STSM Report: Domain-Specific Verification Languages – Application to Autonomous Multicopters

Bart Meyers

17 November 2016

Chapter 1

Short-Term Scientific Mission

The STSM was 8 working days (7 November 2016 - 16 November 2016) at Chalmers University of Technology. The original goal was be to design a user-friendly verification language for specifying and verifying temporal properties for missions of autonomous multicopters, and provide tool support. The goal shifted in the sense that the purpose of the verification language is not verification anymore, but to express constraints for mission specification. Using this language, parts of the mission can be specified in a highly declarative way. Therefore, in this report, we state that the goal is to design a *specification language*.

During the STSM, Bart Meyers (the visitor) worked together with prof. Patrizio Pelliccione and Swaib Dragule of the Software Engineering Division at Chalmers University of Technology. The following will describe the STSM, and the further goals, which would culminate in a paper submission to SEAMS 2017, The 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems.

1.1 Background

In this section, the scientific background of the project is described.

1.1.1 Multi-Robot Missions

Patrizio Pelliccione and Swaib Dragule work on mission specification of mobile multi-robot systems (MMRS). Domain-specific modelling for multi-robot missions is at the core of Mr. Dragule's PhD thesis. In this context, a framework has been developed [3], where most of the work has been carried out for the FlyAQ project, targeting collaborative missions for multicopters [9, 2]. The goal of this research is to contain the inherent complexity of defining a mission for multiple robots. This complexity stems from the many details regarding the robots that need to be taken into account, as well as the need for technical expertise about the static and dynamic characteristics (e.g., movement dynamics and hardware capabilities) of the robots. Currently, on-site operators require

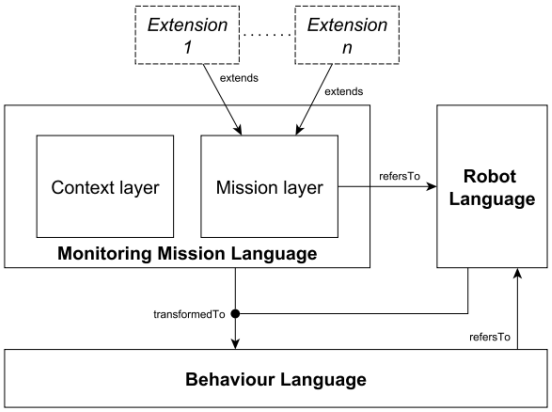

Figure 1.1: The family of DSMLs (from [3]).

deep knowledge about all these characteristics, while simultaneously controlling a swarm of robots.

The approach makes use of Domain-Specific Modelling (DSM), and Domain-Specific Modelling Languages (DSMLs). The rationale behind using DSM is manyfold: DSMLs allow the specification of software by non-technical domain users, DSMLs represent a family of systems in a domain, DSM allows support for analysis, DSM enables code generation which enhance software quality.

The approach proposes a family of Domain-Specific Modelling Languages (DSMLs) for the specification of missions of MMRSs, as shown in Figure 1.1:

- *Monitoring Mission Language* (MML): this DSML consists of the *context layer* and *mission layer*. This DSML is meant to be used by domain experts, to model missions. Missions are represented in the mission layer as sequences of tasks in certain spatial areas, which can be ordered. The context layer provide additional constrains over the mission area, such as obstacles and no-fly zones;
- *Robot Language* (RL): using this DSML, individual robots can be defined by a robot engineer, mapping out their capabilities and characteristics;
- *Behaviour Language* (BL): this language allows the definition of step-wise atomic movements and actions of each robot, such as: move to a point, take a picture, send some data.

A central part of the MML is a task. A task is composed of:

- an area: a point, line, plain or volume in which the task should be carried out;
- a strategy: defining what visit strategy needs to be employed:
  - sweep: perform the task over the entire area;
  - search: find a target in the area, and stop after the target has been found;
  - track: like sweep, but start and stop tracking depending on a message (allowing multiple passes over the area);

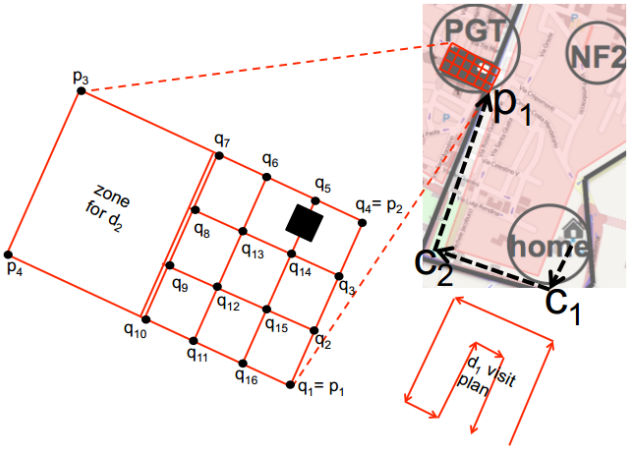

Figure 1.2: Obtaining the BL model by using the *divide*, *appr* and *cover* functions (from [9]).

- an action: the concrete action that needs to be performed for this task, e.g., taking pictures, finding a target, etc.;
- a flag indicating whether this action is instantaneous or continuous: taking a picture is instantaneous, recording a video is continuous.

Tasks can be scheduled in a simple way, which includes order, forks and joins.

The goal of the framework is to automatically generate BL models from MML and RL models. This shields the mission planner from underlying complexity laid out in the BL models. The framework includes means for generating a correct BL mission, from the given tasks, context and robot characteristics. It is very important to note that this means that the mission is fully generated at design time. One of the main benefits of the framework is that the area (i.e., lines, volumes or plains) in which tasks need to be performed can be split over multiple robots and is spatially discretized. This discretization is acquired by three functions, as shown in Figure 1.3:

- *divide*: divide the area in multiple areas so that multiple robots can be deployed for one task. In this example, the area PGT is divided into two areas, for drone d_1 and drone d_2;
- *appr*: define a way to approach the area in which the task must be carried out. In this example, a path from home, over points c_1, c_2 and p_1 is calculated;
- *cover*: defines an algorithm to cover the area, meaning that the area is discretized into points, according to a given diameter. In the example, a visit plan for drone d_1 is shown, covering all points.

As suggested by Figure 1.1, extensions can be defined by the *platform extender* in the framework, allowing the specification of a DSML for a specific type of missions. On top of such robot-type-specific framework extensions, different actions depending on the robot (and/or mission) can be defined. This results in three layers, in order of domain-specificity:

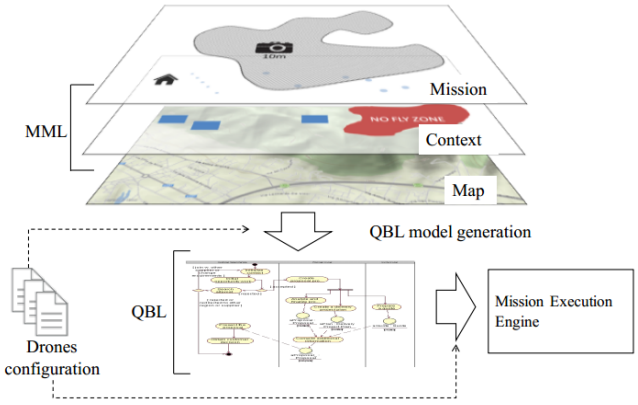

Figure 1.3: The FlyAQ platform (from [9]).

- the core framework layer of missions of MMRSs (e.g., scanning an area, or finding an object);
- robot-type-specific extensions (e.g., unmanned areal vehicles, or autonomous underwater vehicles);
- robot-specific extensions (e.g., defining a task for taking a picture, changing the BL generation algorithms).

One such extension is the FlyAQ platform, of which an overview is shown in Figure 1.3. The extension includes notions like "number of propellers", "launch type" (horizontal or vertical takeoff), maximum altitude, etc. to the RL. The BL is extended with movements like "TakeOff" and "Land", and a "GoToStrategy" (move first over the horizontal or vertical axis, or move diagonally?). In the MML, a mission and context are defined on a map. For example, missions can be defined to photograph areas as shown in Figure 1.4. The framework generates BL models, for which code generation is straightforward. This code can be uploaded on the individual multicopters.

In its current state however, the framework does not support:

- advanced temporal constraints (other than order, fork or join) over various tasks or robots in the 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);
- run-time adaptation of a mission due to some information at run-time, e.g., taking pictures of areas where high temperature was detected by another drone, or reacting to a loss of signal of a drone.

In this project, we aim at addressing these shortcomings.

1.1.2 Domain-Specific Property Languages

Verifying whether a model satisfies its requirements is an important challenge in DSM, but is nevertheless mostly neglected by current DSM approaches. Currently, domain users need to have a profound knowledge of some logic to express

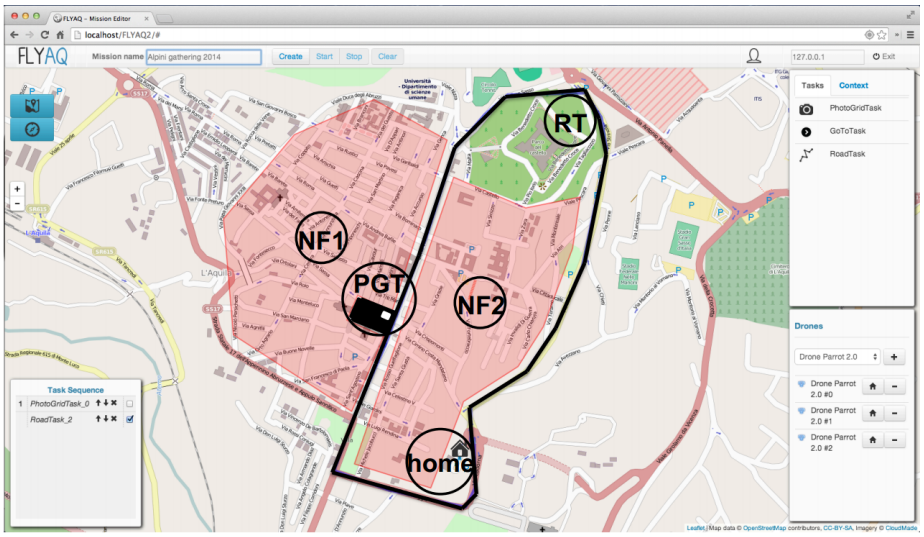

Figure 1.4: A public event monitoring mission with FlyAQ (from [9]).

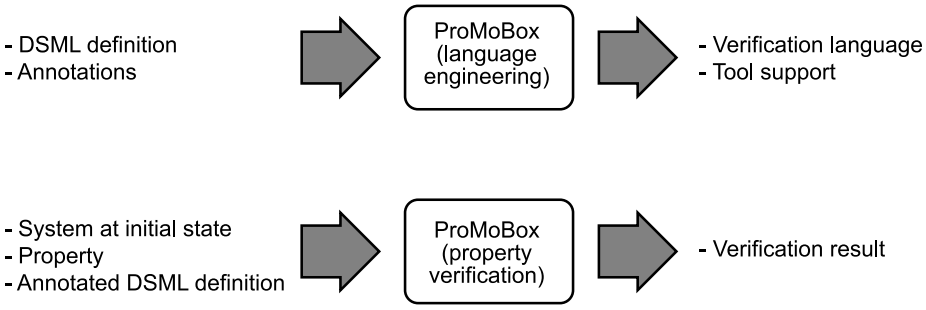

Figure 1.5: Overview of the ProMoBox approach.

properties. This violates the principles of DSM. As with design models, the level of abstraction for specification and verification tasks needs to be raised to the domain level, as domain users should not be exposed to underlying technologies. Consequently, there is a consensus that in DSM, it is better to use a DSML as a property language instead of LTL or another temporal logic. More precisely, DSM should not only address modelling the design of a system, but also its properties, its environment, its run-time state, and its execution traces, which should all be modelled at the domain level, in their own DSML.

The ProMoBox approach [8, 7, 4] presents a DSML engineering framework that aims to pull up property specification and verification tasks to the domain level.

The contribution of the ProMoBox framework is twofold. Firstly, as shown in the upper part of Figure 1.5, it includes a fully automated method to automatically generate verification support for a given design DSML, if additional *annotations* are provided. This includes a domain-specific verification language, and tool support including peripheral domain-specific *sublanguages* (for modelling environment, static design design, run-time state and execution traces of

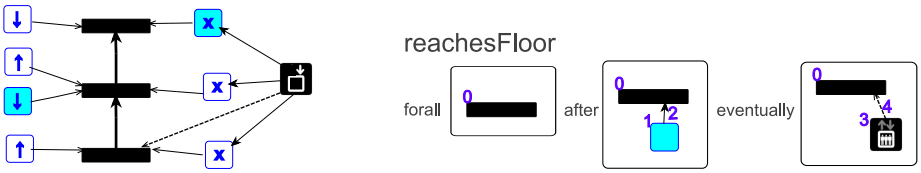

Figure 1.6: An example elevator system (left) and a property, modelled in the generated domain-specific property language (right).

the system) to enable the specification and verification of these properties. The property sublanguage is based on the specification patterns by Dwyer et al. [5]. The *temporal patterns* are the following:

- existence: a proposition P has to occur at leas once;
- absence: P can never occur;
- universality: P should always be valid;
- bounded existence: P should occur at most n times;
- response: if P occurs, then eventually Q occurs;
- precedence: every occurrence of P is preceeded by an occurence of Q;
- chained response: similar to response, but with multiple ordered causes and reactions;
- chained precedence: similar to precedence, but with multiple ordered causes and reactions;

These patterns can be defined over scopes: globally, after R, before R, between R and S, and after R until S. Propositions like, P, Q, R and S are *structural patterns*, defining a pattern on the current run-time state of the system, e.g., an elevator is at a particular floor, its doors are open, or a button is pressed. An example is shown in Figure 1.6, where an instance of a DSML for specifying elevator systems is shown on the left side, and a property on the right side (using the existence pattern with after scope), that reuses the domain-specific concepts in the structural patterns. The DSML (i.e., sublanguage) in which this property has been modelled, is fully automatically generated from the DSML definition, which has been annotated for this purpose.

Secondly, it provides a fully automated mapping to a suitable verification backbone for model checking (temporal) properties. This is illustrated in the lower part of Figure 1.5, where ProMoBox takes as input a system modelled in the DSML, a property modelled in the newly generated verification language, and the DSML definition with annotations, to fully automatically produce a verification result (i.e., an output trace in case of a counterexample). All inputs and outputs of both ProMoBox processes are raised to the domain level, so that users (domain users and language engineers alike) are shielded from the underlying temporal logic and formal models. Flexibility and automation are key in ProMoBox. ProMoBox supports definition and verification of temporal properties for any discrete-time behavioural DSML, for which the semantics can be described as a schedule of graph rewrite rules.

1.2 Activities

This section discusses the main activities of the STSM.

1.2.1 Discussions on property languages for robot missions

The majority of the collaboration consisted of discussions on property languages for robot missions, with Patrizio and Swaib.

The Specification Language for Missions of MMRSs

At first, the goal was to devise a property language for robot missions. We extensively discussed the possible use of such languages. We focus on temporal aspects, so the order of tasks, and movements or actions, is of interest. One running example was in the domain of agriculture, representing a mission for pest control. A possible mission can be: one drone scans an area for pests, and another drone can spray the infested parts of the area. A property could be defined, stating that a point can only be sprayed if a pest was detected.

Many similar properties can be devised. However, we noted that it is not interesting to verify a property that checks whether the correct order is maintained. The order is usually strongly represented in the definition of the mission. More interesting properties would be properties that describe emergent behaviour of robot collaboration. Unfortunately, while there is inherent complexity in planning missions for multiple robots, typical emergent behaviour like in parallel computing (e.g., deadlocks, etc.) tends not to occur in the domain of missions of MMRSs. A reason for this is that the framework in its current form avoids such problems by defining missions at design time, with the *divide-apprcover* functions. This avoids problems by construction. A second reason is that currently, robots do not communicate with each other during a missions. This avoids non-preplanned behaviour, and typical problems with race conditions, like starvation and deadlock.

Instead, we discussed that this language for specifying properties can also be used to describe the mission itself. Therefore, we call the property language a *specification language*. This turns out to be a more valuable use of properties. The specification language allows users to define temporal mission constraints in a highly declarative way. This complements the MML, where areas are selected, and specific tasks, obstacles and no-fly zones are plotted on the map. The specification language replaces the order, fork and join of MML, supporting more expressive constraints. The declarative constraint specification shields the user from the actual planning. For example, let us revisit the example where pests are detected, and the areas are sprayed. The user simply may use a precedence pattern to say that a pest needs to be detected at a point before this point is sprayed. This constraint can be met in a number of equally valid ways:

- If there is only one robot that can detect pests an spray, it may first perform the detection task, and when the entire area is scanned, spray where necessary.
- If there is no way to send information about detection to another drone, the mission may be completed by first performing the detection task, then

returning to the base. With the information gathered through this finished task, a second mission is generated for a robot that can spray, and is executed. Note that the current FlyAQ platform is expressive enough to support this solution.

- Two drones perform the task in parallel: one drone sends coordinates of detected pests to the other drone, which only sprays infected points. The second drone may follow a preplanned path (currently supported), or may plan its path at run-time, according to the received coordinates (currently not supported). Collisions may occur, or may be avoided by flying at different altitudes.

- Multiple robots detect pests, and multiple robots spray. If robots can adapt their mission at run-time, this may involve advanced scheduling.

This shows that a declarative language can be supported by very simple to very advanced algorithms.

Some other examples of properties are:

- (absence, between scope) Between entering and exiting an area, you can never exceed a given altitude. Note that this between scope may be more intuitively expressed as "during" or "while". This is an interesting case to introduce some syntactic sugar.

- (absence, between scope) Between receiving a "stop" message and a "start" message, pictures cannot be taken. This is an interesting property, as it is related to using a drone as a security camera. Consequently, rules for privacy need to be adhered to.

- (precedence) A drone can only start its activity if another drone is in a given position to monitor this activity.

Up to now, we only discussed properties for mission planning. We can use similar properties for the platform extender as well. Suppose that the platform extender wants to create an action that does image recognition, and only records the images between a "start" and "stop" message. Note that this behaviour could be exposed, i.e., editable by the domain user, but the platform extender may choose to not make this action editable, by making it an atomic action for the domain user. Currently, this is not supported by the framework, as actions are performed for each point. With the specification language, the platform extender can express this task's behaviour at the BL level, and therefore implement the correct translation of the task to BL.

Run-time Adaptation Robot Missions

In its current state, the platform generates robot missions at design time. This means that robots cannot be adaptive with respect to their missions. The current state of the framework is at step 1 of the robotics multi-annual roadmap 2016 [10]. We intend to provide support for mission planning at run-time, effectively enabling step 2 of the roadmap.

Indeed, many of the examples of properties stated above hint to run-time adaptation of robot missions. We intend to support the run-time recalculation of missions, possibly by the robot or by the ground station, in case information

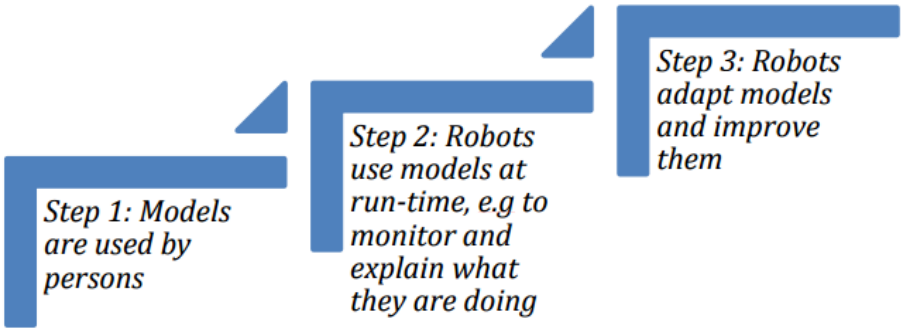

Figure 1.7: Maturity levels for using MDE in robotics (from [10]).

at run-time prompts the robots to change the mission (e.g., a pest is detected, a task is added by the domain user). In any case, this requires support for sending and receiving data, i.e., information that may prompt re-evaluating the mission (e.g., coordinates for spraying), and possibly new missions (i.e., in case the ground station plans and sends new missions).

When using run-time adaptation, the MML model created by the domain user will not contain all necessary information to fully generate the BL model. In other words, the MML model will contain "holes" that need to be filled in at run-time. This prompts a regeneration of the mission. For example, the robot that sprays pests will have to wait until it receives coordinates from the robot that detects pests. Nevertheless, it can already move to the edge of the task zone, and remain idle there. Once it receives coordinates, a new mission task will be generated, consisting of spraying one point (in contrast to spraying an area, as specified in MML). Subsequently, new coordinates are received each time a pest is detected, resulting in new spray tasks. When regenerating the mission, only parts may be regenerated or rescheduled. Often, the mission entering and mission leaving parts of a mission are left untouched, as well as existing tasks. A reordering of tasks may occur, and we will investigate reassignment of tasks to other robots.

Generation of the Specification Language

We are especially interested in a specification language that is robot-type-specific and robot-specific, because of the generative approach of ProMoBox. This way, it becomes interesting to generate the specification language from the MML/RL/BL extension. Similar to the ProMoBox approach, structural patterns can be used as propositions for the temporal patterns. To be meaningful in the context of temporal properties, these propositions represent the (partial) state of the mission at run-time. In this sense, examples of a robots can be: being in a given position, having a given target, two robots being within a given distance, the moment of receiving a message, detection of a pest, current battery level, recording a video, etc. Note that the first three examples are not domain-specific as they express a spatial state, but the four latter examples are domain-specific. For these two, the structural patterns use domain-specific concepts (i.e., pest detection, type of message) that were added by the platform

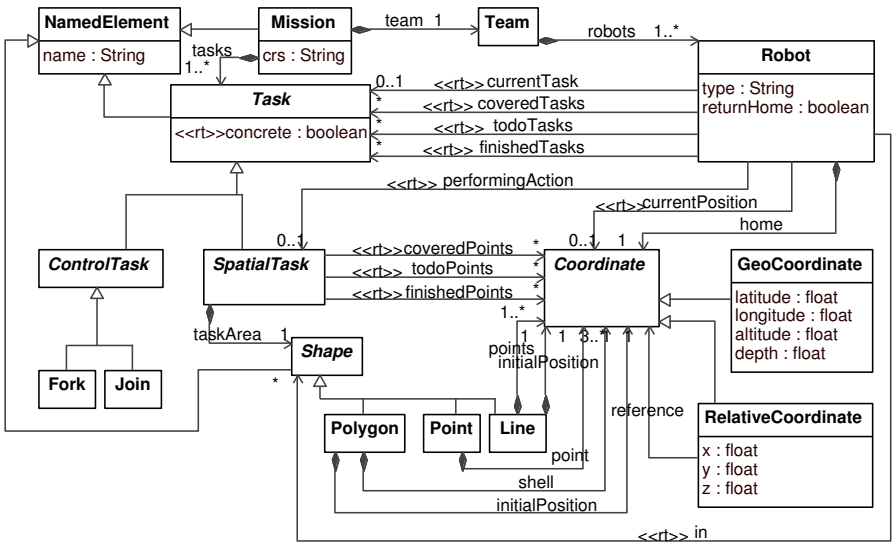

Figure 1.8: The annotated class diagram of the MML extension.

extender. If there was no generative approach, each of these domain-specific concepts would have had to be modelled a second time in the specification language. Indeed, one cannot simply reuse the same classes for e.g., pest detection. In a property, one may be interested in a certain level (e.g., above 20 bugs detected), which represents a condition for the proposition to evaluate to *true*. In the ProMoBox approach, an approach is used to generate such a "pattern version" of a language, of which the instances (i.e., propositions) are patterns that can be matched (in which case the proposition evaluates to *true*) or not (in which case the proposition evaluates to *false*). To get the highest benefit from this generation process, attributes and associations in the extension are most interested. We notice however that usually, such metamodel constructs are not overly used in the examples for missions of MMRSs. We expect however that more complex metamodel extensions are desirable for adaptive missions, because it is desirable to explicitly model the relations between involved data/actions in the metamodel extensions. For example, if a picture of a bar code is taken by one drone and sent to the other so that it can retrieve the respective product from a warehouse, it is desirable to explicitly model this picture as data sent between two robots.

MML Extension

We define the MML as an annotated metamodel as shown in Figure 1.8. This metamodel can be extended with robot-type-specific or robot-specific features. These features now have to be defined only once, and can be used for specifying missions in the original MML as well as in the new specification language.

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 as presented in

[3]. In particular, the new Polygon class serves now as superclass of Area in the context layer of MML in [3]. This new Shape class will allow users to specify new shapes on the map, that may trigger intricate 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 (i.e., after application of the *divide*, *appr* and *cover* functions), tasks are split up into multiple *concrete* tasks, each for one robot.

- We added run-time concepts, so that specifications can be defined in terms of the current state of the mission. We added the following run-time information at the level 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. 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 at the level 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.

  Note that these concepts may change very often during the mission. We deliberately do not define whether they should be implemented by constantly updating the mission "run-time" model (corresponding to the ProMoBox run-time sublanguage), or by a function (e.g., an "in" function that, when called, gives all shapes the robot is currently in).

- TaskDependency has been removed from the MML compared to [3]. Its functionality will be subsumed by the specification language.

- We may want to leave out ControlTask, Fork and Join, because this represents functionality of the task dependency graph, which is now subsumed by the specification language. This will cause Task and SpatialTask to be merged. This change has not been applied, and should still be discussed.

The extended MML language, featuring both original MML and specification language, is shown in Figure 1.9. It is generated from Figure 1.8, using the ProMoBox approach. Note how this metamodel consists of three parts:

- on top: a variant of the original MML metamodel. This part is generated from the annotated metamodel, by leaving out run-time concepts. It is used to specify missions like in the original MML. Additionally, shapes can be defined, that can be used in the specification language. In case of an MML extension, extensions will also appear in this part;
- in the middle (shaded): the temporal language template. This is the property template directly taken from ProMoBox [7], for temporal properties. It may still be changed, to cater the specific needs of the specification language.
- at the bottom: the pattern version of the MML. This part is generated form the annotated metamodel, by including all run-time concepts, and converting to a pattern language (i.e., attributes are now of type Condition, abstract classes are concrete, and minimum multiplicities are removed). This part can be used to specify domain-specific propositions for the specification language, e.g., a robot being at a specific point, a robot performing a specific task, a task being planned for a robot, etc. In case of an MML extension, "pattern versions" of the extensions will also appear in this part.

We present some examples in abstract syntax form. Concerning concrete syntax, we can present some general ideas. Instances of the temporal language template (the middle part of Figure 1.9) are best described textually, as in [1]. Instances of the pattern language (the bottom part of Figure 1.9) can be described both graphically (as typically done with the ProMoBox approach. However, a textual syntax can be devised, where each of the associations can form a subsentence with the two attached instances. For example, "a Robot currently on a GeoCoordinate" denotes 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 are terminals), and serves as an extension of [1]:

```
Proposition ::= InstanceExpression [Association InstanceExpression]
InstanceExpression ::= 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)
```

With this grammar, patterns involving multiple links can be expressed with AndPatterns. A textual editor that directs the user in writing correct temporal specifications can be generated easily with Xtext [6], by using content assist.

Figures 1.10 to 1.12 show example temporal specifications. The MML is extended as shown in Figure 1.13 to define robot-specific tasks:

- DetectPest: scanning for a pest and in case of detection, send some coordinates;
- ReceiveCoordinates: receiving coordinates where a pest has been detected;
- Spray: spraying pesticides;
- TakePicture: taking a picture.

Note that currently, sending messages is part of the tasks, and thus is abstracted for the mission planner. In the future, we may feel the need to explicitly incorporate messages in MML (now present in BL), at the expense of making MML more complex to use. This principle can be applied in general: more detail may be exposed in MML to increase its expressiveness at the expense of simplicity.

Like the metamodel of Figure 1.9, the abstract syntax that is an instance of the temporal language template is shaded in the examples below. Instances that are named the same represent the same instance, reused in different parts of the temporal specification.

The example of Figure 1.10 represents "spraying must be preceded by a pest detection". The leftmost AtomicPattern is a forAll quantification, stating that the temporal specification must hold for all coordinates in all spray tasks. The middle AtomicPattern states the condition, saying that a robot r must have received coordinates at a point p. The rightmost AtomicPattern describes the aforementioned robot r, spraying at aforementioned point p. We feel that forAll quantification is not required for temporal specifications, because indeed, the temporal pattern must be true in all occasions. Therefore, quantification is not used in the examples below. Note that the coveredPoints links are superfluous, because if the robot is currently performing an action of a task, it must be inside the task area. Also note how currently, no distinction can be made by "performing" an action, and "finishing performing" an action. In structured English grammar, the temporal specification would be as follows (leaving out the superfluous quantification and coveredPoints link): "Globally, if a Robot r performing a Spray and r on a Coordinate p then it must have been the case that r on p and r performing a ReceiveCoordinates."

The example of Figure 1.11 represents "in a certain area, a robot can never exceed a given altitude". In structured English grammar, the temporal specification would be as follows: "Globally, it is never the case that a Robot r in a Area with name as "lowflyzone" and r on a GeoCoordinate with altitude less than 20."

The example of Figure 1.12 represents "a robot can only perform a certain task if another robot is at a certain position". In structured English grammar, the temporal specification would be as follows: "Globally, it is always the case that a Robot r1 performing a Task and a Robot r2 on a RelativeCoordinate with x as 100 and y as 200 and z as 10."

1.2.2 Familiarization with the FlyAQ platform

We used the FlyAQ platform[1] to examine how the different DSMLs are exposed to the user. We intend to extend this user interface, to support the specification language. The front-end is built with Javascript and Google Maps.

1.2.3 Risk assessment for Verification

In light of using verification for the FlyAQ platform, we have assessed the feasibility of using model checking for tasks. We have made a small demo that executes the pest control example on an area that is split into 16 points. Two drones were used: one for pest detection, and one for spraying. The drone for pest detection goes through all the points in the area, and scans them, thus representing the normal behaviour in the FlyAQ platform. The spraying drone has a request queue, which it will process in the order of receiving. The pace at which both drones travel from point to point, is not predetermined. Properties that were checked were: (response) upon detection of a pest at point 10, that point must be sprayed eventually, and (precedence) only if a pest is detected in point 10, it is sprayed. Note that a specific point is selected, because there are no quantification operators in LTL. In both cases, the model checking time went up exponentially with the number of points. While 16 points were covered quickly (0.3s), 20 points already turns out to be infeasible (90s). This was an additional factor to not use the specification language for verification.

1.2.4 Presentation on ProMoBox

A seminar was organised at the Software Engineering Division during the STSM, where the ProMoBox approach was presented to the audience in a 45 minute presentation.

1.2.5 Explaining DSM to Swaib

Time was taken during the STSM to familiarize Swaib with domain-specific modelling. Course material, presentations and exercises were exchanged, and a demo was given about the tool AToMPM, a tool for modelling and using DSMLs.

[1] http://www.flyaq.org

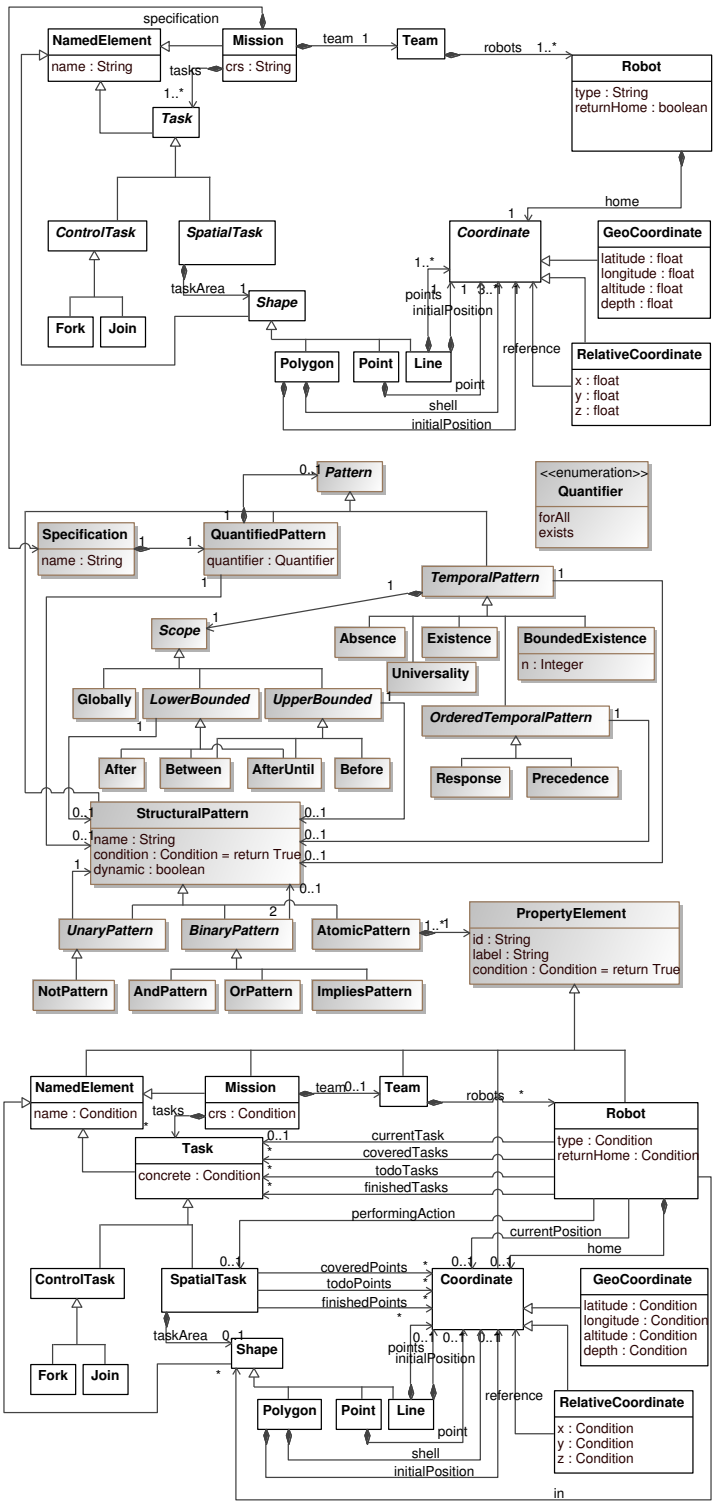

Figure 1.9: The extended MML language, generated from Figure 1.8.

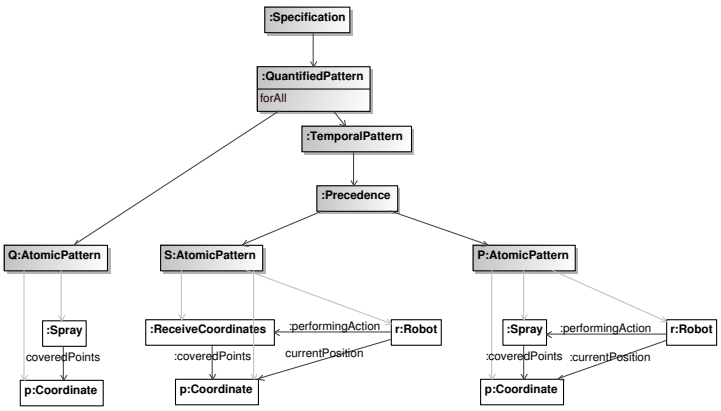

Figure 1.10: Abstract syntax of "spraying must be preceded by a pest detection".

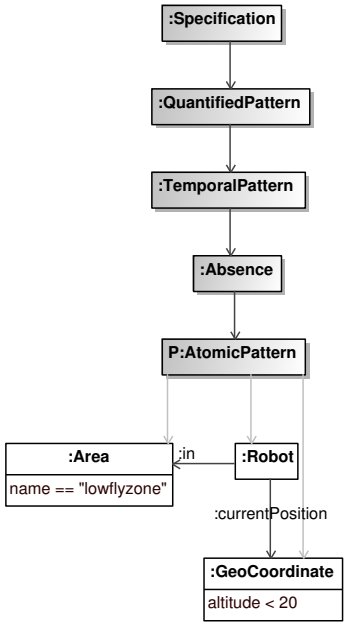

Figure 1.11: Abstract syntax of "in a certain area, a robot can never exceed a given altitude".

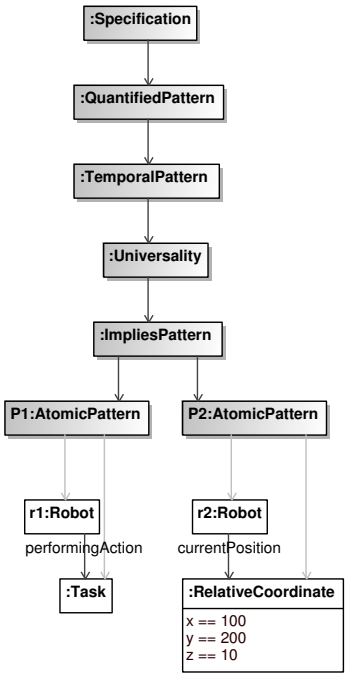

Figure 1.12: Abstract syntax of "a robot can only perform a certain task if another robot is at a certain position".

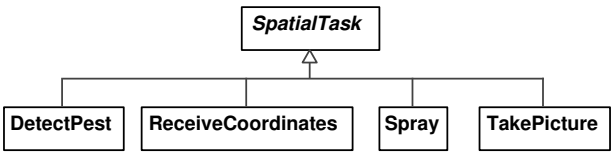

Figure 1.13: Robot-specific MML extensions.

Chapter 2

Collaboration Goals

This chapter discusses the future steps that will be taken, which result from this STSM.

2.1 Publication Target

We have committed to publish the results of this research to SEAMS 2017, International Symposium on Software Engineering for Adaptive and Self-Managing Systems. Our research topic is very relevant to this conference. Additionally, Patrizio is well embedded in the community of adaptive systems. Swaib, Patrizio and I will author this paper. Moreover, we will contact Davide Di Ruscio or another author of the FlyAQ framework, to help with the implementation. If this collaboration is possible, then this person will become co-author of the paper.

2.2 Research Plan

The main results of this STSM can be summarized as follows:

- risk assessment: we explored the possibilities of an extension of FlyAQ sufficiently to assess the impact on the FlyAQ framework;
- scoping: we defined a clear roadmap for further research, which is at the core of Swaib's PhD research;
- dissemination: research was extensively shared during the STSM, including the FlyAQ platform, ProMoBox, and DSM.

The concrete research plan consists of five phases, and should result in a publication. As this research is at the very core of his PhD research, Swaib will take the lead in further collaboration.

2.2.1 Phase 1: Extend Mission Language and Platform with Property Specifications

In this phase, the MML, RL, BL and transformation to BL are extended so that property specifications are supported. The changes are as non-intrusive as

possible. This will be mainly carried out by Swaib, in collaboration with the FlyAQ author.

2.2.2 Phase 2: Generate Property Specifications from MML

The techniques from the ProMoBox framework will be employed to generate the Property Specification language from the extended MML. This phase will be mainly carried out by Bart.

2.2.3 Phase 3: Use Case: Extend FlyAQ User Interface

The FlyAQ front-end needs to be updated, so that properties can be modelled by the domain user. This effectively means the replacement of the window for specifying the order of tasks. This will be mainly carried out by Swaib, in collaboration with the FlyAQ author.

2.2.4 Phase 4: Extend Platform with Run-time Adaptation

Run-time adaptation of robots is treated as a separate phase, as this may prove to be optional for a first publication. Nevertheless, for a publication at SEAMS, this is a mandatory contribution. This phase will be in fact extension of phase 1 to 3, and will be mainly carried out by the respective researchers.

2.2.5 Phase 5: Validation

We will validate our findings by applying them to several robot types. Current candidates are: (a) unmanned areal vehicles (the main use case), (b) underwater autonomous vehicles, and (c) a robot with two arms.

We will define a number of representative exemplary missions. Current candidates are: (1) synchronisation between two robots (e.g., starting a task after the surveillance robot is in position), (2) run-time planning, where the required information is acquired to execute a task (e.g., pest detection and spraying), and (3) replanning, focusing on a newly created task (e.g., the domain user adds a task at run-time).

The above choice may change during the course of our research. This phase will be mainly carried out by Swaib, Patrizio and Bart.

2.2.6 Phase 6: Paper Writing

The paper will be written by Swaib, Patrizio and Bart.

Bibliography

[1] Marco Autili, Lars Grunske, Markus Lumpe, Patrizio Pelliccione, and Antony Tang. Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar. *IEEE Trans. Software Eng.*, 41(7):620–638, 2015.

[2] Darko Bozhinoski, Davide Di Ruscio, Ivano Malavolta, Patrizio Pelliccione, and Massimo Tivoli. FLYAQ: enabling non-expert users to specify and generate missions of autonomous multicopters. In Myra B. Cohen, Lars Grunske, and Michael Whalen, editors, *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, pages 801–806. IEEE Computer Society, 2015.

[3] Federico Ciccozzi, Davide Di Ruscio, Ivano Malavolta, and Patrizio Pelliccione. Adopting mde for specifying and executing civilian missions of mobile multi-robot systems. *IEEE Access*, 4:6451 – 6466, 2016.

[4] Romuald Deshayes, Bart Meyers, Tom Mens, and Hans Vangheluwe. Promobox in practice : A case study on the GISMO domain-specific modelling language. In Daniel Balasubramanian, Christophe Jacquet, Pieter Van Gorp, Sahar Kokaly, and Tamás Mészáros, editors, *Proceedings of the 8th Workshop on Multi-Paradigm Modeling co-located with the 17th International Conference on Model Driven Engineering Languages and Systems, MPM@MODELS 2014, Valencia, Spain, September 30, 2014.*, volume 1237 of *CEUR Workshop Proceedings*, pages 21–30. CEUR-WS.org, 2014.

[5] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In Barry W. Boehm, David Garlan, and Jeff Kramer, editors, *Proceedings of the 1999 International Conference on Software Engineering, ICSE' 99, Los Angeles, CA, USA, May 16-22, 1999.*, pages 411–420. ACM, 1999.

[6] Moritz Eysholdt and Heiko Behrens. Xtext: implement your language faster than the quick and dirty way. In William R. Cook, Siobhán Clarke, and Martin C. Rinard, editors, *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*, pages 307–309. ACM, 2010.

[7] Bart Meyers, Romuald Deshayes, Levi Lucio, Eugene Syriani, Hans Vangheluwe, and Manuel Wimmer. ProMoBox: A framework for generating domain-specific property languages. In *Software Language Engineering*,

volume 8706 of *Lecture Notes in Computer Science*, pages 1–20. Springer International Publishing, 2014.

[8] Bart Meyers, Manuel Wimmer, and Hans Vangheluwe. Towards domain-specific property languages: The ProMoBox approach. In *Proceedings of the 2013 ACM Workshop on Domain-specific Modeling*, pages 39–44. ACM New York, NY, USA, 2013.

[9] Davide Di Ruscio, Ivano Malavolta, Patrizio Pelliccione, and Massimo Tivoli. 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, pages 45–55, New York, NY, USA, 2016. ACM.

[10] SPARC. Horizon 2020, robotics multi-annual roadmap 2016.