A Graphic Specification of a High-Voltage Station

Gianna Reggio
DISI
Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova
Via Dodecaneso, 35 – Genova 16146 – Italy
reggio @ disi.unige.it
http://www.disi.unige.it

In this report we present the development of an industrial case study using the SMoLCS formal method, see [Reg98, LR97], precisely the high-voltage stations for the distribution of the electric power used by ENEL, the Italian National Company of Electricity. The stations are the nodes of the electric high-voltage net. They allow: the setting of energy coming from the production centers and/or from other nodes, the transformation of the levels of voltage and the distribution to other nodes where the energy is used, for example transformation cabins.

We have organized the development of such case study in the following phases.

Capture and specification of the requirements Determination and specification the fundamental requirements on the high-voltage stations considered by ENEL.

First development step Refinement of such stations by considering only those managed by a software automatism: the components of the station that constitute the plant are completely designed. We give a functional description of the automatism, without structuring it, so that we leave as much freedom as possible to its implementation.

Second development step Specification of the particular automatism designed by ENEL. This specification has to be interfaced with those already defined at the second level, to obtain the complete specification of the ENEL stations. The most relevant difficulty found at this level has been to understand correctly the concurrent activity of the components of the automatism, starting from the informal description provided by ENEL. It has been possible to overcome this difficulty thanks to a constant verification with the designers of the automatism.

In this report we present both the formal and the informal specifications of the stations produced at the various steps using for both of them a graphic notation; sometimes we report the informal specification as comments within the formal one.

Acknowledgements. I thank V. Filippi of ENEL for her cooperation to the development of this case study and A. Morgavi that has given a very preliminary specification of these stations in hers master thesis.

STATION.PHASE 1

In this phase we specify the *high-voltage stations* (shortly *stations* from now on) considered by ENEL by giving the fundamental properties of their interactions with their external environment.

STATION.PHASE 1: **Natural Description**

The stations are the nodes of the electric high-voltage net. They allow: the setting of energy coming from the production centers and/or from other nodes, the transformation of the levels of voltage; and the distribution to other nodes where the energy is used, for example transformation cabins.

In this case study we consider stations formed by a double bar (i.e., a pair of wires) and by a set of devices, breakers and isolators connected to the bars and to lines reaching the station. The connections between lines and bars are realized with the opening and closing of devices suitable to this kind of connections.

The problem of automatizing the management of these stations has brought to an analysis of their structure to permit a classification of the devices composing them in sets accordingly to their different functionalities; every identified set, called *functional unit*, performs a precise duty for what concerns the automation of the station, at a higher level of abstraction with respect to the simple opening and closing of a single device. Thus we can see a station as formed by a set of functional units, each belonging to an identified typology.

The class of stations considered is defined by the set of types of functional units associated with constraints on their possible topological combinations.

In this case study we consider only stations with two bars denoted by "A" and "B", respectively, and with three kinds of functional units: *Ae*, *Dd* and *Fa*.

Fig. 1 presents the kinds of functional units with their positions.

An Fa is a functional unit that makes the electric connection between the pair of bars and a line either for one bar or for the other; it can be in three positions:

- *open*, when the line is not electrically connected to any bar;
- *closed on bar* A, when the line is electrically connected to bar A;
- *closed on bar* B, when the line is electrically connected to bar B.

A Dd is a functional unit which electrically connects the two bars equalizing their actual tensions; the possible positions are:

- *open*, when the two bars are not connected;
- *closed*, when the two bars are connected.

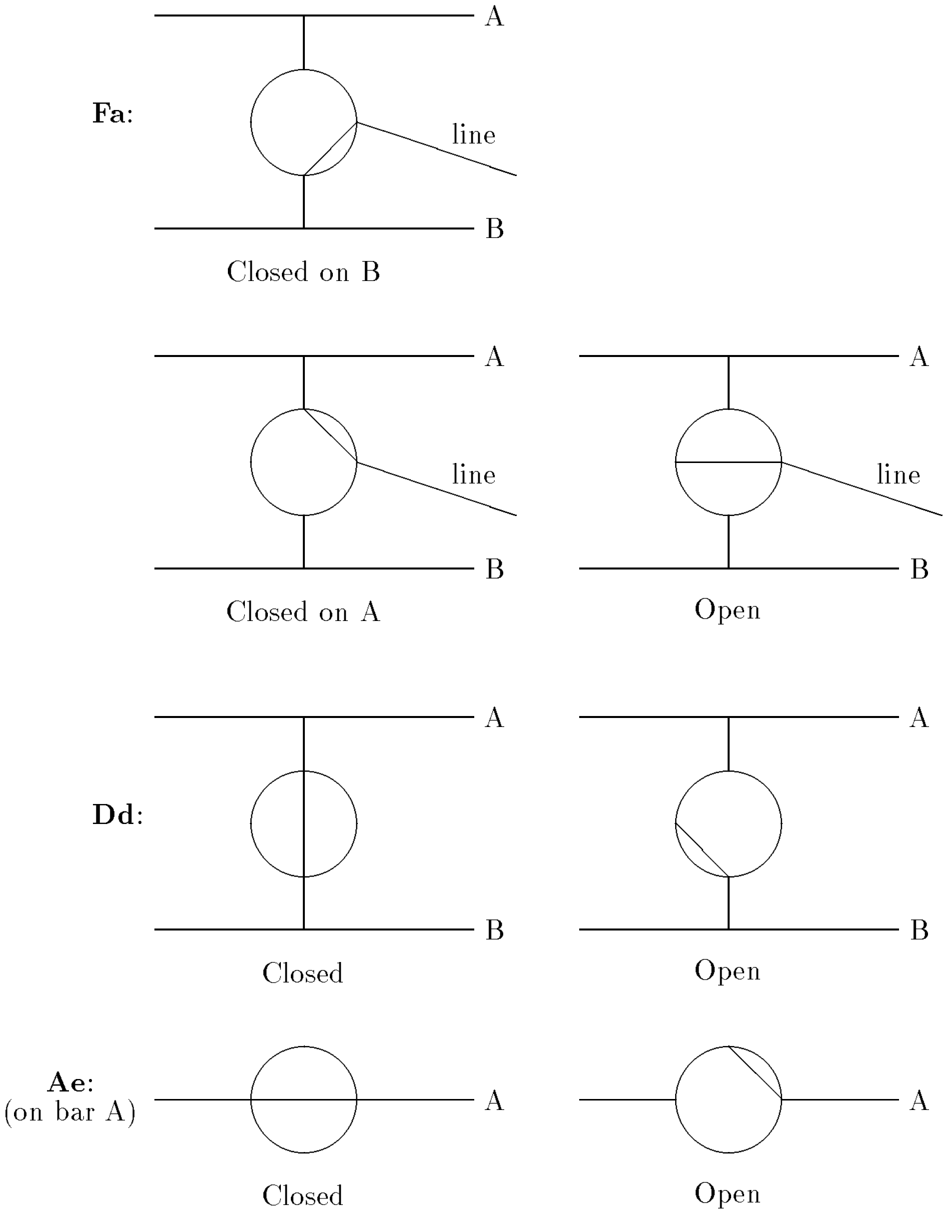


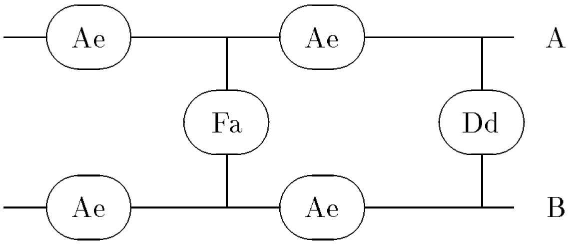
Figure 1: The kinds of the functional units and their positions

An Ae is a functional unit which allows to isolate electrically sections of a bar to permit upkeep; it is *closed (open)* when the two bar sections are connected (not connected).

Every station can be represented with a combination of elements belonging to such types. The elements are placed down on bars, with the following bounds:

- Ae's are always in pairs (one for each bar);
- bars can be sectioned by any number of pairs of Ae's (also none);
- there must exist at least one Fa and
- there must exist at least one Dd.

Below we graphically present an example of an admissible station.



The operator can require operations on the functional units of a station by using non-detailed commands, remitting to the station the task of managing other possible functional units and in particular of managing the composing devices. The request of an operation consists in the selection of a functional unit, by means of an identifier, and of the final position that the unit has to reach. The possible final positions are: for units of kind Fa, closed on bar A, closed on bar B and open; while for units of kind Ae and Dd are closed and open.

If the execution of an operation is going on, the station does not accept requests of other operations.

Moreover, if the required operation is useless (the position of the unit required by that operation is already reached), then it must not be executed and the operator has to be informed.

Before of executing any operation, the station has to verify that the bars involved are under tension; on the contrary an error rises, treated as a generic failure.

Below we briefly explain how to perform the various operations on the functional units.

The functional unit of kind Ae and Dd can be opened and closed, and those of kind Fa opened.

To close on bar A an Fa closed on bar B, or converse, is called "bar exchange"; to execute this operation it is necessary that the two bars are connected each other at least by a Dd. The station has to identify a *closing path* and to operate on the corresponding Dd without the operator, if necessary.

We identify the closing path using the following rules in decreasingly order of priority:

- if there exists one closed Dd s.t. every Ae, which possibly separates it from the Fa, is closed, the operation can take place;

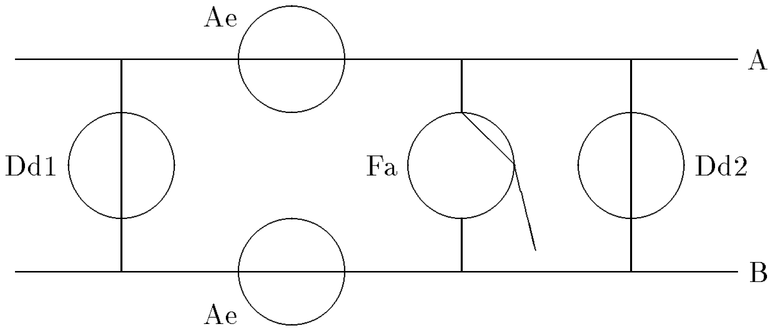


Figure 2: Either Dd1 or Dd2 may be chosen for building the closing path for Fa.

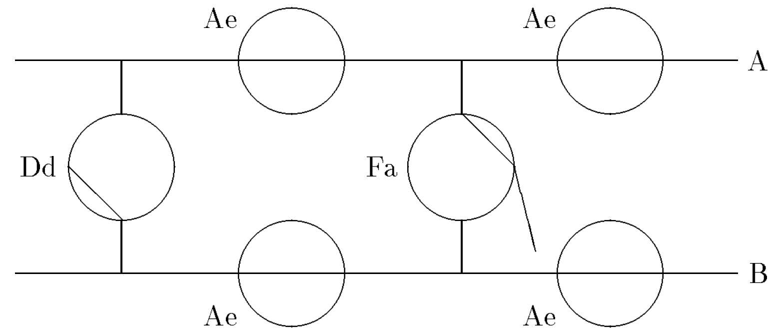


Figure 3: Dd must be closed for building the closing path for Fa.

- otherwise, if there exists some open Dd s.t. every Ae, which possibly separates it from the Fa, is closed, it is necessary to close one of them and after the operation can take place
- otherwise, the station has to signal to the operator that it is impossible to execute the operation (an Ae has to be closed).

Fig. 2, 3 and 4 show examples of such three cases.

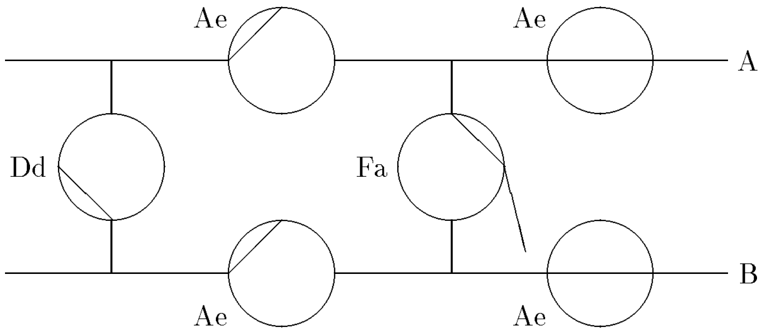


Figure 4: No closing path for Fa is possible, some Ae must be closed.

STATION.PHASE 1: **Border Determination**

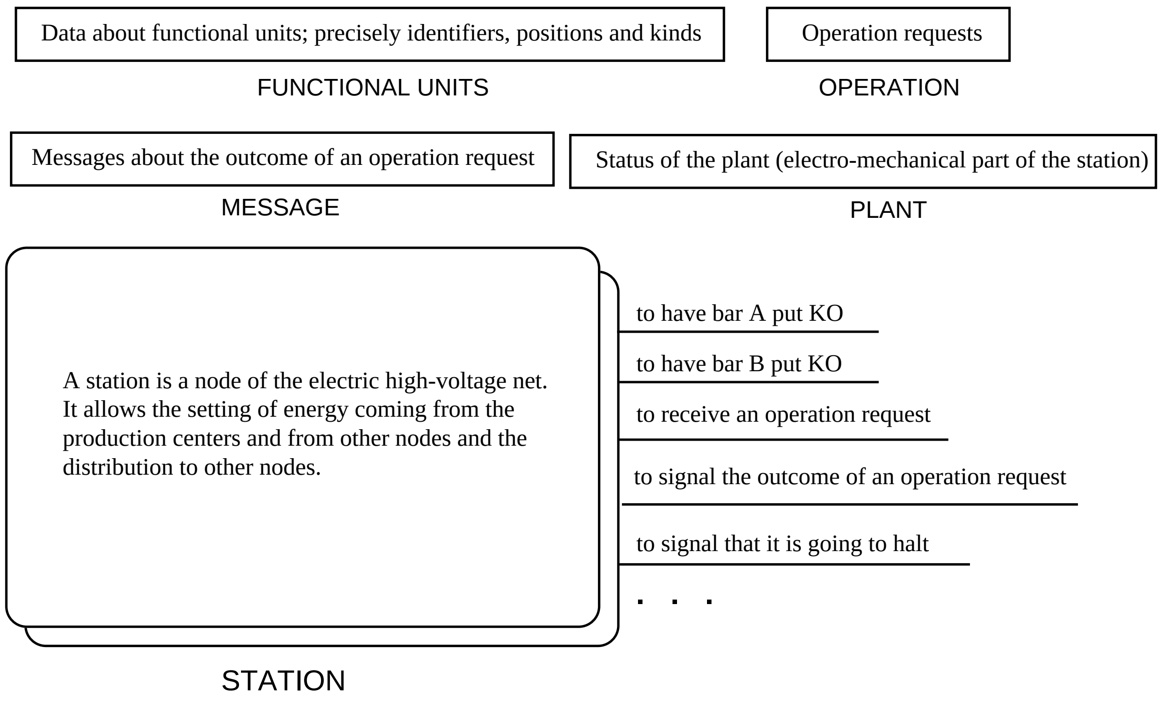
A station is an open system, because it may interact with the external environment consisting of an operator and of the other nodes of the high-voltage electric net. There are no hypotheses on the behaviour of the nodes and of the operator, so they are considered outside the specified system.

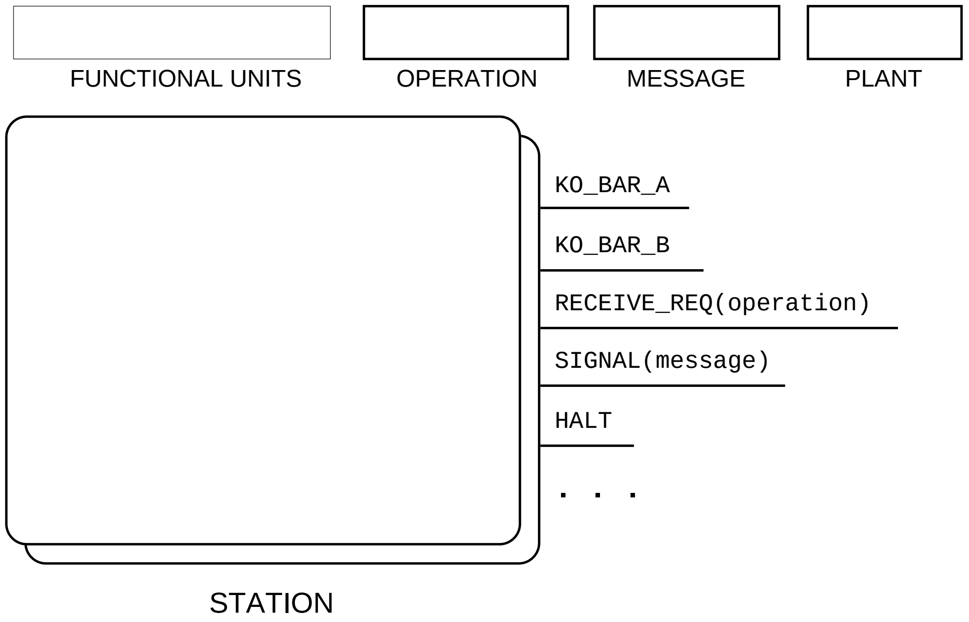
STATION.PHASE 1: **Shadow Spots**

The Natural Description does not say what to do when the station receives a wrong operation request, for example to operate on a non-existing unit. We have assumed that the station will receive only correct operation requests.

STATION.PHASE 1: **Specification**

Structure & Interactions





STATION.PHASE 1: **Basic Data Structures**

Data about functional units; precisely identifiers, positions (Open, Close, Close on)show

FUNCTIONAL UNITS

FUNCTIONAL UNITS

Operation requests, consisting of the identifier of the unit on which to operate and of the position to reach.

OPERATION

Messages about the result of an operation request: Completed, Useless, Impossible.

MESSAGE

```
unit_identifier = NAT
unit_kind ::= Ae | Dd | Fa
unit_position ::= Open | Close | Close_On_A | Close_On_B | Broken
```

FUNCTIONAL UNITS

FUNCTIONAL UNITS

```
operation ::= Opr(unit_ident,unit_position)
```

OPERATION

```
message ::= Completed | Useless | Impossible
```

MESSAGE

PLANT

FUNCTIONAL UNITS

The status of a functional unit, characterized by the identity, the kind and the position. A unit of kind AE or Dd may beopen, closed or broke; and one of kind Fa may be open, closed on A, closed on B or broken.

FUNCTIONAL_UNIT_STATUS

The electrical status of a bar: in tension and not in tension.

BAR_STATUS

BAR_STATUS

FUNCTIONAL_UNIT_STATUS

The status of a plant is given by the status of the two bars and of the functional units connected to each bar. In each case Fa's and Dd's are connected to both bars, while Ae's are connected only to one bar, but there is one Ae on a bar iff there is another Ae on the other bar.

In the plants of the adimissible stations there is at least one Dd and one Fa.

PLANT

```
FUNCTIONAL UNITS

op  Kind: functional_unit  -> unit_kind

op  Position: functional_unit -> unit_position

op  Id: functional_unit -> unit_ident
----------------------------------------------------------------------
if  Kind(fu) = Ae or Kind(fu) = Dd then
    (Position(fu) = Open or Position(fu) = Closed or Position(fu) = Broken)

if  Kind(fu) = Fa then
    Position(fu) = Open or Position(fu) = Close_On_A or
    Position(fu) = Close_On_B or Position(fu) = Broken
```

```
fu:  functional_unit
```

FUNCTIONAL_UNIT_STATUS

```
bar ::= OK | KO
```

BAR_STATUS

```
BAR_STATUS    FUNCTIONAL_UNIT_STATUS

plant ::=
  < bar bar > |
  < functional_unit functional_unit > & plant |
  < functional_unit > & plant

**  The status of the plant consists either
**  of just the status of the two bars,
**  or of the status of two functional units (one for each bar) plus the status of
**  the remain plant,
**  or of the status of a functional unit (connected to both bars) plus the status
**  of the remaining plant
```

```
**  checks if a functional unit is connect to bar A (B)
pr  On_A, On_B: functional_unit  plant

ax  On_A(fu,<fu> & pl)
ax  On_A(fu,<fu1> & pl) if On_A(fu,pl)
ax  On_A(fu,<fu fu1> & pl)
ax  On_B(fu,<fu> & pl)
ax  On_B(fu,<fu1> & pl) if On_B(fu,pl)
ax  On_B(fu,<fu1 fu> & pl)
```

```
**  checks if a functional unit is present in the plant
pr  Is_In: functional_unit  plant

ax  if On_A(fu,pl) or On_B(fu,pl) then Is_In(fu,pl)
```

```
**  checks whether the connections of the functional units on the bars are
**  admissible, i.e. if each one of kind either Fa or Dd is connected to both
**  the bars and those of kind Ae are present in pairs
pr  Ok_Con: plant

ax  Ok_Con(<ba bb>)
ax  if Ok_Con(pl) and Kind(fu1) = Ae and Kind(fu2) = Ae then
    Ok_Con(<fu1 fu2> & pl)
ax  if Ok_Con(pl) and Kind(fu) = Dd  then  Ok_Con(<fu> & pl)
ax  if Ok_Con(pl) and Kind(fu) = Fa  then  Ok_Con(<fu> & pl)
```

```
**  checks whether a station is admissible, i.e. if the connections are admissible
**  and there is at least a functional unit of kind Fa and one of kind Dd
pr  Ok_Plant: plant

ax  Ok_Plant(pl) iff
        Ok_Con(pl) and
        (exists fu: Is_In(fu,pl) and Kind(fu) = Fa) and
        (exists fu: Is_In(fu,pl) and Kind(fu) = Dd)
```

```
**  returns the status of bar B (B)
op  Bar_A, Bar_B: plant -> bar

ax  Bar_A(<ba bb>) = ba
ax  Bar_A(<fu> & pl) = Bar_A(pl)
ax  Bar_A(<fu1 fu> & pl) = Bar_A(pl)
ax  Bar_B(<ba bb>) = bb
ax  Bar_B(<fu> & pl) = Bar_B(pl)
ax  Bar_B(<fu1 fu> & pl) = Bar_B(pl)
```

```
pl : plant              fu fu1: functional_unit       ba bb: bar
```

PLANT

STATION.PHASE 1: **States**

FUNCTIONAL UNITS   OPERATION   MESSAGE   PLANT

```
**  returns the status of the physical components of the station (the plant)
op  Plant: station -> plant

**  checks whether the station is executing an operation
pr  Executing: station

**  checks whether the station is in a initial state
pr  Initial: station
```

STATION1 (states)

STATION.PHASE 1: **Activity**

```
**  checks if there is a failure in station
pr  Failure: station

**  If there exists a broken functional unit, then there is a failure in station
if  exists uid: Position(st,uid) = Broken then Failure(st)

**  If the station has received a request to perform an operation on a functional
**  unit connected to a failed bar, then there is a failure in station
if  exists uid, st’, up: st’ -- RECEIVE_REQ(Opr(uid,up)) --> st  and
    On_Failed_Bar(uid,Plant(st)) then
        Failure(st)

        **  checks if a functional unit is connected to a failed bar
        pr  On_Failed_Bar: unit_ident plant

        if  On_Bar_A(fu,pl) and Bar_A(pl) = KO then On_Failed_Bar(Id(fu),pl)
        if  On_Bar_B(fu,pl) and Bar_B(pl) = KO then On_Failed_Bar(Id(fu),pl)
```

```
**  returns the kind of a functional unit
op  Kind: station unit_ident -> kind

Kind(st,uid) = k  iff (exists fu: fu Is_In Plant(st) and Kind(fu)=k  and Id(fu)=uid)
```

```
**  returns the position of a functional unit
op  Position: station unit_ident -> unit_position

Position(st,uid) = up  iff
    (exists fu: fu Is_In Plant(st) and Position(fu) = up and Id(fu) = uid)
```

```
  **  checks wether a functional unit is a closing path in a station for a given Fa
  pr  _ Is_Closing_Path_For _ In _: unit_ident unit_ident station

  **  If the the functional unit fid is a closing path for idfa, then fid is a
  **  Dd on the same bar section of idfa and either is closed or open and in such
  **  case there does not exist a closed Dd on the same bar section of idfa
  if  fid Is_Closing_Path_For fa In st then
      - Kind(st,uid) = Dd
      - SameSection(Plant(st),fa,uid)
      - Position(st,uid) = Closed  or
        (Position(st,uid) = Open and not exists uid’: Kind(st,uid’) = Dd and
         Position(st,uid’) = Closed and SameSection(Plant(st),fa,uid’))

  **  If there exists a functional unit that is a Dd and on the same bar section
  **  of fa, then there exists a functional unit that is a closing path for fa
  if  exists uid: Kind(st,uid) = Dd and SameSection(Plant(st),fa,uid) then
      exists uid’: fid’ Is_Closing_Path_For fa In st

    **  checks whether two functional units are on the same bar section in a plant
    pr  SameSection: plant unit_ident unit_ident

    if  On_Bar_A(fu,pl) and On_Bar_A(fu’,pl) then  SameSection(pl,Id(fu),Id(fu’))
    if  On_Bar_B(fu,pl) and On_Bar_A(fu’,pl) then  SameSection(pl,Id(fu),Id(fu’))
```

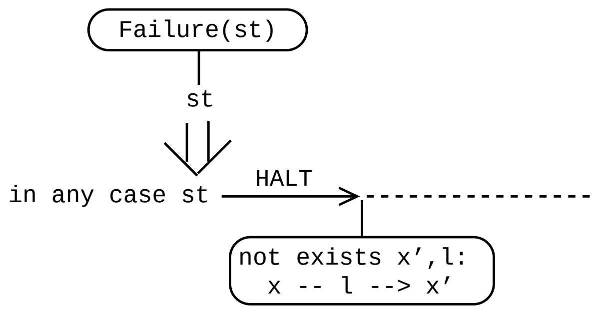
```
uid, uid’, fa: unit_ident       fu, fu’: fun
k: kind                         st: station        up: unit_position
```

(auxiliary)

Initial

```
 **  In the initial state each unit is open and the bars are OK
 if  Initial(st) then
     - forall fu: if Is_In(pl,fu) then Position(fu,pl) = Open
     - Bar_A(Plant(st)) = OK
     - Bar_B(Plant(st)) = OK
```

Failure



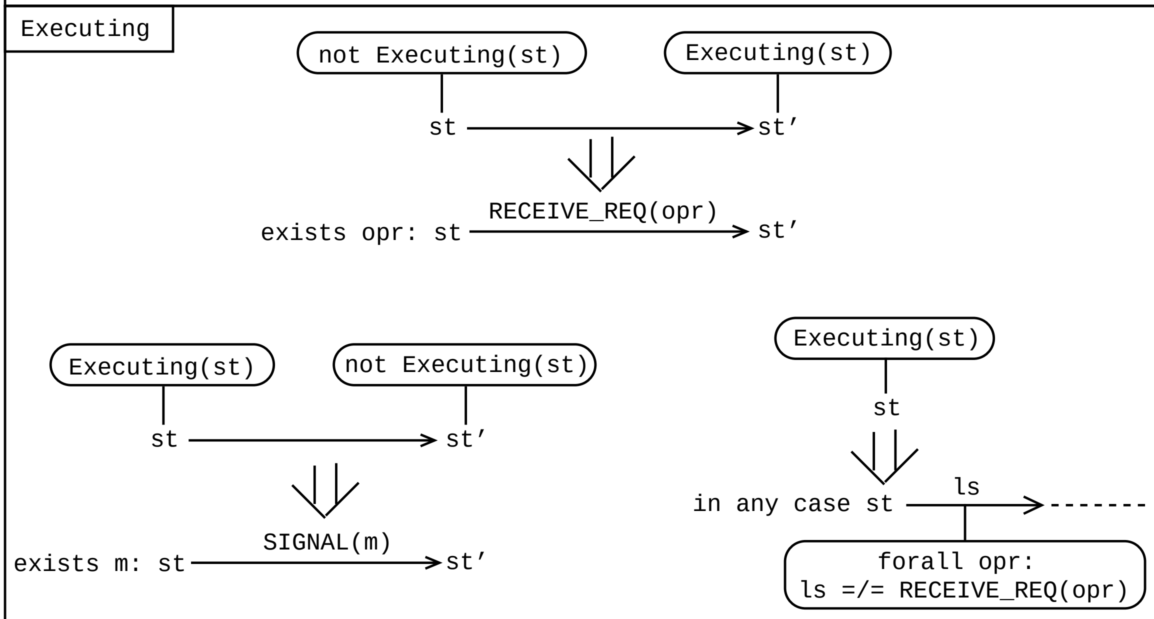
Plant

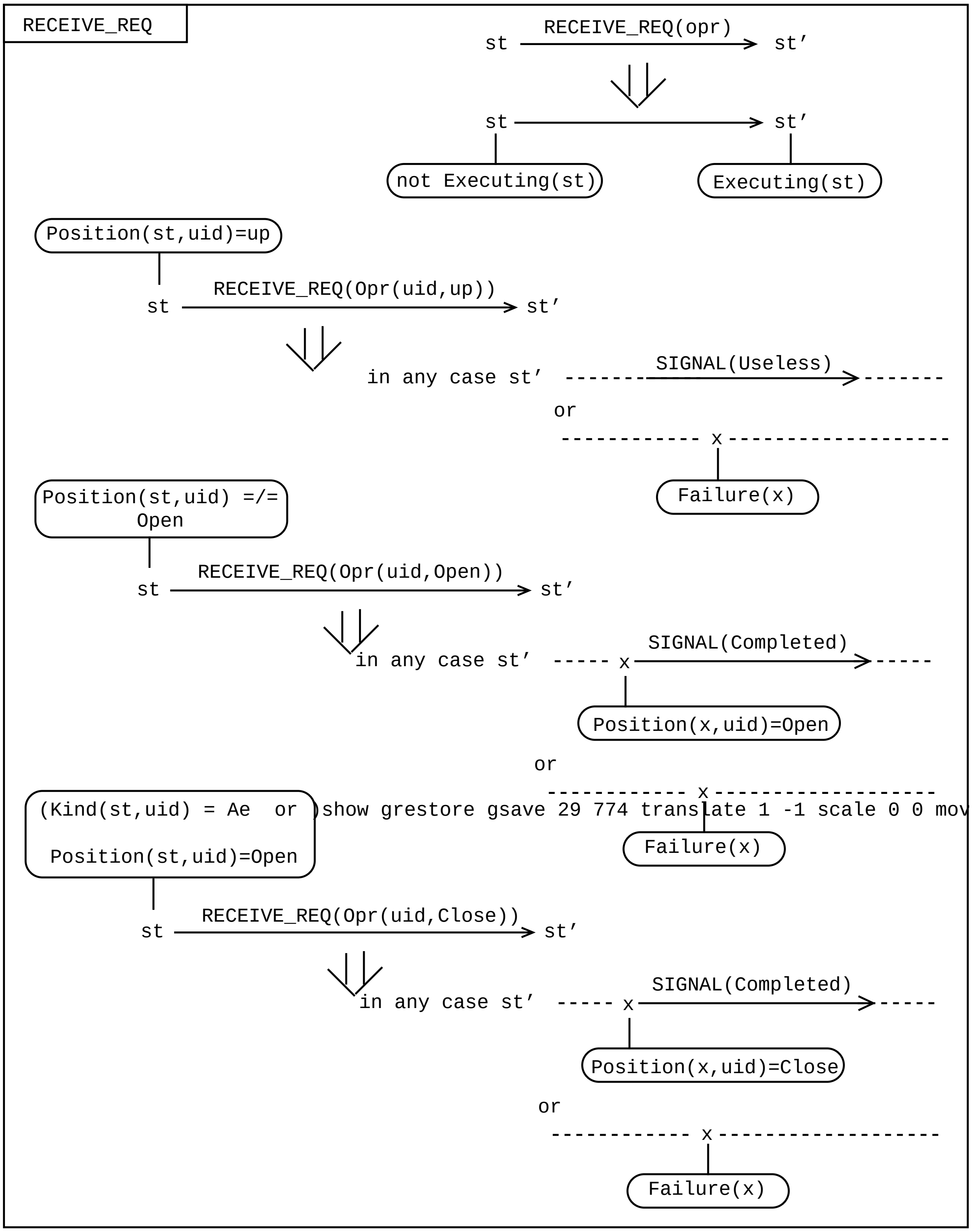
```
Ok_Plant(Plant(st))
```

Kind

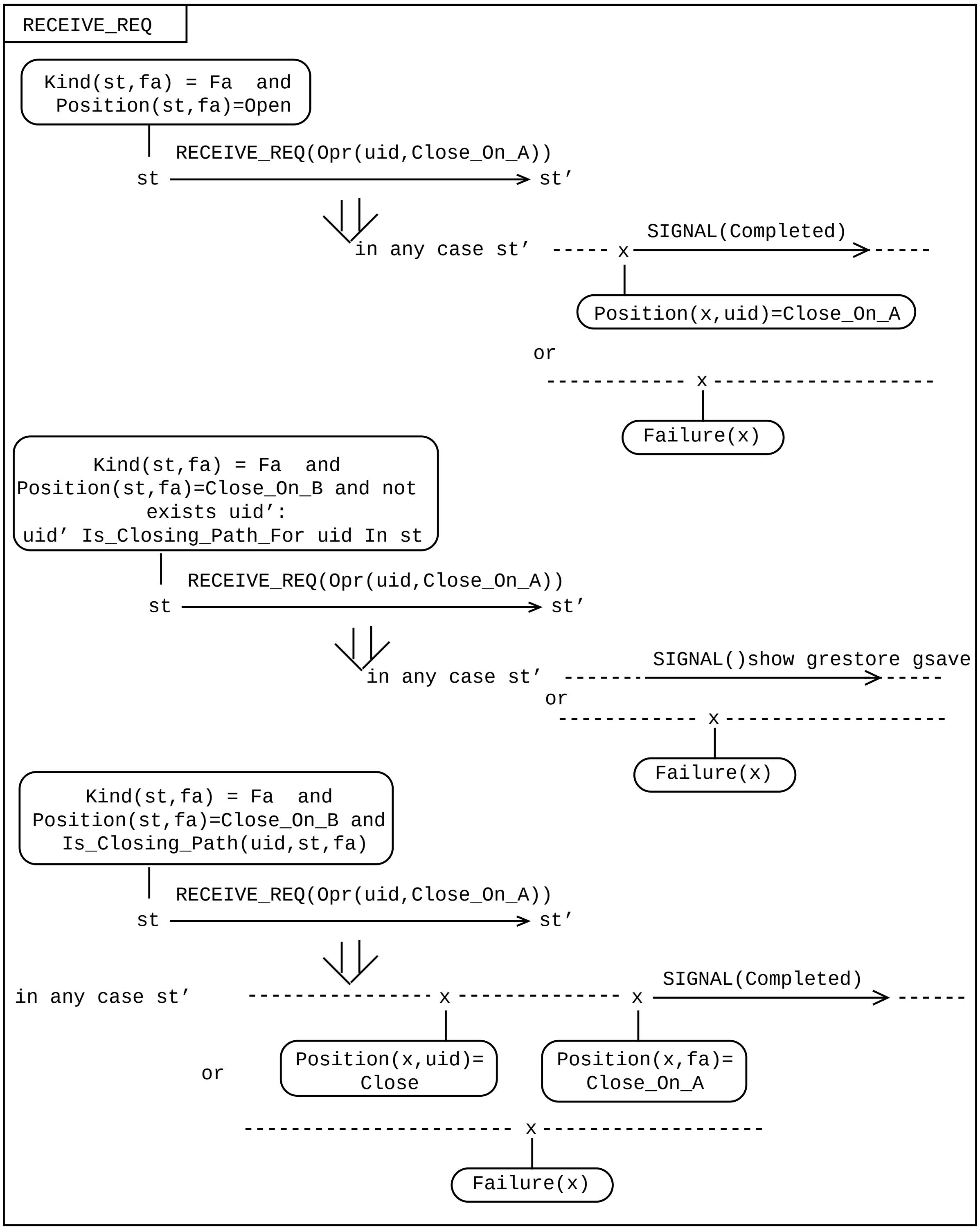
```
**  A functional unit cannot change its kind
if  st -- ls --> st' then Kind(st,fid) = Kind(st',fid)
```

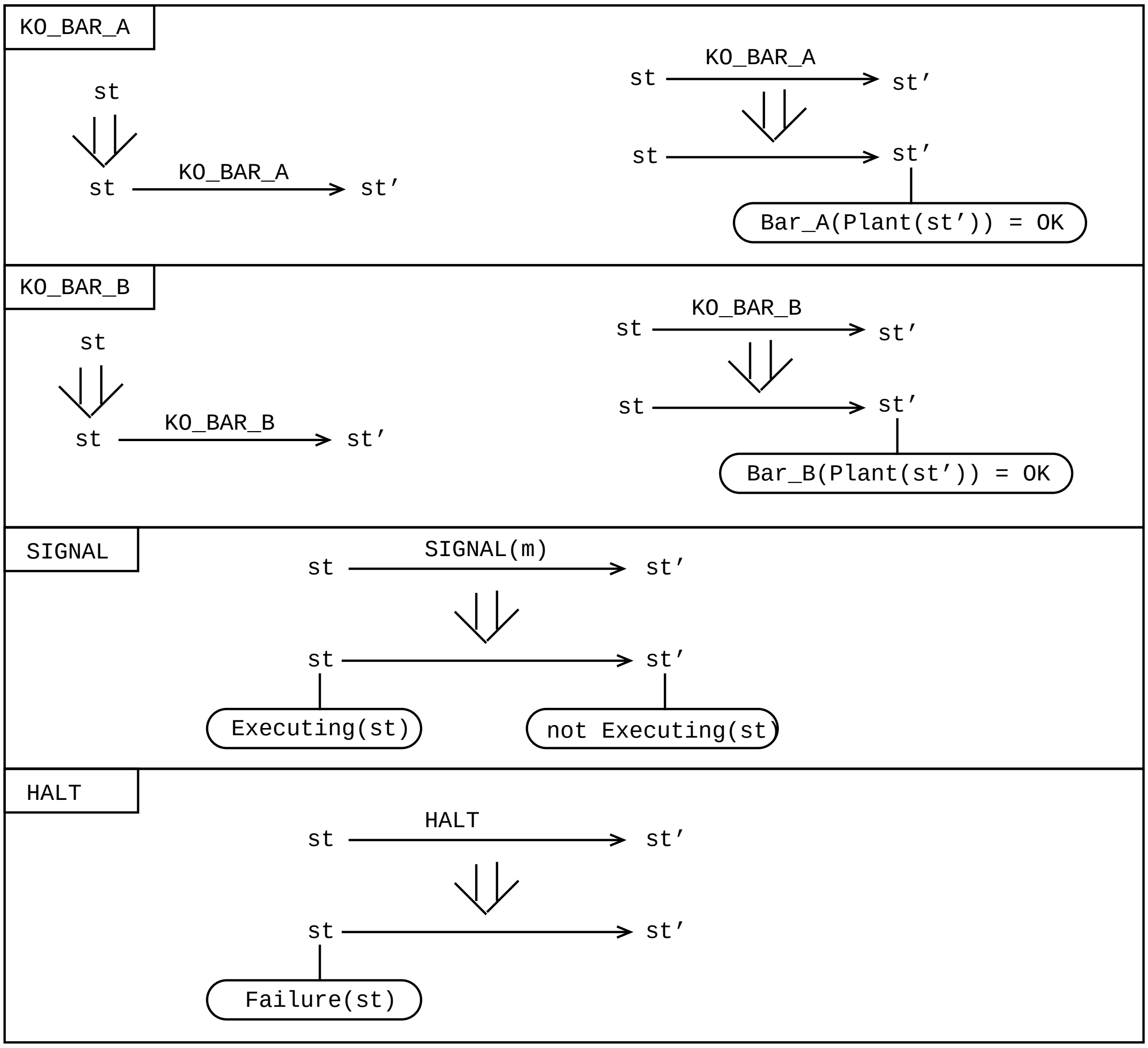
Executing





RECEIVE_REQ
st
RECEIVE_REQ(opr)
st'
st
st'
not Executing(st)
Executing(st)
Position(st,uid)=up
st
RECEIVE_REQ(Opr(uid,up))
st'
in any case st'
SIGNAL(Useless)
or
x
Failure(x)
Position(st,uid) =/= Open
st
RECEIVE_REQ(Opr(uid,Open))
st'
in any case st'
x
SIGNAL(Completed)
Position(x,uid)=Open
or
x
(Kind(st,uid) = Ae or)show grestore gsave 29 774 translate 1 -1 scale 0 0 mov
Position(st,uid)=Open
Failure(x)
st
RECEIVE_REQ(Opr(uid,Close))
st'
in any case st'
x
SIGNAL(Completed)
Position(x,uid)=Close
or
x
Failure(x)


RECEIVE_REQ
Kind(st,fa) = Fa and Position(st,fa)=Open
RECEIVE_REQ(Opr(uid,Close_On_A))
st
st'
in any case st'
x
SIGNAL(Completed)
Position(x,uid)=Close_On_A
or
x
Failure(x)
Kind(st,fa) = Fa and Position(st,fa)=Close_On_B and not exists uid': uid' Is_Closing_Path_For uid In st
RECEIVE_REQ(Opr(uid,Close_On_A))
st
st'
in any case st'
SIGNAL()show grestore gsave
or
x
Failure(x)
Kind(st,fa) = Fa and Position(st,fa)=Close_On_B and Is_Closing_Path(uid,st,fa)
RECEIVE_REQ(Opr(uid,Close_On_A))
st
st'
in any case st'
x
x
SIGNAL(Completed)
Position(x,uid)= Close
Position(x,fa)= Close_On_A
or
x
Failure(x)



m: message	ls: lab_station	opr: operation
st st' x x': station	uid, uid', fa: unit_ident	up: unit_position

STATION.PHASE 2.step 1

In this phase we specify the class of the stations handled by an automatism; clearly they are a subclass of those specified in PHASE 1.

STATION.PHASE 2.step 1: **Natural Description**

A unique operator manages sets of stations and power plants. Thus the absence of local operators in the stations, due to the installation of an automatic system, raises the necessity for the system to satisfy security requirements of command and control; besides, to make easy the tasks of the operator, the command of the operations has to take place by means of orders to the automatism, which ensures also a continual caretaking of the devices of the plant.

The automatism has to be adaptable to different stations, which can be modified and widened also after the installation. To realize its own task, the automatism has to know the topology of the station on which it works, so we can assume that during the installation phase the diagram of the station topology is provided.

The automatism has the task of checking the position of the devices of the plant and of operating them accordingly to operator requests. So it has to collect continually information coming from such devices about their own positions (reading the signals from an interface with the field) and it has to be able to transmit (by means of the same interface) operating orders.

The interface is connected to the automatism by a set of channels, one for each device of the station. Each channel can contain one of the following symbols:

- OP if the device communicates “open”;
- CL if the device communicates “closed”;
- XX, if “open” and “closed” are communicated simultaneously or if a failure of the device is detected by the interface;
- it can be empty if none of the previous conditions is satisfied (for example when the device is moving). Indeed we assume that, as the operation time of the devices is faster than the reading one (order of hundreds of milliseconds), the automatism can read several times the value of a channel before it can find the “closed” symbol. Please notice that the channel keeps no memory of the position left from the device.

Moreover, every bar transmits its state, analogously to devices, through an appropriate channel, which can contain either the symbol “OK” if the corresponding bar works correctly, or the symbol “KO” if the tension is down (breakdown or earth-wire).

In the same way, orders given from the automatism to each device of the plant are symbols transmitted on such channels: CL for “close” and OP for “open”.

To verify the correct execution of an order, it is necessary for the automatism to check the position reached by the used device.

The operation of opening and closing a device has to be executed within a certain time from the sending of the command. If the automatism does not receive within that time the signal that the required state has been reached, the device has to be considered damaged.

The operator must have a self-starting key of the automatism. This key works also as a "reset hardware" key; when the system reaches an irrecoverable error state, due, for example, to the reading of symbol XX transmitted from a device of the plant through the interface towards the field.

The automatism performs two fundamental tasks: *monitoring* and *management* of the station components.

The monitoring has to be performed continuously, from system starting to its stopping and during every operation. It consists in verifying:

- that the devices remain in the positions required by the operator and
- the rising of failures and abnormal situations.

In case of failure the system has to signal that to the operator and to stop its own activity. In this case study we do not distinguish the different kinds of failure and we do not perform recover activity. The automatism has to be able to recognize, when starting, an inconsistent state of the station.

The management activity consists in verifying the feasibility of the operations required by the operator and in executing these operations and checking their results.

The operator can guide the functional unit operations using not detailed commands, remitting to the automatism the task of managing other possible functional units and in particular of managing their devices. The request of an operation consists in the selection of a functional unit, by means of an identifier, and of the final state that the functional unit has to reach. The possible final states are: closed on bar A (CA), closed on bar B (CB) and open (OP); for functional units like Ae and Dd, the commands CA and CB stand for the "closed" command.

If the execution of an operation is going on, the automatism does not accept request of other operations.

Moreover, if the operation ordered has already been executed (the position of the functional unit required by that operation is already reached) it must not be executed and the operator has to be informed.

Before of executing any operation, the automatism has to verify that bars involved are under tension; on the contrary an error rises, treated as a generic failure.

Below we briefly explain how the functional units are made of devices and how the various operations are performed on them.

The devices present in the station (breakers, isolators) can be in two positions: on and off (open and close in the following).

The breakers can be put on/off under tension and have operation time of the order of hundreds of milliseconds; isolators cannot be used undertension (except bar isolators) and have operation time of the order of some seconds. The result is that functional units own a breaker for opening the whole unit, before using isolators and for closing it at the end of the operation.

In this case study we have analyzed three kinds of functional units:

- Ae made by a bar isolator;

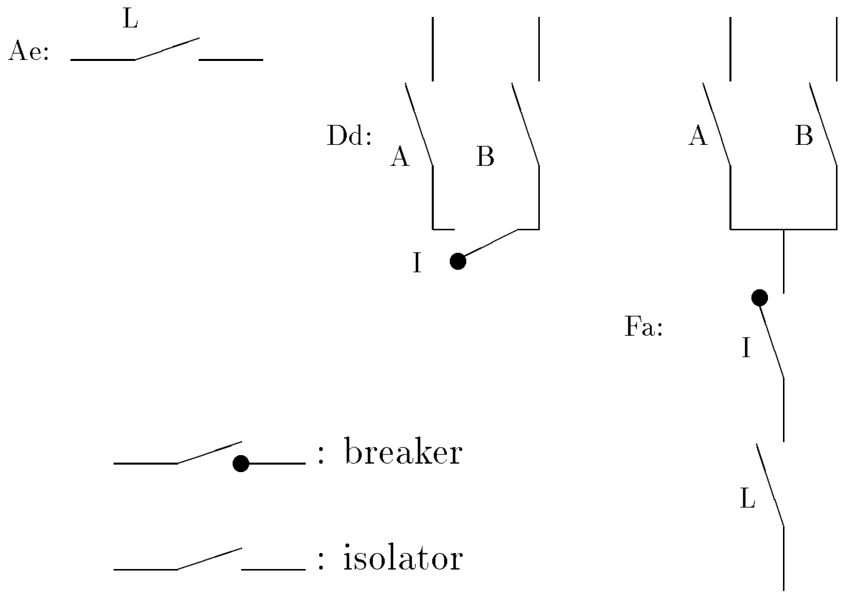


Figure 5: Schemas of the functional units

- Dd made by a breaker, a bar isolator connected to bar A and a bar isolator connected to bar B;
- Fa made by a line isolator, a breaker, a bar isolator connected to bar A and a bar isolator connected to bar B.

The schemas of the various functional unit kinds are presented in Fig. 5.

An Fa is a functional unit that makes the electric connection between the pair of bars and a line either for one bar or for the other; it can be in three positions:

- *open*, when the three isolators are open (thus the line is not electrically connected to any bar);
- *closed on bar* A, when the isolator on bar A, the line isolator L and the breaker I are closed, while the isolator on bar B is open (thus the line is electrically connected to bar A);
- *closed on bar* B, when the isolator on bar B, the line isolator and the line breaker are closed, while the bar isolator A is open (thus the line is electrically connected to bar B).

A Dd is a functional unit which electrically connects the two bars equalizing the two actual tensions; the possible positions are:

- "open", when the two isolators and the breaker are open (thus the two bars are not connected);
- "closed", when the two isolators and the breaker are closed (thus the two bars are connected).

An Ae is a functional unit, consisting simply of a bar isolator, which allows to isolate electrically sections of bar to permit upkeep; it is open/closed when its isolator is open/closed.

AE operations The functional unit Ae is composed by a single device (an isolator); so its closing/opening corresponds to the opening/closing of its isolator.

Dd operations The functional unit Dd is composed by a breaker S and two isolators A and B; it can be closed or open. The opening operation consists of opening, in the following order, S, A and B; and the closing operation consists of closing, in the following order, S, A and B.

Fa operations The functional unit Fa is composed by a breaker S and three isolators L, A, B; it can be closed on bar A, or on bar B, or open.

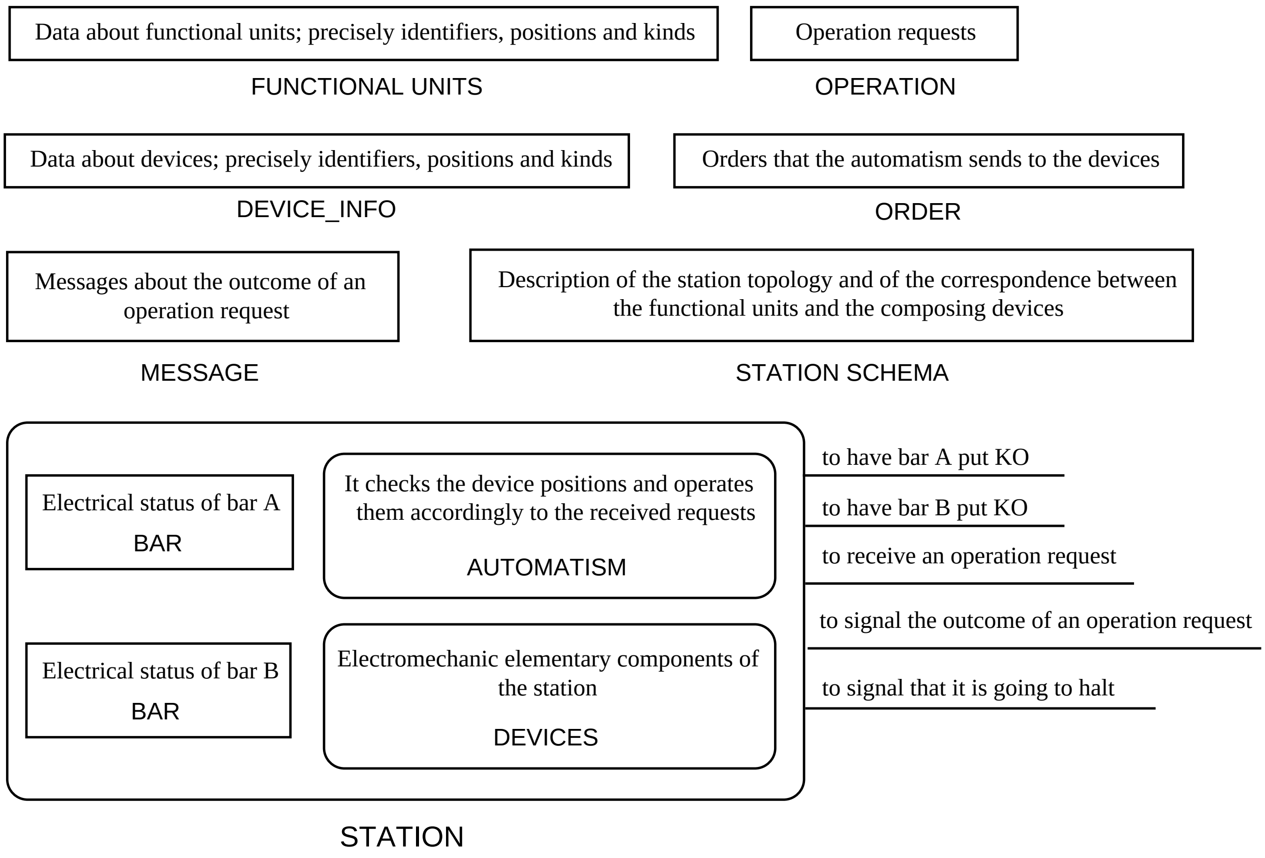
The opening operation consists of opening S, then A and then B. The closing operation on bar A of an Fa open consists of closing A, then closing L and then closing S, analogously for closing on bar B.

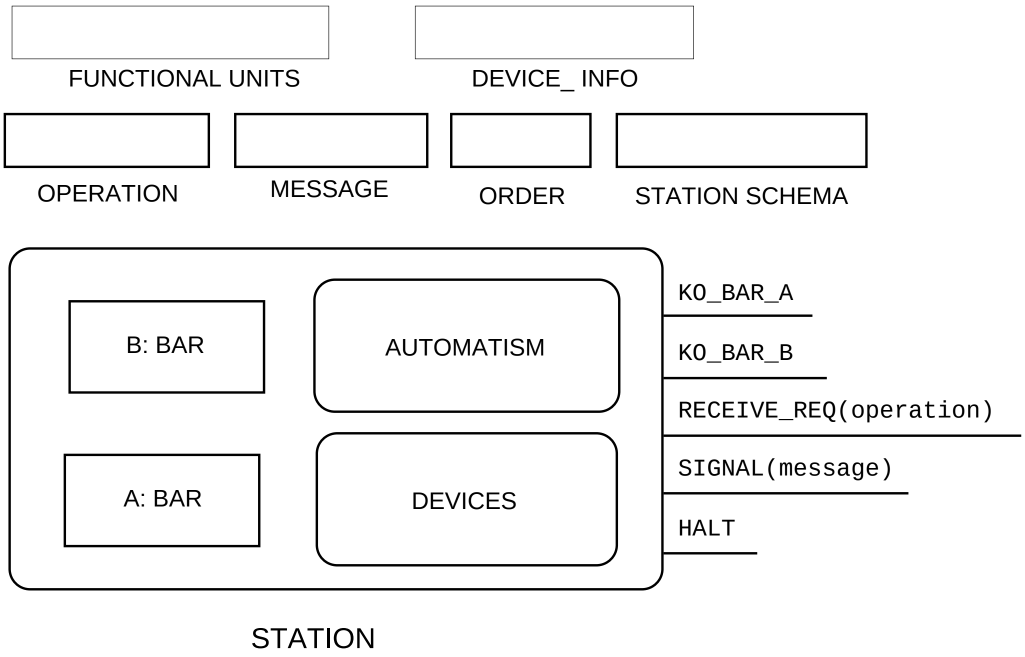
Closing operation on bar A of an Fa closed on bar B, or converse, is called "bar exchange"; to execute this operation it is necessary that the two bars are connected each other by a Dd. The automatism has to identify a *closing path* and to operate the corresponding Dd without the operator, if necessary.

We identify the closing path with the rules already presented in the natural description of PHASE 1.

STATION.PHASE 2.step 1: Specification

Structure & Interactions





STATION.PHASE 2.step 1: **Basic Data Structures**

Data about functional units; precisely identifiers, positions (Open, Close, Close on bar A, Close on bar B, Broken) and kinds (Ae, Dd, Fa)

FUNCTIONAL UNITS

FUNCTIONAL UNITS
Operation requests, consisting of the identifier of the unit on which to operate and of the position to reach.

OPERATION

Messages about the resulkt of an operation request: Completed, Useless, Impossible.

MESSAGE

Data about devices; precisely identifiers, positions (open, close and broken [XX]) and kinds (breaker and isolator).

DEVICE_INFO

DEVICE_INFO
Orders sent from the automatism to the device, consisting of the identifier of the interested devices and of the position that it must reach.

ORDER

```
unit_identifier = NAT
unit_kind ::= Ae | Dd | Fa
unit_position ::= Open | Close | Close_On_A | Close_On_B | Broken
```

FUNCTIONAL UNITS

FUNCTIONAL UNITS

```
operation ::= Opr(unit_ident,unit_position)
```

OPERATION

```
device_ident = NAT
device_kind ::= Br | Is
device_position ::= OP | CL | XX
```

DEVICE_INFO

DEVICE_INFO

```
order ::= Open: device_ident | Close: device_ident
```

ORDER

```
message ::= Completed | Useless | Impossible
```

MESSAGE

STATION SCHEMA

DEVICE_INFO

The schemas of the devices composing a functional unit. Each device is represented by its identifier. An Ae has just an isolator; a Dd has an isolator on bar A, one on bar B and a breaker; an Fa has a line isolator, a breaker and an isolator on bar A and one on bar B.

DEVICE SCHEMA

DEVICE SCHEMA FUNCTIONAL UNIT

The schemas of the functional units, characterized by their identifiers and by the schema of their devices.

UNIT SCHEMA

UNIT SCHEMA

The station schemas describe the station topology and the correspondence between the functional units and the composing devices.

STATION SCHEMA

DEVICE_INFO

```
devices_schema ::=
  Ae(device_ident) |
  Dd(device_ident,device_ident,device_ident) |
  Fa(device_ident,device_ident,device_ident,device_ident)

**  The device schema of a functional unit is either the identifier of the
**  isolator (Ae), or the identifiers of the isolator on bar A, of that on bar B
**  and of the breaker (Dd), or the identifiers of the line isolator, of the
**  breaker and of the isolator on bar A and of that on bar B (Fa)
```

DEVICE SCHEMA

DEVICE SCHEMA   FUNCTIONAL UNIT

```
unit_schema ::= Fu(unit_ident,devices_schema)
```

UNIT SCHEMA

UNIT SCHEMA

```
station_schema ::=
  E |
  < unit_schema > & station_schema |
  < unit_schema unit_schema > & station_schema

**  The station is either empty or consists of a functional unit (an Fa or a Dd)
**  plus the reamining of the station  or consists of two functional units
**  (two Ae's) plus the reamining of the station
```

```
**  given a station schema, checks if a functional unit is connected to bar A/B
pr  On_BarA, On_BarB: unit_ident station_schema

ax  On_BarA(uid,<Fu(uid,dsch)> & sch)
ax  if On_BarA(uid,sch) then On_BarA(uid,<Fu(uid1,dsch)> & sch)
ax  On_BarA(uid1,<Fu(uid1,dsch1) Fu(uid2,dsch2)> & sch)
ax  if On_BarA(uid,sch) then On_BarA(uid,<Fu(uid1,dsch1) Fu(uid2,dsch2)> & sch)

ax  On_BarB(uid,<Fu(uid,dsch)> & sch)
ax  if On_BarB(uid,sch) then On_BarB(uid,<Fu(uid1,dsch)> & sch)
ax  On_BarB(uid1,<Fu(uid1,dsch1) Fu(uid2,dsch2)> & sch)
ax  if On_BarB(uid,sch) then On_BarB(uid,<Fu(uid1,dsch1) Fu(uid2,dsch2)> & sch)
ax  if On_BarB(uid,sch) then On_BarB(uid,<Fu(uid1,dsch1) Fu(uid2,dsch2)> & sch)
```

```
**  given a station schema and a functional unit identifier, returns its kind
op  Kind: station_schema unit_ident -> kind
```

```
ax  Kind(<Fu(uid,Ae(id1)) Fu(fid2,Ae(id2))> & sch,uid)=Ae
ax  Kind(<Fu(uid1,Ae(id1)) Fu(uid,Ae(id2))> & sch,uid)=Ae
ax  if uid=/=uid1 and uid=/=fid2 then
    Kind(<Fu(uid1,Ae(id1)) Fu(uid2,Ae(id2))> & sch,uid)=Kind(sch,uid)
ax  Kind(<Fu(uid,Dd(id1,id2,id3))> & sch,uid)=Dd
ax  Kind(<Fu(uid,Fa(id1,id2,id3,id4))> & sch,uid)=Fa
ax  if uid=/=uid' then  Kind(<Fu(uid',dsch)> & sch,uid)=Kind(sch,uid)
```

```
sch: station_schema                          uid uid' uid1 uid2: unit_ident
dsch dsch1 dsch2: device_schema              id1, id2, id3, id4: device_ident
```

STATION SCHEMA (continues)

```
**  given a station schema and a functional unit identifier return respectively
**  the identifier of its isolator on bar A/on bar B, if any, i.e. if it is
**  either a Dd or an Fa, of its breaker, if any, i.e. if it is either a Dd or
**  an Fa, of its isolator, if any, i.e. if it is an Ae

op  Isolator_On_A, Isolator_On_B, Breaker, Isolator:
                          station_schema unit_ident -> device_ident partial

ax  Isolator_On_A(<us1 us2> & sch,uid)=Isolator_On_A(sch,uid)
ax  Isolator_On_A(<Fu(uid,Dd(id1,id2,id3))> & sch,uid)=id1
ax  Isolator_On_A(<Fu(uid,Fa(id1,id2,id3,id4))> & sch,uid)=id3
ax  if uid =/= uid' then
    Isolator_On_A(<Fu(uid',dsch)> & sch,uid)=Isolator_On_A(sch,uid)

ax  Isolator_On_B(<us1 us2> & sch,uid) = Isolator_On_B(sch,uid)
ax  Isolator_On_B(<Fu(uid,Dd(id1,id2,id3))> & sch,uid)=id1
ax  Isolator_On_B(<Fu(uid,Fa(id1,id2,id3,id4))> & sch,uid)=id3
ax  if uid=/=uid' then
    Isolator_On_B(<Fu(uid',dsch)> & sch,uid) = Isolator_On_B(sch,uid)

ax  Breaker(<us1 us2> & sch,v) = Breaker(sch,uid)
ax  Breaker(<Fu(uid,Dd(id1,id2,id3))> & sch,uid)=id3
ax  Breaker(<Fu(uid,Fa(id1,id2,id3,id4))> & sch,uid)=id2
ax  if uid=/=uid' then  Breaker(<Fu(uid',dsch)> & sch,uid)=Breaker(sch,uid)

ax  Isolator(<Fu(uid,Ae(id1)) Fu(uid',Ae(id2))> & sch,uid)=id1
ax  Isolator(<Fu(uid',Ae(id1)) Fu(uid,Ae(id2))> & sch,uid)=id2
ax  if uid=/=uid1 and uid=/=uid2 then
    Isolator(<Fu(uid1,Ae(id1)) Fu(uid2,Ae(id2))> & sch,uid)=Isolator(sch,uid)
ax  Isolator(<us1> & sch,uid)=Isolator(sch,uid)
```

```
sch: station_schema                        uid uid' uid1 uid2: unit_ident
dsch dsch1 dsch2: devices_schema           us1 us2: unit_schema
id1,id2,id3,id4: device_ident
```

STATION SCHEMA (end)

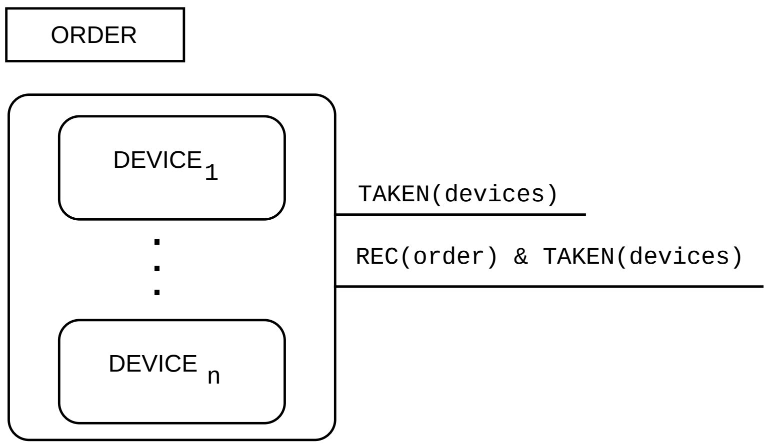
BAR: **Specification**

```
**  Electrical status of the bar:  in tension and not in tension
bar ::= OK | KO
```

BAR

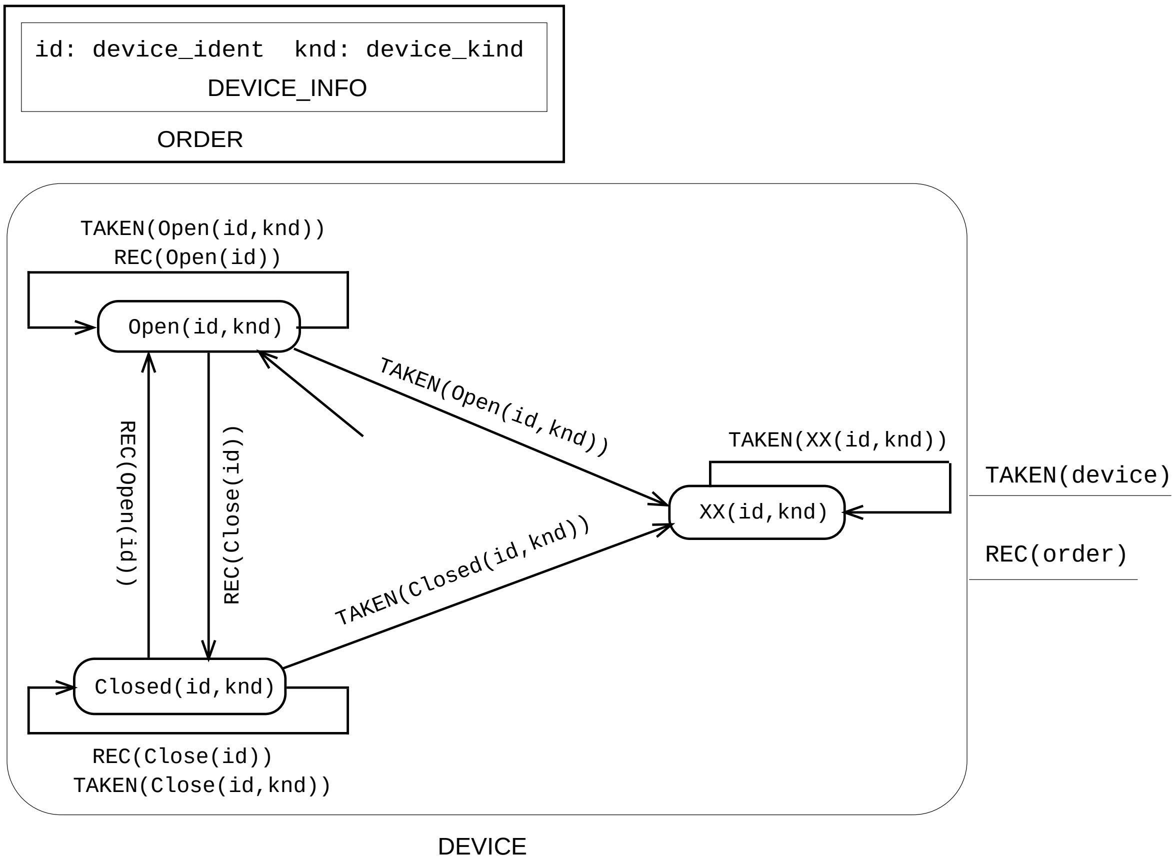
DEVICES: **Specification**

Structure & Interactions

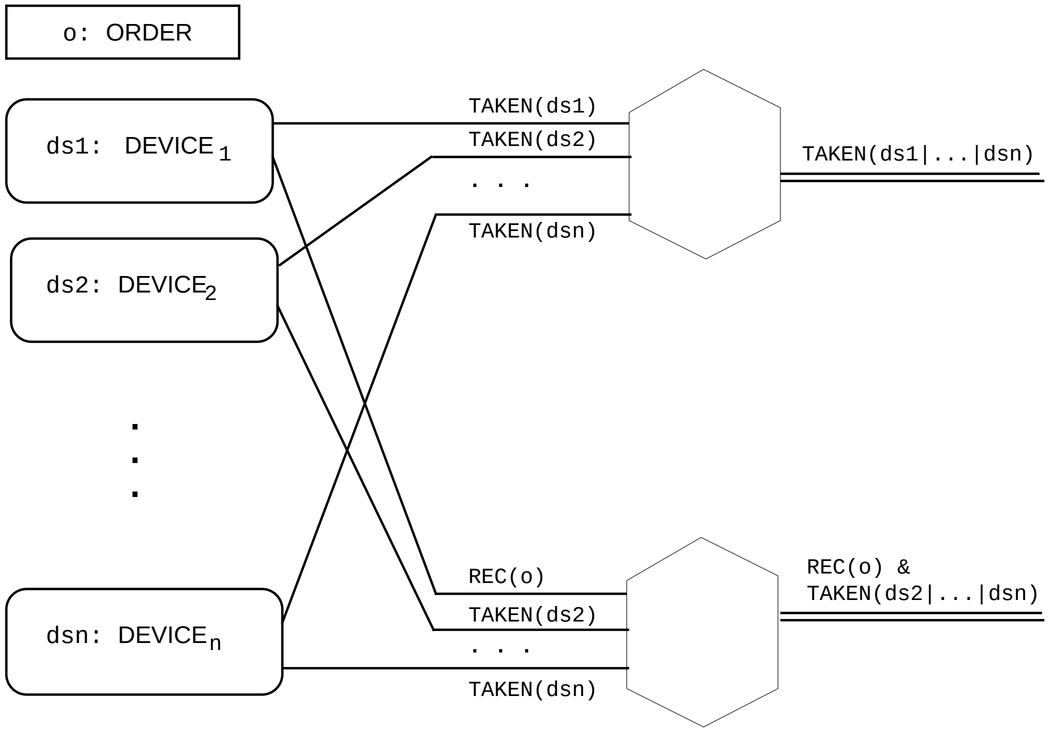


DEVICES

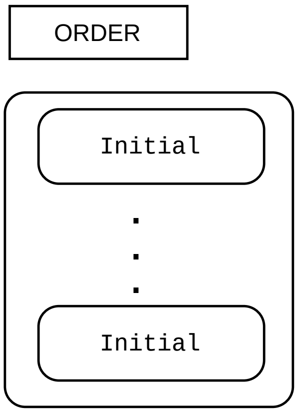
DEVICE: **Specification**



DEVICES: **Activity**



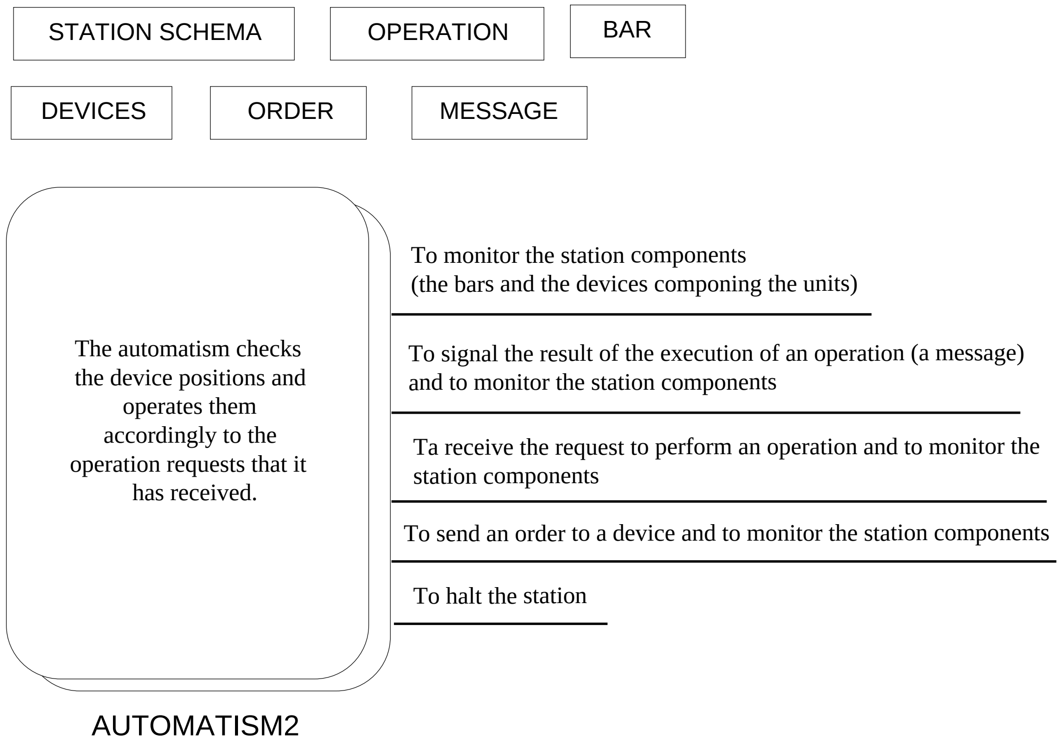
Initial States

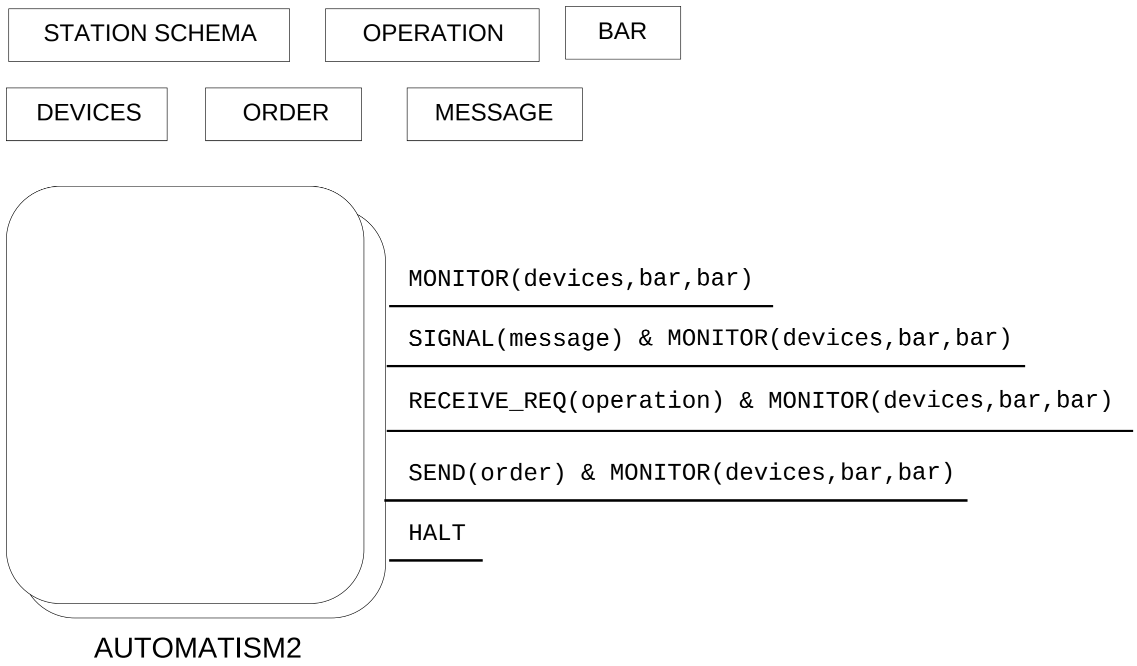


DEVICES (initial states)

AUTOMATISM.PHASE 2.step 1: **Specification**

Structure & Interactions





AUTOMATISM.PHASE 2.step 1: **States**

```
DEVICES    STATION_SCHEMA

**  checks if the automatism is in an initial state
pr  Initial: automatism

**  given an automatism returns the schema of the handled station
op  Schema: automatism -> station_schema

**  checks if the automatism is executing an operation
pr  Executing: automatism

**  returns the recorded position of a device, if any
op  Device_Position: device_ident automatism -> position partial

**  checks if the automatism has detected a failure in the station
pr  Failure: automatism
```

AUTOMATISM2 (states)

AUTOMATISM.PHASE 2.step 1: **Activity**

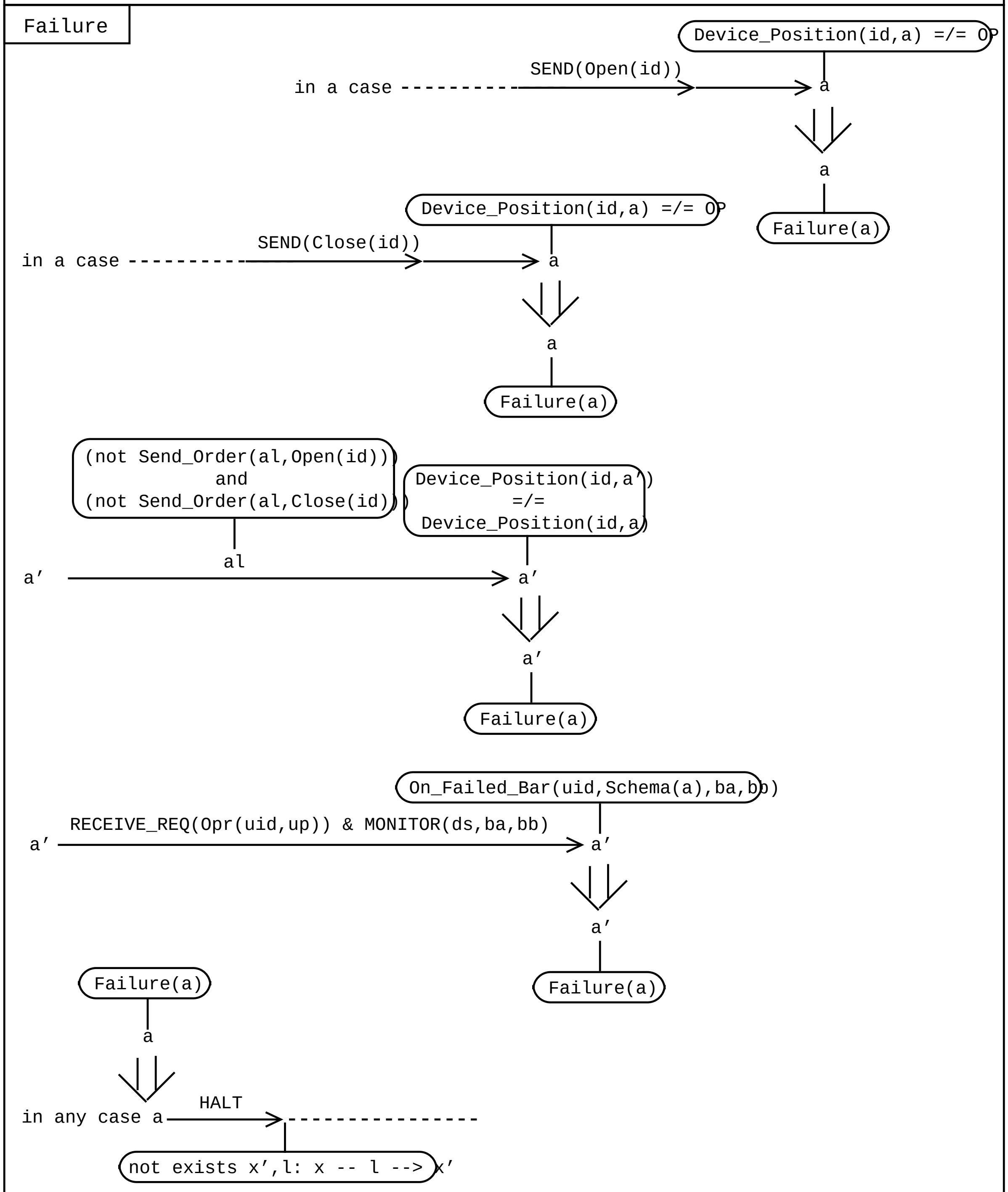
Initial

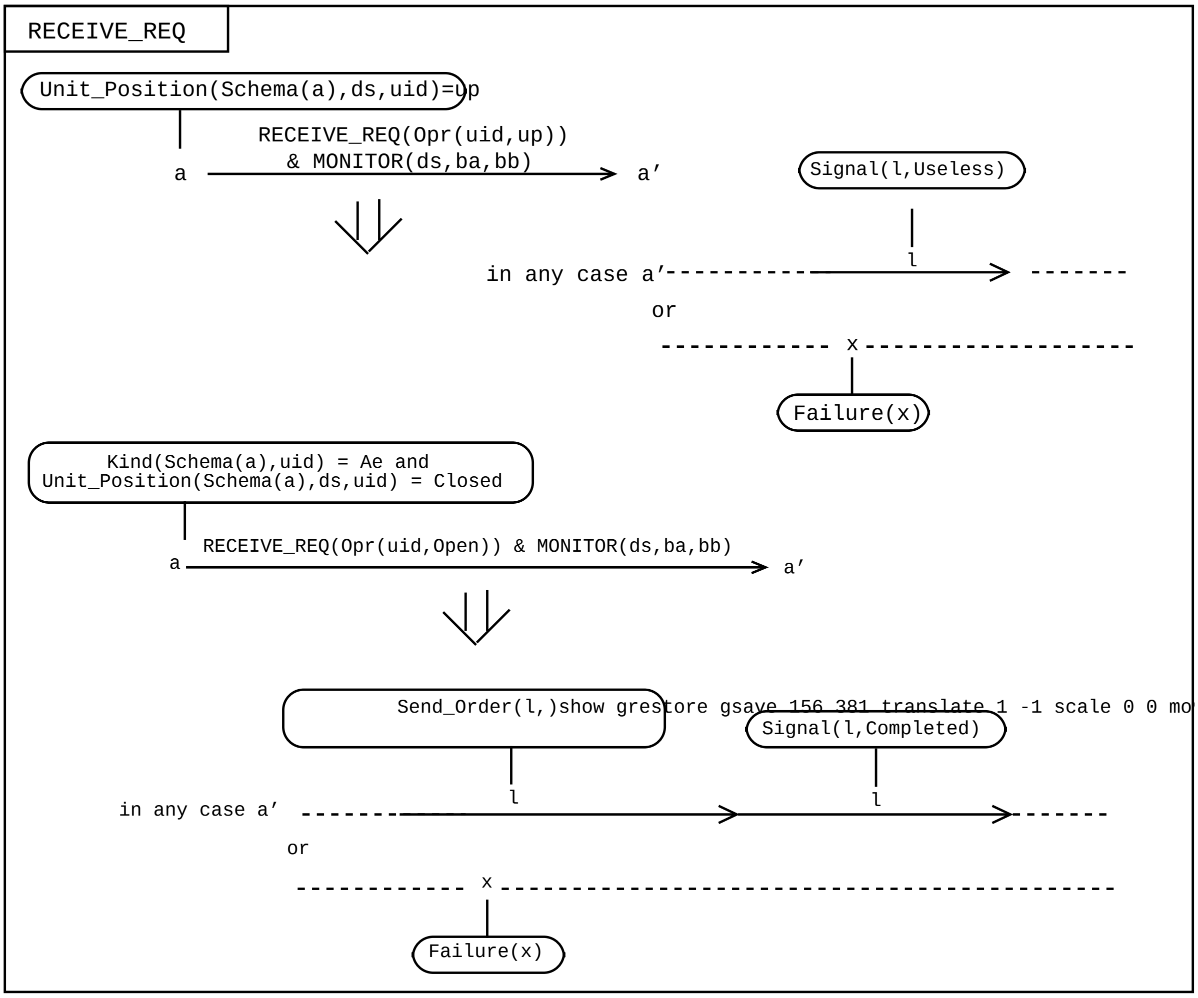
```
if Initial(a) then
 - if OK(Device_Position(id,a)) then Device_Position(id,a)=OP
 - not Executing(a)
```

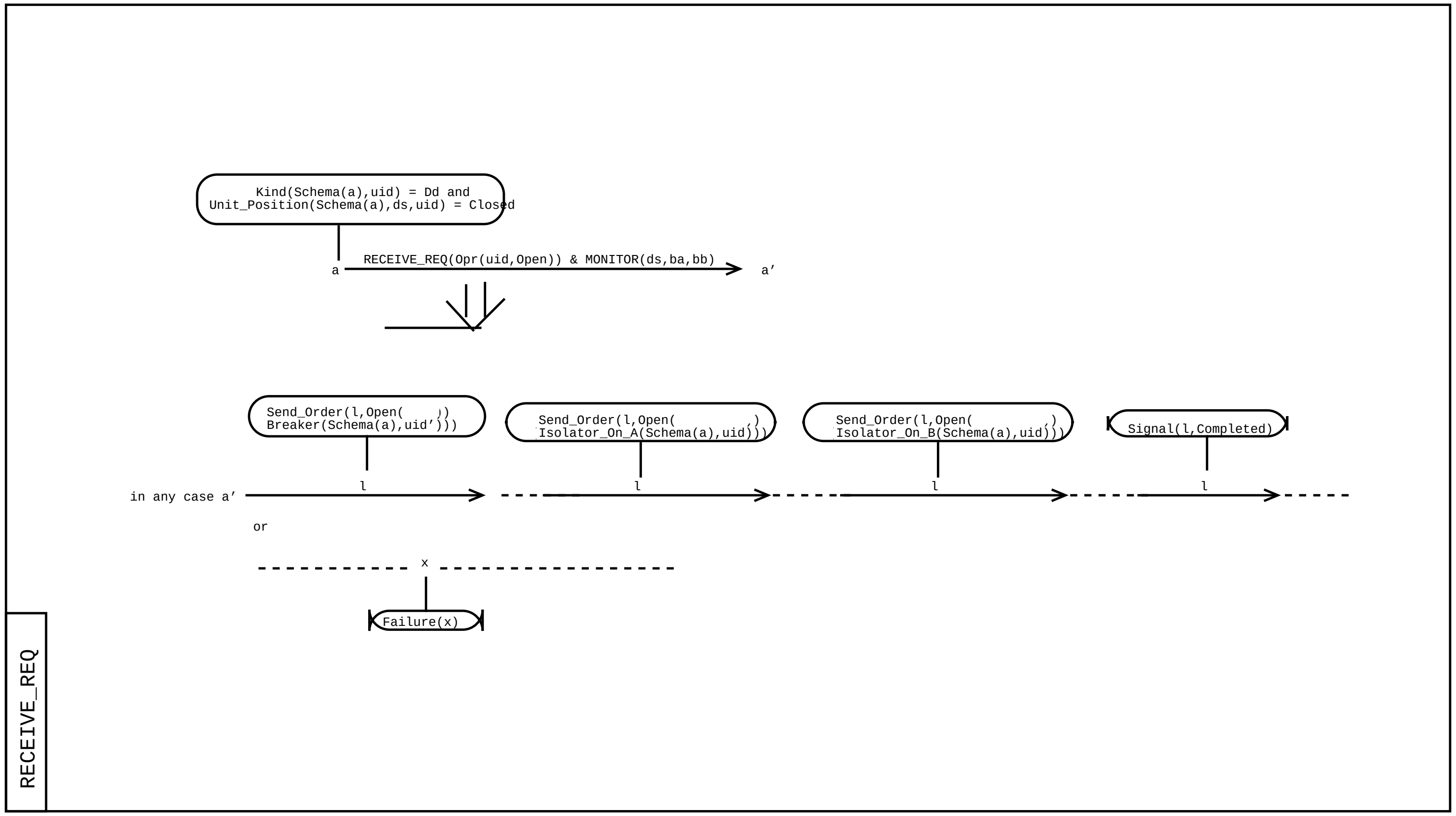
Schema

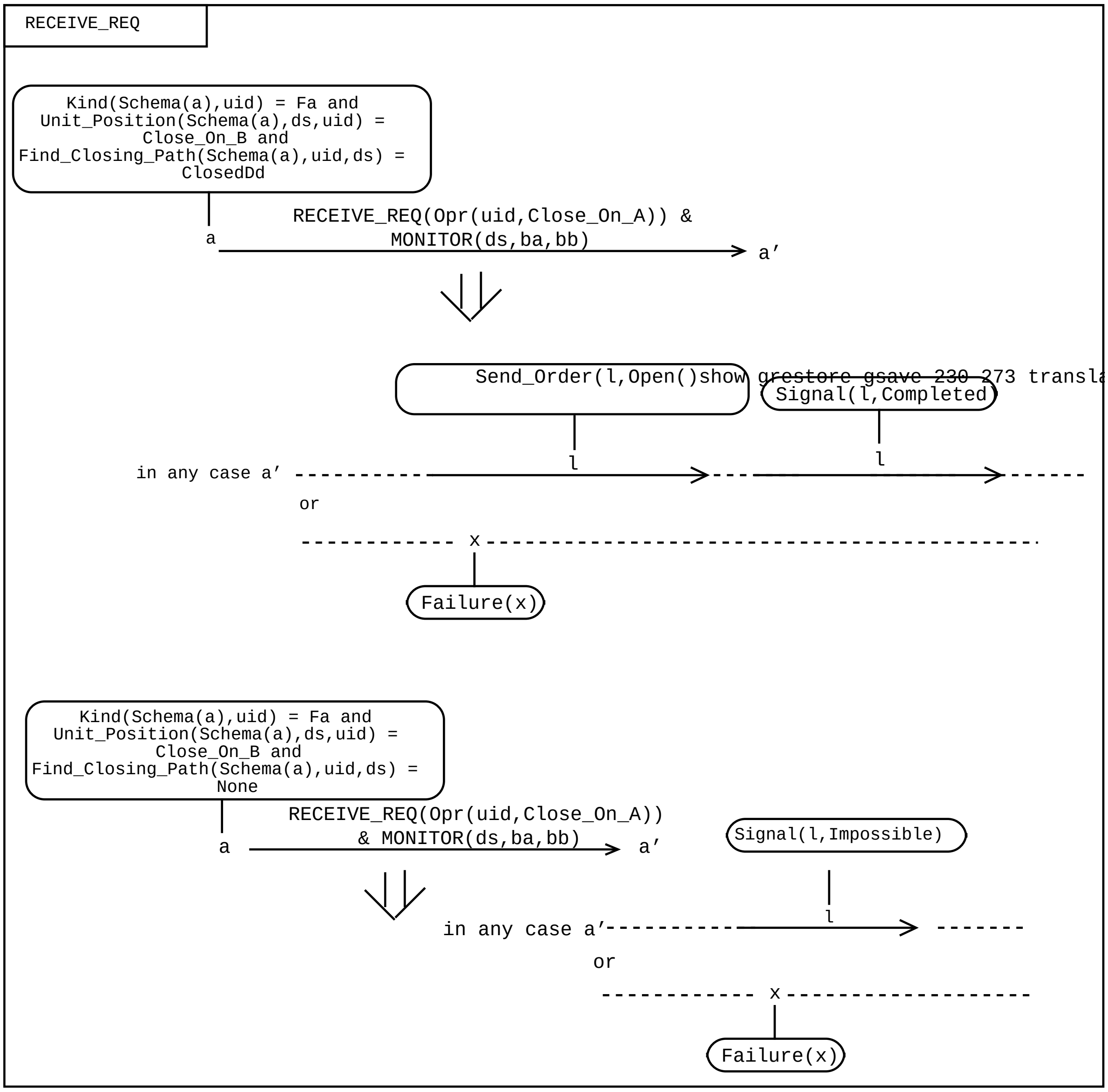
```
**  The station schema does not change
if  a -- l --> a' then Schema(a)=Schema(a')
```

Failure

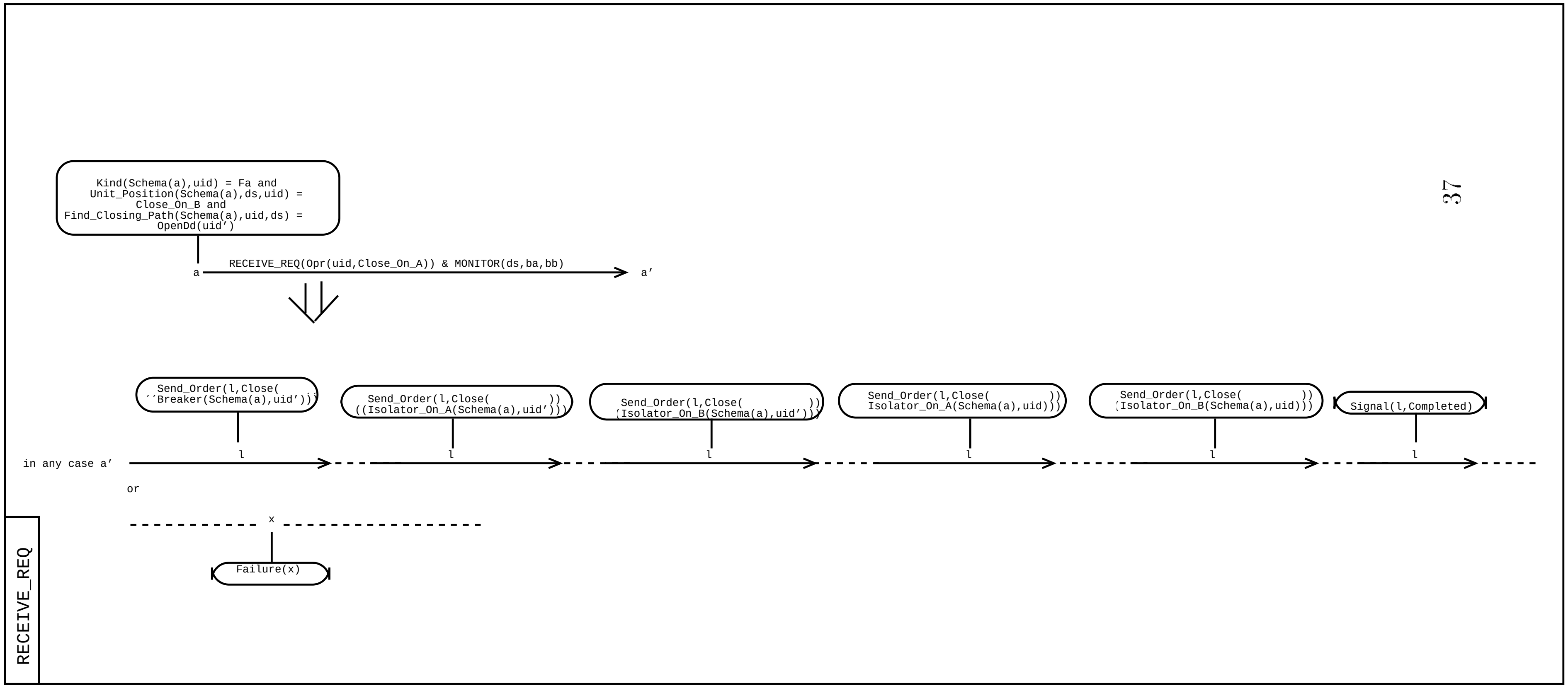



RECEIVE_REQ
Unit_Position(Schema(a),ds,uid)=up
RECEIVE_REQ(Opr(uid,up))
& MONITOR(ds,ba,bb)
a
a'
Signal(l,Useless)
l
in any case a'
or
x
Failure(x)
Kind(Schema(a),uid) = Ae and
Unit_Position(Schema(a),ds,uid) = Closed
RECEIVE_REQ(Opr(uid,Open)) & MONITOR(ds,ba,bb)
a
a'
Send_Order(l,)show grestore gsave 156 381 translate 1 -1 scale 0 0 mo
Signal(l,Completed)
l
l
in any case a'
or
x
Failure(x)


Kind(Schema(a),uid) = Dd and
Unit_Position(Schema(a),ds,uid) = Closed
a
RECEIVE_REQ(Opr(uid,Open)) & MONITOR(ds,ba,bb)
a’
Send_Order(l,Open(
Breaker(Schema(a),uid’)))
Send_Order(l,Open(
Isolator_On_A(Schema(a),uid)))
Send_Order(l,Open(
Isolator_On_B(Schema(a),uid)))
Signal(l,Completed)
in any case a’
l
l
l
l
or
x
Failure(x)
RECEIVE_REQ



RECEIVE_REQ
Kind(Schema(a),uid) = Fa and Unit_Position(Schema(a),ds,uid) = Close_On_B and Find_Closing_Path(Schema(a),uid,ds) = ClosedDd
a
RECEIVE_REQ(Opr(uid,Close_On_A)) & MONITOR(ds,ba,bb)
a’
Send_Order(l,Open()show grestore gsave 230 273 transla
Signal(l,Completed)
l
l
in any case a’
or
x
Failure(x)
Kind(Schema(a),uid) = Fa and Unit_Position(Schema(a),ds,uid) = Close_On_B and Find_Closing_Path(Schema(a),uid,ds) = None
RECEIVE_REQ(Opr(uid,Close_On_A)) & MONITOR(ds,ba,bb)
a
a’
Signal(l,Impossible)
l
in any case a’
or
x
Failure(x)


Kind(Schema(a),uid) = Fa and
Unit_Position(Schema(a),ds,uid) =
Close_On_B and
Find_Closing_Path(Schema(a),uid,ds) =
OpenDd(uid')
a
RECEIVE_REQ(Opr(uid,Close_On_A)) & MONITOR(ds,ba,bb)
a'
Send_Order(l,Close(
Breaker(Schema(a),uid')))
Send_Order(l,Close(
(Isolator_On_A(Schema(a),uid')))
Send_Order(l,Close(
Isolator_On_B(Schema(a),uid')))
Send_Order(l,Close(
Isolator_On_A(Schema(a),uid)))
Send_Order(l,Close(
Isolator_On_B(Schema(a),uid)))
Signal(l,Completed)
in any case a'
l
or
x
Failure(x)
RECEIVE_REQ

```
**  Given a station schema, the states of the devices and a functional unit
**  identifier returns the position of such unit
op  Unit_Position: station_schema devices  unit_ident -> unit_position

ax  Unit_Position(E,ds,fid)=OO
ax  Unit_Position(<FU(uid,dsch)> & sch,ds,uid)=Position(dsch,ds)
ax  if uid =/= uid’ then
    Unit_Position(<FU(uid’,dsch)> & sch,ds,uid)=Unit_Position(sch,ds,uid)
ax  Unit_Position(<FU(uid,dsch1) FU(uid2,dsch2)> & sch,ds,uid)=
    Position(dsch1,ds)
ax  Unit_Position(<FU(uid1,dsch1) FU(uid,dsch2)> & sch,ds,uid)=
    Position(dsch2,ds)
ax  if uid =/= uid1 and uid =/= uid2 then
    Unit_Position(<FU(uid1,dsch1) FU(uid2,dsch2)> & sch,ds,uid)=
    Unit_Position(sch,ds,uid)
```

```
**  checks whether an automatism interaction includes to signal a message
pr  Signalling: lab_automatism message

ax  Signalling(al,m)  iff  exists ds,ba,bb: al = SIGNAL(m) & MONITOR(ds,ba,bb)
```

```
**  checks whether an automatism interaction includes to send an order
pr  Send_Order: lab_automatism order

ax  Send_Order(al,o)  iff  exists ds,ba,bb: al = SEND(o) & MONITOR(ds,ba,bb)
```

AUTOMATISM2 (auxiliary)

Initial
If the automatism is in in an initial situaion, then all devices are open and it is not executing.
Schema
The station schema never does not change
Failure
If the automatism has detected a failure in the station, then in any case it signals that the station is going to halt and after stops If a device is in position XX, then the automatism detects a failure in the station If the automatism has sent the open order to a device id and sees that id is not open, then it detects a failure in the station If the automatism has sent the close order to a device id and sees that id is not closed, then it detects a failure in the station If a device changes position without receiving an order, then the automatism detects a failure in the station If the automatism receives an operation request for an unit connected to afailed bar, then it detects a failure in the station
On_Failed_Bar Checks whether a functional unit is connected to a failed bar
Device_Position
If the automatism monitors the plant seeing that a device id has position p, then the recorded position of id is p
Executing
If the automatism is executing and becomes not executing, then it signals the end of an operation If the automatism is not executing and become executing, then it receives an operation request
RECEIVE_REQ
If the automatism monitors the plant and receives the request of putting a unit in the actual position, then in any case either eventually it will signal that the required operation is useless or eventually it will detect a failure in the station If the automatism monitors the plant and receives the request of opening a closed Ae, then in any case either eventually it will order to the Ae's isolator of opening and after will signal that the operation has been completed or eventually it will detect a failure in the station If the automatism monitors the plant and receives the request of opening a closed Dd uid, then in any case either eventually it will order to the uid's breaker of opening, after to the uid's isolator on the bar A of opening, after to the uid's isolator on bar B of opening and after it will signal that the operation has been completed or eventually it will detect a failure in the station If the automatism monitors the plant and receives the request of closing on bar A an Fa uid closed on bar B, and there exists a closing path made by a closed Dd, then in any case either eventually it will order to the uid's isolator on A of closing, after to the uid's isolator on B of opening, and after it will signal that the operation has been completed or eventually there will be a failure in the station If the automatism monitors the plant and receives the request of closing on bar A an Fa uid closed on bar B and there exists a closing path made by the open Dd uid', then in any case either eventually it will order to the breaker of uid' of closing, after to the isolator on bar A of uid' of closing, after to the isolator on bar B of uid' of closing, after to the isolator on A of uid of closing, after to the isolator on B of uid of opening and after it will signal that the operation has been completed or eventually there will be a failure in the station If the automatism monitors the plant and receives the request of closing on bar A an Fa uid, sees that it is closed on bar B and there exists no closing path, then in any case either eventually it will signal that the required operation is impossible or eventually it will detect a failure in the station

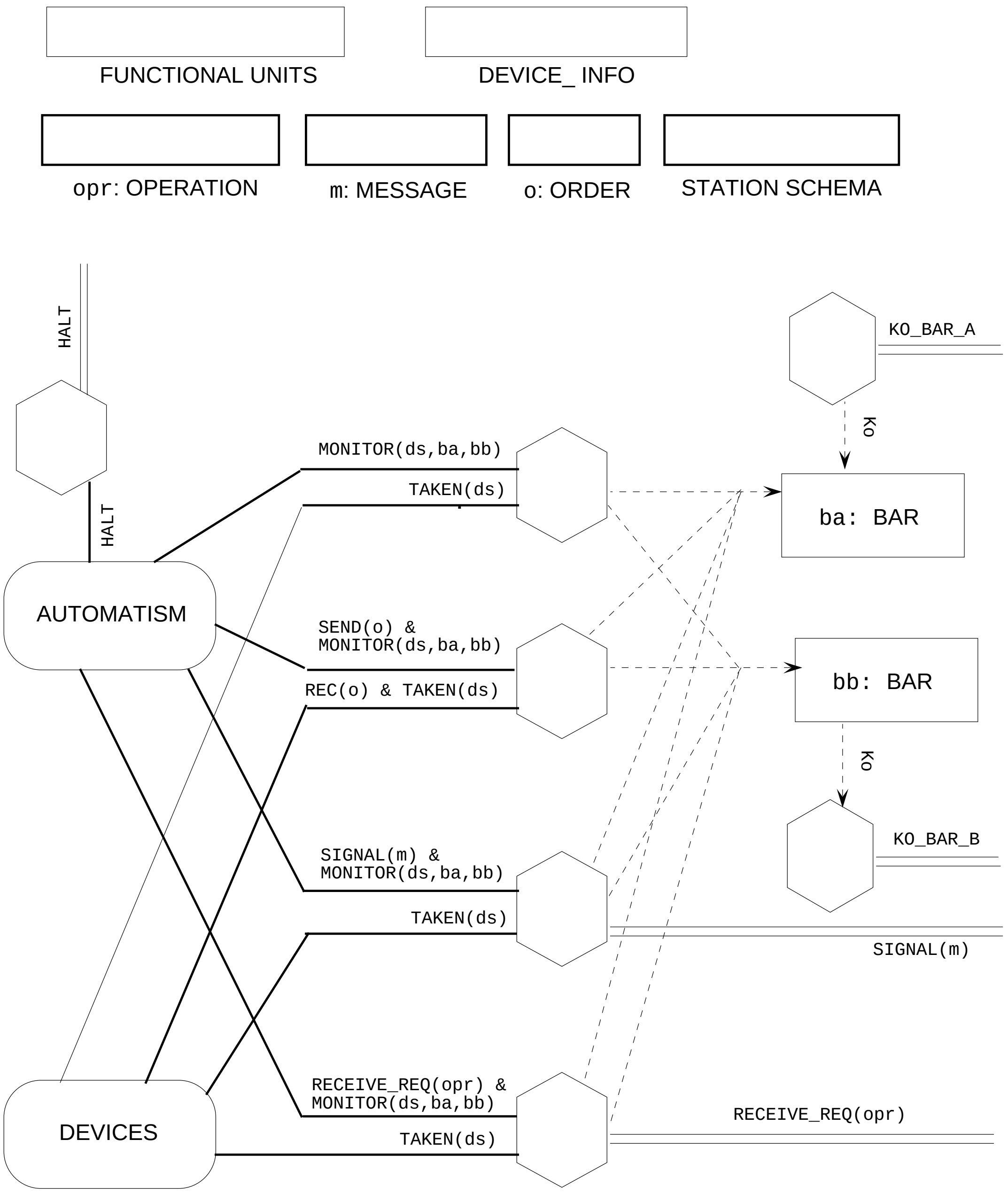
```
**  given the schema of the devices of a functional unit and the states of its
**  devices, returns the functional unit position
op  Position: devices_schema devices -> unit_position

**  The position of an Ae is equal to that of its isolator
ax  if Open(id,Is) In ds then Position(Ae(id),ds) = Open
ax  if Closed(id,Is) In ds then Position(Ae(id),ds) = Close
ax  if XX(id,Is) In ds then Position(Ae(id),ds) = Broken

**  The position of a Dd whose devices are all open is open
ax  if Open(id1,Br)|Open(id2,Is)|Open(id3,Is) SubEq ds then
    Position(Dd(id1,id2,id3),ds) = Open

**  The position of a Dd whose devices are all closed is closed
ax  if Closed(id1,Br)|Closed(id2,Br)|Closed(id3,Is) SubEq ds then
    Position(Dd(id1,id2,id3),ds) = Close

**  If a device is broken then the position of a Dd is broken
ax  if XX(id1,Br) In ds then Position(Dd(id1,id2,id3),ds) = Broken
ax  if XX(id2,Is) In ds then Position(Dd(id1,id2,id3),ds) = Broken
ax  if XX(id3,Is) In ds then Position(Dd(id1,id2,id3),ds) = Broken

**  In any other case the position of a Dd is moving
ax  if Open(id1,Br)|Closed(id2,Br) SubEq ds then Position(Dd(id1,id2,id3),ds) = OO
ax  if Open(id1,Br)|Closed(id3,Is) SubEq ds then Position(Dd(id1,id2,id3),ds) = OO
ax  if Closed(id1,Br)|Open(id2,Is) SubEq ds then Position(Dd(id1,id2,id3),ds) = OO
ax  if Closed(id1,Br)|Open(id3,Is) SubEq ds then Position(Dd(id1,id2,id3),ds) = OO

**  The position of an Fa whose devices are all open is open
ax  if Open(id1,Is)|Open(id2,Br)|Open(id3,Is)|Open(id4,Is) SubEq ds then
    Position(Fa(id1,id2,id3,id4),ds) = Open

**  The position of an Fa whose isolator is closed, the breaker is closed, the
**  isolator on bar A is closed and that on bar B is open, is closed on bar A
ax  if Closed(id1,Is)|Closed(id2,Br)|Closed(id3,Is)|Open(id4,Is) SubEq ds then
    Position(Fa(id1,id2,id3,id4),ds) = Closed_On_A

**  The position of an Fa whose isolator is closed, the breaker is closed, the
**  isolator on bar B is closed and that on bar A is open, is closed on bar B
ax  if Closed(id1,Is)|Closed(id2,Br)|Open(id3,Is)|Closed(id4,Is) SubEq ds then
    Position(Fa(id1,id2,id3,id4),ds) = Closed_On_B

**  If a device is broken then the position of an Fa is broken
ax  if XX(id1,Is) In ds then Position(Fa(id1,id2,id3,id4),ds) = Broken
ax  if XX(id2,Br) In ds then Position(Fa(id1,id2,id3,id4),ds) = Broken
ax  if XX(id3,Is) In ds then Position(Fa(id1,id2,id3,id4),ds) = Broken
ax  if XX(id4,Is) In ds then Position(Fa(id1,id2,id3,id4),ds) = Broken

ax  if Open(id1,Is)|Closed(id3,Is) SubEq ds then
    Position(Fa(id1,id2,id3,id4),ds) = OO
ax  if Open(id1,Is)|Closed(id4,Is) SubEq ds then
    Position(Fa(id1,id2,id3,id4),ds) = OO
ax  if Open(id2,Br)|Closed(id3,Is) SubEq ds then
    Position(Fa(id1,id2,id3,id4),ds) = OO
ax  if Open(id2,Br)|Closed(id3,Is) SubEq ds then
    Position(Fa(id1,id2,id3,id4),ds) = OO
ax  if Closed(id1,Is)|Open(id3,Is)|Open(id4,Is) SubEq ds then
    Position(Fa(id1,id2,id3,id4),ds) = OO
ax  if Closed(id2,Br)|Open(id3,Is)|Open(id4,Is) SubEq ds then
    Position(Fa(id1,id2,id3,id4),ds) = OO
ax  if Closed(id1,Is)|Closed(id2,Br)|Closed(id3,Is)|Closed(id4,Is) SubEq ds then
    Position(Fa(id1,id2,id3,id4),ds) = OO
```


AUTOMATISM2 (auxiliary)

```
answer ::= None | ClosedDd | OpenDd(unit_ident)

**  Find_Closing_Path given a station schema, a devices state and the identifier
**  of an Fa uid says whether either for uid there exists no closing path, or
there exists a closing
**  path made by an open or closed Dd
op  Find_Closing_Path: station_schema unit_ident devices -> answer

**  Find_Closing_Path returns closed Dd iff there exists a closed Dd on the same
**  bar section of the FA
ax  Find_Closing_Path(sch,fa,ds)=ClosedDd iff
        exists fid: Position(sch,ds,fid)=Open and Kind(sch,fid)=Dd and
              SameSection(sch,fa,fid,ds)

**  Find_Closing_Path returns an open Dd uid iff uid is an open
**  Dd on the same bar section of fa and there does not exist a closed Dd on
**  the same bar setion of fa
ax  Find_Closing_Path(sch,fa,ds)=OpenDd(uid)  iff
        (Position(sch,ds,uid)=Open and Kind(sch,uid)=Dd and
         SameSection(sch,fa,uid) and
         not exists fid':
            (Position(sch,ds,uid')=Open and Kind(sch,uid') and
             SameSection(Schema(st),fa,uid')))

**  Find_Closing_Path returns that no closing path exist iff there does not exist
**  a Dd on the same bar setion of fa
ax  Find_Closing_Path(sch,fa,ds)=None iff
        not exists fid': (Is_Dd(st,fid') and SameSection(Schema(st),fa,fid'))

     **  checks whether two units are on the same bar section in a station
     pr  SameSection: station_schema unit_ident unit_ident

     ax  if uid =/= uid1 and uid =/= uid2 and SameSection(sch,uid1,uid2,ds) then
         SameSection(Fu(uid,dsch) & sch,uid1,fid2,ds)
     ax  if Connect(sch,uid,ds) then
         SameSection(<Fu(uid,dsch)> & sch,uid,uid',ds)
     ax  if Connect(sch,uid,ds) then
         SameSection(<Fu(uid,dsch)> & sch,uid',uid,ds)
     ax  if SameSection(sch,uid1,uid2,ds) then
         SameSection(<fus1 fus2> & sch,uid1,uid2,ds)

         **  checks whether there is no cut before a given functional unit in a
         **  station
         pr  Connect: station_schema fun_unit_ident devices

         ax  if uid =/= uid' and Connect(sch,uid',ds) then
             Connect(<Fu(uid,dsch)> & sch,uid',ds)
         ax  Connect(<Fu(uid,dsch)> & sch,uid,ds)
         ax  if Position(Ae(id1),ds)=Open  and Position(Ae(id2),ds)=Open and
             Connect(sch,uid,ds) then
             Connect(<Fu(uid1,Ae(id1)) Fu(uid2,Ae(id2))> & sch,uid,ds)
```

AUTOMATISM2 (auxiliary)

STATION.PHASE 2.step 1: **Activity**



STATION.PHASE 2.step 2

At this level we specify the stations handled by the automatism designed by ENEL, because the automatism was the only part given by a requirement specification in PHASE 2.step 1, in this step we just give its design specification.

STATION.PHASE 2.step 2: Natural Description

Task of the automatism The automatism has the task of collecting information from the devices, of interpreting them for determining the positions of the corresponding functional units and of managing such devices to perform the operations required by the operator. The automatism must have a representation of the situation of the station, which evolves dynamically following the variations of situations of the physical system. Such representation contains information on the station topology and on the composing functional units.

Structure of the automatism The automatism is made by the *console*, the *coordinator*, the *bar managers* and the *functional unit managers*.

The console is the interface of the automatism towards the operator, while the bar and functional unit managers are those towards the station components; each functional unit manager is associated with a functional unit, of whom memorizes the current position depending on the positions of the component devices, and to whom sends the operations required by the operator.

The coordinator supervises the management activity, verifying the practicability of the operations; to do that it analyses the topology of the station and the positions of the functional units (information present in the functional unit managers).

Activity of the automatism When the automatism starts, each functional unit manager begins to monitor the devices of the associate functional unit; if it detects a failure, then it informs the console and the station stops. The failures of the bars have no immediate effect: the station stops when someone attempts to perform an operation on a functional unit connected to a failed bar.

If the operator requires an operation, the console sends it to the manager of the selected functional unit, which, if the functional unit is not already in the required position, requires the authorization to the coordinator and, if it receives an affirmative answer, translates the operation in orders for the single devices composing the functional unit; afterwards it communicates to the coordinator the result of the operation.

Since the operations must be done in sequential way, the console cannot receive a request from the moment of sending an operation until it receives the message about the result of the same.

In the case of bar exchange operation, the coordinator after having looked for the closing path, if the operation is impossible, then it informs directly the console and denies the authorization to the involved Fa; if it is needed to close a path, then it sends the closing operation to the manager of the Dd to be used for such operation, and when it receives the message that the operation has been successfully completed, it gives the authorization to the execution of the bar exchange to the Fa that have required it.

Components

Console The console is the interface of the automatism towards the operator; it filters the requests of operations from the operator and sends them to the functional unit managers. Moreover it receives from the coordinator and from the functional unit managers messages about the station functioning and communicates them to the operator. To perform its activity the console needs some information; in particular it must know: which operation request has received from the operator and the messages received by the other components of the automatism.

Managers Each functional unit present in the station is controlled by a manager that is the interface between the functional unit itself and the automatism. These managers have two tasks:

- to check that the devices of the associate functional unit keep their positions, sending a failure signal to the coordinator and to the console otherwise;
- to interpret the operations received either from the console or from the coordinator and managing the devices of the associate functional unit to reach the required position.

When a manager receives an operation, it checks the positions of the devices of the associate functional unit obtaining by them the position of the functional unit itself; if this is equal to that required it informs the console that the operation is useless, otherwise, if the operation arrives from the console, it requires to the coordinator the authorization for its execution and, if it receives an affirmative answer, it translates the operation into a sequence of orders for the single devices realizing it; at the end it checks the position reached by the functional unit and communicates the result to the coordinator.

For the managers of functional unit of kind Fa, the operation close on bar A (respectively on bar B) has different interpretation depending on the functional unit position: if it is open, then there is the simple closing, if it is closed on bar B (respectively on bar A), there is the bar exchange.

Coordinator The coordinator has the task of managing the activity of the functional units (through their managers), depending on the operation required by the operator by means of the console, and on the information on the situation of the station obtained by combining those known from the managers (current situations of the various units and bars), with those contained in the schema (topology of the station). Moreover it transmits to the console the messages about to the result of the operation. Another task of the coordinator is the control of the situation of the station: if it finds a failure, then it orders to all managers and to the console the end of the activity.

When the coordinator receives from a manager the authorization request for executing an operation on a functional unit, then this is ready for such execution, i.e., it is not already in the required position and there are not failures in the station.

The coordinator manages in different way the three kinds of operations: opening, closing and bar exchange.

In each case it checks that the operation is valid, i.e., that the bars connected to the functional unit to be used are not failed; to do that it reads in the corresponding managers the situations of the involved bars and if one of them is failed, it informs the console and all managers that there is a failure.

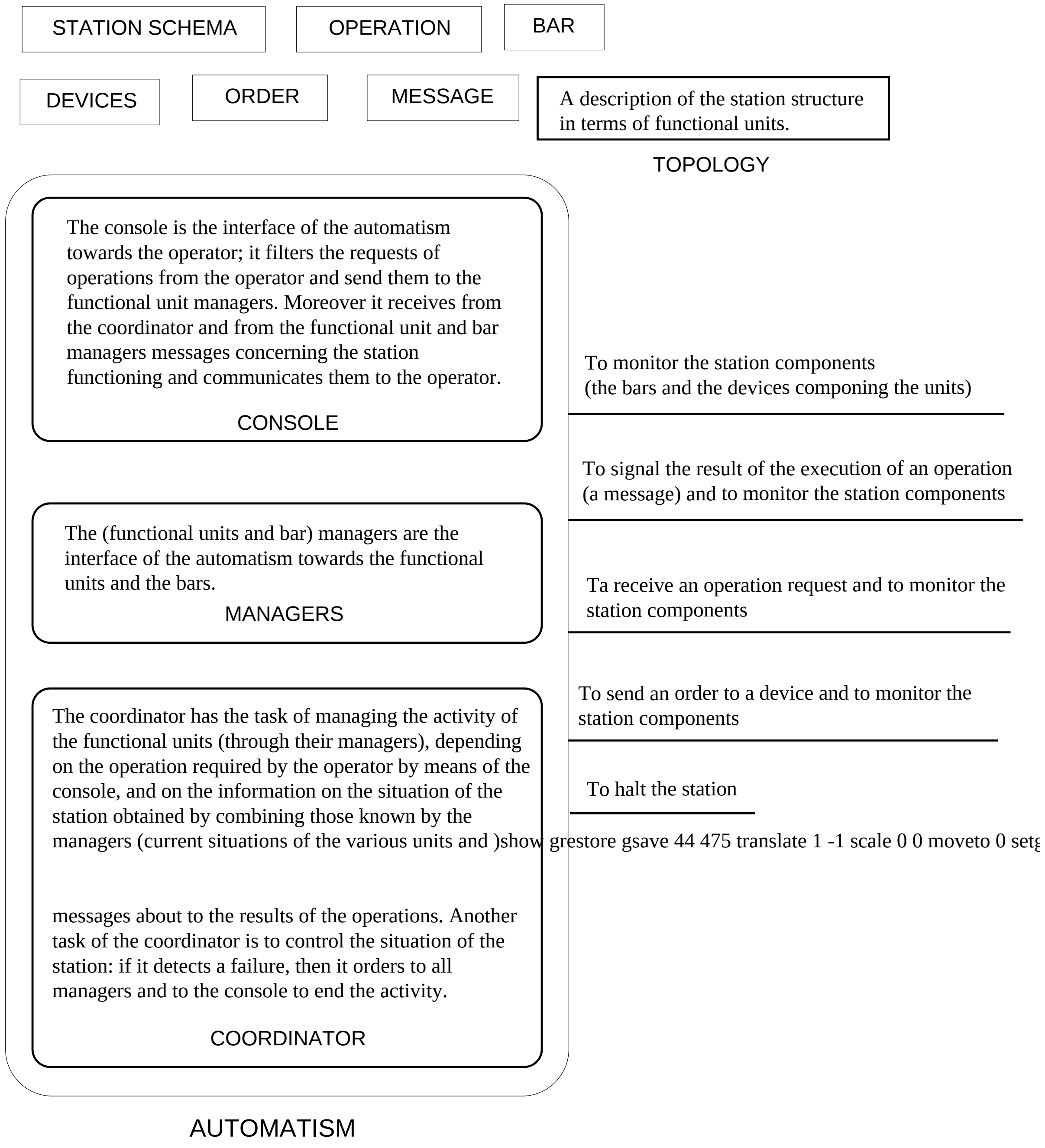
In the case of closing on a bar of a functional unit of kind Fa, the coordinator must determine if the operation is either of bar exchange or of closing; to do that it checks the situation of such Fa (reading it in the corresponding manager).

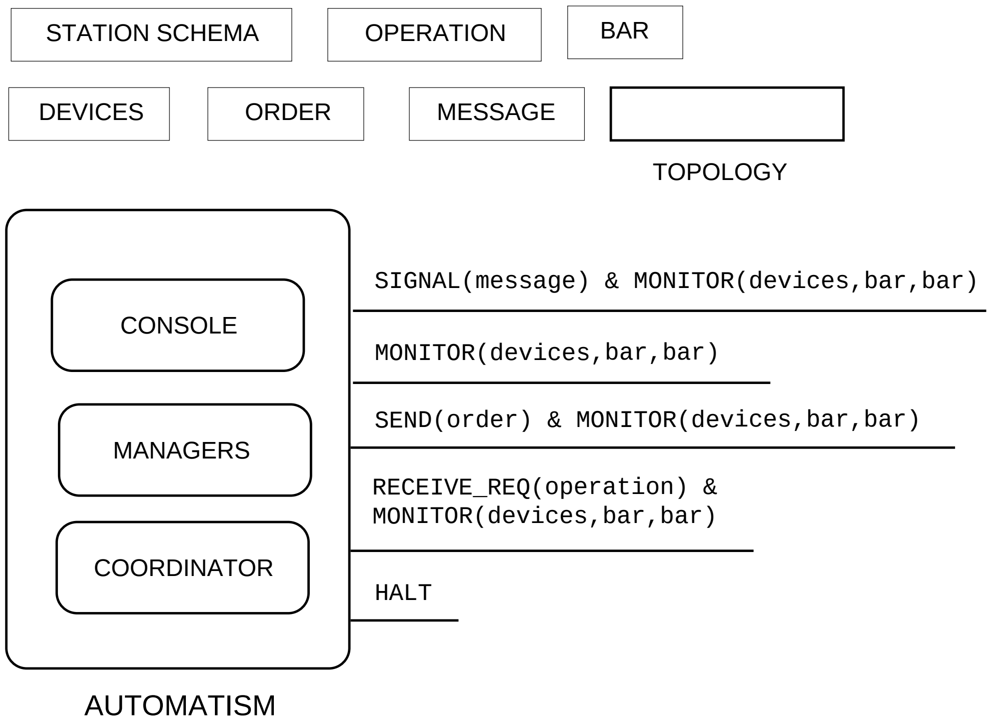
For the bar exchange operation, it must analyze the schema for determining the closing path:

- if there is already a closed Dd, it allows the operation;
- if it is needed to close a proper or not proper Dd but electrically connected, i.e., such that the isolators that divides the pieces of bar of the Fa and of the Dd are closed, it orders to the Dd to close and, after that, allows the operation of the Fa;
- if it is needed to close a not proper Dd and not electrically connected, it does not allow the operation of the Fa and informs the console that it is needed to close an Ae.

AUTOMATISM.PHASE 2.step 2: **Specification**

Structure & Interactions




STATION SCHEMA
OPERATION
BAR
DEVICES
ORDER
MESSAGE
TOPOLOGY
CONSOLE
MANAGERS
COORDINATOR
AUTOMATISM
SIGNAL(message) & MONITOR(devices,bar,bar)
MONITOR(devices,bar,bar)
SEND(order) & MONITOR(devices,bar,bar)
RECEIVE_REQ(operation) & MONITOR(devices,bar,bar)
HALT

FUNCTIONAL UNITS

```
**  The station topology, i.e. a description of the station structure in terms of
**  functional units

topology ::=
  E  |
  Fa(_) & _: unit_ident topology |
  Dd(_) & _: unit_ident topology  |
  Ae(_, _) & _: unit_ident unit_ident topology
```

```
**  given a topology return the station parts on the right and on the left of a
**  functional unit respectively
op  RPart, LPart: topology unit_ident -> topology

ax  RPart(Fa(fid) & tp,fid) = tp
ax  RPart(Dd(fid) & tp,fid) = tp
ax  RPart(Ae(fid1,fid) & tp,fid) = tp
ax  RPart(Ae(fid,fid1) & tp,fid) = tp
ax  if fid =/= fid1 then RPart(Fa(fid1) & tp,fid) = RPart(tp,fid)
ax  if fid =/= fid1 then RPart(Dd(fid1) & tp,fid) = RPart(tp,fid)
ax  if fid =/= fid1 and fid =/= fid2 then
    RPart(Ae(fid1,fid) & tp,fid) = RPart(tp,fid)
ax  LPart(tp,fid) = LPart1(tp,fid,E)
```

```
op  LPart1: topology unit_ident topology -> topology
ax  LPart1(Dd(fid) & tp,fid,tp1) = tp1
ax  LPart1(Fa(fid) & tp,fid,tp1) = tp1
ax  LPart1(Ae(fid1,fid) & tp,fid,tp1) = tp1
ax  LPart1(Ae(fid,fid1) & tp,fid,tp1) = tp1
ax  if fid =/= fid1 then
    LPart1(Fa(fid1) & tp,fid,tp1) = LPart1(tp,fid,Fa(fid1) & tp1)
ax  if fid =/= fid1 then
    LPart1(Dd(fid1) & tp,fid,tp1) = LPart1(tp,fid,Dd(fid1) & tp1)
ax  if fid =/= fid1 and fid =/= fid2 then
    LPart1(Ae(fid1,fid2) & tp,fid,tp1) = LPart1(tp,fid,Ae(fid1,fid2) & tp1)
```

```
**  given a station topology and a functional unit identifier returns its kind,
**  if it is part ofthe station
op  Kind: topology unit_ident -> kind (partial)

ax  Kind(Ae(fid,fid1) & tp,fid) = Ae
ax  Kind(Ae(fid,fid1) & tp,fid1) = Ae
ax  Kind(Dd(fid) & tp,fid) = Dd
ax  Kind(Fa(fid) & tp,fid) = Fa
ax  if fid =/= fid1 then  Kind(Dd(fid) & tp,fid1) = Kind(tp,fid1)
ax  if fid =/= fid1 then  Kind(Fa(fid) & tp,fid1) = Kind(tp,fid1)
ax  if fid =/= fid1 and fid =/= fid2 then
    Kind(Ae(fid1,fid2) & tp,fid) = Kind(tp,fid)
```

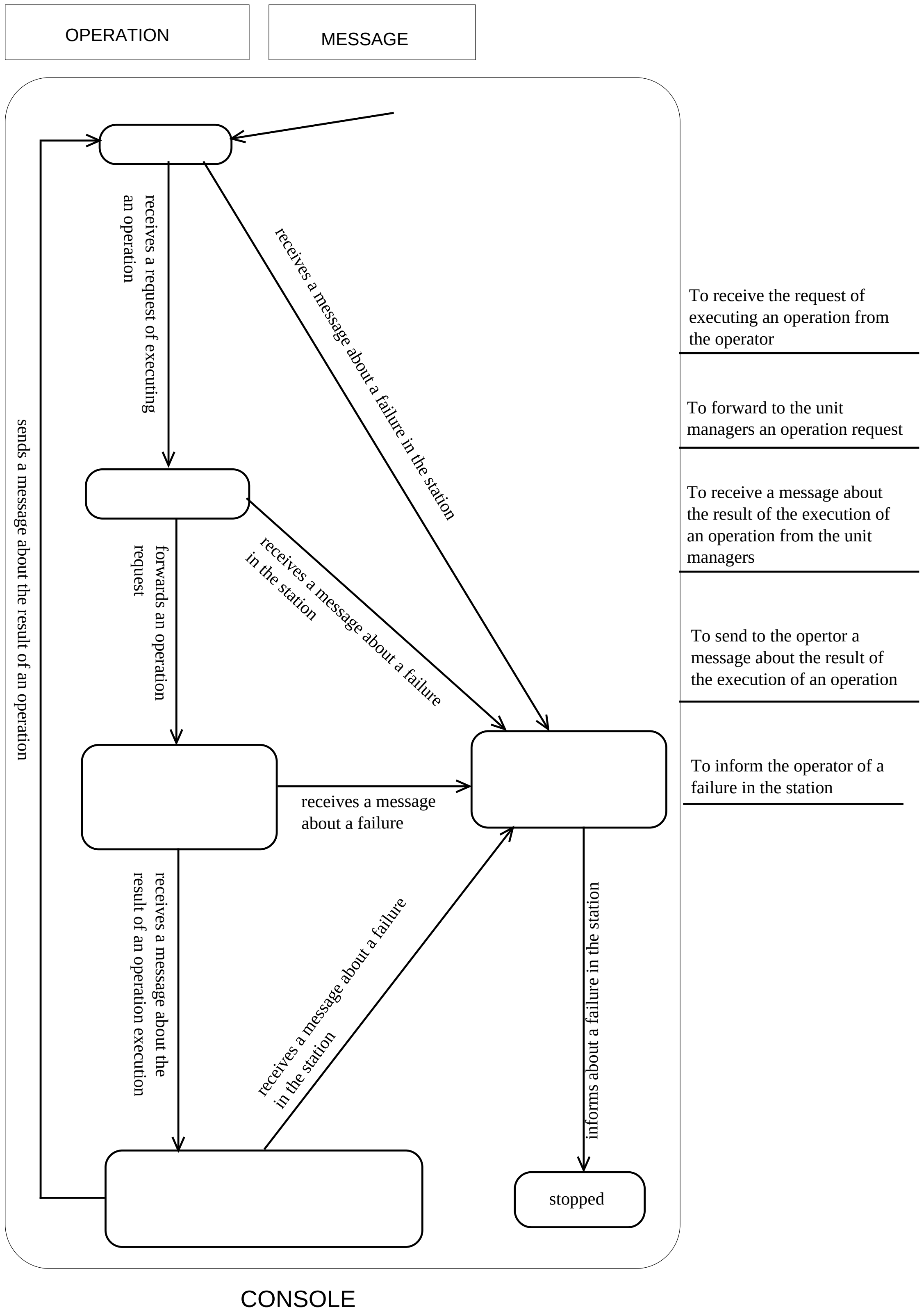
TOPOLOGY (continues)

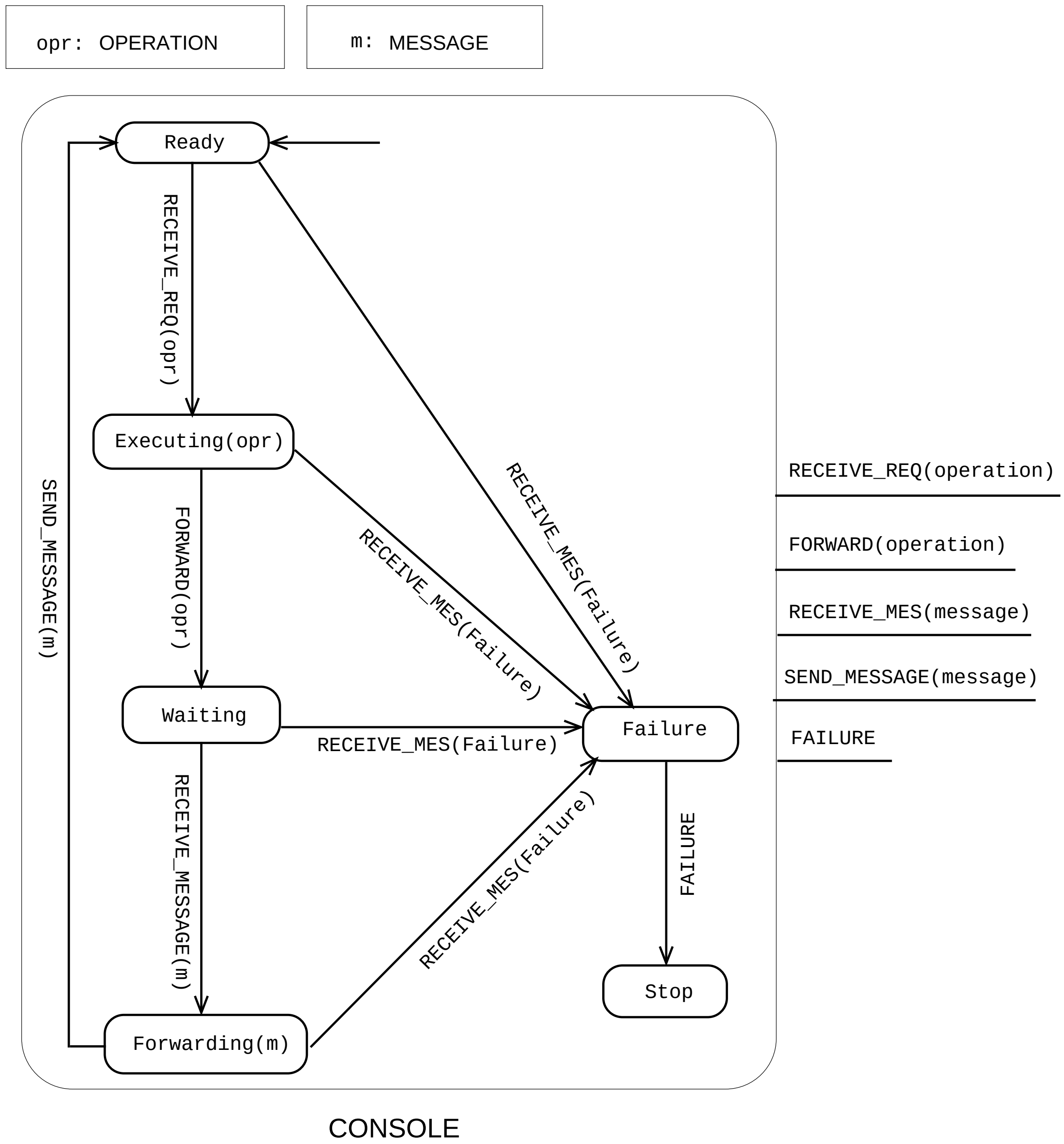
```
**  given a station topology checks whether in such topology a functional unit
**  is connected to a bar
pr  On_BarA, On_BarB: unit_ident topology

ax  On_BarA(uid,Fa(uid) & tp)
ax  On_BarA(uid,Dd(uid) & tp)
ax  On_BarA(uid,Ae(uid,uid1) & tp)
ax  if On_BarA(uid,tp) then On_BarA(uid,Fa(uid1) & tp)
ax  if On_BarA(uid,tp) then On_BarA(uid,Dd(uid1) & tp)
ax  if On_BarA(uid,tp) then On_BarA(uid,Ae(uid1,uid2) & tp)

ax  On_BarB(uid,Fa(uid) & tp)
ax  On_BarB(uid,Dd(uid) & tp)
ax  On_BarB(uid,Ae(uid1,uid) & tp)
ax  if On_BarB(uid,tp) then On_BarA(uid,Fa(uid1) & tp)
ax  if On_BarB(uid,tp) then On_BarA(uid,Dd(uid1) & tp)
ax  if On_BarB(uid,tp) then On_BarA(uid,Ae(uid1,uid2) & tp)
```

```
 tp tp1: topology            uid uid1 uid2: unit_ident
```

TOPOLOGY (end)

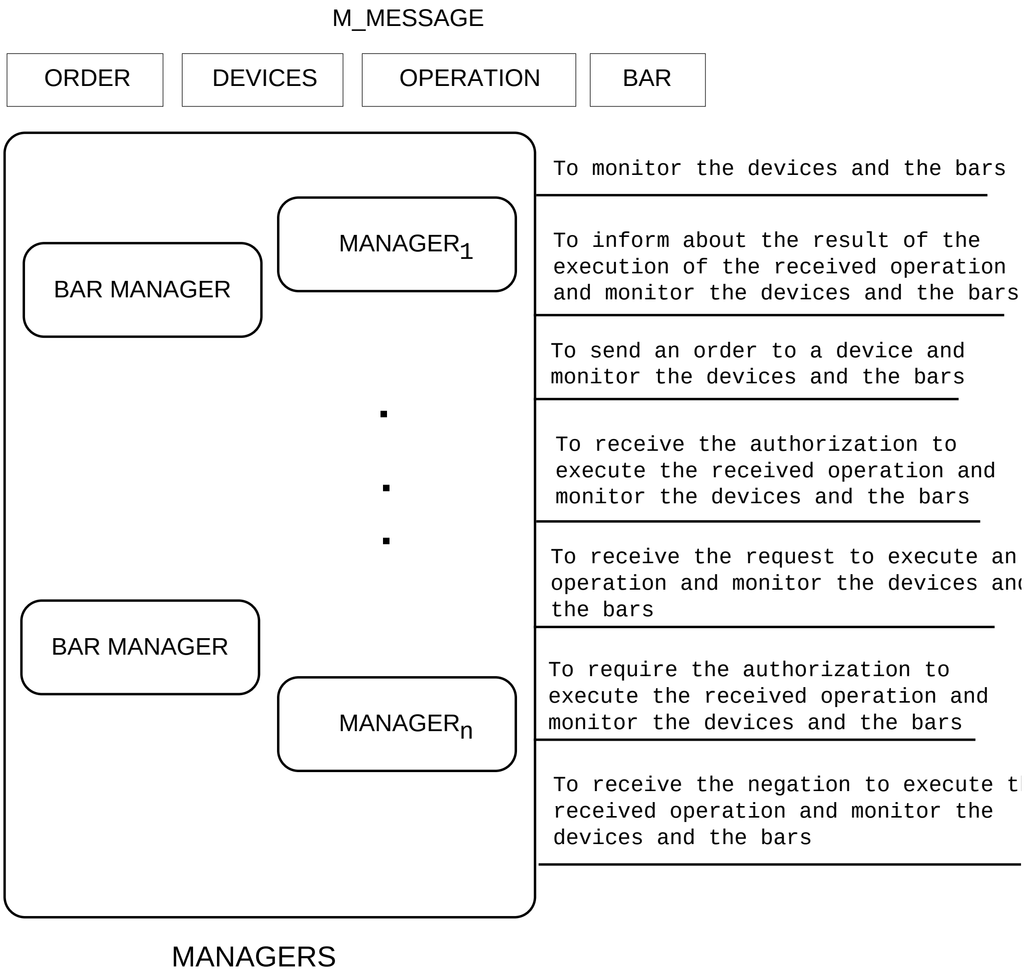
CONSOLE: **Specification**

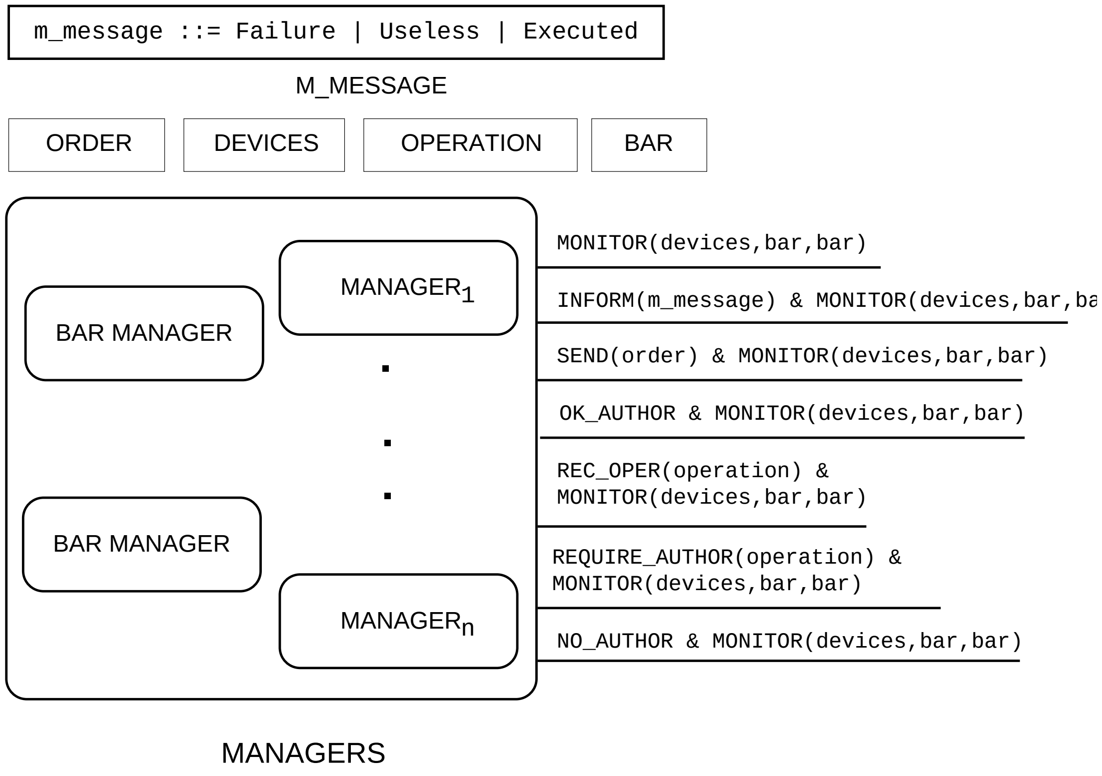



opr: OPERATION
m: MESSAGE
Ready
RECEIVE_REQ(opr)
Executing(opr)
FORWARD(opr)
Waiting
RECEIVE_MESSAGE(m)
Forwarding(m)
SEND_MESSAGE(m)
RECEIVE_MES(Failure)
RECEIVE_MES(Failure)
RECEIVE_MES(Failure)
RECEIVE_MES(Failure)
Failure
FAILURE
Stop
CONSOLE
RECEIVE_REQ(operation)
FORWARD(operation)
RECEIVE_MES(message)
SEND_MESSAGE(message)
FAILURE

MANAGERS: **Specification**

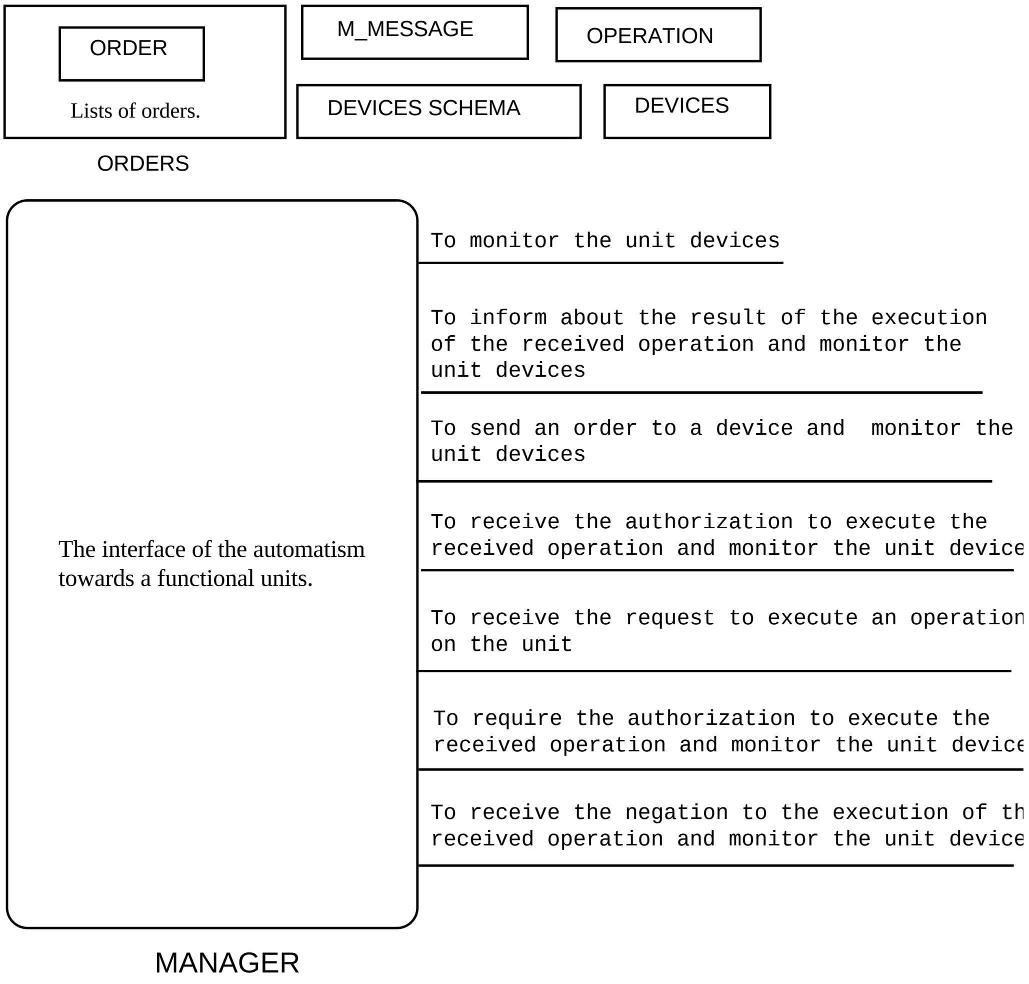
Manager messages: there is a failure, the required operation is useless/has been executed.

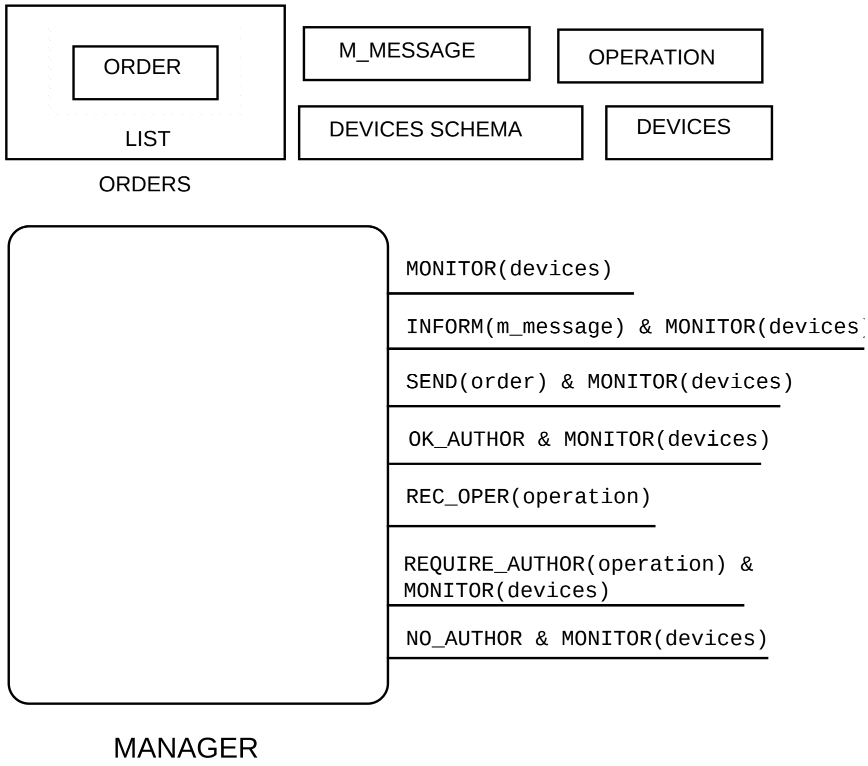



m_message ::= Failure | Useless | Executed
M_MESSAGE
ORDER
DEVICES
OPERATION
BAR
BAR MANAGER
BAR MANAGER
MANAGER1
MANAGERn
MONITOR(devices,bar,bar)
INFORM(m_message) & MONITOR(devices,bar,ba
SEND(order) & MONITOR(devices,bar,bar)
OK_AUTHOR & MONITOR(devices,bar,bar)
REC_OPER(operation) &
MONITOR(devices,bar,bar)
REQUIRE_AUTHOR(operation) &
MONITOR(devices,bar,bar)
NO_AUTHOR & MONITOR(devices,bar,bar)
MANAGERS

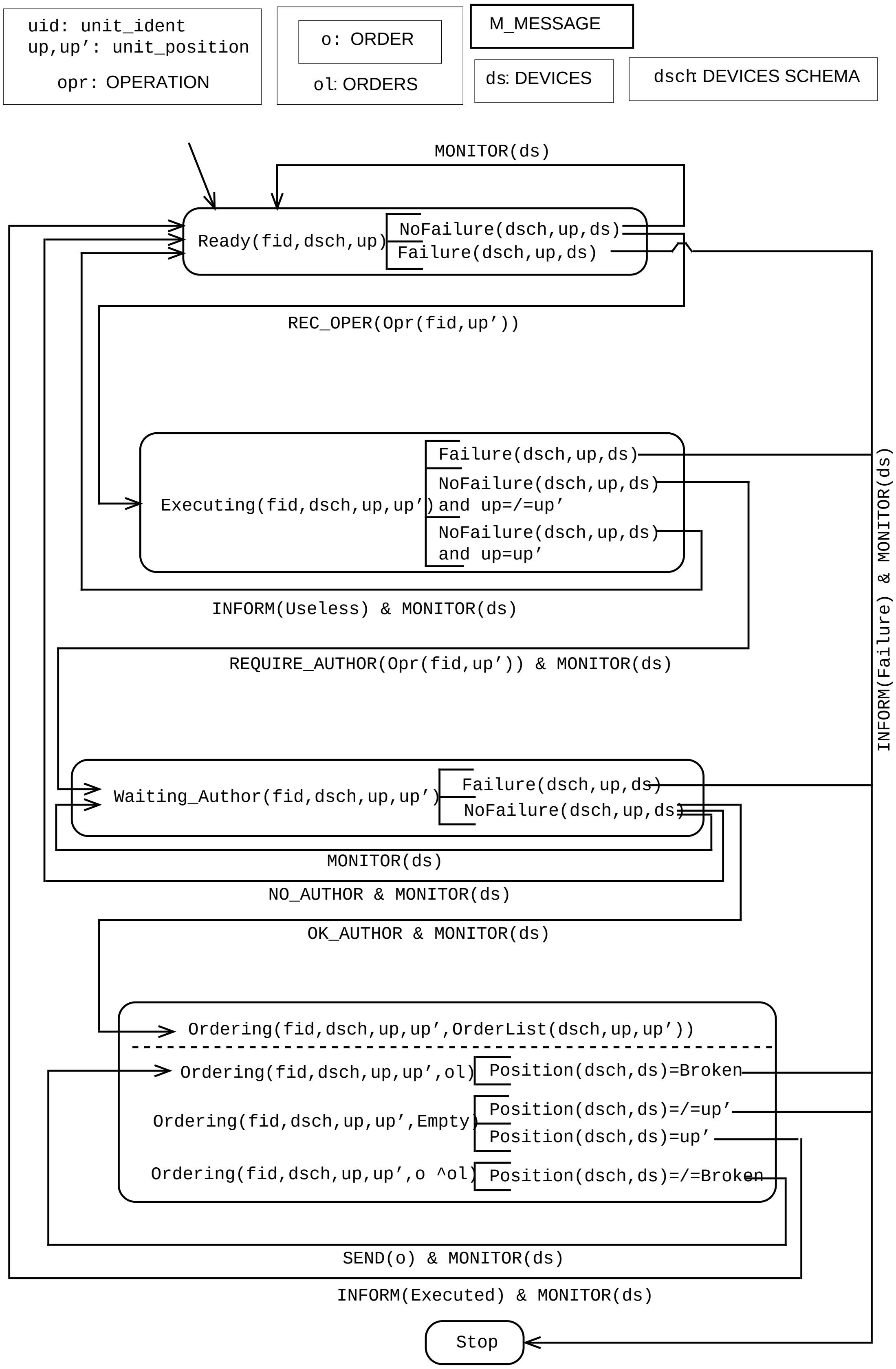
MANAGER: Specification

Structure & Interactions




ORDER
LIST
ORDERS
M_MESSAGE
OPERATION
DEVICES SCHEMA
DEVICES
MONITOR(devices)
INFORM(m_message) & MONITOR(devices)
SEND(order) & MONITOR(devices)
OK_AUTHOR & MONITOR(devices)
REC_OPER(operation)
REQUIRE_AUTHOR(operation) & MONITOR(devices)
NO_AUTHOR & MONITOR(devices)
MANAGER

MANAGER: **Activity**



```
**  given the schema of a functional unit and the set of the states of its
**  devices returns its position
op  Position: devices_schema devices -> unit_position         partial

**  The position of an Ae is equal to that of its isolator
ax  if Open(id,Is) In ds then Position(Ae(id),ds) = Open
ax  if Closed(id,Is) In ds then Position(Ae(id),ds) = Close
ax  if XX(id,Is) In ds then Position(Ae(id),ds) = Broken

**  The position of a Dd whose devices are all open is open
ax  if Open(id1,Br) | Open(id2,Is) | Open(id3,Is) SubEq ds then
    Position(Dd(id1,id2,id3),ds) = Open

**  The position of a Dd whose devices are all closed is closed
ax  if Closed(id1,Br) | Closed(id2,Br) | Closed(id3,Is) SubEq ds then
    Position(Dd(id1,id2,id3),ds) = Close

**  If a device is broken then the position of a Dd is broken
ax  if XX(id1,Br) In ds then Position(Dd(id1,id2,id3),ds) = Broken
ax  if XX(id2,Is) In ds then Position(Dd(id1,id2,id3),ds) = Broken
ax  if XX(id3,Is) In ds then Position(Dd(id1,id2,id3),ds) = Broken

**  The position of an Fa whose devices are all open is open
ax  if Open(id1,Is) | Open(id2,Br) | Open(id3,Is) | Open(id4,Is) SubEq ds then
    Position(Fa(id1,id2,id3,id4),ds)=Open

**  The position of an Fa whose isolator is closed, the breaker is closed, the
**  isolator on bar A is closed and that on bar B is open, is closed on bar A
ax  if Closed(id1,Is)|Closed(id2,Br)|Closed(id3,Is)|Open(id4,Is) SubEq ds then
    Position(Fa(id1,id2,id3,id4),ds)=Close_On_A

**  The position of an Fa whose isolator is closed, the breaker is closed, the
**  isolator on bar B is closed and that on bar A is open, is closed on bar B
ax  if Closed(id1,Is)|Closed(id2,Br)|Open(id3,Is)|Closed(id4,Is) SubEq ds then
    Position(Fa(id1,id2,id3,id4),ds)=Close_On_B

**  If a device is broken then the position of an Fa is broken
ax  if XX(id1,Is) In ds then Position(Fa(id1,id2,id3,id4),ds)=Broken
ax  if XX(id2,Br) In ds then Position(Fa(id1,id2,id3,id4),ds)=Broken
ax  if XX(id3,Is) In ds then Position(Fa(id1,id2,id3,id4),ds)=Broken
ax  if XX(id4,Is) In ds then Position(Fa(id1,id2,id3,id4),ds)=Broken
```

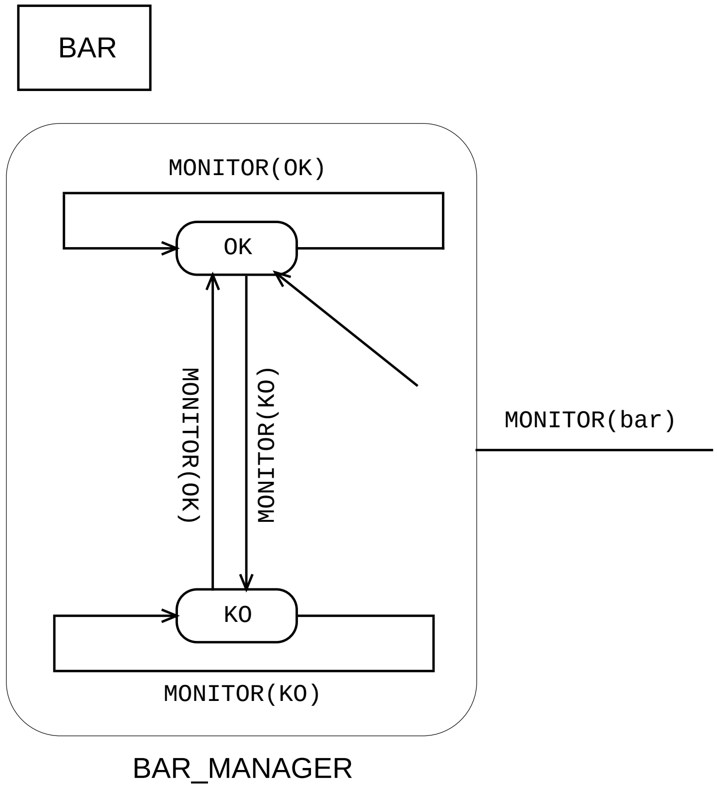
```
**  checks if there is/is not a failure in the managed functional unit
pr  Failure, No_Failure: devices_schema unit_position devices

ax if Position(dsch,ds) =/= Broken then  No_Failure(dsch,Position(dsch,ds),ds)
ax if Position(dsch,ds) = Broken  then Failure(dsch,up,ds)
ax if Position(dsch,ds) =/= up then Failure(dsch,up,ds)
```

auxiliary (continues)

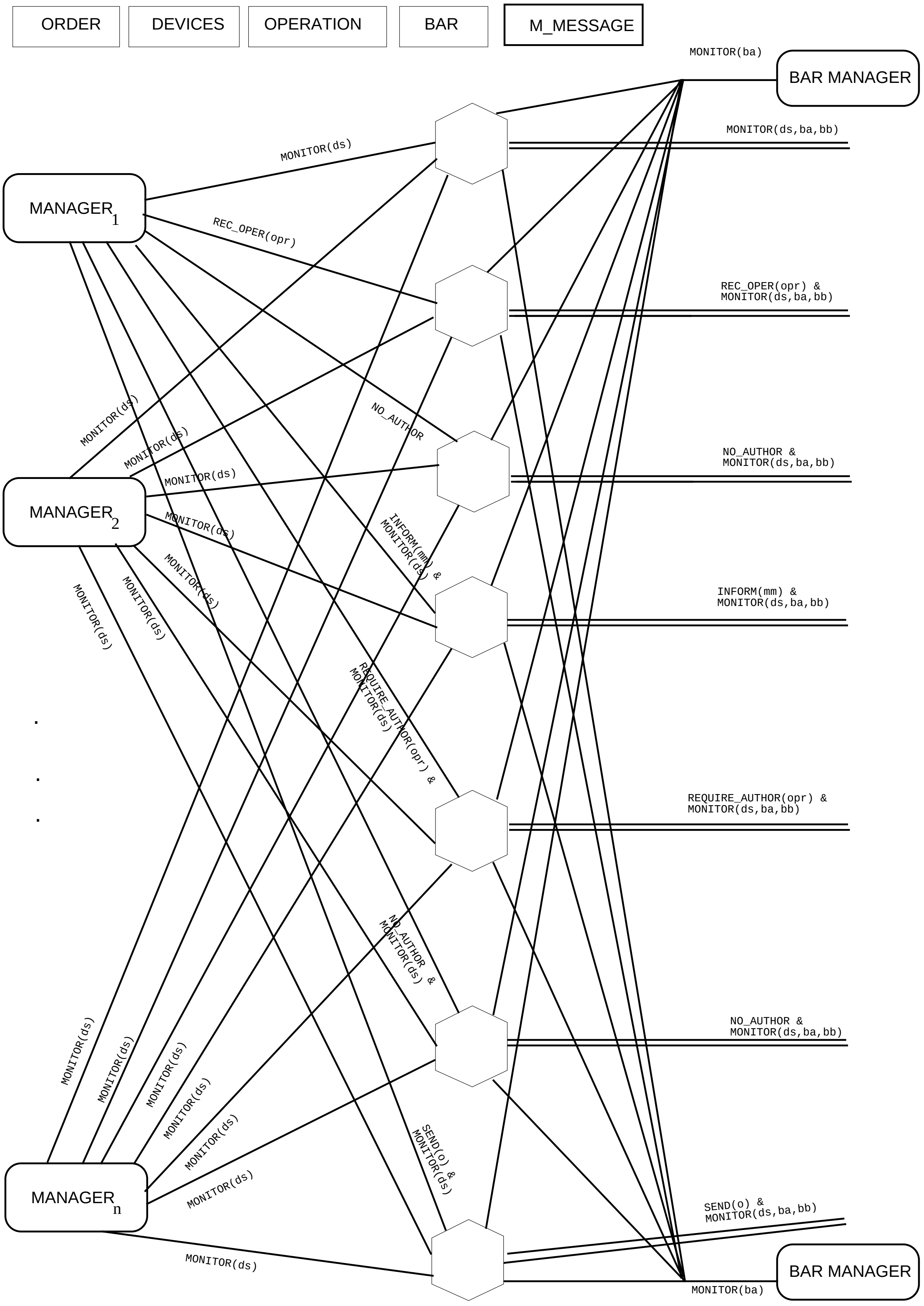
```
**  given the schema a functional unit, its position and the position to reach,
**  returns the list of the orders to be sent to its devices
op  Order_List: devices_schema unit_position unit_position -> orders

**  The order corresponding to an operation on an Ae is go to the required
**  position
ax  Order_List(Ae(id),up,Open) = Open(id) Empty
ax  Order_List(Ae(id),up,Close) = Close(id) Empty

**  The orders corresponding to opening a Dd are: open the breaker, the isolator
**  on bar A and then the isolator on bar B
ax  Order_List(Dd(id1,id2,id3),up,Open) = Open(id1) Open(id2) Open(id3) Empty

**  The orders corresponding to closing a Dd are: close the isolator on bar B,
**  the isolator on bar A and then the breaker
ax  Order_List(Dd(id1,id2,id3),up,Close) = Close(id3) Open(id2) Close(id1) Empty

**  The orders corresponding to closing on a bar an open Fa are: close the
**  isolator on the corresponding bar, the isolator and then the breaker
ax  Order_List(Fa(id1,id2,id3,id4),Open,Close_On_A) =
    Close(id3) Close(id1) Close(id2) Empty
ax  Order_List(Fa(id1,id2,id3,id4),Open,Close_On_B) =
    Close(id4) Close(id1) Close(id2) Empty

**  The orders corresponding to the operation of bar exchange of an Fa are:
**  close the isolator on the bar that is open and open the one that is closed
ax  Order_List(Fa(id1,id2,id3,id4),Close_On_A,Close_On_B) =
    Close(id4) Open(id3) Empty
ax  Order_List(Fa(id1,id2,id3,id4),Close_On_B,Close_On_A) =
    Close(id3) Open(id4) Empty
```

auxiliary (end)

BAR MANAGER: **Specification**



MANAGERS: **Activity**



COORDINATOR: **Specification**

OPERATION

Unit positions allows to represent the positions of the units as known by the coordinator

TOPOLOGY

UNITS POSITIONS

The coordinator has the task of managing the activity of the functional units (through their managers), depending on the operation required by the operator by means of the console, and on the information on the situation of the station obtained by combining those known from the managers (current)show grestore gsave 33 230 translate 1 -1 scale 0 0 mo

(topology of the station). Moreover it transmits to the console the messages about to the result of the operation. Another task of the coordinator is the control of the situation of the station: if it finds a failure, then it orders to all managers and to the console the end of the activity.
When the coordinator receives from a manager the authorization request for executing an operation on a functional unit, then this is ready for such execution, ie, it is not already in the required position and there are not failures in the station.
The coordinator manages in different way the three kinds of operations: opening, closing and bar exchange.
In each case it checks that the operation is valid, ie, that the bars connected to the functional unit to be used are not failed; to do that it reads in the corresponding managers the situations of the involved bars and if one of them is failed, it informs the console and all managers that there is a failure.
In the case of closing on a bar of a functional unit of kind Fa, the coordinator must determine if the operation is either of bar exchange or of closing; to do that it checks the situation of such Fa (reading it in the corresponding)show grestore gsave 33 446 translate 1 -1 scale 0 0 move

For the bar exchange operation, it must analyze the schema for determining the closing path:
 - if there is already a closed Dd, it allows the operation;
 - if it is needed to close a proper or not proper Dd but electrically connected, i.e., such that the isolators that divides the pieces of bar of the Fa and of the Dd are closed, it orders to the Dd to close and, after that, allows the operation of the Fa;
 - if it is needed to close a not proper Dd and not electrically connected, it does not allow the operation of the Fa and informs the console that it is needed to close an Ae.

COORDINATOR

To halt the station due to some failure

To detect a failure in a unit

To receive an authorization request

To signal the completition of an operation

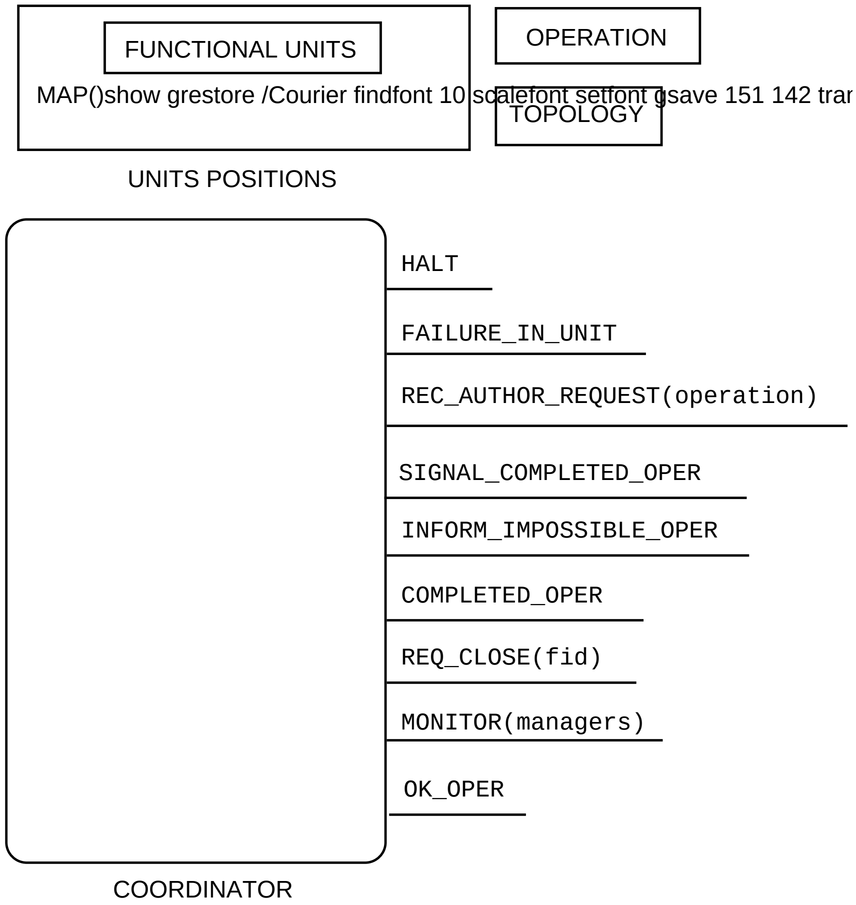
To deny the authorization to the execution of an operation

To be informed that the operation has been completed

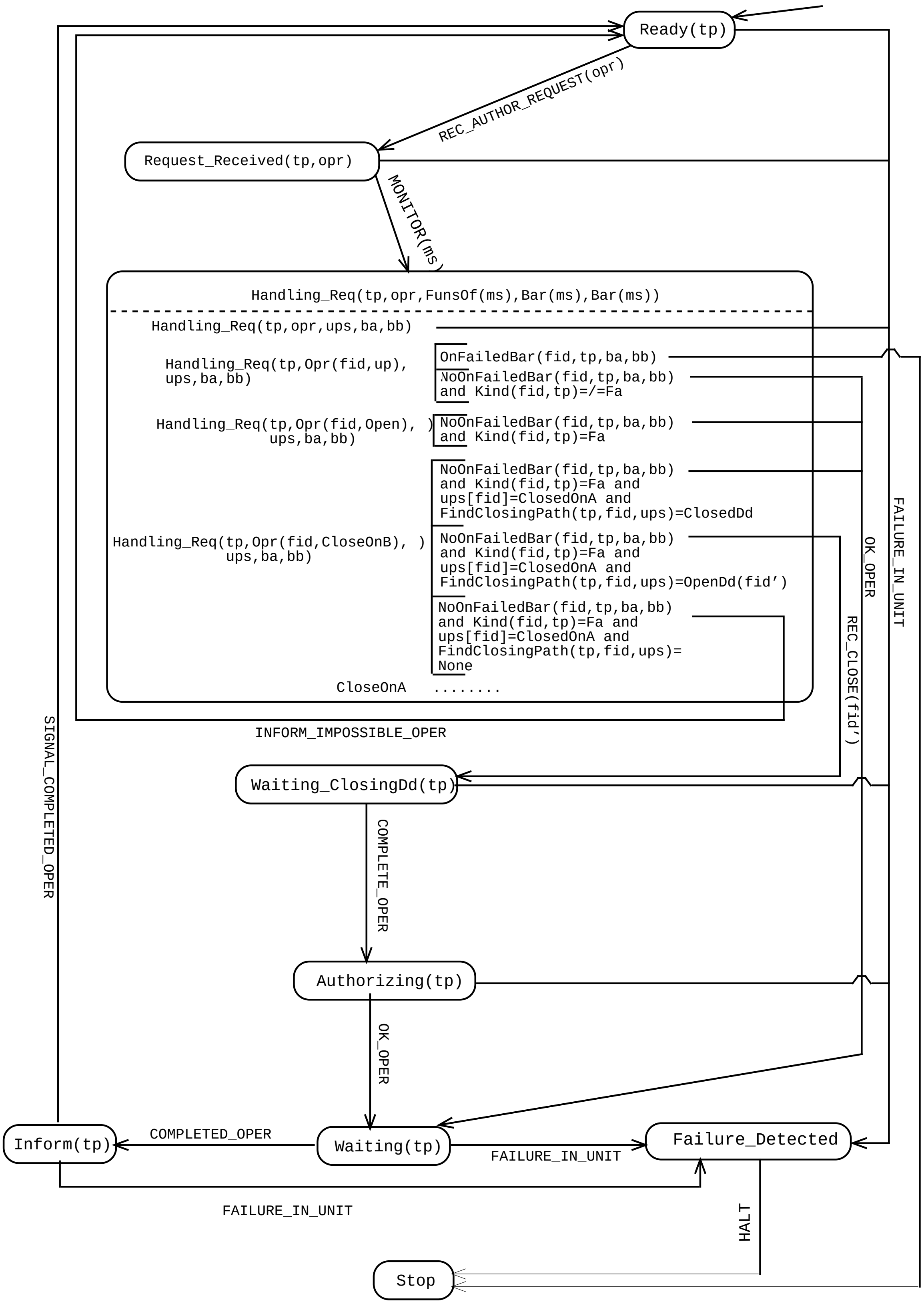
To require to close a Dd

To monitor the units and the bars throughout their managers

To authorize the execution of an operation


FUNCTIONAL UNITS
MAP()show grestore /Courier findfont 10 scalefont setfont gsave 151 142 tran
OPERATION
TOPOLOGY
UNITS POSITIONS
HALT
FAILURE_IN_UNIT
REC_AUTHOR_REQUEST(operation)
SIGNAL_COMPLETED_OPER
INFORM_IMPOSSIBLE_OPER
COMPLETED_OPER
REQ_CLOSE(fid)
MONITOR(managers)
OK_OPER
COORDINATOR

COORDINATOR: **Activity**



```
answer ::=
  None, ClosedDd: answer |  OpenDd: unit_ident
```

```
**  given a set of functional unit managers returns the positions of the
**  associated functional units
op  Positions_From: managers -> units_positions

ax  Positions_From({})=[]
ax  Positions_From(Ready(uid,dsch,up) | ms)= Positions_From(ms)[up / uid]
ax  Positions_From(Executing(uid,dsch,up,up')|ms)=Positions_From(ms)[up/uid]
ax  Positions_From(Waiting_Authorization(uid,dsch,up,up')|ms)=
    Positions_From(ms)[up/uid]
ax  Positions_From(Ordering(fid,dsch,up,up',ol)|ms)=Positions_From(ms)[up/uid]
```

```
**  Find_Closing_Path given a station topology, the functional unit positions and
**  the identifier of an Fa returns an answer saying whether for such unit no
**  closing path exists, a closing path made by an open or by a closed Dd exists
op  Find_Closing_Path: topology unit_ident units_positions -> answer

**  If on the right of the functional unit fa there exists a proper closing path,
**  then there exists a proper closing path for fa
ax  if Path(RPart(tp,fa),ups,None) = ClosedDd then
        Find_Closing_Path(tp,fa,ups) = ClosedDd

**  If on the left of the functional unit fa there exists a proper closing path,
**  then there exists a proper closing path for fa
ax  if Path(LPart(tp,fa),ups,None) = ClosedDd then
        Find_Closing_Path(tp,fa,ups) = ClosedDd

**  If on the right of the functional unit fa there exists a non-proper closing
**  path and on the left of fa there does not exist a proper closing path, then
**  there exists a non-proper closing path for fa
ax  if Path(RPart(tp,fa),ups,None) = Open(uid) and
    Path(LPart(tp,fa),ups,None) =/= ClosedDd then
        Find_Closing_Path(tp,fa,ups) = Open(uid)

**  If on the left of the functional unit fa there exists a non-proper closing
**  path and on the right of fa there does not exist a closing path, then there
**  exists a non-proper closing path for fa
ax  if Path(LPart(tp,fa),ups,None) = Open(uid) and
     Path(RPart(tp,fa),ups,None) = None then
        Find_Closing_Path(tp,fa,ups) = Open(uid)

**  If on the left and on the right of the functional unit fa no closing path
**  exist, then no closing path exists for fa
ax  if Path(LPart(tp,fa),ups,None) = None and
       Path(RPart(tp,fa),ups,None) = None then
        Find_Closing_Path(tp,fa,ups) = None
```

auxiliary (continues)

```
**  Path given the topology of one part of the station, the functional unit
**  positions and the identifier of an Fa returns an answer saying whether in
**  such part of the station for such unit no closing path exists, a closing
**  path made by an open or by a closed Dd exists
op  Path: topology units_positions answer -> answer

**  If the station has been scanned until the end Path returns the recorded
**  answer
ax  Path(E,ups,a) = a

**  If uid is a closed Dd then Path returns there is a closing path made by
**  a closed Dd
ax  if ups[uid] = Close then  Path(Dd(uid) & tp,ups,a) = ClosedDd

**  If uid is an open Dd and there is already recorded an open Dd, then the
**  scanning of the bars goes on
ax  if ups[uid] = Open then
        Path(Dd(uid) & tp,ups,OpenDd(uid')) = Path(tp,ups,OpenDd(uid'))

**  If uid is an open Dd and nothing is recorded, then the scanning of the
**  bars goes on recording it
ax  if ups[uid] = Open then
        Path(Dd(uid) & tp,ups,None) = Path(tp,ups,OpenDd(uid))

**  If uid1 and uid2 are two closed Ae, then the scanning of the bars goes on
ax  if ups[uid] = Close and ups[uid] = Close then
        Path(Ae(uid1,uid2) & tp,ups,a) = Path(tp,ups,a)

**  If either uid1 or uid2 is open, then the recorded answer is returned
ax  if (ups[uid] = Open or ups[uid] = Open) then
        Path(Ae(uid1,uid2) & tp,ups,a) = a

**  If uid is an Fa, then the scanning of the bars goes on
ax  Path(Fa(uid) & tp,ups,a) = Path(tp,ups,a)
```

```
**  checks whether a functional unit is/is not connected to a failed bar
op  No_On_Failed_Bar, On_Failed_Bar: unit_ident topology bar bar

ax  if Non_On_BarA(uid,tp) then No_On_Failed_Bar(uid,tp,KO,bb)
ax  if Non_On_BarB(uid,tp) then No_On_Failed_Bar(uid,tp,ba,KO)

op  On_Failed_Bar: unit_ident topology bar bar
ax  if On_BarA(uid,tp) then On_Failed_Bar(uid,tp,KO,bb)
ax  if On_BarB(uid,tp) then On_Failed_Bar(uid,tp,ba,KO)
```

```
op  Bar: bar_manager -> bar
ax  Bar(OK) = OK
ax  Bar(KO) = KO
```

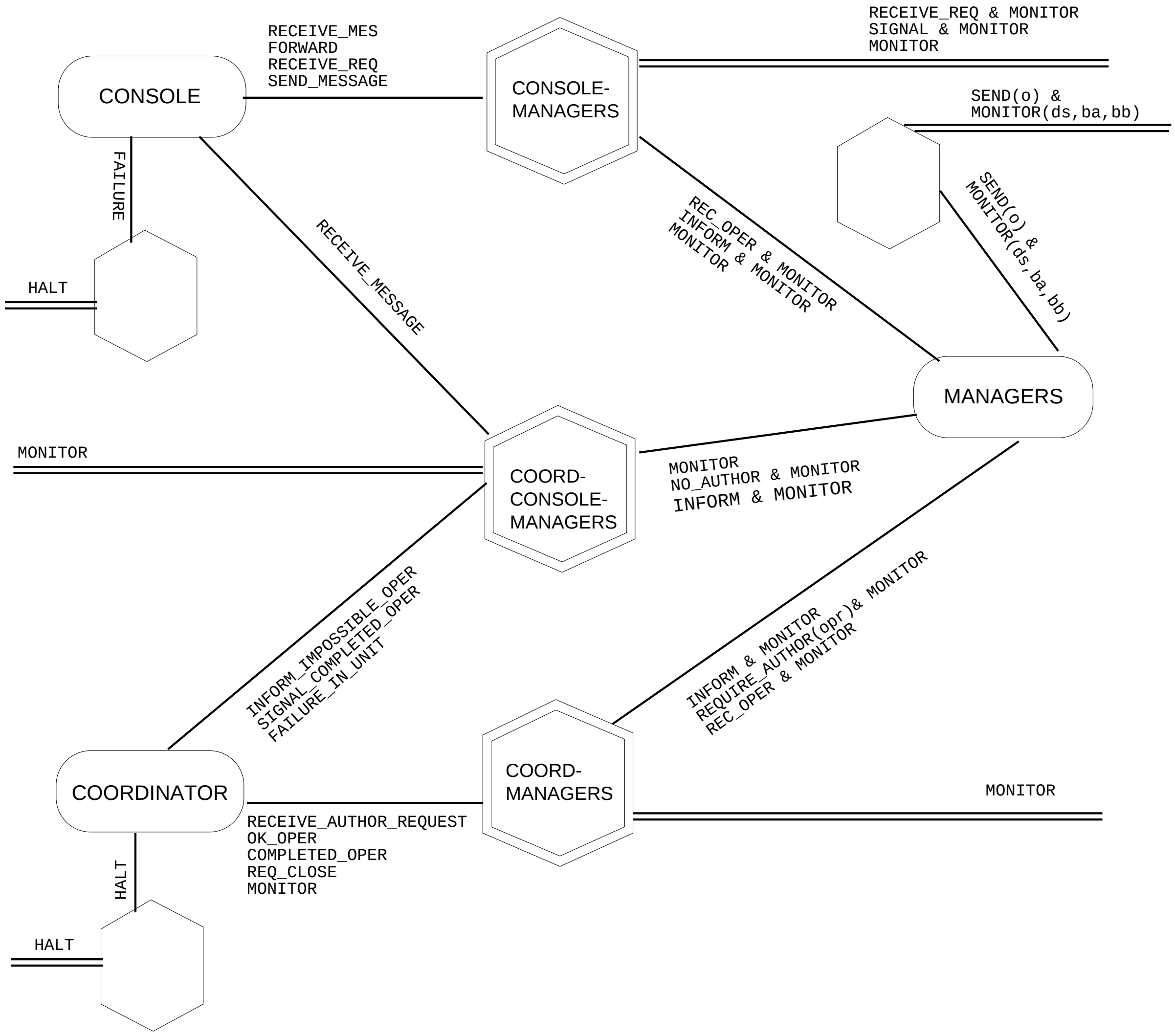
```
opr: operation              a: answer              ups: unit_positions
uid, uid', fa: unit_ident   ms: set(manager)       ba,bb: bar
tp: topology                dsch: devices_schema   ol: orders
up up': unit_position
```

auxiliary (end)

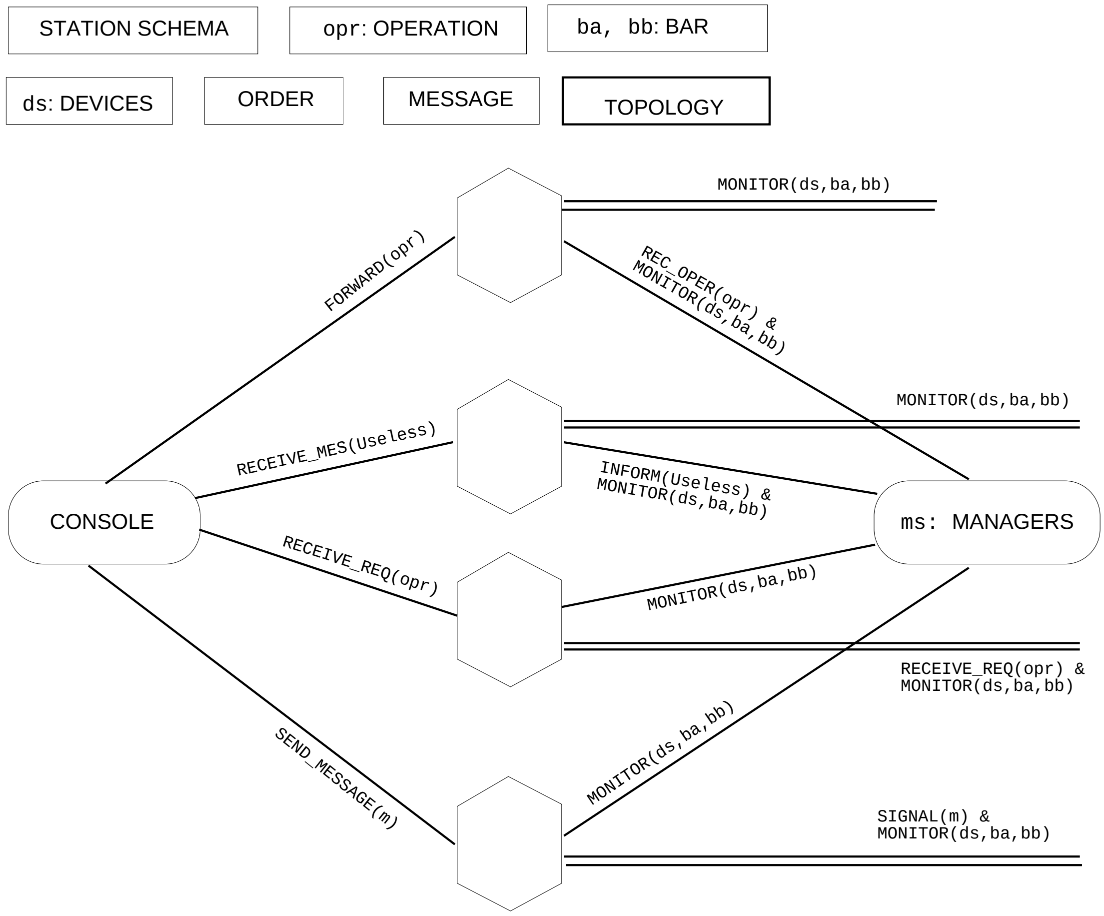
AUTOMATISM: **Activity**

STATION SCHEMA | OPERATION | BAR

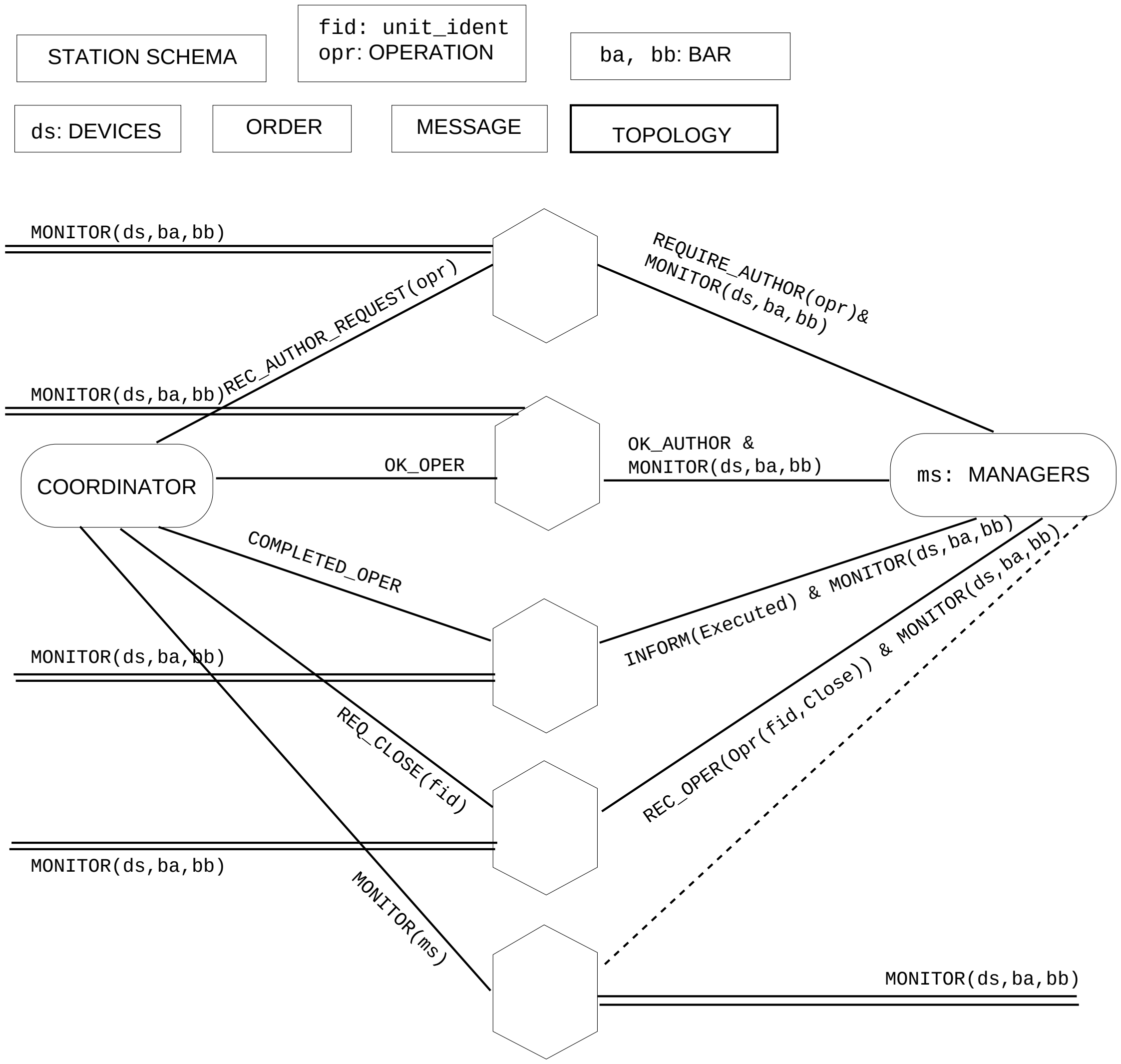
DEVICES | ORDER | MESSAGE | TOPOLOGY



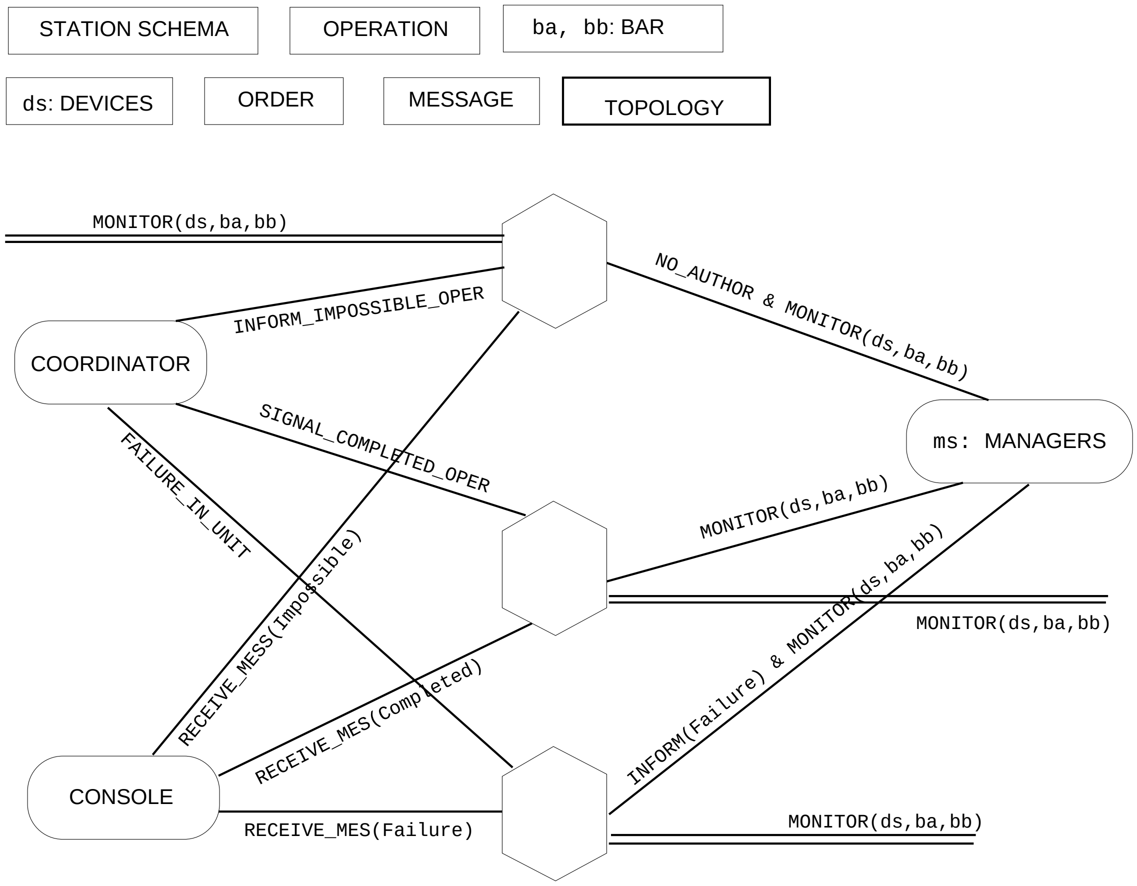
CONSOLE-MANAGERS



COORD-MANAGERS



COORD-CONSOLE-MANAGERS



References

[LR97] M. Larosa and G. Reggio. A Graphic Notation for Formal Specifications of Dynamic Systems. Technical Report DISI–TR–97–3, DISI – Università di Genova, Italy, 1997. Full Version.

[Reg98] G. Reggio. A Guide to the Use of the SMoLCS Methodology. Technical Report DISI–TR–98–3, DISI – Università di Genova, Italy, 1998.