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], precisely the high-voltage stations for the distribution of the electric power used by ENEL, the Italian National Company of Electricity.

The development of such case study has been organized in the following phases.

Capture and specification of the requirements Initially we have determined and specified the fundamental requirements on the high-voltage stations considered by ENEL.

First development step Here we have refined such stations by considering only those managed by a software automatism: the components of the station which constitute the plant are completely defined. We give a functional description of the automatism, without structuring it, so that we leave as much freedom as possible to its implementation. Thus, at this step, we give the design specification of the devices and of the bars (the plants components), while we still have the requirements on the automatism.

Second development step Here we specify the 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 how the concurrent activity of the components of the automatism works, 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 the informal specification is reported 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 which 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 (that is to say a pair of wires) and by a set of devices, breakers and isolators connected to bars and/or 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 in order 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*.

The various functional unit kinds and their positions are presented in Fig. 1.

An Fa is a functional unit which 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.

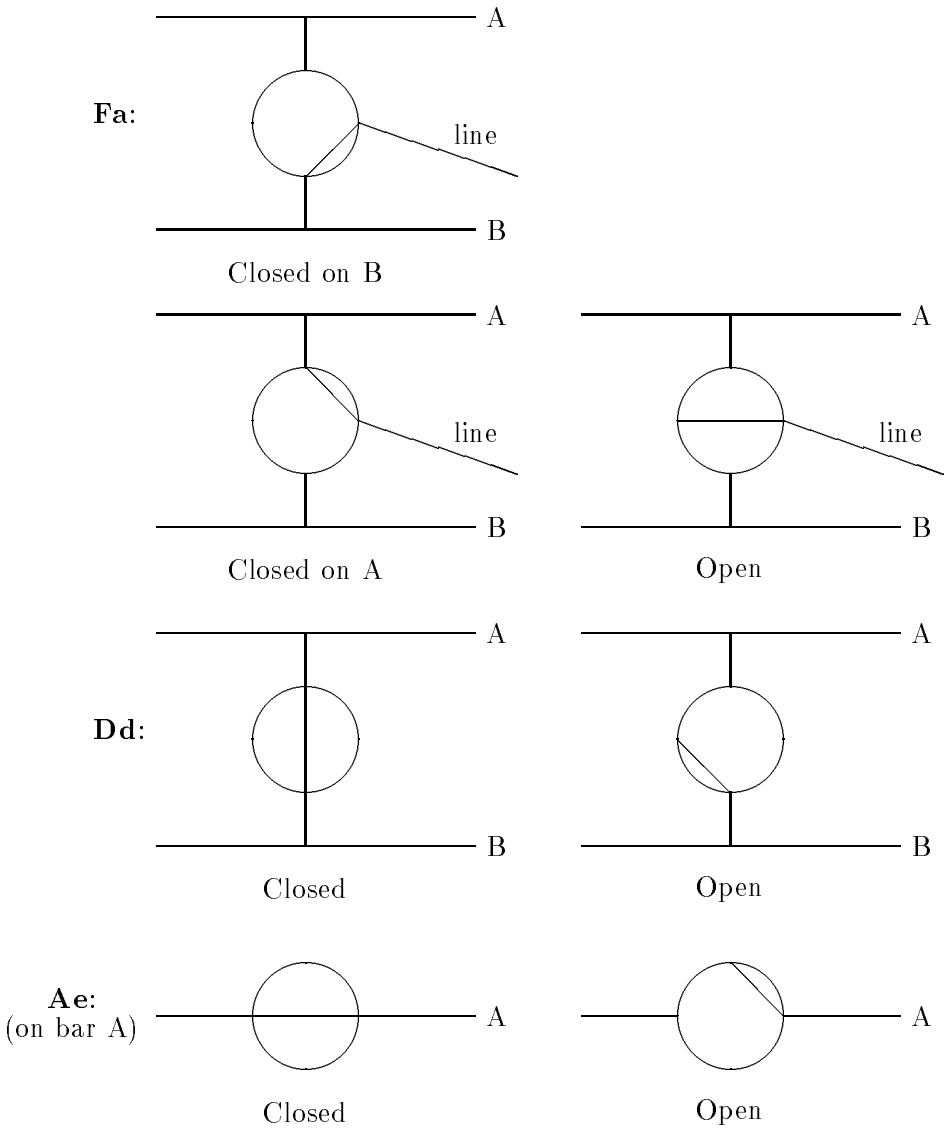


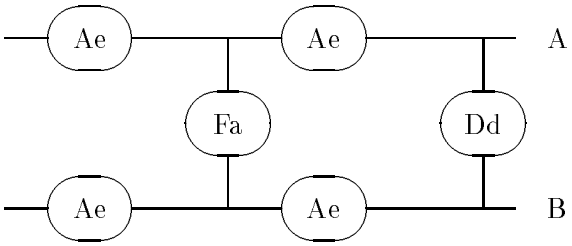
Figure 1: The kinds of functional units and their positions

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

An example of an admissible station is graphically represented below.



The operator can require operations on the functional units of a station 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 which 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 the various operations are performed on the functional units.

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

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 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.

The closing path has to be identified with 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;

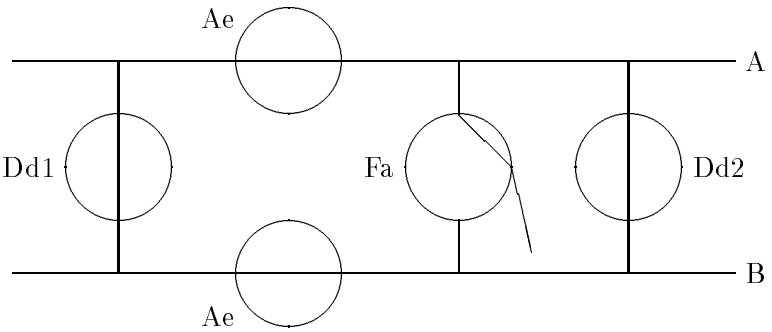


Figure 2: Either Dd1 or Dd2 may be chosen for building the closing path for Fa.

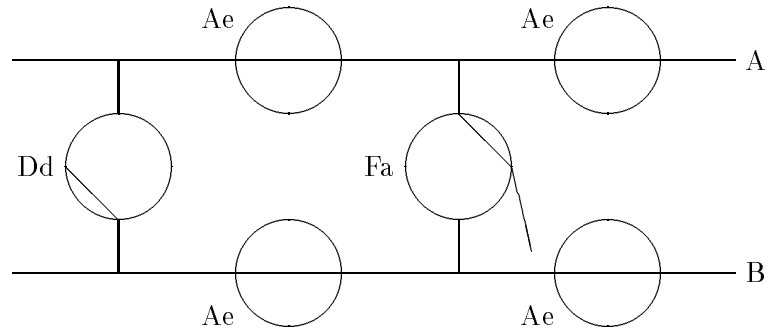


Figure 3: Dd must be closed for building the closing path for Fa.

- 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).

Examples of the above three cases are shown in Fig. 2, 3 and 4 respectively.

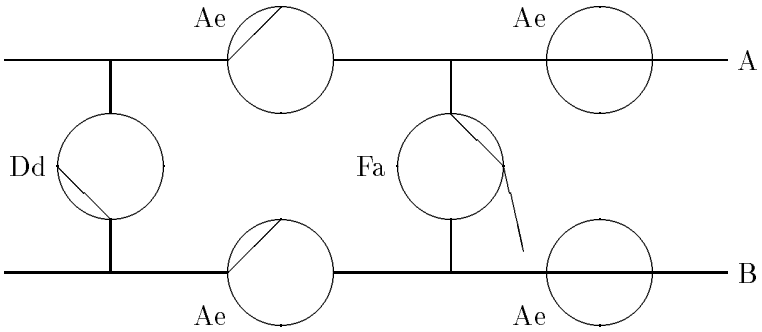


Figure 4: No closing path for Fa is possible, some Ae must be closed.

STATION.PHASE 1: **Border Determination**

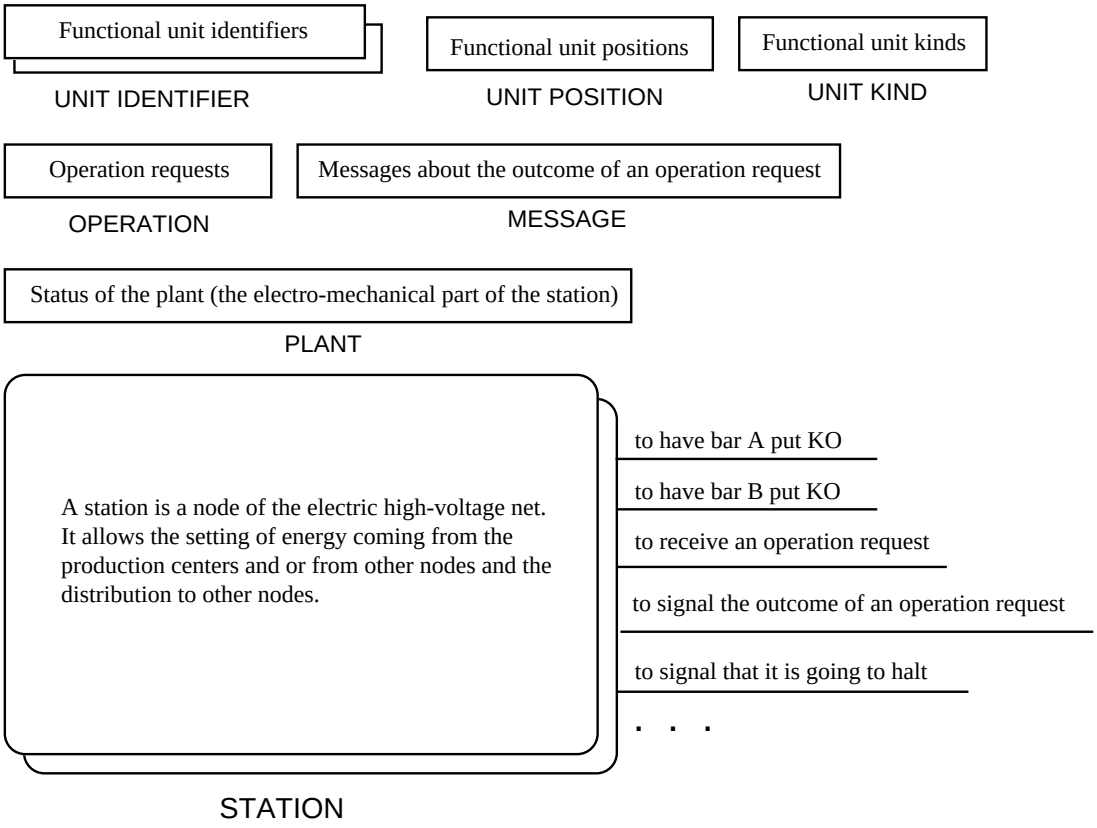
A station is an open system, i.e., it may interact with the external environment consisting of an operator and of the other nodes of the high-voltage electric net. There are no hypothesis on the behaviour of the nodes and of the operator, so they are considered outside the specified system.

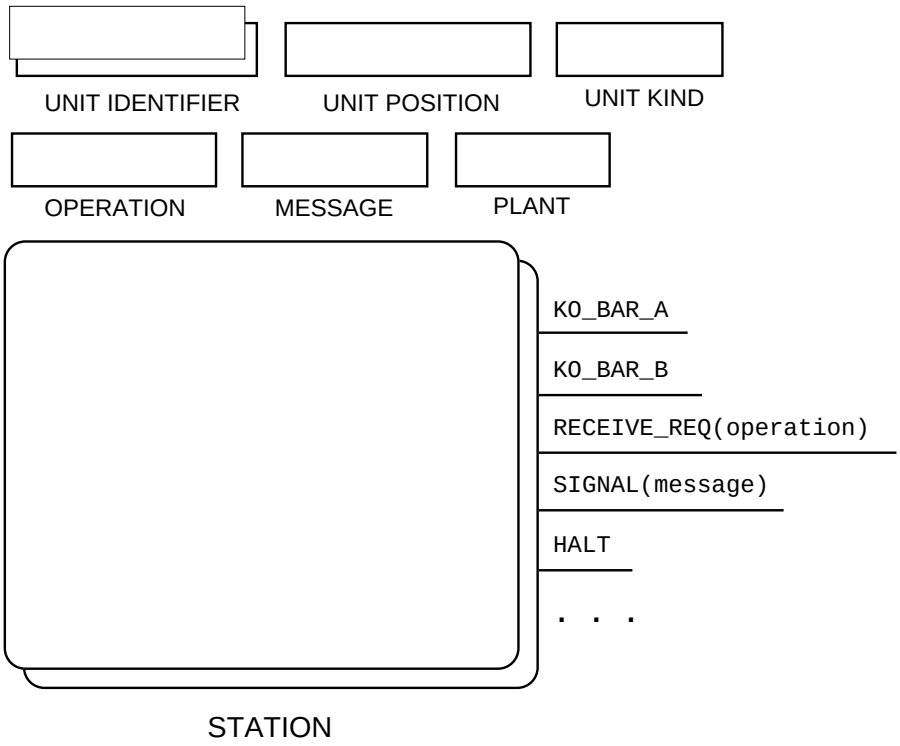
STATION.PHASE 1: **Shadow Spots**

Nothing is said about to receive wrong operation requests, e.g., to perform an operation on a non-existig unit. We have assumed that only correct operation requests will arrive.

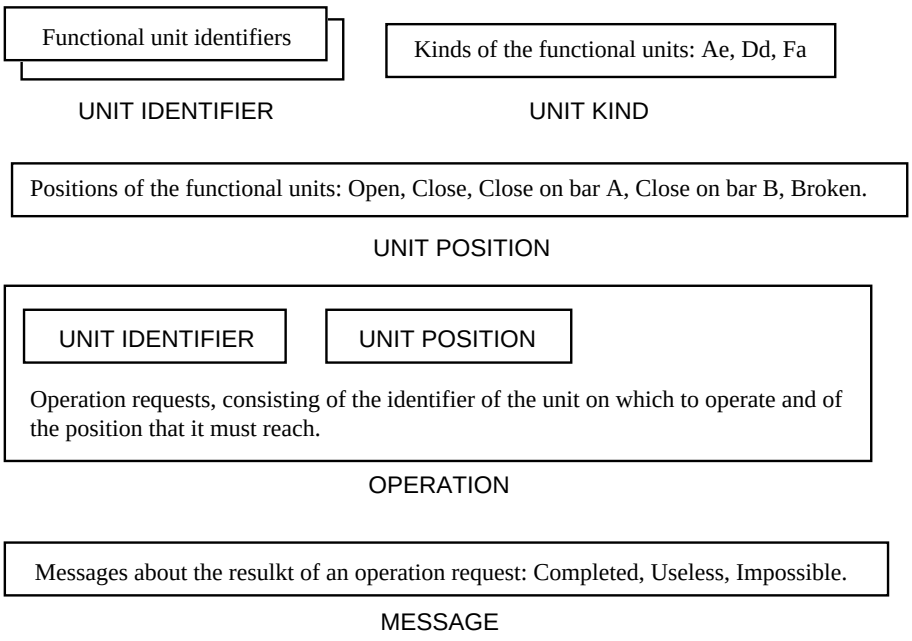
STATION.PHASE 1: **Specification**

Structure & Interactions





STATION.PHASE 1: Basic Data Structures



UNIT IDENTIFIER

```
enum:  Ae, Dd, Fa
```

UNIT KIND

```
enum: Open, Close, Close_On_A, Close_On_B, Broken
```

UNIT POSITION

UNIT IDENTIFIER | UNIT POSITION

```
op  Opr: unit_ident unit_position -> operation
```

OPERATION

```
enum: Completed, Useless, Impossible
```

MESSAGE

PLANT

UNIT KIND | UNIT IDENTIFIER | UNIT POSITION

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

FUNCTIONAL_UNIT

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

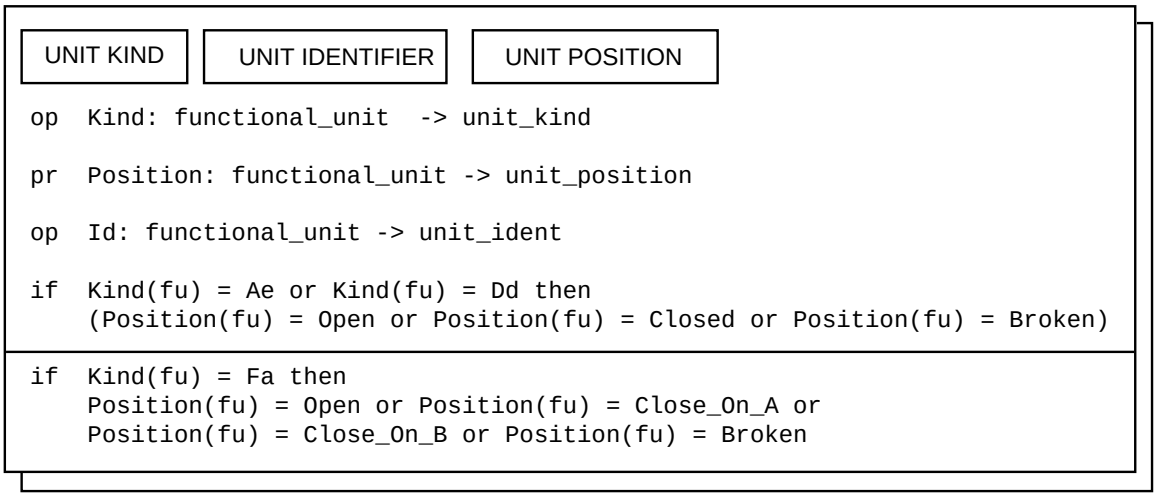
BAR

BAR | FUNCTIONAL_UNIT

The status of a plant is given by the status of the two bars and of the functional units connected to each bar; Fa and Dd 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.

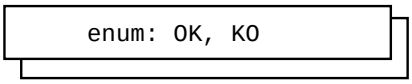
In the plants of the adimissible stations the units are connected to the bars as hinted above and there is at least one Dd and one Fa.

PLANT




```
fu:  functional_unit
```

FUNCTIONAL_UNIT



BAR

BAR | FUNCTIONAL_UNIT

```
**  The status of the plant may consists just of the status of the two bars
op  < _ _ >: bar bar  -> plant
**  adds two functional units to the status of a plant, one for each bar
op  < _ _ > & _: functional_unit functional_unit plant  -> plant
**  adds a functional unit to the status of a plant, such functional unit is
**  connected to both bars
op  < _ > & _: functional_unit plant  -> 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)
```

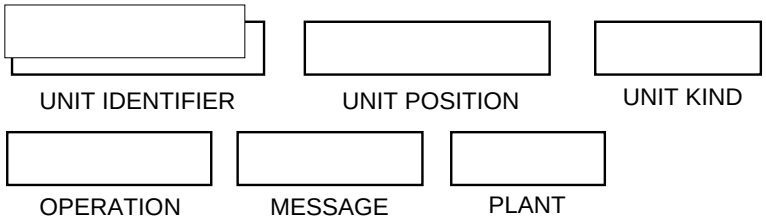
```
**  checks whether the connections of the functional units on the bars are
**  admissible, i.e. if each one of kind either Fa or Dd is connected to both
**  the bars and those of kind Ae are present in pairs
pr  Ok_Con: plant

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




```
**  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 if a station state is initial
pr  Initial: station
```

```
**  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 fid: Position(st,fid) = 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 fid, st', up: st' -- RECEIVE_REQ(Opr(fid,up)) --> st  and
    On_Failed_Bar(fid,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 Id(fu)=fid and Bar_A(pl)=KO then On_Failed_Bar(fid,pl)
if  On_Bar_B(fu,pl) and Id(fu)=fid and Bar_B(pl)=KO then On_Failed_Bar(fid,pl)
```

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

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

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

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

```
**  checks wether a functional unit is a closing path in a station for a given Fa
pr  Is_Closing_Path: unit_ident station fun_unit_ident

**  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
**  such case there does not exist a closed Dd on the same bar section of idfa
if  Is_Closing_Path(fid,st,idfa) then
    - Kind(st,fid) = Dd
    - SameSection(Plant(st),idfa,fid)
    - Position(st,fid) = Closed  or
      (Position(st,fid) = Open and not exists fid': Kind(st,fid') = Dd and
       Position(st,fid') = Closed and SameSection(Plant(st),idfa,fid'))

**  If there exists a functional unit which is a Dd and on the same bar section
**  of idfa, then Is_Closing_Path is true for some functional unit
if  exists fid: Kind(st,fid) = Dd and SameSection(Plant(st),idfa,fid) then
    exists fid': Is_Closing_Path(fid',st,idfa)
```

```
**  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 Id(fu)=fid and On_Bar_A(fu',pl) and Id(fu')=idf'
then  SameSection(pl,fid1,fid')
if  On_Bar_B(fu,pl) and Id(fu)=fid and On_Bar_A(fu',pl) and Id(fu')=idf'
then  SameSection(pl,fid,fid')
```

```
fid, fid', idfa: unit_ident     fu, fu': fun
k: kind                         st: station        up: unit_position
```

(auxiliary)

Kind

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

Plant

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

Failure

```
**  If there is a failure in the station, then in any case it signals that
**  is going to halt and after stops
if  Failure(st) then
    st in any case <HALT> and after [ not exists x', l: x -- l --> x' ]
```

Executing

```
**  If the station becomes executing, then it has received an operation
**  request
if  st -- ls --> st' and not Executing(st) and  Executing(st')  then
    exists opr: ls = RECEIVE_REQ(opr)

**  If the station becomes nonexecuting, then it signals the end of an
**  operation
if  st -- ls --> st' and Executing(st) and not Executing(st')  then
    exists m: ls = SIGNAL(m)

**  If the station is executing an operation, then it cannot accept another
**  request
if  Executing(st) then
    not st at least in a case <exists opr: RECEIVE_REQ(opr)>
```

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
```

KO_BAR_A

```
**  A bar may be always put KO
st at least in a case <KO_BAR_A>

if   st -- KO_BAR_A --> st' then Bar_A(Plant(st')) = OK
```

KO_BAR_B

```
**  A bar may be always put KO
st at least in a case <KO_BAR_B>

if   st -- KO_BAR_B --> st' then Bar_B(Plant(st')) = OK
```

RECEIVE_REQ

```
**  If the station receives an operation request, then it is not executing
**  and in any case it is executing until the operation will end
if st -- RECEIVE_REQ(opr) --> st' then
   - not Executing(st)
   - st' in any case [Executing(x)] until <exists m: ls=SIGNAL(m)>

**  If the station receives the request of putting a unit in the actual
**  position, then in any case eventually it either will signal that the
**  required operation is useless or will fail
if  st -- RECEIVE_REQ(Opr(fid,up)) --> st' and Position(st,fid)=up then
    st' in any case
      eventually <SIGNAL(Useless)>
      or
      eventually [Failure(x)]

**  If the station receives the request of opening a non-open unit fid, then
**  in any case eventually either fid will become open and after it will
**  signal that the required operation has been completed or it will fail
if  st -- RECEIVE_REQ(Opr(fid,Open)) --> st' and Position(st,fid)=/=Open then
    st' in any case
      eventually [Position(x,fid)=Open] and after <SIGNAL(Completed)>
      or
      eventually [Failure(x)]

**  If the station receives the request of closing an open Ae or Dd fid, then
**  in any case eventually either fid will become closed and after it will
**  signal that the required operation has been completed or it will fail
if  st -- RECEIVE_REQ(Opr(fid,Close)) --> st' and
    (Kind(st,fid)=Ae  or Kind(st,fid)=Dd) and Position(st,fid)=Open then
    st' in any case
       eventually [ Position(x,fid) = Closed ] and after <SIGNAL(Completed)>
       or
       eventually [Failure(x)]

**  If the station receives the request of closing on bar A an open Fa fid,
**  then in any case eventually either fid will be closed on A and after it
**  will signal that the required operation has been completed orit will fail
if  st -- RECEIVE_REQ(Opr(fid,Close_On_A)) --> st' and Kind(st,fid)=Fa  and
       Position(st,fid)=Open then
    st' in any case
       eventually [Position(x,fid) = Close_On_A] and after <SIGNAL(Completed)>
       or
       eventually [Failure(x)]
```

RECEIVE_REQ

```
**  If the station receives the request of closing on the bar A an Fa that is
**  closed on the bar B and there is not a Dd to be used for the closing path,
**  then in any case eventually either it will signal that the required
**  operation is impossible or it will fail
if  st -- RECEIVE_REQ(Opr(fid,Close_On_A)) --> st' and Kind(st,fid)=Fa and
    Position(st,fid)=Close_On_B and
    not exists fid': Is_Closing_Path(fid',st,fid) then
    st' in any case
       eventually <SIGNAL(Impossible)>
       or
       eventually [Failure(x)]

**  If the station receives the request of closing on the bar A an Fa fid that
**  is closed on the bar B and there is a Dd fid' to be used for the closing
**  path, then in any case eventually either fid' will be closed and after
**  eventually fid will be closed on the bar A and after it signal that the
**  operation has been completed or it will fail
if  st -- RECEIVE_REQ(Opr(fid,Close_On_A)) --> st' and Kind(st,fid)=Fa and
    Position(st,fid)=Close_On_B and Is_Closing_Path(fid',st,fid) then
    st' in any case
       eventually [Position(x,fid') = Closed] and after
            eventually [Position(x,fid) = Close_On_A] and after
                 <SIGNAL(Completed)>
       or
       eventually [Failure(x)]

**  Similar properties for the closing of an Fa on the bar B.
```

SIGNAL

```
**  If the station signals the end of an operation, then it was executing and
**  becomes nonexecuting
if  st -- SIGNAL(m) --> st' then  Executing(st) and not Executing(st')
```

HALT

```
**  If the station halts, then there was a failure
if  st -- HALT --> st' then Failure(st')
```

```
fu fu1 fu2: functional_unit          m: message           ls: lab_station
pl: plant  opr: operation            st st' x x': station
fid, fid': unit_ident                up: unit_position
```

STATION.PHASE 2.step 1

At this level 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**

Sets of stations and power plants are managed by only one operator. Thus the absence of local operators in the stations, due to the installation of an automatic system, rises 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 informative 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.

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 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 which 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 or 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 undertension 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;

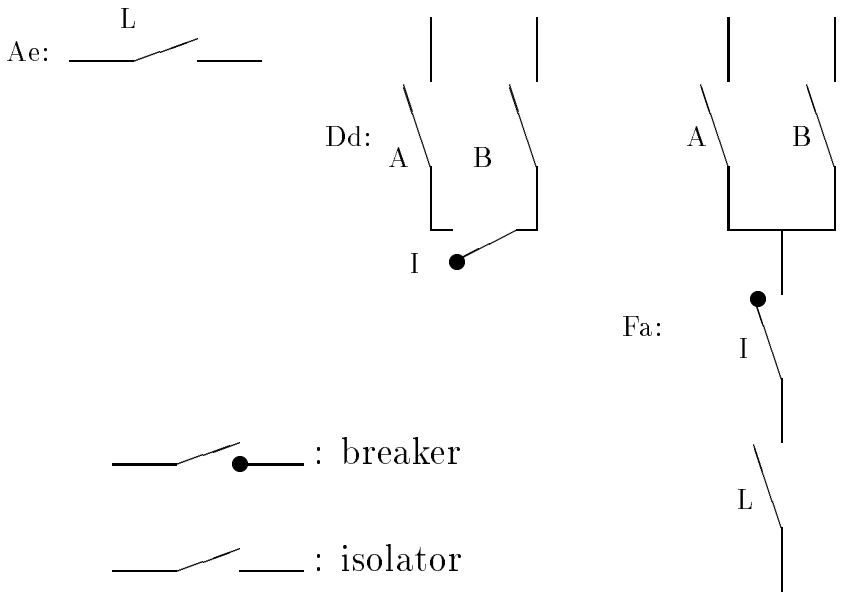


Figure 5: Schemas of the functional units

- 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 which 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 electrically isolate 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.

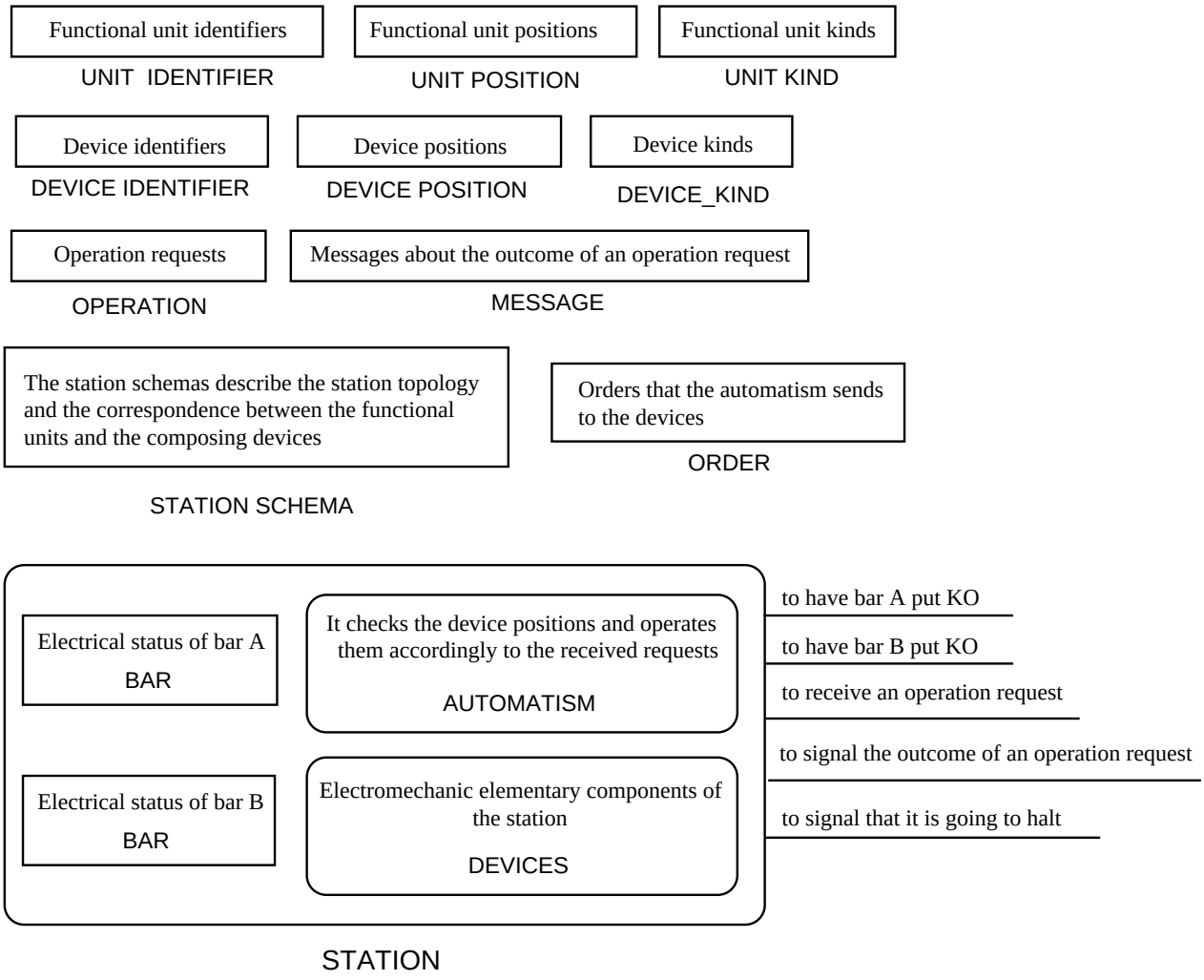
Fa operations The functional unit Fa is composed by a breaker S and three isolators L, A, B; it can be closed on bar A, or on bar B, or open.

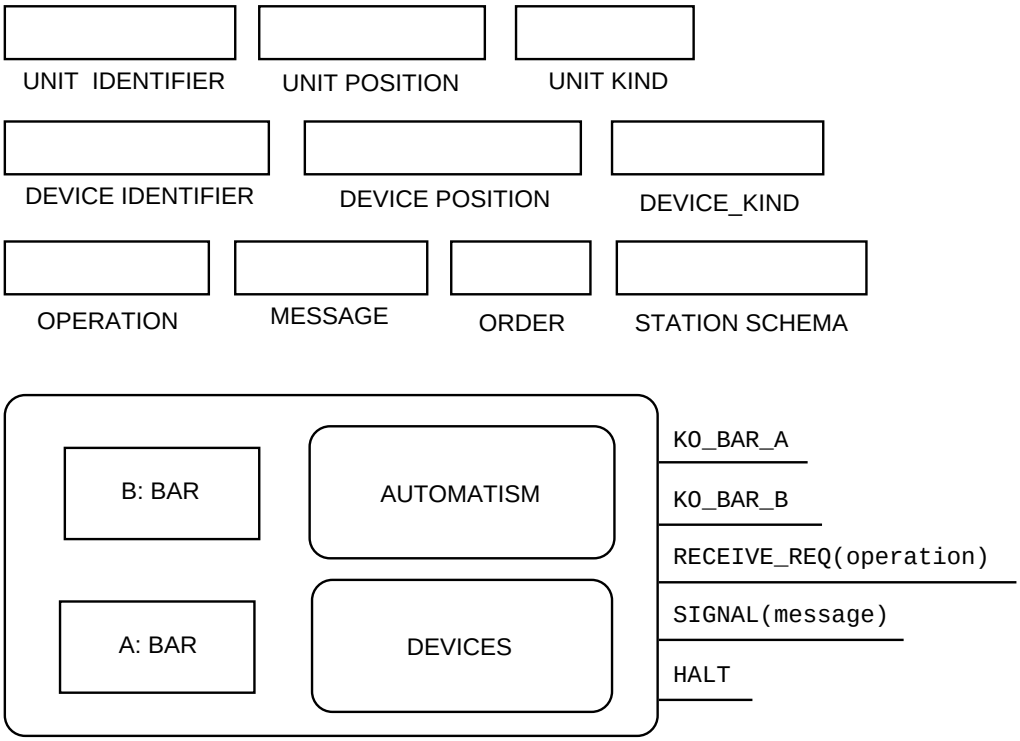
The opening operation consists of opening S, then A and then B. The closing operation on bar A of an Fa open consists of closing A, then closing L and then closing S, analogously for closing on bar B.

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.

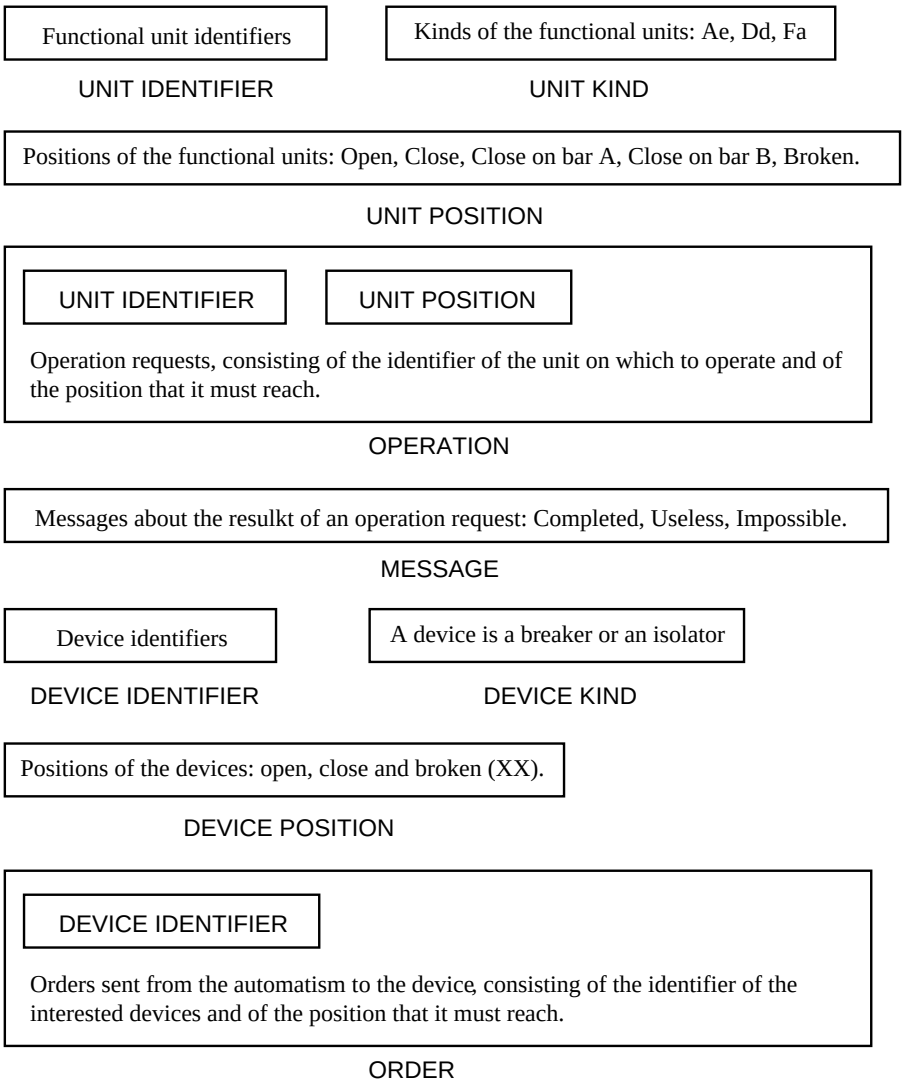
The closing path has to be identified with the rules already presented in the natural description of PHASE 1.

Structure & Interactions




UNIT IDENTIFIER
UNIT POSITION
UNIT KIND
DEVICE IDENTIFIER
DEVICE POSITION
DEVICE_KIND
OPERATION
MESSAGE
ORDER
STATION SCHEMA
B: BAR
AUTOMATISM
A: BAR
DEVICES
KO_BAR_A
KO_BAR_B
RECEIVE_REQ(operation)
SIGNAL(message)
HALT

STATION


Functional unit identifiers
UNIT IDENTIFIER
Kinds of the functional units: Ae, Dd, Fa
UNIT KIND
Positions of the functional units: Open, Close, Close on bar A, Close on bar B, Broken.
UNIT POSITION
UNIT IDENTIFIER
UNIT POSITION
Operation requests, consisting of the identifier of the unit on which to operate and of the position that it must reach.
OPERATION
Messages about the resulkt of an operation request: Completed, Useless, Impossible.
MESSAGE
Device identifiers
DEVICE IDENTIFIER
A device is a breaker or an isolator
DEVICE KIND
Positions of the devices: open, close and broken (XX).
DEVICE POSITION
DEVICE IDENTIFIER
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

```
rename sort nat to unit_ident in NAT
```

UNIT IDENTIFIER

```
enum: Ae, Dd, Fa
```

UNIT KIND

```
enum: Open, Close, Close_On_A, Close_On_B, Broken
```

UNIT POSITION

UNIT IDENTIFIER | UNIT POSITION

```
op  Opr: unit_ident unit_position -> operation
```

OPERATION

```
rename sort nat to device_ident in NAT
```

DEVICE IDENTIFIER

```
enum: Br, Is
```

DEVICE KIND

```
enum: OP, CL, XX
```

DEVICE POSITION

```
enum: Completed, Useless, Impossible
```

MESSAGE

DEVICE IDENTIFIER

```
op  Open, Close: device_ident -> order
```

ORDER

STATION SCHEMA

DEVICE_IDENT

The schemas of the set of 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.

DEVICES SCHEMA

DEVICES SCHEMA | UNIT IDENTIFIER

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

UNIT SCHEMA

UNIT SCHEMA | UNIT KIND

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

STATION SCHEMA

DEVICE_IDENT

```
**  given the identifier of the isolator returns the schema of an Ae
op  Ae: device_ident -> devices_schema

**  given the identifiers of the isolator on bar A, of that on bar B and of the
**  breaker returns the schema of a Dd
op  Dd: device_ident device_ident device_ident -> devices_schema

**  given the identifiers of the line isolator, of the breaker and of the
**  isolator on bar A and of that on bar B returns the schema of an Fa
op  Fa: device_ident device_ident device_ident device_ident -> devices_schema
```

DEVICES SCHEMA

DEVICES SCHEMA | UNIT IDENTIFIER

```
op  Fu: unit_ident devices_schema -> unit_schema
```

UNIT SCHEMA

UNIT SCHEMA | UNIT KIND

```
cn  E: station_schema    **  schema of the empty station
**  adds a functional unit to a schema (an Fa or a Dd)
op  < _ > & _: unit_schema station_schema -> station_schema
**  adds two functional units to a schema (two Ae's)
op  < _ _ > & _: unit_schema unit_schema station_schema -> station_schema
```

```
**  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(fid,<Fu(fid,dsch)> & sch)
ax  if On_BarA(fid,sch) then On_BarA(fid,<Fu(fid1,dsch)> & sch)
ax  On_BarA(fid1,<Fu(fid1,dsch1) Fu(fid2,dsch2)> & sch)
ax  if On_BarA(fid,sch) then On_BarA(fid,<Fu(fid1,dsch1) Fu(fid2,dsch2)> & sch)

ax  On_BarB(fid,<Fu(fid,dsch)> & sch)
ax  if On_BarB(fid,sch) then On_BarB(fid,<Fu(fid1,dsch)> & sch)
ax  On_BarB(fid1,<Fu(fid1,dsch1) Fu(fid2,dsch2)> & sch)
ax  if On_BarB(fid,sch) then On_BarB(fid,<Fu(fid1,dsch1) Fu(fid2,dsch2)> & sch)
ax  if On_BarB(fid,sch) then On_BarB(fid,<Fu(fid1,dsch1) Fu(fid2,dsch2)> & sch)
```

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

ax  Kind(<Fu(fid,Ae(id1)) Fu(fid2,Ae(id2))> & sch,fid)=Ae
ax  Kind(<Fu(fid1,Ae(id1)) Fu(fid,Ae(id2))> & sch,fid)=Ae
ax  if fid=/=fid1 and fid=/=fid2 then
    Kind(<Fu(fid1,Ae(id1)) Fu(fid2,Ae(id2))> & sch,fid)=Kind(sch,fid)
ax  Kind(<Fu(fid,Dd(id1,id2,id3))> & sch,fid)=Dd
ax  Kind(<Fu(fid,Fa(id1,id2,id3,id4))> & sch,fid)=Fa
ax  if fid=/=fid' then  Kind(<Fu(fid',dsch)> & sch,fid)=Kind(sch,fid)
```

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

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

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

ax  Isolator(<Fu(fid,Ae(id1)) Fu(fid2,Ae(id2))> & sch,fid)=id1
ax  Isolator(<Fu(fid1,Ae(id1)) Fu(fid,Ae(id2))> & sch,fid)=id2
ax  if fid=/=fid1 and fid=/=fid2 then
    Isolator(<Fu(fid1,Ae(id1)) Fu(fid2,Ae(id2))> & sch,fid)=Isolator(sch,fid)
ax  Isolator(<us1> & sch,fid)=Isolator(sch,fid)
```

```
sch: station_schema                         fid fid1 fid2: unit_ident
dsch dsch1 dsch2: devices_schema            us1 us2: unit_schema
id1,id2,id3,id4: device_ident
```

STATION SCHEMA (end)

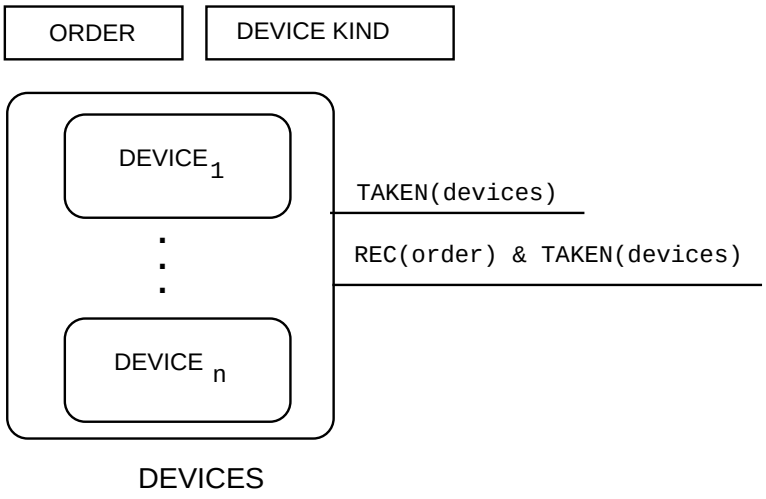
BAR: Specification

```
**  Electrical status of the bar:  in tension and not in tension
enum: OK, KO
```

BAR

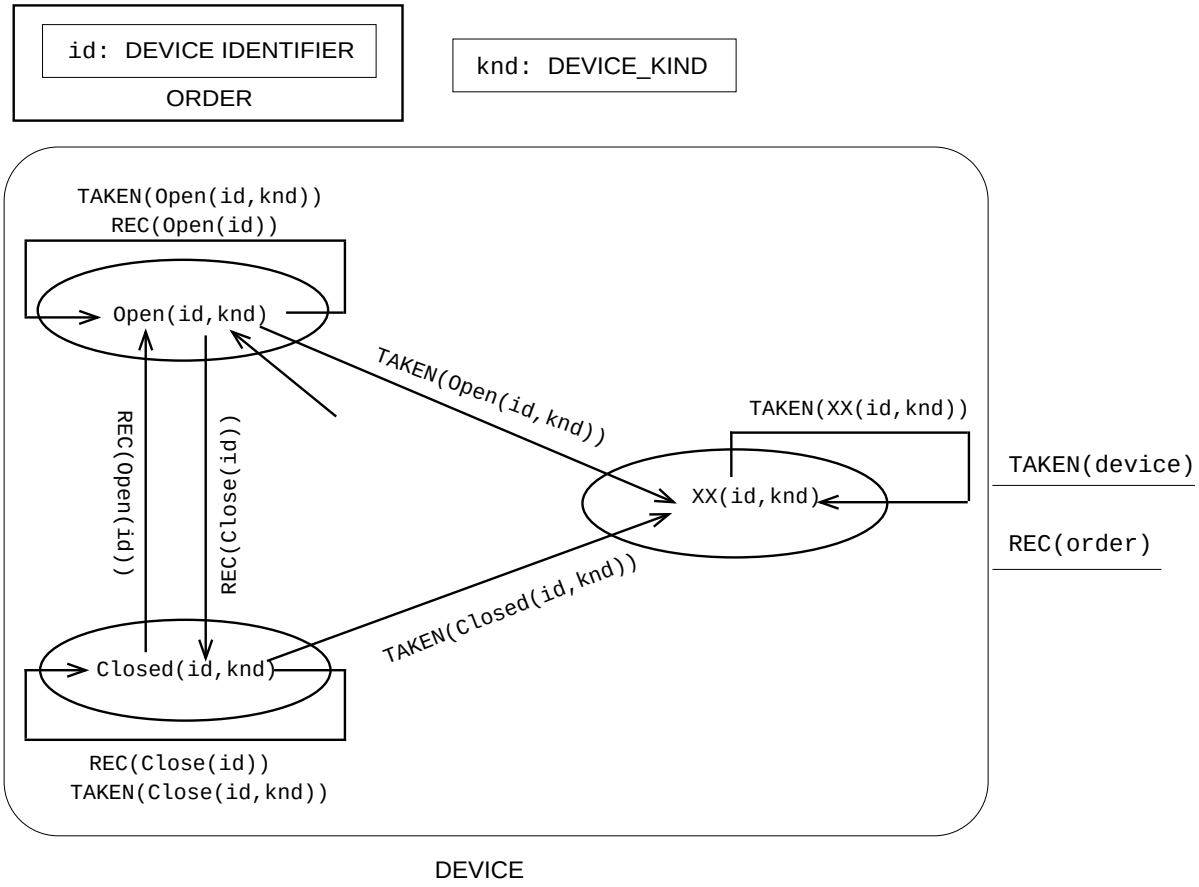
DEVICES: Specification

Structure & Interactions



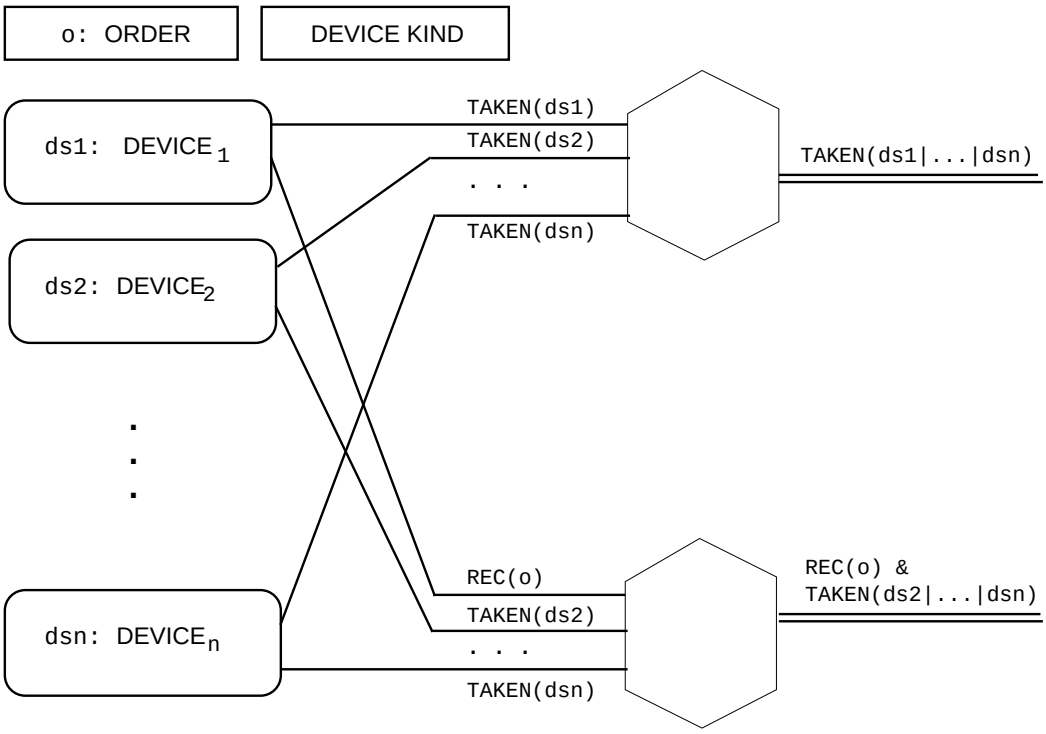
DEVICES

DEVICE: **Specification**

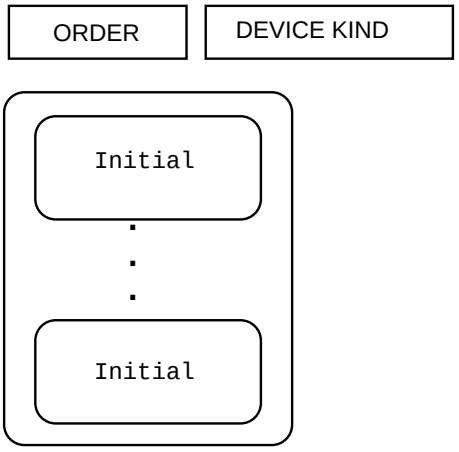


DEVICE

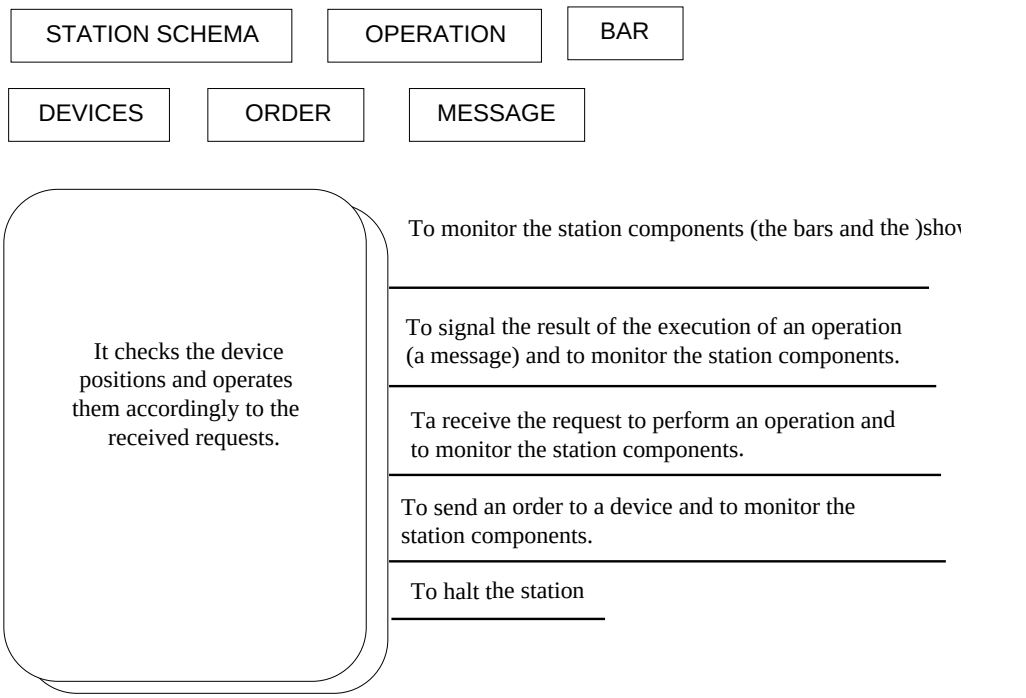
DEVICES: **Activity**

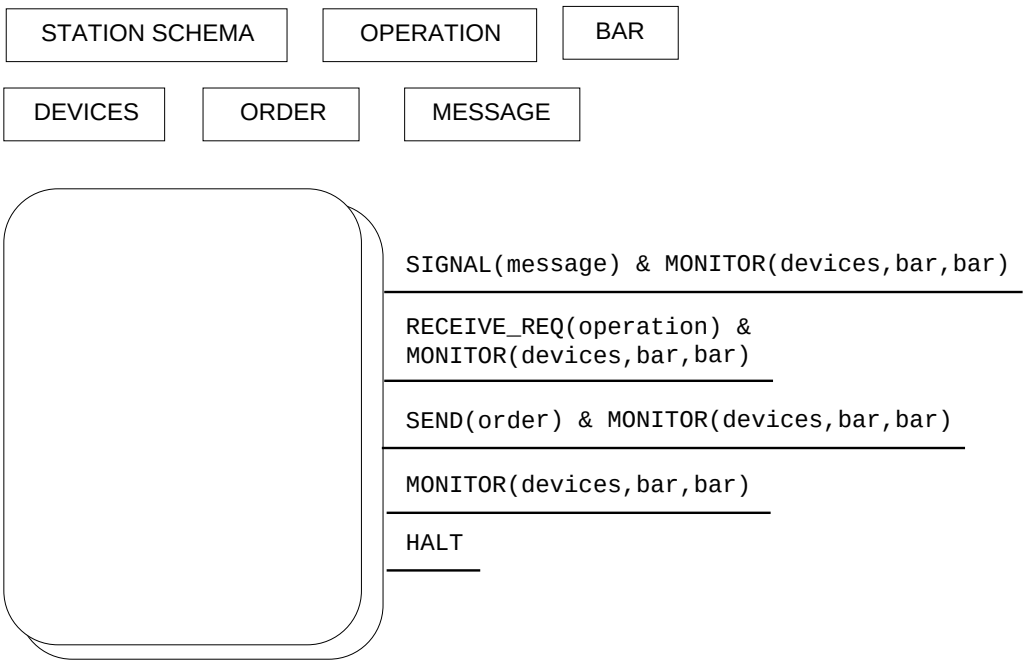


Initial States



Structure & Interactions





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
```

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

```
**  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 Failure(a) then
   a in any case <HALT> and after [not exists y, x': x -- y --> x']

**  If a device is in position XX, then there is a failure in the station
if  Device_Position(id,a)=XX  then  Failure(a)

**  If the automatism has sent to a device the order of opening and sees that it
**  is not open, then there is a failure in the station
if  a at least in a case P<SEND(Open(id))> and
    a -- ... & MONITOR(ds,ba,bb) --> a' and not Device_Position(id,a') = OP then
    Failure(a')

**  If the automatism has sent to a device the order of closing and sees that it
**  is not closed, then there is a failure in the station
if  a at least in a case P<SEND(Close(id))> and
    a -- ... & MONITOR(ds,ba.bb) --> a' and not Device_Position(id,a')=CL then
    Failure(a')

**  If a device changes position without receiving an order, then there is a
**  failure in the station
if  a -- la --> a' and Device_Position(id,a')=/=Device_Position(id,a) and
    forall ds, ba,bb: la=/=SEND(Open(id)) & MONITOR(ds,ba,bb) and
    la=/=SEND(Close(id)) & MONITOR(ds,ba,bb)  then
    Failure(a')

**  If the automatism receives an operation request for an unit connected
**  to a failed bar, then there will be a failure in the station
if  a -- RECEIVE_REQ(Opr(fid,up)) & MONITOR(ds,ba,bb) --> a' and
    On_Failed_Bar(fid,Schema(a),ba,bb)  then
    Failure(a)
```

```
**  checks whether a functional unit is connected to a failed bar
op  On_Failed_Bar: fun_unit_ident station_schema bar bar
ax  if On_BarA(fid,sch) then On_Failed_Bar(fid,sch,KO,bb)
ax  if On_BarB(fid,sch) then On_Failed_Bar(fid,sch,ba,KO)
```

Device_Position

```
**  If the automatism monitors the plant seeing that a device id has position p,
**  then the recorded position of id is p

if  a -- la --> a' and
    (la=MONITOR(ds,ba,bb) or la=SEND(0) & MONITOR(ds,ba,bb) or  )show grestore gs

    (Br(id,p) In ds or Is(id,p) In ds)  then
Device_Position(id,a') = p
```

Executing

```
**  If the automatism is executing and becomes not executing, then it signals
**  the end of an operation
if  a -- l --> a' and Executing(a) and not Executing(a') then
    exists ds,ba,bb,m: l = SIGNAL(m) & MONITOR(ds,ba,bb)

**  If the automatism is not executing and become executing, then it receives an
**  operation request
if  a -- l --> a' and not Executing(a) and  Executing(a') then
    exists opr,ds,ba,bb: l = RECEIVE_REQ(opr) & MONITOR(ds,ba,bb)
```

RECEIVE_REQ

```
**  If the automatism monitors the plant and receives the request of opening
**  an open functional unit, then in any case eventually either it signals
**  that the required operation is useless or there will be a failure in the
**  station
if  a -- RECEIVE_REQ(Opr(fid,Open)) & MONITOR(ds,ba,bb) --> a' and
    Unit_Position(Schema(a),ds,fid) = Open then
    a' in any case
        eventually <exists ds,ba,bb: SIGNAL(Useless) & MONITOR(ds,ba,bb)>
      or
        eventually [Failure(x)]

**  If the automatism monitors the plant and receives the request of opening
**  a closed Ae, then in any case eventually either it orders to the isolator
**  of opening and after signals that the operation has been completed or
**  there will be a failure in the station
if  a -- RECEIVE_REQ(Opr(fid,Open)) & MONITOR(ds,ba,bb) --> a' and
    Kind(Schema(a),fid) = Ae and Unit_Position(Schema(a),ds,fid) = Closed then
    a' in any case
      eventually
      <exists ds,ba,bb:
         l = SEND(Open(Isolator(Schema(a),fid))) & MONITOR(ds,ba,bb)>
         and after <exists ds,ba,bb: SIGNAL(Completed) & MONITOR(ds,ba,bb)>
      or
      eventually [Failure(x)]

**  If the automatism monitors the station components and receives the request
**  of opening a closed Dd, then in any case eventually either it orders to the
**  breaker of fid of opening, after to the isolator on the bar A of fid of
**  opening, after orders to the isolator on bar B of fid of opening and after
**  signals that the operation has been completed or there will be a failure in
**  the station
if  a -- RECEIVE_REQ(Opr(fid,Open)) & MONITOR(ds,ba,bb) --> a' and
    Kind(Schema(a),fid) = Dd and Unit_Position(Schema(a),ds,fid) = Closed then
    a' in any case
    eventually
    <exists ds,ba,bb: l=SEND(Open(Breaker(Schema(a),fid))) & MONITOR(ds,ba,bb)>
    and after
    <exists ds,ba,bb:
     l=SEND(Open(Isolator_On_A(Schema(a),fid))) & MONITOR(ds,ba,bb)>
    and after
    <exists ds,ba,bb:
    l=SEND(Open(Isolator_On_B(Schema(a),fid))) & MONITOR(ds,ba,bb)>
    and after <SIGNAL(Completed)>
    or eventually [Failure(x)]
```

RECEIVE_REQ

```
**  If the automatism monitors the station components and receives the request
**  of closing on bar A an Fa closed on bar B, and there exists a closing path
**  made by a closed Dd, then in any case eventually either it orders to the
**  isolator on A of closing, after orders to the isolator on B of opening,
**  after signals that the operation has been completed or there will be a
**  failure in the station
if  a -- RECEIVE_REQ(Opr(fid,Close_On_A)) & MONITOR(ds,ba,bb) --> a' and
    Kind(Schema(a),fid) = Fa and Unit_Position(Schema(a),ds,fid) = Close_On_B
    and Find_Closing_Path(Schema(a),fid,ds) = ClosedDd then
    a' in any case eventually
      <exists ds,ba,bb:
         l = SEND(Close(Isolator_On_A(Schema(a),fid))) & MONITOR(ds,ba,bb)>
      and after
        <exists ds,ba,bb:
          l = SEND(Open(Isolator_On_B(Schema(a),fid))) & MONITOR(ds,ba,bb)>
     and after
         <SIGNAL(Completed)>
     or eventually [Failure(x)]

**  If the automatism monitors the station components and receives the request
**  of closing on bar A an Fa closed on bar B and there exists a closing path
**  made by the open Dd fid', then in any case eventually
**  either it orders to the breaker of fid' of closing, after
**  orders to the isolator on bar A of fid' of closing, after
**  orders to the isolator on bar B of fid' of closing, after
**  orders to the isolator on A of fid of closing, after
**  orders to the isolator on B of fid of opening and
**  signals that the operation has been completed or
**  there will be a failure in the station
if  a -- RECEIVE_REQ(Opr(fid,Close_On_A)) & MONITOR(ds,ba,bb) --> a' and
    Kind(Schema(a),fid) = Fa and
    Unit_Position(Schema(a),ds,fid) = Close_On_B and
    Find_Closing_Path(Schema(a),fid,ds) = OpenDd(fid') then
    a' in any case eventually
    <exists ds,ba bb:
       l = SEND(Close(Breaker(Schema(a),fid'))) & MONITOR(ds,ba,bb)>
     and after
      <exists ds,ba,bb:
       l = SEND(Close(Isolator_On_A(Schema(a),fid'))) & MONITOR(ds,ba,bb)>
     and after
      <exists ds,ba,bb:
       l = SEND(Open(Isolator_On_A(Schema(a),fid'))) & MONITOR(ds,ba,bb)>
     and after
      <exists ds,ba,bb:
       l = SEND(Close(Isolator_On_A(Schema(a),fid))) & MONITOR(ds,ba,bb)>
     and after
      <exists ds,ba,bb:
       l = SEND(Open(Isolator_On_B(Schema(a),fid))) & MONITOR(ds,ba,bb)>
     and after
      <SIGNAL(Completed)>
     or eventually [Failure(x)]
```

```
**  If the automatism monitors the station components and receives the request
**  of closing on bar A an Fa, sees that it is closed on bar B and there exists
**  no closing path, then in any case eventually either it signals that the
**  required operation is impossible or there will be a failure in the station
if  a -- RECEIVE_REQ(Opr(fid,Close_On_A)) & MONITOR(ds,ba,bb) --> a' and
    Kind(Schema(a),fid) = Fa and and
    Unit_Position(Schema(a),ds,fid) = Close_On_B and
    Find_Closing_Path(Schema(a),fid,ds) = None then
    a' in any case
    eventually <SIGNAL(Impossible)> or eventually [Failure(x)]
```

```
**  given the schema of the devices of a functional unit and the states of the
**  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
```

```
cn  None, ClosedDd: answer
op  OpenDd: unit_ident -> answer
```

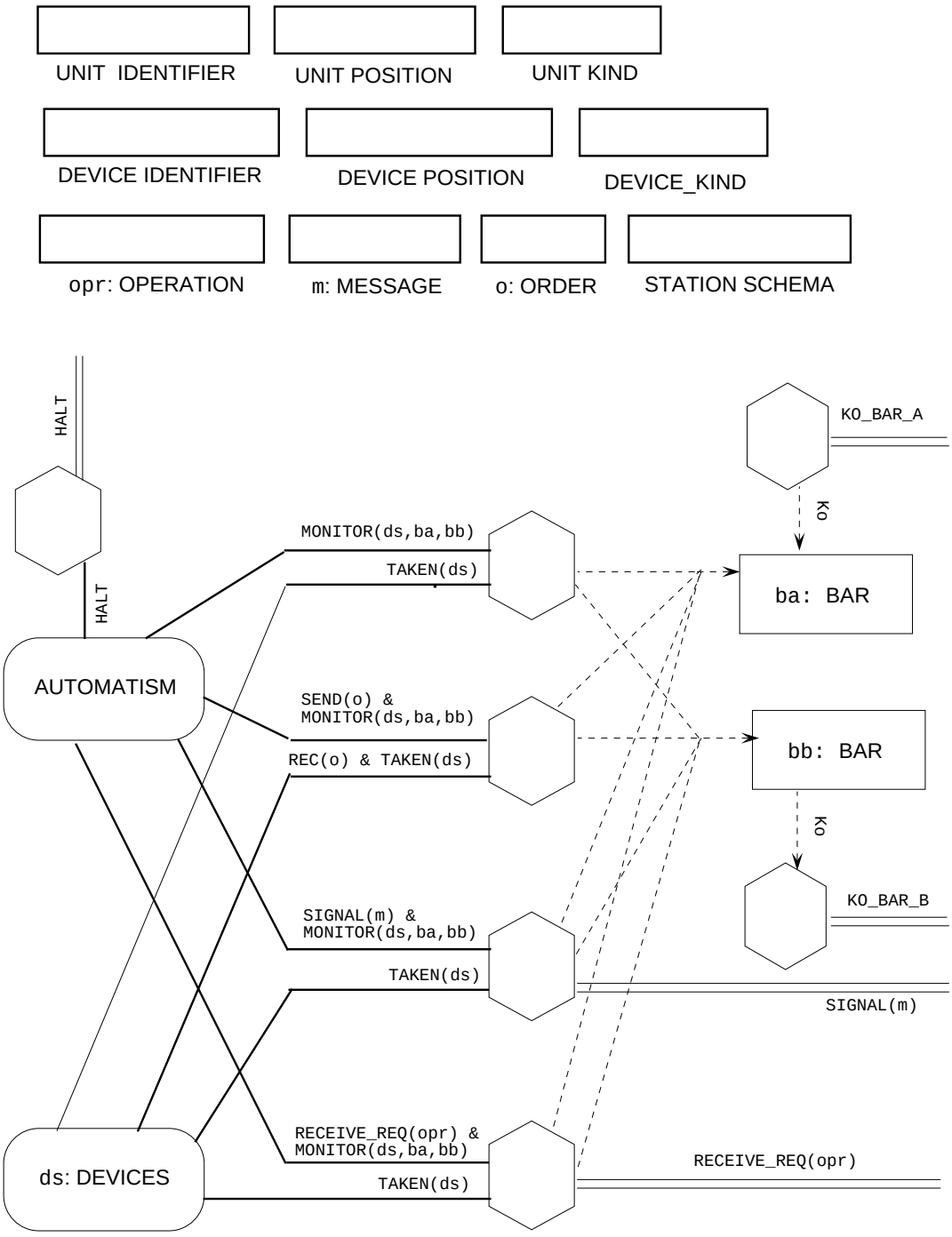
```
**  Find_Closing_Path given a station schema, a devices state and the identifier
**  of an Fa fid says whether either for fid no closing path exists, a closing
**  path made by an open or by a closed Dd exists
op  Find_Closing_Path: station_schema unit_ident devices -> answer

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

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

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

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

ax  if fid =/= fid1 and fid =/= fid2 and SameSection(sch,fid1,fid2,ds) then
    SameSection(Fu(fid,dsch) & sch,fid1,fid2,ds)
ax  if Connect(sch,fid,ds) then
    SameSection(<Fu(fid,dsch)> & sch,fid,fid',ds)
ax  if Connect(sch,fid,ds) then
    SameSection(<Fu(fid,dsch)> & sch,fid',fid,ds)
ax  if SameSection(sch,fid1,fid2,ds) then
    SameSection(<fus1 fus2> & sch,fid1,fid2,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 fid =/= fid' and Connect(sch,fid',ds) then
    Connect(<Fu(fid,dsch)> & sch,fid',ds)
ax  Connect(<Fu(fid,dsch)> & sch,fid,ds)
ax  if Position(Ae(id1),ds)=Open  and Position(Ae(id2),ds)=Open and
    Connect(sch,fid,ds) then
    Connect(<Fu(fid1,Ae(id1)) Fu(fid2,Ae(id2))> & sch,fid,ds)
```

```
**  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(fid,dsch)> & sch,ds,fid)=Position(dsch,ds)
ax  if fid =/= fid' then
    Unit_Position(<FU(fid',dsch)> & sch,ds,fid)=Unit_Position(sch,ds,fid)
ax  Unit_Position(<FU(fid,dsch1) FU(fid,dsch2)> & sch,ds,fid)=
    Position(dsch1,ds)
ax  Unit_Position(<FU(fid1,dsch1) FU(fid,dsch2)> & sch,ds,fid)=
    Position(dsch2,ds)
ax  if fid =/= fid1 and fid =/= fid2 then
    Unit_Position(<FU(fid1,dsch1) FU(fid2,dsch2)> & sch,ds,fid)=
    Unit_Position(sch,ds,fid)
```

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; since 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 is started, each functional unit manager begins to monitor the devices of the associate functional unit; if it detects a failure, the console is informed 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 send 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 which 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.

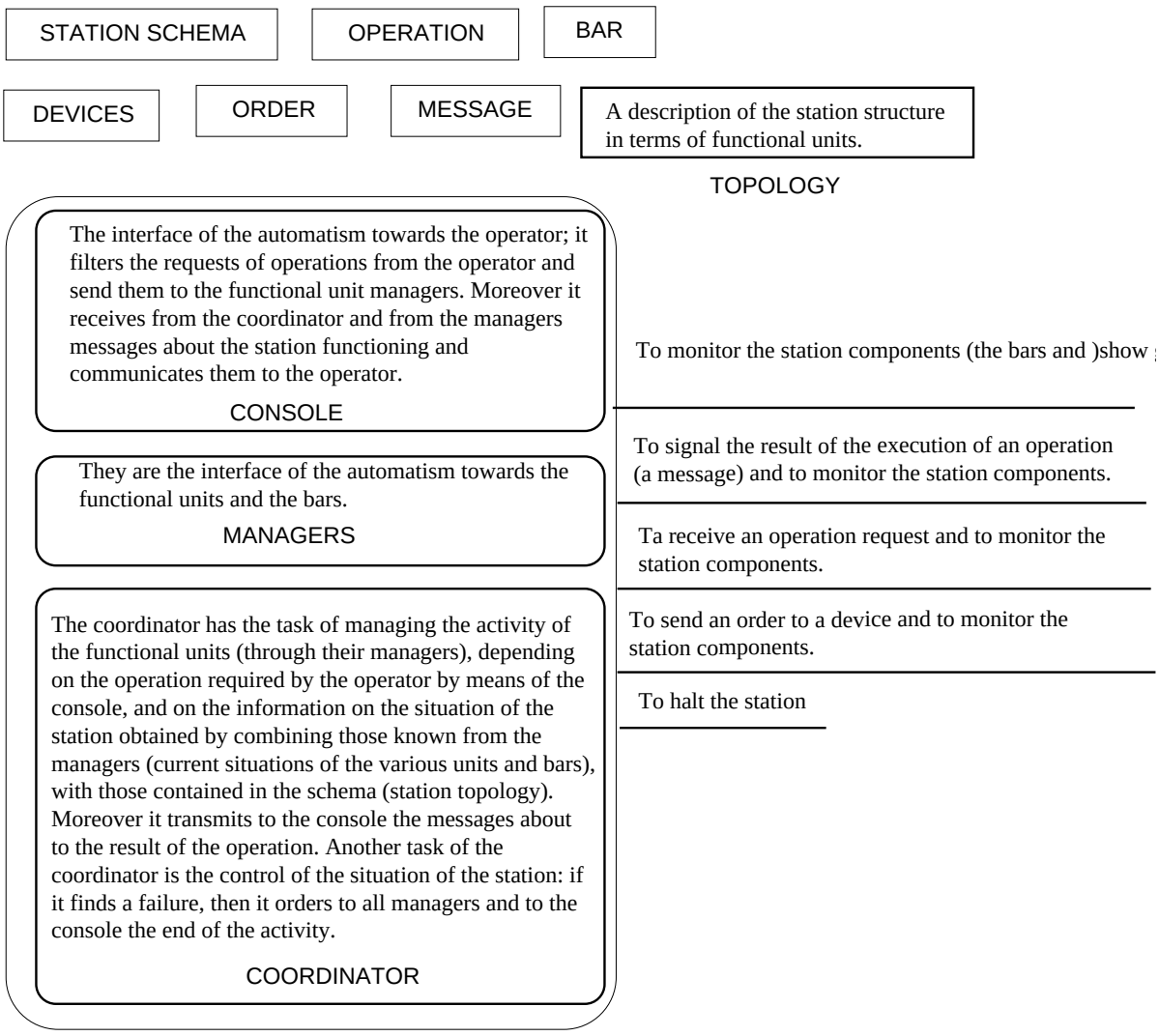
In each case it checks that the operation is valid, i.e., that the bars connected to the functional unit to be used are not failed; to do that it reads in the corresponding managers the situations of the involved bars and if one of them is failed, it informs the console and all managers that there is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

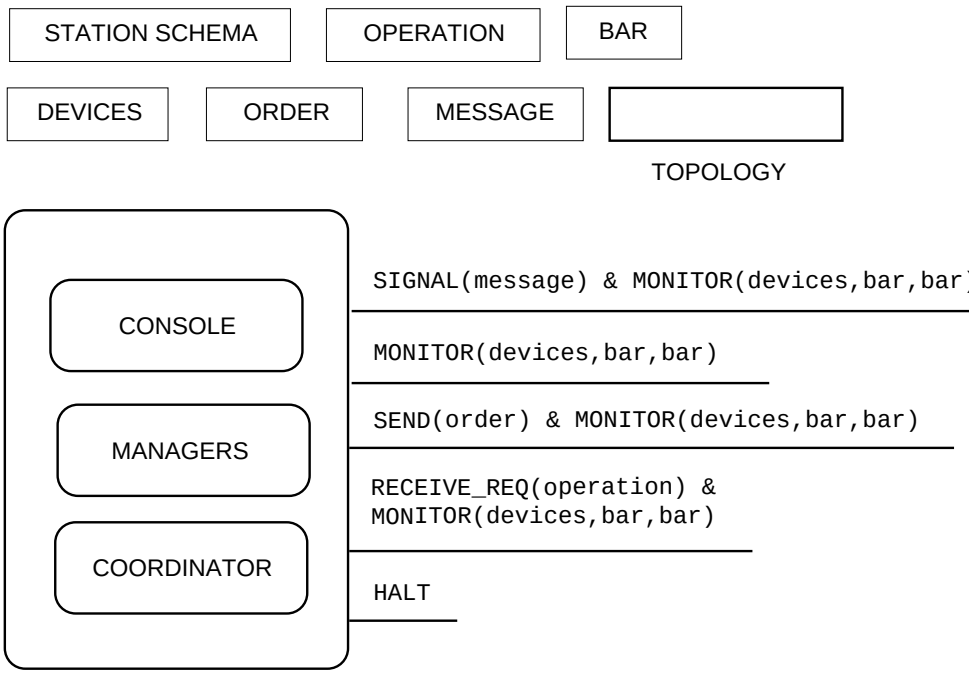
In the case of closing on a bar of a functional unit of kind Fa, the coordinator must determine if the operation is either of bar exchange or of closing; to do that it checks the situation of such Fa (reading it in the corresponding manager).

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.

Structure & Interactions




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

UNIT IDENTIFIER | UNIT KIND

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

cn  E: topology  **  empty station topology
op  Fa(_) & _: unit_ident topology -> topology
op  Dd(_) & _: unit_ident topology -> topology
op  Ae(_, _) & _: unit_ident unit_ident topology -> topology
```

```
**  given a topology return the part 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)
```

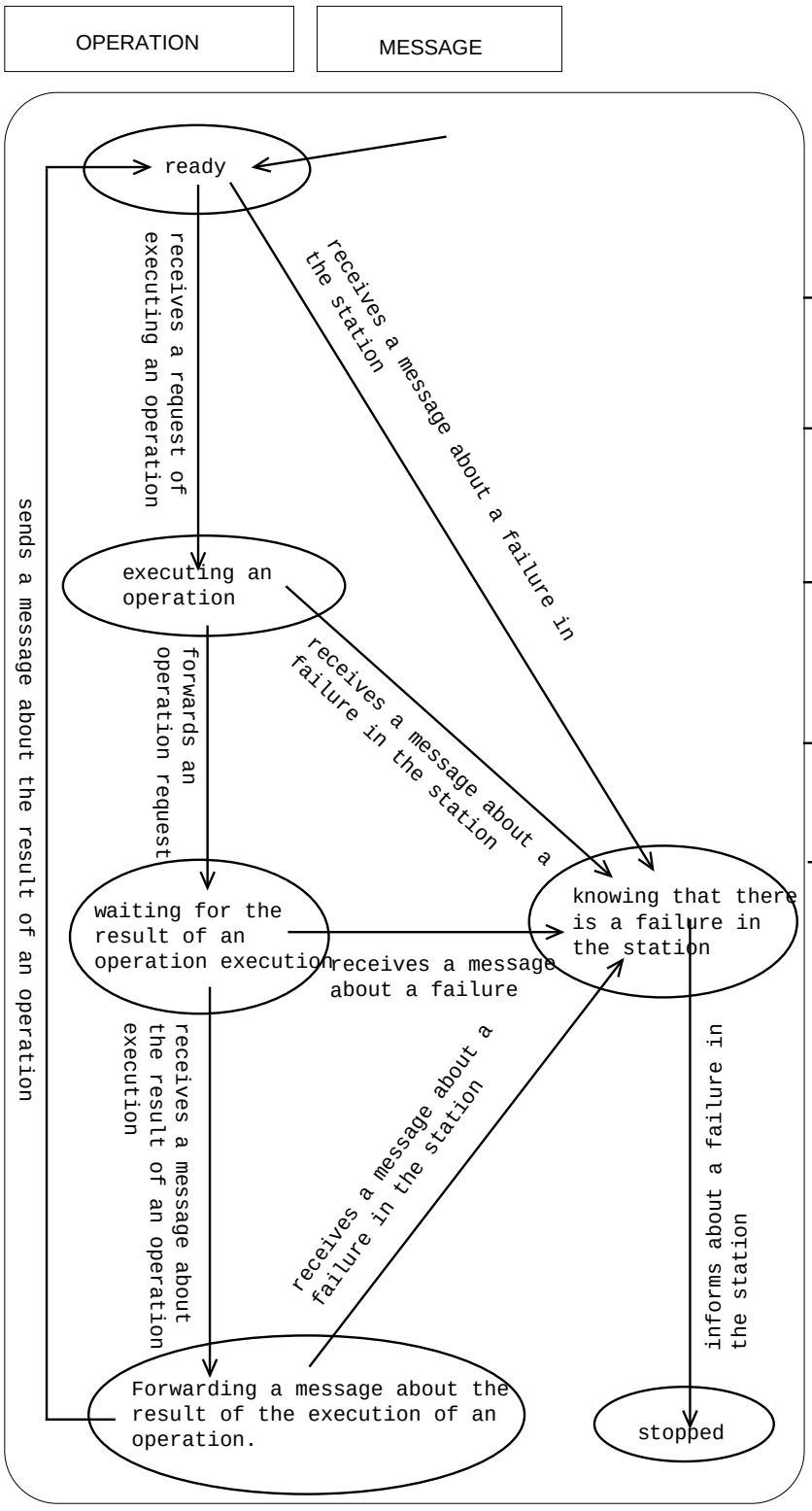
```
**  given a station topology and a functional unit identifier returns its kind
op  Kind: topology unit_ident -> kind (partial)

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)



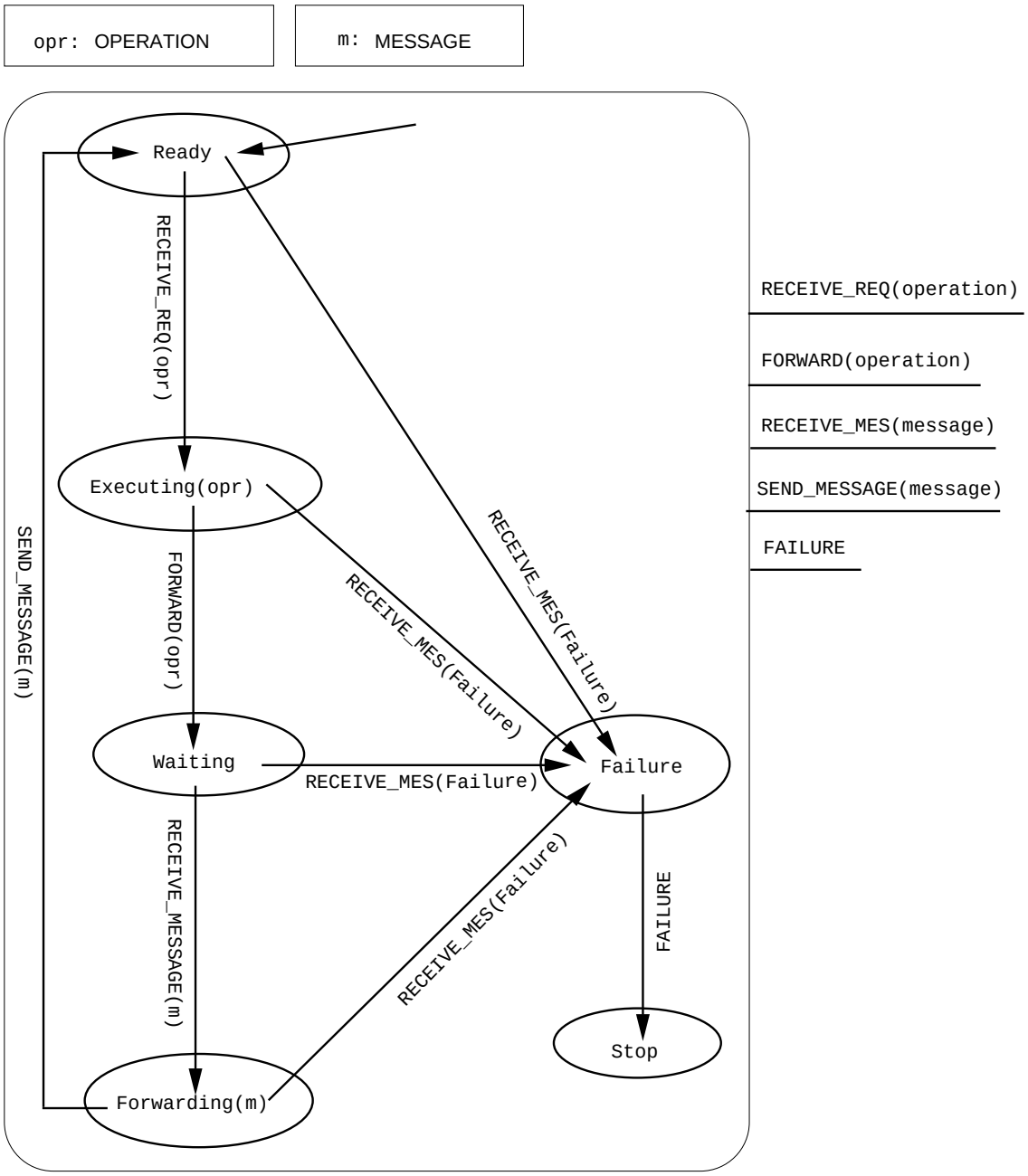
To receive the request of executing an operation from the operator.

To forward to the unit managers an operation request.

To receive a message about the result of the execution of an operation from the unit managers.

To send to the opertor a message about the result of the execution of an operation.

To inform the operator of a failure in the station.

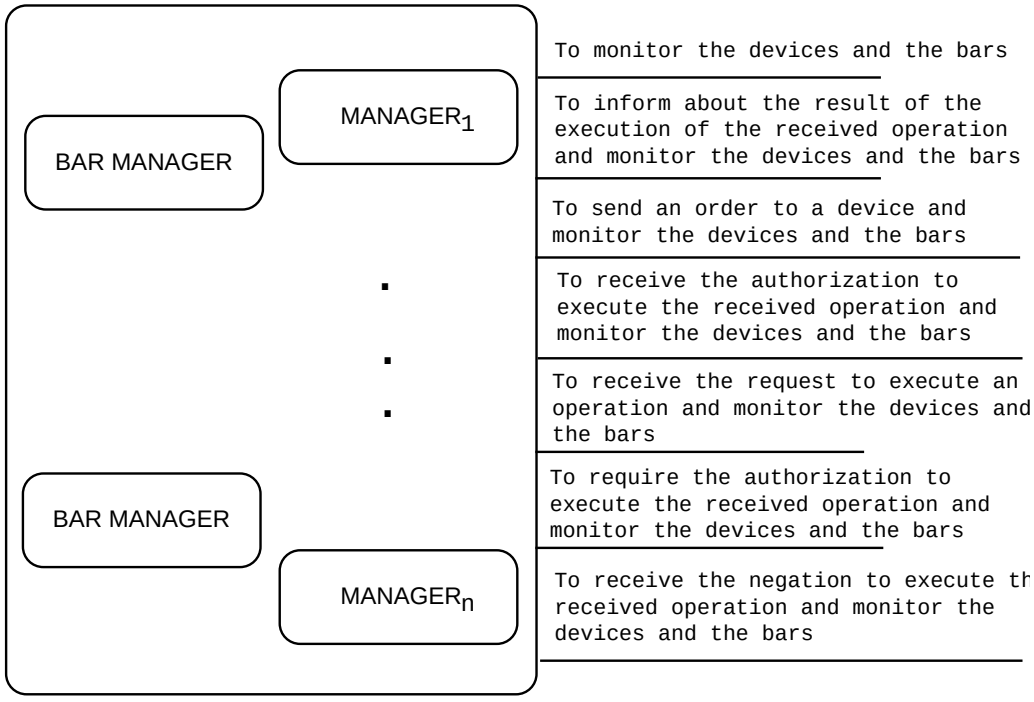

opr: OPERATION
m: MESSAGE
Ready
RECEIVE_REQ(opr)
Executing(opr)
FORWARD(opr)
Waiting
RECEIVE_MESSAGE(m)
Forwarding(m)
SEND_MESSAGE(m)
RECEIVE_MES(Failure)
RECEIVE_MES(Failure)
RECEIVE_MES(Failure)
Failure
FAILURE
Stop
RECEIVE_REQ(operation)
FORWARD(operation)
RECEIVE_MES(message)
SEND_MESSAGE(message)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

MANAGERS: Specification

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

M_MESSAGE

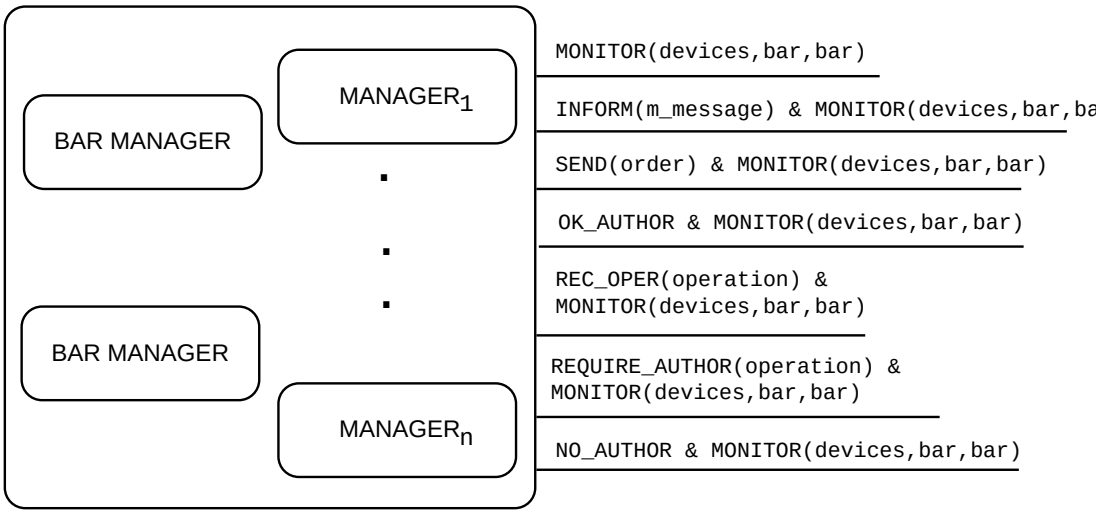
ORDER	DEVICES	OPERATION	BAR



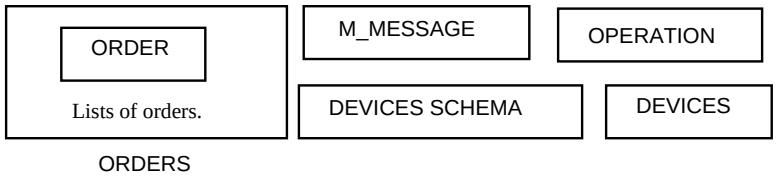
enum: Failure, Useless, Executed

M_MESSAGE

ORDER	DEVICES	OPERATION	BAR



Structure & Interactions



ORDERS

The interface of the automatism towards a functional units.

To monitor the unit devices

To inform about the result of the execution of the received operation and monitor the unit devices

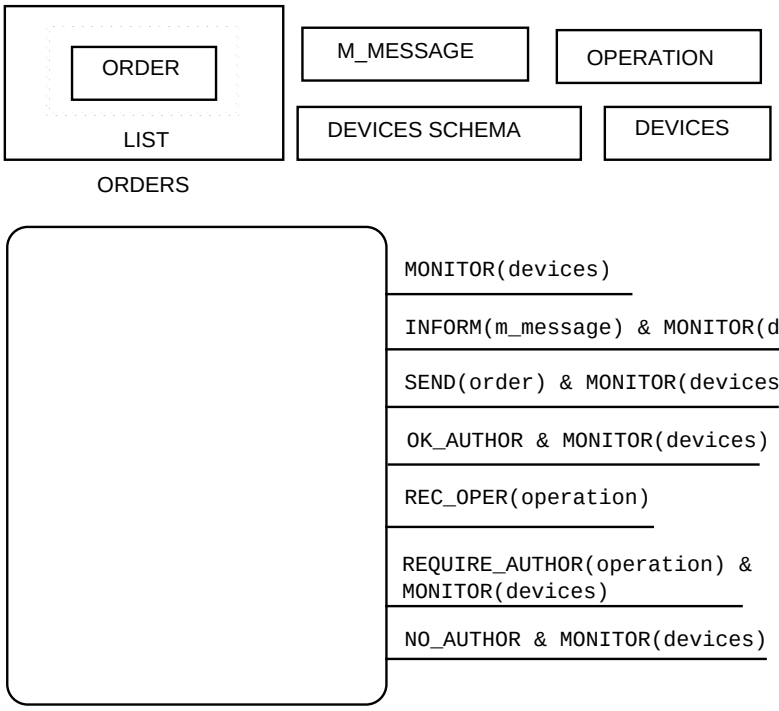
To send an order to a device and monitor the unit devices

To receive the authorization to execute the received operation and monitor the unit device

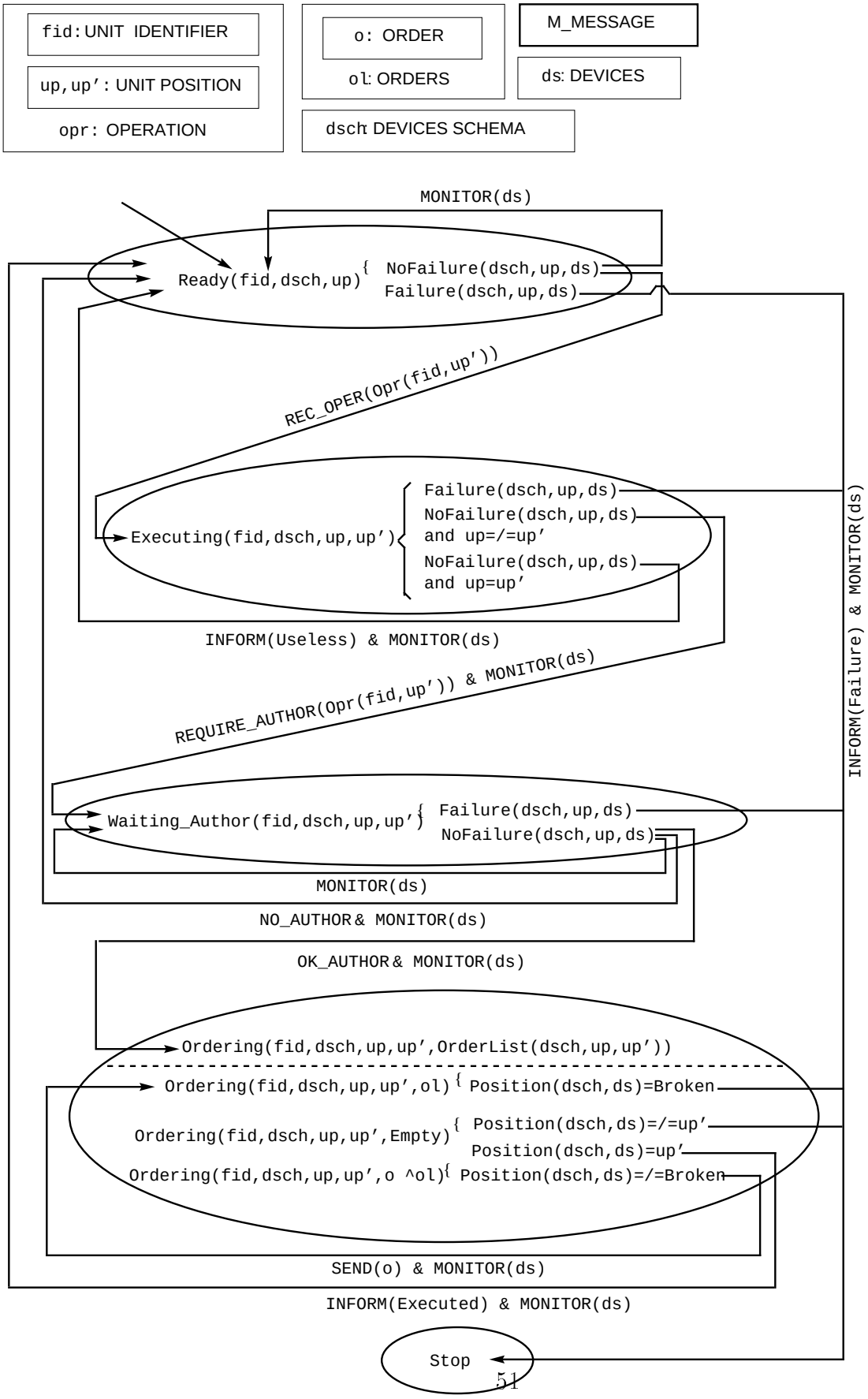
To receive the request to execute an operation on the unit

To require the authorization to execute the received operation and monitor the unit device

To receive the negation to the execution of th received operation and monitor the unit device


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: **Activity**



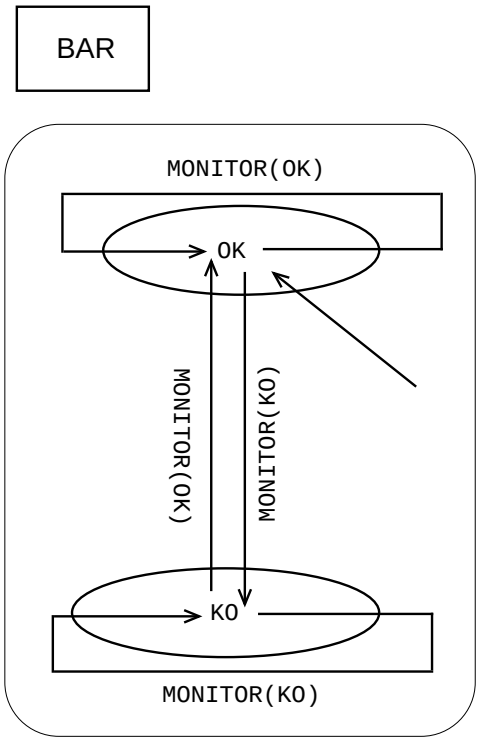
```
**  given the schema of a functional unit and the set of the states of its
**  devices returns its position
op  Position: devices_schema devices -> unit_position         partial

**  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 is a failure in the managed functional unit
pr  Failure, No_Failure: devices_schema unit_position devices

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

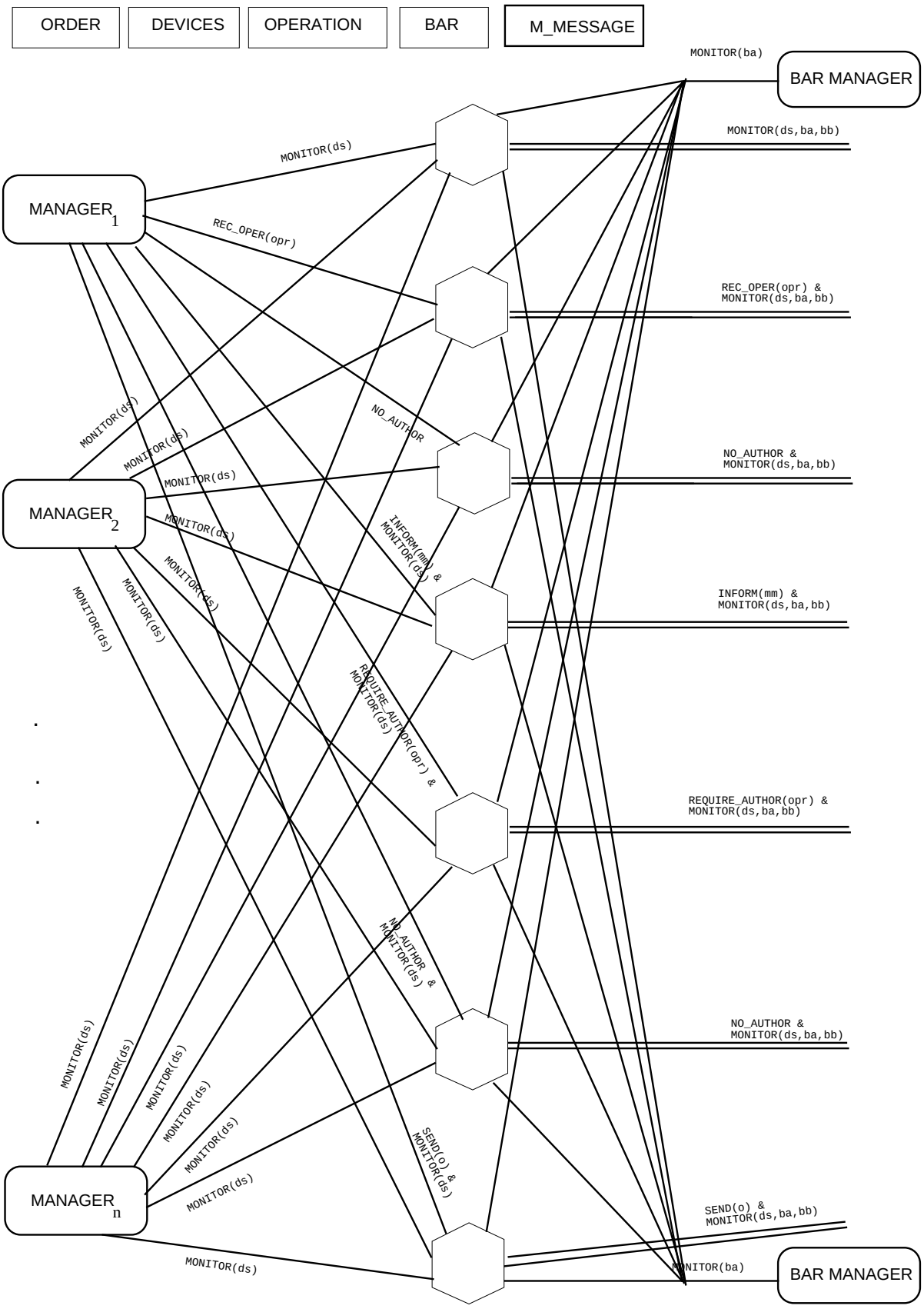
auxiliary (continues)

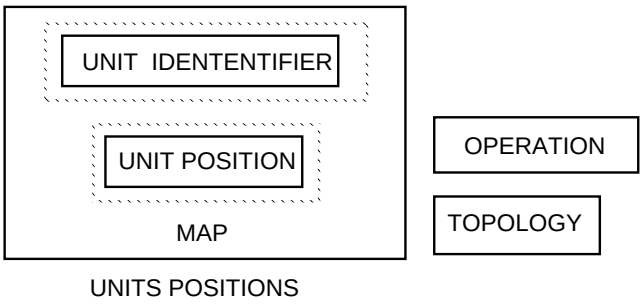
```
**  given the schema a functional unit, its position and the position to reach,
**  returns the list of the orders to be sent to its devices
op  Order_List: devices_schema unit_position unit_position -> orders

**  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 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 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 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
MONITOR(OK)
OK
MONITOR(KO)
MONITOR(OK)
KO
MONITOR(KO)

MANAGERS: **Activity**





UNITS POSITIONS

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

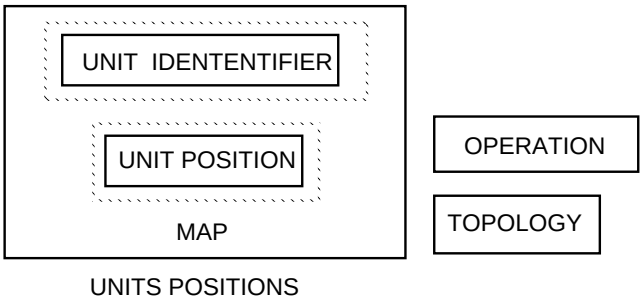
To deny the authorization to the execution of an operation

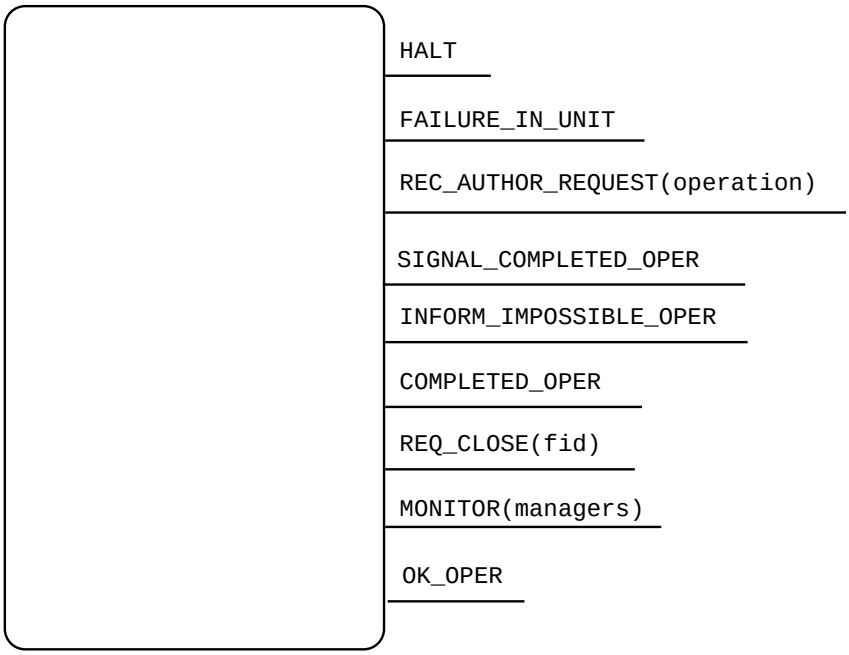
To be informed that the operation has been completed

To require to close a Dd

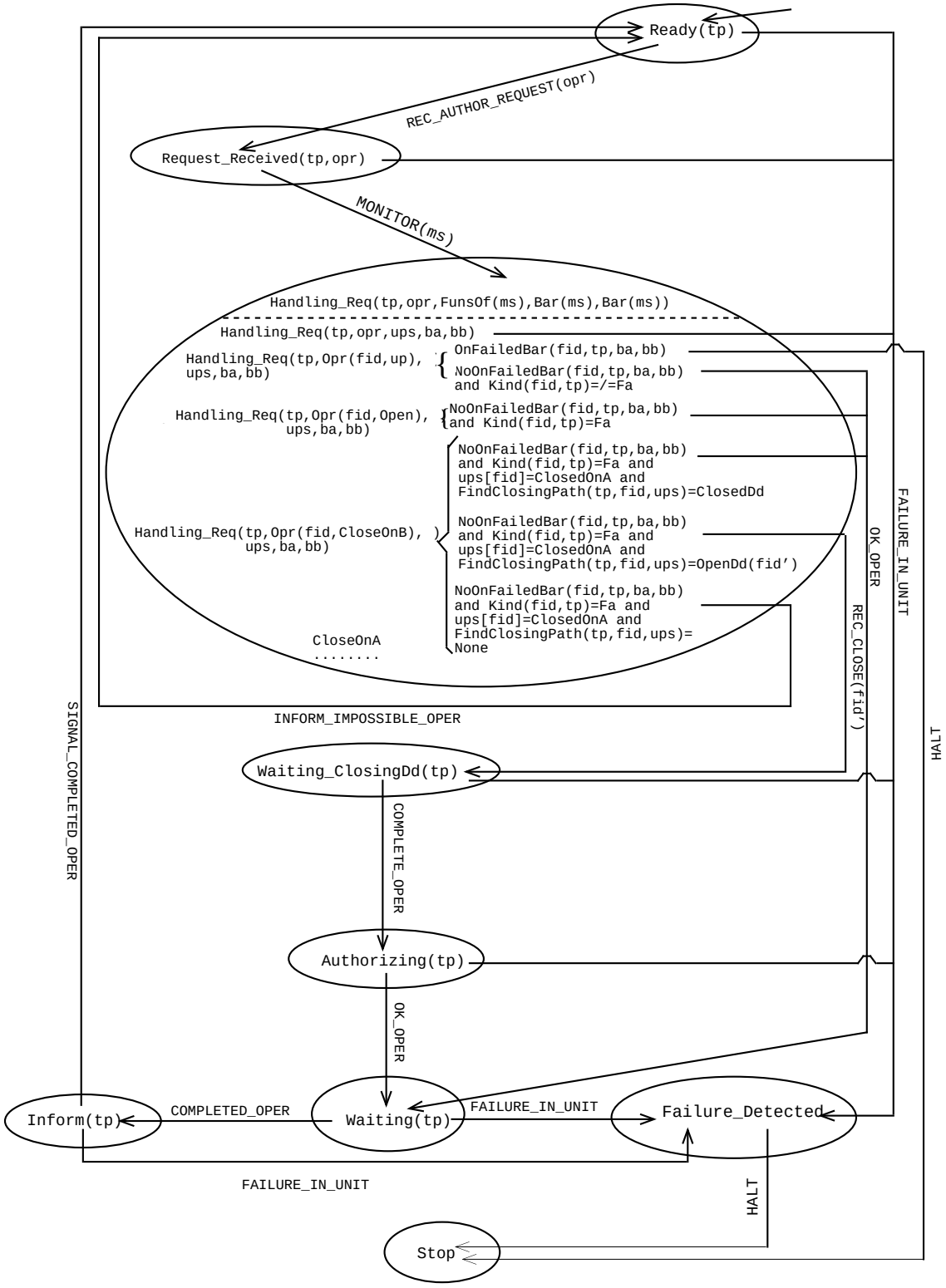
To monitor the units and the bars throughout their managers

To authorize the execution of an operation


UNIT IDENTENTIFIER
UNIT POSITION
MAP
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: **Activity**



```
sort  answer
cn  None, ClosedDd: answer
op  OpenDd: unit_ident -> answer
```

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

```
**  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 fida there exists a proper closing
**  path, then there exists a proper closing path for idfa
ax  if Path(RPart(tp,idfa),ups,None)=ClosedDd then
        Find_Closing_Path(tp,idfa,ups)=ClosedDd

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

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

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

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

auxiliary (continues)

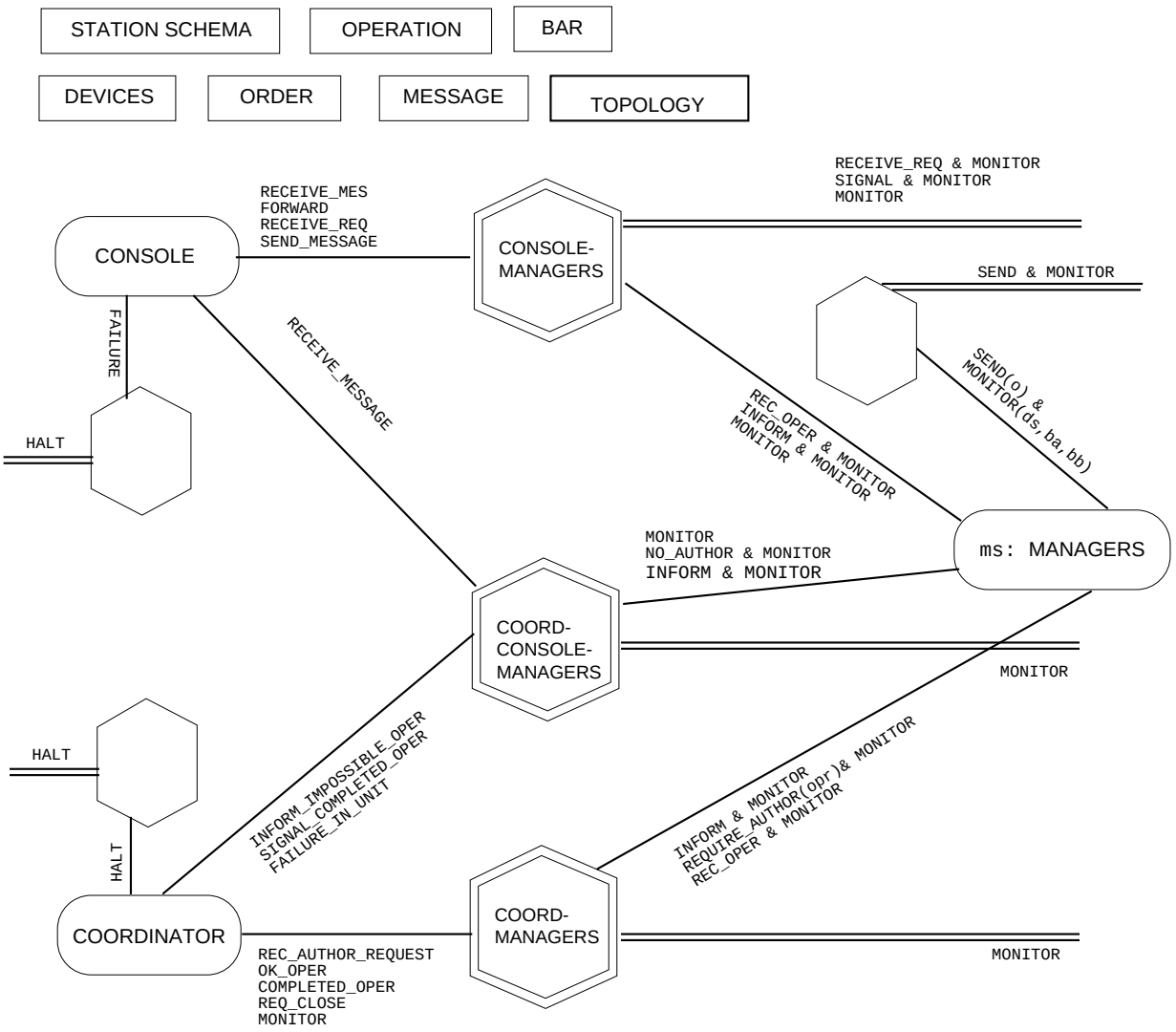
```
  **  Path given the topology of one part of the station, the functional unit
  **  positions and the identifier of an Fa returns an answer saying whether in
  **  such part of the station for such unit no closing path exists, a closing
  **  path made by an open or by a closed Dd exists
  op  Path: 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 fid is a closed Dd then Path returns there is a closing path made by
  **  a closed Dd
  ax  if ups[fid] = Close then  Path(Dd(fid) & tp,ups,a) = ClosedDd

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

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

  **  If fid1 and fid2 are two closed Ae, then the scanning of the bars goes on
  ax  if ups[fid] = Close and ups[fid] = Close then
          Path(Ae(fid1,fid2) & tp,ups,a) = Path(tp,ups,a)

  **  If either fid1 or fid2 is open, then the recorded answer is returned
  ax  if (ups[fid] = Open or ups[fid] = Open) then
          Path(Ae(fid1,fid2) & tp,ups,a) = a

  **  If fid is an Fa, then the scanning of the bars goes on
  ax  Path(Fa(fid) & 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(fid,tp) then No_On_Failed_Bar(fid,tp,KO,bb)
ax  if Non_On_BarB(fid,tp) then No_On_Failed_Bar(fid,tp,ba,KO)

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

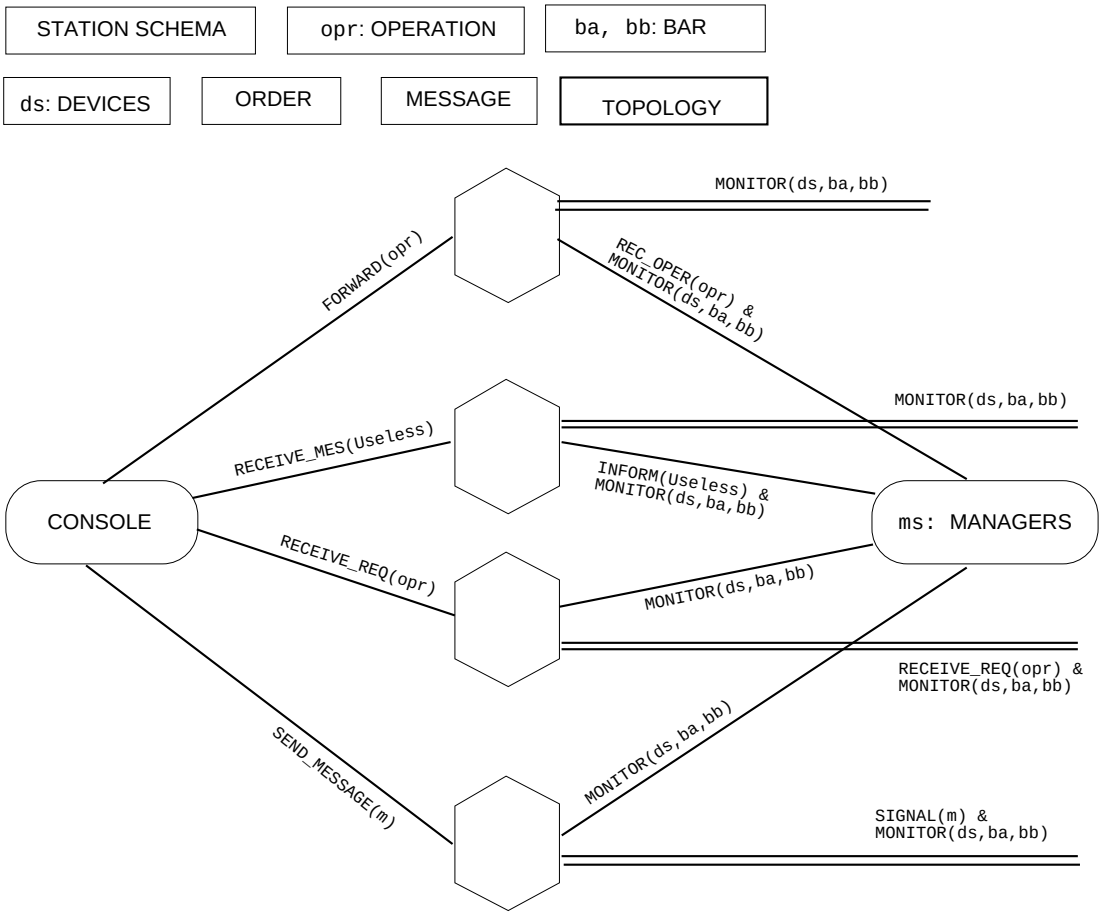
```
op  Bar: bar_manager -> bar
ax  Bar(OK) = OK
ax  Bar(KO) = KO
```

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

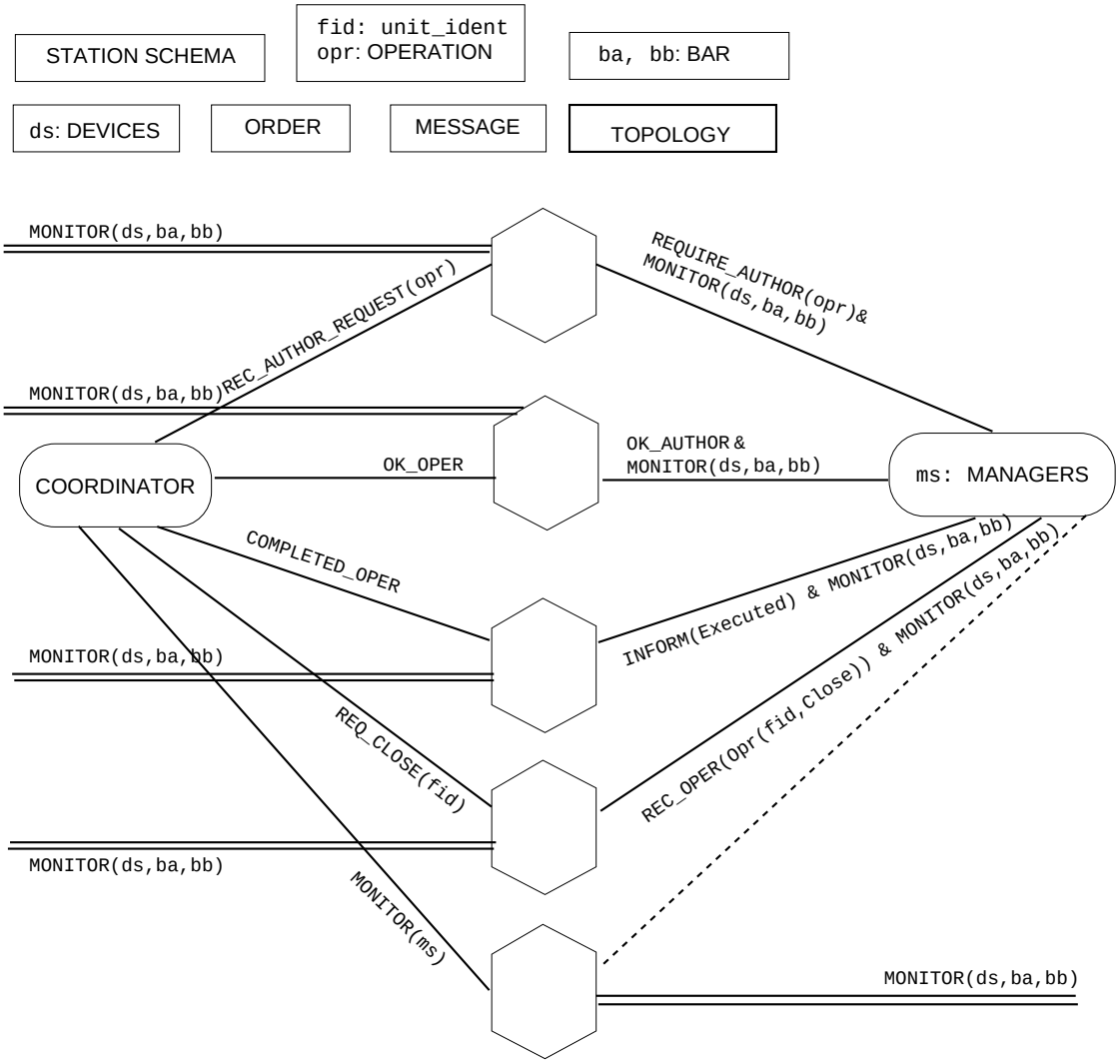
auxiliary (end)

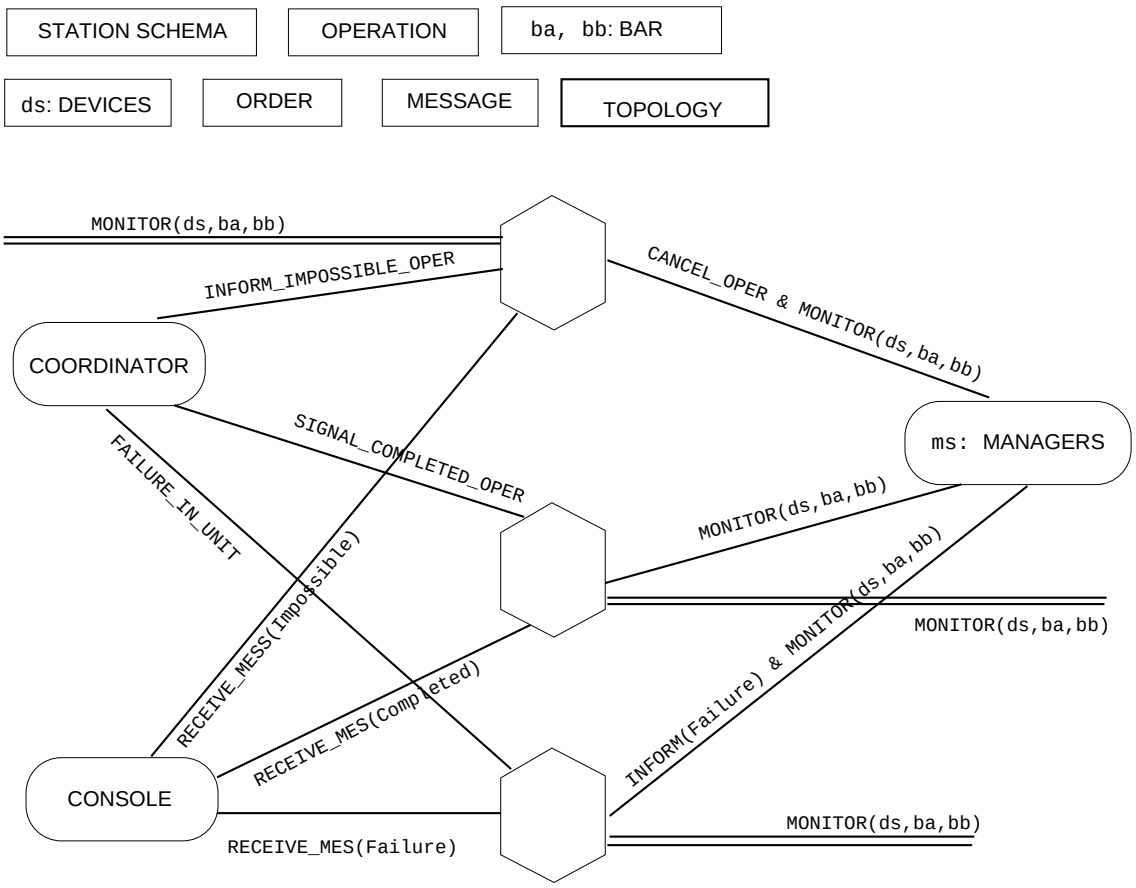

STATION SCHEMA
OPERATION
BAR
DEVICES
ORDER
MESSAGE
TOPOLOGY
CONSOLE
RECEIVE_MES
FORWARD
RECEIVE_REQ
SEND_MESSAGE
CONSOLE-
MANAGERS
RECEIVE_REQ & MONITOR
SIGNAL & MONITOR
MONITOR
SEND & MONITOR
FAILURE
HALT
RECEIVE_MESSAGE
REC_OPER & MONITOR
INFORM & MONITOR
MONITOR
SEND(o) &
MONITOR(ds, ba, bb)
ms: MANAGERS
MONITOR
NO_AUTHOR & MONITOR
INFORM & MONITOR
COORD-
CONSOLE-
MANAGERS
MONITOR
HALT
HALT
INFORM_IMPOSSIBLE_OPER
SIGNAL_COMPLETED_OPER
FAILURE_IN_UNIT
INFORM & MONITOR
REQUIRE_AUTHOR(opr)& MONITOR
REC_OPER & MONITOR
COORDINATOR
COORD-
MANAGERS
MONITOR
REC_AUTHOR_REQUEST
OK_OPER
COMPLETED_OPER
REQ_CLOSE
MONITOR

COORD-CONSOLE-MANAGERS



COORD-MANAGERS





References

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