Contents

6.9. UTM Directives Framework Implementation on UAS

Summary: The standard framework implementation (sec. **??**) needs to be enhanced for UTM directives following. The rule engine software architecture supports the addition and removal of rules and regulations .

Introduction: This section is follow up of *UTM functionality definition* (sec. **??**), outlining realization of *UTM directives* on *UAS* side (sec. 6.9.1, 6.9.2).

Reasoning: The *Avoidance* process and *UTM directives fulfillment* are different in every national airspace. The ICAO issues recommendation [1, 2] which are implemented by every member country, some of the procedures are stricter some are implemented differently.

The *UTM* collision case calculation and procedures may be universal, but their realization by *UAS* will be heavily impacted by local legislation and procedures. The *approach* must account the need for *variable parts* of *obstacle avoidance process*. The *dynamic parts* need to be woven to hard-coded processes.

Note. Please refer to *Template Programming* and *Aspect Oriented Programming* for further explanation.

Inspiration: There was a *Maritime Rules* implementation [3] in the form of *Movement Restrictions* and *Waypoint Changes*.

6.9.1. Rule Engine Architecture

Summary: The implementation of the rule engine architecture in our framework environment.

Purpose: The *core process* of *Avoidance Grid Run* (sec. **??**) and *Mission Control Run* (sec. **??**) needs to be enhanced based on the situation. The architecture is based on *aspect-oriented approach* [4]. The key ideas and concepts are taken from rule engine implementation for multiagent navigation system [5].

Rule Engine: The program module to inject and run *rules* modifying standard work-flow based on triggering events. The *aspect-oriented* approach enables to configure rules in *run-time* via predefined process hooks - *Decision Points*.
A rules the in context of this work are pieces of code which have a semi-static structure consisting of following parts (fig. 6.1):

1. *Decision Point* - hook point in the process where the rule can be attached/detached. If more than one rule is hooked the priority of execution needs to be defined.

2. *Context* - the *run time context* in a time of *invocation* in our case the *copy* of *Mission Control*, *Avoidance/Navigation Grid* and, *Collision Cases*.

3. *Parser Method* - optional helper method to parse interesting data set from *Context*. The *parsed data* have better readability.

4. *Condition Check Method* - implementation of the trigger. If the sufficient condition is met, the rule body is applied.

5. *Rule application* - calculations and data structure changes. Mainly, by *disabling trajectories* in *Reach Set* in our implementation.

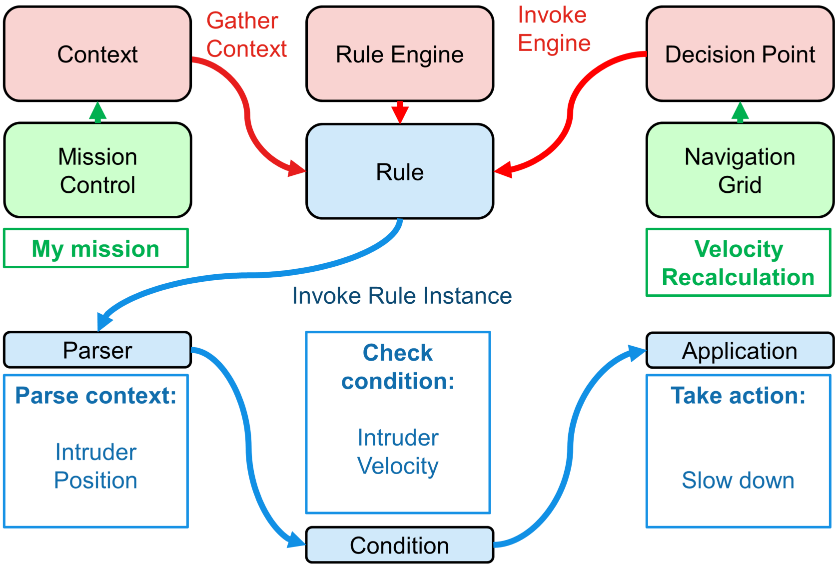

Figure 6.1: Rule engine components overview.

Example: The *UAS* is flying in controlled airspace. The *intruder* shows in front of *UAS*. The *UAS* is faster than an *intruder*. The *UAS* tries to obtain permission for *Overtake*. The *UTM* does not allow *overtake*, because of *insufficient UAS maneuverability capability*. The *Rule* (fig. 6.1) with:

1. *Context* - UAS Mission Control, containing the actual mission goal and UAS IMU parameters.

2. *Decision Point* (Joint Point) - Navigation grid, containing projected constraints and reach set approximation.

3. *The rule is invoked:*

    a. *The parser* parses the context which is *intruder‘s Position Notification* containing its heading and velocity.

    b. *The condition* is checked to *relative intruder velocity*. The *evaluation* is positiv, when the UAS is *pursuing the intruder*.

    c. *Application* of *Rule* is the last step, in this case, the *UAS* will slow down.

Configurability: The *Rule Engine* enables real-time configuration. The *Enabled Rules Table* have been implemented to enforce specific rules in a specific context.

The *Rules* can be invoked from *Rule Application*; this enables effective rule chaining and piece-wise functionality split.

6.9.2. Rule Engine Setup

Summary: The setup to cover collision case resolution according to (sec. **??**).

Configuration: The *Rule Engine Architecture* (fig. 6.1) is configured to handle *UTM functionality* for *Collision Case Resolution* (sec. **??**). The overview of *Context* (Green), *Decision Points* (red) and *Rules to be Invoked* (cyan) is given in (fig. 6.2).

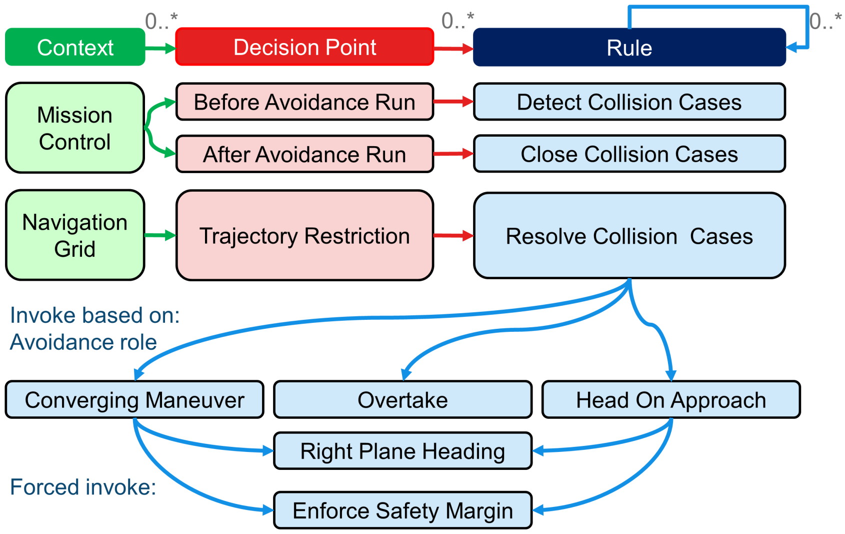

Figure 6.2: Rule engine initialization with Rules of the air.

Decision Points: The *Decisions* are bounded to *Mission Control Run Process* (fig. **??**) in the following manner:

1. *Before Avoidance Run* (before step 7.) - Context: *Mission Control* (Received Collision Cases) - the *UTM* can send directives. It is required to find which ones are impacting our *UAS*.
2. *Trajectory Restrictions* (after step 7.) - Context: *Navigation Grid* (Trajectory Restrictions) - an adaptation of *behavior* imposed by *active collision cases*.
3. *After Avoidance Run* (after step 11.) - Context: *Mission Control* (Collision Case Resolutions) - our *UAS* will update the status of *Collision Cases* then it checks the *avoidance conditions*. The *Resolution Notification* resolution notifications are sent to UTM afterward.

Note. The *Weather Case* (app. **??**) is handled similarly. The mission control loop (fig. **??**) have rules with separate *Decision Points* to enforce *hard constraints* (before step 9.) and *soft constraints* (before step 10.).

Road map: The *implemented rules*(cyan) are separated into the following categories:

1. *Management Rules* - managing collision cases (additional control flow):

   a. *Detect Collision Cases* (app. **??**) - the detection of active participation in received *collision cases* and generation of *restrictions*.

   b. *Resolve Collision Cases* (app. **??**) - the enforcement of *active avoidance roles* in *collision cases*. The one *Restriction Rule* is invoked directly.

   c. *Close Collision Cases* (app. **??**) - impact calculation and *Resolution Notification* to *UTM* authority.

2. *Restriction Rules* - restricting the *Navigation Grid* trajectories or altering *goal waypoint* based on *selected collision cases*:

   a. *Converging Maneuver* (app. **??**) implementation of *Converging Avoidance* (sec. **??**).

   b. *Head On Approach* (app. **??**) implementation of *Virtual Roundabout Enforcement* (sec. **??**).

   c. *Overtake* (app. **??**) implementation of *overtaking maneuver* for *Overtaking plane* (sec. **??**).

3. *Miscellaneous Rules* - reused pieces of code in *Head-On* and *Converging Situations*:

   a. *Right Plane Heading* (app. **??**) - restrict all trajectories heading to space separated by parametric plane in *Avoidance Grid* which is heading or belonging to plane.

   b. *Enforce Safety Margin* (app. **??**) - restrict all *Trajectories Segments* which are in proximity of *Collision Point* lesser than *Enforced Safety Margin*.

Bibliography

[1] ICAO. 4444: Procedures for air navigation services. Technical report, ICAO, 2018.

[2] ICAO. Annex 2 (rules of the air). Technical report, ICAO, 2018.

[3] Michael R Benjamin, Joseph A Curcio, John J Leonard, and Paul M Newman. Navigation of unmanned marine vehicles in accordance with the rules of the road. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 3581–3587. IEEE, 2006.

[4] Ernest Friedman Hill. *Jess in action: Java rule-based systems*. Manning Publications Co., 2003.

[5] Georg S Seyboth, Dimos V Dimarogonas, and Karl H Johansson. Event-based broadcasting for multi-agent average consensus. *Automatica*, 49(1):245–252, 2013.