A Generated Property Specification Language for Resilient Multirobot Missions

Swaib Dragule[1,4], Bart Meyers[2,3], and Patrizio Pelliccione[1]

[1] Dep. of Computer Science and Engineering
Chalmers University of Technology | University of Gothenburg
Göteborg, Sweden
dragule@chalmers.se, patrizio.pelliccione@gu.se
[2] Antwerp Systems and Software Modelling
University of Antwerp
Antwerpen, Belgium
bart.meyers@uantwerpen.be
[3] Flanders Make vzw
[4] Makerere University, Uganda

Abstract The use of robots is gaining considerable traction in several domains, since they are capable of assisting and replacing humans for everyday tasks. To harvest the full potential of robots, it must be possible to define missions for robots that are domain-specific, resilient, and collaborative. Currently, robot vendors provide low-level APIs to program such missions, making mission definition a task-specific and error-prone activity. There is a need for quick definition of new missions, by users that lack programming expertise, such as farmers and emergency workers. In this paper, we extend the existing FLYAQ platform to support the high-level specification of adaptive and highly-resilient missions. We present an extensible specification language that allows users to declaratively specify domain-specific constraints as properties of missions, thus complementing the existing FLYAQ mission language. This permits to move at runtime, the actual generation of low-level operations to satisfy the declaratively specified mission. We show how this specification language can be automatically generated from a domain-specific FLYAQ mission language by using the generative ProMoBox approach. Next, we show how mission goals are achieved taking mission properties into account, and how missions may change due to unexpected circumstances.

Keywords: Domain-Specific Languages, Robotics, Model-Driven Engineering, Resilient Systems, Cyber-Physical Systems.

1 Introduction

The use of multirobot systems in civilian missions requires high variability due to the diversity of domains [4,21]. Moreover, robotic systems are defined through a craftsmanship instead of established engineering processes. Programming missions for robots requires high knowledge of robotic programming and robot mechatronics. While domain users are experts in their domains (e.g., emergency, commercial and agriculture) they are not trained to program missions for multirobot execution in their domains using the low-level APIs provided by robot vendors. Not much has been done to enable domain experts to easily use robots to execute missions in the respective domains.

To address this problem, Di Ruscio et al. introduced FLYAQ [3,18]. FLYAQ is a platform designed to enable non-expert domain users to program missions for a team of multicopters. The platform has been then generalized to different types of robots in [4,6]. The platform is extensible, so that domain-specific robots and missions can be defined. Unfortunately, this platform can only define missions at design time. This is unrealistic since most missions will be faced by unforeseeable and emergent situations during mission execution, and, consequently, robots should be resilient to these unforeseeable and emergent situations. For example, one robot may malfunction calling for re-planning so that another robot can take the roles this robot was executing. This need for run-time adaptation is clearly described in the Robotics Multi-Annual Roadmap 2020 [21]. In this context, the document describes the degree in which models can be used in robotics in three steps ([21] § Section 5.2). Step 1 assumes that models are used to define missions by people at design time. Step 2 requires robots to use models at run-time to interact and explain what they are doing. Step 3 means that robots adapt and improve models to redefine what they are doing based on artificial intelligence.

The FLYAQ platform uses models according to step 1. In this line of research, we intend to improve FLYAQ to support self-adaptive robots at the mission level, thus achieving step 3. This means that robots can change their behaviour to successfully carry out missions under unforeseen circumstances. We achieve this by introducing a declarative language for describing mission goals and constraints. In this research we exclusively focus on the high-level strategic, domain-specific, collaborative aspects of self-adaptation. To this end, we specify mission objectives in a declarative way, as properties, using a language we call the Mission Specification Language (MSL). We present a technique that allows the generation of such a MSL for a specific FLYAQ extension (e.g., emergency, commercial, agriculture). As MSL is declarative, it does not specify *how* the mission is planned for a team of robots, but instead specifies what goals must be achieved and what constraints cannot be violated. This way, missions become fully specified only at run-time and they can be re-planned at run-time.

Paper structure: Section 2 discusses the background of this research. Section 3 introduces the property specification language. Section 4 evaluates the approach by showing an implementation of the property specification language. Section 5 discusses related work. Section 6 concludes the paper with opportunities for future works.

2 Background

In this section, we briefly explain domain-specific modelling, and the FLYAQ platform, on which we build our research.

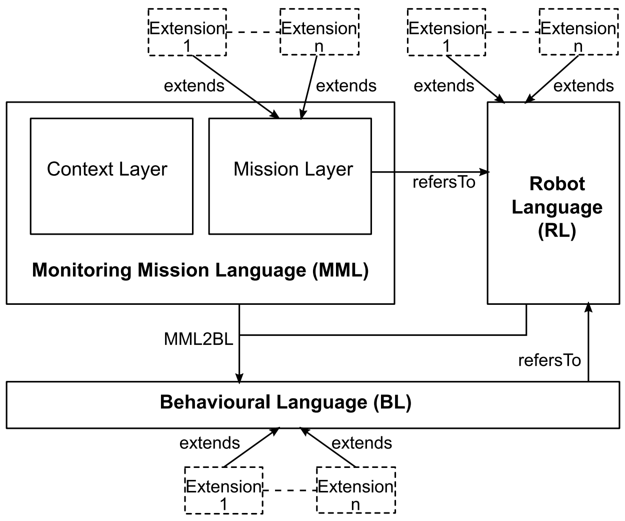


Figure 1. The family of FLYAQ DSMLs (adapted from [6]).

Domain Specific Modelling In Domain-Specific Modelling (DSM) [13], a methodology in model-driven software engineering, the general goal is to provide means for domain users to model systems in their problem domain. Model-driven techniques such as metamodelling and model transformation enable the creation of Domain-Specific Modelling Languages (DSMLs). These DSMLs can be used by domain experts, to specify, for example, missions for a team of robots. Current DSM techniques allow domain users to model at the domain level and simulate, optimise, and transform the model to other formalisms, synthesise code, generate documentation, etc.

FLYAQ platform The FLYAQ platform [18,3,4] employs domain-specific modelling to take care of the various domains involved in mission definition and specification. The approach proposes a family of DSMLs for the specification of missions of multirobot systems (MMRSs), as shown in Figure 1:

– Monitoring Mission Language (MML): this DSML consists of the context layer and mission layer. This DSML is meant to be used by domain users, to model missions. Missions are represented in the mission layer as sequences of tasks on a map, as shown in Figure 2. The context layer provides additional constrains over the mission area, such as obstacles and no-fly zones;
– Robot Language (RL): using this DSML, types of robots or individual robots can be defined by a robot engineer, mapping out their capabilities and characteristics;
– Behaviour Language (BL): this language allows the definition of sequential atomic movements and actions of each robot that are used to instruct the individual robots. The BL serves as the low-level language, to which high-

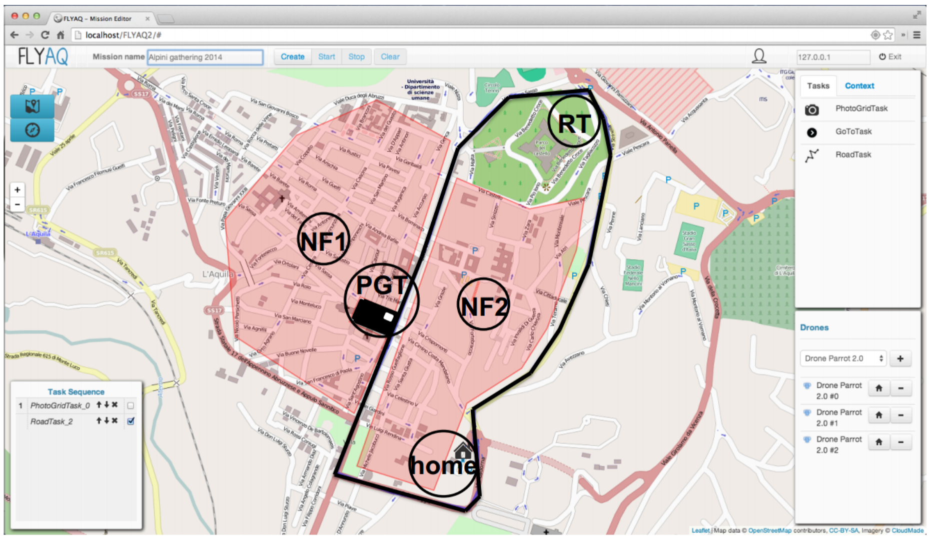


Figure 2. A screenshot of the FLYAQ tool (from [18]).

level missions defined in MML can be transformed automatically using the MML2BL transformation. This transformation takes care of low-level planning, such as path finding, covering areas, etc. while achieving the high-level goals. Code can be easily generated from the generated BL models, and then it can be uploaded to the individual robots.

Mission goals, robot characteristics, and actions should be customised to the application domains. Therefore, extensions can be defined on MML, RL and BL, as shown in Figure 1. In case of MML, extensions may define a task to “scan an area by taking pictures”. Example extensions to RL may include domain-specific notions like “number of propellers”, “launch type” (horizontal or vertical takeoff), “maximum altitude”, etc. BL may be extended with movements like “take off” and “land”, and a “go to strategy” (move first over the horizontal or vertical axis, or move diagonally?), “take a picture”, “start recording a video”, etc.

For example, it is possible to define extensions in FLYAQ to allow flying robots to take pictures of areas. Using this extension, one can specify missions to e.g., survey an area where a public event is being held. Another example is in the domain of agriculture. One multicopter is able to detect pests by taking pictures and using image recognition techniques. If a pest is detected, another multicopter that is able to spray insecticide must spray the infected plants. It should only spray plants that are infected. We use these examples throughout the paper.

Despite its extension mechanism, FLYAQ does not support (a) advanced temporal constraints (other than order, fork or join) over various tasks or robots in MML, e.g., a certain task can only start if another robot is surveying the task area (for safety reasons), or video recording can only start after clearance (for privacy reasons); and (b) run-time adaptation of a mission due to some

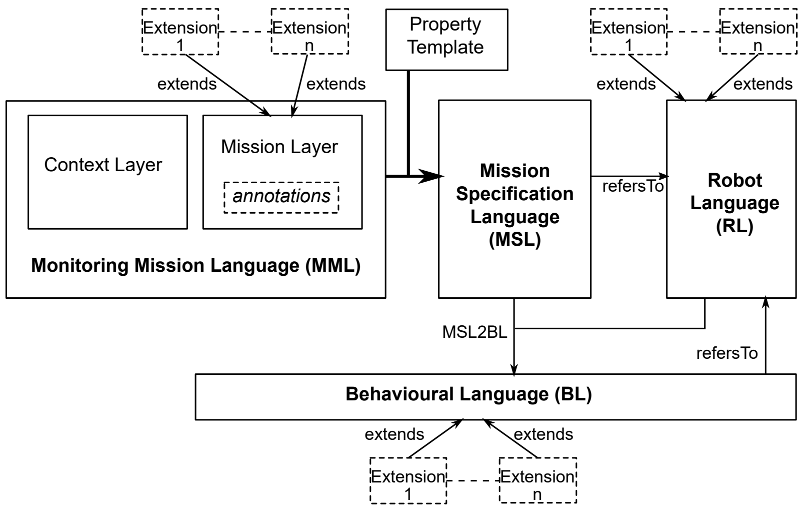


Figure 3. Overview of the approach as an extension of the FLYAQ platform.

information at run-time, e.g., taking pictures of areas where high temperature was detected by another robot, or reacting to a loss of signal of a robot. The research presented in this paper addresses these shortcomings.

3 Mission Specification Language

Our approach extends the FLYAQ platform as shown in Figure 3. The mission layer of MML is annotated, and as a consequence a *Mission Specification Language* (MSL) can be generated automatically from MML and a *Property Template* to better match the platform extensions of MML. MSL extends MML with language constructs to define temporal properties for robot missions. Our approach ensures that, when an extension is defined as done in FLYAQ, no additional effort is required to generate MSL.

3.1 Mission Specification Language

The mission specification language (MSL) is intended to specify properties of a mission that allows users to define temporal mission constraints in a highly declarative way. This complements MML, where areas are selected, and specific tasks, obstacles and no-fly zones are plotted on the map. MSL replaces the order, fork and join of MML, supporting more expressive constraints. We use a number of temporal patterns, taken from Dwyer et al. [7] and Autili et al. [1], as a basis for the Property Template from which MSL is generated. According to this work, properties consist of a *temporal pattern* in a *scope*, over some propositions P, Q, R and S (i.e., occurrences of something, e.g., spraying, entering an area, etc.). Temporal patterns can be *absence* (something should never occur), *universality* (something should always occur), *existence* (something should eventually occur), *bounded existence* (something should occur at most n times), *precedence* (an occurrence of P must be preceded by an occurrence of Q), or *response* (an occurrence of P must be followed by an occurrence of Q). Scopes can be *globally*, *after* the occurrence of R, *before* the occurrence of S, *between* occurrences of R and S, or after an occurrence of R until an occurrence of S (*after until*).

The declarative constraint specification shields the user from the actual planning. For example, if pests are detected, the corresponding areas are sprayed. This is an example of a response pattern with global scope. The user may use a precedence pattern to say that a pest needs to be detected at a location before this point is sprayed. This constraint can be met in a number of equally valid ways. A first option would be that one robot first detects all locations, then returns to the base where its data is downloaded and locations of infected plants are uploaded to a second robot, who goes out to spray the infected plants. A second option would be that two robots perform the task in parallel: one robot sends coordinates of detected pests to the other robot, which only sprays infected points. The second robot may follow a preplanned path, or may plan its path at run-time, according to the received coordinates. Collisions may occur, or may be avoided by flying at different altitudes. A third option would be that multiple robots detect pests, and multiple robots spray. If robots can adapt their mission at run-time, this may involve advanced scheduling, employing run-time monitors [5]. This shows that a declarative language can be supported by very simple to very advanced algorithms. The goal of MSL is that the domain user is shielded from such advanced planning algorithms.

To further illustrate MSL, we give some more examples of properties.

– Between entering and exiting an area, a robot can never exceed a given altitude. According to Dwyer et al. [7], this is an absence pattern with between scope. Note that this between scope may be more intuitively expressed as "during" or "while".
– Between receiving a "stop" message and a "start" message, pictures cannot be taken.
– A robot can only start its activity if another robot is in a given position to monitor this activity.

3.2 Run-time Adaptation of Multirobot Missions

In its current state, the MML platform generates robot missions at design time. This means that robot missions cannot be adapted at run-time. We intend to support the run-time recalculation of BL models (i.e., robot commands) from a declarative mission description; this is needed in case information at run-time prompts the robots to change the mission. Our approach is applicable to various implementation techniques: for example, the mission recalculation may be achieved by the robot or by the ground station, and may be specified off-line or at run-time, or a mix of these.

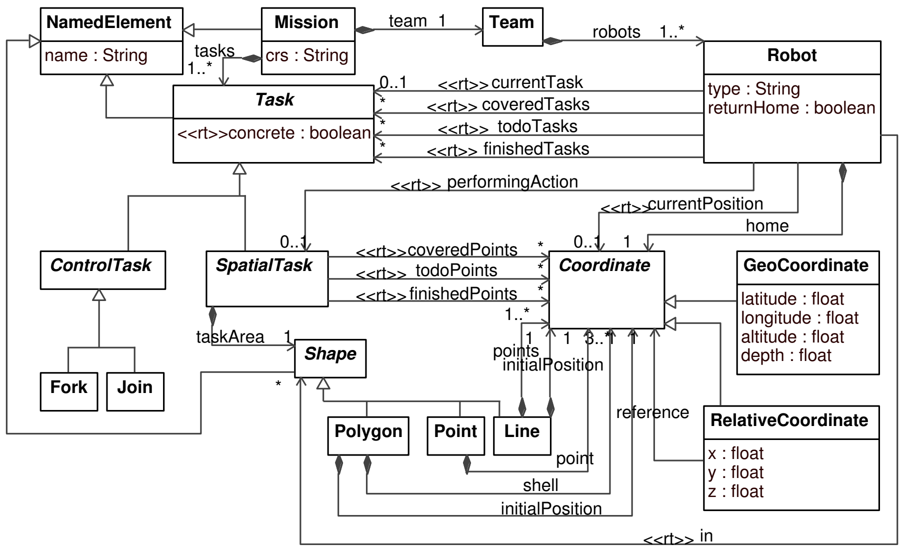


Figure 4. The annotated class diagram of the MML mission layer (context layer remains unchanged and is not shown).

In order to allow run-time information in a mission specification in MSL, we altered the existing MML mission layer from [4], as shown in Figure 4. We have changed the metamodel in several ways:

- We have extracted a Shape class (and Polygon, Point, Line subclasses) from the original PolygonTask, LineTask, and PointTask. In particular, the new Polygon class serves now as superclass of Area in the context layer of MML in [6]. This new Shape class will allow users to specify new shapes on the map that may trigger rules like: do not record within a specific area.
- The meaning of Task has been extended. At mission specification time, a task may be addressed by multiple robots. After mission generation, tasks are split up into multiple concrete tasks, each for one robot.
- TaskDependency has been removed from MML and its functionality will be subsumed by the specification language.
- We added run-time language constructs (annotated with *rt*), so that specifications can be defined in terms of the current state of the mission in terms of tasks and position. We added the following run-time information in terms of tasks:
  - *currentTask*: the task a robot is currently working on;
  - *coveredTasks*: the concrete tasks that are planned for a robot;
  - *todoTasks*: the concrete tasks that a robot still needs to perform;
  - *finishedTasks*: the concrete tasks that a robot has done;

- *performingAction*: the action (defined in the task) a robot is currently performing. It may be none if e.g., the robot is moving and the action is instantaneous (e.g., taking a picture).

We added the following run-time information in terms of position:

- *currentPosition*: the current position of a robot;
- *coveredPoints*: the points of a concrete task that are defined by the cover function;
- *todoPoints*: the points of a concrete task that still need to be visited;
- *finishedPoints*: the points of a concrete task that have been visited;
- *in*: the shapes the robot is currently in.

As is usual in FLYAQ, extensions can be defined for specific application domains, as shown in Figure 3. Note that for brevity, we do not show the MML context layer and RL (which can be extended in its own right).

3.3 Generation of the Property Specification Language

As shown in Figure 3, a domain-specific MSL can be generated from the annotated MML (as shown in Figure 3), with defined extensions (e.g., to enable detection of pests in an area, and spraying certain plants). This means that extensions have to be defined only once, and can be used for specifying missions in the original MML as well as in MSL. The metamodel of MSL, which results from the language generation process without an extension, is shown in Figure 5. It consists of three parts:

- Mission layer: the upper part (unshaded) represents our variant to the original MML mission layer, which allows the user to define missions at design-time like in the original MML. For example, “pictures should be taken in an area, with a distance of x from each other”. Additionally, shapes can be defined, that can be used in MSL properties. In case of an MML extension, extensions will also appear in this part.
- Temporal pattern layer: the middle part (shaded) represents the temporal patterns, which allow the user to define temporal constraints based on the patterns by Dwyer et al. [7]. For example, “after R happens, P must be followed by Q”.
- Proposition layer: the bottom part (unshaded) represents the language fragment to define propositions P, Q, R, and S of temporal patterns. More specifically, it allows the user to specify a condition on the state of a mission (i.e., a structural pattern). For example, “a robot is in a specific area”, or “a task is completed”. In case of an MML extension, pattern versions of extensions will also appear in this part.

With MSL, a mission can be specified by plotting an area on the map, and defining a DetectPest and Spray task in this area, using the MSL mission layer, which is extended with language concepts from agriculture. With the MSL temporal pattern and MSL proposition layer, a property can be specified that states that detecting a pest at a location must result in spraying that location.

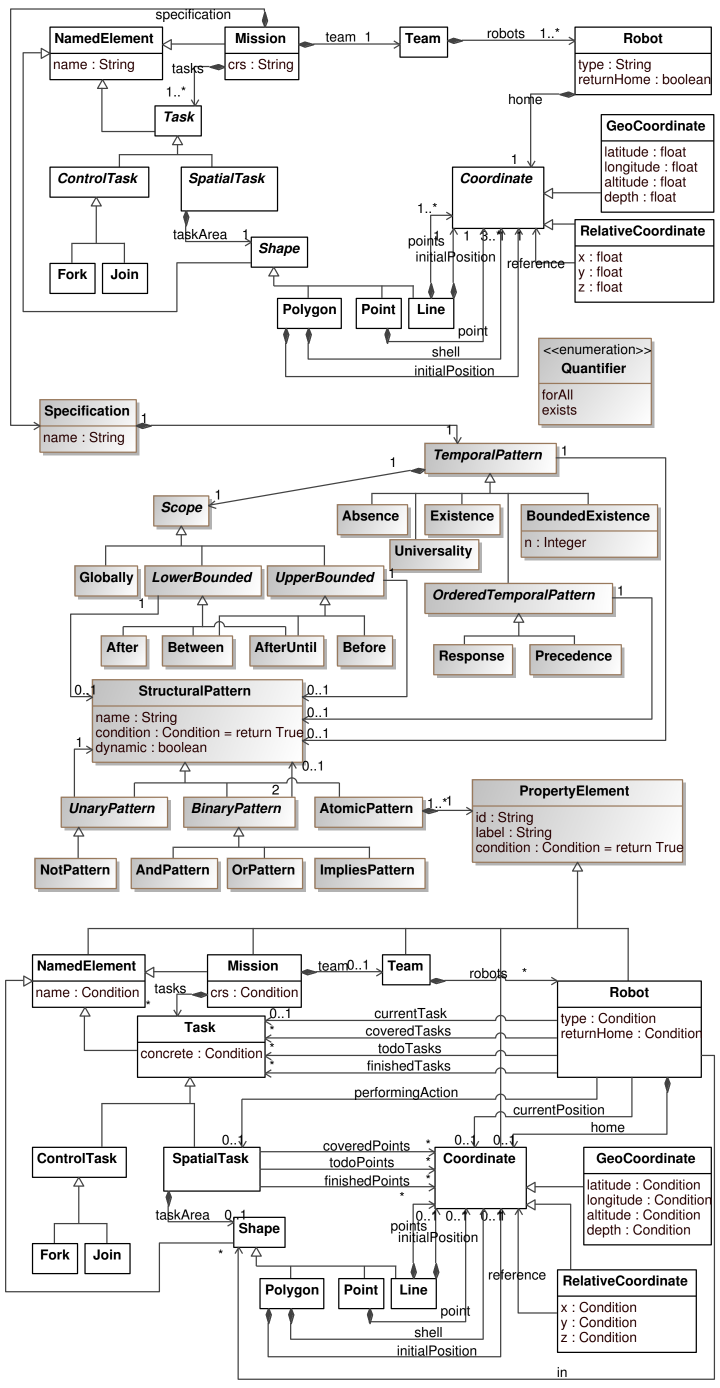


Figure 5. The generated metamodel of MSL without extensions.

The MSL metamodel of Figure 5 is generated fully automatically from an annotated (and possibly extended) MML metamodel (Figure 4) and the generic property template (shaded part of Figure 5). Why do we need to automatically generate an MSL metamodel? Please note that, if there was no generative approach, each of the domain-specific language constructs (e.g., spraying a pest, maximum altitude of a robot, etc.) would have had to be modelled a second time in MSL. With our approach, the extension mechanisms of FLYAQ can be reused as described in [4], and a domain-specific MSL is generated without any additional effort.

We use techniques from the ProMoBox framework [17,16] to achieve this. First, the MSL mission layer is generated by taking the annotated MML metamodel and removing all run-time language constructs, which are annotated with *rt*, thus creating the unshaded upper part of Figure 5. Next, the property template is merged into this model by adding an association called *specification* from Mission to Specification. Finally, the annotated MML metamodel is taken for a second time, and the run-time language constructs are kept (by removing the annotation). This time, the metamodel is converted into a structural pattern language by using the RAMification process [15]: relaxing all lower multiplicities, making all abstract classes concrete, and changing all attribute types to Condition, as can be seen in the proposition layer of Figure 5. This RAMified metamodel is merged into MSL by generating inheritance links from all top-level classes to PropertyElement.

3.4 Transforming MSL to BL

Transforming MSL to BL (see MSL2BL in Figure 3) can be done according to several stategies. Given the tight relation between MSL and MML, the transformation algorithms of MML2BL [18] (i.e., path finding, covering areas) can be reused. Moreover, various implementation strategies as mentioned in Section 3.1 can be covered by MSL2BL: mission recalculation may be achieved by a mix of the robot or the ground station, off-line or at run-time. These strategies may requiring enhancement of BL to e.g., explicitly support data communication or monitoring. As this paper focuses on the definition and generation of MSL, this is left as future work.

4 Evaluation: Implementation of MSL as Textual DSL

In this section, we evaluate the MSL by introducing an implementation as a textual language in Xtext [8], and show how missions can be expressed in this language.

4.1 A Concrete Syntax for MSL

According to what described in [1], temporal properties (the shaded part of Figure 5) might be profitably described using a structured English grammar. For instance, we can devise a textual syntax for the MSL proposition layer (the bottom part of Figure 5), where each of the associations can form a subsentence with the two attached instances. For example, "*a Robot currently on a GeoCoordinate*" denotes the presence of an instance of Robot and an instance of GeoCoordinate, with a currentPosition link in between. More intricate, "*a Robot r currently on a GeoCoordinate with latitude lower than 100*" denotes additional conditions on the robot, etc. A structured English grammar to represent a subsentence for one association is defined as follows (id, Label, Value, Attribute, Class, Association are terminals):

Proposition ::= Proposition (**and also** Proposition)+
| Proposition (**or** Proposition)+
| Proposition (**implies** Proposition)+ | AtomicProposition
AtomicProposition ::= Expression [Association Expression]
Expression ::= Instance [InstanceCondition]
InstanceCondition ::= **with** (ValueCondition | BooleanCondition (**and** ValueCondition | BooleanCondition)*)
ValueCondition ::= {Attribute} (**as** | **less than** | **greater than**) {Value}
BooleanCondition ::= [**not**] {Attribute}
Instance ::= {id} | {Label} | **a** {Class} [{Label}]
Association ::= (**that is a task of** | **that is a team of** | **that is in** | [**currently**] **doing** | **that has scheduled** | **that has planned in the future** | **that has finished** | [**currently**] **performing** | **in** | [**currently**] **on** | **with as home** | **with task area** | **which visits** | **which will visit in the future** | **which has visited** | **with points** | **with initial position** | **which references** | {Association})

The above grammar is combined with the grammar for temporal properties presented in [1] so that temporal properties can be described in standard LTL or CTL. This might enable the use of model checking approaches, like UPPAAL[5]. With this grammar, temporal patterns involving multiple links can be expressed with AndPatterns. MML extensions can be used by instantiating classes defined by the extension. This is illustrated below in the examples.

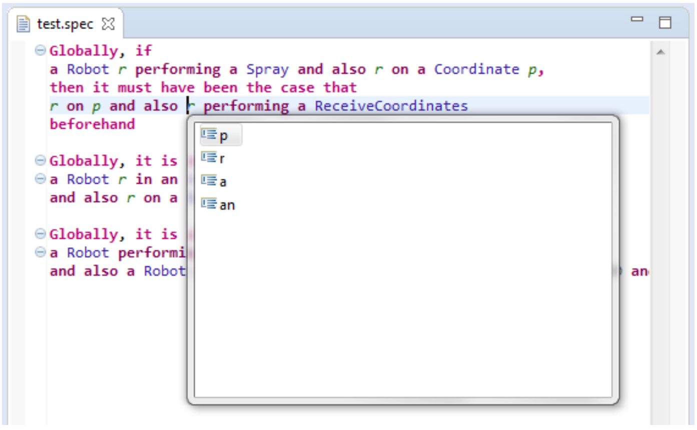
Our current implementation in Xtext includes variable name resolution, parse error visualisation, auto-completion and syntax highlighting[6]. A screenshot of the MSL editor is shown in Figure 6. Since both the FLYAQ platform and MSL are implemented on top of the Ecore platform, they can be easily merged at the EMF layer [19].

4.2 Examples of MSL

This section presents examples of temporal properties defined in MSL, while illustrating the relation between the grammar presented above and the MSL metamodel presented in Figure 5. For these examples, we define a MML extension in the agricultural domain as shown in Figure 7, with:

– DetectPest: scanning for a pest in an area and in case of detection, send some coordinates;

[5] http://www.uppaal.org/
[6] An implementation of this grammar can be found at http://msdl.cs.mcgill.ca/people/bart/flyaq/flyaq.html.



```
Globally, if
a Robot r performing a Spray and also r on a Coordinate p,
then it must have been the case that
r on p and also r performing a ReceiveCoordinates
beforehand

Globally, it is
a Robot r in an
and also r on a

Globally, it is
a Robot performi
and also a Robot                                            an
```

Figure 6. Screenshot of the MSL in Xtext.

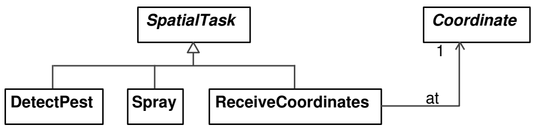


Figure 7. The MML extension.

- Spray: spraying pesticides at a point;
- ReceiveCoordinates: receiving coordinates where a pest has been detected, with the "at" association referring to the received coordinates.

Note that, after generation of MSL, these additional language constructs will occur twice in MSL, namely in the MSL mission layer and in the MSL proposition layer.

The example of Figure 8 (top) shows the abstract syntax of the MSL property "a robot only sprays at a location if it has received these coordinates to spray at that location" as an object diagram. The Specification consists of a Precedence pattern. The left AtomicPattern states the condition Q, saying that a ReceiveCoordinates task is executed at a coordinate **p**. Note how the "at" association is used. The right AtomicPattern P describes a robot **r**, spraying at aforementioned point **p**. Note that the coveredPoints link is superfluous, because if the robot is currently performing an action of a task, it must be inside the task area. In structured English grammar, the temporal specification is as follows (leaving out the superfluous quantification and coveredPoints link): "*Globally, if a SprayRobot **r** performing a Spray and **r** on a Coordinate **p**, then it must have been the case that a ReceiveCoordinates **at p** beforehand*". Note how "at" is automatically resolved to an instance of the "at" association.

The example of Figure 8 (bottom left) represents "in a certain area, a robot can never exceed a given altitude". Note that the Area class (now a subclass of Polygon) is part of the MML context layer and is not shown in Figure 5. In

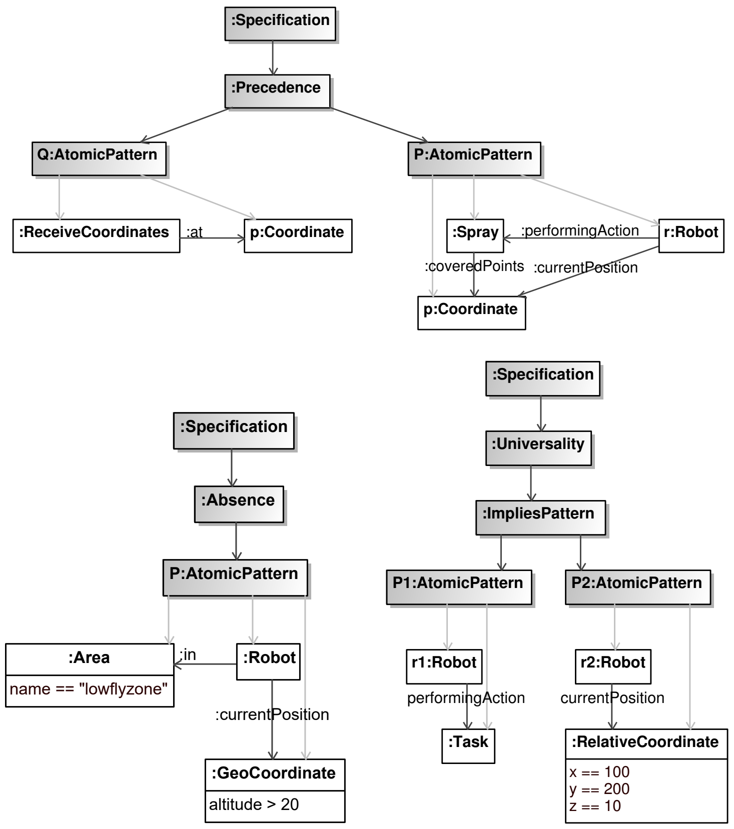


Figure 8. Three examples of temporal specifications as instances of MSL.

structured English grammar, the temporal specification is as follows: "*Globally, it is never the case that a Robot **r** in an Area with name as "lowflyzone" and also **r** on a GeoCoordinate with altitude more than 20*".

The example of Figure 8 (botton right) represents "a robot can only perform a certain task if another robot is at a certain position". In structured English grammar, the temporal specification is as follows: "*Globally, it is always the case that a Robot performing a Task implies a Robot on a RelativeCoordinate with **x** as 100 and **y** as 200 and **z** as 10*". Note that, since spatial constraints are at the very core of FLYAQ, it is interesting to introduce syntactic sugar for a robot being at a coordinate, e.g., by allowing syntax like "a robot on (100, 200, 10)".

5 Related Work

In this section, we present related works on run-time adaptation of multirobot missions. There are several works that focus on robotics and self-adaptation, like [10,20,9]. For the sake of space in this section we focus on related works in run-time adaptation of (robot) missions, with a focus on MDE approaches. While most of the mission specification tools (e.g., [23]) and the FLYAQ platform [3,6] provide for specification of multirobot missions at design time, there is need to have specification and recalculation of missions at run-time for missions executed under uncertain environments.

In an effort to leverage run-time adaptation for UAV based systems, the work in [2] uses an ensemble concept to aggregate teams collaborating in a mission at run-time. This platform focuses on the aggregation of agents but on not the high-level expressiveness of the mission properties. Using run-time models for automatic reorganization of multirobot system, the work in [24] focuses more on techniques for task distribution based on the goals and organisation of the teams, but not how goals are expressed so that adaptation at run-time is made easy. In [11] robots adapt models at run time, but configurations are made by an expert programmer, not domain experts declaratively. The work in [14] focuses on the behavioural model and how it auto-validates at run-time, yet we employ a generative approach. The work in [22] focuses on design-time to run-time explication of models, however this work does not really deal with adaptation triggered by run-time uncertainties. The work in [12] proposes an approach that uses models at design-time and run-time for collaboration. The proposed approach is specific to a particular domain without a clear path to adapt it for working in other domains.

6 Conclusion and Future Work

In this paper, we extended the FLYAQ platform with MSL, a highly declarative language that allows users to describe robot missions with temporal properties as constraints. The declarative nature of MSL allows run-time adaptation of these missions in case of unforeseen circumstances. We showed how MSL can be automatically tailored with domain-specific extensions by a generative approach. Additionally we presented a structured English grammar for MSL.

Future work will mainly focus on the mapping from MSL to BL (a language for describing individual robot movements and actions), allowing run-time adaptation and exploring different execution strategies. We intend to model communication between robots and/or the ground station explicitly in BL to achieve this. Furthermore, we are planning to incorporate real-time constraints in missions. Moreover, since our approach for enabling run-time adaptation of missions is model-driven and relies on code generation, we intend to analyse the feasibility of generating code in real-time.

Acknowledgement

This research is partially supported by Flanders Make vzw. This research is also partially funded by the COST action IC1404 “MPM4CPS". This work has been carried out within the framework of the MBSE4Mechatronics project (grant nr.130013) of the agency for Innovation by Science and Technology in Flanders (IWT-Vlaanderen). More support for this work was from the SIDA Bright 317 project. Finally, this work is partially supported from the EU H2020 Research and Innovation Programme under GA No. 731869 (Co4Robots).

References

1. Autili, M., Grunske, L., Lumpe, M., Pelliccione, P., Tang, A.: Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar. IEEE Trans. Software Eng. 41(7), 620–638 (2015), http://dx.doi.org/10.1109/TSE.2015.2398877
2. Bozhinoski, D., Bucchiarone, A., Malavolta, I., Marconi, A., Pelliccione, P.: Leveraging Collective Run-Time Adaptation for UAV-Based Systems. 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) pp. 214–221 (2016), http://ieeexplore.ieee.org/document/7592799/
3. Bozhinoski, D., Di Ruscio, D., Malavolta, I., Pelliccione, P., Tivoli, M.: FLYAQ: Enabling non-expert users to specify and generate missions of autonomous multicopters. Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015 pp. 801–806 (2015)
4. Ciccozzi, F., Di Ruscio, D., Malavolta, I., Pelliccione, P.: Adopting MDE for Specifying and Executing Civilian Missions of Mobile Multi-Robot Systems. IEEE Access 3536(c), 1–1 (2016), http://ieeexplore.ieee.org/document/7576686/
5. Cohen, D., Feather, M.S., Narayanaswamy, K., Fickas, S.: Automatic monitoring of software requirements. In: Adrion, W.R., Fuggetta, A., Taylor, R.N., Wasserman, A.I. (eds.) Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997. pp. 602–603. ACM (1997), http://doi.acm.org/10.1145/253228.253493
6. Di Ruscio, D., Malavolta, I., Pelliccione, P.: A family of domain-specific languages for specifying civilian missions of multi-robot systems. CEUR Workshop Proceedings 1319, 16–29 (2014)
7. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Boehm, B.W., Garlan, D., Kramer, J. (eds.) Proceedings of the 1999 International Conference on Software Engineering, ICSE' 99, Los Angeles, CA, USA, May 16-22, 1999. pp. 411–420. ACM (1999), http://portal.acm.org/citation.cfm?id=302405.302672
8. Eysholdt, M., Behrens, H.: Xtext: implement your language faster than the quick and dirty way. In: Cook, W.R., Clarke, S., Rinard, M.C. (eds.) Companion to the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, SPLASH/OOPSLA 2010, October 17-21, 2010, Reno/Tahoe, Nevada, USA. pp. 307–309. ACM (2010), http://doi.acm.org/10.1145/1869542.1869625
9. Filieri, A., Tamburrelli, G., Ghezzi, C.: Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. IEEE Transactions on Software Engineering 42(1), 75–99 (Jan 2016)
10. Franco, J.M., Correia, F., Barbosa, R., Zenha-Rela, M., Schmerl, B., Garlan, D.: Improving self-adaptation planning through software architecture-based stochastic modeling. Journal of Systems and Software 115, 42 – 60 (2016), http://www.sciencedirect.com/science/article/pii/S0164121216000212

11. Gherardi, L., Hochgeschwender, N.: RRA: Models and tools for robotics run-time adaptation. IEEE International Conference on Intelligent Robots and Systems 2015-Decem, 1777–1784 (2015)
12. Götz, S., Leuthäuser, M., Reimann, J., Schroeter, J., Wende, C., Wilke, C., Aßmann, U.: A role-based language for collaborative robot applications. Communications in Computer and Information Science 336 CCIS(1), 1–15 (2012)
13. Gray, J., Neema, S., Tolvanen, J., Gokhale, A.S., Kelly, S., Sprinkle, J.: Domain-specific modeling. In: Fishwick, P.A. (ed.) Handbook of Dynamic System Modeling. Chapman and Hall/CRC (2007), http://dx.doi.org/10.1201/9781420010855.pt2
14. Kim, Y., Jung, J.W., Gallagher, J.C., Matson, E.T.: An Adaptive Goal-Based Model for Autonomous Multi-Robot Using HARMS and NuSMV. The International Journal of Fuzzy Logic and Intelligent Systems 16(2), 95–103 (2016), http://www.ijfis.org/journal/view.html?doi=10.5391/IJFIS.2016.16.2.95
15. Kühne, T., Mezei, G., Syriani, E., Vangheluwe, H., Wimmer, M.: Explicit transformation modeling. In: Ghosh, S. (ed.) Models in Software Engineering, Workshops and Symposia at MODELS 2009, Denver, CO, USA, October 4-9, 2009, Reports and Revised Selected Papers. Lecture Notes in Computer Science, vol. 6002, pp. 240–255. Springer (2009), http://dx.doi.org/10.1007/978-3-642-12261-3_23
16. Meyers, B., Denil, J., Dávid, I., Vangheluwe, H.: Automated testing support for reactive domain-specific modelling languages. In: Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering - SLE 2016. pp. 181–194. ACM Press, New York, New York, USA (2016), http://dl.acm.org/citation.cfm?doid=2997364.2997367
17. Meyers, B., Deshayes, R., Lucio, L., Syriani, E., Vangheluwe, H., Wimmer, M.: ProMoBox: A Framework for Generating Domain-Specific Property Languages. In: International Conference on Software Language Engineering (SLE). vol. 8706, pp. 1–20. Springer, Cham (2014), http://link.springer.com/10.1007/978-3-319-11245-9{_}1
18. Ruscio, D.D., Malavolta, I., Pelliccione, P., Tivoli, M.: Automatic generation of detailed flight plans from high-level mission descriptions. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems - MODELS '16. pp. 45–55. ACM Press, New York, New York, USA (2016), http://dl.acm.org/citation.cfm?doid=2976767.2976794
19. Schätz, B.: Formalization and rule-based transformation of EMF ecore-based models. In: Gasevic, D., Lämmel, R., Wyk, E.V. (eds.) Software Language Engineering, First International Conference, SLE 2008, Toulouse, France, September 29-30, 2008. Revised Selected Papers. Lecture Notes in Computer Science, vol. 5452, pp. 227–244. Springer (2008), https://doi.org/10.1007/978-3-642-00434-6_15
20. Shevtsov, S., Weyns, D.: Keep it simplex: Satisfying multiple goals with guarantees in control-based self-adaptive systems. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 229–241. FSE 2016, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2950290.2950301
21. SPARC: Robotics 2020 Multi-Annual Roadmap 2016, 325 (2015)
22. Steck, A., Lotz, A., Schlegel, C.: Model-driven engineering and run-time model-usage in service robotics. Proceedings of the 10th ACM international conference on Generative programming and component engineering - GPCE '11 p. 73 (2011), http://dl.acm.org/citation.cfm?doid=2047862.2047875
23. Ulam, P., Endo, Y., Wagner, A., Arkin, R.: Integrated mission specification and task allocation for robot teams - Design and implementation. In: Proceedings - IEEE International Conference on Robotics and Automation. pp. 4428–4435 (2007)

24. Zhong, C., DeLoach, S.A.: Runtime models for automatic reorganization of multi-robot systems. In: Proceeding of the 6th international symposium on Software engineering for adaptive and self-managing systems - SEAMS '11. p. 20. ACM Press, New York, New York, USA (2011), http://portal.acm.org/citation.cfm?doid=1988008.1988012