSE $\equiv$ NOT((Status_Agv = FAULT))

$AI_Bridage = TRUE \equiv ((st_Bridage = FAULT) \vee ((st_Bridage \neq BUSY) \wedge ((vitesse(dn_Bridage) \neq 0)))$
 * *L'alarme du bridage est à TRUE (on a des données incohérentes ou*
 * *FAULT) si soit - le statut est FAULT*
 * *- la vitesse de bridage est différente de 0 et le statut n'est*
 * *pas BUSY.*

$AI_Bridage = FAULT \equiv NOT((st_Bridage = FAULT) \vee ((st_Bridage \neq BUSY) \wedge ((vitesse(dn_Bridage) \neq 0))))$
 * *L'alarme du bridage est à FALSE (on a des données cohérentes)*

• **Contraintes locales**

Comportement de l'état

/

Effets des actions

Tour.Maj_Tour(equip,infos): st_Tour = equip
 dn_Tour = infos
 * *Lorsque l'action Maj_Tour en provenance du tour à lieu, les anciennes*
 * *valeurs du monitoring sont remplacées par les nouvelles contenues*
 * *dans l'action.*

Fraiseuse.Maj_Fraise(equip, infos): st_Fraise = equip
 dn_Fraise = infos
 * *Lorsque l'action Maj_Fraise en provenance de la fraiseuse à lieu, les*
 * *anciennes valeurs du monitoring sont remplacées par les nouvelles*
 * *contenues dans l'action.*

Robot.Maj_Robot(equip,infos) with Status(equip) = FAULT:
 Status(Status_Fraise) = Status(equip)
 Données_Robot = UNDEF
 * *Si le status du robot est FAULT, alors aucune donnée le concernant*
 * *n'est valide du point de vue du monitoring.*

Robot.Maj_Robot(equip,infos) with Status(equip) $\neq$ FAULT:
 Status(Status_Fraise) = Status(equip)
 Données_Robot = infos
 * *Si le status du robot est différent de FAULT alors toutes les données*
 * *sont valides du point de vue du monitoring.*

Agv.Maj_Agv(vehic, infos) with Status = FAULT
 Status(Status_Agv) = Status(vehic)
 Données_Agv = UNDEF
 * *Si le status de l'Agv est FAULT, alors aucune donnée le concernant*
 * *n'est valide du point de vue du monitoring.*

Agv.Maj_Agv(vehic, infos) with Station(infos) $\neq$ CLAMP
 Status(Status_Agv) = Status(vehic)
 Données_Agv = infos
 * *Si la station est différente de CLAMP, alors toutes les données*
 * *concernant l'Agv sont valide du point de vue du monitoring.*

Agv.Maj_Agv(vehic, infos) with Station(infos) = CLAMP

Status(Status_Agv) = Status(vehic)
 Station(Données_Agv) = Station(infos)
 Pallet_Post(Données_Agv) = UNDEF
 Dock_State(Données_Agv) = Dock_State(infos)
 Distance(Données_Agv) = Distance(infos)
 Speed(Données_Agv) = Speed(infos)
 Direction(Données_Agv) = Direction(infos)

- * *Si la STATION est CLAMP, alors toutes les données de l'Agv sont*
- * *pertinentes pour le monitoring sauf PALLET_POST car le*
- * *bridage n'a qu'un seul support palette.*

Bridge.Maj_Bridge(equip, infos): st_Bridge = equip
 dn_Bridge = infos

- * *Lorsque l'action Maj_Bridge en provenance du bridage à lieu, les*
- * *anciennes valeurs du monitoring sont remplacées par les nouvelles*
- * *contenues dans l'action.*

Causalité

Tour.Maj_Tour $\hat{\diamond} \leq 5s \rightarrow$ Tour_Dem

Agv.Maj_Agv $\hat{\diamond} \leq 5s \rightarrow$ Agv_Dem

Fraise.Maj_Fraise $\hat{\diamond} \leq 5s \rightarrow$ Fraise_Dem

Robot.Maj_Robot $\hat{\diamond} \leq 5s \rightarrow$ Robot_Dem

Bridge.Maj_Bridge $\hat{\diamond} \leq 5s \rightarrow$ Bridge_Dem

- * *Lorsqu'une action de mise à jour des valeurs d'une machine a lieu,*
- * *dans les 5 secondes qui suivent, le monitoring doit faire une nouvelle*
- * *demande d'information sur la machine dont l'action de mise à jour a eu*
- * *lieu.*

Capacité

O(Fraise_Dem / Status_Fraise = UNDEF)

- * *Si Status_Fraise est à UNDEF, alors une*
- * *action Fraise_Dem est générée.(cela signifie qu'on a aucune données*
- * *sur cette machine et qu'il faut donc en demander)*

O(Robot_Dem / Status_Robot = UNDEF)

- * *Si Status_Robot est à UNDEF, alors une*
- * *action Robot_Dem est générée.(cela signifie qu'on a aucune données*
- * *sur cette machine et qu'il faut donc en demander)*

O(Tour_Dem / Status_Tour = UNDEF)

- * *Si Status_Tour est à UNDEF, alors une*
- * *action Tour_Dem est générée.(cela signifie qu'on a aucune données*
- * *sur cette machine et qu'il faut donc en demander)*

O(Agv_Dem / Status_Agv = UNDEF)

- * *Si Status_Agv est à UNDEF, alors une*
- * *action Agv_Dem est générée.(cela signifie qu'on a aucune données*
- * *sur cette machine et qu'il faut donc en demander)*

O(Bridge_Dem / Status_Bridge = UNDEF)

- * *Si Status_Bridge est à UNDEF, alors une*
- * *action Bridge_Dem est générée.(cela signifie qu'on a aucune données*
- * *sur cette machine et qu'il faut donc en demander)*

• **Contraintes de coopération**

Perception des actions

K (Tour.Maj_Tour(equip, infos) / TRUE)
 K (Fraiseuse.Maj_Fraise(equip, infos) / TRUE)
 K (Bridage.Maj_Bridage(equip, infos) / TRUE)
 K (Robot.Maj_Robot(equip, infos) / TRUE)
 K (Agv.Maj_Agv(vehicule, infos) / TRUE)

- * *Lorsqu'une réponse, à une demande d'information du moniteur, arrive*
- * *d'une machine, elle doit toujours être prise en compte.*

Perception de l'état

/

Informations sur les actions

K (Tour_Dem().Tour / TRUE)
 K (Fraise_Dem().Fraiseuse / TRUE)
 K (Robot_Dem().Robot / TRUE)
 K (Agv_Dem().Agv / TRUE)
 K (Bridage_Dem().Bridage / TRUE)

- * *Lorsqu'une action de demande d'info est générée par le moniteur, elle*
- * *est rendue visible à la machine concernée.*

Informations sur l'état

K(Manager.Status_Fraise / TRUE)
 K(Manager.Status_Agv / TRUE)
 K(Manager.Status_Bridage / TRUE)
 K(Manager.Status_Tour / TRUE)
 K(Manager.Status_Robot / TRUE)
 K(Manager.Données_Fraise / TRUE)
 K(Manager.Données_Agv / TRUE)
 K(Manager.Données_Bridage / TRUE)
 K(Manager.Données_Tour / TRUE)
 K(Manager.Données_Robot / TRUE)

- * *Le moniteur laisse toujours voir toutes ses données au manager*

4.2) Les machines n'envoient plus d'informations

L'agent Machine Agv

• **Contraintes de Base**

Etat initial

/

Composants dérivés

/

• **Contraintes locales**

Comportement de l'état

/

Effets des actions

Run_Agv(a,b): Agv = a
 Données_Agv = b

- * *L'action Run met à jour les paramètres de la machine.*

Causalité

Moniteur.Agv_Dem $\hat{0} \leq 1s \rightarrow$ Maj_Agv(vehicule,infos) with vehicule = Agv
 infos = Données_Agv

$\oplus DAC^2$

²DAC signifie Dummy Action. Il s'agit d'une action ne réalisant rien.

Capacité

F (Run_AGV(,_) / Status(Agv) = FAULT)

- * *L'action run_Agv ne peut avoir lieu si le statut de la machine*
- * *est à FAULT.*

• **Contraintes de coopération**

Perception des actions

/

Perception de l'état

/

Informations sur les actions

/

Informations sur l'état

/

L'agent Machine-Bridge

• **Contraintes de Base**

Etat initial

/

Composants dérivés

/

• **Contraintes locales**

Comportement de l'état

/

Effets des actions

Run_Bridge(a,b): Bridge = a

Données_Bridge = b

- * *L'action Run met à jour les paramètres de la machine.*

Causalité

Moniteur.Bridge_Dem $\overset{\Delta}{\leq} 1s \rightarrow$ Maj_Bridge(equip,infos) with equip = a
infos = Données_Bridge

$\oplus$ DAC

Capacité

F (Run_Bridge(,_) / Status(Bridge) = FAULT)

- * *L'action run_Bridge ne peut avoir lieu si le statut de la machine*
- * *est à fault.*

• **Contraintes de coopération**

Perception des actions

/

Perception de l'état

/

Informations sur les actions

/

Informations sur l'état

/

L'agent Machine-Robot

• **Contraintes de Base**

Etat initial

/

Composants dérivés

/

- Contraintes locales

- Comportement de l'état

- /

- Effets des actions

Run_Robot(a,b): Robot = a
Données_Robot = b

* *L'action Run met à jour les paramètres de la machine.*

- Causalité

Moniteur.Robot_Dem $\overset{\Delta}{\leq} 1s \rightarrow$ Maj_Robot(equip,infos) with equip = a
infos = Données_Robot

$\oplus$ DAC

- Capacité

F (Run_Robot(.,.) / Status(Robot) = FAULT)

* *L'action run_Robot ne peut avoir lieu si le statut de la machine
est à fault.*

- Contraintes de coopération

- Perception des actions

- /

- Perception de l'état

- /

- Informations sur les actions

- /

- Informations sur l'état

- /

L'agent Machine-Tour

- Contraintes de Base

- Etat initial

- /

- Composants dérivés

- /

- Contraintes locales

- Comportement de l'état

- /

- Effets des actions

Run_Tour(a,b): Tour = a
Données_Tour = b

* *L'action Run met à jour les paramètres de la machine.*

- Causalité

Moniteur.Tour_Dem $\overset{\Delta}{\leq} 1s \rightarrow$ Maj_Tour(equip,infos) with equip = a
infos = Données_Tour

$\oplus$ DAC

- Capacité

F (Run_Tour(.,.) / Status(Tour) = FAULT)

* *L'action run_Tour ne peut avoir lieu si le statut de la machine
est à fault.*

- Contraintes de coopération

- Perception des actions

- /

Perception de l'état

/

Informations sur les actions

/

Informations sur l'état

/

L'agent Machine-Fraiseuse

• **Contraintes de Base**

Etat initial

/

Composants dérivés

/

• **Contraintes locales**

Comportement de l'état

/

Effets des actions

Run_Fraiseuse(a,b): Fraiseuse = a

Données_Fraiseuse = b

* *L'action Run met à jour les paramètres de la machine.*

Causalité

Moniteur.Fraise_Dem $\hat{\leq}$ 1s $\rightarrow$ Maj_Fraise(equip,infos) with equip = a

infos = Données_Tour

$\oplus$ DAC

Capacité

F (Run_Fraiseuse(.,.) / Status(Fraiseuse) = FAULT)

* *L'action run_Fraiseuse ne peut avoir lieu si le statut de la*

* *machine est à fault.*

• **Contraintes de coopération**

Perception des actions

/

Perception de l'état

/

Informations sur les actions

/

Informations sur l'état

/

L'agent Moniteur

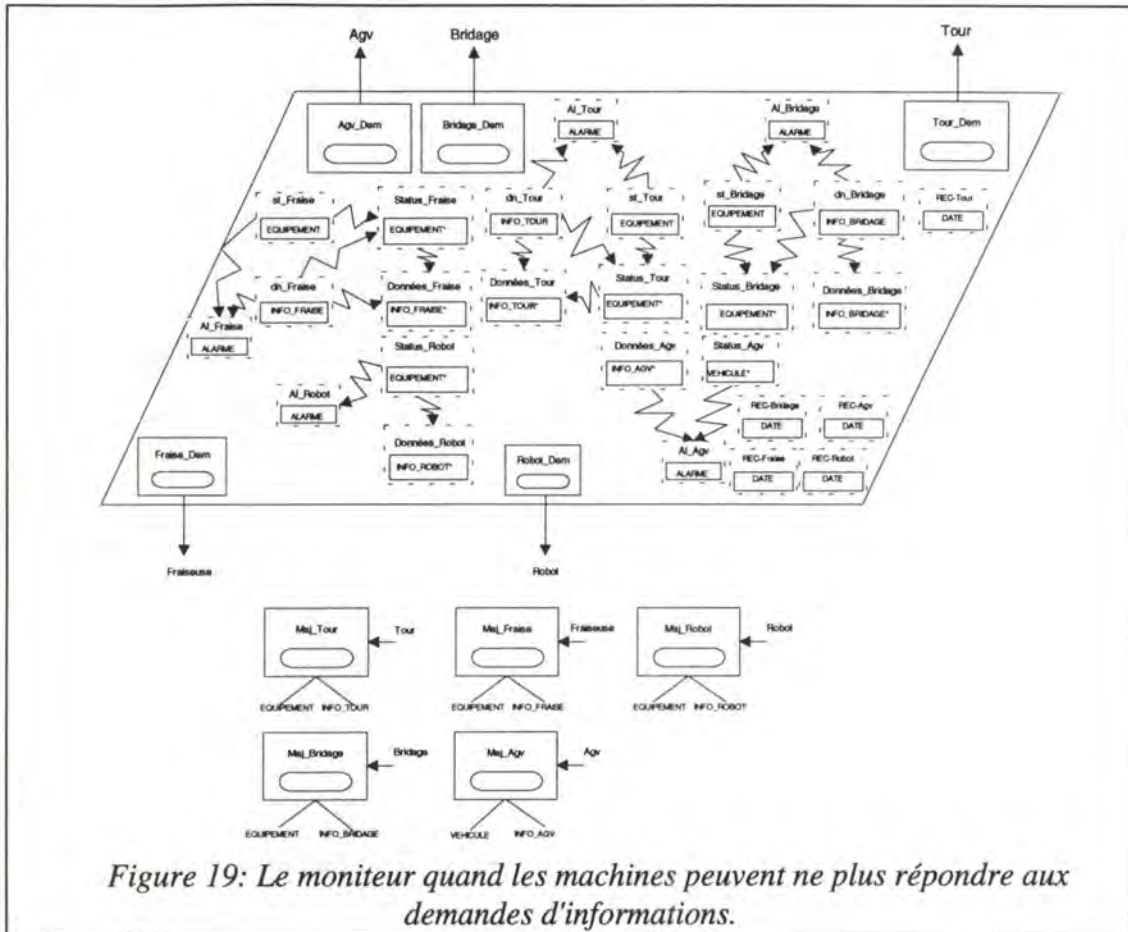


Figure 19: Le moniteur quand les machines peuvent ne plus répondre aux demandes d'informations.

• Contraintes de Base

Etat initial

Status_Fraise = UNDEF
 Status_Robot = UNDEF
 Status_Tour = UNDEF
 Status_Agv = UNDEF
 Status_Bridge = UNDEF
 Données_Fraise = UNDEF
 Données_Robot = UNDEF
 Données_Tour = UNDEF
 Données_Agv = UNDEF
 Données_Bridge = UNDEF

- * A l'état initial, les différentes données représentant les machines sont
- * à UNDEF.

Composants dérivés

Status_Fraise = IDLE $\equiv$ (st_Fraise = IDLE) $\wedge$ (Dock_State(dn_Fraise) = STOPPED)

- * Si le status de la machine est IDLE et le Dock_State est STOPPED,
- * alors le status IDLE est valide pour le monitoring.

Status_Fraise = FAULT $\equiv$ (Dock_State(dn_Fraise) = UNKNOWN)

$\vee ((st_Fraise = IDLE) \wedge (Dock_State(dn_Fraise) \neq STOPPED))$
 $\vee ((st_Fraise = BUSY) \wedge ((Dock_State(dn_Fraise) = STOPPED)$
 $\vee (Dock_State(dn_Fraise) = UNKNOWN))$

- * Si le Dock_State de la machine est UNKNOWN, ou que le statut de la

- * *machine est IDLE avec un Dock_State différent de STOPPED, ou que le*
- * *statut de la machine est BUSY avec un Doc_State à STOPPED ou*
- * *UNKNOWN, alors le seul status valable pour le monitoring est FAULT*

Status_Fraise = BUSY $\equiv$ (st_Fraise = BUSY)
 $\wedge$ (Dock_State(dn_Fraise) $\neq$ STOPPED)
 $\wedge$ (Dock_State(dn_Fraise) $\neq$ UNKNOWN)

- * *Si le Dock_State de la machine est différent de STOPPED et de*
- * *UNKNOWN et le status est BUSY alors il est correct et le status pour*
- * *le monitoring est aussi BUSY.*

Dock_State(Données_Fraise) = STOPPED $\equiv$ (Status_Fraise = IDLE)

- * *Si le status de la fraiseuse pour le monitoring est IDLE, alors le*
- * *Dock_State est STOPPED.*

PGM(Données_Fraise) = UNDEF $\equiv$ (Status_Fraise = IDLE)

Pallet_Post(Données_Fraise) = UNDEF $\equiv$ (Status_Fraise = IDLE)

POSXYZ(Données_Fraise) = UNDEF $\equiv$ (Status_Fraise = IDLE)

- * *Si le status de la fraiseuse pour le monitoring est IDLE, alors toutes*
- * *les données autres que le Dock_State et le status sont non*
- * *pertinentes.*

Dock_State(Données_Fraise) = Dock_State(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)

PGM(Données_Fraise) = PGM(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)

Pallet_Post(Données_Fraise) = Pallet_Post(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)

POSXYZ(Données_Fraise) = POSXYZ(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)

- * *Si le status de la fraiseuse pour le monitoring est BUSY, alors toutes*
- * *les données sont pertinentes.*

Données_Fraise = UNDEF $\equiv$ (Status_Fraise = FAULT)

- * *Si le status de la fraiseuse pour le monitoring est FAULT, alors aucune*
- * *donnée concernant le tour n'est valide du point de vue du monitoring.*

Status_Tour = IDLE $\equiv$ (st_Tour = IDLE) $\wedge$ (Dock_State(dn_Tour) = STOPPED)

- * *Si le status de la machine est IDLE et le Dock_State est STOPPED,*
- * *alors le status IDLE est valide pour le monitoring.*

Status_Tour = FAULT $\equiv$ (Dock_State(dn_Tour) = UNKNOWN)

$\vee$ ((st_Fraise = IDLE) $\wedge$ (Dock_State(dn_Tour) $\neq$ STOPPED))

$\vee$ ((st_Fraise = BUSY) $\wedge$ ((Dock_State(dn_Tour) = STOPPED)

$\vee$ (Dock_State(dn_Tour) = UNKNOWN))

- * *Si le Dock_State de la machine est UNKNOWN, ou que le statut de la*
- * *machine est IDLE avec un Dock_State différent de STOPPED, ou que le*
- * *statut de la machine est BUSY avec un Doc_State à STOPPED ou*
- * *UNKNOWN, alors le seul status valable pour le monitoring est FAULT*

Status_Tour = BUSY $\equiv$ (st_Tour = BUSY)

$\wedge$ (Dock_State(dn_Tour) $\neq$ STOPPED)

$\wedge$ (Dock_State(dn_Tour) $\neq$ UNKNOWN)

- * *Si le Dock_State de la machine est différent de STOPPED et de*
- * *UNKNOWN et le status est BUSY alors il est correct et le status pour*
- * *le monitoring est aussi BUSY.*

Dock_State(Données_Tour) = STOPPED $\equiv$ (Status_Tour = IDLE)

- * *Si le status du tour pour le monitoring est IDLE, alors le*
- * *Dock_State est STOPPED.*

PGM(Données_Tour) = UNDEF $\equiv$ (Status_Tour = IDLE)
Pallet_Post(Données_Tour) = UNDEF $\equiv$ (Status_Tour = IDLE)
POSXYZ(Données_Tour) = UNDEF $\equiv$ (Status_Tour = IDLE)

- * *Si le status du tour pour le monitoring est IDLE, alors toutes*
- * *les données autres que le Dock_State et le status sont non*
- * *pertinentes.*

Dock_State(Données_Tour) = Dock_State(dn_Tour) $\equiv$ (Status_Tour = BUSY)
PGM(Données_Tour) = PGM(dn_Tour) $\equiv$ (Status_Tour = BUSY)
Pallet_Post(Données_Tour) = Pallet_Post(dn_Tour) $\equiv$ (Status_Tour = BUSY)
POSXYZ(Données_Tour) = POSXYZ(dn_Tour) $\equiv$ (Status_Tour = BUSY)

- * *Si le status du tour pour le monitoring est BUSY, alors toutes*
- * *les données sont pertinentes.*

Données_Tour = UNDEF $\equiv$ (Status_Tour = FAULT)

- * *Si le status du tour pour le monitoring est FAULT, alors aucune donnée*
- * *concernant le tour n'est valide du point de vue du monitoring.*

Status_Bridage = BUSY $\equiv$ (vitesse(dn_Bridage) $\neq$ 0)

- * *Si la vitesse de bridage est différente de 0, alors le status doit être*
- * *BUSY pour le monitoring.*

Status_Bridage = st_Bridage $\equiv$ (vitesse(dn_Bridage) = 0)

- * *Si la vitesse de bridage vaut 0, alors le status du bridage correspond*
- * *à ce qu'il doit être pour le monitoring.*

Données_Bridage = dn_Bridage $\equiv$ (Status_Bridage $\neq$ FAULT)
Données_Bridage = UNDEF $\equiv$ (Status_Bridage = FAULT)

- * *Les autres données que le statut ne sont pertinentes que si le statut*
- * *n'est pas à FAULT.*

Al_Fraise = TRUE $\equiv$ ((st_Fraise = FAULT))
 $\vee$ ((st_Fraise = IDLE) $\wedge$ (Dock_State(dn_Fraise) $\neq$ STOPPED))
 $\vee$ ((st_Fraise $\neq$ FAULT) $\wedge$ (Dock_State(dn_Fraise) = UNKNOWN))
 $\vee$ ((st_Fraise $\neq$ BUSY) $\wedge$ ((Dock_State(dn_Fraise) = UNKNOWN)
 $\vee$ (Dock_State(dn_Fraise) = STOPPED)))
 $\vee$ (TIME() - REC_Fraise > 10s)

- * *L'alarme de la fraiseuse est à TRUE (il existe des données incohérentes*
- * *ou bien le statut est à FAULT)*
- * *si soit - le statut est IDLE et le Dock_State est différent de STOPPED*
- * *- le statut est différent de FAULT bien que le Dock_State soit*
- * *UNKNOWN*
- * *- le statut n'est pas BUSY bien que le Dock_State doit STOPPED*
- * *ou UNKNOWN*
- * *- la fraiseuse n'a plus répondu depuis plus de 10 secondes.*

Al_Fraise = FALSE $\equiv$ NOT ((st_Fraise = FAULT)
 $\vee$ ((st_Fraise = IDLE) $\wedge$ (Dock_State(dn_Fraise) $\neq$ STOPPED))
 $\vee$ ((st_Fraise $\neq$ FAULT) $\wedge$ (Dock_State(dn_Fraise)=UNKNOWN))
 $\vee$ ((st_Fraise $\neq$ BUSY) $\wedge$ ((Dock_State(dn_Fraise) = UNKNOWN)
 $\vee$ (Dock_State(dn_Fraise)= STOPPED)))
 $\vee$ (TIME() - REC_Fraise > 10s))

- * *L'alarme de la fraiseuse est à FALSE (toutes les données sont cohérentes) dans tous les autres cas.*

$AI_Tour = TRUE \equiv ((st_Tour = FAULT) \vee ((st_Tour = IDLE) \wedge (Dock_State(dn_Tour) \neq STOPPED)) \vee ((st_Tour \neq FAULT) \wedge (Dock_State(dn_Tour) = UNKNOWN)) \vee ((st_Tour \neq BUSY) \wedge ((Dock_State(dn_Tour) = UNKNOWN) \vee (Dock_State(dn_Tour) = STOPPED)))) \vee (TIME() - REC_Tour > 10s)$

- * *L'alarme du tour est à TRUE (il existe des données incohérentes ou bien le statut est à FAULT)*
- * *si soit - le statut est à FAULT*
- * *- le statut est IDLE et le Dock_State est différent de STOPPED*
- * *- le statut est différent de FAULT bien que le Dock_State soit UNKNOWN*
- * *- le statut n'est pas BUSY bien que le Dock_State soit STOPPED ou UNKNOWN*
- * *- le tour n'a plus répondu depuis plus de 10 secondes*

$AI_Tour = TRUE \equiv NOT((st_Tour = FAULT) \vee ((st_Tour = IDLE) \wedge (Dock_State(dn_Tour) \neq STOPPED)) \vee ((st_Tour \neq FAULT) \wedge (Dock_State(dn_Tour) = UNKNOWN)) \vee ((st_Tour \neq BUSY) \wedge ((Dock_State(dn_Tour) = UNKNOWN) \vee (Dock_State(dn_Tour) = STOPPED)))) \vee (TIME() - REC_Tour > 10s))$

- * *L'alarme du tour est à FALSE (toutes les données sont cohérentes) dans tous les autres cas.*

$AI_Robot = TRUE \equiv ((Status_Robot = FAULT) \vee (TIME() - REC_Robot > 10s))$

$AI_Robot = FALSE \equiv NOT((Status_Robot = FAULT) \vee (TIME() - REC_Robot > 10s))$

$AI_Agv = TRUE \equiv ((Status_Agv = FAULT) \vee (TIME() - REC_Agv > 10s))$

$AI_Agv = FALSE \equiv NOT((Status_Agv = FAULT) \vee (TIME() - REC_Agv > 10s))$

$AI_Bridage = TRUE \equiv (st_Bridage = FAULT) \vee ((st_Bridage \neq BUSY) \wedge (vitesse(dn_Bridage) \neq 0)) \vee (TIME() - REC_Bridage > 10s)$

- * *L'alarme du bridage est à TRUE (il existe des données incohérentes ou bien le statut est à FAULT)*
- * *si soit - le statut est FAULT*
- * *- la vitesse de bridage est différente de 0 et le statut n'est pas BUSY.*
- * *- le bridage n'a plus répondu depuis plus de 10 secondes*

$AI_Bridage = FALSE \equiv NOT((st_Bridage = FAULT) \vee ((st_Bridage \neq BUSY) \wedge (vitesse(dn_Bridage) \neq 0)) \vee (TIME() - REC_Bridage > 10s))$

- * *L'alarme du bridage est à FALSE (toutes les données sont cohérentes) dans tous les autres cas.*

• Contraintes locales

Comportement de l'état

/

Effets des actions

Tour.Maj_Tour(equ,inf):st_Tour = equ
dn_Tour = inf

REC_Tour = TIME(Tour.Maj_Tour(equ,inf))

- * Lorsque l'action Maj_Tour en provenance du tour à lieu, les anciennes
- * valeurs du monitoring sont remplacées par les nouvelles contenues
- * dans l'action et on conserve le "moment" où ça a lieu.

Fraiseuse.Maj_Fraise(equ, inf): st_Fraise = equ
dn_Fraise = inf

REC_Fraise = TIME(Fraise.Maj_Fraise(equ,inf))

- * Lorsque l'action Maj_Fraise en provenance de la fraiseuse à lieu, les
- * anciennes valeurs du monitoring sont remplacées par les nouvelles
- * contenues dans l'action et on conserve le "moment" où ça a lieu.

Robot.Maj_Robot(equip,infos) with Status(equip) = FAULT:

Status(Status_Fraise) = Status(equip)

Données_Robot = UNDEF

REC_Robot = TIME(Robot.Maj_Robot(equ,inf))

- * Si le status du robot est FAULT, alors aucune donnée le concernant
- * n'est valide du point de vue du monitoring.
- * On conserve également le "moment" de l'action.

Robot.Maj_Robot(equip,infos) with Status(equip) ≠ FAULT:

Status(Status_Fraise) = Status(equip)

Données_Robot = infos

REC_Robot = TIME(Robot.Maj_Robot(equ,inf))

- * Si le status du robot est différent de FAULT alors toutes les données
- * sont valides du point de vue du monitoring.
- * On conserve également le "moment" de l'action.

Agv.Maj_Agv(vehic, infos) with Status = FAULT

Status(Status_Agv) = Status(vehic)

Données_Agv = UNDEF

REC_Agv = TIME(Agv.Maj_Agv(veh,inf))

- * Si le status de l'Agv est FAULT, alors aucune donnée le concernant
- * n'est valide du point de vue du monitoring.
- * On conserve également le "moment" de l'action.

Agv.Maj_Agv(vehic, infos) with Station(infos) ≠ CLAMP

Status(Status_Agv) = Status(vehic)

Données_Agv = infos

REC_Agv = TIME(Agv.Maj_Agv(veh,inf))

- * Si la station est différente de CLAMP, alors toutes les données
- * concernant l'Agv sont valide du point de vue du monitoring.
- * On conserve également le "moment" de l'action.

Agv.Maj_Agv(vehic, infos) with Station(infos) = CLAMP

Status(Status_Agv) = Status(vehic)

Station(Données_Agv) = Station(infos)

Pallet_Post(Données_Agv) = UNDEF

Dock_State(Données_Agv) = Dock_State(infos)

Distance(Données_Agv) = Distance(infos)

Speed(Données_Agv) = Speed(infos)

Direction(Données_Agv) = Direction(infos)

REC_Agv = TIME(Agv.Maj_Agv(veh,inf))

- * Si la STATION est CLAMP, alors toutes les données de l'Agv sont

- * pertinentes pour le monitoring sauf PALLET_POST car le
- * bridage n'a qu'un seul support palette.
- * On conserve également le "moment" de l'action.

Bridage.Maj_Bridage(equ, inf):st_Bridage = equ
 dn_Bridage = inf
 REC_Bridage = TIME(Bridage.Maj_Bridage(equ,inf))

- * Lorsque l'action Maj_Bridage en provenance du bridage à lieu, les
- * anciennes valeurs du monitoring sont remplacées par les nouvelles
- * contenues dans l'action et on conserve le "moment" où ça a lieu.

Causalité

Tour.Maj_Tour $\hat{0} \leq 5s \rightarrow$ Tour_Dem
 Agv.Maj_Agv $\hat{0} \leq 5s \rightarrow$ Agv_Dem
 Fraise.Maj_Fraise $\hat{0} \leq 5s \rightarrow$ Fraise_Dem
 Robot.Maj_Robot $\hat{0} \leq 5s \rightarrow$ Robot_Dem
 Bridage.Maj_Bridage $\hat{0} \leq 5s \rightarrow$ Bridage_Dem

- * Lorsqu'une action de mise à jour des valeurs d'une machine a lieu,
- * dans les 5 secondes qui suivent, le monitoring doit faire une nouvelle
- * demande d'information sur la machine dont l'action de mise à jour a eu
- * lieu.

Capacité

O(Fraise_Dem / ((Status_Fraise = UNDEF) $\vee$ (TIME() - REC_Fraise > 10s))

- * Si Status_Fraise est à UNDEF ou elle ne répond plus, alors une
- * action Fraise_Dem est générée.(cela signifie qu'on a aucune données
- * sur cette machine et qu'il faut donc en demander)

O(Robot_Dem / ((Status_Robot = UNDEF) $\vee$ (TIME() - REC_Robot > 10s))

- * Si Status_Robot est à UNDEF ou il ne répond plus, alors une
- * action Robot_Dem est générée.(cela signifie qu'on a aucune données
- * sur cette machine et qu'il faut donc en demander)

O(Tour_Dem / ((Status_Tour = UNDEF) $\vee$ (TIME() - REC_Tour > 10s))

- * Si Status_Tour est à UNDEF ou il ne répond plus, alors une
- * action Tour_Dem est générée.(cela signifie qu'on a aucune données
- * sur cette machine et qu'il faut donc en demander)

O(Agv_Dem / ((Status_Agv = UNDEF) $\vee$ (TIME() - REC_Agv > 10s))

- * Si Status_Agv est à UNDEF ou il ne répond plus, alors une
- * action Agv_Dem est générée.(cela signifie qu'on a aucune données
- * sur cette machine et qu'il faut donc en demander)

O(Bridage_Dem / ((Status_Bridage = UNDEF) $\vee$ (TIME() - REC_Bridage > 10s))

- * Si Status_Bridage est à UNDEF ou il ne répond plus, alors une
- * action Bridage_Dem est générée.(cela signifie qu'on a aucune données
- * sur cette machine et qu'il faut donc en demander)

• Contraintes de coopération

Perception des actions

K (Tour.Maj_Tour(equip, infos) / TRUE)
 K (Fraiseuse.Maj_Fraise(equip, infos) / TRUE)
 K (Bridage.Maj_Bridage(equip, infos) / TRUE)
 K (Robot.Maj_Robot(equip, infos) / TRUE)
 K (Agv.Maj_Agv(vehicule, infos) / TRUE)

- * *Lorsqu'une réponse, à une demande d'information du moniteur, arrive*
- * *d'une machine, elle doit toujours être prise en compte.*

Perception de l'état

/

Informations sur les actions

K (Tour_Dem().Tour / TRUE)
 K (Fraise_Dem().Fraiseuse / TRUE)
 K (Robot_Dem().Robot / TRUE)
 K (Agv_Dem().Agv / TRUE)
 K (Bridage_Dem().Bridage / TRUE)

- * *Lorsqu'une action de demande d'info est générée par le moniteur, elle*
- * *est rendue visible à la machine concernée.*

Informations sur l'état

K(Manager.Status_Fraise / TRUE)
 K(Manager.Status_Agv / TRUE)
 K(Manager.Status_Bridage / TRUE)
 K(Manager.Status_Tour / TRUE)
 K(Manager.Status_Robot / TRUE)
 K(Manager.Données_Fraise / TRUE)
 K(Manager.Données_Agv / TRUE)
 K(Manager.Données_Bridage / TRUE)
 K(Manager.Données_Tour / TRUE)
 K(Manager.Données_Robot / TRUE)

- * *Le moniteur laisse toujours voir toutes ses données au manager*

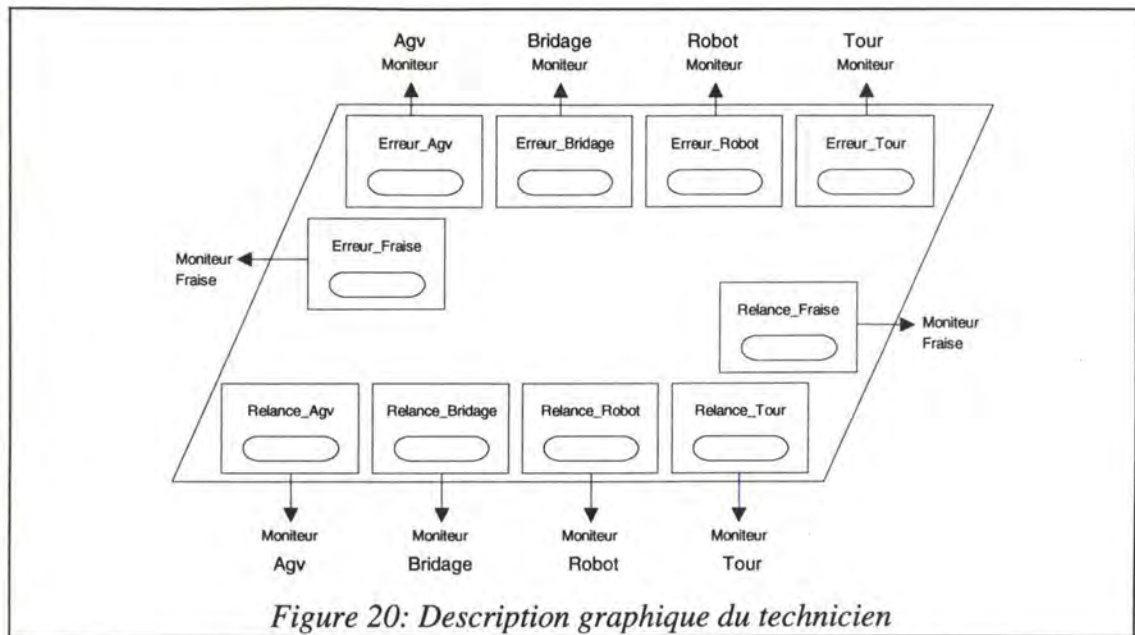
5) Version 5 : Le Technicien

a) Définition des nouveaux³ types de données utilisées

PANNE: {EN_PANNE, EN_ETAT}

b) Description des agents

³pour la définition des autres types, se reporter au point "a) Les types de données utilisées" du chapitre "3.1 Version 1: Le Moniteur voit tout - Les Machines montrent tout" et au point "a) Les nouveaux types de données utilisées" du chapitre "3.4.1 Les machines envoient des informations incohérentes"

L'agent Technicien

• Contraintes de Base

Etat initial

/

Composants dérivés

/

• Contraintes locales

Comportement de l'état

/

Effets des actions

/

Causalité

/

Capacité

/

• Contraintes de coopération

Perception des actions

/

Perception de l'état

/

Informations sur les actions

K (Erreur_Agv().Moniteur / TRUE)
 K (Erreur_Robot().Moniteur / TRUE)
 K (Erreur_Bridge().Moniteur / TRUE)
 K (Erreur_Tour().Moniteur / TRUE)
 K (Erreur_Fraise().Moniteur / TRUE)
 K (Erreur_Agv().Agv / TRUE)
 K (Erreur_Robot().Robot / TRUE)
 K (Erreur_Bridge().Bridge / TRUE)
 K (Erreur_Tour().Tour / TRUE)
 K (Erreur_Fraise().Fraiseuse / TRUE)

- * Lorsque le technicien détecte une panne de machine, le moniteur
- * et la machine en sont informés.

K (Relance_Agv().Moniteur / TRUE)
 K (Relance_Robot().Moniteur / TRUE)
 K (Relance_Bridge().Moniteur / TRUE)
 K (Relance_Tour().Moniteur / TRUE)
 K (Relance_Fraise().Moniteur / TRUE)
 K (Relance_Agv().Agv / TRUE)
 K (Relance_Robot().Robot / TRUE)
 K (Relance_Bridge().Bridge / TRUE)
 K (Relance_Tour().Tour / TRUE)
 K (Relance_Fraise().Fraiseuse / TRUE)

* Lorsque le technicien a réparé une machine et la relance, le moniteur
 * et la machine en sont informés.

Informations sur l'état

/

L'agent Moniteur

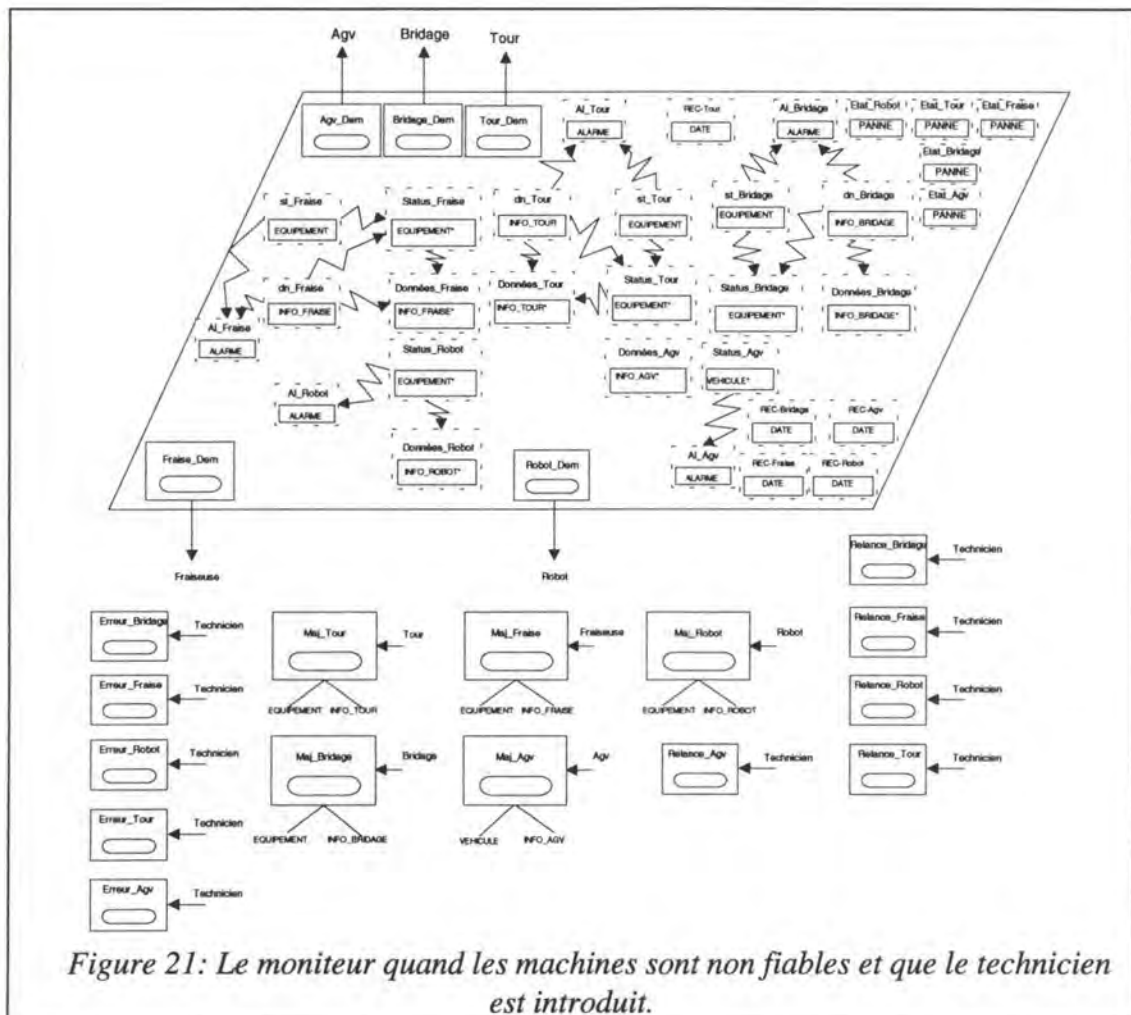


Figure 21: Le moniteur quand les machines sont non fiables et que le technicien est introduit.

• Contraintes de Base

Etat initial

Status_Fraise = UNDEF
 Status_Robot = UNDEF
 Status_Tour = UNDEF
 Status_Agv = UNDEF

Status_Bridage = UNDEF
 Données_Fraise = UNDEF
 Données_Robot = UNDEF
 Données_Tour = UNDEF
 Données_Agv = UNDEF
 Données_Bridage = UNDEF

- * A l'état initial, les différentes données représentant les machines sont
- * à UNDEF.

Etat_Agv = EN_ETAT
 Etat_Tour = EN_ETAT
 Etat_Fraise = EN_ETAT
 Etat_Robot = EN_ETAT
 Etat_Bridage = EN_ETAT

- * A l'état initial, toutes les machines sont considérées comme étant en
- * état de marche.

Composants dérivés

Status_Fraise = IDLE $\equiv$ (st_Fraise = IDLE) $\wedge$ (Dock_State(dn_Fraise) = STOPPED)

- * Si le status de la machine est IDLE et le Dock_State est STOPPED,
- * alors le status IDLE est valide pour le monitoring.

Status_Fraise = FAULT $\equiv$ (Dock_State(dn_Fraise) = UNKNOWN)
 $\vee$ ((st_Fraise = IDLE) $\wedge$ (Dock_State(dn_Fraise) $\neq$ STOPPED))
 $\vee$ ((st_Fraise = BUSY) $\wedge$ ((Dock_State(dn_Fraise) = STOPPED)
 $\vee$ (Dock_State(dn_Fraise) = UNKNOWN))

- * Si le Dock_State de la machine est UNKNOWN, ou que le statut de la
- * machine est IDLE avec un Dock_State différent de STOPPED, ou que le
- * statut de la machine est BUSY avec un Dock_State à STOPPED ou
- * UNKNOWN, alors le seul status valable pour le monitoring est FAULT

Status_Fraise = BUSY $\equiv$ (st_Fraise = BUSY)
 $\wedge$ (Dock_State(dn_Fraise) $\neq$ STOPPED)
 $\wedge$ (Dock_State(dn_Fraise) $\neq$ UNKNOWN)

- * Si le Dock_State de la machine est différent de STOPPED et de
- * UNKNOWN et le status est BUSY alors il est correct et le status pour
- * le monitoring est aussi BUSY.

Dock_State(Données_Fraise) = STOPPED $\equiv$ (Status_Fraise = IDLE)

- * Si le status de la fraiseuse pour le monitoring est IDLE, alors le
- * Dock_State est STOPPED.

PGM(Données_Fraise) = UNDEF $\equiv$ (Status_Fraise = IDLE)

Pallet_Post(Données_Fraise) = UNDEF $\equiv$ (Status_Fraise = IDLE)

POSXYZ(Données_Fraise) = UNDEF $\equiv$ (Status_Fraise = IDLE)

- * Si le status de la fraiseuse pour le monitoring est IDLE, alors toutes
- * les données autres que le Dock_State et le status sont non
- * pertinentes.

Dock_State(Données_Fraise) = Dock_State(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)

PGM(Données_Fraise) = PGM(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)

Pallet_Post(Données_Fraise) = Pallet_Post(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)

POSXYZ(Données_Fraise) = POSXYZ(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)

- * Si le status de la fraiseuse pour le monitoring est BUSY, alors toutes
- * les données sont pertinentes.

Données_Fraise = UNDEF $\equiv$ (Status_Fraise = FAULT)

- * Si le status de la fraiseuse pour le monitoring est FAULT, alors aucune donnée concernant le tour n'est valide du point de vue du monitoring.

Status_Tour = IDLE $\equiv$ (st_Tour = IDLE) $\wedge$ (Dock_State(dn_Tour) = STOPPED)

- * Si le status de la machine est IDLE et le Dock_State est STOPPED,
- * alors le status IDLE est valide pour le monitoring.

Status_Tour = FAULT $\equiv$ (Dock_State(dn_Tour) = UNKNOWN)

$\vee ((\text{st_Fraise} = \text{IDLE}) \wedge (\text{Dock_State}(\text{dn_Tour}) \neq \text{STOPPED}))$

$\vee ((\text{st_Fraise} = \text{BUSY}) \wedge ((\text{Dock_State}(\text{dn_Tour}) = \text{STOPPED})$

$\vee (\text{Dock_State}(\text{dn_Tour}) = \text{UNKNOWN})))$

- * Si le Dock_State de la machine est UNKNOWN, ou que le statut de la machine est IDLE avec un Dock_State différent de STOPPED, ou que le statut de la machine est BUSY avec un Dock_State à STOPPED ou UNKNOWN, alors le seul status valable pour le monitoring est FAULT

Status_Tour = BUSY $\equiv$ (st_Tour = BUSY)

$\wedge (\text{Dock_State}(\text{dn_Tour}) \neq \text{STOPPED})$

$\wedge (\text{Dock_State}(\text{dn_Tour}) \neq \text{UNKNOWN})$

- * Si le Dock_State de la machine est différent de STOPPED et de UNKNOWN et le status est BUSY alors il est correct et le status pour le monitoring est aussi BUSY.

Dock_State(Données_Tour) = STOPPED $\equiv$ (Status_Tour = IDLE)

- * Si le status du tour pour le monitoring est IDLE, alors le Dock_State est STOPPED.

PGM(Données_Tour) = UNDEF $\equiv$ (Status_Tour = IDLE)

Pallet_Post(Données_Tour) = UNDEF $\equiv$ (Status_Tour = IDLE)

POSXYZ(Données_Tour) = UNDEF $\equiv$ (Status_Tour = IDLE)

- * Si le status du tour pour le monitoring est IDLE, alors toutes les données autres que le Dock_State et le status sont non pertinentes.

Dock_State(Données_Tour) = Dock_State(dn_Tour) $\equiv$ (Status_Tour = BUSY)

PGM(Données_Tour) = PGM(dn_Tour) $\equiv$ (Status_Tour = BUSY)

Pallet_Post(Données_Tour) = Pallet_Post(dn_Tour) $\equiv$ (Status_Tour = BUSY)

POSXYZ(Données_Tour) = POSXYZ(dn_Tour) $\equiv$ (Status_Tour = BUSY)

- * Si le status du tour pour le monitoring est BUSY, alors toutes les données sont pertinentes.

Données_Tour = UNDEF $\equiv$ (Status_Tour = FAULT)

- * Si le status du tour pour le monitoring est FAULT, alors aucune donnée concernant le tour n'est valide du point de vue du monitoring.

Status_Bridage = BUSY $\equiv$ (vitesse(dn_Bridage) $\neq$ 0)

- * Si la vitesse de bridage est différente de 0, alors le status doit être BUSY pour le monitoring.

Status_Bridage = st_Bridage $\equiv$ (vitesse(dn_Bridage) = 0)

- * Si la vitesse de bridage vaut 0, alors le status du bridage correspond à ce qu'il doit être pour le monitoring.

Données_Bridage = dn_Bridage $\equiv$ (Status_Bridage $\neq$ FAULT)

Données_Bridage = UNDEF $\equiv$ (Status_Bridage = FAULT)

- * *Les autres données que le statut ne sont pertinentes que si le statut*
- * *n'est pas à FAULT.*

Al_Fraise = TRUE $\equiv$ (st_Fraise = FAULT)

$\vee((\text{st_Fraise} = \text{IDLE}) \wedge (\text{Dock_State}(\text{dn_Fraise}) \neq \text{STOPPED}))$

$\vee((\text{st_Fraise} \neq \text{FAULT}) \wedge (\text{Dock_State}(\text{dn_Fraise}) = \text{UNKNOWN}))$

$\vee((\text{st_Fraise} \neq \text{BUSY}) \wedge ((\text{Dock_State}(\text{dn_Fraise}) = \text{UNKNOWN})$

$\vee(\text{Dock_State}(\text{dn_Fraise}) = \text{STOPPED})))$

$\vee(\text{TIME}() - \text{REC_Fraise} > 10\text{s})$

- * *L'alarme de la fraiseuse est à TRUE (il existe des données incohérentes*
- * *ou bien le statut est à FAULT)*
- * *si soit - le statut est IDLE et le Dock_State est différent de STOPPED*
- * *- le statut est différent de FAULT bien que le Dock_State soit*
- * *UNKNOWN*
- * *- le statut n'est pas BUSY bien que le Dock_State doit STOPPED*
- * *ou UNKNOWN*
- * *- la fraiseuse n'a plus répondu depuis plus de 10 secondes.*

Al_Fraise = FALSE $\equiv$ NOT ((st_Fraise = FAULT)

$\vee((\text{st_Fraise} = \text{IDLE}) \wedge (\text{Dock_State}(\text{dn_Fraise}) \neq \text{STOPPED}))$

$\vee((\text{st_Fraise} \neq \text{FAULT}) \wedge (\text{Dock_State}(\text{dn_Fraise}) = \text{UNKNOWN}))$

$\vee((\text{st_Fraise} \neq \text{BUSY}) \wedge ((\text{Dock_State}(\text{dn_Fraise}) = \text{UNKNOWN})$

$\vee(\text{Dock_State}(\text{dn_Fraise}) = \text{STOPPED})))$

$\vee(\text{TIME}() - \text{REC_Fraise} > 10\text{s}))$

- * *L'alarme de la fraiseuse est à FALSE (les données sont cohérentes)*
- * *dans tous les autres cas.*

Al_Tour = TRUE $\equiv$ (st_Tour = FAULT)

$\vee((\text{st_Tour} = \text{IDLE}) \wedge (\text{Dock_State}(\text{dn_Tour}) \neq \text{STOPPED}))$

$\vee((\text{st_Tour} \neq \text{FAULT}) \wedge (\text{Dock_State}(\text{dn_Tour}) = \text{UNKNOWN}))$

$\vee((\text{st_Tour} \neq \text{BUSY}) \wedge ((\text{Dock_State}(\text{dn_Tour}) = \text{UNKNOWN})$

$\vee(\text{Dock_State}(\text{dn_Tour}) = \text{STOPPED})))$

$\vee(\text{TIME}() - \text{REC_Tour} > 10\text{s})$

- * *L'alarme du tour est à TRUE (il existe des données incohérentes*
- * *ou bien le statut est à FAULT)*
- * *si soit - le statut est à FAULT*
- * *- le statut est IDLE et le Dock_State est différent de STOPPED*
- * *- le statut est différent de FAULT bien que le Dock_State soit*
- * *UNKNOWN*
- * *- le statut n'est pas BUSY bien que le Dock_State doit STOPPED*
- * *ou UNKNOWN*
- * *- le tour n'a plus répondu depuis plus de 10 secondes*

Al_Tour = TRUE $\equiv$ NOT((st_Tour = FAULT)

$\vee((\text{st_Tour} = \text{IDLE}) \wedge (\text{Dock_State}(\text{dn_Tour}) \neq \text{STOPPED}))$

$\vee((\text{st_Tour} \neq \text{FAULT}) \wedge (\text{Dock_State}(\text{dn_Tour}) = \text{UNKNOWN}))$

$\vee((\text{st_Tour} \neq \text{BUSY}) \wedge ((\text{Dock_State}(\text{dn_Tour}) = \text{UNKNOWN})$

$\vee(\text{Dock_State}(\text{dn_Tour}) = \text{STOPPED})))$

$\vee(\text{TIME}() - \text{REC_Tour} > 10\text{s}))$

- * *L'alarme du tour est à FALSE (les données sont cohérentes) dans tous*
- * *les autres cas.*

$AI_Robot = TRUE \equiv ((Status_Robot = FAULT) \vee (TIME() - REC_Robot > 10s))$

$AI_Robot = FALSE \equiv NOT((Status_Robot = FAULT) \vee (TIME() - REC_Robot > 10s))$

$AI_Agv = TRUE \equiv ((Status_Agv = FAULT) \vee (TIME() - REC_Agv > 10s))$

$AI_Agv = FALSE \equiv NOT((Status_Agv = FAULT) \vee (TIME() - REC_Agv > 10s))$

$AI_Bridge = TRUE \equiv (st_Bridge = FAULT)$

$\vee((st_Bridge \neq BUSY) \wedge (vitesse(dn_Bridge) \neq 0))$

$\vee(TIME() - REC_Bridge > 10s)$

** L'alarme du bridage est à TRUE (il existe des données incohérentes*

** ou bien le statut est à FAULT)*

** si soit - le statut est FAULT*

** - la vitesse de bridage est différente de 0 et le statut n'est pas BUSY.*

** - le bridage n'a plus répondu depuis plus de 10 secondes*

$AI_Bridge = FALSE \equiv NOT((st_Bridge = FAULT)$

$\vee((st_Bridge \neq BUSY) \wedge (vitesse(dn_Bridge) \neq 0))$

$\vee(TIME() - REC_Bridge > 10s))$

** L'alarme du bridage est à FALSE (les données sont cohérentes) dans*

** tous les autres cas.*

• Contraintes locales

Comportement de l'état

/

Effets des actions

Tour.Maj_Tour(equ,inf): $st_Tour = equ$

$dn_Tour = inf$

$REC_Tour = TIME(Tour.Maj_Tour(equ,inf))$

** Lorsque l'action Maj_Tour en provenance du tour à lieu, les anciennes*

** valeurs du monitoring sont remplacées par les nouvelles contenues*

** dans l'action et on conserve le "moment" où ça a lieu.*

Fraiseuse.Maj_Fraise(equ, inf): $st_Fraise = equ$

$dn_Fraise = inf$

$REC_Fraise =$

$TIME(Fraise.Maj_Fraise(equ,inf))$

** Lorsque l'action Maj_Fraise en provenance de la fraiseuse à lieu, les*

** anciennes valeurs du monitoring sont remplacées par les nouvelles*

** contenues dans l'action et on conserve le "moment" où ça a lieu.*

Robot.Maj_Robot(equip,infos) with $Status(equip) = FAULT$:

$Status_Fraise = Status(equip)$

$Données_Robot = UNDEF$

$REC_Robot = TIME(Robot.Maj_Robot(equ,inf))$

** Si le status du robot est FAULT, alors aucune donnée le concernant*

** n'est valide du point de vue du monitoring.*

** On conserve également le "moment" de l'action.*

Robot.Maj_Robot(equip,infos) with $Status(equip) \neq FAULT$:

$Status_Fraise = Status(equip)$

$Données_Robot = infos$

$REC_Robot = TIME(Robot.Maj_Robot(equ,inf))$

- * *Si le status du robot est différent de FAULT alors toutes les données*
- * *sont valides du point de vue du monitoring.*
- * *On conserve également le "moment" de l'action.*

Agv.Maj_Agv(vehic, infos) with Status = FAULT
 Status(Status_Agv) = Status(vehic)
 Données_Agv = UNDEF
 REC_Agv = TIME(Agv.Maj_Agv(vehic,inf))
 * *Si le status de l'Agv est FAULT, alors aucune donnée le concernant*
 * *n'est valide du point de vue du monitoring.*
 * *On conserve également le "moment" de l'action.*

Agv.Maj_Agv(vehic, infos) with Station(infos) ≠ CLAMP
 Status(Status_Agv) = Status(vehic)
 Données_Agv = infos
 REC_Agv = TIME(Agv.Maj_Agv(vehic,inf))
 * *Si la station est différente de CLAMP, alors toutes les données*
 * *concernant l'Agv sont valide du point de vue du monitoring.*
 * *On conserve également le "moment" de l'action.*

Agv.Maj_Agv(vehic, infos) with Station(infos) = CLAMP
 Status(Status_Agv) = Status(vehic)
 Station(Données_Agv) = Station(infos)
 Pallet_Post(Données_Agv) = UNDEF
 Dock_State(Données_Agv) = Dock_State(infos)
 Distance(Données_Agv) = Distance(infos)
 Speed(Données_Agv) = Speed(infos)
 Direction(Données_Agv) = Direction(infos)
 REC_Agv = TIME(Agv.Maj_Agv(vehic,inf))
 * *Si la STATION est CLAMP, alors toutes les données de l'Agv sont*
 * *pertinentes pour le monitoring sauf PALLET_POST car le*
 * *bridage n'a qu'un seul support palette.*
 * *On conserve également le "moment" de l'action.*

Bridage.Maj_Bridage(equ, inf): st_Bridage = equ
 dn_Bridage = inf
 REC_Bridage =
 TIME(Bridage.Maj_Bridage(equ,inf))
 * *Lorsque l'action Maj_Bridage en provenance du bridage à lieu, les*
 * *anciennes valeurs du monitoring sont remplacées par les nouvelles*
 * *contenues dans l'action et on conserve le "moment" où ça a lieu.*

Technicien.Erreur_Agv(): Etat_Agv = EN_PANNE
 Technicien.Erreur_Tour(): Etat_Tour = EN_PANNE
 Technicien.Erreur_Robot(): Etat_Robot = EN_PANNE
 Technicien.Erreur_Bridage(): Etat_Bridage = EN_PANNE
 Technicien.Erreur_Fraise(): Etat_Fraise = EN_PANNE
 * *Lorsque le moniteur perçoit l'action du technicien signalant qu'une*
 * *machine donnée est en panne, il le mémorise.*

Technicien.Relance_Agv(): (Etat_Agv = EN_ETAT) ∧ (Status_Agv= UNDEF)
 Technicien.Relance_Tour(): (Etat_Tour = EN_ETAT) ∧ (Status_Tour= UNDEF)
 Technicien.Relance_Robot(): Etat_Robot = EN_ETAT
 Status_Robot = UNDEF
 Technicien.Relance_Bridage(): Etat_Bridage = EN_ETAT
 Status_Bridage = UNDEF
 Technicien.Relance_Fraise(): Etat_Fraise = EN_ETAT

Status_Fraise = UNDEF

- * *Lorsque le moniteur perçoit l'action du technicien signalant qu'une*
- * *machine donnée vient d'être relancée, il doit la considérer comme*
- * *en état de marche et réinitialisée.*

Causalité

Tour.Maj_Tour $\hat{O} \leq 5s \rightarrow$ Tour_Dem

Agv.Maj_Agv $\hat{O} \leq 5s \rightarrow$ Agv_Dem

Fraise.Maj_Fraise $\hat{O} \leq 5s \rightarrow$ Fraise_Dem

Robot.Maj_Robot $\hat{O} \leq 5s \rightarrow$ Robot_Dem

Bridage.Maj_Bridage $\hat{O} \leq 5s \rightarrow$ Bridage_Dem

- * *Lorsqu'une action de mise à jour des valeurs d'une machine a lieu,*
- * *dans les 5 secondes qui suivent, le monitoring doit faire une nouvelle*
- * *demande d'information sur la machine dont l'action de mise à jour a eu*
- * *lieu.*

Capacité

O(Fraise_Dem / ((Status_Fraise = UNDEF) $\vee$ (TIME() - REC_Fraise > 10s))
 $\wedge$ (Etat_Fraise = EN_ETAT))

- * *Si Status_Fraise est à UNDEF ou que la machine ne répond plus, alors*
- * *une action Fraise_Dem est générée.(cela signifie qu'on a aucune*
- * *données sur cette machine et qu'il faut donc en demander).*
- * *Il faut en plus que la machine soit en état de marche (qu'elle ne soit pas*
- * *signalée en panne par le technicien).*

O(Robot_Dem / ((Status_Robot = UNDEF) $\vee$ (TIME() - REC_Robot > 10s))
 $\wedge$ (Etat_Robot = EN_ETAT))

- * *Si Status_Robot est à UNDEF ou que la machine ne répond plus, alors*
- * *une action Robot_Dem est générée.(cela signifie qu'on a aucune*
- * *données sur cette machine et qu'il faut donc en demander)*
- * *Il faut en plus que la machine soit en état de marche (qu'elle ne soit pas*
- * *signalée en panne par le technicien).*

O(Tour_Dem / ((Status_Tour = UNDEF) $\vee$ (TIME() - REC_Tour > 10s))
 $\wedge$ (Etat_Tour = EN_ETAT))

- * *Si Status_Tour est à UNDEF ou que la machine ne répond plus, alors*
- * *une action Tour_Dem est générée.(cela signifie qu'on a aucune données*
- * *sur cette machine et qu'il faut donc en demander).*
- * *Il faut en plus que la machine soit en état de marche (qu'elle ne soit pas*
- * *signalée en panne par le technicien).*

O(Agv_Dem / ((Status_Agv = UNDEF) $\vee$ (TIME() - REC_Agv > 10s))
 $\wedge$ (Etat_Agv = EN_ETAT))

- * *Si Status_Agv est à UNDEF ou que la machine ne répond plus, alors*
- * *une action Agv_Dem est générée.(cela signifie qu'on a aucune données*
- * *sur cette machine et qu'il faut donc en demander).*
- * *Il faut en plus que la machine soit en état de marche (qu'elle ne soit pas*
- * *signalée en panne par le technicien).*

O(Bridage_Dem / ((Status_Bridage = UNDEF) $\vee$ (TIME() - REC_Bridage > 10s))
 $\wedge$ (Etat_Bridage = EN_ETAT))

- * *Si Status_Bridage est à UNDEF ou que la machine ne répond plus, alors*
- * *une action Bridage_Dem est générée.(cela signifie qu'on a aucune*
- * *données sur cette machine et qu'il faut donc en demander).*

- * *Il faut en plus que la machine soit en etat de marche (qu'elle ne soit pas*
- * *signalée en panne par le technicien).*

• **Contraintes de coopération**

Perception des actions

K (Tour.Maj_Tour(equip, infos) / TRUE)
 K (Fraiseuse.Maj_Fraise(equip, infos) / TRUE)
 K (Bridage.Maj_Bridage(equip, infos) / TRUE)
 K (Robot.Maj_Robot(equip, infos) / TRUE)
 K (Agv.Maj_Agv(vehicule, infos) / TRUE)

- * *Lorsqu'une réponse, à une demande d'information du moniteur, arrive*
- * *d'une machine, elle doit toujours être prise en compte.*

K(Technicien.Erreur_Robot() / TRUE)
 K(Technicien.Erreur_Agv() / TRUE)
 K(Technicien.Erreur_Tour() / TRUE)
 K(Technicien.Erreur_Fraise() / TRUE)
 K(Technicien.Erreur_Bridage() / TRUE)

- * *Lorsqu'une action signalant la panne d'une machine arrive au moniteur,*
- * *elle doit toujours être prise en compte.*

K(Technicien.Relance_Robot() / TRUE)
 K(Technicien.Relance_Agv() / TRUE)
 K(Technicien.Relance_Tour() / TRUE)
 K(Technicien.Relance_Fraise() / TRUE)
 K(Technicien.Relance_Bridage() / TRUE)

- * *Lorsqu'une action signalant la relance d'une machine arrive au*
- * *moniteur, elle doit toujours être prise en compte.*

Perception de l'état

/

K(Manager.Status_Fraise / TRUE)
 K(Manager.Status_Agv / TRUE)
 K(Manager.Status_Bridage / TRUE)
 K(Manager.Status_Tour / TRUE)
 K(Manager.Status_Robot / TRUE)
 K(Manager.Données_Fraise / TRUE)
 K(Manager.Données_Agv / TRUE)
 K(Manager.Données_Bridage / TRUE)
 K(Manager.Données_Tour / TRUE)
 K(Manager.Données_Robot / TRUE)

- * *Le moniteur laisse toujours voir toutes ses données au manager*

Informations sur les actions

K (Tour_Dem().Tour / TRUE)
 K (Fraise_Dem().Fraiseuse / TRUE)
 K (Robot_Dem().Robot / TRUE)
 K (Agv_Dem().Agv / TRUE)
 K (Bridage_Dem().Bridage / TRUE)

- * *Lorsqu'une action de demande d'info est générée par le moniteur, elle*
- * *est rendue visible à la machine concernée.*

Informations sur l'état

K(Manager.Status_Fraise / TRUE)
 K(Manager.Status_Agv / TRUE)
 K(Manager.Status_Bridage / TRUE)
 K(Manager.Status_Tour / TRUE)
 K(Manager.Status_Robot / TRUE)

K(Manager.Données_Fraise / TRUE)
 K(Manager.Données_Agv / TRUE)
 K(Manager.Données_Bridage / TRUE)
 K(Manager.Données_Tour / TRUE)
 K(Manager.Données_Robot / TRUE)

** Le moniteur laisse toujours voir toutes ses données au manager*

L'agent Machine Agv

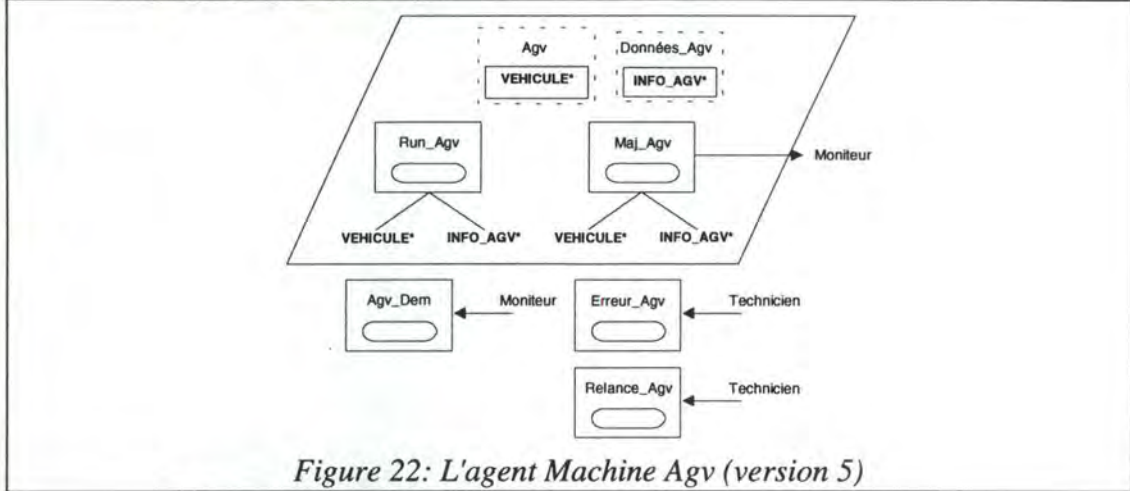


Figure 22: L'agent Machine Agv (version 5)

• Contraintes de Base

Etat initial

/

Composants dérivés

/

• Contraintes locales

Comportement de l'état

/

Effets des actions

Technicien.Erreur_Agv(): Status(Agv) = FAULT

Technicien.Relance_Agv(): Status(Agv) = IDLE

Run_Agv(a,b): Agv = a
 Données_Agv = b

Causalité

Moniteur.Agv_Dem $\hat{=}$ 1s $\rightarrow$ Maj_Agv(vehicule,infos) with vehicule = Agv
 infos = Données_Agv

$\oplus$ DAC⁴

Capacité

F (Run_Agv(.,.) / Status(Agv) = FAULT)

** L'action run_Agv ne peut avoir lieu si le statut de la machine*

** est à FAULT.*

• Contraintes de coopération

Perception des actions

K(Technicien.Erreur_Agv() / TRUE)

K(Technicien.Relance_Agv() / TRUE)

Perception de l'état

/

⁴DAC signifie Dummy ACTION. Il s'agit d'une action ne réalisant rien.

Informations sur les actions

/

Informations sur l'état

/

L'agent Machine-Bridge

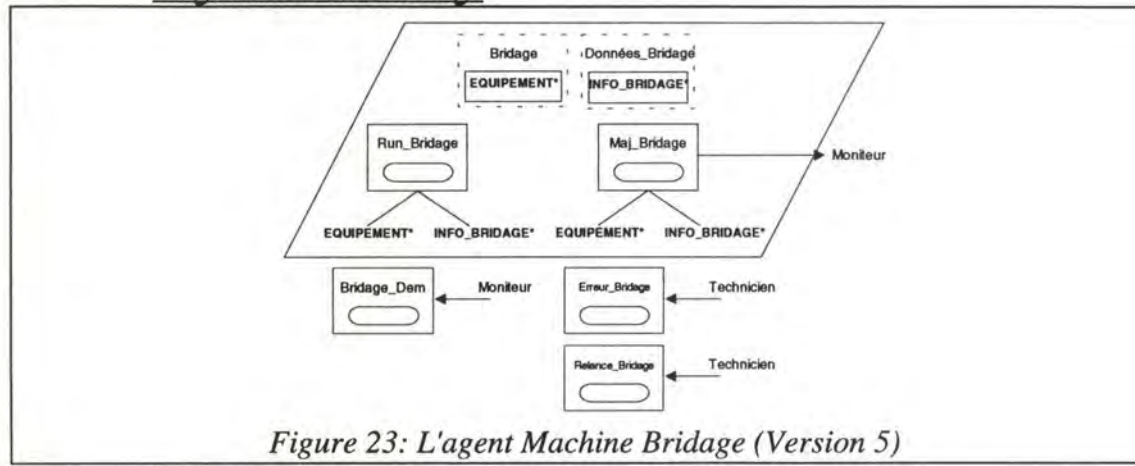


Figure 23: L'agent Machine Bridge (Version 5)

• Contraintes de Base

Etat initial

/

Composants dérivés

/

• Contraintes locales

Comportement de l'état

/

Effets des actions

Technicien.Erreur_Bridge(): Status(Bridge) = FAULT

Technicien.Relance_Bridge(): Status(Bridge) = IDLE

Run_Bridge(a,b): Bridge = a

Données_Bridge = b

Causalité

Moniteur.Bridge_Dem $\forall \leq 1s \rightarrow$ Maj_Bridge(equip,infos) with equip = Bridge
infos = Données_Bridge

$\oplus$ DAC

Capacité

F (Run_Bridge(.,.) / Status(Bridge) = FAULT)

* L'action run_Bridge ne peut avoir lieu si le statut de la machine

* est à fault.

• Contraintes de coopération

Perception des actions

K(Technicien.Erreur_Bridge() / TRUE)

K(Technicien.Relance_Bridge() / TRUE)

Perception de l'état

/

Informations sur les actions

/

Informations sur l'état

/

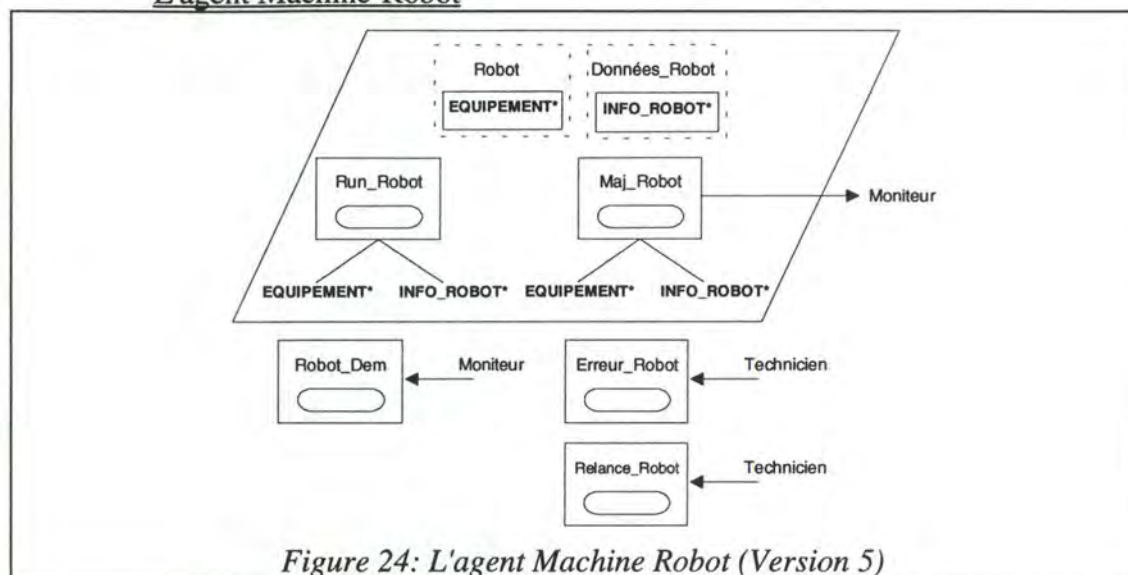
L'agent Machine-Robot

Figure 24: L'agent Machine Robot (Version 5)

• Contraintes de Base

Etat initial

/

Composants dérivés

/

• Contraintes locales

Comportement de l'état

/

Effets des actions

Technicien.Erreur_Robot(): Status(Robot) = FAULT

Technicien.Relance_Robot(): Status(Robot) = IDLE

Run_Robot(a,b): Robot = a
Données_Robot = bCausalitéMoniteur.Robot_Dem $\hat{0} \leq 1s \rightarrow$ Maj_Robot(equip,infos) with equip = Robot
infos = Données_Robot $\oplus$ DACCapacité

F (Run_Robot(.,.) / Status(Robot) = FAULT)

- * L'action run_Robot ne peut avoir lieu si le statut de la machine
- * est à fault.

• Contraintes de coopération

Perception des actions

K(Technicien.Erreur_Robot() / TRUE)

K(Technicien.Relance_Robot() / TRUE)

Perception de l'état

/

Informations sur les actions

/

Informations sur l'état

/

L'agent Machine-Tour

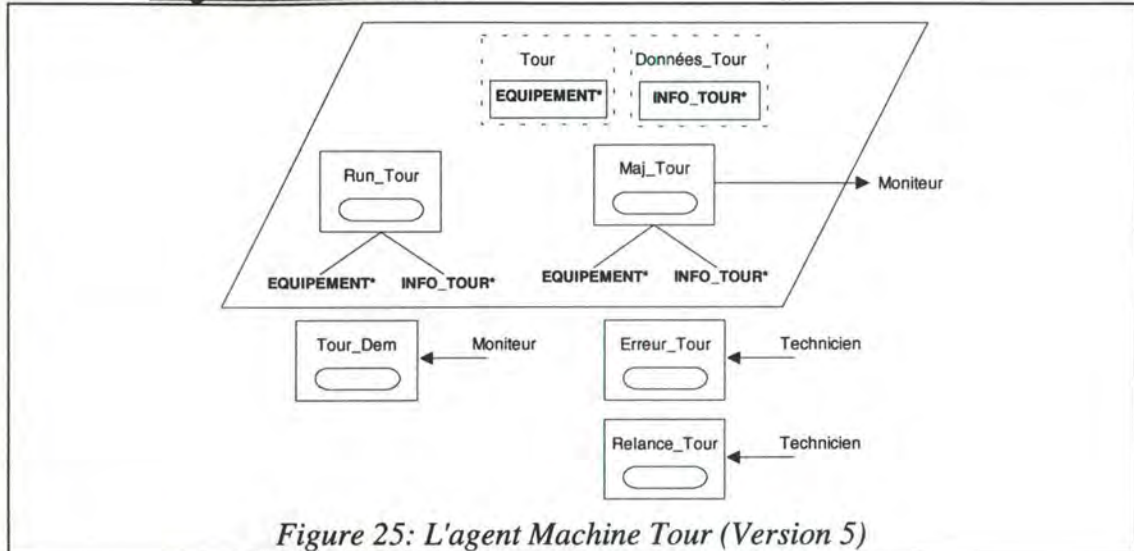


Figure 25: L'agent Machine Tour (Version 5)

• Contraintes de Base

Etat initial

/

Composants dérivés

/

- **Contraintes locales**

Comportement de l'état

1

Effets des actions

Techniciens.Erreur_Tour(): Status(Tour) = FAULT

Technicien.Relance_Tour(): Status(Tour) = IDLE

```
Run_Tour(a,b):  Tour = a
                Données_Tour = b
```

Causalité

$$\text{Moniteur.Tour_Dem} \stackrel{\delta \leq 1s}{\rightarrow} \text{Maj_Tour}(\text{equip}, \text{infos}) \text{ with } \text{equip} = \text{Tour} \\ \text{infos} = \text{Données_Tour}$$
 $\oplus$ DAC

Capacité

$$F(\text{Run_Tour}(_, _) / \text{Status}(\text{Tour}) = \text{FAULT})$$

* *L'action run_Tour ne peut avoir lieu si le statut de la machine*

* *est à fault.*

• Contraintes de coopération

Perception des actions

K(Technicien.Erreur_Tour() / TRUE)

K(Technicien.Relance_Tour() / TRUE)

Perception de l'état

1

Informations sur les actions

1

Informations sur l'état

/

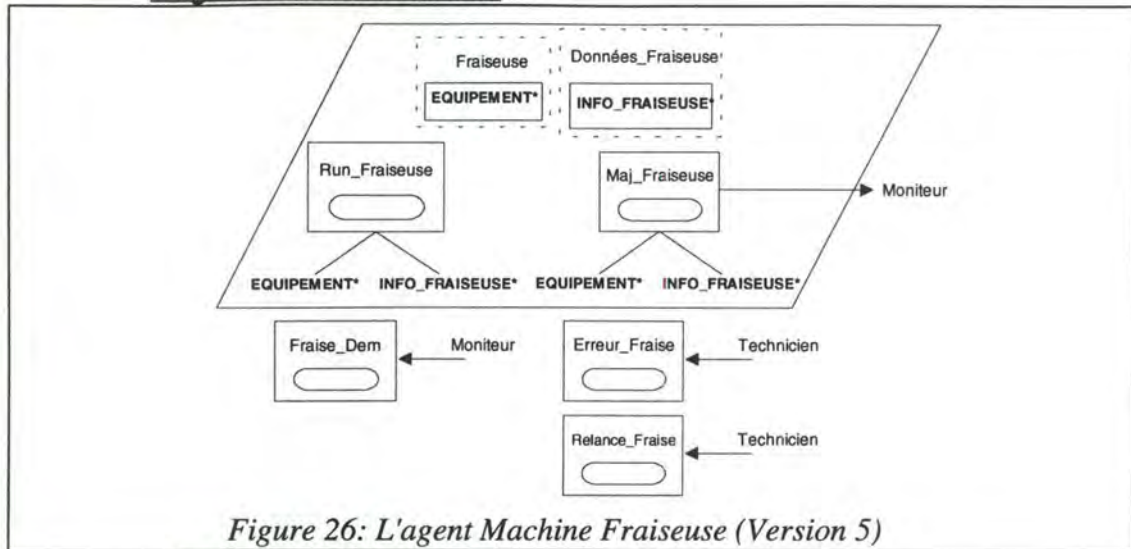
L'agent Machine-Fraiseuse

Figure 26: L'agent Machine Fraiseuse (Version 5)

• Contraintes de Base

Etat initial

/

Composants dérivés

/

• Contraintes locales

Comportement de l'état

/

Effets des actions

Technicien.Erreur_Fraise(): Status(Fraise) = FAULT

Technicien.Relance_Fraise(): Status(Fraise) = IDLE

Run_Fraiseuse(a,b): Tour = a

Données_Fraiseuse = b

Causalité

$$\text{Moniteur.Fraise_Dem} \stackrel{0 \leq 1s}{\rightarrow} \text{Maj_Fraise}(\text{equip}, \text{infos}) \text{ with } \text{equip} = \text{Fraiseuse} \\ \text{infos} = \text{Données_Fraiseuse}$$
 $\oplus$ DACCapacité

F (Run_Fraiseuse(.,.) / Status(Fraiseuse) = FAULT)

* L'action run_Fraiseuse ne peut avoir lieu si le statut de la

* machine est à fault.

• Contraintes de coopération

Perception des actions

K(Technicien.Erreur_Fraise() / TRUE)

K(Technicien.Relance_Fraise() / TRUE)

Perception de l'état

/

Informations sur les actions

/

Informations sur l'état

/

6) Version 6 : Plus de communication directe entre le moniteur et les machines

a) Définitions des nouveaux⁵ types de données

TYPE_MES: {AGVTOMONI, TOURTOMONI, FRAITOMONI, BRIDTOMONI, ROBOTOMONI, TECHTOMONI, MONITOTOUR, MONITOAGV, MONITOBRI, MONITOROBO, MONITOFRAI}

CORPS_MES:

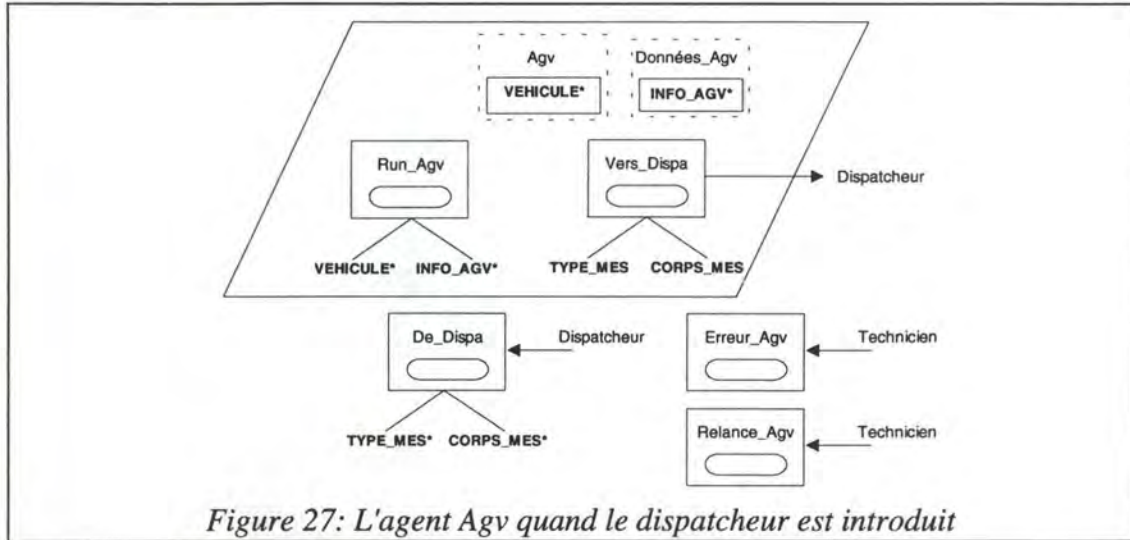
- ⊂ [CP: [PGM: PGM,
Pallet_Post: PALLET_POST,
Dock_State: DOCK_STATE,
POSXYZ: POSXYZ],
CP: [PGM: PGM,
Pallet_Post: PALLET_POST,
Dock_State: DOCK_STATE,
POSXYZ: POSXYZ],
CP: [Station: STATION,
Pallet_Post: PALLET_POST,
Dock_State: DOCK_STATE,
Distance: DISTANCE,
Speed: SPEED,
Direction: DIRECTION],
CP: [Brid_State: BRID_STATE,
Vitesse: VITESSE,
POSXY: POSXY],
CP: [PGM: PGM,
POSXYZ POSXYZ],
CP: [Machine: MACHINE]

MACHINE: {EN_ETAT, EN_PANNE}

⁵pour la définition des autres types, se reporter au point "a) Les types de données utilisées" du chapitre "3.1 Version 1: Le Moniteur voit tout - Les Machines montrent tout", au point "a) Les nouveaux types de données utilisées" du chapitre "3.4.1 Les machines envoient des informations incohérentes" et au point "a) Les nouveaux types de données utilisées" du chapitre "3.5 Version 5 : Le Technicien"

b) Description des agents

L'agent Machine Agv



• Contraintes de Base

Etat initial

/

Composants dérivés

1

- **Contraintes locales**

Comportement de l'état

/

Effets des actions

Technicien.Erreur_Agv(): Status(Agv) = FAULT

Technicien.Relance_Agv(): Status(Agv) = IDLE

Run_Agv(a,b): Agv = a
 Données_Agv = b

Causalité

$$\text{Dispatcheur.De_Dispa}^{\Diamond \leq 1s} \rightarrow \text{Vers_Dispa}(\text{type}, \text{corps}) \text{ with } \text{type} = \text{AGVTOMONI}$$

$$\text{corps} = \langle \text{Agv}, \text{Données_Agv} \rangle$$
 $\oplus$ DAC

Capacité

$$F(\text{Run_Agv}(_,_) / \text{Status}(\text{Agv}) = \text{FAULT})$$

* L'action `run_Agv` ne peut avoir lieu si le statut de la machine

* *est à FAULT.*

- **Contraintes de coopération**

Perception des actions

K(Technicien.Erreur_Agv() / TRUE)

K(Technicien.Relance_Agv() / TRUE)

Perception de l'état

1

Informations sur les actions

/

Informations sur l'état

1

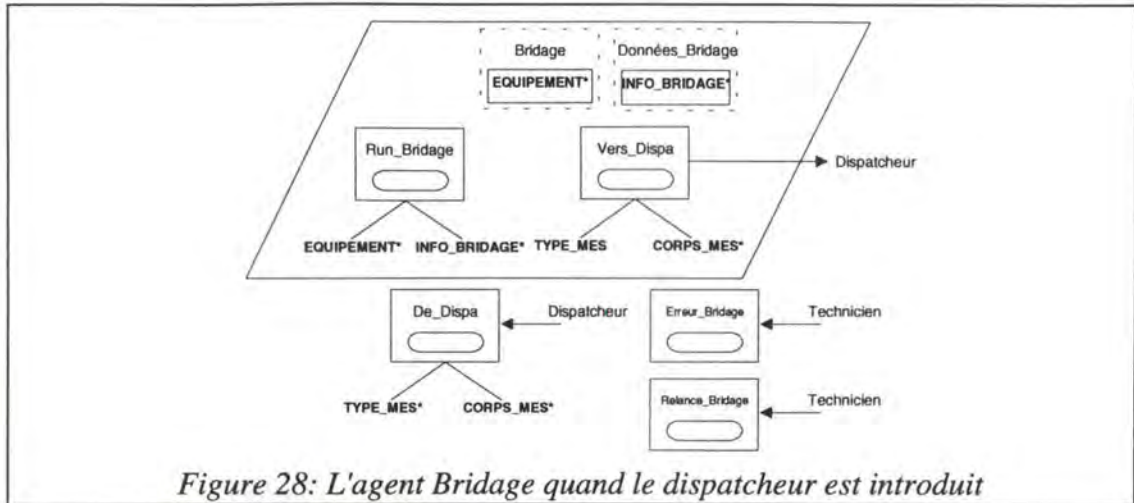
L'agent Machine-Bridge

Figure 28: L'agent Bridge quand le dispatcheur est introduit

• Contraintes de Base

Etat initial

/

Composants dérivés

/

• Contraintes locales

Comportement de l'état

/

Effets des actions

Technicien.Erreur_Bridge(): Status(Bridge) = FAULT

Technicien.Relance_Bridge(): Status(Bridge) = IDLE

Run_Bridge(a,b): Bridge = a
Données_Bridge = bCausalitéDispatcheur.De_Dispa() $\hat{0} \leq 1s \rightarrow$ Vers_Dispa(type,corps) with type = BRIDTOMONI
corps = <Bridge, Données_Bridge> $\oplus$ DACCapacité

F (Run_Bridge(.,.) / Status(Bridge) = FAULT)

* L'action run_Bridge ne peut avoir lieu si le statut de la machine

* est à fault.

• Contraintes de coopération

Perception des actions

K(Technicien.Erreur_Bridge() / TRUE)

K(Technicien.Relance_Bridge() / TRUE)

Perception de l'état

/

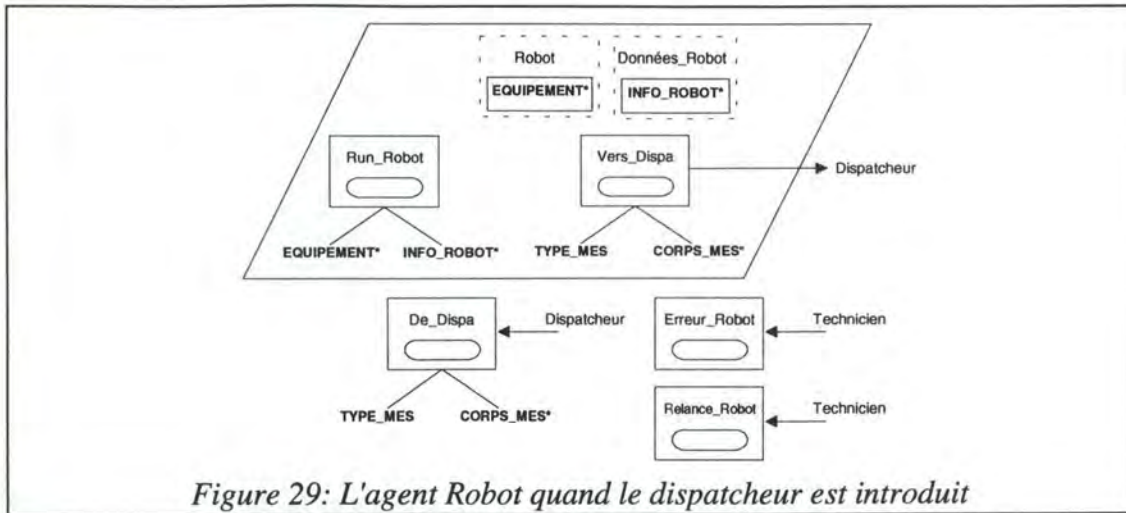
Informations sur les actions

/

Informations sur l'état

/

L'agent Machine-Robot



• Contraintes de Base

Etat initial

1

Composants dérivés

/

- **Contraintes locales**

Comportement de l'état

/

Effets des actions

Technicien.Erreur_Robot(): Status(Robot) = FAULT

```
Technicien.Relance_Robot(): Status(Robot) = IDLE
```

```
Run_Robot(a,b):  Robot = a
                  Données_Robot = b
```

Causalité

Dispatcheur.De_Dispa() $\hat{\leq} 1s \rightarrow$ Vers_Dispa(type,corps) with type = ROBOTOMONI
corps = <Robot, Données_Robot>

 $\oplus$ DAC

Capacité

$$F(\text{Run_Robot}(_, _) / \text{Status}(\text{Robot}) = \text{FAULT})$$

* *L'action run_Robot ne peut avoir lieu si le statut de la machine*
 * *est à fault.*

- **Contraintes de coopération**

Perception des actions

K(Technicien.Erreur_Robot / TRUE)

K(Technicien.Relance_Robot / TRUE)

Perception de l'état

A

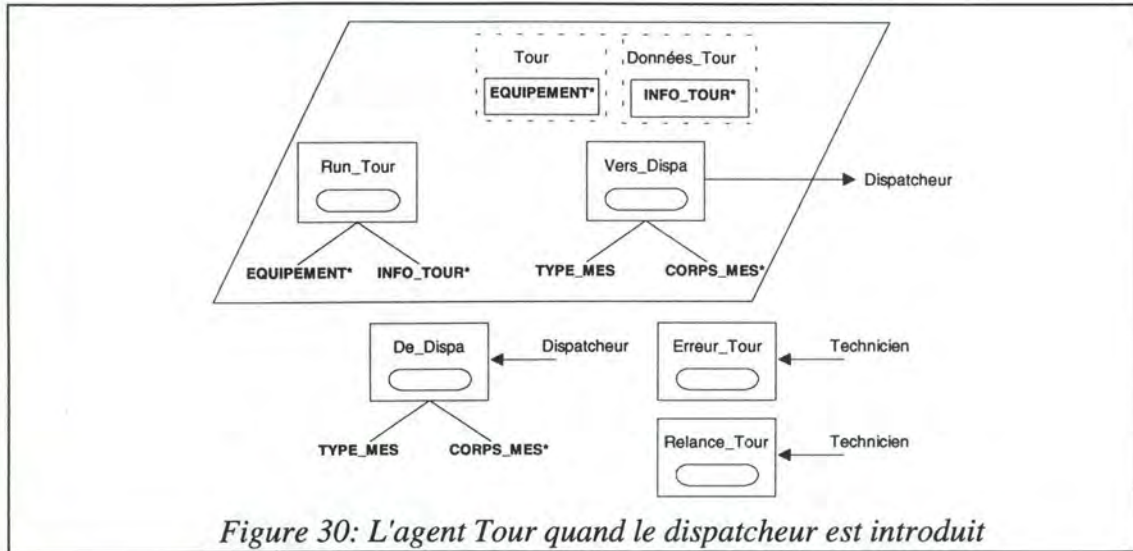
Informations sur les actions

L

Informations sur l'état

1

L'agent Machine-Tour



• Contraintes de Base

Etat initial

/

Composants dérivés

1

• Contraintes locales

Comportement de l'état

/

Effets des actions

Technicien.Erreur_Tour(): Status(Tour) = FAULT

Technicien.Relance_Tour(): Status(Tour) = IDLE

```
Run_Tour(a,b):   Tour = a
                 Données_Tour = b
```

Causalité

Dispatcheur.De_Dispa() $\delta \leq 1s \rightarrow$ Vers_Dispa(type,corps) with type = TOURTOMONI
corps = <Tour, Données_Tour>

 $\oplus$ DAC

Capacité

$$F(\text{Run_Tour}(_, _) / \text{Status}(\text{Tour}) = \text{FAULT})$$

* *L'action run_Tour ne peut avoir lieu si le statut de la machine*
 * *est à fault.*

- **Contraintes de coopération**

Perception des actions

K(Technicien.Erreur_Tour() / TRUE)

K(Technicien.Relance_Tour() / TRUE)

Perception de l'état

/

Informations sur les actions

/

Informations sur l'état

L

L'agent Machine-Fraiseuse

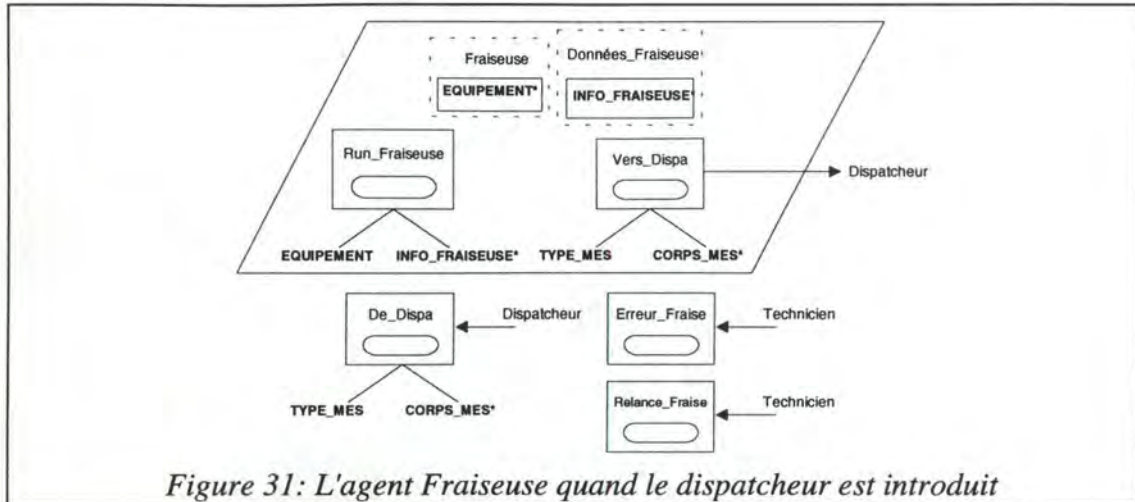


Figure 31: L'agent Fraiseuse quand le dispatcheur est introduit

- **Contraintes de Base**

- Etat initial

/

- Composants dérivés

/

- **Contraintes locales**

- Comportement de l'état

/

- Effets des actions

Technicien.Erreur_Fraise(): Status(Fraiseuse) = FAULT

Technicien.Relance_Fraise(): Status(Fraiseuse) = IDLE

Run_Fraiseuse(a,b): Fraiseuse = a

Données_Fraiseuse = b

- Causalité

Dispatcheur.De_Dispa() $\hat{\diamond} \leq 1s \rightarrow$ Vers_Dispa(type,corps) with type = FRAITOMONI
corps = <Fraiseuse, Données_Fraiseuse>

$\oplus$ DAC

- Capacité

F (Run_Fraiseuse(.,.) / Status(Fraiseuse) = FAULT)

* L'action run_Fraiseuse ne peut avoir lieu si le statut de la

* machine est à fault.

- **Contraintes de coopération**

- Perception des actions

K(Technicien.Erreur_Fraise() / TRUE)

K(Technicien.Relance_Fraise() / TRUE)

- Perception de l'état

/

- Informations sur les actions

/

- Informations sur l'état

/

L'agent Technicien

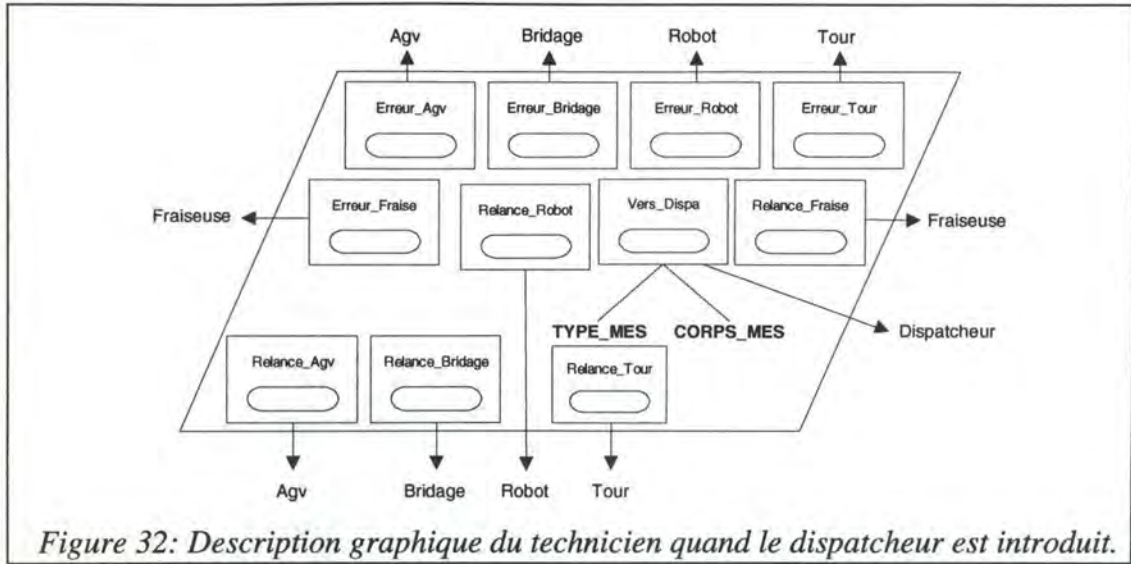


Figure 32: Description graphique du technicien quand le dispatcheur est introduit.

• Contraintes de Base

Etat initial

/

Composants dérivés

/

• Contraintes locales

Comportement de l'état

/

Effets des actions

/

Causalité

Erreur_Agv(); Vers_Dispa(type,corps)
 Erreur_Bridage(); Vers_Dispa(type,corps)
 Erreur_Robot(); Vers_Dispa(type,corps)
 Erreur_Tour(); Vers_Dispa(type,corps)
 Erreur_Fraise(); Vers_Dispa(type,corps)

- * Lorsque le technicien a détecté une panne d'une machine et la met hors service, il en informe juste après le moniteur au moyen d'un message qui transitera par le dispatcheur.

Relance_Agv(); Vers_Dispa(type,corps)
 Relance_Bridage(); Vers_Dispa(type,corps)
 Relance_Robot(); Vers_Dispa(type,corps)
 Relance_Tour(); Vers_Dispa(type,corps)
 Relance_Fraise(); Vers_Dispa(type,corps)

- * Lorsque le technicien a réparé une machine et la remise en service, il en informe juste après le moniteur au moyen d'un message qui transitera par le dispatcheur.

Capacité

F(Vers_Dispa(type,corps) / (type ≠ TECHTOMONI)
 ∨ ((Machine(corps) ≠ AGV)
 ∧ (Machine(corps) ≠ BRIDAGE)
 ∧ (Machine(corps) ≠ TOUR)

$\wedge(\text{Machine}(\text{corps}) \neq \text{FRAISEUSE})$
 $\wedge(\text{Machine}(\text{corps}) \neq \text{ROBOT})$
 $\vee ((\text{Etat}(\text{corps}) \neq \text{EN_PANNE})$
 $\wedge(\text{Etat}(\text{corps}) \neq \text{EN_ETAT}))$

- * *L'action Vers_Dispa(type, corps) ne peut avoir lieu avec d'autres valeurs que celle correspondant aux machines.*

• **Contraintes de coopération**

Perception des actions

/

Perception de l'état

/

Informations sur les actions

K (Vers_Dispa(type, corps).Dispatcheur / TRUE)

- * *Lorsque le technicien détecte une panne de machine ou la relance, le moniteur en est informé.*

K(Erreur_Agv().Agv / TRUE)
 K(Erreur_Robot().Robot / TRUE)
 K(Erreur_Tour().Tour / TRUE)
 K(Erreur_Fraise().Fraiseuse / TRUE)
 K(Erreur_Bridge().Bridge / TRUE)
 K(Relance_Agv().Agv / TRUE)
 K(Relance_Robot().Robot / TRUE)
 K(Relance_Tour().Tour / TRUE)
 K(Relance_Fraise().Fraiseuse / TRUE)
 K(Relance_Bridge().Bridge / TRUE)

- * *Le technicien informe toujours les machines lorsqu'il émet une action à leur intention.*

Informations sur l'état

/

L'agent Dispatcheur

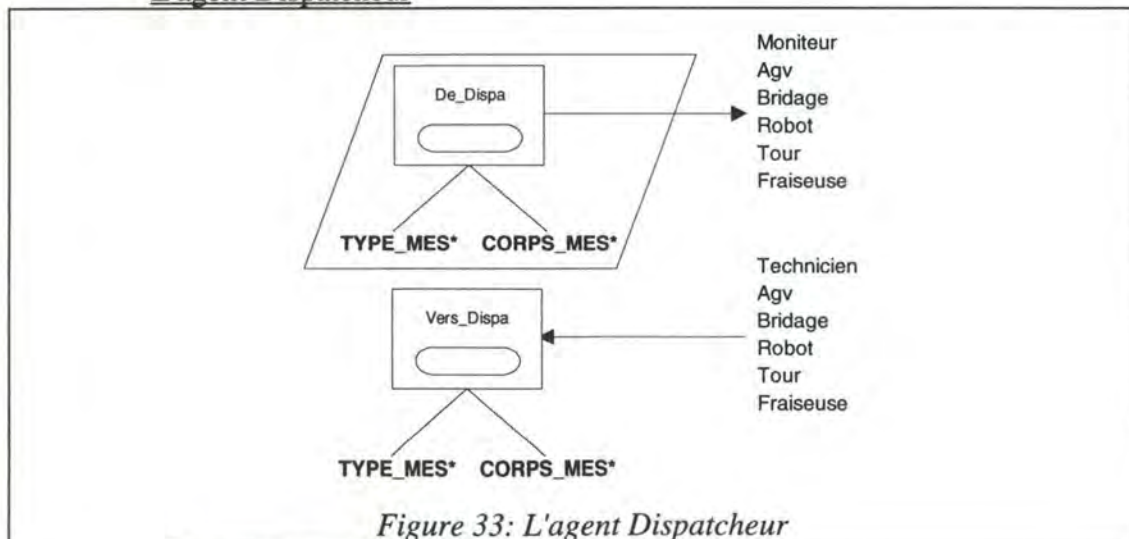


Figure 33: L'agent Dispatcheur

• **Contraintes de Base**

Etat initial

/

Composants dérivés

/

- Contraintes locales

- Comportement de l'état

- /

- Effets des actions

- /

- Causalité

Agv.Vers_Dispa(type,corps) $\hat{0} \leq 0.01s \rightarrow De_Dispa(type,corps)$

Tour.Vers_Dispa(type,corps) $\hat{0} \leq 0.01s \rightarrow De_Dispa(type,corps)$

Fraiseuse.Vers_Dispa(type,corps) $\hat{0} \leq 0.01s \rightarrow De_Dispa(type,corps)$

Bridage.Vers_Dispa(type,corps) $\hat{0} \leq 0.01s \rightarrow De_Dispa(type,corps)$

Robot.Vers_Dispa(type,corps) $\hat{0} \leq 0.01s \rightarrow De_Dispa(type,corps)$

Moniteur.Vers_Dispa(type,corps) $\hat{0} \leq 0.01s \rightarrow De_Dispa(type,corps)$

Technicien.Vers_Dispa(type,corps) $\hat{0} \leq 0.01s \rightarrow De_Dispa(type,corps)$

** Lorsqu'un message arrive d'un agent au dispatcheur, il doit*

** immédiatement le faire suivre.*

- Capacité

- Contraintes de coopération

- Perception des actions

K(_).Vers_Dispa(type,corps) / TRUE)

** Lorsqu'un message est rendu visible par un agent, le dispatcheur*

** doit en tenir compte car il est fiable.*

- Perception de l'état

- /

- Informations sur les actions

XK(De_Dispa(type,corps).Agv / type = MONITOAGV)

XK(De_Dispa(type,corps).Tour / type = MONITOTOUR)

XK(De_Dispa(type,corps).Fraise / type = MONITOFRAI)

XK(De_Dispa(type,corps).Bridage / type = MONITOBRI)

XK(De_Dispa(type,corps).Robot / type = MONITOROBO)

XK(De_Dispa(type,corps).Moniteur / ((type = AGVTOMONI)

$\vee$ (type = TOURTOMONI)

$\vee$ (type = ROBOTOMONI)

$\vee$ (type = FRAITOMONI)

$\vee$ (type = BRIDTOMONI)

$\vee$ (type = TECHTOMONI)))

** Un message ne peut être visible qu'à son destinataire uniquement*

- Informations sur l'état

- /

L'agent Moniteur

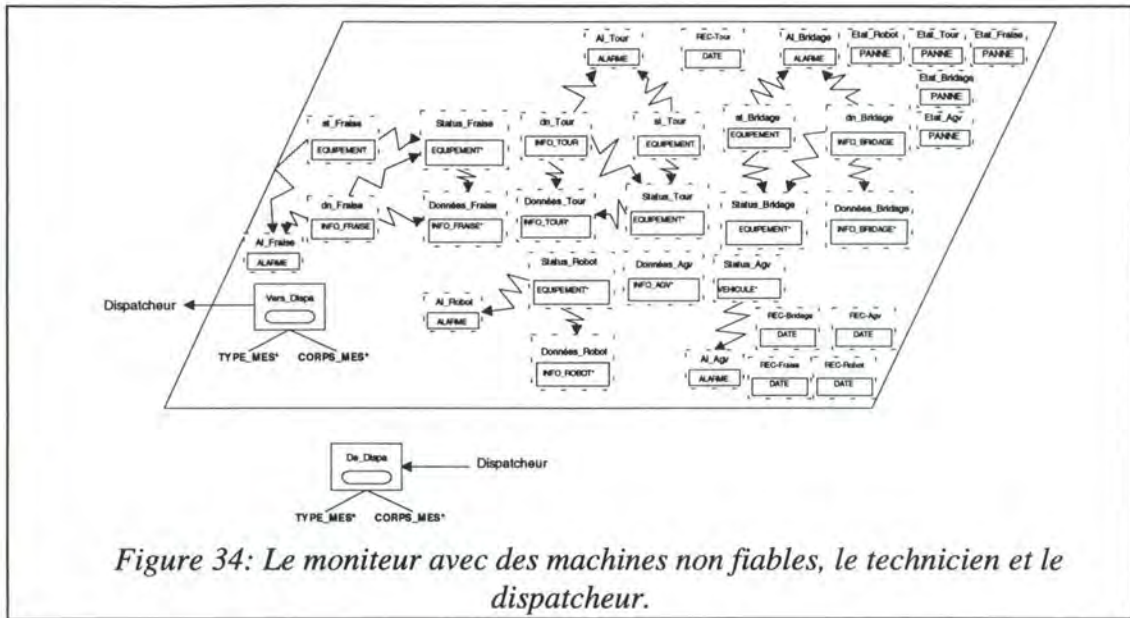


Figure 34: Le moniteur avec des machines non fiables, le technicien et le dispatcheur.

• Contraintes de Base

Etat initial

Status_Fraise = UNDEF
 Status_Robot = UNDEF
 Status_Tour = UNDEF
 Status_Agv = UNDEF
 Status_Bridage = UNDEF
 Données_Fraise = UNDEF
 Données_Robot = UNDEF
 Données_Tour = UNDEF
 Données_Agv = UNDEF
 Données_Bridage = UNDEF

- * A l'état initial, les différentes données représentant les machines sont
- * à UNDEF.

Etat_Agv = EN_ETAT
 Etat_Tour = EN_ETAT
 Etat_Fraise = EN_ETAT
 Etat_Robot = EN_ETAT
 Etat_Bridage = EN_ETAT

- * A l'état initial, toutes les machines sont considérées comme étant en
- * état de marche.

Composants dérivés

Status_Fraise = IDLE $\equiv$ (st_Fraise = IDLE) $\wedge$ (Dock_State(dn_Fraise) = STOPPED)

- * Si le status de la machine est IDLE et le Dock_State est STOPPED,
- * alors le status IDLE est valide pour le monitoring.

Status_Fraise = FAULT $\equiv$ (Dock_State(dn_Fraise) = UNKNOWN)
 $\vee$ ((st_Fraise = IDLE) $\wedge$ (Dock_State(dn_Fraise) $\neq$ STOPPED))
 $\vee$ ((st_Fraise = BUSY) $\wedge$ ((Dock_State(dn_Fraise) = STOPPED)
 $\vee$ (Dock_State(dn_Fraise)=UNKNOWN)))

- * Si le Dock_State de la machine est UNKNOWN, ou que le statut de la
- * machine est IDLE avec un Dock_State différent de STOPPED, ou que le

- * *statut de la machine est BUSY avec un Doc_State à STOPPED ou*
- * *UNKNOWN, alors le seul status valable pour le monitoring est FAULT*

Status_Fraise = BUSY $\equiv$ (st_Fraise = BUSY)
 $\wedge$ (Dock_State(dn_Fraise) $\neq$ STOPPED)
 $\wedge$ (Dock_State(dn_Fraise) $\neq$ UNKNOWN)
 * *Si le Dock_State de la machine est différent de STOPPED et de*
 * *UNKNOWN et le status est BUSY alors il est correct et le status pour*
 * *le monitoring est aussi BUSY.*

Dock_State(Données_Fraise) = STOPPED $\equiv$ (Status_Fraise = IDLE)
 * *Si le status de la fraiseuse pour le monitoring est IDLE, alors le*
 * *Dock_State est STOPPED.*

PGM(Données_Fraise) = UNDEF $\equiv$ (Status_Fraise = IDLE)
 Pallet_Post(Données_Fraise) = UNDEF $\equiv$ (Status_Fraise = IDLE)
 POSXYZ(Données_Fraise) = UNDEF $\equiv$ (Status_Fraise = IDLE)
 * *Si le status de la fraiseuse pour le monitoring est IDLE, alors toutes*
 * *les données autres que le Dock_State et le status sont non*
 * *pertinentes.*

Dock_State(Données_Fraise) = Dock_State(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)
 PGM(Données_Fraise) = PGM(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)
 Pallet_Post(Données_Fraise) = Pallet_Post(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)
 POSXYZ(Données_Fraise) = POSXYZ(dn_Fraise) $\equiv$ (Status_Fraise = BUSY)
 * *Si le status de la fraiseuse pour le monitoring est BUSY, alors toutes*
 * *les données sont pertinentes.*

Données_Fraise = UNDEF $\equiv$ (Status_Fraise = FAULT)
 * *Si le status de la fraiseuse pour le monitoring est FAULT, alors aucune*
 * *donnée concernant le tour n'est valide du point de vue du monitoring.*

Status_Tour = IDLE $\equiv$ (st_Tour = IDLE) $\wedge$ (Dock_State(dn_Tour) = STOPPED)
 * *Si le status de la machine est IDLE et le Dock_State est STOPPED,*
 * *alors le status IDLE est valide pour le monitoring.*

Status_Tour = FAULT $\equiv$ (Dock_State(dn_Tour) = UNKNOWN)
 $\vee$ ((st_Fraise = IDLE) $\wedge$ (Dock_State(dn_Tour) $\neq$ STOPPED))
 $\vee$ ((st_Fraise = BUSY) $\wedge$ ((Dock_State(dn_Tour) = STOPPED)
 $\vee$ (Dock_State(dn_Tour) = UNKNOWN)))
 * *Si le Dock_State de la machine est UNKNOWN, ou que le statut de la*
 * *machine est IDLE avec un Dock_State différent de STOPPED, ou que le*
 * *statut de la machine est BUSY avec un Doc_State à STOPPED ou*
 * *UNKNOWN, alors le seul status valable pour le monitoring est FAULT*

Status_Tour = BUSY $\equiv$ (st_Tour = BUSY)
 $\wedge$ (Dock_State(dn_Tour) $\neq$ STOPPED)
 $\wedge$ (Dock_State(dn_Tour) $\neq$ UNKNOWN)
 * *Si le Dock_State de la machine est différent de STOPPED et de*
 * *UNKNOWN et le status est BUSY alors il est correct et le status pour*
 * *le monitoring est aussi BUSY.*

Dock_State(Données_Tour) = STOPPED $\equiv$ (Status_Tour = IDLE)
 * *Si le status du tour pour le monitoring est IDLE, alors le*

*** Dock_State est STOPPED.**

PGM(Données_Tour) = UNDEF $\equiv$ (Status_Tour = IDLE)

Pallet_Post(Données_Tour) = UNDEF $\equiv$ (Status_Tour = IDLE)

POSXYZ(Données_Tour) = UNDEF $\equiv$ (Status_Tour = IDLE)

- * Si le status du tour pour le monitoring est IDLE, alors toutes**
- * les données autres que le Dock_State et le status sont non**
- * pertinentes.**

Dock_State(Données_Tour) = Dock_State(dn_Tour) $\equiv$ (Status_Tour = BUSY)

PGM(Données_Tour) = PGM(dn_Tour) $\equiv$ (Status_Tour = BUSY)

Pallet_Post(Données_Tour) = Pallet_Post(dn_Tour) $\equiv$ (Status_Tour = BUSY)

POSXYZ(Données_Tour) = POSXYZ(dn_Tour) $\equiv$ (Status_Tour = BUSY)

- * Si le status du tour pour le monitoring est BUSY, alors toutes**
- * les données sont pertinentes.**

Données_Tour = UNDEF $\equiv$ (Status_Tour = FAULT)

- * Si le status du tour pour le monitoring est FAULT, alors aucune donnée**
- * concernant le tour n'est valide du point de vue du monitoring.**

Status_Bridage = BUSY $\equiv$ (vitesse(dn_Bridage) $\neq$ 0)

- * Si la vitesse de bridage est différente de 0, alors le status doit être**
- * BUSY pour le monitoring.**

Status_Bridage = st_Bridage $\equiv$ (vitesse(dn_Bridage) = 0)

- * Si la vitesse de bridage vaut 0, alors le status du bridage correspond**
- * à ce qu'il doit être pour le monitoring.**

Données_Bridage = dn_Bridage $\equiv$ (Status_Bridage $\neq$ FAULT)

Données_Bridage = UNDEF $\equiv$ (Status_Bridage = FAULT)

- * Les autres données que le statut ne sont pertinentes que si le statut**
- * n'est pas à FAULT.**

Al_Fraise = TRUE $\equiv$ (st_Fraise = FAULT)

$\vee((\text{st_Fraise} = \text{IDLE}) \wedge (\text{Dock_State}(\text{dn_Fraise}) \neq \text{STOPPED}))$
 $\vee((\text{st_Fraise} \neq \text{FAULT}) \wedge (\text{Dock_State}(\text{dn_Fraise}) = \text{UNKNOWN}))$
 $\vee((\text{st_Fraise} \neq \text{BUSY}) \wedge ((\text{Dock_State}(\text{dn_Fraise}) = \text{UNKNOWN})$
 $\vee(\text{Dock_State}(\text{dn_Fraise}) = \text{STOPPED})))$
 $\vee(\text{TIME}() - \text{REC_Fraise} > 10\text{s}))$

- * L'alarme de la fraiseuse est à TRUE (on a des données incohérentes ou**
- * FAULT)**
- * si soit - le statut est IDLE et le Dock_State est différent de STOPPED**
- * - le statut est différent de FAULT bien que le Dock_State soit**
- * UNKNOWN**
- * - le statut n'est pas BUSY bien que le Dock_State soit STOPPED**
- * ou UNKNOWN**
- * - la fraiseuse n'a plus répondu depuis plus de 10 secondes.**

Al_Fraise = FALSE $\equiv$ NOT ((st_Fraise = FAULT)

$\vee((\text{st_Fraise} = \text{IDLE}) \wedge (\text{Dock_State}(\text{dn_Fraise}) \neq \text{STOPPED}))$
 $\vee((\text{st_Fraise} \neq \text{FAULT}) \wedge (\text{Dock_State}(\text{dn_Fraise}) = \text{UNKNOWN}))$
 $\vee((\text{st_Fraise} \neq \text{BUSY}) \wedge ((\text{Dock_State}(\text{dn_Fraise}) = \text{UNKNOWN})$
 $\vee(\text{Dock_State}(\text{dn_Fraise}) = \text{STOPPED})))$
 $\vee(\text{TIME}() - \text{REC_Fraise} > 10\text{s}))$

- * *L'alarme de la fraiseuse est à FALSE (on a des données cohérentes)*
- * *dans tous les autres cas.*

$Al_Tour = TRUE \equiv (st_Tour = FAULT)$
 $\vee((st_Tour = IDLE) \wedge (Dock_State(dn_Tour) \neq STOPPED))$
 $\vee((st_Tour \neq FAULT) \wedge (Dock_State(dn_Tour) = UNKNOWN))$
 $\vee((st_Tour \neq BUSY) \wedge ((Dock_State(dn_Tour) = UNKNOWN)$
 $\vee(Dock_State(dn_Tour) = STOPPED)))$
 $\vee(TIME() - REC_Tour > 10s)$

- * *L'alarme du tour est à TRUE (on a des données incohérentes ou FAULT)*
- * *si soit - le status est à FAULT*
- * *- le statut est IDLE et le Dock_State est différent de STOPPED*
- * *- le statut est différent de FAULT bien que le Dock_State soit*
- * *UNKNOWN*
- * *- le statut n'est pas BUSY bien que le Dock_State doit STOPPED*
- * *ou UNKNOWN*
- * *- le tour n'a plus répondu depuis plus de 10 secondes*

$Al_Tour = TRUE \equiv NOT((st_Tour = FAULT)$
 $\vee((st_Tour = IDLE) \wedge (Dock_State(dn_Tour) \neq STOPPED))$
 $\vee((st_Tour \neq FAULT) \wedge (Dock_State(dn_Tour) = UNKNOWN))$
 $\vee((st_Tour \neq BUSY) \wedge ((Dock_State(dn_Tour) = UNKNOWN)$
 $\vee(Dock_State(dn_Tour) = STOPPED)))$
 $\vee(TIME() - REC_Tour > 10s))$

- * *L'alarme du tour est à FALSE (on a des données cohérentes) dans tous*
- * *les autres cas.*

$Al_Robot = TRUE \equiv ((Status_Robot = FAULT) \vee (TIME() - REC_Robot > 10s))$

$Al_Robot = FALSE \equiv NOT((Status_Robot = FAULT) \vee (TIME() - REC_Robot > 10s))$

$Al_Agv = TRUE \equiv ((Status_Agv = FAULT) \vee (TIME() - REC_Agv > 10s))$

$Al_Agv = FALSE \equiv NOT((Status_Agv = FAULT) \vee (TIME() - REC_Agv > 10s))$

$Al_Bridage = TRUE \equiv (st_Bridage = FAULT)$
 $\vee((st_Bridage \neq BUSY) \wedge (vitesse(dn_Bridage) \neq 0))$
 $\vee(TIME() - REC_Bridage > 10s)$

- * *L'alarme du bridage est à TRUE (on a des données incohérentes ou*
- * *FAULT) si soit - le statut est FAULT*
- * *- la vitesse de bridage est différente de 0 et le statut n'est*
- * *pas BUSY.*
- * *- le bridage n'a plus répondu depuis plus de 10 secondes*

$Al_Bridage = FALSE \equiv NOT((st_Bridage = FAULT)$
 $\vee((st_Bridage \neq BUSY) \wedge (vitesse(dn_Bridage) \neq 0))$
 $\vee(TIME() - REC_Bridage > 10s))$

- * *L'alarme du bridage est à FALSE (on a des données cohérentes) dans*
- * *tous les autres cas.*

• Contraintes locales

Comportement de l'état

/

Effets des actions

De_Dispatcheur(type,corps) with type = AGVTOMONI:

st_Agv = Status(corps)
dn_Agv = Données(corps)
REC_Agv = DATE()

De_Dispatcheur(type,corps) with type = TOURTOMONI:

st_Tour = Status(corps)
dn_Tour = Données(corps)
REC_Tour = DATE()

De_Dispatcheur(type,corps) with type = FRAITOMONI:

st_Fraise = Status(corps)
dn_Fraise = Données(corps)
REC_Fraise = DATE()

De_Dispatcheur(type,corps) with type = BRIDTOMONI

Status(corps) = FAULT:

Status_Bridage = Status(corps)
Données_Bridage = UNDEF
REC_Bridage = DATE()

De_Dispatcheur(type,corps) with type = BRIDTOMONI

Status(corps) ≠ FAULT:

Status_Bridage = Status(corps)
Données_Bridage = Données(corps)
REC_Bridage = DATE()

De_Dispatcheur(type,corps) with type = ROBOTOMONI

Status(corps) = FAULT:

Status_Robot = Status(corps)
Données_Robot = UNDEF
REC_Robot = DATE()

De_Dispatcheur(type,corps) with type = ROBOTOMONI

Status(corps) ≠ FAULT:

Status_Robot = Status(corps)
Données_Robot = Données(corps)
REC_Robot = DATE()

** Lorsque des données arrive d'une machine, l'agent moniteur met à jour*

** les données de la machine correspondante.*

De_Dispatcheur(type,corps) with type = TECHTOMONI

Machine(corps) = AGV
Etat(corps) = EN_PANNE:

Etat_Agv = EN_PANNE

De_Dispatcheur(type,corps) with type = TECHTOMONI

Machine(corps) = ROBOT
Etat(corps) = EN_PANNE:

Etat_Robot = EN_PANNE

De_Dispatcheur(type,corps) with type = TECHTOMONI

Machine(corps) = TOUR
Etat(corps) = EN_PANNE:

Etat_Tour = EN_PANNE

De_Dispatcheur(type,corps) with type = TECHTOMONI

Machine(corps) = FRAISEUSE
Etat(corps) = EN_PANNE:

Etat_Fraise = EN_PANNE

De_Dispatcheur(type,corps) with type = TECHTOMONI

Machine(corps) = BRIDAGE
Etat(corps) = EN_PANNE:

Etat_Bridage = EN_PANNE

** Lorsqu'un signal de panne est perçu par le moniteur, le moniteur en*

** tient compte pour la machine concernée.*

De_Dispatcheur(type,corps) with type = TECHTOMONI
 Machine(corps) = AGV
 Etat(corps) = EN_ETAT:
 Etat_Agv = EN_ETAT
 Status_Agv = UNDEF

De_Dispatcheur(type,corps) with type = TECHTOMONI
 Machine(corps) = ROBOT
 Etat(corps) = EN_ETAT:
 Etat_Robot = EN_ETAT
 Status_Robot = UNDEF

De_Dispatcheur(type,corps) with type = TECHTOMONI
 Machine(corps) = TOUR
 Etat(corps) = EN_ETAT:
 Etat_Tour = EN_ETAT
 Status_Tour = UNDEF

De_Dispatcheur(type,corps) with type = TECHTOMONI
 Machine(corps) = FRAISEUSE
 Etat(corps) = EN_ETAT:
 Etat_Fraise = EN_ETAT
 Status_Fraise = UNDEF

De_Dispatcheur(type,corps) with type = TECHTOMONI
 Machine(corps) = BRIDAGE
 Etat(corps) = EN_ETAT:
 Etat_Bridage = EN_ETAT
 Status_Bridage = UNDEF

** Lorsqu'un signal de remise en marche est perçu par le moniteur, le
 * moniteur en tient compte pour la machine concernée.*

Causalité

De_Dispa(TOURTOMONI,_) $\hat{O} \leq 5s \rightarrow$ Vers_Dispa(MONITOTOUR,_)
 De_Dispa(AGVTOMONI,_) $\hat{O} \leq 5s \rightarrow$ Vers_Dispa(MONITOAGV,_)
 De_Dispa(FRAITOMONI,_) $\hat{O} \leq 5s \rightarrow$ Vers_Dispa(MONITOFRAI,_)
 De_Dispa(ROBOTOMONI,_) $\hat{O} \leq 5s \rightarrow$ Vers_Dispa(MONITOROBO,_)
 De_Dispa(BRIDTOMONI,_) $\hat{O} \leq 5s \rightarrow$ Vers_Dispa(MONITOBRI,_)
** Lorsqu'une action de mise à jour des valeurs d'une machine a lieu,
 * dans les 5 secondes qui suivent, le monitoring doit faire une nouvelle
 * demande d'information sur la machine dont l'action de mise à jour a eu
 * lieu.*

Capacité

O(Vers_Dispa(MONITOFRAI,_) / ((Status_Fraise = UNDEF) $\vee$ (TIME() - REC_Fraise > 10s))
 $\wedge$ (Etat_Fraise = EN_ETAT))
** Si Status_Fraise est à UNDEF ou que la machine ne répond plus, alors
 * une action Fraise_Dem est générée.(cela signifie qu'on a aucune
 * données sur cette machine et qu'il faut donc en demander).
 * Il faut en plus que la machine soit en état de marche (qu'elle ne soit pas
 * signalée en panne par le technicien).*

O(Vers_Dispa(MONITOROBO,_) / ((Status_Robot = UNDEF) $\vee$ (TIME() - REC_Robot > 10s))
 $\wedge$ (Etat_Robot = EN_ETAT))
** Si Status_Robot est à UNDEF ou que la machine ne répond plus, alors
 * une action Robot_Dem est générée.(cela signifie qu'on a aucune
 * données sur cette machine et qu'il faut donc en demander)
 * Il faut en plus que la machine soit en état de marche (qu'elle ne soit pas
 * signalée en panne par le technicien).*

O(Vers_Dispa(MONITOTOUR,_) / ((Status_Tour = UNDEF) $\vee$ (TIME() - REC_Tour > 10s))
 $\wedge$ (Etat_Tour = EN_ETAT))

- * Si Status_Tour est à UNDEF ou que la machine ne répond plus, alors
- * une action Tour_Dem est générée.(cela signifie qu'on a aucune données
- * sur cette machine et qu'il faut donc en demander).
- * Il faut en plus que la machine soit en état de marche (qu'elle ne soit pas
- * signalée en panne par le technicien).

O(Vers_Dispa(MONITOAGV,_) / ((Status_Agv = UNDEF) $\vee$ (TIME() - REC_Agv > 10s))
 $\wedge$ (Etat_Agv = EN_ETAT))

- * Si Status_Agv est à UNDEF ou que la machine ne répond plus, alors
- * une action Agv_Dem est générée.(cela signifie qu'on a aucune données
- * sur cette machine et qu'il faut donc en demander).
- * Il faut en plus que la machine soit en état de marche (qu'elle ne soit pas
- * signalée en panne par le technicien).

O(Vers_Dispa(MONITOBID,_) / ((Status_Bridage=UNDEF) $\vee$ (TIME() - REC_Bridage>10s))
 $\wedge$ (Etat_Bridage = EN_ETAT))

- * Si Status_Bridage est à UNDEF ou que la machine ne répond plus, alors
- * une action Bridage_Dem est générée.(cela signifie qu'on a aucune
- * données sur cette machine et qu'il faut donc en demander).
- * Il faut en plus que la machine soit en état de marche (qu'elle ne soit pas
- * signalée en panne par le technicien).

• Contraintes de coopération

Perception des actions

K (Dispatcheur.De_Dispa(,_) / TRUE)

- * Lorsqu'un message arrive du dispatcheur, d'une machine, il doit
- * toujours être pris en compte.

Perception de l'état

/

Informations sur les actions

K (Vers_Dispa(,_.).Dispatcheur / TRUE)

- * Lorsqu'un message de demande d'info est générée par le moniteur, il
- * est rendu visible au dispatcheur.

Informations sur l'état

K(Manager.Status_Fraise / TRUE)

K(Manager.Status_Agv / TRUE)

K(Manager.Status_Bridage / TRUE)

K(Manager.Status_Tour / TRUE)

K(Manager.Status_Robot / TRUE)

K(Manager.Données_Fraise / TRUE)

K(Manager.Données_Agv / TRUE)

K(Manager.Données_Bridage / TRUE)

K(Manager.Données_Tour / TRUE)

K(Manager.Données_Robot / TRUE)

- * Le moniteur laisse toujours voir toutes ses données au manager

Annexes II: Programmation C

Dans cette annexe, nous allons présenter les procédures C les plus importantes à l'implémentation de l'application. C'est-à-dire les routines RPC du routeur et des serveurs ainsi que les clients associés. Enfin, nous montrerons les routines de manipulation de listes et de zones de mémoire partagées.

1) Le routeur et ses procédures

1.1) Protocol description file du ROUTEUR

```

/* Protocol description file du ROUTEUR*/
/* Structure utilisée pour le protocole de communication du monitoring */

typedef string str<20>;
typedef int entiers<10>;
typedef str caracts<10>;

struct response
{
    string message<15>;
};

struct MESSAGE_CIM
{
    string typ_dispatch<3>;
    int type_service;
    entiers Entiers;
    caracts Caracts;
    string destinataire<20>;
};

program PROGCIM {
    version PROGCIMVERS {
        response procrouter(MESSAGE_CIM) = 1;
        response Boot(void) = 2;
    } = 1;

} = 60;

```

1.2) Les procédures du ROUTEUR

```

#include "incCIM.h"
#include "cimdef.h" /* Shared memory zones definition */
#include "cimtr.h"

/* Routing procedure */
response *procrouter_1(params)
    MESSAGE_CIM *params;
{
    static response result;
    int idshmtab; /* Shared memory ID */
    int idsemtab; /* Semaphore ID */
    int idshmclt;
    int idsemclt;
    int i;
    char *adresse;
    SHMTAB_ZONE *tab_datas, local_tab;
    SHMCLT_ZONE *clt_datas;
    struct sembuf sb;
    char *stream;
    OBL_INT *point_int;
    OBL_STRING *point_str;
    int *temp_int;
    str temp_str;
    /******
    /* Create shared memory zones */
    /******
    idshmtab = shmget(KEYSHMTAB, sizeof(*tab_datas), 0777);
    idshmclt = shmget(KEYSHMCLT, sizeof(*clt_datas), 0777);
    /* Create 1 Semaphore to control access to shared memory zone */ idsemtab = semget(KEYSEMTAB, 1, 0777);

```



```

idsemclt = semget(KEYSEMCLT,1,0777);
/* Attach shared memory zones to associated buffer */
tab_datas = (SHMTAB_ZONE *) shmat(idshmtab,(char *)0,0);
clt_datas = (SHMCLT_ZONE *) shmat(idshmclt,(char *)0,0);
/*****
/* Params treatment */
/*****
if (strcmp(params->destinataire, "ROUTEUR_RPC")) /* Append router's table */
{ /* Get semaphore */
    sb.sem_num = 0;
    sb.sem_op = -1;
    sb.sem_flg = 0;
    semop(idsemtab,&sb,1);

/* Append router's table */
strcpy(tab_datas->obj_receiver[tab_datas->long_tab], *(params->Caracts.caracts_val));
strcpy(tab_datas->host_receiver[tab_datas->long_tab], *(params->Caracts.caracts_val));
tab_datas->long_tab++;
/* Release semaphore */
    sb.sem_num = 0;
    sb.sem_op = 1;
    sb.sem_flg = 0;
    semop(idsemtab,&sb,1);

/* Detach Shared Memory Zones */
    shmdt(tab_datas);
    shmdt(clt_datas);

/* Router's table is up to date */
    result.message = "OK";
    return(&result);
}

/* Value for "ATELIER" */
if (strcmp(params->destinataire, "ATELIER"))
{
    stream = fopen("execution.txt", "w");
    /* Copy datas to file */
    temp_str = &(params->Caracts);
    point_str = *(temp_str + 1);
    fputs(*point_str, stream);
    fclose(stream);
    result.message = "OK";
    return(&result);
}

/* Search router's table for the receiver */
/* Get semaphore */
sb.sem_num = 0;
sb.sem_op = -1;
sb.sem_flg = 0;
semop(idsemtab,&sb,1);
/* Copy zone to local table */
memcpy(&local_tab, tab_datas, sizeof(*tab_datas));
/* Release semaphore */
sb.sem_num = 0;
sb.sem_op = 1;
sb.sem_flg = 0;
semop(idsemtab,&sb,1);
i = 0;
strcpy(adresse, "NULL");
while ((i < local_tab.long_tab) && (strcmp(adresse, "NULL")))
{
    if (local_tab.obj_receiver[local_tab.long_tab] == params->destinataire)
        strcpy(adresse, params->destinataire);
    i++;
}

/* Detach Shared Memory Zone */
shmdt(tab_datas);
if (adresse != "NULL")
/*****
/* Receiver found */
/*****
{
/* Get semaphore */
    sb.sem_num = 0;
    sb.sem_op = -1;
    sb.sem_flg = 0;
    semop(idsemclt,&sb,1);

```

```

/* Copy params to clt_datas + host_receiver */
/* Copy dispatch */
strcpy(&(clt_datas->dispatch_type),params->typ_dispatch);
/* Copy Service type */
clt_datas->service_type = params->type_service;
/* Copy list int */
clt_datas->long_tint = params->Entiers.entiers_len;
temp_int = params->Entiers.entiers_val;
for (i = 0; i < params->Entiers.entiers_len; i++)
{
    point_int = *(temp_int + i);
    clt_datas->tint[i] = *point_int;
}
/* Copy list string */
clt_datas->long_tstr = params->Caracts.caracts_len; temp_str = *(params->Caracts.caracts_val);
for (i = 0; i < params->Caracts.caracts_len; i++)
{
    point_str = *(temp_str + i);
    strcpy(clt_datas->tstr[i], *point_str);
}
/* Copy receiver */
strcpy(clt_datas->receiver, params->destinataire);
/* Copy Host_receiver */
strcpy(clt_datas->host_receiver, adresse);
/* Release semaphore */
sb.sem_num = 0;
sb.sem_op = 1;
sb.sem_flg = 0;
semop(idsemclt,&sb,1);
/* Detach Shared Memory Zone */
shmdt(clt_datas);
/* Launch client process to propagate message */ system("/users/crp/degrood/OBLOG/Arthur/propclt/propclt &");
result.message = "OK";
return(&result);
}
/* Detach Shared Memory Zone */
shmdt(clt_datas);
result.message = "ERROR"; return(&result);
}

/*****/
/* Boot sequence */
/*****/
response *
boot_1(void)
{
    static response result;
    int idshmtab; /* Shared memory ID */
    int idsemtab; /* Semaphore ID */
    int idshmclt;
    int idsemclt;
    SHMTAB_ZONE *tab_datas,local_tab;
    SHMCLT_ZONE *clt_datas;
    struct sembuf sb;
    /*****/
    /* Create and init shared memory zones */
    /*****/
    /* Create Shared Memory zones */
    idshmtab = shmget(KEYSHMTAB,sizeof(*tab_datas),0777|IPC_CREAT);
    idshmclt = shmget(KEYSHMCLT,sizeof(*clt_datas),0777|IPC_CREAT);
    /* Create 1 Semaphore to control access to shared memory zone */
    idsemtab = semget(KEYSEMTAB,1,0777|IPC_CREAT);
    idsemclt = semget(KEYSEMCLT,1,0777|IPC_CREAT);
    /* Attach shared memory zones to associated buffer */
    tab_datas = (SHMTAB_ZONE *) shmat(idshmtab,(char *)0,0);
    clt_datas = (SHMCLT_ZONE *) shmat(idshmclt,(char *)0,0);
    /* Init Shared memory zones */
    /* Put object "DIEU" as first element of the shared memory zone */ /* adress = "arthur" */
    strcpy(tab_datas->obj_receiver[0], "DIEU");
    strcpy(tab_datas->host_receiver[0], "arthur");
    tab_datas->long_tab = 1;
    /* Init semaphores */
    semctl(idsemtab,0,1);

```

```

semctl(idsemctl,0,1);
/* Detach Shared Memory Zones */
shmdt(tab_datas);
shmdt(clt_datas);
result.message = "OK";
return(&result);
}

```

2) Les serveurs et leurs procédures

2.1) Protocol Description File du serveur

```

/* Protocol description file */
/* Structure utilisee pour le protocole de communication du monitoring */
typedef string str<20>;
typedef int entiers<10>;
typedef str caracts<10>;
struct response
{
    string message<15>;
};
struct MESSAGE_CIM
{
    string typ_dispatch<3>;
    int type_service;
    entiers Entiers;
    caracts Caracts;
    string destinataire<20>;
};
program SRVCIM {
    version SRVCIMVERS {
        response procboot(void) = 1;
        response procInt(MESSAGE_CIM) = 2;
        MESSAGE_CIM get_ps_list(void) = 3;
    } = 1;
} = 50;

```

2.2) Les procédures du serveur

2.2.1) Initialisation du site local au serveur

```

/* Launch process on host and create & init shared memory zone(s) */

#include "incCIM.h"
#include "cimsrv.h"
#include "cimdef.h" /* Shared memory zone definition */
response *
procboot_1(void)
{
    static response result;
    MESSAGE_CIM *params;
    int idshm; /* Shared memory ID */
    int idsem; /* Semaphore ID */
    SHM_ZONE *shm_datas;
    struct sembuf sb;
    /* Create and init shared memory zone */
    /* Create Shared Memory zone */
    idshm = shmget(KEYSHM,sizeof(*params),0777|IPC_CREAT);
    /* Create 1 Semaphore to control access to shared memory zone */
    idsem = semget(KEYSEM,1,0777|IPC_CREAT);
    /* Attach shared memory zone to buffer */
    shm_datas = (SHM_ZONE *) shmat(idshm,(char *)0,0);
    /* Init zone */
    shm_datas->flag = 0; /* Shared memory zone accessible to write */
    /* Init Semaphore ( Shared memory zone accessible) */
}

```



```

semctl(idsem,0,1);
/* Launch the local oblog Process */
system("/users/crp/degrood/OBLOG/Arthur/PROCOBL &"); /* Process to launch in background */
result.message = "OK";
return (&result);
}

```

2.2.2) Procédure principale du serveur

```

response *
procclnt_1(params)
    MESSAGE_CIM *params;
{
    static response result;
    /* Receive a structure from a client and copy it to a shared memory zone */
    int idshm; /* Shared memory ID */
    int idsem; /* Semaphore ID */
    SHM_ZONE *shm_datas, local_datas;
    struct sembuf sb;
    int i; /* counter */
    OBL_INT *point_int;
    OBL_STRING *point_str;
    int *temp_int;
    str temp_str;
    /* Create Shared Memory zone */
    idshm = shmget(KEYSHM, sizeof(*shm_datas), 0777);
    /* Create 1 Semaphore to control access to shared memory zone */ idsem = semget(KEYSEM, 1, 0777);
    /* Attach shared memory zone to buffer */
    shm_datas = (SHM_ZONE *) shmat(idshm, (char *) 0, 0);
    /* Init local datas from params */
    local_datas.flag = 1;
    /* Copy dispatcher type */ strcpy(local_datas.dispatch_type, params->typ_dispatch);
    /* Copy destinataire */
    strcpy(local_datas.receiver, params->destinataire);
    /* Copy service type */
    local_datas.service_type = params->type_service;
    /* Copy int table */
    local_datas.long_tint = params->Entiers.entiers_len; temp_int = params->Entiers.entiers_val;
    for (i=0; i < local_datas.long_tint; i++) {
        point_int = *(temp_int + i); local_datas.tint[i] = *point_int;
    }
    /* Copy string table */
    local_datas.long_tstr = params->Caracts.caracts_len; temp_str = *(params->Caracts.caracts_val);
    for (i=0; i < local_datas.long_tint; i++)
    {
        point_str = *(temp_str + i); strcpy(local_datas.tstr[i], *point_str);
    }
    /* Get semaphore */
    attach: sb.sem_num = 0;
    sb.sem_op = -1;
    sb.sem_flg = 0;
    semop(idsem, &sb, 1);
    /* Copy local datas to shared memory zone */
    if (shm_datas->flag == 1) /* Have to wait until zone was readed by other process */ {
        /* Release semaphore */
        sb.sem_num = 0;
        sb.sem_op = 1; sb.sem_flg = 0; semop(idsem, &sb, 1); /* wait a time */
        for (i=0; i<20; i++); /* temporisation */ goto attach;
    }
    memcpy(shm_datas, &local_datas, sizeof(local_datas));
    /* Release semaphore */
    sb.sem_num = 0;
    sb.sem_op = 1;
    sb.sem_flg = 0;
    semop(idsem, &sb, 1);

    /* Detach Shared Memory Zone */ shmdt(shm_datas);
    result.message = "OK"; return(&result);
}
/*****

```

```

MESSAGE_CIM *
get_ps_list_1(void)
{
    FILE *ps_stream;
    char line[40], name[10], *ptr;
    char PS_PROC[5][10] = {"srv", "sh", "csh", "PROCOBL", ""};
    MESSAGE_CIM result;
    struct hostent *host_entry;
    char *host_name;
    int i, j;
    result.Caracts.caracts_len = 0;
    system("ps >pslist.txt");
    /* Create liste of 'str' */
    CreateStr(*(result.Caracts.caracts_val));
    i = 0;
    ps_stream = fopen("pslist.txt", "r"); while( fgets(line, 40, ps_stream) != NULL)
    { ptr = strchr(line, ' ');
      ptr++;
      for (j=0; j < 4; j++)
      { if (strcmp(ptr, PS_PROC[j], strlen(PS_PROC[j])))
        { AppendStr(*(result.Caracts.caracts_val), i, PS_PROC[j]); result.Caracts.caracts_len++;
          i++;
        }
      }
    }
    return(&result);
}

```

3) Les procédures des Clients

3.1) Le client du routeur

```

/* RPC client :*/
/* This client send OBL_DATAS to the RPC "router" located on the host ADR_ROUTER */
#include "incCIM.h"
#include "cimrtr.h" /* Created by rpcgen, prefix = same as .x */
#include "dispatching__ext.h" /* Oblog structure definition */
#include "cimdef.h" /* Shared memory zone structure definition */
#define NULL 0
response *clientRPC(OBL_DATAS)
    DISPATCHING_send_etrange__FOB *OBL_DATAS; /* datas given by oblog */
{
    /* intern types for client */
    register CLIENT *cl;
    struct hostent *serv_arg;
    struct sockaddr_in server;
    int sock = RPC_ANYSOCK;
    /* Others types */
    MESSAGE_CIM init;
    response *result;
    int i, j;
    int *point_init_i;
    int *point_obl_i;
    OBL_STRING *point_init_s;
    OBL_STRING *point_obl_s;
    FILE *stream;
    char *host_name;
    OBL_INT *point_obl_int;
    OBL_STRING *point_obl_str, point_snd_str;
    int *temp_obl_int;
    str temp_obl_str, temp_snd_str;
    char *ADR_ROUTER = "arthur";
    fprintf(stderr, "\n*****\n");
    fprintf(stderr, "*****Client_RPC*****\n");
    fprintf(stderr, "*****\n");
    serv_arg = gethostbyname(ADR_ROUTER);
    bcopy(serv_arg->h_addr, (caddr_t)&server.sin_addr, serv_arg->h_length);
    server.sin_family = AF_INET;
}

```

```

server.sin_port = 0;
/* Create client handle */
/* to use UDP in place of TCP, replace tcp by udp */
/* but TCP is more reliable than UDP */
cl = clnt_create(ADR_ROUTER,PROGCIM,PROGCIMVERS,"tcp");
if (cl == NULL) {
/* Couldn't establish connection with server */ result->message = "Connexion impossible"; return(&result);
}
/*****
/* Init structure to send from oblog structure */
*****/
/* Dispatcher type */ strcpy(init.typ_dispatch,OBL_DATAS->type_dispatch);
/* Receiver */ strcpy(init.destinataire,OBL_DATAS->destinataire);
/* Service type */
init.type_service = OBL_DATAS->type_service;
/* Int table */
init.Entiers.entiers_len = ENTIERSLength(OBL_DATAS->entiers);
temp_obl_int = OBL_DATAS->entiers;
/* Create int list */ CreateInt(*(init.Entiers.entiers_val));
for (i=1; i<=init.Entiers.entiers_len; i++)
{
j = i-1;
point_obl_int = *(temp_obl_int + i);
/* Append int list */
AppendInt(*(init.Entiers.entiers_val),j,*point_obl_int);
}
/* Strings table */
init.Caracts.caracts_len = CARACTSLength(OBL_DATAS->caracts);
temp_obl_str = OBL_DATAS->caracts;
/* Create str list */
CreateStr(*(init.Caracts.caracts_val));
for (i=1; i<=init.Caracts.caracts_len; i++)
{
j = i-1;
point_obl_str = *(temp_obl_int + i);
/* Append str list */ AppendStr(*(init.Caracts.caracts_val),j,*point_obl_str);
}
/* An object is born, so we need its hostname */
if ((strcmp(OBL_DATAS->destinataire,"ROUTEUR_RPC") == 0))
{ system("hostname > name.txt");
stream = fopen("name.txt", "r"); fgets(host_name,10,stream);
fclose(stream);
point_snd_str = *(temp_snd_str + init.Caracts.caracts_len); strcpy(*point_snd_str, host_name);
}
/* Call to remote procedure */
result = procrouter_1(&init,cl);
if (result == NULL) {
/* Error Call Back client */ result->message = "Erreur Rappel : TIMEOUT";
}
/* Call back from the server. Results are returned */
fprintf(stderr,"%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s\n"); return(result);
}

```

3.2) Le client du serveur

```

/* Send datas located in a shared memory zone to the specified host */
#include "incCIM.h"
#include "cimsrv.h" /* Created by rpcgen, prefix = same as .x */
#include "dispatching__ext.h" /* Oblog structure definition */
#include "cimdef.h" /* Shared memory zone structure definition */ #define NULL 0
main ()
{
/* intern types for client */
register CLIENT *cl;
struct hostent *serv_arg;
struct sockaddr_in server;
int sock = RPC_ANYSOCK;
/* Others types */
MESSAGE_CIM init;
response *result;
int i;

```



```

int *point_init_i;
int *point_obl_i;
OBL_STRING *point_init_s;
OBL_STRING *point_obl_s;
int idshm; /* Shared memory ID */
int idsem; /* Semaphore ID */
SHMCLT_ZONE *tab_datas, local_datas;
struct sembuf sb;
char *adresse;

/*****
/* Create shared memory zones */
*****/

/* Create Shared Memory zones */
idshm = shmget(KEYSHMCLT, sizeof(*tab_datas), 0777);
/* Create 1 Semaphore to control access to shared memory zone */
idsem = semget(KEYSEMCLT, 1, 0777);
/* Get semaphore */
sb.sem_num = 0;
sb.sem_op = -1;
sb.sem_flg = 0;
semop(idsem, &sb, 1);
/* Copy Shared memory zone to local_datas */
memcpy(&local_datas, tab_datas, sizeof(*tab_datas));
/* Release semaphore */
sb.sem_num = 0;
sb.sem_op = 1;
sb.sem_flg = 0;
semop(idsem, &sb, 1);
/* Detach Shared Memory Zone */ shmdt(tab_datas);
*****/
/* Init message to send */
*****/
/* Init host adress */
strcpy(adresse, local_datas.host_receiver);
/* Init dispatcher type */ strcpy(init.typ_dispatch, local_datas.dispatch_type);
/* Init Service type */
init.type_service = local_datas.service_type;
/* Init receiver */
strcpy(init.destinataire, local_datas.receiver);
/* Init int table */
init.Entiers.entiers_len = local_datas.long_tint;
CreateInt(&init.Entiers.entiers_val);
for (i = 0; i < local_datas.long_tint; i++)
{ AppendInt(&init.Entiers.entiers_val, i, local_datas.tint[i]);
}
/* Init str table */
init.Caracts.caracts_len = local_datas.long_tstr;
CreateStr(&init.Caracts.caracts_val);
for (i = 0; i < local_datas.long_tstr; i++)
{
AppendStr(&init.Caracts.caracts_val, i, local_datas.tstr[i]);
}
*****/
serv_arg = gethostbyname(adresse);
bcopy(serv_arg->h_addr, (caddr_t)&server.sin_addr, serv_arg->h_length);
server.sin_family = AF_INET;
server.sin_port = 0;
/* Create client handle */
/* to use UDP in place of TCP, replace tcp by udp */
/* but TCP is more reliable than UDP */
clnt_create(&cl, server, PROGCIM, PROGCIMVERS, "tcp");
if (cl == NULL) {
/* Couldn't establish connection with server */
fprintf(stderr, " ***** ROUTER CLIENT : Couldn't connect *****\n");
}

/* Call to remote procedure */
result = procclnt_1(&init, cl);
}

```

4) Procédures diverses

4.1) La manipulation des listes

```

/* Functions to handle lists (from OBLOG to RPC used lists) */
#include "incCIM.h"
#include "cimsrv.h"
#include "cimdef.h" /* Shared memory zone definition */
/*****
/* Functions to handle list of 'int' */
*****/

void FreeIntlist(list)
    int *list;
{
    int i;
    i = 0;
    while (list[i] != NULL) free(list[i]);
}

CreateInt(list)
    int *list;
{
    if (list != NULL) FreeIntlist(*list);
    list = calloc(1, sizeof(int *));
}

void AppendInt( list, length, val)
    OBL_INT *list;
    int length;
    OBL_INT val;
{
    /* Init new element */
    ((list[length])) = &val;
    /* Reajust size of memory used by list */
    *list = (OBL_INT*) realloc(*list, (length+1)*sizeof(OBL_INT *));
    /* Allocate space for the next value */
    (list)[length+1] = (OBL_INT *) calloc(1, sizeof(OBL_INT));
}
/*****
/* Functions to handle list of 'str' */
*****/

void FreeStrlist(list)
    str *list;
{
    int i;
    i = 0;
    while (list[i] != NULL) free(list[i]);
}

CreateStr(list)
    str *list;
{
    if (list != NULL) FreeStrlist(*list);
    list = calloc(1, sizeof(str *));
}

void AppendStr(list, length, val) OBL_STRING *list;
    int length;
    OBL_STRING val;
{
    /* Init new element */ *((list)[length]) = val;
    /* Reajust size of memory used by list */
    *list = (str*) realloc(*list, (length+1)*sizeof(OBL_STRING *));
    /* Allocate space for the new value */
    (list)[length+1] = (OBL_STRING *) calloc(1, sizeof(OBL_STRING));
}

```

4.2) La manipulation du flag

```

/* Set shared memory zone :readed . So it's possible to copy a new message */
#include "incCIM.h"
#include "cimdef.h" /* Shared memory zone structure definition */
putflag(void)
{
    int idshm; /* Shared memory ID */
    int idsem; /* Semaphore ID */
    struct sembuf sb;
    SHM_ZONE *shm_datas;
    /* Create Shared Memory zone */
    idshm = shmget(KEYSHM,sizeof(*shm_datas),0777);
    /* Create 1 Semaphore to control access to shared memory zone */ idsem = semget(KEYSEM,1,0777);
    /* Attach shared memory zone to buffer */
    shm_datas = (struct SHMSTRUCT *) shmat(idshm,(char *)0,0);
    /* Get semaphore */
    sb.sem_num = 0;
    sb.sem_op = -1;
    sb.sem_flg = 0;
    semop(idsem,&sb,1);
    /* Shared memory zone has been readed */ shm_datas->flag = 0;
    /* Release semaphore */
    sb.sem_num = 0;
    sb.sem_op = 1;
    sb.sem_flg = 0;
    semop(idsem,&sb,1);
    /* Detach Shared Memory Zone */ shmdt(shm_datas);
}

```

4.3) Conversion d'une zone au format RPC vers le format OBLOG

```

/* Reading process from a shared memory zone */
/* Datas to read SHMSTRUCT from are in a structure converted from Oblog */
/* Result : OBL_DATAS */
/* Differences between the two structures : Datas in shared memory zone are */
/* in a array. OBL_DATAS are in form of a pointer list */
#include "incCIM.h"
#include "dispatching__ext.h" /* Oblog structure definition */
#include "cimdef.h" /* Shared memory zone structure definition */
DISPATCHING_send_etrange__FOB *readshm( readed)
    char *readed;
{
    DISPATCHING_send_etrange__FOB *result; int idshm; /* Shared memory ID */
    int idsem; /* Semaphore ID */
    struct sembuf sb;
    SHM_ZONE *shm_datas;
    SHM_ZONE local_datas; /* local datas */
    u_int i;
    readed = "KO";
    /* Create Shared Memory zone */
    idshm = shmget(KEYSHM,sizeof(*shm_datas),0777);
    /* Create 1 Semaphore to control access to shared memory zone */ idsem = semget(KEYSEM,1,0777);
    /* Attach shared memory zone to buffer */
    shm_datas = (struct SHMSTRUCT *) shmat(idshm,(char *)0,0);
    /* Get semaphore */
    sb.sem_num = 0;
    sb.sem_op = -1;
    sb.sem_flg = 0;
    semop(idsem,&sb,1);
    /* Copy Shared memory to local datas */
    memcpy(&local_datas,shm_datas,sizeof(*shm_datas));
    /* Release semaphore */
    sb.sem_num = 0;
    sb.sem_op = 1;
    sb.sem_flg = 0;
    semop(idsem,&sb,1);
}

```



```

/* Detach Shared Memory Zone */
shmdt(shm_datas);
/* Treatment on local datas */
/* Here : conversion from shared memory zone format to oblog structure format */
if (local_datas.flag == 1)
{
    readed = "OK";
    /* Copy dispatcher type */
    strcpy(result->type_dispatch, local_datas.dispatch_type);
    /* Copy service type */
    result->type_service = local_datas.service_type;

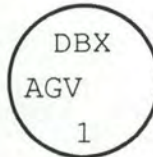
    /* Copy destinataire */ strcpy(result->destinataire, local_datas.receiver);
    /* Copy entiers */
    ENTIERSINew(result->entiers);
    result->entiers = (*(int *) (local_datas.long_tint)); for (i = 1; i <= local_datas.long_tint; i++)
    { ENTIERSIAppend(result->entiers, local_datas.tint[i-1]); }
    /* Copy caracts */
    CARACTSINew(result->caracts);
    result->caracts = (*(OBL_STRING *) (local_datas.long_tstr)); for (i = 1; i <= local_datas.long_tstr; i++)
    { CARACTSIAppend(result->caracts, local_datas.tstr[i-1]); }
}
return(result);
}

```

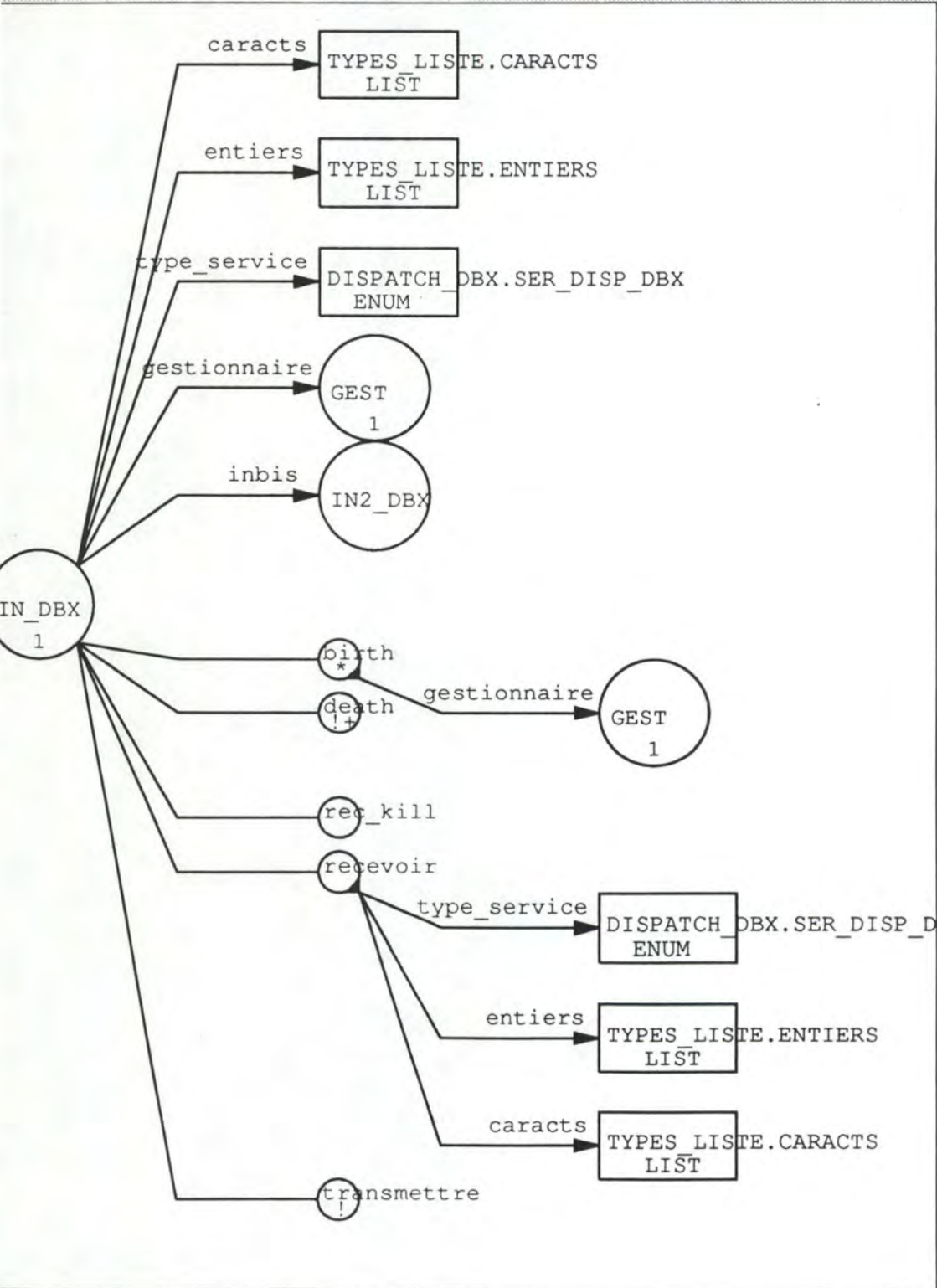
Annexes III: Spécifications OBLOG

Dans cette partie, nous allons présenter les spécifications OBLOG les plus représentatives de l'application développée.

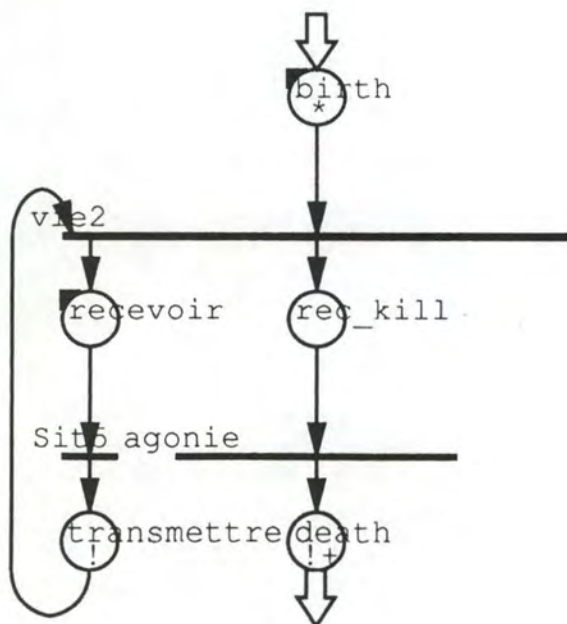
Community Diagram: INTERFACE



Declaration Diagram: INTERFACE\IN_DBX



Behavior Diagram: INTERFACE\IN_DBX



*Attribute Updates of object class **IN_DBX** are:*

*For Action **birth***

gestionnaire := birth.gestionnaire

*For Action **recevoir***

type_service := recevoir.type_service

caracts := recevoir.caracts

entiers := recevoir.entiers

*Calls of object class **IN_DBX** are:*

NORMAL call

*Caller action is: **transmettre***

*Conditioned by: **TRUE***

*Called action is: **INTERFACE.IN2_DBX.recevoir***

*Identifying attribute is: **inbis***

Instantiations are:

recevoir.entiers := entiers

recevoir.gestionnaire := gestionnaire

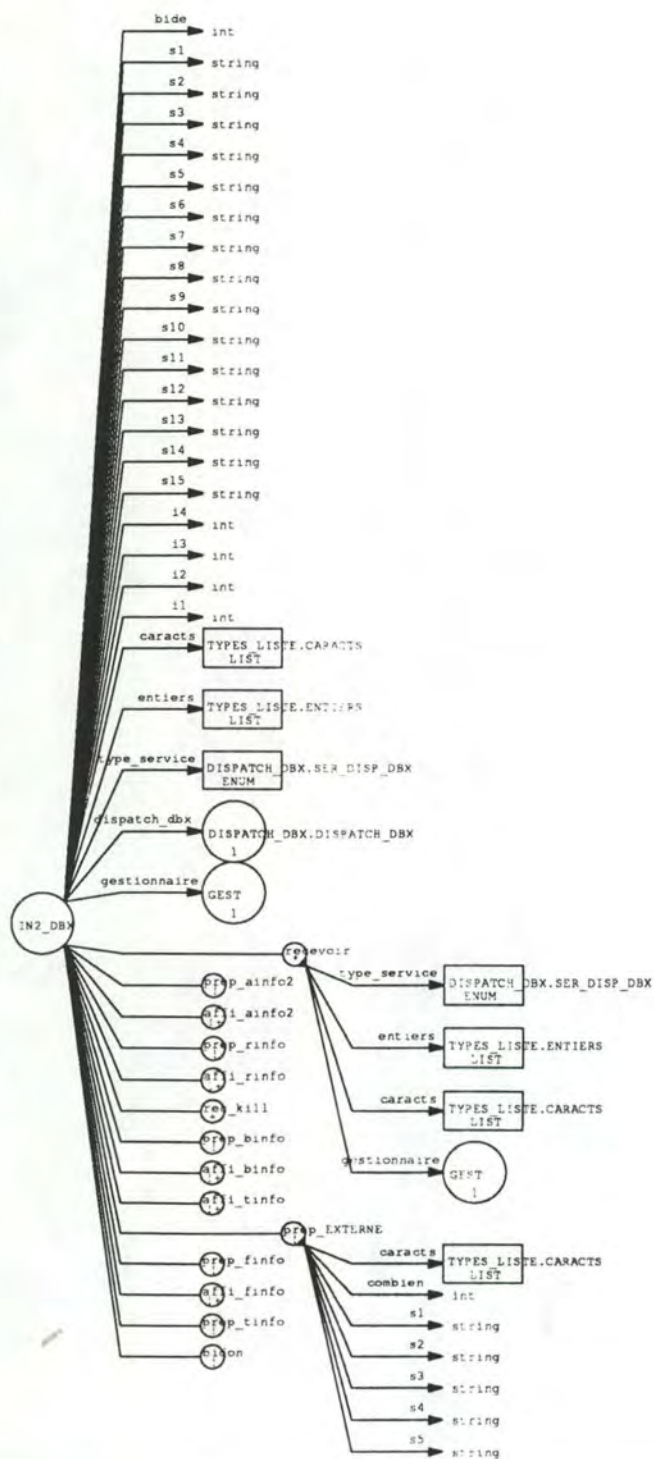
recevoir.caracts := caracts

recevoir.type_service := type_service

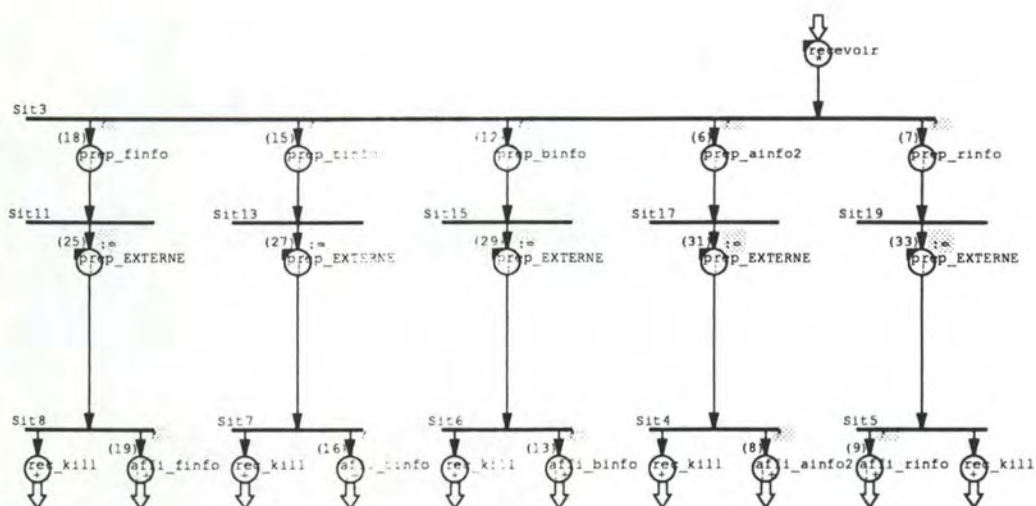
*Behavior Conditions and Instantiations of object class **IN_DBX** are:*

There are no conditions and instantiations of this object class

Declaration Diagram: INTERFACE\IN2__DBX



Behavior Diagram: INTERFACE\IN2_DBX



Attribute Updates of object class IN2_DBX are:

For Action prep_rinfo

i1 := FETCH(entiers, 1)

i2 := FETCH(entiers, 2)

i3 := FETCH(entiers, 3)

i4 := FETCH(entiers, 4)

For Action recevoir

type_service := recevoir.type_service

bide := 1

entiers := recevoir.entiers

caracts := recevoir.caracts

gestionnaire := recevoir.gestionnaire

For Action prep_ainfo2

i2 := FETCH(entiers, 2)

i1 := FETCH(entiers, 1)

For Action prep_binfo

i2 := FETCH(entiers, 2)

i3 := FETCH(entiers, 3)

i1 := FETCH(entiers, 1)

For Action prep_finfo

i1 := FETCH(entiers, 1)

i3 := FETCH(entiers, 3)

i4 := FETCH(entiers, 4)

i2 := FETCH(entiers, 2)

For Action prep_tinfo

i3 := FETCH(entiers, 3)

i1 := FETCH(entiers, 1)

i2 := FETCH(entiers, 2)

i4 := FETCH(entiers, 4)

For Action prep_EXTERNE

s2 := prep_EXTERNE.s2

s4 := prep_EXTERNE.s4

s5 := prep_EXTERNE.s5

s1 := prep_EXTERNE.s1

s3 := prep_EXTERNE.s3

Calls of object class IN2_DBX are:

NORMAL call

Caller action is: **affi_ainfo2**

Conditioned by: **TRUE**

Called action is: **INTERFACE.GEST.rec_ainfo2**

Identifying attribute is: **gestionnaire**

Instantiations are:

rec_ainfo2.pallet_pos := s3

rec_ainfo2.station := s4

rec_ainfo2.distance := i1

rec_ainfo2.speed := s1

rec_ainfo2.status := s5

rec_ainfo2.direction := i2

rec_ainfo2.dock_state := s2

NORMAL call

Caller action is: **affi_binfo**

Conditioned by: **TRUE**

Called action is: **INTERFACE.GEST.rec_binfo**

Identifying attribute is: **gestionnaire**

Instantiations are:

rec_binfo.x_pos := i2

rec_binfo.dock_state := s1

rec_binfo.status := s2

rec_binfo.speed := i3

rec_binfo.y_pos := i1

NORMAL call

Caller action is: **affi_finfo**

Conditioned by: **TRUE**

Called action is: **INTERFACE.GEST.rec_finfo**

Identifying attribute is: **gestionnaire**

Instantiations are:

rec_finfo.z_pos := i4

rec_finfo.nom_fichier := s1

rec_finfo.distance := i1

rec_finfo.type_prg := s5

rec_finfo.x_pos := i2

rec_finfo.status := s4

rec_finfo.y_pos := i3

rec_finfo.dock_state := s2
rec_finfo.pallet_post := s3

NORMAL call

Caller action is: affi_tinfo

Conditioned by: TRUE

Called action is: INTERFACE.GEST.rec_tinfo

Identifying attribute is: gestionnaire

Instantiations are:

rec_tinfo.distance := i1
rec_tinfo.nom_fichier := s1
rec_tinfo.x_pos := i2
rec_tinfo.dock_state := s2
rec_tinfo.status := s4
rec_tinfo.z_pos := i4
rec_tinfo.pallet_post := s3
rec_tinfo.type_prg := s5
rec_tinfo.y_pos := i3

NORMAL call

Caller action is: affi_rinfo

Conditioned by: TRUE

Called action is: INTERFACE.GEST.rec_rinfo

Identifying attribute is: gestionnaire

Instantiations are:

rec_rinfo.status := s2
rec_rinfo.z_pos := i4
rec_rinfo.nom_fichier := s1
rec_rinfo.num_prg := i1
rec_rinfo.y_pos := i3
rec_rinfo.x_pos := i2

TO EXTERIOR call

Caller action is: prep_EXTERNE

OUT Instantiations are:

caracts
combien

IN Instantiations are:

s3

s1
s5
s2
s4

Behavior Conditions and Instantiations of object class IN2_DBX are:

(15) **prep_tinfo**

Conditioned by: (type_service = SER_DISP_DBX\$AFFI_TINFO:DISPATCH
_DBX)

(25) **prep_EXTERNE**

Conditioned by: TRUE

Instantiations are:

prep_EXTERNE.combien := 5

prep_EXTERNE.caracts := caracts

(31) **prep_EXTERNE**

Conditioned by: TRUE

Instantiations are:

prep_EXTERNE.combien := 5

prep_EXTERNE.caracts := caracts

(29) **prep_EXTERNE**

Conditioned by: TRUE

Instantiations are:

prep_EXTERNE.combien := 2

prep_EXTERNE.caracts := caracts

(19) **affi_finfo**

Conditioned by: EXISTS[GEST:INTERFACE|TRUE]

(9) **affi_rinfo**

Conditioned by: EXISTS[GEST:INTERFACE|TRUE]

(16) **affi_tinfo**

Conditioned by: EXISTS[GEST:INTERFACE|TRUE]

(13) **affi_binfo**

Conditioned by: EXISTS[GEST:INTERFACE|TRUE]

(7) **prep_rinfo**

Conditioned by: (type_service = SER_DISP_DBX\$AFFI_RNPRG:DISPATC
H_DBX)

(27) **prep_EXTERNE**

Conditioned by: TRUE

Instantiations are:

prep_EXTERNE.combien := 5

prep_EXTERNE.caracts := caracts

(12) prep_binfo

Conditioned by: **(type_service = SER_DISP_DBX\$AFFI_BINFO:DISPATCH
_DBX)**

(33) prep_EXTERNE

Conditioned by: **TRUE**

Instantiations are:

prep_EXTERNE.combien := 2

prep_EXTERNE.caracts := caracts

(6) prep_ainfo2

Conditioned by: **(type_service = SER_DISP_DBX\$AFFI_AINFO2:DISPATCH
H_DBX)**

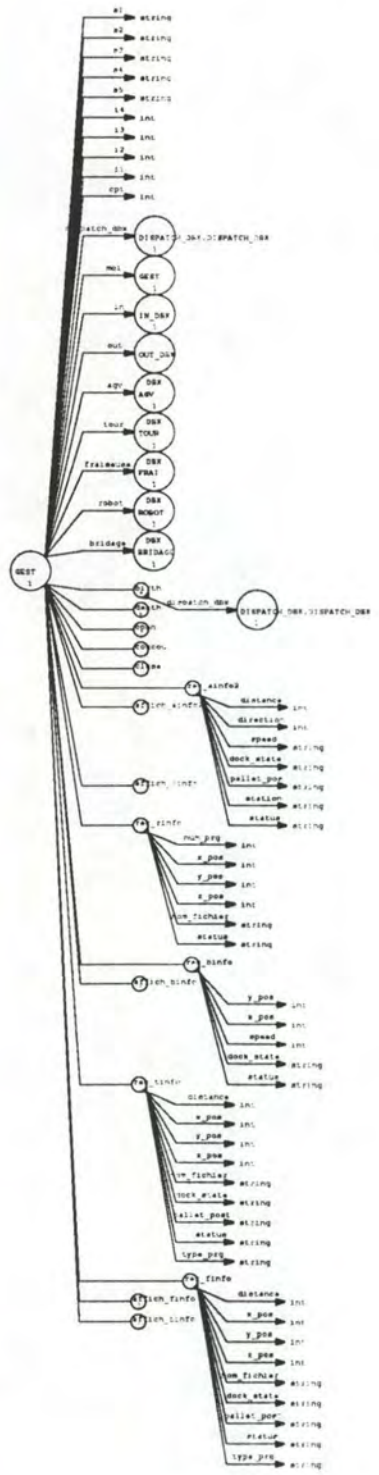
(18) prep_finfo

Conditioned by: **(type_service = SER_DISP_DBX\$AFFI_FINFO:DISPATCH
_DBX)**

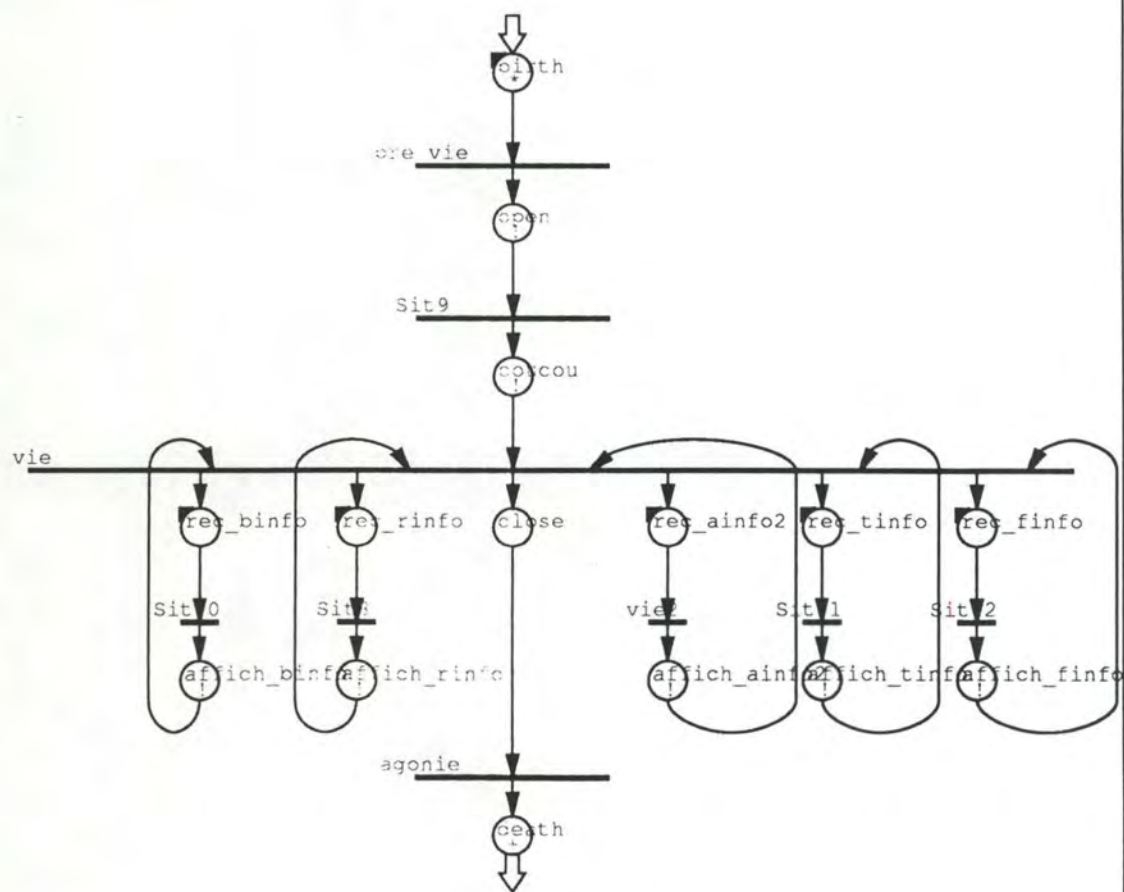
(8) affi_ainfo2

Conditioned by: **EXISTS[GEST:INTERFACE|TRUE]**

Declaration Diagram: INTERFACE\GEST



Behavior Diagram: INTERFACE\GEST



Attribute Updates of object class GEST are:

For Action birth

moi := SELF

cpt := 0

dispatch_dbx := birth.dispatch_dbx

For Action rec_ainfo2

s4 := rec_ainfo2.station

s5 := rec_ainfo2.status

i1 := rec_ainfo2.distance

s3 := rec_ainfo2.pallet_pos

i2 := rec_ainfo2.direction

s1 := rec_ainfo2.speed

s2 := rec_ainfo2.dock_state

For Action rec_binfo

s2 := rec_binfo.status

s1 := rec_binfo.dock_state

i2 := rec_binfo.x_pos

i1 := rec_binfo.y_pos

i3 := rec_binfo.speed

For Action rec_finfo

s3 := rec_finfo.pallet_post

i1 := rec_finfo.distance

i4 := rec_finfo.z_pos

s4 := rec_finfo.status

s5 := rec_finfo.type_prg

s2 := rec_finfo.dock_state

i3 := rec_finfo.y_pos

s1 := rec_finfo.nom_fichier

i2 := rec_finfo.x_pos

For Action rec_tinfo

s4 := rec_tinfo.status

s3 := rec_tinfo.pallet_post

s1 := rec_tinfo.nom_fichier

i3 := rec_tinfo.y_pos

s5 := rec_tinfo.type_prg

i1 := rec_tinfo.distance

s2 := rec_tinfo.dock_state

i2 := rec_tinfo.x_pos

i4 := rec_tinfo.z_pos

For Action rec_rinfo

i1 := rec_rinfo.num_prg

s1 := rec_rinfo.nom_fichier

i2 := rec_rinfo.x_pos

s2 := rec_rinfo.status

i3 := rec_rinfo.y_pos

i4 := rec_rinfo.z_pos

Calls of object class **GEST** *are:*

NORMAL call

Caller action is: **close**

Conditioned by: **TRUE**

Called action is: **INTERFACE.FRAI.close**

Identifying attribute is: **fraiseuse**

NORMAL call

Caller action is: **close**

Conditioned by: **TRUE**

Called action is: **INTERFACE.ROBOT.close**

Identifying attribute is: **robot**

NORMAL call

Caller action is: **close**

Conditioned by: **TRUE**

Called action is: **INTERFACE.BRIDAGE.close**

Identifying attribute is: **bridage**

NORMAL call

Caller action is: **close**

Conditioned by: **TRUE**

Called action is: **INTERFACE.AGV.close**

Identifying attribute is: **agv**

NORMAL call

Caller action is: **close**

Conditioned by: **TRUE**

Called action is: **INTERFACE.TOUR.close**

Identifying attribute is: **tour**

NORMAL call

Caller action is: **open**
Conditioned by: **TRUE**
Called action is: **INTERFACE.AGV.birth**
Identifying attribute is: **agv**
Instantiations are:

birth.maitre := moi

NORMAL call

Caller action is: **open**
Conditioned by: **TRUE**
Called action is: **INTERFACE.ROBOT.birth**
Identifying attribute is: **robot**
Instantiations are:

birth.maitre := moi

NORMAL call

Caller action is: **open**
Conditioned by: **TRUE**
Called action is: **INTERFACE.BRIDAGE.birth**
Identifying attribute is: **bridage**
Instantiations are:

birth.maitre := moi

NORMAL call

Caller action is: **open**
Conditioned by: **TRUE**
Called action is: **INTERFACE.TOUR.birth**
Identifying attribute is: **tour**
Instantiations are:

birth.maitre := moi

NORMAL call

Caller action is: **open**
Conditioned by: **TRUE**
Called action is: **INTERFACE.FRAI.birth**
Identifying attribute is: **fraiseuse**
Instantiations are:

birth.maitre := moi

NORMAL call

Caller action is: **open**

Conditioned by: **TRUE**

Called action is: **INTERFACE.IN_DBX.birth**

Identifying attribute is: **in**

Instantiations are:

birth.gestionnaire := moi

NORMAL call

Caller action is: **affich_ainfo2**

Conditioned by: **TRUE**

Called action is: **INTERFACE.AGV.rec_ainfo2**

Identifying attribute is: **agv**

Instantiations are:

rec_ainfo2.pallet_pos := s3

rec_ainfo2.station := s4

rec_ainfo2.dock_state := s2

rec_ainfo2.status := s5

rec_ainfo2.speed := s1

rec_ainfo2.direction := i2

rec_ainfo2.distance := i1

NORMAL call

Caller action is: **affich_binfo**

Conditioned by: **TRUE**

Called action is: **INTERFACE.BRIDAGE.rec_binfo**

Identifying attribute is: **bridge**

Instantiations are:

rec_binfo.dock_state := s1

rec_binfo.speed := i3

rec_binfo.status := s2

rec_binfo.y_pos := i1

rec_binfo.x_pos := i2

NORMAL call

Caller action is: **affich_finfo**

Conditioned by: **TRUE**

Called action is: **INTERFACE.FRAI.affiche**

Identifying attribute is: **fraiseuse**

Instantiations are:

affiche.type_prg := s5

affiche.y_pos := i3
affiche.nom_fichier := s1
affiche.dock_state := s2
affiche.status := s4
affiche.z_pos := i4
affiche.pallet_post := s3
affiche.nprg := i1
affiche.x_pos := i2

NORMAL call

Caller action is: affich_tinfo

Conditioned by: TRUE

Called action is: INTERFACE.TOUR.affiche

Identifying attribute is: tour

Instantiations are:

affiche.y_pos := i3
affiche.pallet_post := s3
affiche.dock_state := s2
affiche.x_pos := i2
affiche.nom_fichier := s1
affiche.nprg := i1
affiche.status := s4
affiche.z_pos := i4
affiche.type_prg := s5

NORMAL call

Caller action is: affich_rinfo

Conditioned by: TRUE

Called action is: INTERFACE.ROBOT.rec_info

Identifying attribute is: robot

Instantiations are:

rec_info.num_prg := i1
rec_info.y_pos := i3
rec_info.nom_fichier := s1
rec_info.z_pos := i4
rec_info.x_pos := i2
rec_info.status := s2

NORMAL call

Caller action is: **coucou**

Conditioned by: **TRUE**

Called action is: **INTERFACE.OUT_DBX.coucou**

Identifying attribute is: **out**

Instantiations are:

coucou.in := in

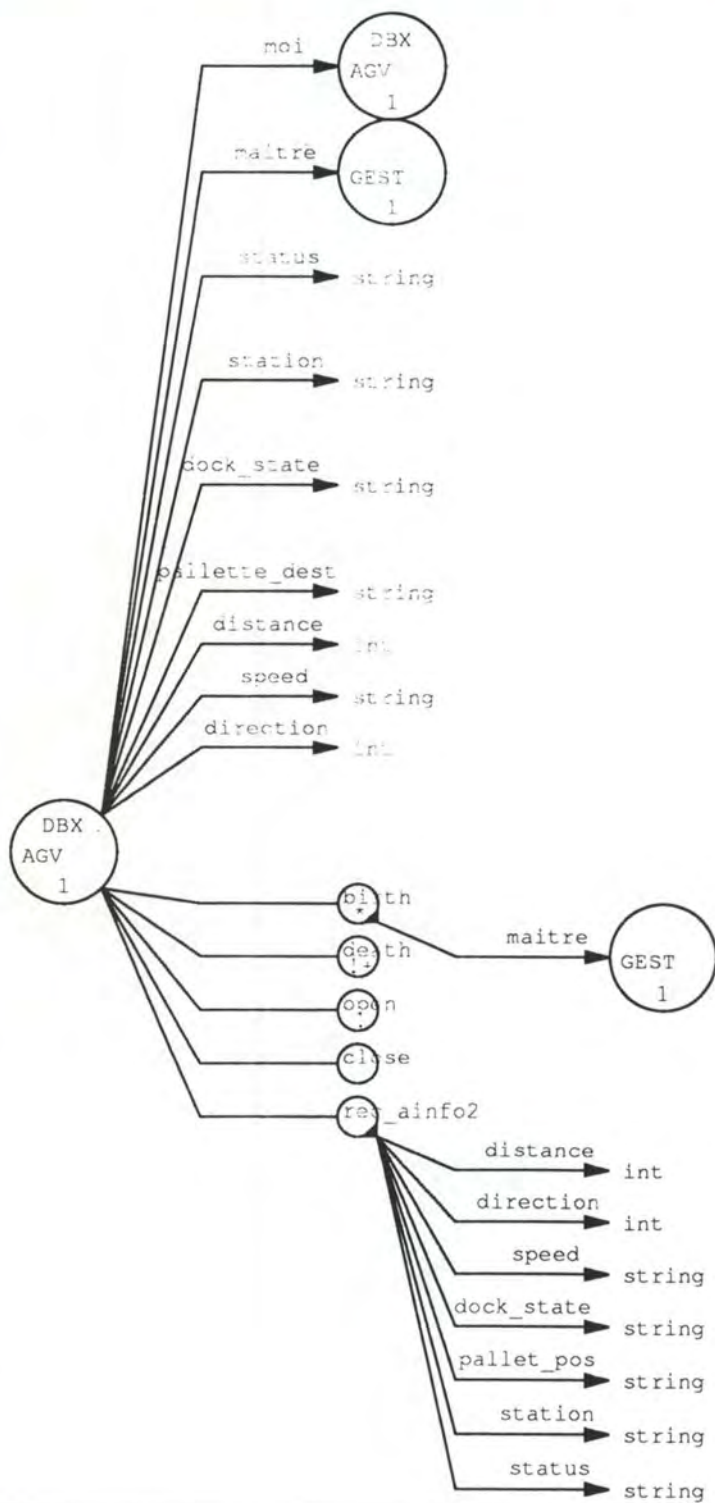
coucou.dispatch_dbx := dispatch_dbx

coucou.gestionnaire := moi

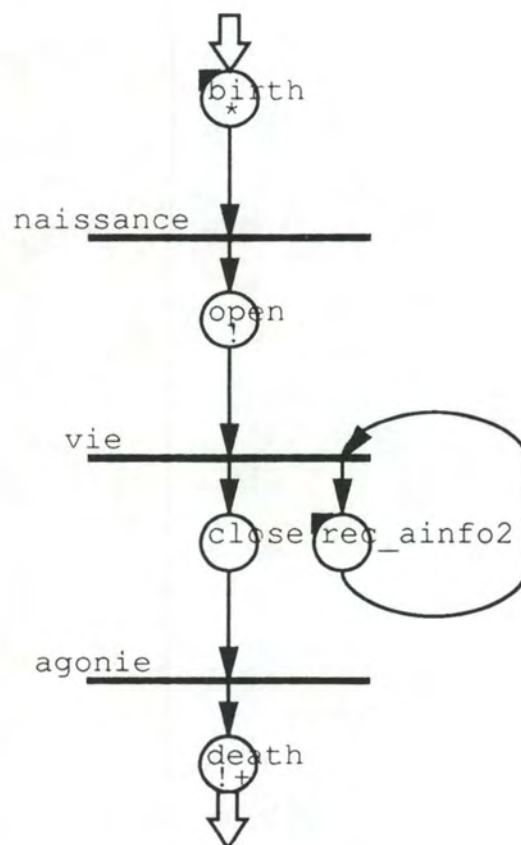
*Behavior Conditions and Instantiations of object class **GEST** are:*

There are no conditions and instantiations of this object class

Declaration Diagram: INTERFACE\AGV



Behavior Diagram: INTERFACE\AGV



Status:

Station:

Pallette Dest:

Distance:

Speed:

Direction:

Block state:

Attribute Updates of object class AGV are:

For Action rec_ainfo2

speed := rec_ainfo2.speed
pallette_dest := rec_ainfo2.pallet_pos
status := rec_ainfo2.status
direction := rec_ainfo2.direction
station := rec_ainfo2.station
distance := rec_ainfo2.distance
dock_state := rec_ainfo2.dock_state

For Action birth

moi := SELF
maitre := birth.maitre

Calls of object class AGV are:

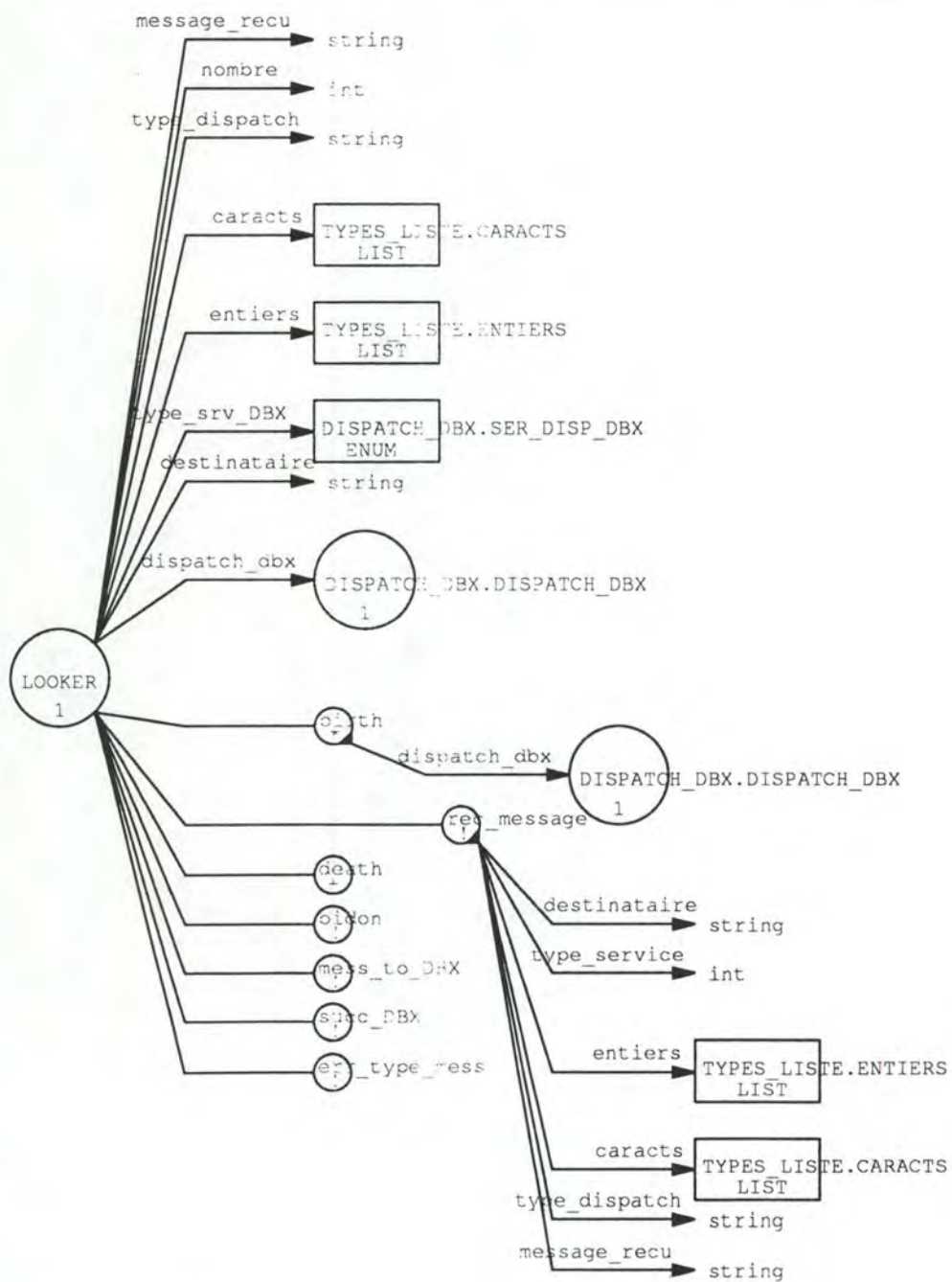
Behavior Conditions and Instantiations of object class AGV are:

There are no conditions and instantiations of this object class

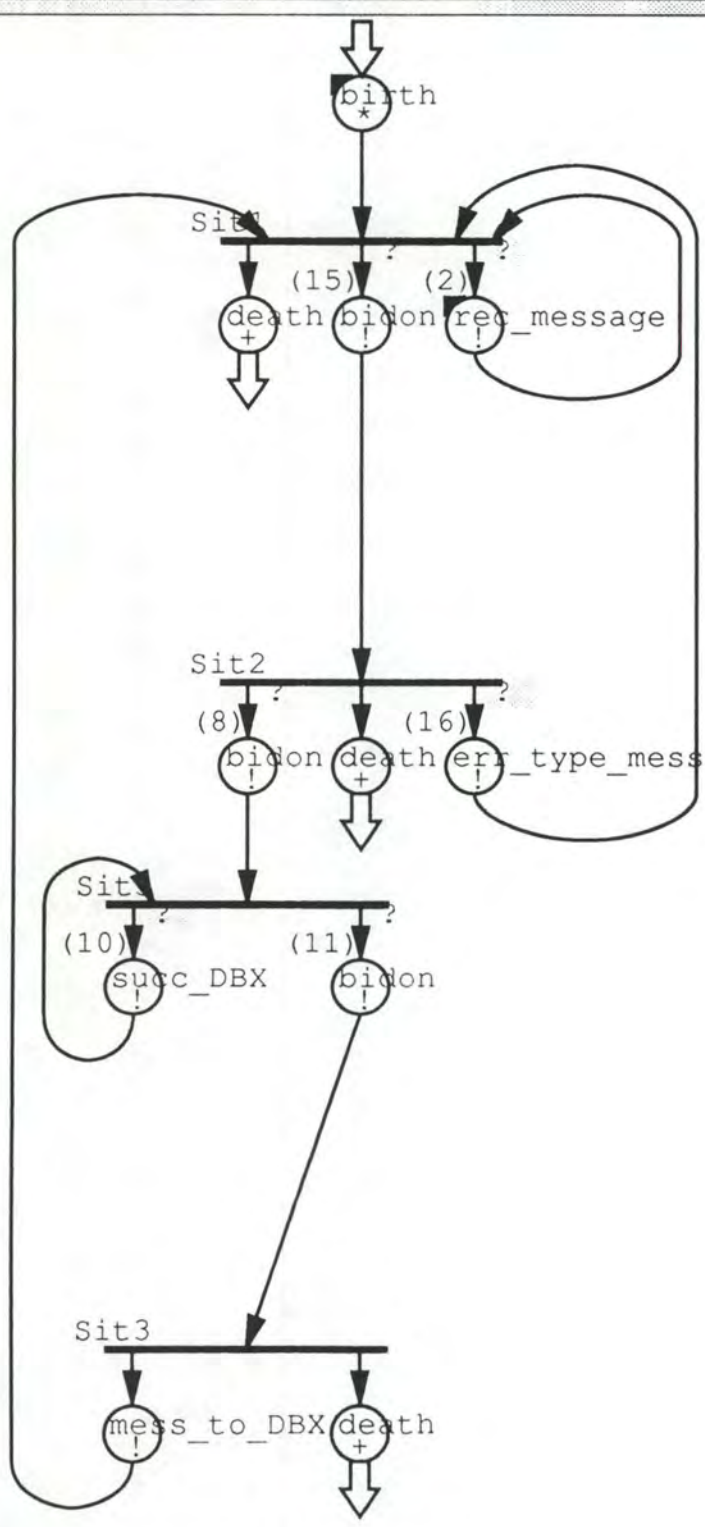
Community Diagram: LOOKERS



Declaration Diagram: LOOKERS\LOOKER



Behavior Diagram: LOOKERS\LOOKER



Attribute Updates of object class LOOKER are:

For Action rec_message

type_srv_DBX := SER_DISP_DBX\$AFFI_AINFO2:DISPATCH_DBX
message_recu := rec_message.message_recu
destinataire := rec_message.destinataire
type_dispatch := rec_message.type_dispatch
entiers := rec_message.entiers
caracts := rec_message.caracts
nombre := rec_message.type_service

For Action birth

message_recu := "NON"
dispatch_dbx := birth.dispatch_dbx

For Action mess_to_DBX

message_recu := "NON"

For Action succ_DBX

nombre := (nombre - 1)
type_srv_DBX := SUCC(type_srv_DBX)

For Action err_type_mess

message_recu := "NON"

Calls of object class LOOKER are:

FOREIGNER call

Caller action is: mess_to_DBX

Conditioned by: TRUE

Called action is: DISPATCH_DBX.DISPATCH_DBX.recevoir

Identifying attribute is: dispatch_dbx

Instantiations are:

recevoir.entiers := entiers
recevoir.caracts := caracts
recevoir.destinataire := destinataire
recevoir.type_service := type_srv_DBX

TO EXTERIOR call

Caller action is: rec_message

IN Instantiations are:

message_recu
destinataire
entiers
caracts

type_dispatch

type_service

*Behavior Conditions and Instantiations of object class **LOOKER** are:*

(2) **rec_message**

Conditioned by: (message_recu = "NON")

(16) **err_type_mess**

Conditioned by: NOT((type_dispatch = "DBX"))

(11) **bidon**

Conditioned by: (nombre <= 0)

(10) **succ_DBX**

Conditioned by: (nombre > 0)

(8) **bidon**

Conditioned by: (type_dispatch = "DBX")

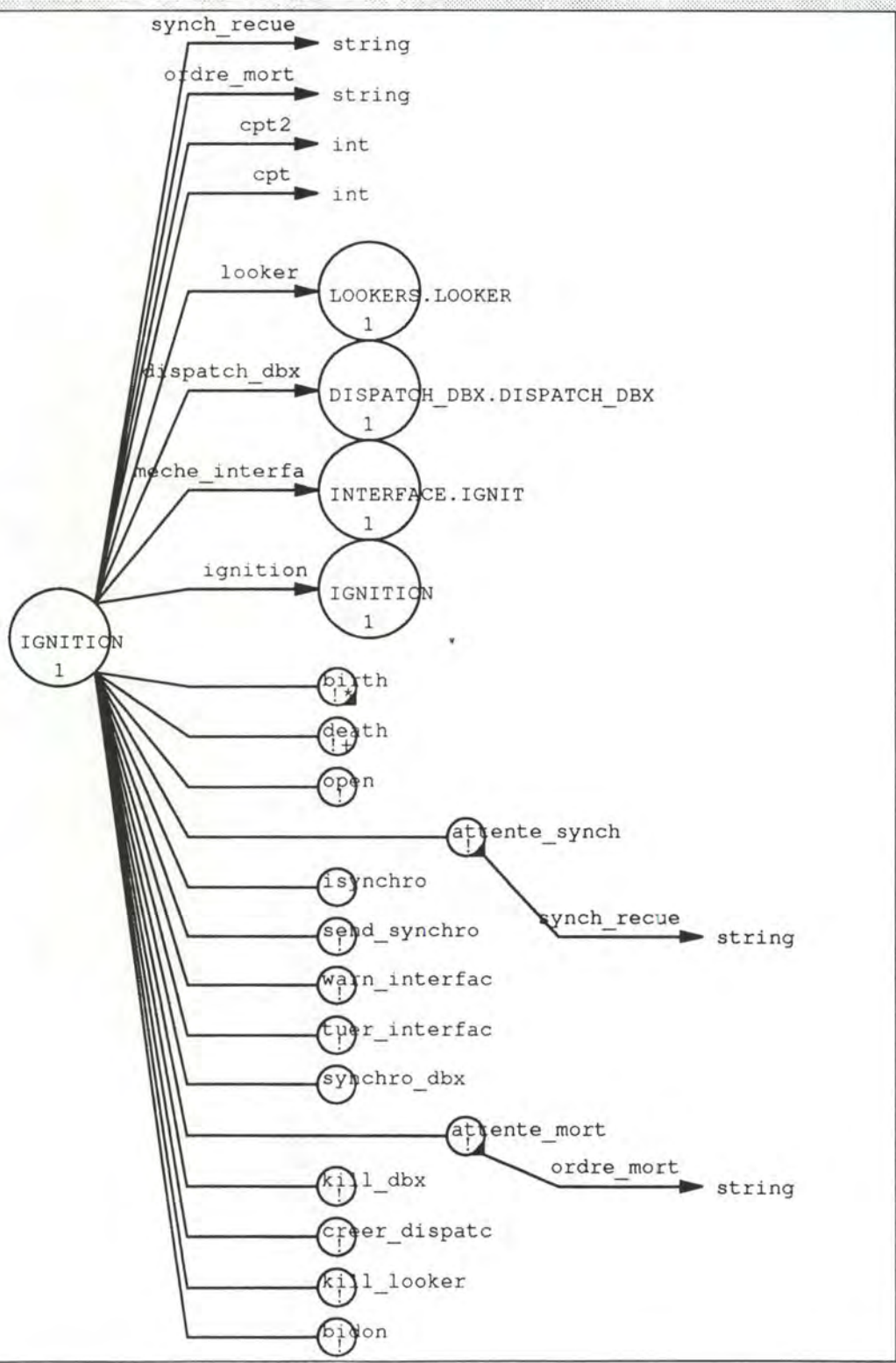
(15) **bidon**

Conditioned by: (message_recu = "OUI")

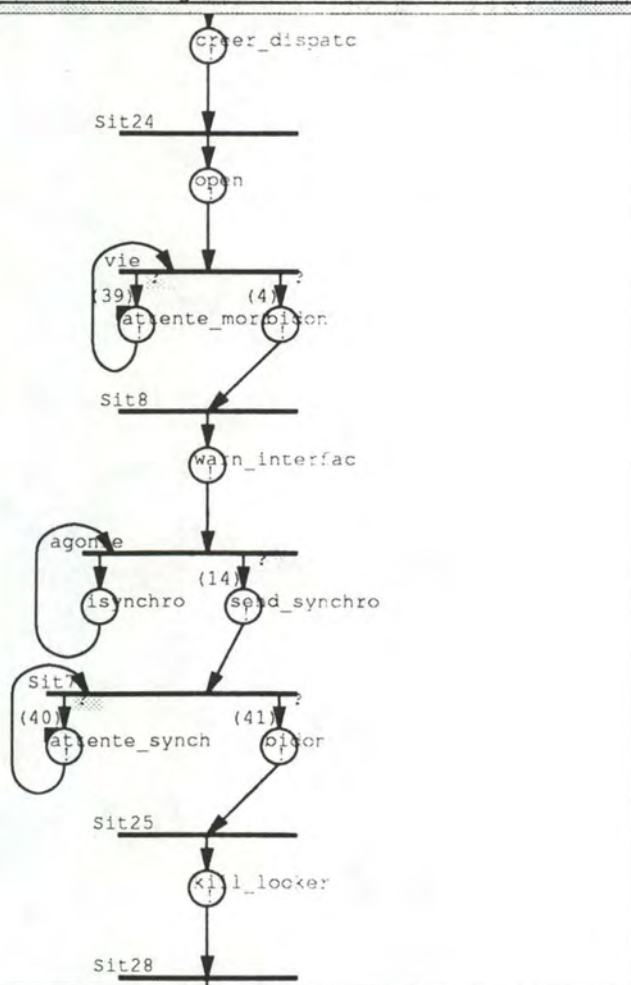
Community Diagram: IGNITIONS



Declaration Diagram: IGNITIONS\IGNITION



Behavior Diagram: IGNITIONS\IGNITION



Attribute Updates of object class IGNITION are:

For Action synchro_dbx

cpt2 := (cpt2 + 1)

For Action attente_synch

synch_recue := attente_synch.synch_recue

For Action attente_mort

ordre_mort := attente_mort.ordre_mort

For Action send_synchro

synch_recue := "NON"

For Action isynchro

cpt := (cpt + 1)

For Action birth

cpt := 0

ignition := SELF

dispatch_dbx := birth.dispatch_dbx

cpt2 := 0

ordre_mort := "NON"

Calls of object class IGNITION are:

FOREIGNER call

Caller action is: kill_looker

Conditioned by: TRUE

Called action is: LOOKERS.LOOKER.death

Identifying attribute is: looker

FOREIGNER call

Caller action is: open

Conditioned by: TRUE

Called action is: INTERFACE.IGNIT.birth

Identifying attribute is: meche_interfa

Instantiations are:

birth.dispatch_dbx := dispatch_dbx

birth.ignition := ignition

FOREIGNER call

Caller action is: open

Conditioned by: TRUE

Called action is: LOOKERS.LOOKER.birth

Identifying attribute is: looker

Instantiations are:

birth.dispatch_dbx := dispatch_dbx

FOREIGNER call

Caller action is: tuer_interfac

Conditioned by: TRUE

Called action is: INTERFACE.IGNIT.rec_synchro

Identifying attribute is: meche_interfa

FOREIGNER call

Caller action is: warn_interfac

Conditioned by: TRUE

Called action is: INTERFACE.IGNIT.rec_warn

Identifying attribute is: meche_interfa

FOREIGNER call

Caller action is: creer_dispatc

Conditioned by: TRUE

Called action is: DISPATCH_DBX.DISPATCH_DBX.birth

Identifying attribute is: dispatch_dbx

Instantiations are:

birth.ignition := ignition

FOREIGNER call

Caller action is: kill_dbx

Conditioned by: TRUE

Called action is: DISPATCH_DBX.DISPATCH_DBX.warn

Identifying attribute is: dispatch_dbx

TO EXTERIOR call

Caller action is: attente_synch

IN Instantiations are:

synch_recue

TO EXTERIOR call

Caller action is: attente_mort

IN Instantiations are:

ordre_mort

TO EXTERIOR call

Caller action is: send_synchro

Behavior Conditions and Instantiations of object class IGNITION are:

(39) **attente_mort**

Conditioned by: (ordre_mort <> "OUI")

(40) **attente_synch**

Conditioned by: (synch_recue <> "OUI")

(41) **bidon**

Conditioned by: (synch_recue = "OUI")

(14) **send_synchro**

Conditioned by: (cpt = 1)

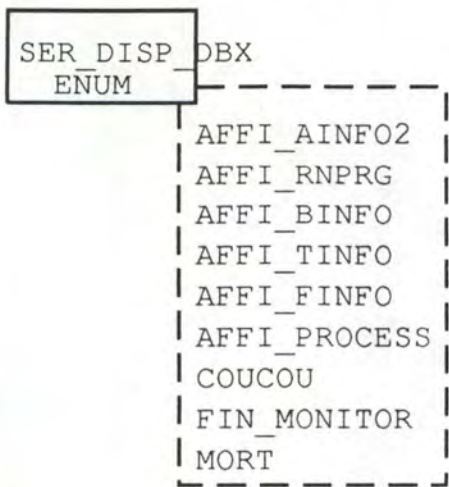
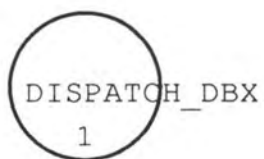
(4) **bidon**

Conditioned by: (ordre_mort = "OUI")

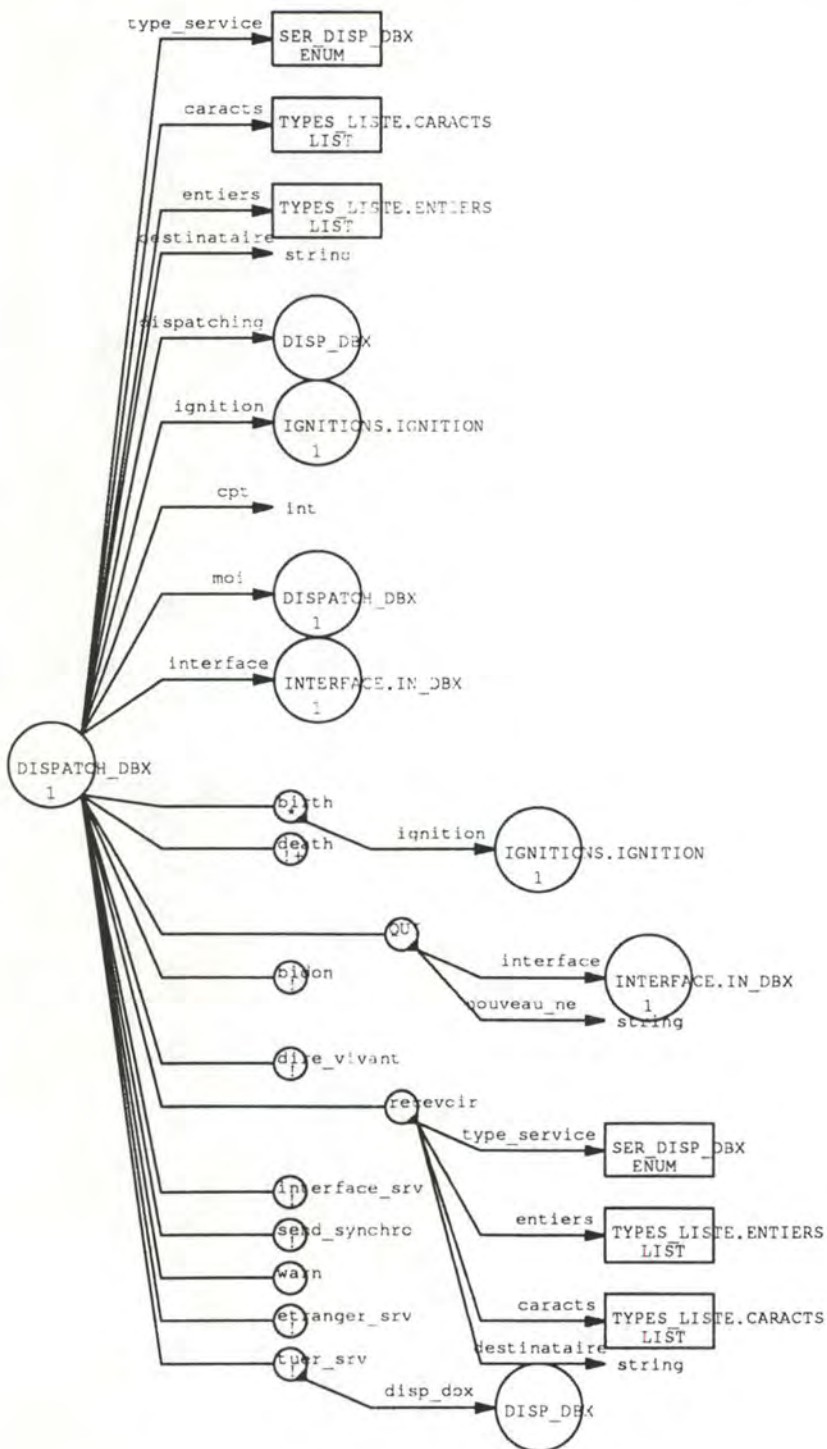
(21) **tuer_interfac**

Conditioned by: (cpt2 = 1)

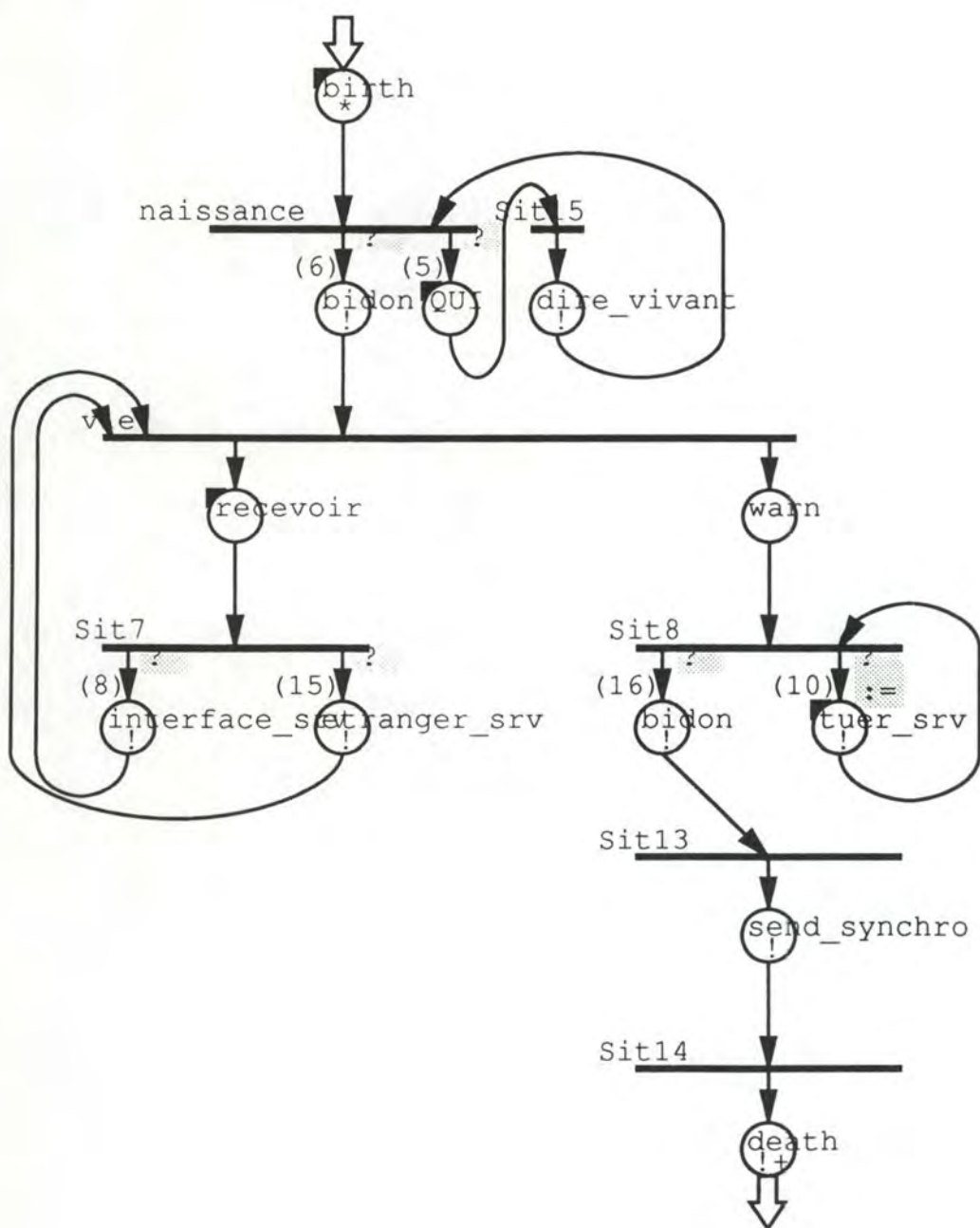
Community Diagram: DISPATCH_DBX



Declaration Diagram: DISPATCH_DBX\DISPATCH_DBX



Behavior Diagram: DISPATCH_DBX\DISPATCH_DBX



Attribute Updates of object class DISPATCH_DBX are:

For Action recevoir

type_service := recevoir.type_service
destinataire := recevoir.destinataire
caracts := recevoir.caracts
entiers := recevoir.entiers

For Action birth

interface := UNDEFINED
moi := SELF
cpt := 0
ignition := birth.ignition

For Action QUI

destinataire := "ROUTEUR_RPC"
cpt := (cpt + 1)
interface := QUI.interface
type_service := SER_DISP_DBX\$COUCOU:DISPATCH_DBX
caracts := APPEND(NEW(caracts), QUI.nouveau_ne)

Calls of object class DISPATCH_DBX are:

NORMAL call

Caller action is: interface_srv

Conditioned by: TRUE

Called action is: DISPATCH_DBX.DISP_DBX.interface_srv

Identifying attribute is: dispatching

Instantiations are:

interface_srv.type_service := type_service
interface_srv.destinataire := destinataire
interface_srv.entiers := entiers
interface_srv.interface := interface
interface_srv.caracts := caracts

NORMAL call

Caller action is: tuer_srv

Conditioned by: TRUE

Called action is: DISPATCH_DBX.DISP_DBX.avant_mort

Identifying attribute is: tuer_srv.disp_dbx

FOREIGNER call

Caller action is: send_synchro

Conditioned by: TRUE

Called action is: **IGNITIONS.IGNITION.synchro_dbx**

Identifying attribute is: **ignition**

NORMAL call

Caller action is: **dire_vivant**

Conditioned by: **TRUE**

Called action is: **DISPATCH_DBX.DISP_DBX.etranger_srv**

Identifying attribute is: **dispatching**

Instantiations are:

etranger_srv.caracts := caracts

etranger_srv.type_service := type_service

etranger_srv.entiers := entiers

etranger_srv.destinataire := destinataire

NORMAL call

Caller action is: **etranger_srv**

Conditioned by: **TRUE**

Called action is: **DISPATCH_DBX.DISP_DBX.etranger_srv**

Identifying attribute is: **dispatching**

Instantiations are:

etranger_srv.destinataire := destinataire

etranger_srv.caracts := caracts

etranger_srv.entiers := entiers

etranger_srv.type_service := type_service

Behavior Conditions and Instantiations of object class DISPATCH_DBX are:

(8) **interface_srv**

Conditioned by: **(destinataire = "INTERFACE")**

(6) **bidon**

Conditioned by: **(cpt = 2)**

(5) **QUI**

Conditioned by: **((cpt < 2) AND NOT(EXISTS[DISP_DBX:DISPATCH_DBX|(type_service = SER_DISP_DBX\$COUCOU:DISPATCH_DBX)]))**

(15) **etranger_srv**

Conditioned by: **(destinataire <> "INTERFACE")**

(16) **bidon**

Conditioned by: **NOT(EXISTS[DISP_DBX:DISPATCH_DBX|TRUE])**

(10) **tuer_srv**

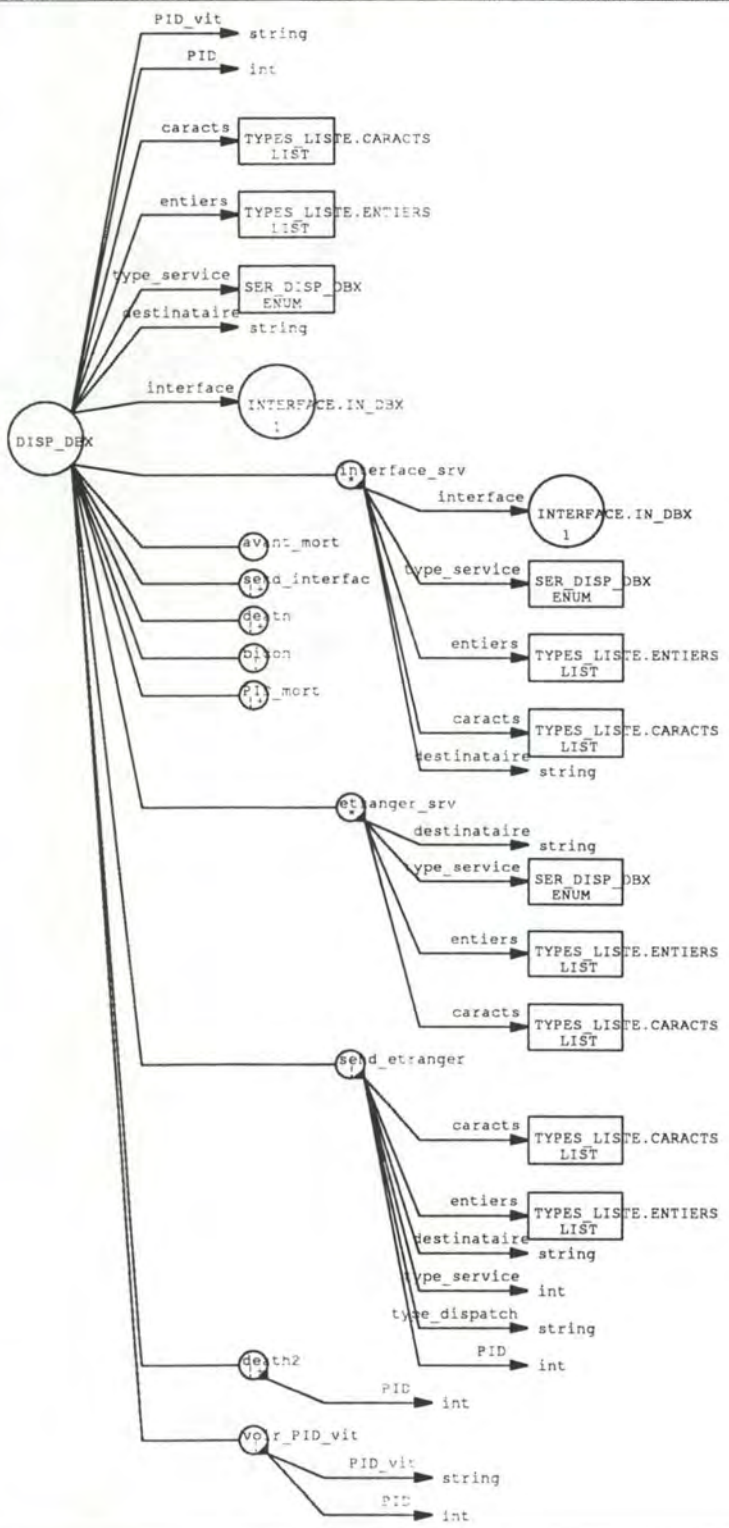
Conditioned by: **(EXISTS[DISP_DBX:DISPATCH_DBX|TRUE] AND EXIS**

**TS[DISP_DBX:DISPATCH_DBX|(type_service <> SER_DISP_DBX\$M
ORT:DISPATCH_DBX))**

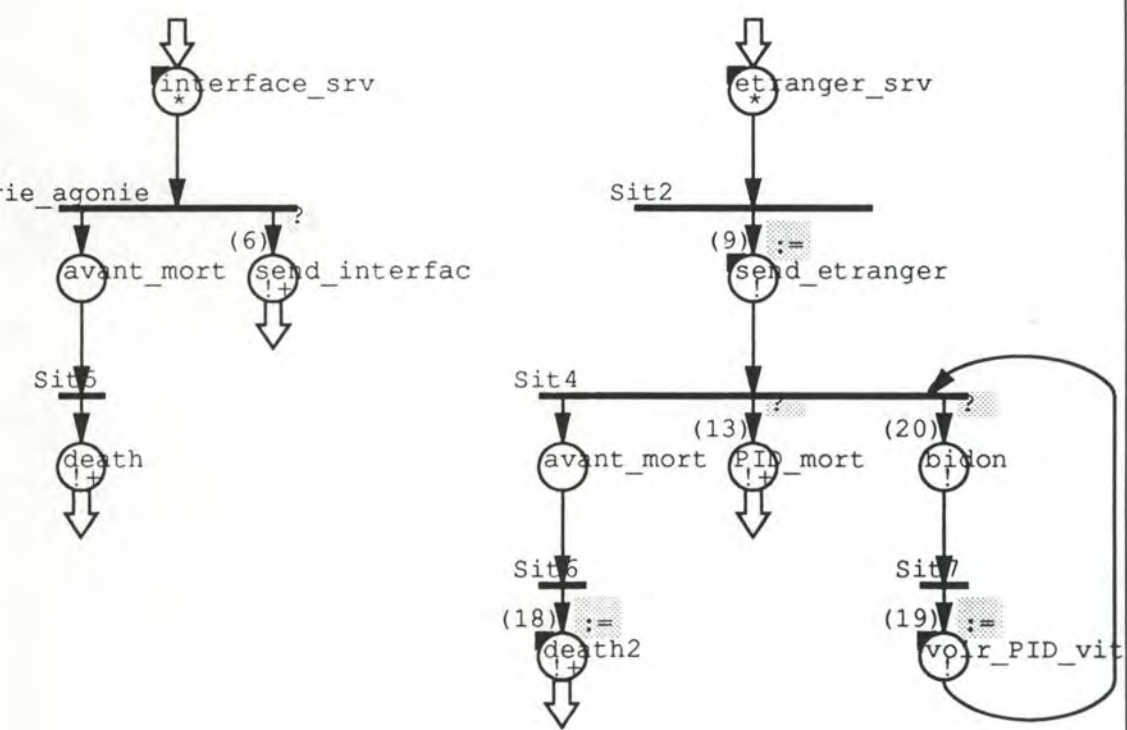
Instantiations are:

tuer_srv.disp_dbx := ONE[DISP_DBX:DISPATCH_DBX|TRUE]

Declaration Diagram: DISPATCH_DBX\DISP_DBX



Behavior Diagram: DISPATCH_DBX\DISP_DBX



Attribute Updates of object class DISP_DBX are:

For Action avant_mort

type_service := SER_DISP_DBX\$MORT:DISPATCH_DBX

For Action interface_srv

interface := interface_srv.interface

entiers := interface_srv.entiers

type_service := interface_srv.type_service

destinataire := interface_srv.destinataire

caracts := interface_srv.caracts

For Action etranger_srv

entiers := etranger_srv.entiers

type_service := etranger_srv.type_service

destinataire := etranger_srv.destinataire

caracts := etranger_srv.caracts

For Action voir_PID_vit

PID_vit := voir_PID_vit.PID_vit

For Action send_etranger

PID_vit := "OUI"

PID := send_etranger.PID

Calls of object class DISP_DBX are:

FOREIGNER call

Caller action is: send_interfac

Conditioned by: TRUE

Called action is: INTERFACE.IN_DBX.recevoir

Identifying attribute is: interface

Instantiations are:

recevoir.caracts := caracts

recevoir.entiers := entiers

recevoir.type_service := type_service

TO EXTERIOR call

Caller action is: death2

OUT Instantiations are:

PID

TO EXTERIOR call

Caller action is: voir_PID_vit

OUT Instantiations are:

PID

IN Instantiations are:

PID_vit

TO EXTERIOR call

Caller action is: send_etranger

OUT Instantiations are:

entiers

type_dispatch

caracts

destinataire

type_service

IN Instantiations are:

PID

*Behavior Conditions and Instantiations of object class **DISP_DBX** are:*

(19) **voir_PID_vit**

Conditioned by: TRUE

Instantiations are:

voir_PID_vit.PID := PID

(6) **send_interfac**

*Conditioned by: NOT(EXISTS[IN2_DBX:INTERFACE|(type_service = SE
LF.type_service)])*

(18) **death2**

Conditioned by: TRUE

Instantiations are:

death2.PID := PID

(20) **bidon**

Conditioned by: (PID_vit = "OUI")

(9) **send_etranger**

Conditioned by: TRUE

Instantiations are:

send_etranger.type_dispatch := "DBX"

send_etranger.destinataire := destinataire

send_etranger.type_service := ORD(type_service)

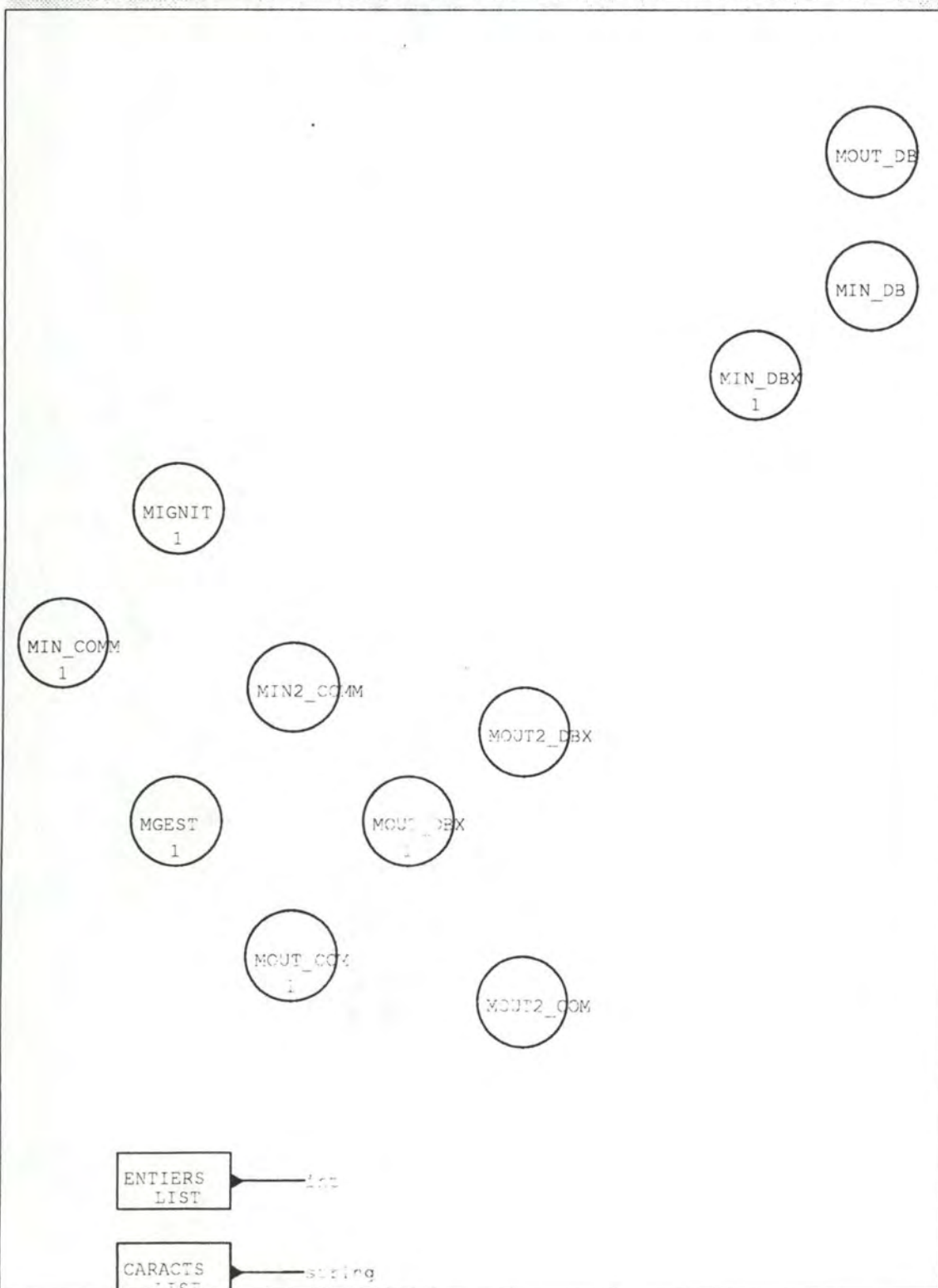
send_etranger.entiers := entiers

send_etranger.caracts := caracts

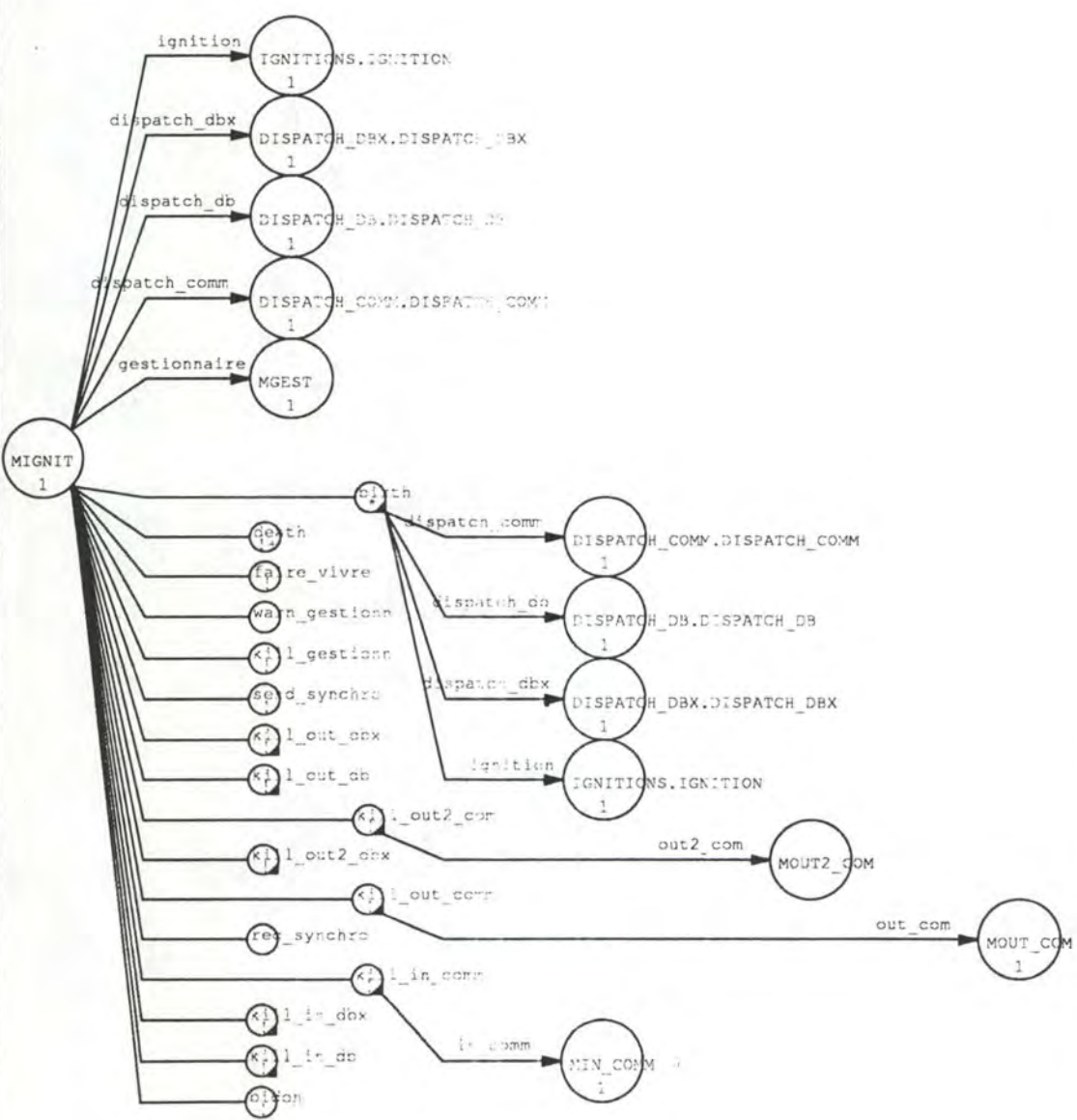
(13) **PID_mort**

Conditioned by: (PID_vit = "NON")

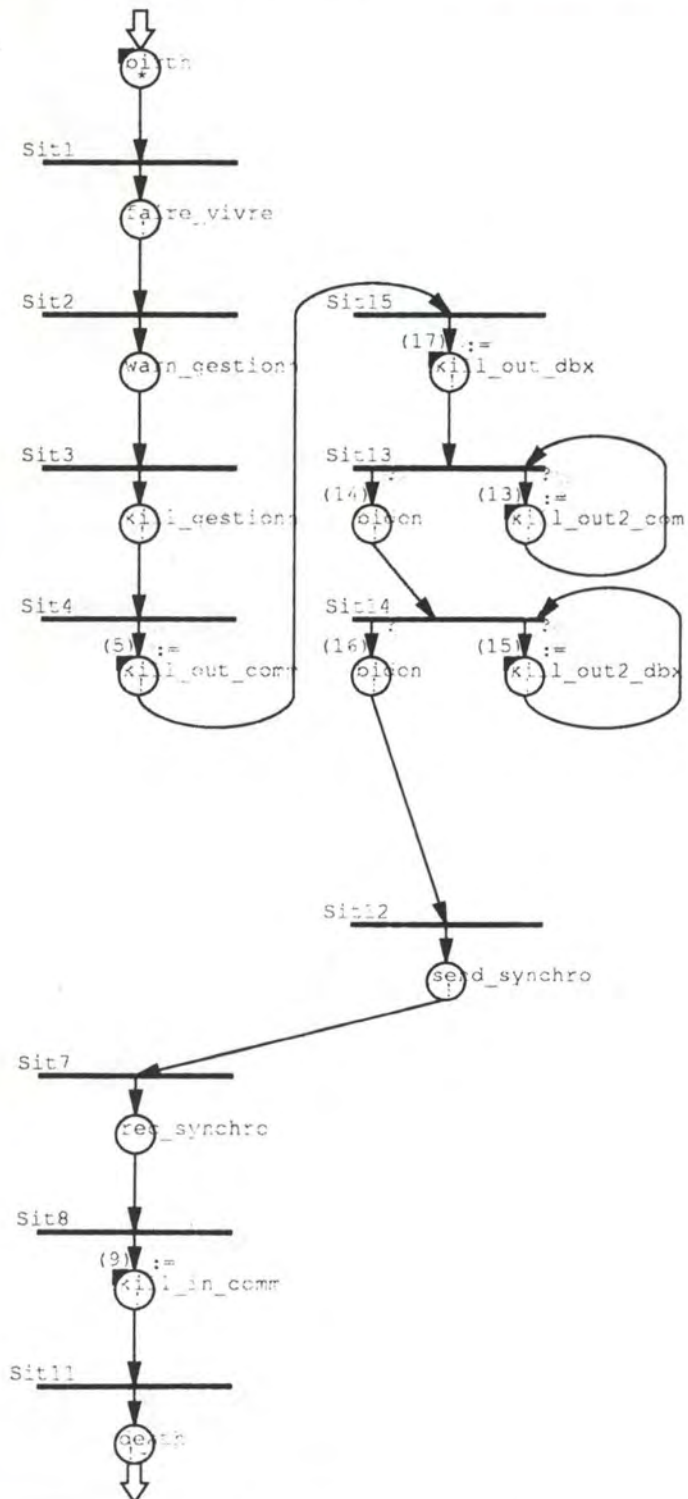
Community Diagram: MONITORING



Declaration Diagram: MONITORING\MIGNIT



Behavior Diagram: MONITORING\MIGNIT



Attribute Updates of object class MIGNIT are:

For Action birth

ignition := birth.ignition

dispatch_dbx := birth.dispatch_dbx

dispatch_db := birth.dispatch_db

dispatch_comm := birth.dispatch_comm

Calls of object class MIGNIT are:

NORMAL call

Caller action is: kill_in_comm

Conditioned by: TRUE

Called action is: MONITORING.MIN_COMM.warn

Identifying attribute is: kill_in_comm.in_comm

NORMAL call

Caller action is: kill_in_db

Conditioned by: TRUE

Called action is: MONITORING.MIN_DB.death

Identifying attribute is: kill_in_db.in_db

NORMAL call

Caller action is: faire_vivre

Conditioned by: TRUE

Called action is: MONITORING.MGEST.birth

Identifying attribute is: gestionnaire

Instantiations are:

birth.dispatch_db := dispatch_db

birth.dispatch_comm := dispatch_comm

birth.dispatch_dbx := dispatch_dbx

FOREIGNER call

Caller action is: send_synchro

Conditioned by: TRUE

Called action is: IGNITIONS.IGNITION.msynchro

Identifying attribute is: ignition

NORMAL call

Caller action is: kill_out_dbx

Conditioned by: TRUE

Called action is: MONITORING.MOUT_DBX.death

Identifying attribute is: kill_out_dbx.out_dbx

NORMAL call

Caller action is: kill_out_db

Conditioned by: TRUE

Called action is: MONITORING.MOUT_DB.death

Identifying attribute is: kill_out_db.out_db

NORMAL call

Caller action is: kill_in_dbx

Conditioned by: TRUE

Called action is: MONITORING.MIN_DBX.death

Identifying attribute is: kill_in_dbx.in_dbx

NORMAL call

Caller action is: kill_gestionn

Conditioned by: TRUE

Called action is: MONITORING.MGEST.death

Identifying attribute is: gestionnaire

NORMAL call

Caller action is: kill_out_comm

Conditioned by: TRUE

Called action is: MONITORING.MOUT_COM.death

Identifying attribute is: kill_out_comm.out_com

NORMAL call

Caller action is: warn_gestionn

Conditioned by: TRUE

Called action is: MONITORING.MGEST.rec_warn

Identifying attribute is: gestionnaire

NORMAL call

Caller action is: kill_out2_com

Conditioned by: TRUE

Called action is: MONITORING.MOUT2_COM.rec_mort

Identifying attribute is: kill_out2_com.out2_com

NORMAL call

Caller action is: kill_out2_dbx

Conditioned by: TRUE

Called action is: MONITORING.MOUT2_DBX.rec_mort

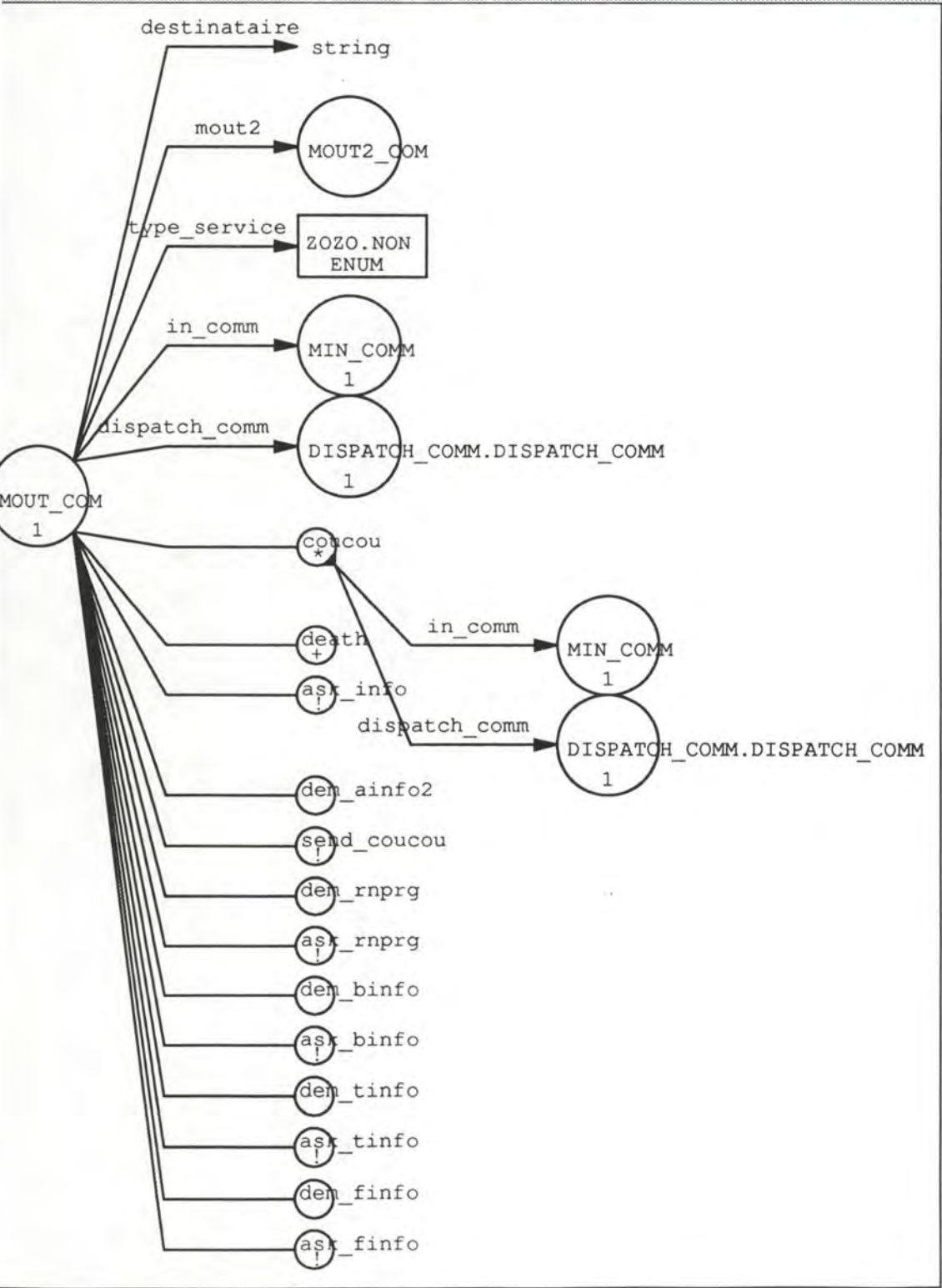
Identifying attribute is: kill_out2_dbx.out2_dbx

Behavior Conditions and Instantiations of object class MIGNIT are:

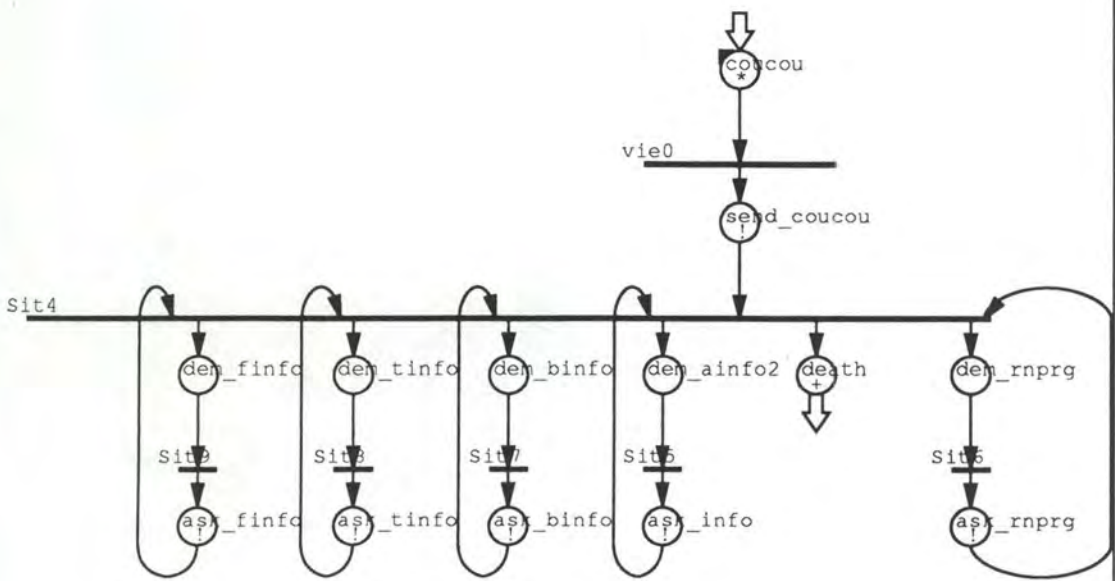
(16) **bidon**

- Conditioned by:* NOT(EXISTS[MOUT2_DBX:MONITORING|TRUE])
- (5) **kill_out_comm**
Conditioned by: TRUE
Instantiations are:
 kill_out_comm.out_com := ONE[MOUT_COM:MONITORING|TRUE]
- (13) **kill_out2_com**
Conditioned by: EXISTS[MOUT2_COM:MONITORING|TRUE]
Instantiations are:
 kill_out2_com.out2_com := ONE[MOUT2_COM:MONITORING|TRUE
]
- (14) **bidon**
Conditioned by: NOT(EXISTS[MOUT2_COM:MONITORING|TRUE])
- (15) **kill_out2_dbx**
Conditioned by: EXISTS[MOUT2_DBX:MONITORING|TRUE]
Instantiations are:
 kill_out2_dbx.out2_dbx := ONE[MOUT2_DBX:MONITORING|TRUE]
- (9) **kill_in_comm**
Conditioned by: TRUE
Instantiations are:
 kill_in_comm.in_comm := ONE[MIN_COMM:MONITORING|TRUE]
- (17) **kill_out_dbx**
Conditioned by: TRUE
Instantiations are:
 kill_out_dbx.out_dbx := ONE[MOUT_DBX:MONITORING|TRUE]

Declaration Diagram: MONITORING\MOUT_COM



Behavior Diagram: MONITORING\MOUT_COM



Attribute Updates of object class MOUT_COM are:

For Action dem_finfo

destinataire := "FRAISEUSE"

type_service := NON\$DEM_FINFO:ZOZO

For Action dem_binfo

type_service := NON\$DEM_BINFO:ZOZO

destinataire := "BRIDAGE"

For Action dem_rnprg

type_service := NON\$DEM_RNPRG:ZOZO

destinataire := "ROBOT"

For Action dem_ainfo2

destinataire := "AGV"

type_service := NON\$DEM_AINFO2:ZOZO

For Action dem_tinfo

type_service := NON\$DEM_TINFO:ZOZO

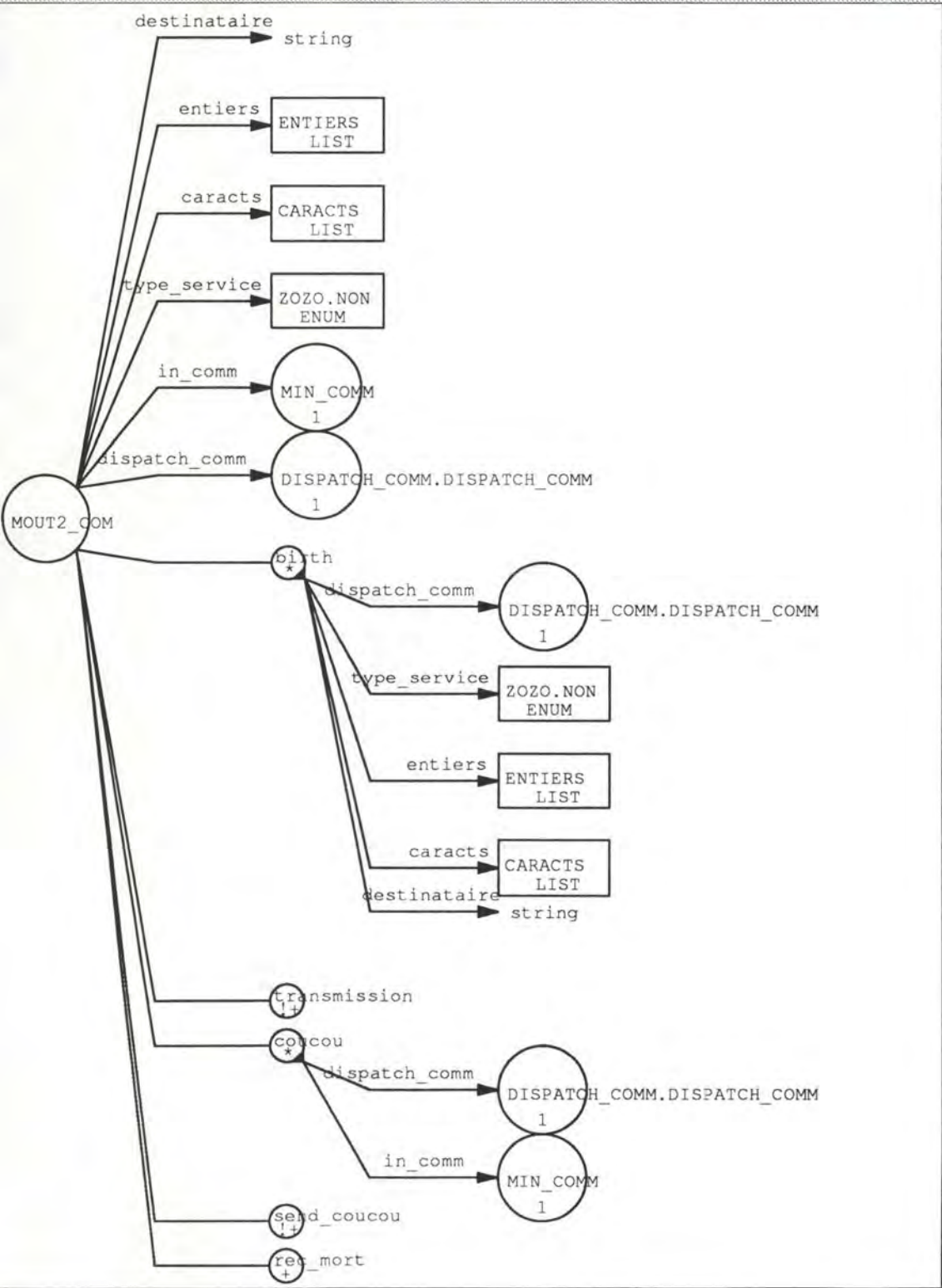
destinataire := "TOUR"

For Action coucou

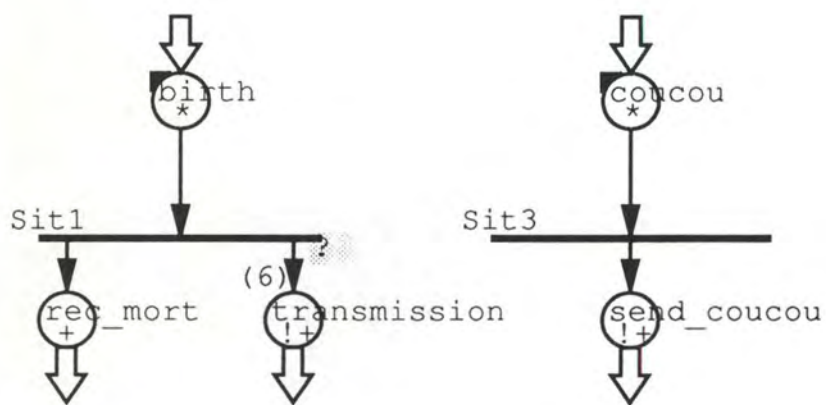
dispatch_comm := coucou.dispatch_comm

in_comm := coucou.in_comm

Declaration Diagram: MONITORING\MOUT2_COM



Behavior Diagram: MONITORING\MOUT2_COM



Attribute Updates of object class MOUT2_COM are:

For Action coucou

in_comm := coucou.in_comm

dispatch_comm := coucou.dispatch_comm

For Action birth

type_service := birth.type_service

destinataire := birth.destinataire

caracts := birth.caracts

entiers := birth.entiers

dispatch_comm := birth.dispatch_comm

Calls of object class MOUT2_COM are:

FOREIGNER call

Caller action is: **send_coucou**

Conditioned by: **TRUE**

Called action is: **DISPATCH_COMM.DISPATCH_COMM.QUI**

Identifying attribute is: **dispatch_comm**

Instantiations are:

QUI.monitoring := in_comm

FOREIGNER call

Caller action is: **transmission**

Conditioned by: **TRUE**

Called action is: **DISPATCH_COMM.DISPATCH_COMM.recevoir**

Identifying attribute is: **dispatch_comm**

Instantiations are:

recevoir.destinataire := destinataire

recevoir.type_service := type_service

recevoir.entiers := entiers

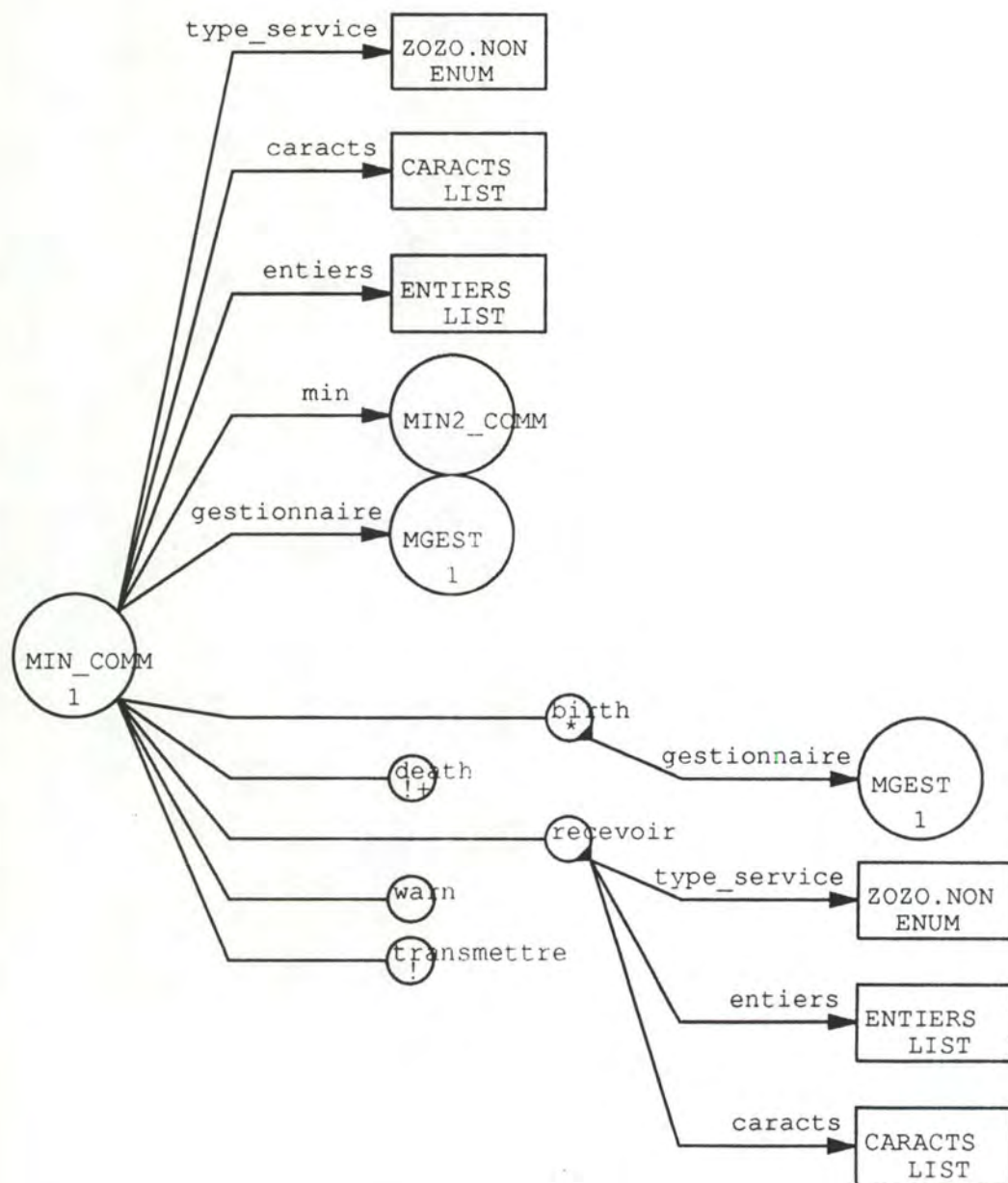
recevoir.caracts := caracts

Behavior Conditions and Instantiations of object class MOUT2_COM are:

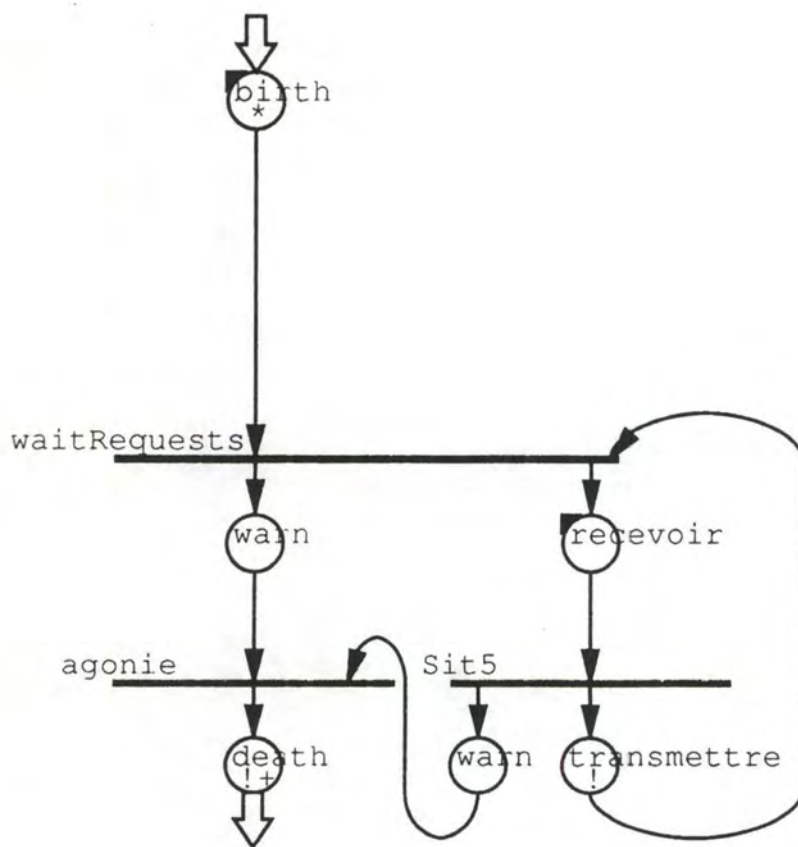
(6) transmission

Conditioned by: **NOT(EXISTS[DISPATCHING:DISPATCH_COMM|(type_service = SELF.type_service)])**

Declaration Diagram: MONITORING\MIN_COMM



Behavior Diagram: MONITORING\MIN_COMM



Attribute Updates of object class MIN_COMM are:

For Action recevoir

caracts := recevoir.caracts

entiers := recevoir.entiers

type_service := recevoir.type_service

For Action birth

gestionnaire := birth.gestionnaire

Calls of object class MIN_COMM are:

NORMAL call

Caller action is: transmettre

Conditioned by: TRUE

Called action is: MONITORING.MIN2_COMM.recevoir

Identifying attribute is: min

Instantiations are:

recevoir.caracts := caracts

recevoir.entiers := entiers

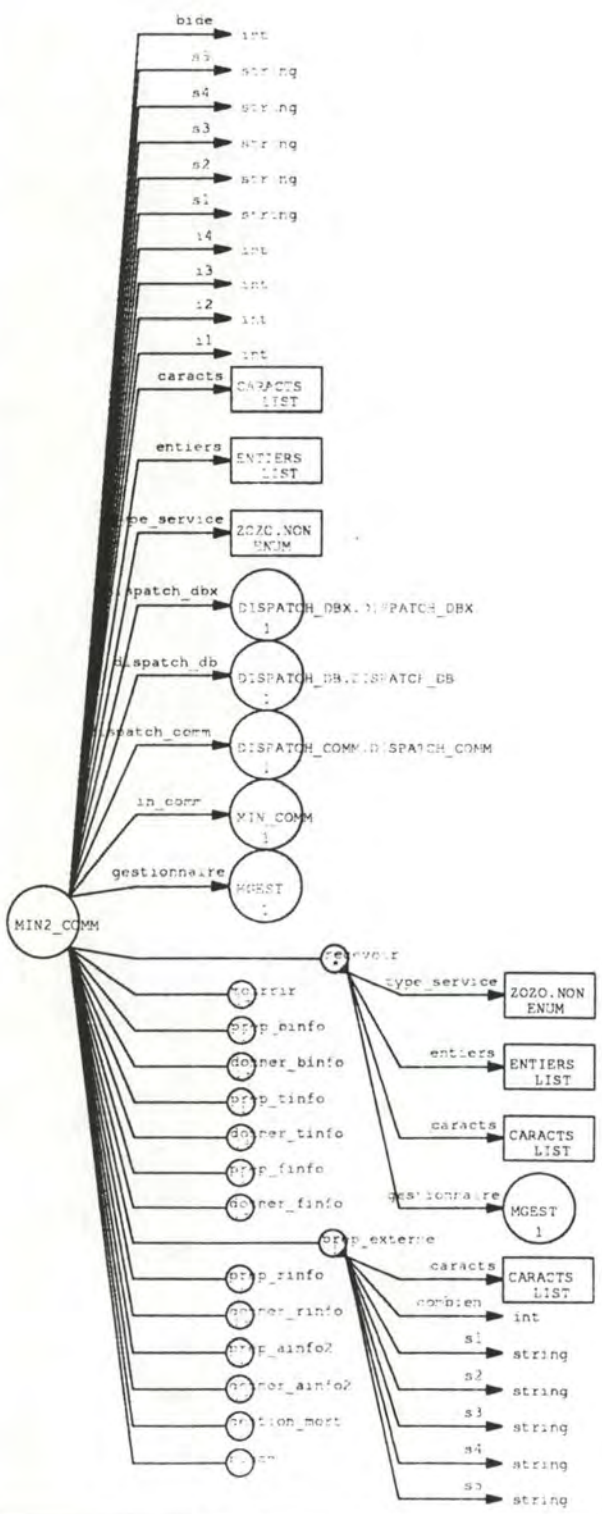
recevoir.type_service := type_service

recevoir.gestionnaire := gestionnaire

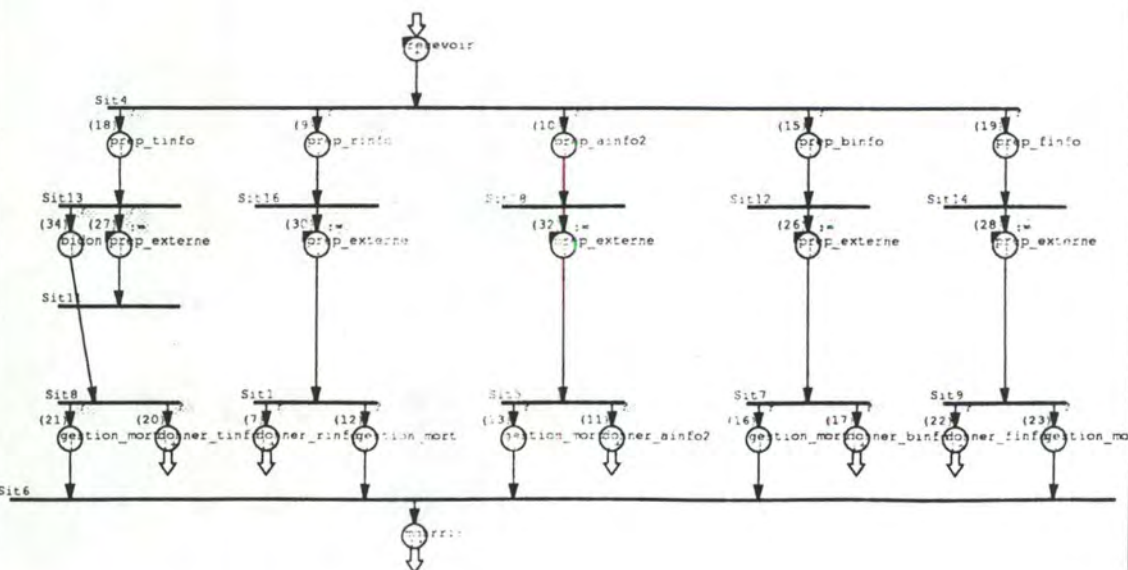
Behavior Conditions and Instantiations of object class MIN_COMM are:

There are no conditions and instantiations of this object class

Declaration Diagram: MONITORING\MIN2_COMM



Behavior Diagram: MONITORING\MIN2_COMM



Attribute Updates of object class **MIN2_COMM** are:

For Action **prep_binfo**

i1 := FETCH(entiers, 1)

i2 := FETCH(entiers, 2)

i3 := FETCH(entiers, 3)

For Action **prep_finfo**

i4 := FETCH(entiers, 4)

i3 := FETCH(entiers, 3)

i2 := FETCH(entiers, 2)

i1 := FETCH(entiers, 1)

For Action **prep_tinfo**

i4 := FETCH(entiers, 4)

i1 := FETCH(entiers, 1)

i2 := FETCH(entiers, 2)

i3 := FETCH(entiers, 3)

For Action **recevoir**

caracts := recevoir.caracts

bide := 1

gestionnaire := recevoir.gestionnaire

type_service := recevoir.type_service

entiers := recevoir.entiers

For Action **prep_externe**

s1 := prep_externe.s1

s3 := prep_externe.s3

s5 := prep_externe.s5

s2 := prep_externe.s2

s4 := prep_externe.s4

For Action **prep_ainfo2**

i2 := FETCH(entiers, 2)

i1 := FETCH(entiers, 1)

For Action **prep_rinfo**

i1 := FETCH(entiers, 1)

i4 := FETCH(entiers, 4)

i3 := FETCH(entiers, 3)

i2 := FETCH(entiers, 2)

Calls of object class **MIN2_COMM** are:

NORMAL call

Caller action is: **donner_finfo**

Conditioned by: **TRUE**

Called action is: **MONITORING.MGEST.rec_finfo**

Identifying attribute is: **gestionnaire**

Instantiations are:

rec_finfo.x_pos := i2
rec_finfo.y_pos := i3
rec_finfo.dock_state := s2
rec_finfo.type_prg := s5
rec_finfo.z_pos := i4
rec_finfo.nom_fichier := s1
rec_finfo.status := s4
rec_finfo.pallet_post := s3
rec_finfo.nprg := i1

NORMAL call

Caller action is: **donner_tinfo**

Conditioned by: **TRUE**

Called action is: **MONITORING.MGEST.rec_tinfo**

Identifying attribute is: **gestionnaire**

Instantiations are:

rec_tinfo.status := s4
rec_tinfo.nprg := i1
rec_tinfo.z_pos := i4
rec_tinfo.pallet_post := s3
rec_tinfo.dock_state := s2
rec_tinfo.y_pos := i3
rec_tinfo.nom_fichier := s1
rec_tinfo.x_pos := i2
rec_tinfo.type_prg := s5

NORMAL call

Caller action is: **donner_ainfo2**

Conditioned by: **TRUE**

Called action is: **MONITORING.MGEST.rec_ainfo2**

Identifying attribute is: **gestionnaire**

Instantiations are:

rec_ainfo2.dock_state := s2

rec_ainfo2.status := s5
rec_ainfo2.distance := i1
rec_ainfo2.direction := i2
rec_ainfo2.speed := s1
rec_ainfo2.pallet_pos := s3
rec_ainfo2.station := s4

NORMAL call

Caller action is: **donner_binfo**

Conditioned by: **TRUE**

Called action is: **MONITORING.MGEST.rec_binfo**

Identifying attribute is: **gestionnaire**

Instantiations are:

rec_binfo.dock_state := s1
rec_binfo.x_pos := i2
rec_binfo.status := s2
rec_binfo.speed := i3
rec_binfo.y_pos := i1

NORMAL call

Caller action is: **donner_rinfo**

Conditioned by: **TRUE**

Called action is: **MONITORING.MGEST.rec_rinfo**

Identifying attribute is: **gestionnaire**

Instantiations are:

rec_rinfo.x_pos := i2
rec_rinfo.y_pos := i3
rec_rinfo.z_pos := i4
rec_rinfo.status := s2
rec_rinfo.num_prg := i1
rec_rinfo.nom_fichier := s1

TO EXTERIOR call

Caller action is: **prep_externe**

OUT Instantiations are:

caracts
combien

IN Instantiations are:

s5

s2
s1
s3
s4

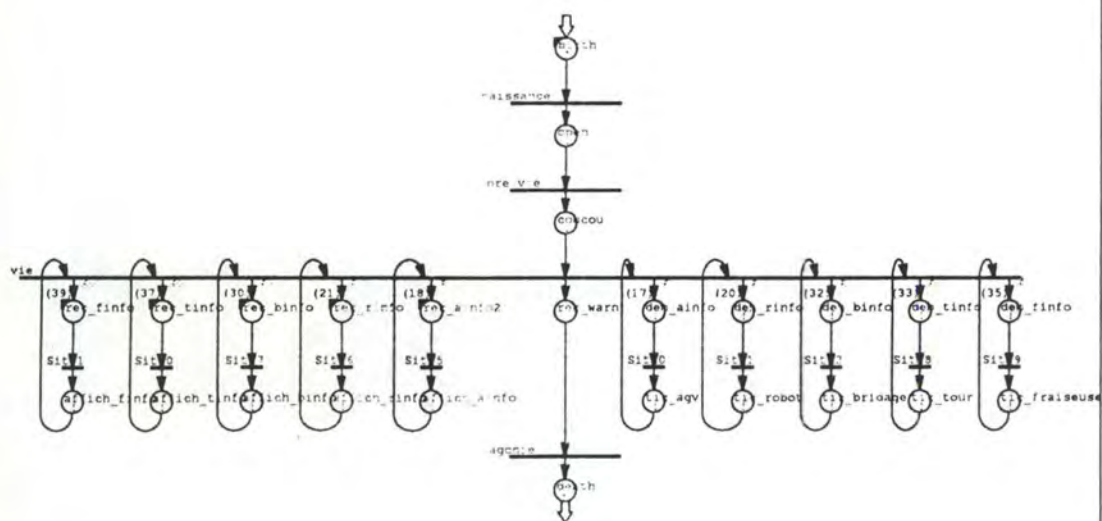
Behavior Conditions and Instantiations of object class MIN2_COMM are:

- (19) **prep_fininfo**
Conditioned by: (type_service = NON\$RET_FININFO:ZOZO)
- (22) **donner_fininfo**
Conditioned by: EXISTS[MGEST:MONITORING|TRUE]
- (30) **prep_externe**
Conditioned by: TRUE
Instantiations are:
 - prep_externe.caracts := caracts**
 - prep_externe.combien := 2**
- (10) **prep_ainfo2**
Conditioned by: (type_service = NON\$RET_AINFO2:ZOZO)
- (13) **gestion_mort**
Conditioned by: NOT(EXISTS[MGEST:MONITORING|TRUE])
- (20) **donner_tinfo**
Conditioned by: EXISTS[MGEST:MONITORING|TRUE]
- (11) **donner_ainfo2**
Conditioned by: EXISTS[MGEST:MONITORING|TRUE]
- (12) **gestion_mort**
Conditioned by: NOT(EXISTS[MGEST:MONITORING|TRUE])
- (21) **gestion_mort**
Conditioned by: NOT(EXISTS[MGEST:MONITORING|TRUE])
- (27) **prep_externe**
Conditioned by: (bide = 0)
Instantiations are:
 - prep_externe.combien := 5**
 - prep_externe.caracts := caracts**
- (18) **prep_tinfo**
Conditioned by: (type_service = NON\$RET_TINFO:ZOZO)
- (7) **donner_rinfo**
Conditioned by: EXISTS[MGEST:MONITORING|TRUE]
- (9) **prep_rinfo**

- Conditioned by:* (type_service = NON\$RET_RNPRG:ZOZO)
- (34) **bidon**
Conditioned by: (bide <> 0)
- (32) **prep_externe**
Conditioned by: TRUE
Instantiations are:
 prep_externe.combien := 5
 prep_externe.caracts := caracts
- (26) **prep_externe**
Conditioned by: TRUE
Instantiations are:
 prep_externe.caracts := caracts
 prep_externe.combien := 2
- (15) **prep_binfo**
Conditioned by: (type_service = NON\$RET_BINFO:ZOZO)
- (16) **gestion_mort**
Conditioned by: NOT(EXISTS[MGEST:MONITORING|TRUE])
- (17) **donner_binfo**
Conditioned by: EXISTS[MGEST:MONITORING|TRUE]
- (28) **prep_externe**
Conditioned by: TRUE
Instantiations are:
 prep_externe.caracts := caracts
 prep_externe.combien := 5
- (23) **gestion_mort**
Conditioned by: NOT(EXISTS[MGEST:MONITORING|TRUE])

Declaration Diagram: MONITORING\MGEST

Behavior Diagram: MONITORING\MGEST



Attribute Updates of object class MGEST are:

For Action rec_finfo

```
i3 := rec_finfo.y_pos  
s4 := rec_finfo.status  
s3 := rec_finfo.pallet_post  
s5 := rec_finfo.type_prg  
vals_de_qui := "FRAISEUSE"  
i1 := rec_finfo.nprg  
s2 := rec_finfo.dock_state  
i4 := rec_finfo.z_pos  
flag_fraise := 0  
s1 := rec_finfo.nom_fichier  
i2 := rec_finfo.x_pos
```

For Action rec_ainfo2

```
vals_de_qui := "AGV"  
s1 := rec_ainfo2.speed  
i1 := rec_ainfo2.distance  
flag_agv := 0  
s4 := rec_ainfo2.station  
i2 := rec_ainfo2.direction  
s3 := rec_ainfo2.pallet_pos  
s5 := rec_ainfo2.status  
s2 := rec_ainfo2.dock_state
```

For Action birth

```
dispatch_db := birth.dispatch_db  
dispatch_comm := birth.dispatch_comm  
moi := SELF  
dispatch_dbx := birth.dispatch_dbx
```

For Action rec_rinfo

```
i3 := rec_rinfo.y_pos  
flag_robot := 0  
i1 := rec_rinfo.num_prg  
i4 := rec_rinfo.z_pos  
i2 := rec_rinfo.x_pos  
vals_de_qui := "ROBOT"  
s2 := rec_rinfo.status  
s1 := rec_rinfo.nom_fichier
```

For Action tic_fraiseuse

flag_fraise := 1

For Action tic_tour

flag_tour := 1

For Action tic_robot

flag_robot := 1

For Action rec_binfo

i2 := rec_binfo.x_pos

s1 := rec_binfo.dock_state

i3 := rec_binfo.speed

i1 := rec_binfo.y_pos

s2 := rec_binfo.status

vals_de_qui := "BRIDAGE"

flag_bridage := 0

For Action tic_agv

flag_agv := 1

For Action rec_tinfo

flag_tour := 0

vals_de_qui := "TOUR"

s4 := rec_tinfo.status

s5 := rec_tinfo.type_prg

i2 := rec_tinfo.x_pos

i1 := rec_tinfo.nprg

i3 := rec_tinfo.y_pos

s3 := rec_tinfo.pallet_post

i4 := rec_tinfo.z_pos

s1 := rec_tinfo.nom_fichier

s2 := rec_tinfo.dock_state

For Action tic_bridage

flag_bridage := 1

Calls of object class MGEST are:

NORMAL call

Caller action is: dem_rinfo

Conditioned by: TRUE

Called action is: MONITORING.MOUT_COM.dem_rnprg

Identifying attribute is: out_comm

NORMAL call

Caller action is: **affich_tinfo**

Conditioned by: **TRUE**

Called action is: **MONITORING.MOUT_DBX.show_tinfo**

Identifying attribute is: **out_dbx**

Instantiations are:

show_tinfo.x_pos := i2

show_tinfo.type_prg := s5

show_tinfo.nom_fichier := s1

show_tinfo.z_pos := i4

show_tinfo.status := s4

show_tinfo.y_pos := i3

show_tinfo.nprg := i1

show_tinfo.dock_state := s2

show_tinfo.pallet_post := s3

NORMAL call

Caller action is: **affich_binfo**

Conditioned by: **TRUE**

Called action is: **MONITORING.MOUT_DBX.show_binfo**

Identifying attribute is: **out_dbx**

Instantiations are:

show_binfo.status := s2

show_binfo.dock_state := s1

show_binfo.speed := i3

show_binfo.x_pos := i2

show_binfo.y_pos := i1

NORMAL call

Caller action is: **coucou**

Conditioned by: **TRUE**

Called action is: **MONITORING.MOUT_DBX.coucou**

Identifying attribute is: **out_dbx**

Instantiations are:

coucou.dispatch_dbx := dispatch_dbx

NORMAL call

Caller action is: **coucou**

Conditioned by: **TRUE**

Called action is: **MONITORING.MOUT_COM.coucou**

Identifying attribute is: **out_comm**

Instantiations are:

coucou.in_comm := in

coucou.dispatch_comm := dispatch_comm

NORMAL call

Caller action is: **affich_ainfo**

Conditioned by: **TRUE**

Called action is: **MONITORING.MOUT_DBX.show_ainfo2**

Identifying attribute is: **out_dbx**

Instantiations are:

show_ainfo2.speed := s1

show_ainfo2.distance := i1

show_ainfo2.station := s4

show_ainfo2.status := s5

show_ainfo2.direction := i2

show_ainfo2.dock_state := s2

show_ainfo2.pallet_pos := s3

NORMAL call

Caller action is: **open**

Conditioned by: **TRUE**

Called action is: **MONITORING.MIN_COMM.birth**

Identifying attribute is: **in**

Instantiations are:

birth.gestionnaire := moi

NORMAL call

Caller action is: **dem_tinfo**

Conditioned by: **TRUE**

Called action is: **MONITORING.MOUT_COM.dem_tinfo**

Identifying attribute is: **out_comm**

NORMAL call

Caller action is: **affich_rinfo**

Conditioned by: **TRUE**

Called action is: **MONITORING.MOUT_DBX.show_rinfo**

Identifying attribute is: **out_dbx**

Instantiations are:

show_rinfo.y_pos := i3
show_rinfo.x_pos := i2
show_rinfo.z_pos := i4
show_rinfo.status := s2
show_rinfo.nom_fichier := s1
show_rinfo.num_prg := i1

NORMAL call

Caller action is: dem_finfo

Conditioned by: TRUE

Called action is: MONITORING.MOUT_COM.dem_finfo

Identifying attribute is: out_comm

NORMAL call

Caller action is: dem_ainfo

Conditioned by: TRUE

Called action is: MONITORING.MOUT_COM.dem_ainfo2

Identifying attribute is: out_comm

NORMAL call

Caller action is: affich_finfo

Conditioned by: TRUE

Called action is: MONITORING.MOUT_DBX.show_finfo

Identifying attribute is: out_dbx

Instantiations are:

show_finfo.z_pos := i4
show_finfo.pallet_post := s3
show_finfo.status := s4
show_finfo.dock_state := s2
show_finfo.nprg := i1
show_finfo.x_pos := i2
show_finfo.type_prg := s5
show_finfo.y_pos := i3
show_finfo.nom_fichier := s1

NORMAL call

Caller action is: dem_binfo

Conditioned by: TRUE

Called action is: MONITORING.MOUT_COM.dem_binfo

Identifying attribute is: out_comm

Behavior Conditions and Instantiations of object class MGEST are:

(17) **dem_ainfo**

Conditioned by: (NOT(EXISTS[MOUT2_COM:MONITORING|(type_service = NON\$DEM_AINFO2:ZOZO)]) AND (flag_agv = 0))

(33) **dem_tinfo**

Conditioned by: (NOT(EXISTS[MOUT2_COM:MONITORING|(type_service = NON\$DEM_TINFO:ZOZO)]) AND (flag_tour = 0))

(18) **rec_ainfo2**

Conditioned by: NOT(EXISTS[MOUT2_DBX:MONITORING|(type_service = SER_DISP_DBX\$AFFI_AINFO2:DISPATCH_DBX)])

(20) **dem_rinfo**

Conditioned by: (NOT(EXISTS[MOUT2_COM:MONITORING|(type_service = NON\$DEM_RNPRG:ZOZO)]) AND (flag_robot = 0))

(30) **rec_binfo**

Conditioned by: NOT(EXISTS[MOUT2_DBX:MONITORING|(type_service = SER_DISP_DBX\$AFFI_BINFO:DISPATCH_DBX)])

(37) **rec_tinfo**

Conditioned by: NOT(EXISTS[MOUT2_DBX:MONITORING|(type_service = SER_DISP_DBX\$AFFI_TINFO:DISPATCH_DBX)])

(21) **rec_rinfo**

Conditioned by: NOT(EXISTS[MOUT2_DBX:MONITORING|(type_service = SER_DISP_DBX\$AFFI_RNPRG:DISPATCH_DBX)])

(35) **dem_finfo**

Conditioned by: (NOT(EXISTS[MOUT2_COM:MONITORING|(type_service = NON\$DEM_FINFO:ZOZO)]) AND (flag_fraise = 0))

(32) **dem_binfo**

Conditioned by: (NOT(EXISTS[MOUT2_COM:MONITORING|(type_service = NON\$DEM_BINFO:ZOZO)]) AND (flag_bridge = 0))

(39) **rec_finfo**

Conditioned by: NOT(EXISTS[MOUT2_DBX:MONITORING|(type_service = SER_DISP_DBX\$AFFI_FINFO:DISPATCH_DBX)])

△ MONITORING

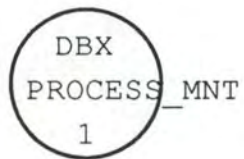
⊞ INTERFACE

☐ CTRL_MNT

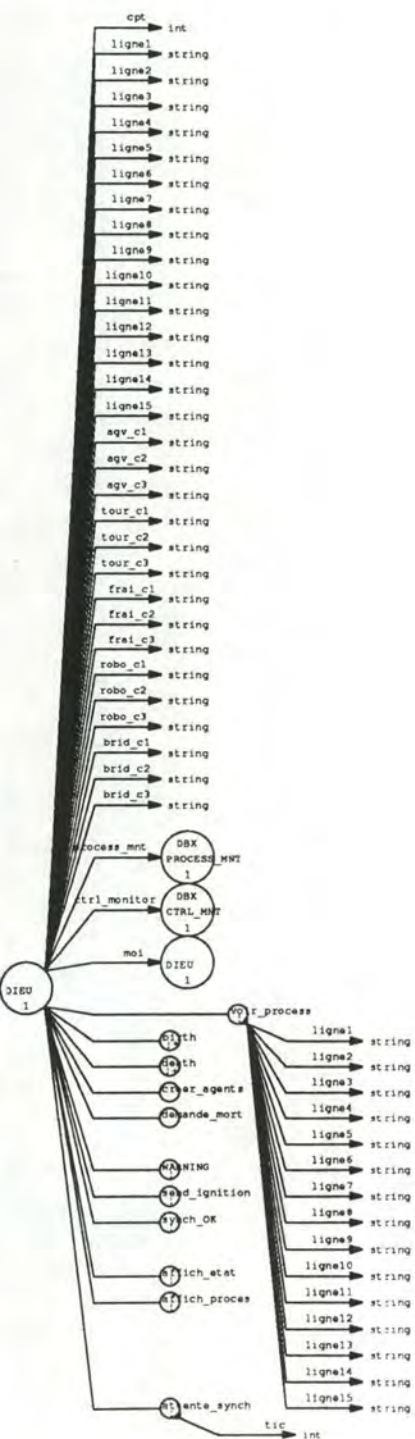
☐ DIEU

☐ PROCESS_MNT

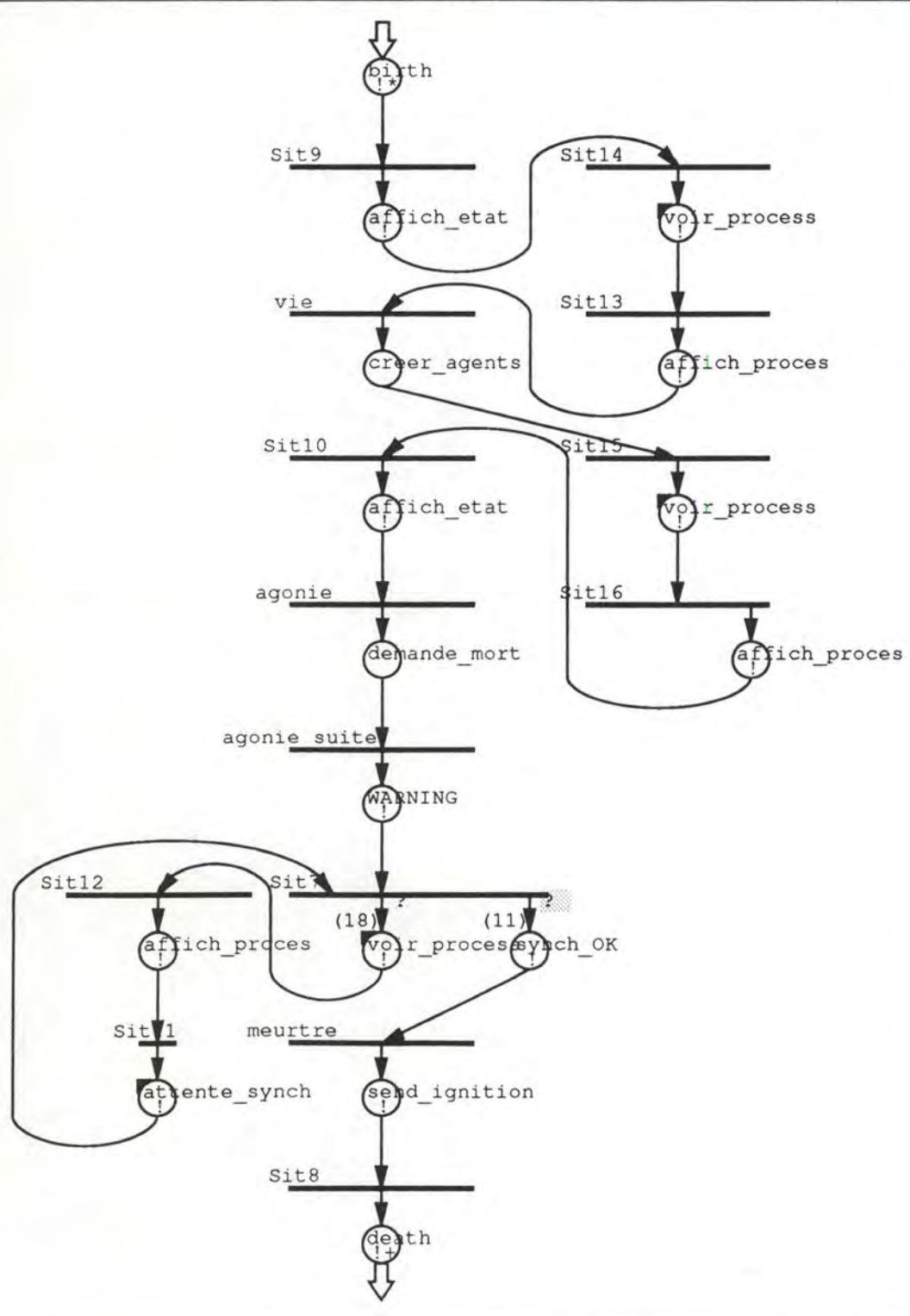
Community Diagram: INTERFACE



Declaration Diagram: INTERFACE\DIEU



Behavior Diagram: INTERFACE\DIEU



Attribute Updates of object class DIEU are:

For Action birth

```
moi := SELF
tour_c2 := "*****"
agv_c2 := "*****"
frai_c2 := "*****"
robo_c2 := "*****"
brid_c2 := "*****"
```

For Action WARNING

```
cpt := 0
```

For Action creer_agents

```
robo_c1 := "*****"
agv_c2 := " "
agv_c1 := "*****"
frai_c3 := " "
brid_c1 := "*****"
robo_c3 := " "
robo_c2 := " "
tour_c2 := " "
agv_c3 := " "
brid_c3 := " "
brid_c2 := " "
frai_c2 := " "
tour_c1 := "*****"
tour_c3 := " "
frai_c1 := "*****"
```

For Action attente_synch

```
cpt := (cpt + attente_synch.tic)
```

For Action voir_process

```
ligne5 := voir_process.ligne5
ligne6 := voir_process.ligne6
ligne1 := voir_process.ligne1
ligne10 := voir_process.ligne10
ligne11 := voir_process.ligne11
ligne8 := voir_process.ligne8
ligne14 := voir_process.ligne14
ligne9 := voir_process.ligne9
```


ligne15 := voir_process.ligne15
ligne7 := voir_process.ligne7
ligne13 := voir_process.ligne13
ligne3 := voir_process.ligne3
ligne4 := voir_process.ligne4
ligne2 := voir_process.ligne2
ligne12 := voir_process.ligne12

Calls of object class DIEU are:

NORMAL call

Caller action is: affich_etat

Conditioned by: TRUE

Called action is: INTERFACE.CTRL_MNT.affich_etat

Identifying attribute is: ctrl_monitor

Instantiations are:

affich_etat.frai_c1 := frai_c1
affich_etat.robo_c2 := robo_c2
affich_etat.robo_c1 := robo_c1
affich_etat.tour_c3 := tour_c3
affich_etat.agv_c3 := agv_c3
affich_etat.brid_c2 := brid_c2
affich_etat.tour_c2 := tour_c2
affich_etat.frai_c2 := frai_c2
affich_etat.frai_c3 := frai_c3
affich_etat.tour_c1 := tour_c1
affich_etat.robo_c3 := robo_c3
affich_etat.agv_c1 := agv_c1
affich_etat.agv_c2 := agv_c2
affich_etat.brid_c3 := brid_c3
affich_etat.brid_c1 := brid_c1

NORMAL call

Caller action is: affich_proces

Conditioned by: TRUE

Called action is: INTERFACE.PROCESS_MNT.afficher

Identifying attribute is: process_mnt

Instantiations are:

afficher.proc10 := ligne10

afficher.proc08 := ligne8
afficher.proc06 := ligne6
afficher.proc07 := ligne7
afficher.proc04 := ligne4
afficher.proc13 := ligne13
afficher.proc15 := ligne15
afficher.proc01 := ligne1
afficher.proc03 := ligne3
afficher.proc05 := ligne5
afficher.proc12 := ligne12
afficher.proc14 := ligne14
afficher.proc11 := ligne11
afficher.proc09 := ligne9
afficher.proc02 := ligne2

NORMAL call

Caller action is: death

Conditioned by: TRUE

Called action is: INTERFACE.PROCESS_MNT.close

Identifying attribute is: process_mnt

NORMAL call

Caller action is: death

Conditioned by: TRUE

Called action is: INTERFACE.CTRL_MNT.fin_monitor

Identifying attribute is: ctrl_monitor

TO EXTERIOR call

Caller action is: WARNING

TO EXTERIOR call

Caller action is: send_ignition

TO EXTERIOR call

Caller action is: creer_agents

TO EXTERIOR call

Caller action is: attente_synch

IN Instantiations are:

tic

TO EXTERIOR call

Caller action is: voir_process

IN Instantiations are:

ligne8
ligne15
ligne2
ligne6
ligne3
ligne14
ligne1
ligne9
ligne12
ligne7
ligne10
ligne11
ligne4
ligne5
ligne13

*Behavior Conditions and Instantiations of object class **DIEU** are:*

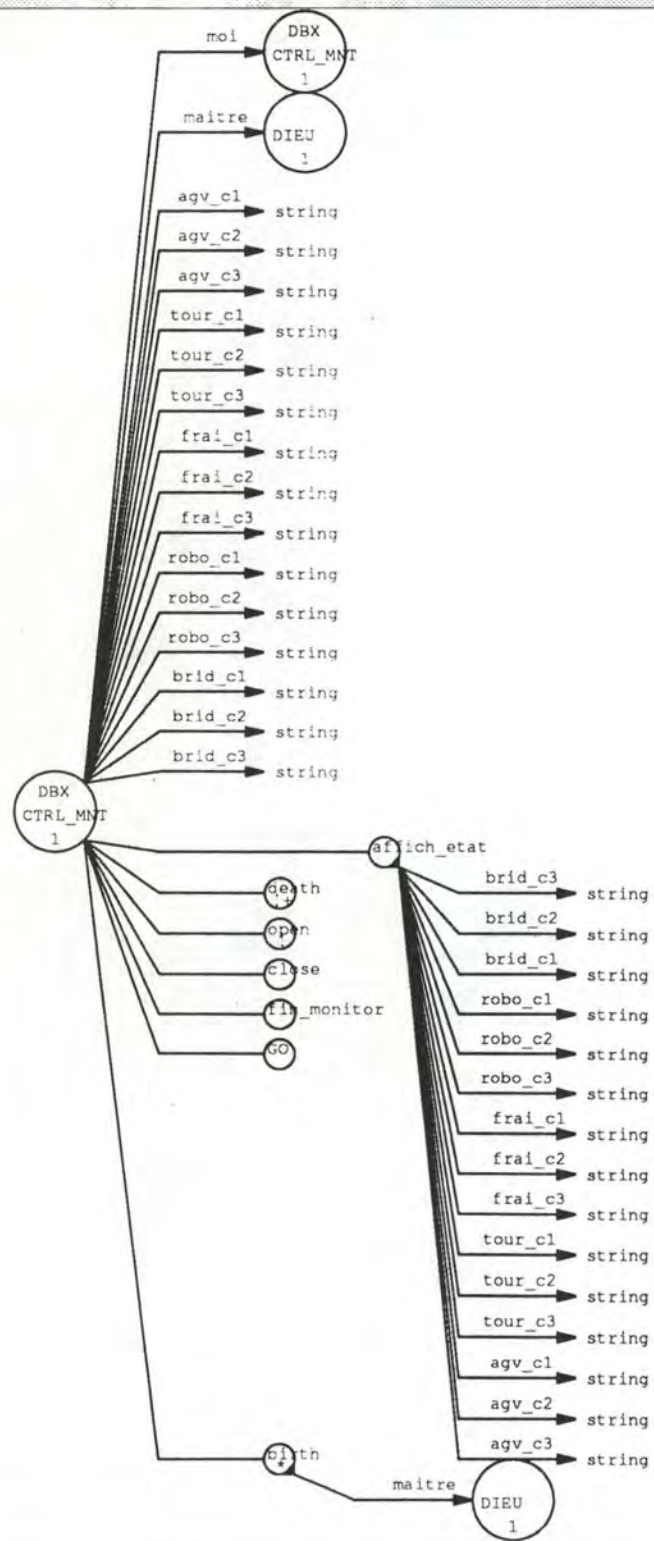
(18) **voir_process**

Conditioned by: (cpt < 3)

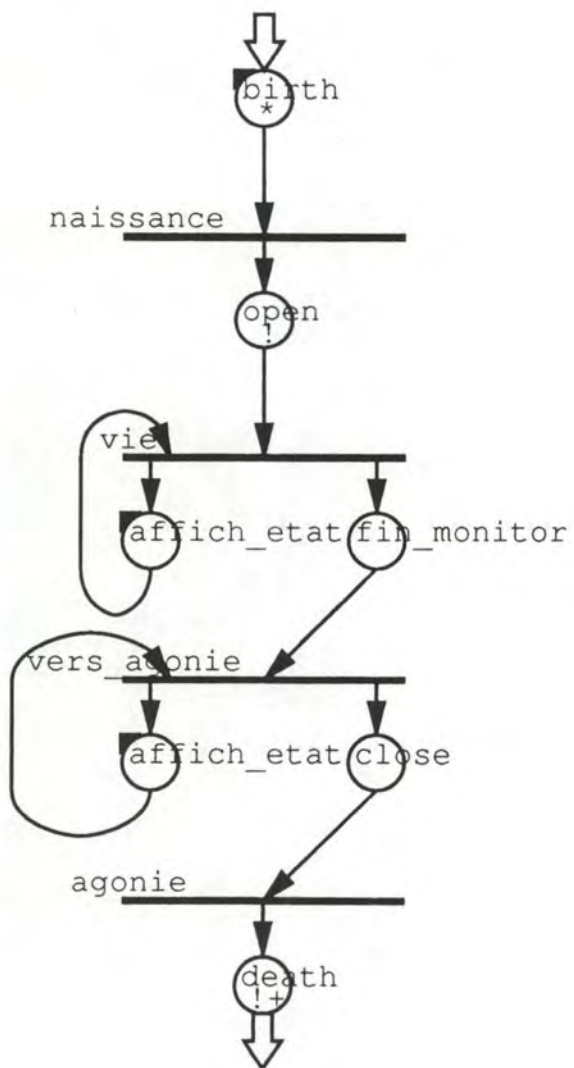
(11) **synch_OK**

Conditioned by: (cpt >= 3)

Declaration Diagram: INTERFACE\CTRL_MNT



Behavior Diagram: INTERFACE\CTRL_MNT



Attribute Updates of object class CTRL_MNT are:

For Action affich_etat

frai_c2 := affich_etat.frai_c2
brid_c2 := affich_etat.brid_c2
brid_c3 := affich_etat.brid_c3
agv_c3 := affich_etat.agv_c3
tour_c3 := affich_etat.tour_c3
brid_c1 := affich_etat.brid_c1
robo_c2 := affich_etat.robo_c2
tour_c2 := affich_etat.tour_c2
tour_c1 := affich_etat.tour_c1
frai_c1 := affich_etat.frai_c1
robo_c1 := affich_etat.robo_c1
agv_c2 := affich_etat.agv_c2
agv_c1 := affich_etat.agv_c1
frai_c3 := affich_etat.frai_c3
robo_c3 := affich_etat.robo_c3

For Action birth

moi := SELF
maitre := birth.maitre

Calls of object class CTRL_MNT are:

NORMAL call

Caller action is: GO
Conditioned by: TRUE
Called action is: INTERFACE.DIEU.creer_agents
Identifying attribute is: maitre

NORMAL call

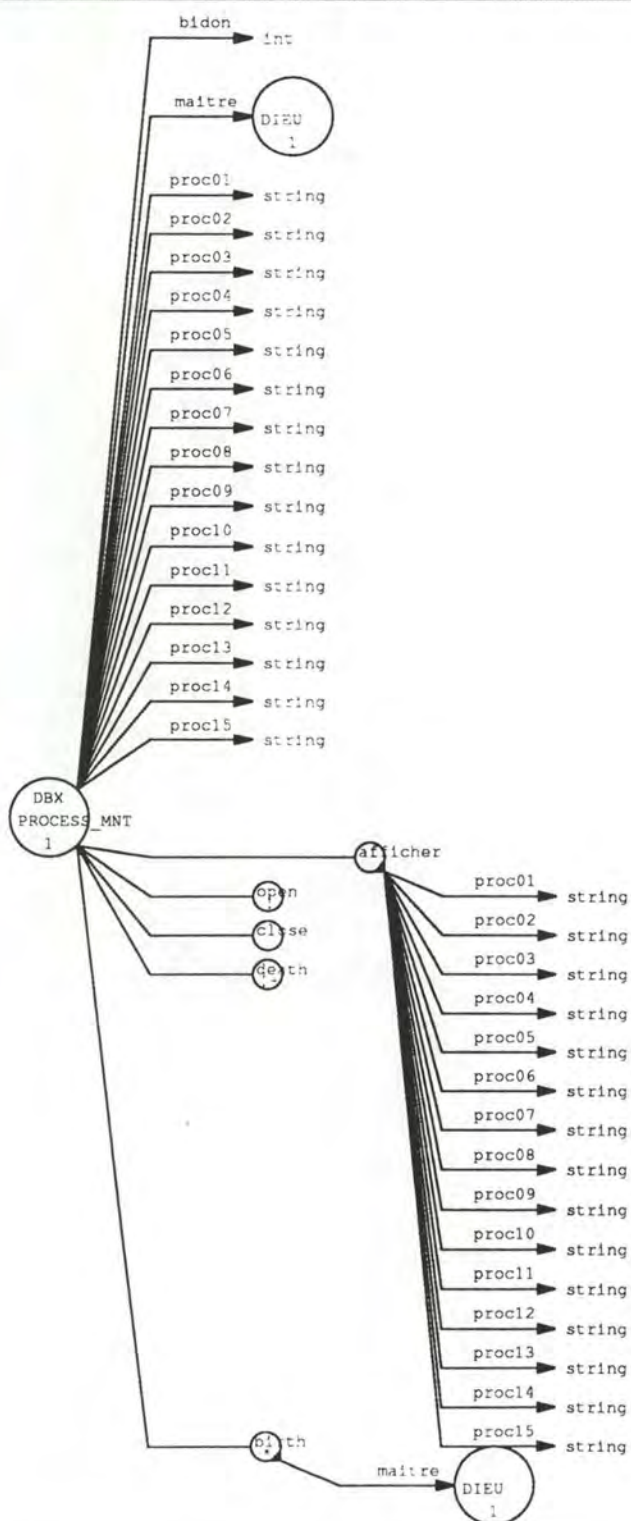
Caller action is: fin_monitor
Conditioned by: TRUE
Called action is: INTERFACE.DIEU.demande_mort
Identifying attribute is: maitre

Behavior Conditions and Instantiations of object class CTRL_MNT are:

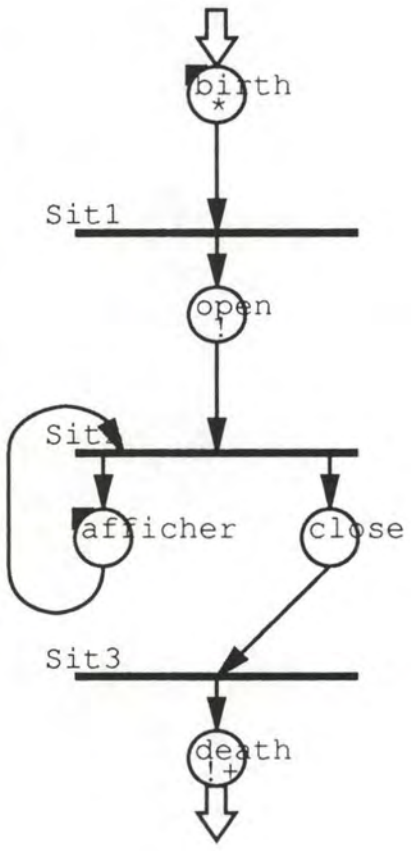
There are no conditions and instantiations of this object class

	Etat connexion		
	OK	TRYING	FAULT
Agv	-oof-	-oof-	-oof-
Tour	-oof-	-oof-	-oof-
raiseuse	-oof-	-oof-	-oof-
Robot	-oof-	-oof-	-oof-
Bridage	-oof-	-oof-	-oof-
Fin Monitoring	GO		

Declaration Diagram: INTERFACE\PROCESS_MNT



Behavior Diagram: INTERFACE\PROCESS_MNT



Attribute Updates of object class **PROCESS_MNT** *are:*

For Action **birth**

maitre := birth.maitre

For Action **afficher**

proc07 := afficher.proc07

proc06 := afficher.proc06

proc12 := afficher.proc12

proc14 := afficher.proc14

proc03 := afficher.proc03

proc01 := afficher.proc01

proc10 := afficher.proc10

proc15 := afficher.proc15

proc13 := afficher.proc13

proc09 := afficher.proc09

proc04 := afficher.proc04

proc02 := afficher.proc02

proc08 := afficher.proc08

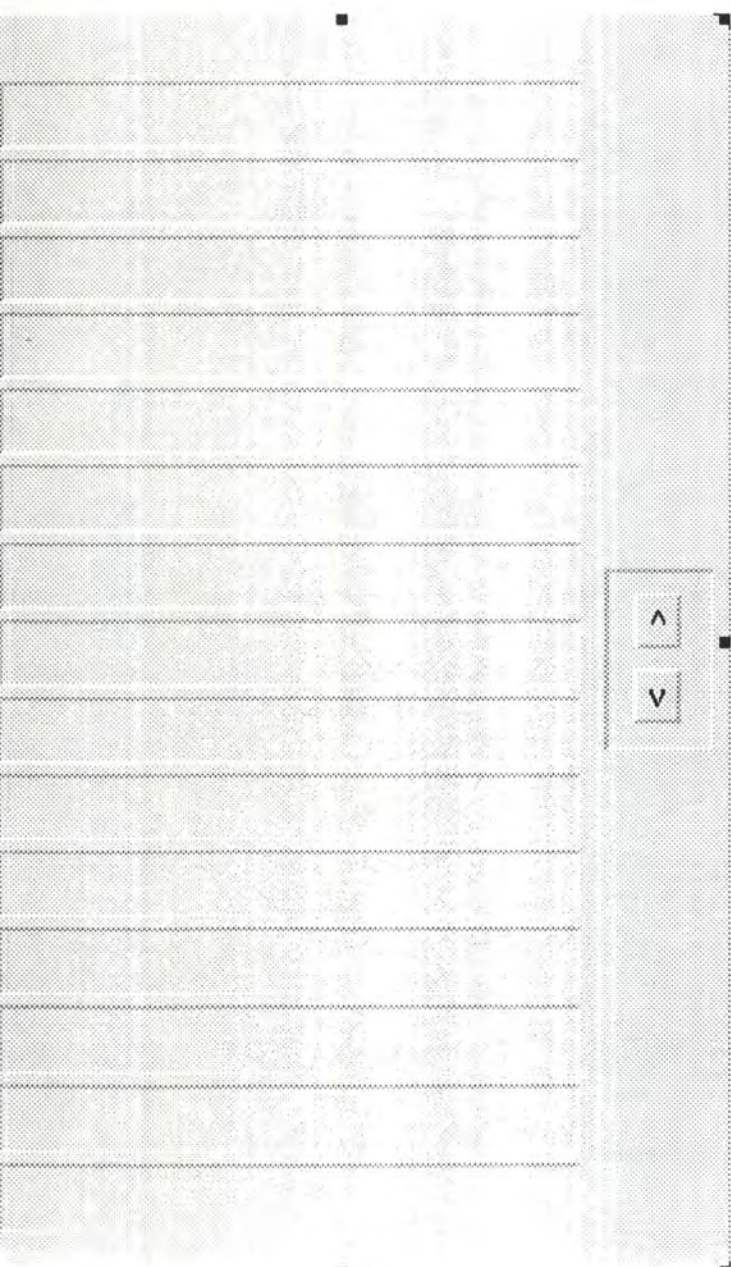
proc05 := afficher.proc05

proc11 := afficher.proc11

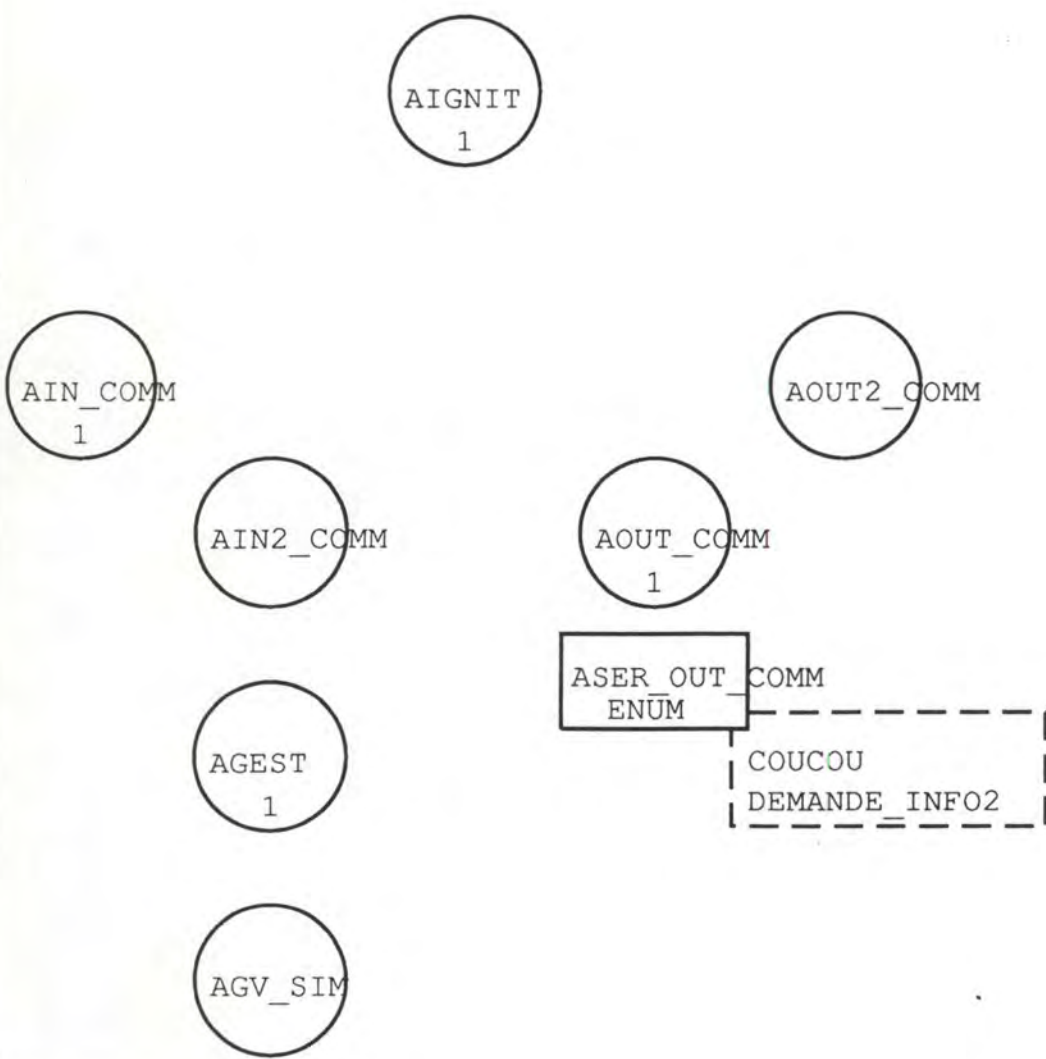
Calls of object class **PROCESS_MNT** *are:*

Behavior Conditions and Instantiations of object class **PROCESS_MNT** *are:*

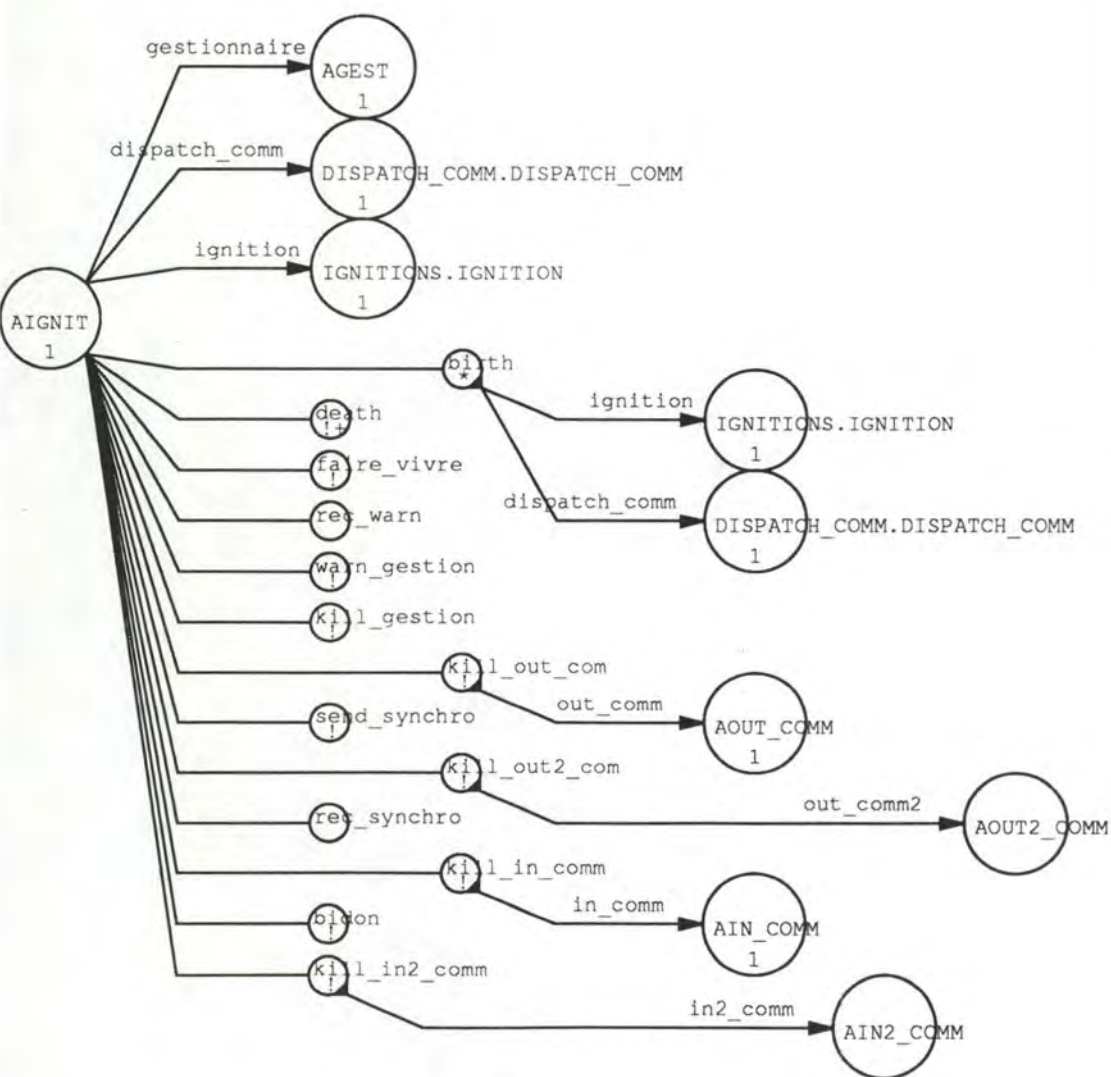
There are no conditions and instantiations of this object class



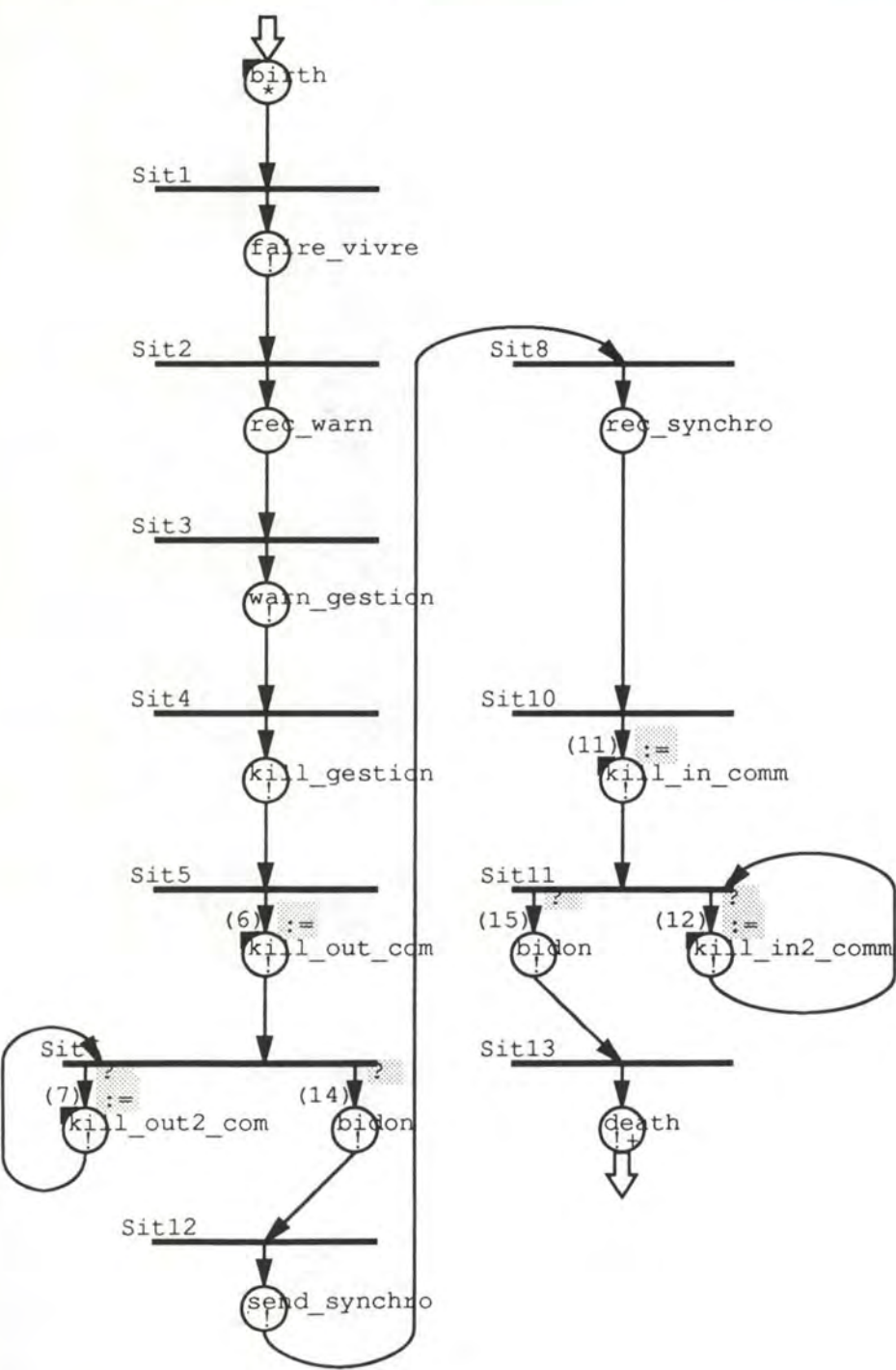
Community Diagram: AGV



Declaration Diagram: AGV\AIGNIT



Behavior Diagram: AGV\AIGNIT



Attribute Updates of object class AIGNIT are:

*For Action **birth***

ignition := birth.ignition

dispatch_comm := birth.dispatch_comm

Calls of object class AIGNIT are:

FOREIGNER call

*Caller action is: **send_synchro***

*Conditioned by: **TRUE***

*Called action is: **IGNITIONS.IGNITION.asynchro***

*Identifying attribute is: **ignition***

NORMAL call

*Caller action is: **warn_gestion***

*Conditioned by: **TRUE***

*Called action is: **AGV.AGEST.warn***

*Identifying attribute is: **gestionnaire***

NORMAL call

*Caller action is: **kill_gestion***

*Conditioned by: **TRUE***

*Called action is: **AGV.AGEST.death***

*Identifying attribute is: **gestionnaire***

NORMAL call

*Caller action is: **kill_out_com***

*Conditioned by: **TRUE***

*Called action is: **AGV.AOUT_COMM.death***

*Identifying attribute is: **kill_out_com.out_comm***

NORMAL call

*Caller action is: **kill_out2_com***

*Conditioned by: **TRUE***

*Called action is: **AGV.AOUT2_COMM.death***

*Identifying attribute is: **kill_out2_com.out_comm2***

NORMAL call

*Caller action is: **kill_in_comm***

*Conditioned by: **TRUE***

*Called action is: **AGV.AIN_COMM.death***

*Identifying attribute is: **kill_in_comm.in_comm***

NORMAL call

*Caller action is: **kill_in2_comm***

Conditioned by: **TRUE**

Called action is: **AGV.AIN2_COMM.death**

Identifying attribute is: **kill_in2_comm.in2_comm**

NORMAL call

Caller action is: **faire_vivre**

Conditioned by: **TRUE**

Called action is: **AGV.AGEST.birth**

Identifying attribute is: **gestionnaire**

Instantiations are:

birth.dispatch_comm := dispatch_comm

Behavior Conditions and Instantiations of object class AIGNIT are:

(12) **kill_in2_comm**

Conditioned by: **EXISTS[AIN2_COMM:AGV|TRUE]**

Instantiations are:

kill_in2_comm.in2_comm := ONE[AIN2_COMM:AGV|TRUE]

(6) **kill_out_com**

Conditioned by: **TRUE**

Instantiations are:

kill_out_com.out_comm := ONE[AOUT_COMM:AGV|TRUE]

(14) **bidon**

Conditioned by: **NOT(EXISTS[AOUT2_COMM:AGV|TRUE])**

(7) **kill_out2_com**

Conditioned by: **EXISTS[AOUT2_COMM:AGV|TRUE]**

Instantiations are:

kill_out2_com.out_comm2 := ONE[AOUT2_COMM:AGV|TRUE]

(15) **bidon**

Conditioned by: **NOT(EXISTS[AIN2_COMM:AGV|TRUE])**

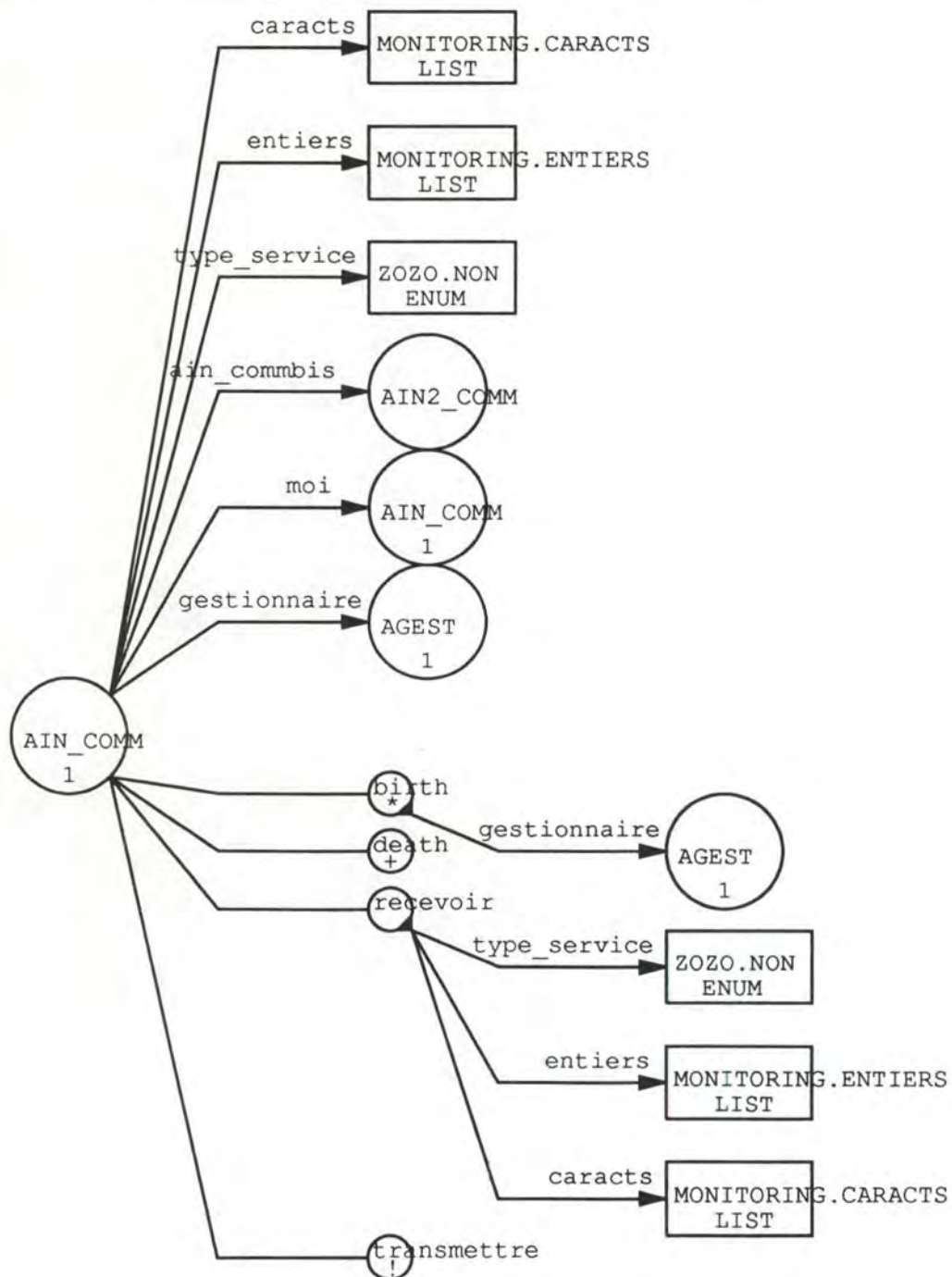
(11) **kill_in_comm**

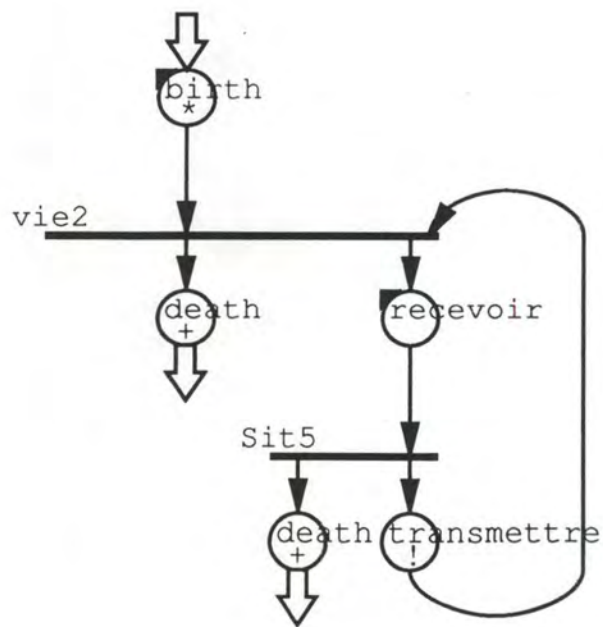
Conditioned by: **TRUE**

Instantiations are:

kill_in_comm.in_comm := ONE[AIN_COMM:AGV|TRUE]

Declaration Diagram: AGV\AIN_COMM





Attribute Updates of object class AIN_COMM are:

*For Action **recevoir***

type_service := recevoir.type_service

caracts := recevoir.caracts

entiers := recevoir.entiers

*For Action **birth***

gestionnaire := birth.gestionnaire

moi := SELF

Calls of object class AIN_COMM are:

NORMAL call

*Caller action is: **transmettre***

*Conditioned by: **TRUE***

*Called action is: **AGV.AIN2_COMM.recevoir***

*Identifying attribute is: **ain_commbis***

Instantiations are:

recevoir.entiers := entiers

recevoir.type_service := type_service

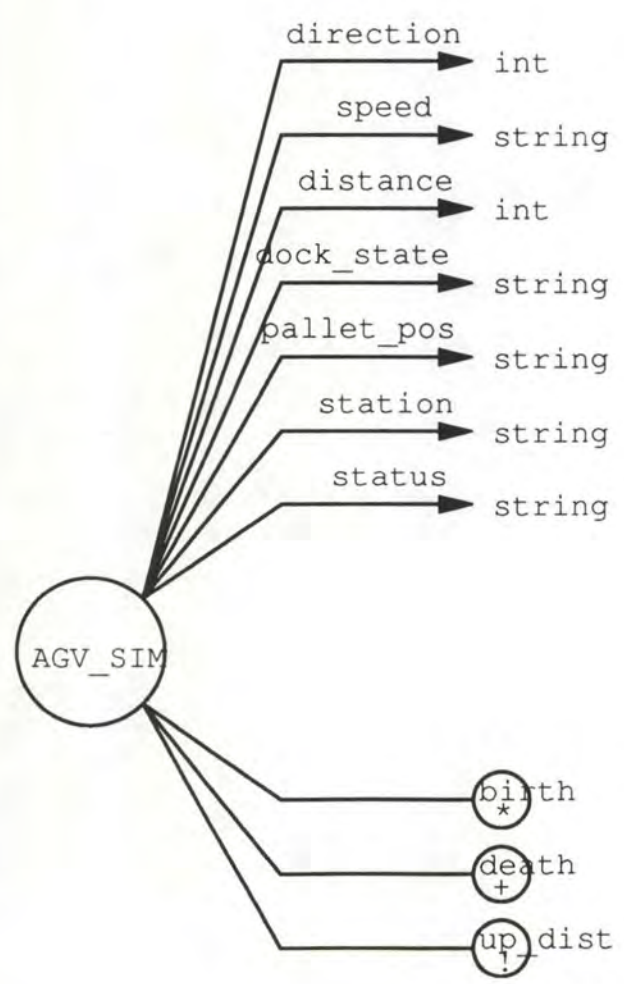
recevoir.gestionnaire := gestionnaire

recevoir.caracts := caracts

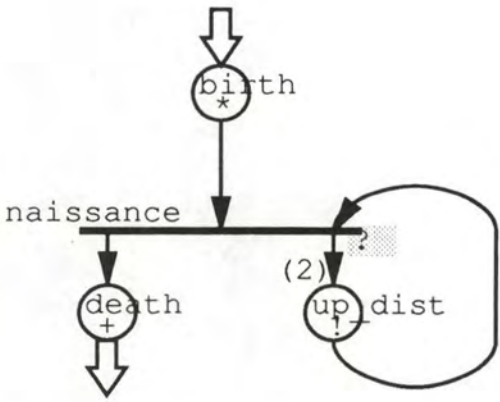
Behavior Conditions and Instantiations of object class AIN_COMM are:

There are no conditions and instantiations of this object class

Declaration Diagram: AGV\AGV_SIM



Behavior Diagram: AGV\AGV_SIM



Attribute Updates of object class AGV_SIM are:

For Action birth

distance := 0

pallet_pos := "A"

speed := "vitesse"

direction := 0

status := "Ca marche!!!"

dock_state := "etat"

station := "A"

For Action up_dist

distance := (distance + 1)

Calls of object class AGV_SIM are:

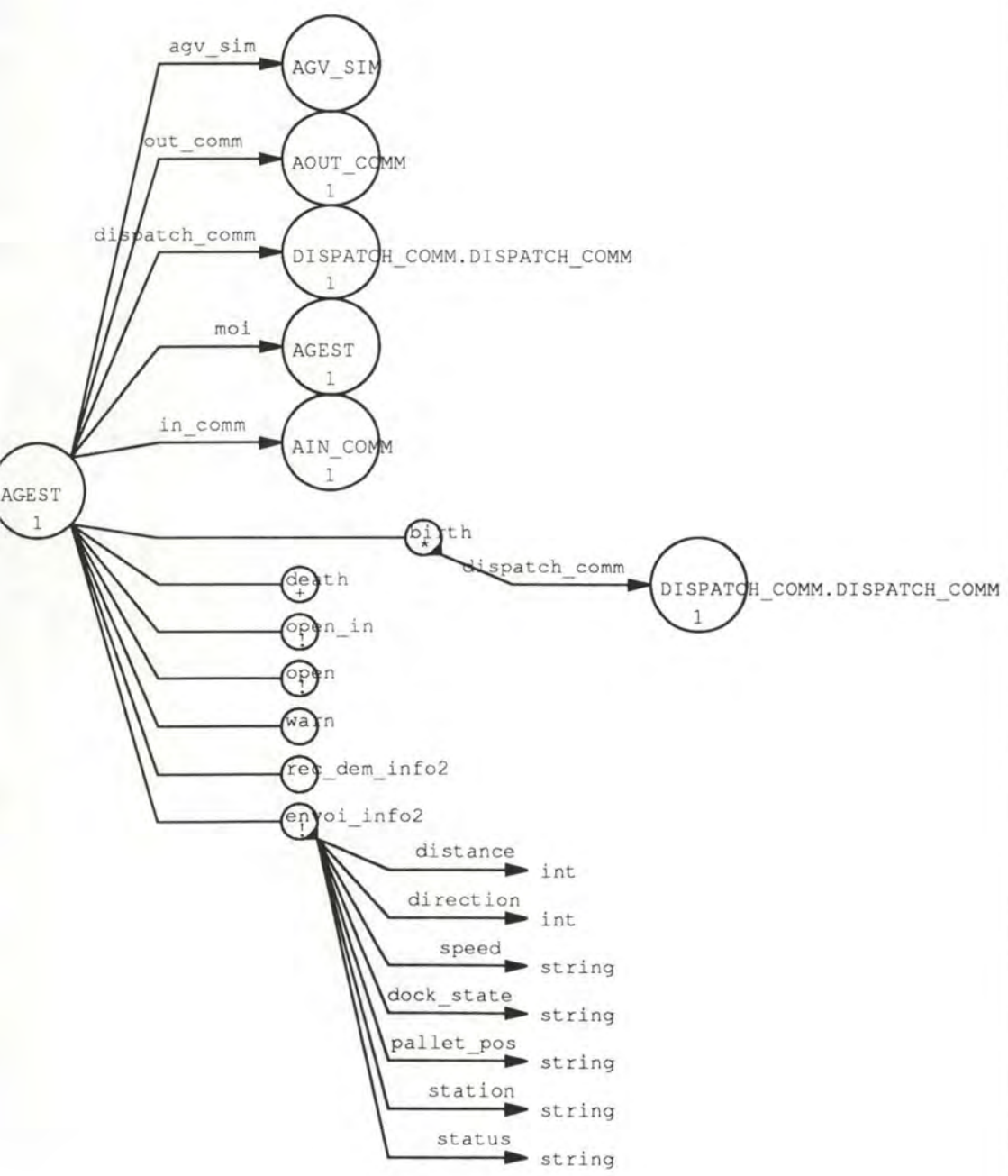
There are no calls of this object class

Behavior Conditions and Instantiations of object class AGV_SIM are:

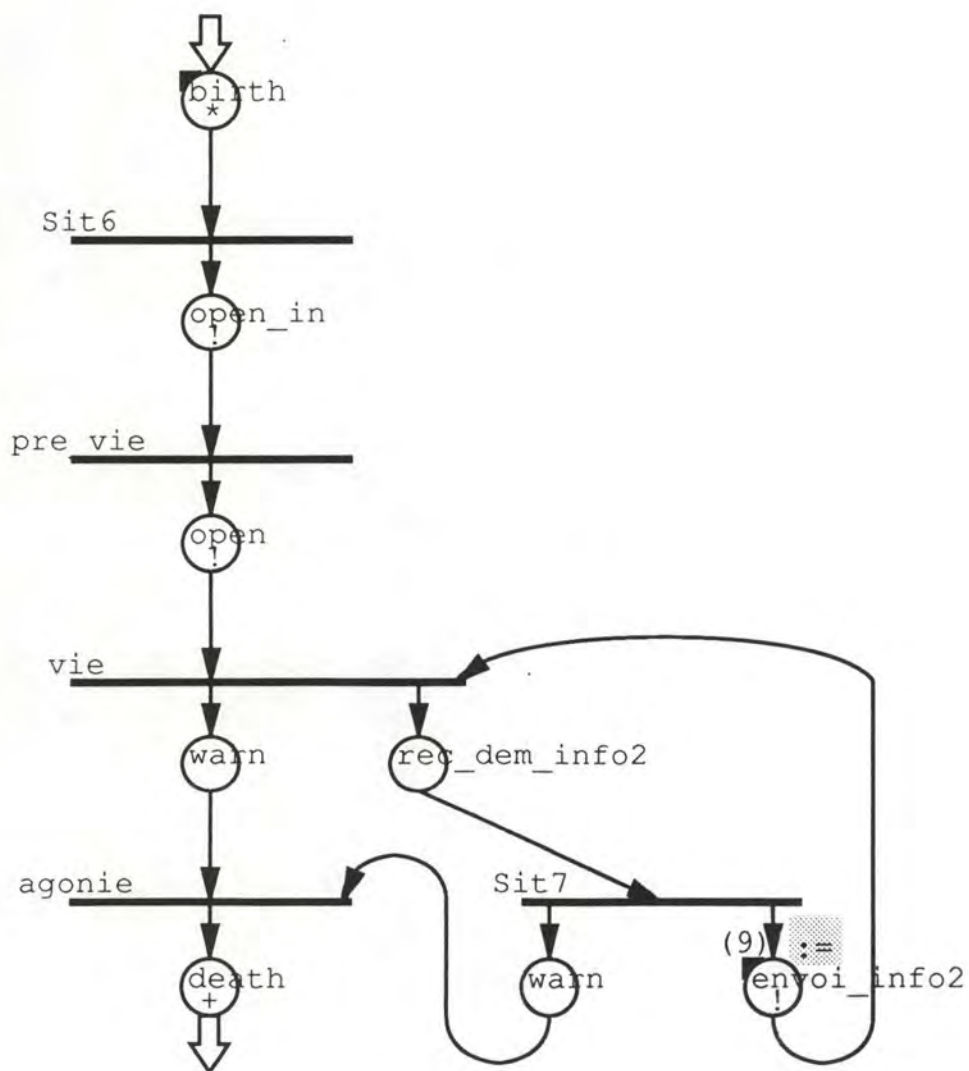
(2) up_dist

Conditioned by: (distance < 99999999)

Declaration Diagram: AGV\AGEST



Behavior Diagram: AGV\AGEST



Attribute Updates of object class AGEST are:

For Action birth

dispatch_comm := birth.dispatch_comm

moi := SELF

Calls of object class AGEST are:

NORMAL call

Caller action is: open_in

Conditioned by: TRUE

Called action is: AGV.AIN_COMM.birth

Identifying attribute is: in_comm

Instantiations are:

birth.gestionnaire := moi

NORMAL call

Caller action is: open

Conditioned by: TRUE

Called action is: AGV.AGV_SIM.birth

Identifying attribute is: agv_sim

NORMAL call

Caller action is: open

Conditioned by: TRUE

Called action is: AGV.AOUT_COMM.coucou

Identifying attribute is: out_comm

Instantiations are:

coucou.in_comm := in_comm

coucou.dispatch_comm := dispatch_comm

NORMAL call

Caller action is: warn

Conditioned by: TRUE

Called action is: AGV.AGV_SIM.death

Identifying attribute is: agv_sim

NORMAL call

Caller action is: envoi_info2

Conditioned by: TRUE

Called action is: AGV.AOUT_COMM.demande_info2

Identifying attribute is: out_comm

Instantiations are:

demande_info2.dock_state := envoi_info2.dock_state

demande_info2.direction := envoi_info2.direction
demande_info2.distance := envoi_info2.distance
demande_info2.station := envoi_info2.station
demande_info2.speed := envoi_info2.speed
demande_info2.pallet_pos := envoi_info2.pallet_pos
demande_info2.status := envoi_info2.status

Behavior Conditions and Instantiations of object class AGEST are:

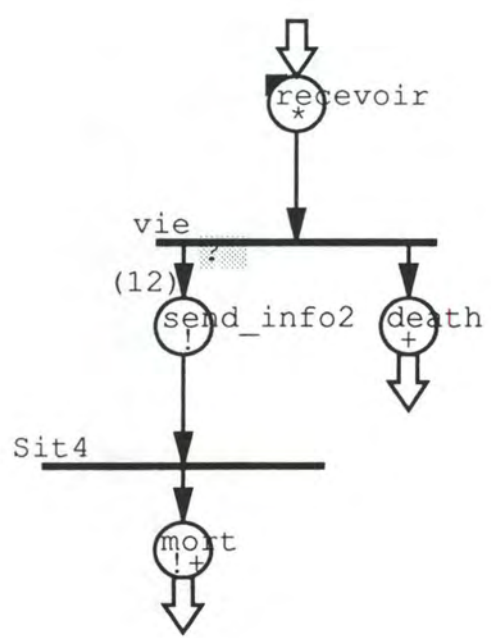
(9) **envoi_info2**

Conditioned by: **TRUE**

Instantiations are:

envoi_info2.dock_state := agv_sim.dock_state
envoi_info2.speed := agv_sim.speed
envoi_info2.distance := agv_sim.distance
envoi_info2.status := agv_sim.status
envoi_info2.pallet_pos := agv_sim.pallet_pos
envoi_info2.direction := agv_sim.direction
envoi_info2.station := agv_sim.station

Behavior Diagram: AGV\AIN2_COMM



Attribute Updates of object class AIN2_COMM are:

For Action recevoir

caracts := recevoir.caracts

type_service := recevoir.type_service

entiers := recevoir.entiers

gestionnaire := recevoir.gestionnaire

Calls of object class AIN2_COMM are:

NORMAL call

Caller action is: send_info2

Conditioned by: TRUE

Called action is: AGV.AGEST.rec_dem_info2

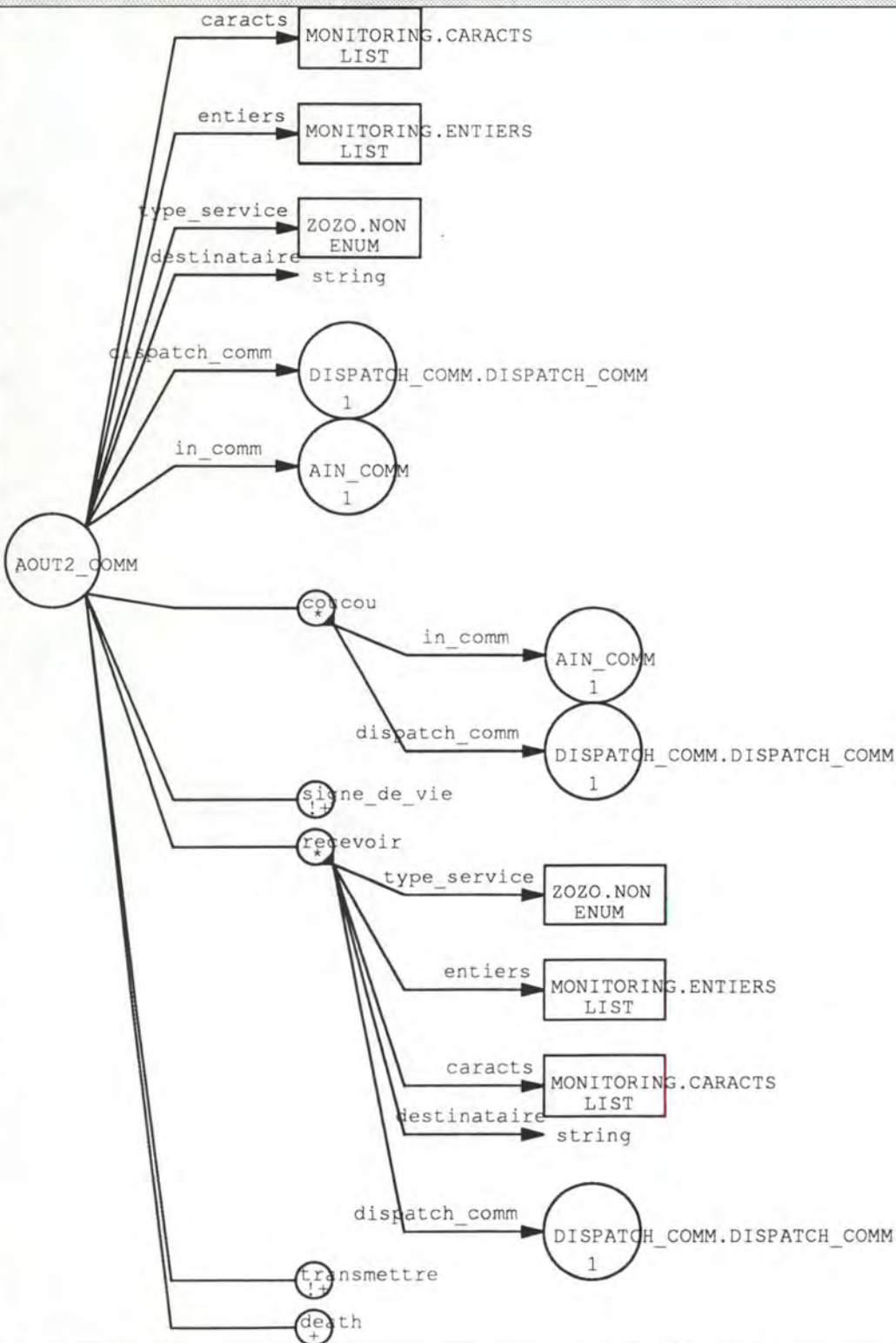
Identifying attribute is: gestionnaire

Behavior Conditions and Instantiations of object class AIN2_COMM are:

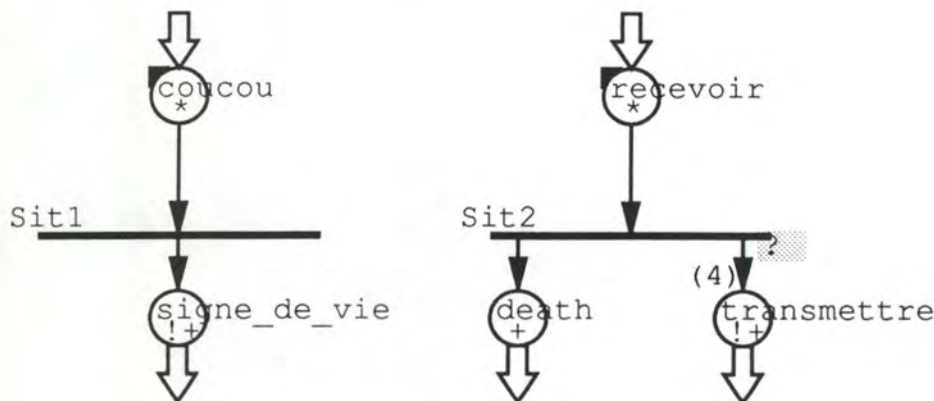
(12) send_info2

Conditioned by: ((type_service = NON\$DEM_AINFO2:ZOZO) AND EXISTS[AGEST:AGV|TRUE])

Declaration Diagram: AGV\AOUT2_COMM



Behavior Diagram: AGV\AOUT2_COMM



Attribute Updates of object class AOUT2_COMM are:

For Action coucou

in_comm := coucou.in_comm

dispatch_comm := coucou.dispatch_comm

For Action recevoir

destinataire := recevoir.destinataire

dispatch_comm := recevoir.dispatch_comm

caracts := recevoir.caracts

type_service := recevoir.type_service

entiers := recevoir.entiers

Calls of object class AOUT2_COMM are:

FOREIGNER call

Caller action is: signe_de_vie

Conditioned by: TRUE

Called action is: DISPATCH_COMM.DISPATCH_COMM.QUI

Identifying attribute is: dispatch_comm

Instantiations are:

QU.agv := in_comm

FOREIGNER call

Caller action is: transmettre

Conditioned by: TRUE

Called action is: DISPATCH_COMM.DISPATCH_COMM.recevoir

Identifying attribute is: dispatch_comm

Instantiations are:

recevoir.destinataire := destinataire

recevoir.entiers := entiers

recevoir.type_service := type_service

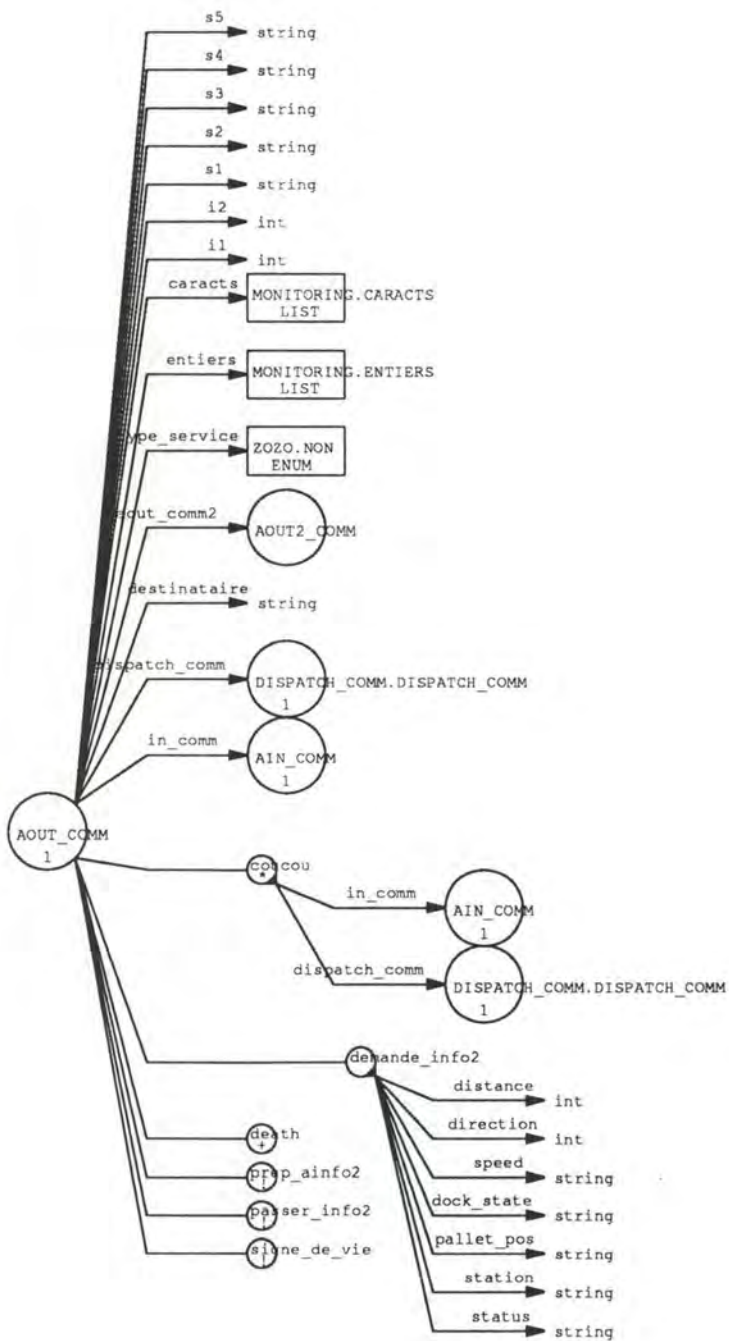
recevoir.caracts := caracts

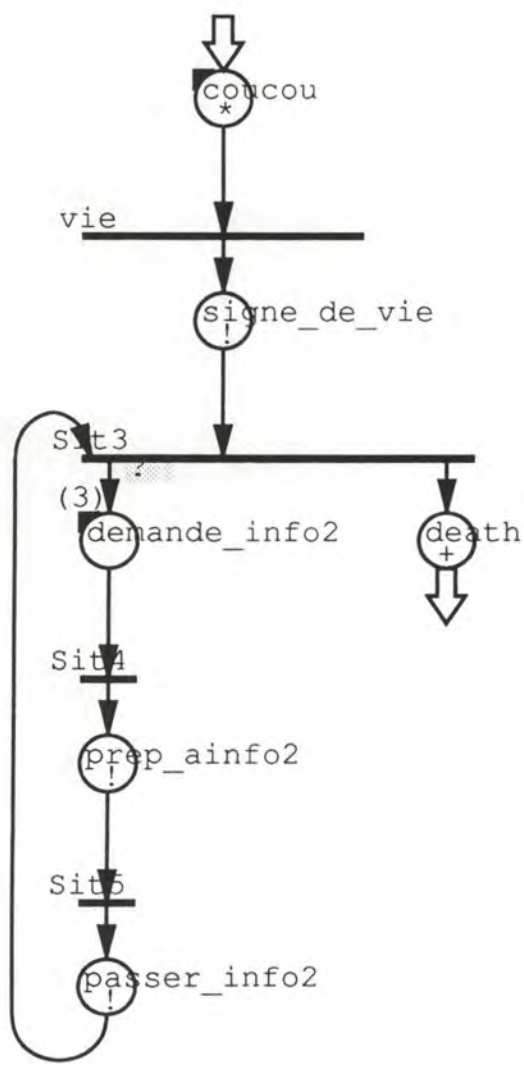
Behavior Conditions and Instantiations of object class AOUT2_COMM are:

(4) transmettre

Conditioned by: NOT(EXISTS[DISPATCHING:DISPATCH_COMM|(type_service = SELF.type_service)])

Declaration Diagram: AGV\AOUT_COMM





Attribute Updates of object class AOUT_COMM are:

For Action demande_info2

s1 := demande_info2.speed
i2 := demande_info2.direction
type_service := NON\$RET_AINFO2:ZOZO
i1 := demande_info2.distance
caracts := NEW(caracts)
s3 := demande_info2.station
s5 := demande_info2.dock_state
entiers := NEW(entiers)
s2 := demande_info2.pallet_pos
s4 := demande_info2.status
destinataire := "MONITORING"

For Action prep_ainfo2

caracts := (((APPEND(NEW(caracts), s1)|| APPEND(NEW(caracts), s2))|| APPEND(NEW(caracts), s3))|| APPEND(NEW(caracts), s4))|| APPEND(NEW(caracts), s5))
entiers := (APPEND(NEW(entiers), i1)|| APPEND(NEW(entiers), i2))

For Action coucou

dispatch_comm := coucou.dispatch_comm
in_comm := coucou.in_comm

Calls of object class AOUT_COMM are:

NORMAL call

Caller action is: passer_info2

Conditioned by: TRUE

Called action is: AGV.AOUT2_COMM.recevoir

Identifying attribute is: aout_comm2

Instantiations are:

recevoir.destinataire := destinataire
recevoir.type_service := type_service
recevoir.caracts := caracts
recevoir.dispatch_comm := dispatch_comm
recevoir.entiers := entiers

NORMAL call

Caller action is: signe_de_vie

Conditioned by: TRUE

Called action is: AGV.AOUT2_COMM.coucou

Identifying attribute is: aout_comm2

Instantiations are:

coucou.in_comm := in_comm

coucou.dispatch_comm := dispatch_comm

Behavior Conditions and Instantiations of object class AOUT_COMM are:

(3) demande_info2

Conditioned by: **NOT(EXISTS[AOUT2_COMM:AGV|(type_service = NON
\$RET_AINFO2:ZOZO)])**

Community Diagram: TYPES_LISTE

