A Family of Domain-Specific Languages for Specifying Civilian Missions of Multi-Robot Systems

Davide Di Ruscio[1], Ivano Malavolta[2], and Patrizio Pelliccione[3]

[1]Department of Information Engineering Computer Science and Mathematics
University of L'Aquila (Italy)
[2]Gran Sasso Science Institute, L'Aquila (Italy)
[3]Department of Computer Science and Engineering
Chalmers University of Technology |University of Gothenburg (Sweden)
davide.diruscio@univaq.it,ivano.malavolta@gssi.infn.it,patrizio.pelliccione@gu.se

Abstract. The next future will be pervaded by robots performing a variety of tasks (e.g., environmental monitoring, patrolling large public areas for security assurance). So far, researchers and practitioners are mainly focusing on hardware/software solutions for specialized and complex tasks; however, despite the accuracy and the advanced capabilities of current solutions, this trend leads to task-specific solutions, difficult to be reused and combined.
In this paper we propose a family of domain-specific languages for specifying missions of multi-robot systems by means of *models* that are (i) independent from the technologies, (ii) ready to be analysed, simulated, and executed, (iii) extensible to new application areas, and (iv) closer to the problem domain, thus democratizing the use of robots to non-technical operators. We show the applicability of the proposed family of languages in a real project in the domain of autonomous unmanned aerial vehicles.

1 Introduction

The next future will be pervaded by robots that, moving underwater, on terrain, or flying, will simplify everyday tasks or will open a myriad of new opportunities. Multi-robot systems will behave as a team, in which each single robot accomplishes a well defined task towards the accomplishment of a global mission. On one side a multi-robot team can accomplish a mission more quickly than a single robot, and on the other side a multi-robot team can accomplish missions requiring particular capabilities that might be impossible or impractical to find on a single robot: a team can make effective use of specialists designed for a single purpose, e.g., scouting an area or picking up objects.

The specification of a mission is difficult when considering a single robot since many details should be taken into account, and it might require technical expertise about the dynamics and the characteristics of the used robot. It becomes even more complex when dealing with missions involving multi-robots. Then, it emerges the need of software engineering approaches and methodologies especially tailored to develop and maintain multi-robot systems.

Model-driven Engineering (MDE) [1] is a promising research field for simplifying the design, implementation and execution of software systems for the robotics platforms of the future. In MDE, we can notice a shift from third generation programming language code to models expressed in domain-specific modeling languages (DSMLs). In

this context, MDE enables the development of multi-robot systems by means of models defined with concepts that are much less bound to the underlying technology and are closer to the problem domain. This makes the models easier to specify, understand, and maintain, helping the understanding of complex problems and their potential solutions through abstractions [2].

In this paper we present a family of domain-specific languages for specifying civilian missions of multi-robot systems. The proposed languages are organized in different layers going from languages conceived for the end user, namely those to describe missions and the environmental context, intermediate language describing the detailed behaviour of each robot (hidden to the user), and the robot language containing the hardware and low-level specification of each type of robot within the team. In order to show the applicability of the proposed languages in practice, we instantiate it to the domain of civilian missions of autonomous quadrotors. The resulting platform is called FLYAQ [3] and it provides to on.site operators a graphical interface enabling the specification of missions at a high-level of abstraction.

Roadmap of the paper. The remainder of the paper is structured as follows: a description of civilian missions is provided in Section 2. The architecture of the proposed family of languages is presented in Section 3, while their instantiation and implementation to manage swarms of autonomous quadrotors is provided in Section 4. Section 5 discusses the related work, whereas Section 6 concludes the paper and outlines some perspective work.

2 Civilian missions

Several civilian missions have been discussed in the literature. Skrzypietz [4] subdivides civilian missions for Unmanned Aircraft Systems (UAS) in six categories:

▷ *Scientific Research*, such as atmospheric, geological research, forestry.

▷ *Disaster Prevention and Management*, like damage assessment after earthquakes, searching for survivors after airplane accidents and disasters.

▷ *Homeland Security*, such as coastal surveillance, securing large public events.

▷ *Protection of Critical Infrastructure*, such as monitoring oil and gas pipelines, protecting maritime transportation from piracy, observing traffic flows.

▷ *Communications*, like broadband communication, telecommunication relays.

▷ *Environmental Protection*, such as pollution emission, protection of water resources.

According to Washington Post[1], venture investors in the United States poured $40.9 million into drone-related start-ups in the first nine months of 2013, more than double the amount for all of 2012. Moreover, according to the "Unmanned Aerial Vehicle (UAV) Market (2013 - 2018)" market research, the total global UAV Market (2013-2018) is expected to reach $8,351.1 million by 2018. [2]. This is justified by the number of advantages that the use of these devices brings: (i) *costs*: civilian missions typically requires high costs for personnel which have to be carried on site, and to the communication overhead required for synchronization purposes of the teams; (ii) *safety*: on-site personnel may be exposed to significant risks (e.g., in case of fire, earthquake, and

[1] http://wapo.st/1bJueLH

[2] http://www.marketsandmarkets.com/Market-Reports/unmanned-aerial-vehicles-uav-market-662.html

flood); (iii) *timing and endurance*: monitoring activities are very time consuming. Also, the activities are stopped during the night, slowing the execution of the mission.

3 The Family of Languages

The family of domain-specific modeling languages that we propose supports the specification of missions and their actual execution by means of swarms of robots. The developed languages stack is shown in Fig. 1.

MML, namely the *Monitoring Mission Language*, is the language especially conceived for domain experts. The MML language is composed of two distinct layers, that are: the mission layer and the context layer. The former enables the specification of civilian missions without referring to specific aspects like the technical characteristics of the robots that will be used to execute the missions; missions are specified as sequences of tasks, suitably linked together via task dependencies, forks and joins. The specification of a mission is complemented by the definition of the context in which the mission will be realized. Elements of the context can be modeled by means of the *Mission Context Language*.

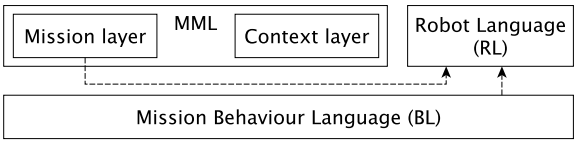


Fig. 1. The family of domain-specific modeling languages

The type and configuration of the robots that will be in charge of realizing the specified mission are described through the *Robot Language* (RL). The *Behaviour Language* (BL) is a language that is hidden to domain experts. It contains a specification of the atomic movements and actions of the robots being considered in the mission. As shown in Fig. 1, both the monitoring mission language and the mission behaviour language have a reference to the robot language (currently those references are implicitly established by the name of the referenced robot). This is needed because: (i) the mission layer of the monitoring mission language contains a list of all the robots of the swarm, and the type of each of them must be specified, and (ii) the mission behaviour language contains all the low-level movements and actions of the robots, whose type must be known in the model in order to correctly instruct the robots at run-time.

It is important to note two important aspects of the family of languages. Firstly, the two layers of the MML language are kept separated in order to allow mission operator to reuse already existing context models across missions and different organizations. Also, the context layer puts restrictions on the mission layer since they share the same location, thus enabling for a straightforward (automatic) composition of the two. Secondly, the family of languages has been designed to support the automatic generation of BL models from MML models; it enables to (i) obtain BL models which are inherently consistent to their corresponding MML models, and (ii) to mask the complexity of low-level details about the used robots (and their actions) to on-site operators. In light of this, the control code for the robots depends only on the constructs of the BL language, and thus can be either automatically generated or directly executed by interpreting BL models at run-time.

Three are the stakeholders of the proposed family of languages, namely:

1. Operator: the in-the-field stakeholder specifying the mission as an MML model; examples of operators include fire fighters, policemen, etc.;
2. Robot Engineer: models a specific kind of robot via an RL model, together with a corresponding controller for instructing the robot according to the basic operations supported by the BL language;
3. Platform Extender: domain and MDE expert who extends the MML mission layer with new kinds of tasks supporting a specific application domain (e.g., agricultural missions, security-oriented missions, smart grid monitoring missions, etc.); those extensions are performed once for each application domain being considered, and can be reused across missions and organizations.

Issues related to mission correctness, e.g., safety and security, are fundamental for civilian missions for multi-robot systems. In this context, the proposed languages have been designed to be generic enough for describing this kind of missions from an high-level point of view, and thus mission correctness is not part of the languages themselves. In any case, the proposed languages provide the right level of abstraction for allowing analysis tools to be executed on the models for proving, for example, safety and security properties. We believe that the Behaviour Language is the best candidate for this kind of analyses, since it can be easily transformed into a corresponding state machine, a process algebra, Petri net, etc., thus allowing engineers to reuse already existing analysis tools.

The remainder of this section will describe each of these languages. Specifically, for each of them we present the corresponding metamodel i.e., the abstract syntax of the language. For what concerns their concrete syntax, the MML language can have a graphical syntax like, e.g., an overlay on a geographical map representing the various tasks, dependencies and contextual elements (see Section 4.2 for a concrete example). Differently, the RL and BL languages are represented by using an XML-based representation since they will be created and managed by domain experts or even by other software components.

3.1 Monitoring Mission Language (MML)

Mission Layer - The mission layer of the MML language has been conceived by analyzing the concepts that are involved when specifying monitoring missions. First of all, each robot has its own *home* position represented by means of the corresponding Coordinate element (see Fig. 2). Then, a monitoring mission consists of a number of dependent Tasks to be executed by a Swarm of Robots.

The MML mission layer contains three abstract task metaclasses, each of them focusing on a specific kind of geometric entity being considered. More specifically, PointTask represents tasks that refer to a specific point of the environment. To this end the *point* reference represents the coordinate of the considered point. LineTask represents tasks that refer to a set of points forming a polyline in the environment. Consequently, a line task consists of the *initialPosition* that the considered robot will have at the beginning ot the task, and a set of *points* consisting of the points of the polyline. Also, PolygonTask represents tasks that refer to specific areas. A polygon task consists of the *initialPosition* reference, and the *shell* referecing the points representing the border of the area that will be involved during the execution of the task.

The ControlTask is an abstract metaclass that represents the synchronization tasks of the mission. In particular, each task of the considered mission can be executed by one or more robots and can be performed in sequence (see the metaclass Join) or in parallel (see the metaclass Fork) to other tasks of the modeled mission.

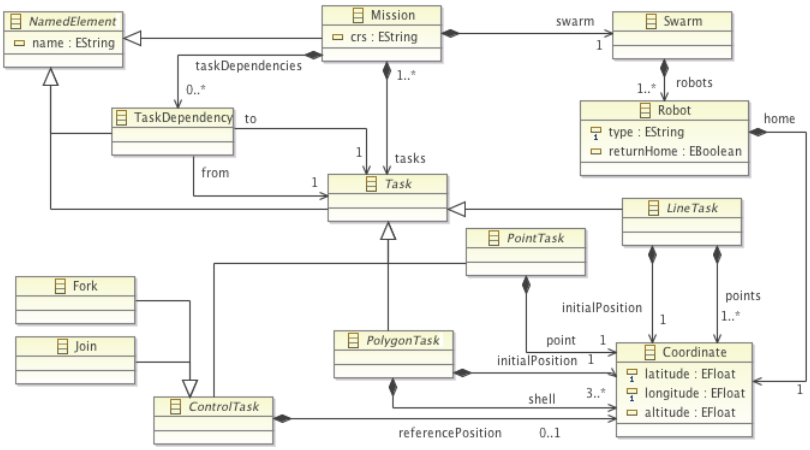


Fig. 2. The Monitoring Mission Language

The mission layer (and so its corresponding metamodel) is defined to be extensible. This means that it specifies only general tasks that need to be specialized according to specific needs, to the actual civilian mission (see Sect. 2 for a description of these missions), and to the robots that will be used. The extensibility of MML allows operators to achieve versatility and strong adherence to the environmental monitoring missions domain. More specifically, MML can be extended with additional constructs that are specifically tailored to the considered domain. For example, if operators are interested to monitoring solar panel installations in a rural environment, MML might be extended with the concept of solar panel groups, thermal image acquisition tasks, and solar panel damage discovery and notification tasks.

Context Layer - As said in the previous section, the specification of monitoring missions includes also the description of the context where they will be executed. By referring to [5], this modeling layer concerns spatial context and situational context. Indeed it represents those portions of geographical areas that have some relevant property, and those elements which can influence the execution of the mission, but that are not part of the mission itself.

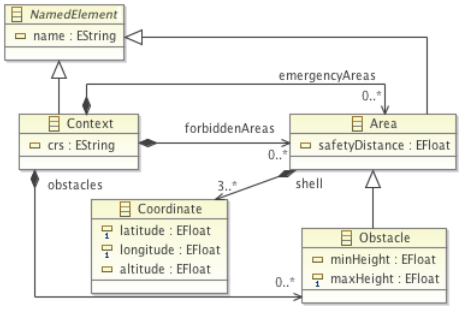


Fig. 3. The Mission Context Language

Fig. 3 shows the metamodel of the context layer of MML. In particular a given monitoring mission will be executed in a Context consisting of a number of *obstacles* and *forbiddedAreas*. Such information will play a key role in order to properly deduce the movements that the robots have to perform in order to execute the missions specified in MML and to satisfy the environmental constraints specified by means of context models.

3.2 Robot Language (RL)

RL has been conceived to enable the specification of the technical characteristics of each type of robot involved in the missions (see Fig. 4). Clearly, Robot is the central concept of the language. The characteristics that the language permits to specify are the following:

- *onBoardObstacleAvoidance*: it permits to specify if the considered robot is endowed with mechanisms able to autonomously avoid obstacles;
- *minVoltage*/*maxVoltage*: they are used to specify the minimal/maximum voltage required/supported by the robot to properly work;
- *maxPowerConsumption:* it is used to specify the maximum power consumption expressed in Watt of the robot being modeled;
- *gps*, *accelerometer*, *magnetometer* and *barometer*: they are boolean attributes used to specify the available on-board sensors of the robot being modeled;
- *communicationRange*: it is used to specify the maximum range expressed in meters of the supported radio control;
- *dataRate*: it permits to specify the data transmission rate expressed in Kbps between the robot and the control station;

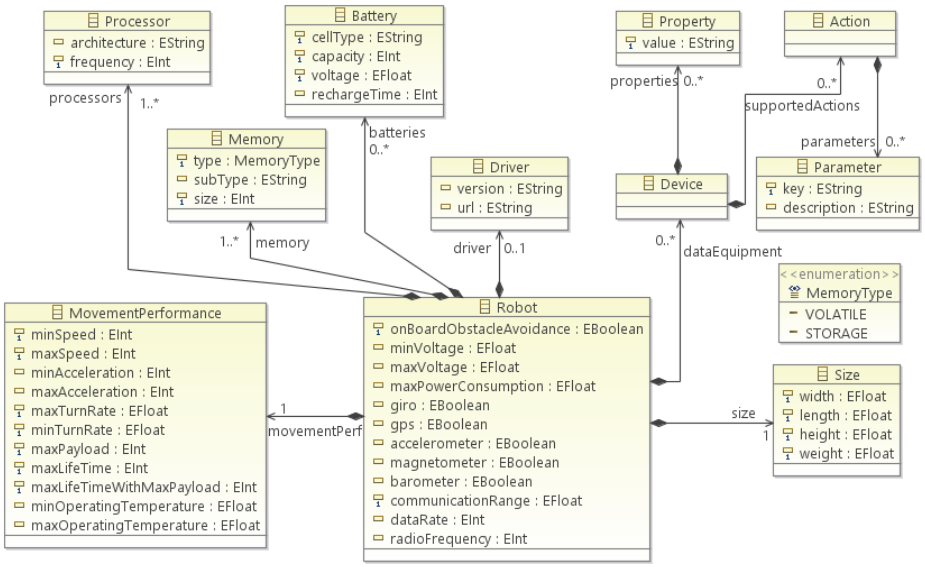


Fig. 4. The Robot Language

– *radioFrequency*: it permits to specify the radio frequency expressed in MHz used by the robot to communicate with the control station.

Furthermore, the Size metaclass is used to specify the size of the robot by means of the attributes such as its *width*, *length*, *height*, and *weight*. The Processor metaclass permits to specify the hardware *architecture* and *frequency* of the processors owned by the robot being modeled. The concept of Memory describes the memory of the considered robot in terms of its type (i.e., if the memory is volatile or permanent), sub-type (e.g., DDR2, DDR3, SSD), and size in kilobytes. Additional devices owned by the robot to gather data (e.g., camera, thermal sensors) and to perform actions (e.g, lights, leds, mechanical actuators, sound emitters) are specified via the Device metaclass. MovementPerformance permits to specify the movement characteristics of the robot (i.e., minimum and maximum speed, acceleration, turn angle). Additionally, it permits to specify the maximum weight of the payload that the robot can bring and consequently also the maximum life time of the robot while bringing a pay load. The minimum and maximum working temperature of the robot can be also specified. Finally, the *Driver* metaclass refers to the software driver, which is required to interact with the robot.

It is important to note that not every robot is specified in RL, but every *type* of robot. This makes RL models reusable and shared across missions, projects, and organizations.

3.3 Behaviour Language (BL)

BL permits to specify atomic movements of each robot in order to perform the missions specified by means of MML specifications. As shown in Fig. 5 a BL model specifies the behaviour of all the robots which will perform the mission and for each of them all the *movements* to be performed are singularly defined. According to Fig. 5 the atomic movements that a robot can perform are the following:

– Start: it represents the first movement used to begin any sequence of movements;
– Stop: it represents the final movement used to end any sequence of movements;
– HeadTo: it represents a rotation in order to head towards the specified *direction*;
– Pause: it permits to specify pauses of robots during their movements;
– Circle: it permits to specify circular movements of robots around a specific point;
– GoTo: it represents the movement towards a given *targetPosition*.

Before executing one move after the end of another one it is possible to specify a transition (see MoveTransition). In particular, if the *fluid* attribute is specified as *true* than the robot will execute the movements seamlessly without any interruption. Alternatively, it is possible to specify a pause (see the metaclass Slot) between two subsequent moves. Additionally, before and after each movement, the robot can perform a number of Actions that can be distinguished as follows:

– DeviceAction: it permits to specify control actions of the robot. To this end the name of the action and the parameters to be used are specified by means of the features *actionName* and *parameters*, respectively;
– CommunicationAction is used to specify the communication among the robots involved in the execution of the considered mission. In this respect, the possible actions that can be performed are the following: (i) CheckNotification: it is used

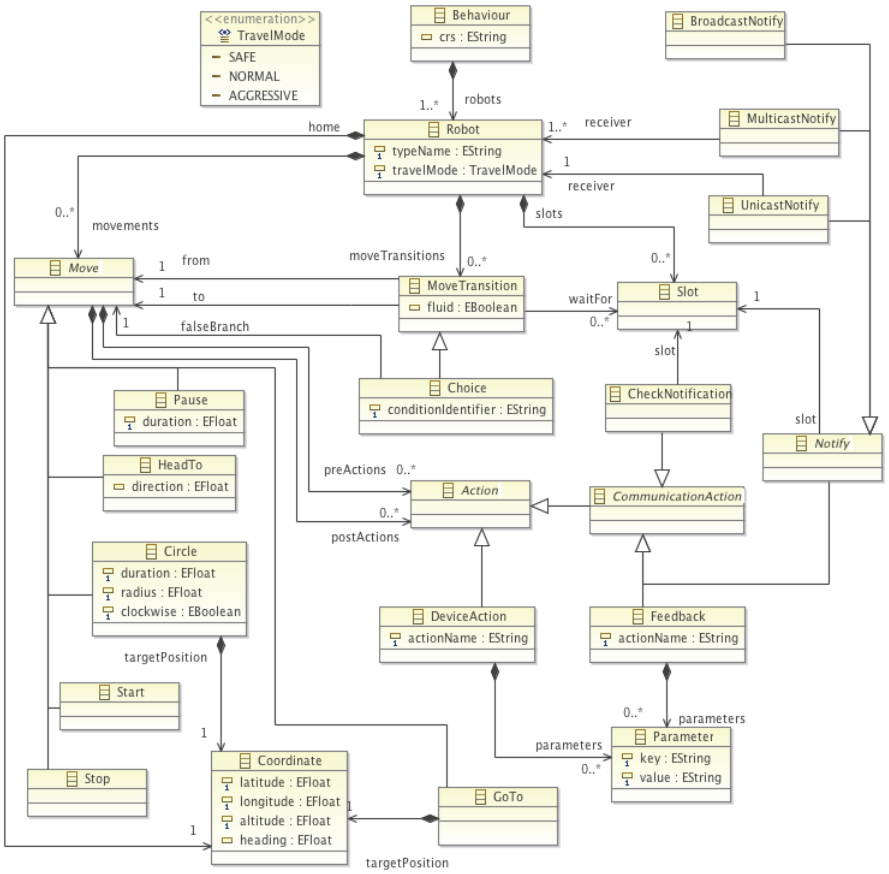
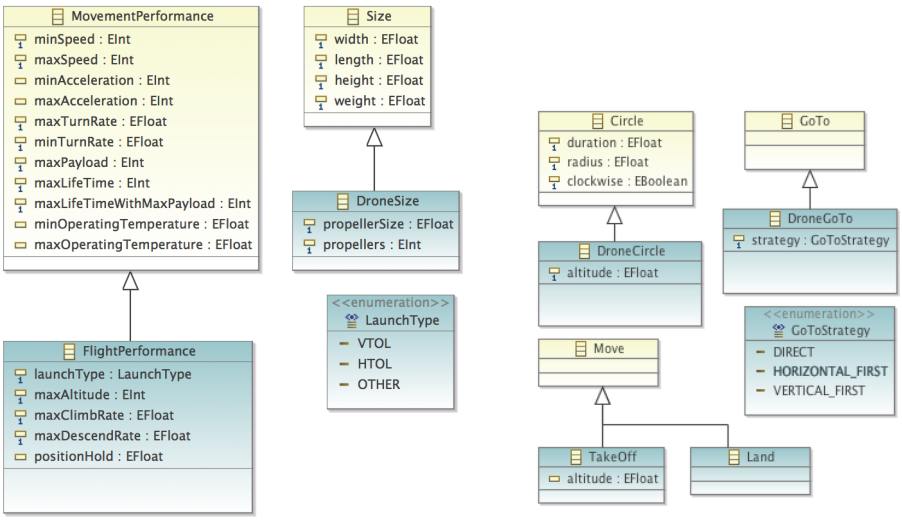


Fig. 5. The Behaviour Language

to manage the reception of notifications from other robots; (ii) Feedback represents a feedback message that the robots can send back to the control station; (iii) Notify is a superclass representing all the possible notification actions that can be distinguished as UnicastNotify, MulticastNotify, and BroadcastNotify.

4 Leveraging the DSL family for autonomous quadrotors

Currently, in collaboration with Telecom Italia we are working on an open-source platform for the specification and the execution of environmental monitoring mission. The platform is called FLYAQ [3] and it allows non-technical operators to straightforwardly define monitoring missions of swarms of flying drones at a high level of abstraction, thus masking the complexity of the low-level and flight dynamics-related information of the drones. More specifically, we employ quadrotors, that are a special kind of unmanned aerial vehicle that takes the form of a multirotor helicopter that is lifted and propelled by four rotors [6]. In this section we present the instantiation of the languages proposed in Section 3 to support the specification of swarms of autonomous quadrotors.



(a) Extension of the robot language (b) Extension of the behaviour language

Fig. 6. Extension of the languages of the DSL family for representing missions of quadrotors

4.1 Extension of the DSL family

In this section we describe how the languages of our DSL family have been extended to support the domain of environmental monitoring missions for autonomous quadrotors. In light of this we evaluated each language of the DSL family, we analysed it in terms of its expressivity with respect to the specific domain in order to check if language concepts fit well with the domain. Interestingly, we did not need to adapt the **Mission** language of the DSL family since it is still a good fit for the new application domain without any extension. Indeed, from an abstract point of view a swarm of autonomous quadrotors can be considered as a swarm of robots performing some task for fulfilling the goal of a global mission. Task may still refer to polygons, polylines, and points within a given mission environment, and tasks may have dependencies and controlled by fork and joins. Even the **Context** language has not been extended since its concepts are still satisfactory in the current state of the FLYAQ project. Indeed, when reasoning about the context of a mission performed by quadrotors the main issues are about: the presence of obstacles (represented by the Obstacle metaclass in the context language metamodel), emergency landing areas (represented by the *emergencyAreas* reference), and no-fly zones where no quadrotor can fly over (represented by the *forbiddenAreas* reference). For what concerns the **Robot** language, we needed to extend it with additional concepts, specific to the nature of the managed robots (i.e., flying quadrotors). Fig. 6(a) shows a fragment of the robot language metamodel focussing on the metaclasses that have been added, they are:

– DroneSize extends the *Size* metaclass of the robot modelling language with two attributes for representing (i) the number of propellers of the drone, and (ii) the size of the propellers of the drone in millimeters. This two additional attributes are necessary since there is a great variability of flying drones with respect to the

number and size of their propellers [6], and flying drones behave very differently depending on those two properties. For example, the popular AscTec Firefly[3] drone has six propellers and, by citing its official data sheet, *the redundant propulsion system enables a controlled flight even with only 5 functioning motors and actively compensates for failure*; it is difficult to imagine a drone with only four propellers providing this peculiar safety function.

– FlightPerformance extends the MovementPerformance metaclass of the mission behaviour language with a set of additional attributes:
  - *launchType* represents the information about how the flying drone can be launched. The value of this attribute can be one among Vertical Take-Off and Landing (*VTOL*), Horizontal Take-Off and Landing (*HTOL*), or any other (*OTHER*);
  - *maxAltitude* represents the maximum altitude that the drone can reach when performing a mission;
  - *maxClimbRate* represents the maximum rate of change in altitude of the drone (in metre per second);
  - *maxDescendRate* represents the maximum rate of change that the drone supports when it is descending (in metre per second);
  - *positionHold* represents the maximum wind speed (in metre per second) supported by the drone to maintain a given geographical location.

The attributes added to the MovementPerformance metaclass are specific to the aerial vehicles domain, and have been necessary for representing specific concepts related to this domain (e.g., *maxAltitude* and *launchType*).

When considering the **Behaviour** language of our DSL family we needed to slightly extend it with some additional concepts, they are shown in Fig. 6(b). Basically, we needed to extend the Circle metaclass for specifying the altitude at which the quadrotor must perform the circle movement. Moreover, we extended the GoTo metaclass for specifying the kind of trajectory that the quadrotor must following when it needs to reach a certain point. The available trajectory kinds represent how the quadrotor can reach a given point in the 3D space, they are three: (i) *DIRECT*, the quadrotor follows a straight line between its current position and the target point; (ii) *HORIZONTAL_FIRST*, firstly the quadrotor moves horizontally until it goes below the target point, and then it adjusts its altitude so that it reaches the target point; (iii) *VERTICAL_FIRST*, firstly the quadrotor moves vertically until it reaches the same altitude of the target point, and then it moves horizontally until it reaches the target. We added the above mentioned concepts to the Behaviour language in order to provide better flexibility when reasoning about the movements of the quadrotors in the environment. For example, the GoTo movement could not be extended at all (e.g., all the quadrotors always move directly to the target point), however we argue that this solution could have been too restrictive for performing real missions in practice.

As a matter of fact, we have been actually quite surprised when noticing that we did not have to extend the Mission and Context modeling languages. This result is positive since those two modeling languages have proven to be expressive enough for the needs

[3] http://www.asctec.de/uav-applications/research/products/asctec-firefly/

of a concrete project. However, when reasoning at a lower level of abstraction some kind of adaptation is needed (e.g., for specifying the propeller size of a quadrotor); still, the extensions performed in the Robot and Behaviour languages are not massive and, most interestingly, they did not disrupted the semantics of the languages being extended.

4.2 Implementation of the extension of the DSL family

In this section we provide a description on how we developed the FLYAQ platform by taking advantage of the languages presented in the previous section. Figure 7 gives an overview of the FLYAQ platform. On-site operators design the mission, store, and monitor the status of ongoing missions via a standard web browser connected with the platform through a secure HTTP connection. This design decision enables operators to (re-)use any kind of device, such as tablets, laptops, etc., which are capable to run standard web browsers. Quadrotors are instructed and controlled by the platform via MAVLink communication so that radio modems can retain control up to eight miles.

More in details, the FLYAQ platform offers a web-based **graphical interface** to specify missions in the ground station at a high-level of abstraction and integrated with Open Street Map. As a matter of fact the web interface of FLYAQ is a domain-specific editor for both the mission (see Section 3.1 and context (see Section 3.1) layers of the the MML language. The graphical interface of FLYAQ is implemented using HTML5, JavaScript, CSS3, and web sockets for real-time communication with the quadrotors (i.e., for getting the telemetry feedback from each quadrotor in the field).

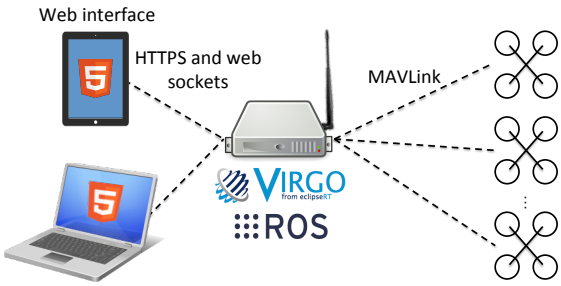


Fig. 7. Overview of the FLYAQ platform

The FLYAQ platform has an internal **engine** that leverages *model transformations* and *formal reasoning* to automatically transform a mission specified using the Mission modeling language into low-level steps conforming to the Behaviour modeling language (see Section 3.3); the transformation step takes also into consideration a model of the context and the models of the drones that will concretely execute the mission in the field.

The FLYAQ internal engine is implemented using Eclipse Virgo[4], the Eclipse Modeling Framework (EMF[5]), Java, and exposes a Rest API to the FLYAQ web interface. So far, the extension of the DSL family languages for adding new concepts related to the UAV domain has been realized manually, i.e., by manually extending the base metamodels of the DSL family in order to obtain the extended metamodels presented in Section 4.1. As a future work we are planning to leverage a more systematic language extension process, with properties such as language independence, possibility to compose and decompose the involved

[4] http://www.eclipse.org/virgo
[5] http://www.eclipse.org/modeling/emf

languages, some level of automation, and so on. The authors have already worked on the topic [7], and the integration of a systematic mechanism for languages extension is currently being evaluated.

Finally, a **layer of controllers** abstracts the types of the specific quadrotors to all the other components of the platform; this is fundamental because it makes the platform totally agnostic of the quadrotors used for executing the mission (abstraction is one of the strongest points of using MDE techniques). The layer managing the controllers is implemented using Java and ROS [8] and its extension rosbridge[6], a middleware communication framework specifically tailored for real-time communication with robots. Each controller can be implemented by using any kind of programming language (thanks to the ROS middleware communication middleware).

5 Related work

Over the last years many research groups have been working on the adoption of model-driven engineering for developing complex robotic systems [9–11]. The advantages of using models for developing this kind of systems are manifold, e.g., higher level abstractions for behaviour descriptions, possibility to apply tools for verifying properties, such as safety, and for generating implementation code.

Typically, the adoption of model-based approaches for developing robot software systems has focused on control or mechanical design aspects. In [9] the authors go further by proposing the adoption of models to manage the complete development of robotic software systems. Similarly, in [10] the authors propose an approach that uses models both at design- and run-time to support robots during their decision making process. In [11] the author proposes an Eclipse based environment for the development of robot control systems, including generation of application code skeleton. In [12] the authors propose a rule-based language for specifying collaborative robot applications. The proposed techniques permit to manage the complexity of specifying collaborative behavior and of managing the communication among robot teams. Concerning the existing work related to the control of quadrotors a very detailed and complete survey on the advances in guidance, navigation, and control of unmanned rotorcrafts systems in general is provided in [13]. Many *algorithms* have been proposed for automatic trajectory generation and control, with a strong focus on either trajectory optimization [14], feasibility [15], or safe obstacle and trajectories intersection avoidance [16].

Differently from the approaches outlined above, our focus is on *i)* the definition of the various tasks of a civilian mission at a high-level of abstraction and *ii)* on the automatic deduction of the behaviour of the robots that will execute the modeled missions. Therefore, the aim of the work that underpins the family of languages presented in this paper is to develop the support for dealing with the specific problem of supporting the specification and execution of civilian missions. In other words, we want to provide non-technical users with the instruments to easily define missions and execute them by means of multitudes of robots. Currently, available technologies somehow permit to develop missions and control the involved robots, even though only software or control engineers and domain experts have the required knowledge and are able to use the

[6] http://robotwebtools.org

complex tools to do so. The proposed languages allow operators to straightforwardly define monitoring missions of swarms of robots by masking all the complexity of the low-level and movement dynamics-related information of the robots.

6 Conclusions and Future work

In this paper we presented a family of domain-specific languages for specifying civilian missions of multi-robot systems. The family of languages is organized in a two-layer architecture, in which the uppermost layer contains languages for the end user, while the other layer, hidden to the user, contains a working language describing the detailed behaviour of each robot of the mission. The family of domain-specific languages has been instantiated to the domain of autonomous unmanned aerial vehicles.

So far, the various tasks in the mission layer are based on their location of interest (i.e., points, lines, and polygons), as future work we are reasoning on how to extend the MML mission layer with timing constraints and other environmental factors, such as path crowdness, convenience (e.g., in terms of used resources), safety, etc. Also, we are planning to experiment the family of domain-specific languages to other kind of robots. This will give us the possibility to further refine the family, and to start experimenting with strategies and mechanisms for combining together different kinds of robots (e.g., ground vehicles with aerial vehicles) that collaboratively perform the same mission. Also, we are analysing the current architecture of the FLYAQ platform in order to better understand how it can be refactored into a fully generic software architecture. In this context, the resulting architecture will enable future third-party software developers and researchers to reuse the generic components of the architecture supporting the core of the DSL family.

References

1. Schmidt, D.: Guest editor's introduction: Model-driven engineering. Computer **39**(2) (Feb 2006) 25–31
2. Selic, B.: The pragmatics of model-driven development. IEEE Softw. **20**(5) (September 2003) 19–25
3. Di Ruscio, D., Malavolta, I., Pelliccione, P.: Engineering a platform for mission planning of autonomous and resilient quadrotors. In: Fifth International Workshop on Software Engineering for Resilient Systems, Springer Berlin Heidelberg (2013) 33–47
4. Skrzypietz, T.: Unmanned Aircraft Systems for Civilian Missions. BIGS policy paper: Brandenburgisches Institut für Gesellschaft und Sicherheit. BIGS (2012)
5. Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D.: A survey of context modelling and reasoning techniques. Pervasive and Mobile Computing **6**(2) (2010) 161 – 180
6. Lim, H., Park, J., Lee, D., Kim, H.J.: Build your own quadrotor: Open-source projects on unmanned aerial vehicles. Robotics Automation Magazine, IEEE **19**(3) (Sept 2012) 33–45
7. Di Ruscio, D., Malavolta, I., Muccini, H., Pelliccione, P., Pierantonio, A.: Developing Next Generation ADLs Through MDE Techniques. In: Procs. ICSE'10, ACM (2010) 85–94
8. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. ICRA workshop on open source software **3**(3.2) (2009) 5

9. Schlegel, C., Hassler, T., Lotz, A., Steck, A.: Robotic software systems: From code-driven to model-driven designs. In: Advanced Robotics, 2009. ICAR 2009. International Conference on. (June 2009) 1–8
10. Steck, A., Lotz, A., Schlegel, C.: Model-driven engineering and run-time model-usage in service robotics. In: Proceedings of the 10th ACM International Conference on Generative Programming and Component Engineering. GPCE '11 (2011) 73–82
11. Trojanek, P.: Model-driven engineering approach to design and implementation of robot control system. CoRR **abs/1302.5085** (2013)
12. Gtz, S., Leuthuser, M., Reimann, J., Schroeter, J., Wende, C., Wilke, C., Amann, U.: A role-based language for collaborative robot applications. In Hhnle, R., Knoop, J., Margaria, T., Schreiner, D., Steffen, B., eds.: Leveraging Applications of Formal Methods, Verification, and Validation. Communications in Computer and Information Science. Springer Berlin Heidelberg (2012) 1–15
13. Kendoul, F.: Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. J. Field Robot. **29**(2) (March 2012) 315–378
14. Hehn, M., D'Andrea, R.: Quadrocopter trajectory generation and control. In: IFAC world congress. (2011) 1485–1491
15. Augugliaro, F., Schoellig, A., D'Andrea, R.: Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In: Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on. (Oct 2012) 1917–1922
16. Leonard, J., Savvaris, A., Tsourdos, A.: Towards a fully autonomous swarm of unmanned aerial vehicles. In: Control (CONTROL), 2012 UKACC International Conference on. (Sept 2012) 286–291