Contents

1 RWS IHM specification using LIDL

2 Textual specification

2.1 Color code

This specification has been annotated using color codes in order to classify sentences in different concepts that can or cannot be expressed using LIDL.

- Data types
- Interfaces
- Interactions
- Architecture
- Timing/External/Ergonomy

2.2 Definitions

1. The RWS is the system in charge of the computation and presentation of alerts. It is composed of two mains part: a RWS core in charge of computing the state of all alerts, and a RWS HMI in charge of the presentation of the alerts to the supervisor.
2. The supervision operator is referred to as supervisor in the specification.
3. The terms **"event occurrence"** refers to something that takes place at a certain place in space and time. The term **"event"** refers to the set of all event occurrences that share certain common characteristics, in the same way as a class describes all the individual objects that share the same structure. An event is a class of event occurrences. "Activation of alert A" is an event. The actual activations of alert A at times t_1, t_2, t_n are occurrences of the event "Activation of alert A".
4. An alert refers to some specific condition on the monitored system. For instance, "left wheel is blocked" is an alert. Satisfaction (or "activation") of the condition "left wheel is blocked" is an event. The actual blocking of the wheel at time t is an occurrence of the event "left wheel is blocked".
5. An alert signal is the means by which the supervisor is informed of the occurrence of the alert activation. The transmission of the alert signal to the supervisor is also called "presentation". It shall be noted that if the occurrence of the alert signal is conditioned by the satisfaction of some alert condition, the signal may continue to exist while the condition is negated. It shall also be noted that the signal may cease to be transmitted to the supervisor for some time or be transmitted to the supervisor on demand (see specification).
6. Activation conditions are evaluated from data produced by the rovers, the supervision station, the in-door stations, and the activation state of other alerts.
7. The monitored system is composed of: (i) a set of rovers, (ii) a supervision station, (iii) a set of in-door GPS stations (3 stations). It also covers all required communication means between these elements.
8. The global system is composed of the monitored system and the supervisor.
9. Alerts are categorized as "caution", "warning" or "advisory".

2.3 Specification

1. **Signalling of alerts** The RWS IHM shall signal alerts occurrences to the supervision operator. The reception of an alert signal shall provide the supervisor with sufficient information about the situation (e.g., which alert, at what time,...) during sufficient time. We consider that situation awareness is possible if the information is presented continuously to the supervisor for at least 3 seconds.

2. **Separation between alerts signals** The signalling of an alert occurrence shall not alter / jeopardize the signalling of another alert occurrence. *Rationale: Let the occurrence of alert A be presented to the supervisor when occurrence of alert B occurs. Then the message related to B shall not cover the message related to A up to the point where it makes message related to A unintelligible. Separation between messages related to alert A and alert B may be ensure in space and/or time.*
3. **Navigation through alert signals** If all alert signals cannot be transmitted at once to the supervisor, s/he shall be able to "navigate" through the set of alert signals in order for the transmission to be done sequentially.
4. **The RWS is hosted on the supervision station.** The rovers communicate directly with the supervision station. The in-door GPS stations communicate indirectly with the supervision station through the rovers.
5. **Uniqueness of alert signals** There shall be a unique signal for a given alert. If the signal for alert A is active (presented or not to the supervisor), then no other occurrence of alert A can be signalled to the supervisor.
6. **Presentation latency** An alert signal shall be transmitted to the supervisor within at most 100ms after the triggering of the activation condition.
7. **Presentation duration** An alert signal shall be transmissible (not necessarily transmitted) to the supervisor until it is cleared.
8. **Acknowledgment** The RWS IHM shall provide the supervisor with a means to confirm that s/he has actually received the alert signal. This is referred to as « acknowledgement ».
9. **Automatic acknowledgment** An alert may be acknowledged automatically when the alert activation condition is negated.
10. **Concurrence between automatic and manual acknowledgment** An alert shall always be acknowledgeable by the supervisor even if automatic acknowledgment is configured.
11. **Acknowledgment configuration** The choice between automatic or supervisor-initiated acknowledgement shall be alert-specific and be defined by configuration.
12. **Acknowledgment state transmission** The alert signal shall transmit the acknowledgement state of the alert.
13. **Indication of non-transmitted alert signals** If the RWS IHM cannot transmit the alert signal to the supervisor (for instance due to lack of space), it shall at least transmit a signal indicating that there is a non-transmissible alert signal.
14. **Provision of means to transmit alert signals** If the RWS IHM cannot transmit the alert signal to the supervisor (for instance due to lack of space), it shall provide some means to activate the transmission of those alert signals. Hints shall be provided to the supervisor to facilitate this operation.
15. **Number of non-transmitted alert signals** If all active alerts cannot be presented simultaneously, an indicator shall be provided to the supervisor to let him know the total number of non-transmitted alert signals.
16. **Manual clearing of alert signals** The RWS HMI shall provide a means for the supervisor to clear an alert signal.
17. **Clearing alert signals** Once cleared, an alert signal shall cease to be transmitted to the supervisor. Note that a new occurrence of the same alert signal may be transmitted later to the supervisor if the associated alert condition becomes asserted again.
18. **Relation between acknowledging and clearing alert signals** An alert may only be cleared after being acknowledged.
19. **Automatic clearing of alert signals** An acknowledged alert signal may be cleared automatically when its activation condition is negated.
20. **Clearing configuration** The choice between automatic or supervisor-initiated clearing shall be alert-specific and be defined by configuration.
21. **Priorities of alert signals** Alert signals shall be transmitted to the supervisor in such a way that alerts with high priorities are received by the supervisor preferably before alerts with lower priorities. Priority is defined according to the following precedence relations (level n+1 is used to sort alerts when attributes at levels n are equal):

    1. "warning" > "caution" > "advisory"
    2. priority within the category

3. “not acked” > “acked”
4. increasing time stamp. Timestamps are unique and monotonically increasing.

22. **Attention getter** The attention of the supervisor shall be captured when an alert signal occurs.
23. **Procedures** The supervisor shall be able to activate the presentation of the procedure associated with any signalled alert. Any alert may have an associated procedure. This is defined by configuration.
24. **Relation between procedures and alert signals** The presentation of the alert / procedure shall clearly establish the relation between the alert signal and the procedure.
25. **Procedure contents** A procedure describes a sequence of actions to be performed by the supervisor. Each actions is called a “step”. Each step is associated with a message to be presented to the supervisor. The message describes an action to be carried out by the supervisor to diagnose, correct, etc. the associated alert. At a given step in the procedure, the last 2 steps (if possible) and the next 2 steps (if possible) shall be presented to the user so that s/he can build a mental representation of what he is doing and anticipate the next actions.
26. **Presentation of the current step** The current step shall be clearly indicated on the presented procedure.
27. **Cancelling a procedure** The supervisor shall be able to cancel a procedure. Cancelling a procedure means « ceasing » to present the procedure. The procedure may be presented again. In that case, it shall start again at the first step of the procedure.
28. **Presentation of the procedure state** The state of the procedure (“not yet started”, “completed”, or “at step ”) shall be transmitted by the alert signal.
29. **Stacking of procedures** It shall be able to present procedure for alert B while procedure for alert A is being presented. The supervisor shall be able to come back to procedure A and continue the procedure from the step where it was interrupted.
30. **Initial step of the procedures** All procedures are set in their initial state at system startup.
31. **Effect of alert signal clearing on procedures** When an alert is manually cleared, the procedure shall be abandoned (even if it was interrupted by the opening of another alert). Note: the procedure shall not be abandoned if the alert is automatically cleared.
32. **Effect of alert signal clearing on procedure stacking** When a procedure is cancelled, any active procedure that was interrupted by the one that is cancelled shall be presented again (in FIFO order).

33. **Procedure actions** The next step of a procedure may be determined by a supervisor action. Actions are “YES/NO/ DONE”.

3 LIDL specification

3.1 Data types

In this section we describe the Data types expressed in the specification. We mainly use the `data` keyword of LIDL.

We use the *new* Algebraic data types notation of LIDL here, because it is much more expressive and convenient than the previous record-only data types.

Note that we also define several interactions which are associated with the data types. These data types and their associated interactions represent the abstract data types we are interested in.

3.1.1 Pre existing data types

LIDL has several built-in types. Here are their “definition” using the algebraic data type syntax:

```
data Boolean is (true)|(false)

data Number is (1)|(2)|(3)| ... |(-1)|... |(-0.1)|...
```

```
data Text is ("a")|("b")| ... | ("aa") | ("ab") | ...

data Date is Number
```

3.1.2 Systems

Here we define the systems expressed in the specification, using data types.

There are several rovers, which are indexed using a number:

```
data Rover is
  (Rover (index:Number))
```

There are several gps stations:

```
data GPSStation is
  (GPS Station (index:Number))
```

There is only one supervision station:

```
data SupervisionStation is
  (Supervision station)
```

What we call a `System` is anything which falls in the 3 categories we just described:

```
data System is
    SupervisionStation
  | Rover
  | GPSStation
```

We could also have written all of the above in one go, this way:

```
data System is
    (Supervision station)
  | (Rover (index:Number))
  | (GPS station (index:Number))
```

Each data type described in LIDL is associated with several interactions. Examples of interactions associated with the `System` data type:

- `(Supervision station)`
- `(Rover (3))`
- `(GPS station (2))`

3.1.3 Alerts

Here, we define the different categories of alerts, using data types

3.1.3.1 Alerts on rovers

There are several categories of alerts concerning rovers

```
data AlertOnRover is
    (GPS Loss)
  | (Data Loss)
  | (Low Battery)
  | (Critical Battery)
  | ((which:Wheel) is Blocked)
```

The last option `(... is Blocked)` refers to a data type called `Wheel` which describes the different types of wheel that a rover has. The rovers have two wheels:

```
data Wheel is
    (Left wheel)
  | (Right wheel)
```

Examples of interactions that are automatically defined by LIDL when the data types above are defined:

- Constructor (Used to create a data flow with a constant value)
  - `(GPS Loss)` `:AlertOnRover out`
  - `((Left wheel) is Blocked)` `:AlertOnRover out`
- Matcher (Used in the switch/case interaction, allows some kind of pattern matching/comparison)
  - `(GPS Loss)` `:{value: AlertOnRover in, didMatch: Boolean out}`
  - `(any AlertOnRover)` `:{value: AlertOnRover in, didMatch: Boolean out}`
  - `((Left wheel) is Blocked)` `:{value: AlertOnRover in, didMatch: Boolean out}`
  - `((any Wheel) is Blocked)` `:{value: AlertOnRover in, didMatch: Boolean out}`
- Mutation (To modify an instance of a type), here they allow to modify instances of alarts…
  - `(do not change)` `:{before: AlertOnRover in, after: AlertOnRover out}`
  - `(change to GPS Loss)` `:{before: AlertOnRover in, after: AlertOnRover out}`
  - `((change to Right wheel)is Blocked)` `:{before: AlertOnRover in, after: AlertOnRover out}`

The automatical definition of these interactions is formally defined, and will be explicited in the appendix of Vincent Lecrubier's Thesis.

3.1.3.2 Alerts on GPS Stations

There are 3 kinds of alerts on GPS stations:

```
data AlertOnGPSStation is
    (GPS Loss)
  | (Data Loss)
  | (Power Loss)
```

Examples of interactions:

- `(GPS Loss)`
- `(Power Loss)`

3.1.3.3 Alerts on supervision stations

There is only 1 kind of alert on the supervision station

```
data AlertOnSupervisionStation is
    (Power Loss)
```

Examples of interactions:

- `(Power Loss)`

3.1.3.4 General case alert

To specify an alert in the general case, we need to specify the *alert* and the *system on which the alert happened*. There a three categories of alerts, one for each category of system:

```
data Alert is
    ((alert:AlertOnRover) on (rover:Rover))
  | ((alert:AlertOnGPSStation) on (gpsStation:GPSStation))
  | ((alert:AlertOnSupervisionStation) on supervision station)
```

Examples of interactions that LIDL automatically associates with this data type:

- `((Low Battery) on (Rover (1)))`
- `((Data Loss) on (GPS Station (2)))`
- `((Power Loss) on supervision station)`

3.1.3.5 Alert occurence

An alert occurence is the product of an *alert* and of a *date*. It is an alert associated with a specific date:

```
data AlertOccurence is
    (Alert (category:Alert) at (date:Date))
```

Examples of interactions:

- `(((Low Battery) on (Rover (1))) at (2016-06-01T06:00:00Z))`

3.1.3.6 Alert levels

There are three alert levels:

```
data AlertLevel is
    (Caution)
  | (Warning)
  | (Advisory)
```

We can write an interaction that associates each alert with its specific alert level:

```
interaction (level of (alert:Alert in) is (level:AlertLevel out))   : Behaviour
  ( switch  (alert)                                                 : (level)
    case    ((GPS Loss) on (Rover (1)))                             : (Caution)
    case    ((GPS Loss) on (Rover (any Number)))                    : (Warning)
    case    ((GPS Loss) on (GPS Station (any Number)))              : (Caution)
    case    ((Power Loss) on supervision station)                   : (Caution)
    case    ((Low Battery) on (any Rover))                          : (Advisory)
    case    (((any Wheel) is blocked) on (any Rover))               : (Advisory)
    case    (any AlertOnGPSStation)                                 : (Warning)
    case    (any AlertOnSupervisionStation)                         : (Caution)
  )
```

Pattern matching using Comparator interactions allows us to be very specific about alert levels. When several cases match, the case which is written higher in the list is selected. For example here, `((GPS Loss) on (Rover (1)))` will be associated with `(Caution)` even though it also matches `((GPS Loss) on (Rover (any Number)))`.

Once this behaviour (that associates a level with each alert) is specified, we can use the `... with behaviour ...` interaction in order to *syntactically project* the behaviour along two prefered directions. This can be used later in order to facilitate the expression of certain behaviours. We can define the following interactions:

```
interaction (level of (alert:Alert in)):AlertLevel out is
  (
    (x) with behaviour
    (level of (alert) is (x))
  )

interaction (an alarm with level (level:AlertLevel out)):Alert in is
  (
    (x) with behaviour
    (level of (x) is (level))
  )
```

Examples of interactions that can be expressed using the definitions above:

- `(level of ((Low Battery)on (Rover (1))))` which gives the value `(Advisory)`
- `(level of ((Power Loss)on supervision station))` which gives the value `(Caution)`
- `((an alarm with level (x))= (my alert))` which means that `(x)` will receive the level of `(my alert)`

3.1.3.7 Alert autoclear and autoack

The same way we defined alert levels, we can express which alert have auto ack and auto clear

```
interaction ((alert:Alert in) has automatic ack (autoAck:Boolean out) and auto clear (autoClr:Boolean out))
      : Behaviour
  ( switch  (alert)                                    : ((autoAck),(autoClr))
    case    ((GPS Loss) on (Rover (any Number)))       : ((true),(true))
    case    ((GPS Loss) on (GPS Station (any Number))) : ((true),(true))
    case    ((Power Loss) on supervision station)      : ((true),(false))
    case    (any AlertOnGPSStation)                    : ((false),(false))
    case    (any AlertOnSupervisionStation)            : ((true),(false))
    case    (any AlertOnRover)                         : ((true),(false))
  )

interaction ((alert:Alert in) has auto ack):Boolean out is
  (
    (x) with behaviour
    ((alert) has automatic ack (x) and auto clear (y))
  )

interaction ((alert:Alert in) has auto clear):Boolean out is
  (
    (y) with behaviour
    ((alert) has automatic ack (x) and auto clear (y))
  )
```

Examples of interactions that can be expressed using the definitions above:

- `(((Low Battery) on (Rover (1))) has auto clear)` which gives the value `(false)` (Last case of the switch interaction)
- `(((GPS Loss) on (Rover (12))) has auto ack)` which gives the value `(true)` (First case of the switch interaction)

3.1.4 Alert signals

An alert signal is a composite data that contains a message, an alert level, a date, a state, and a procedure. This is an abstraction of what is presented to the user.

```
data AlertSignal is
  ( AlertSignal (message:Text) (level:AlertLevel)
    (date:Date) (state:AlertState) (procedure:Procedure) )
```

The alert signal can be in several states which are explicited in various places of the textual specification:

```
data AlertState is
  (active) | (acknowledged) | (cleared)
```

3.1.5 Procedure

A procedure is associated with each alert signal. A procedure can be in several states, not started, completed, or at a specific state:

```
data ProcedureState is
  (not yet started) | (completed) | (at step (index:Number))
```

There are two kinds of steps in a procedure: imperative steps (example : "Do this"->"Done"), or conditional steps (example: "Is it ok ?"->"Yes/No")

```
data ProcedureStep is
    (imperative  (message:Text))
  | (conditional (message:Text))
```

A procedure is a sequence of steps, in a certain state:

```
data Procedure is
   (procedure (steps:[ProcedureStep]) (state:ProcedureState))
 | (no procedure)
```

Example of interactions that can be written using the definitions above:

- Construction (represent a dataflow denoting a procedure)
  - `(procedure ([ (imperative "(Do "this)) (imperative "(Do "that)) (conditional "(Is that done "?) )]) (at step (2)))`
  - `(no procedure)`
- Mutation (modifies a procedure)
  - `(procedure (do not change)(at step (increment)))` increments the step at which the procedure is
  - `(procedure (change element (2)by (imperative ("Do these")))(do not change))` changes the second step of a procedure by replacing it with an imperative step whose message is "Do these", but no not change its state.
- Matcher (matches certain procedures)
  - `(procedure (any [ProcedureStep])(at step (any Number)))` matches any procedure which is started and not completed
  - `(procedure (any [ProcedureStep])(completed))` matches any procedure which is completed

The user can perform 3 actions on a step of a procedure

```
data Action is
  (yes) | (no) | (done)
```

3.2 Interfaces

Here we read the specification and express the Interfaces using LIDL.

3.2.1 Alert signal presentation

```
interface AlertSignalPresentation is
  {
    signal       :  AlertSignal out,
    ack          :  Activation in
    clr          :  Activation in
    launchproc   :  Activation in
    cancelproc   :  Activation in
  }



interface AlertSignalPresentationList is
   AlertSignalPresentation[5]

interface AlertSignalPresentationList is
   {    1: AlertSignalPresentation,
         2: AlertSignalPresentation,
         3: AlertSignalPresentation,
         4: AlertSignalPresentation,
         5: AlertSignalPresentation
   }



interface SignalPresentationAutreMethode is
   {     signal :  [AlertSignal] out,
         ack    :  [Activation] in
   }
```

3.2.2 Procedure presentation

```
interface ProcedureStepPresentation is
  {
     message    : Text out,
     procAction : Action in
  }
```

Interpretation 1 (Bonne):

```
interface ProcedurePresentation is
  {
    currentStep   : ProcedureStepPresentation,
    steps         : ProcedureStepPresentation[MAX_STEP]
  }
```

Interpretation 2, prends la specification vraiment au pied de la lettre:

```
interface ProcedurePresentation is
  {
    stepminus2   : ProcedureStepPresentation,
```

```
    stepminus1   : ProcedureStepPresentation,
    currentStep  : ProcedureStepPresentation,
    stepplus1    : ProcedureStepPresentation,
    stepplus2    : ProcedureStepPresentation
  }
```

3.2.3 User interface

```
interface User is
  {
     alertSignals                          : AlertSignalPresentationList
     currentProcedure                      : ProcedurePresentation
     indicationOfOtherPendingAlertSignals : Boolean out
     numberOfOtherPendingAlertSignals      : Number out
     setTransmissible                      : Number in
  }
```

3.3 Interactions

The specification expresses several Interactions. Here we describe them using LIDL.

TO BE DONE

```
interaction
  (present alert signals ):AlertSignalPresentationList
is
  ()
```