**SCENARIO:**

In order to edit a scenario using the textual editor, the user shall type the sequence, and add content between the brackets:

**scenario {**

**…**

**}**

**PARTICIPANTS: (from PlantUML, adapted to Capella)**

The following keyword are supported in to insert participants in the diagram using the textual editor:

**• actor**

**• entity**

**• role**

**• component**

**• configuration\_item**

**• function**

**• activity**

**actor “actor\_name”**

**function “function\_name”**

**…**

**Features:**

* content assist when typing the participant keyword
* content assist when typing the available participants to insert
* validation of participants (ex: check that the represented part exist before being able to use the participant name in diagram)

**Limitations:**

* cannot use duplicated participants, the participants are uniquely identified by the keyword of the participant and the name
* the participants are inserted; in the first version the participants are not created.

**MESSAGES:**

**SEQUENCE MESSAGE:**

* **participant1 -> participant2 : “exchange\_name”**

**CREATE MESSAGE: (from PlantUML)**

* **participant1 ->+ participant2 : “exchange\_name”**

**DELETE MESSAGE: (from PlantUML)**

* **participant1 ->x participant2 : “exchange\_name”**

**Features:**

* Content assist when proposing the source, target and available exchange
* Validation of the possibility of using the participants and exchanges in different contexts

**Limitations:**

* Cannot use duplicated messages, the sequence messages are uniquely identified by the kind of the message, the source and target and the exchange name
* The exchanges use in text are the ones available to insert, in the first version, the

user does not have the possibility of creating new exchanges

In Capella, we always activate an execution after each sequence message 🡪 each sequence message implies an activation. To mark that a message ends the execution later, we can use the **“withExecution”** keyword**:**

**ACTIVATE EXECUTION:**

* **participant1 -> participant2 withExecution: “exchange \_name”**

**DEACTIVATE EXECUTION:**

* **deactivate participant2**

**RETURN:**

* **participant1 -> participant2 withReturn: “exchange \_name”**

**Observation:** the **“withExecution”** and **“withReturn”** keywords can be used in the same time**;**

**example:**

* **“participant1” -> “participant2” withReturn withExecution: “exchange\_12”**

**deactivate “participant2”**

**Limitations:**

* If we have multiple messages on the same time, but even on different timelines, the text editor will display them sequentially.

**ARM TIMER: (only in Capella)**

* **->> participant : “Arm timer”**

**Features**:

* Supports all the other features as for sequence messages, no return branch.

**COMBINED FRAGMENT**

**ALT: (from PlantUML)**

**alt “condition A” over timelime1, timeline2,… {**

**[something]**

**} else “condition B” {**

**[something\_else]**

**} else “condition C” {**

**[something\_else]**

**}**

**PAR : (from PlantUML)**

**par “condition A” over timelime1, timeline2,… {**

**[something]**

**} “condition B” {**

**[something\_else]**

**} “condition C” {**

**[something\_else]**

**}**

**LOOP, ASSERT, CONSIDER, CRITICAL, IGNORE, NEG, OPT, SEQ, STRICT, UNSET**

Similar to **PAR**

**Features:**

* Content assist on the combined fragment syntax and on the elements that can be used on the combined fragment and inside each operand
* Validation on using the combined fragment

**Limitations:**

* Cannot use duplicated combined fragments (uniquely identified by the kind of combined fragment and operands expressions)
* The operand expression cannot be empty in the first version

**STATE: (only in Capella)**

* **on “Timeline” state “State\_name”**

**ALLOCATED FUNCTION: (only in Capella)**

* **on “Timeline” function “State\_name”**

**REF, DURATION, SHARED DATA, EVENT…**

[Unsupported]

**EXAMPLES**

**FUNCTIONAL\_SCENARIO**

****

**scenario {**

**actor “Launch image acquisition”**

**actor “Acquire images”**

**actor “Elaborate current situation”**

**actor “Forecast weather”**

**“Launch image acquisition” -> “Acquire images” : “image request”**

**“Acquire images” -> “Elaborate current situation” : “collected images”**

**“Elaborate current situation” -> “Forecast weather”: “Forecast weather”**

**}**

**INTERFACE SCENARIO**

****

**scenario {**

**actor “Measurement Engineer”**

**actor “EOLE”**

**actor “Atmosphere”**

**component “Repository”**

**“Measurement Engineer” ->+ “EOLE” : “start” // create Message**

**loop “i: 1:3” over “Measurement Engineer” “EOLE” {**

**“Measurement Engineer”-> “EOLE” withExecution withReturn: “autotest”**

**on “EOLE” state “Self-test”**

**deactivate “EOLE”**

**}**

**on “EOLE” state “Operational”**

**“Measurement Engineer”-> “EOLE” withExecution: “getImage(timeRequest)”**

**“EOLE” -> “Atmosphere” withExecution withReturn: “aquireImage”**

**on“EOLE” state “Image Acquisition”**

**deactivate “Atmosphere”**

**“EOLE” -> “Repository” : “WRITE”**

**deactivate “EOLE”**

**deactivate “Repository”**

**“Measurement Engineer” -> “EOLE” withReturn : “readImage”**

**“Measurement Engineer” ->x “EOLE” : “stop” // delete message**

**}**

**EXCHANGE SCENARIO**

****

**scenario {**

**actor “Measurement Engineer”**

**actor “EOLE”**

**actor “Atmosphere”**

**on “EOLE” state “Operational”**

**“Measurement Engineer” -> “EOLE” withExecution : “image request”**

**“Atmosphere” -> “EOLE” : “weather”**

**on “EOLE” state “Image acquisition”**

**on “EOLE” state “acquire images”**

**deactivate “EOLE”**

**on “EOLE” state “Batch Proccesing”**

**on “EOLE” state “Elaborate current situation”**

**}**

**OPERATIONAL E SCENARIO**

****

**scenario {**

**actor “Atmosphere”**

**actor “Measurement Engineer”**

**actor “Forecaster”**

**actor “Broadcaster”**

**actor “Scientific user”**

**“Atmosphere” -> “Measurement Engineer” withExecution : “weather”**

**on “Measurement Engineer” state “Collect meteo data”**

**deactivate “Measurement Engineer”**

**->> “Measurement Engineer” : “Collected meteo data”**

**}**

**OPERATIONAL ACTIVITY SCENARIO**

****

**scenario {**

**actor “Make weather”**

**actor “Collect meteo data”**

**actor “Elaborate…”**

**actor “Forecast ..”**

**actor “Publish .. “**

**actor “Consult..”**

**“Make weather” -> “Collect meteo data” : “wheater”**

**“Collect meteo data” -> “Elaborate…” withExecution withReturn: “collected meteo data”**

**alt “full” over “Elaborate…” “Forecast ..” {**

**“Elaborate…” -> “Forecast ..” : “current situation”**

**}**

**else “else” {**

**“Elaborate…” -> “Forecast ..” : “specific measure”**

**}**

**deactivate “Elaborate…”**

**“Forecast ..” -> “Publish .. “ : “weather forecast”**

**}**