



New Horizons for Metamorphic Relationships in Formal Verification

Fernando Asteasuain^{1,2} 

¹ Universidad Nacional de Avellaneda, Buenos Aires, Argentina
fasteasuain@undav.edu.ar

² Universidad Abierta Interamericana - Centro de Altos Estudios CAETI, Buenos Aires, Argentina

Abstract. In this work we broadened the impact of the so called Metamorphic relationships (MR's) in the formal verification phase. We showed the potential of our behavioral framework called FVS (Feather Weight Visual Scenarios) to successfully denote metamorphic properties in diverse, complex and meaningful domains such as UAV's flying missions and operating systems for On Board Computers (OBC) for nano satellites. We employed MR's to validate behavior in a BIG-DATA context, where possible a large amount of data and information seen as traces must be verified but also a novel way to relate different goals and UAV's configurations in the context of the dynamic adaption of AUV's missions due to changes in the requirements. In addition, we explored complementary behavior as a possible source for obtaining MR's.

Keywords: Metamorphic behavior · Adaptive Systems · Temporal Planning

1 Introduction

Software engineering can be described as a set of techniques, tools and processes aiming to produce quality software while minimizing its possible flaws and errors. Typical software engineering phases include, among others, exploring and specifying system requirements, to analyze and design its features, to describe high level or architectural interactions between the system's main components, to implement it in some framework or language, and to validate and verify the expected behavior of the system.

Since it was born, Software Engineering has been evolving to cope with the challenges that new software visions and new computer software paradigms bring along. Just to mention two examples architectural aspects were incorporated as a separate and a stand-alone software engineering phase to specify high level components' interactions as the complexity of software system grow and grow [34]. As a second example, agile methodologies [10] were born to leverage the bureaucracy involved in software development processes.

Unarguable the data science big bang can be pointed out as one of the most revolutionary domains in the compute science field in the last few years. Nowadays almost any modern device can send and receive information and as a consequence a huge amount of information is out there available to be explored and analyzed. Important software areas like Machine Learning [28], Data Science [1] or BIG DATA [31] are gaining each day more relevance in order to extract, scrutinize and learn from gathered data. These areas deal with distinctive characteristics and requirements. Some of them include handling volatile, unstructured, diverse and heterogeneous tons of data, managing new ways of dealing with information flow and communication between components and a much more rigorous approach to cope with performance and availability concerns. In this context, some Software Engineering techniques need to evolve somehow to face these new elements [11, 14, 22–24, 30, 36]. Formal verification and validation has been pinpointed as one of the areas that more urgently need to be tackled. Data Science systems need to be formally verified. According to [17, 23], only two of nearly one hundred analyzed approaches addressing new software engineering methods for big data were related to formal validation. These kinds of systems have been defined as “non testable” software [17, 30] because the lack of a proper testing oracle to check their behavior. The problem can be stated as: How the new version of the system, which now includes the analyzed information, can be tested? How can the new system be checked against its expected behavior? How can the expected behavior be specified? How can the software engineer verify if the system is producing the expected outputs?

Metamorphic Relationships (or more simply MRs) [16, 33] have been recently applied as a way to handle this problematic issue. The objective of MR’s is to construct connections between behavior and data that help to verify the system, linking the old version of the system with its newer version. Probably the most iconic example is a machine learning system focusing on sentiment analysis classifying a given sentence as positive, neutral or negative. So as to verify the system MR’s are defined in the following manner: for positive sentences, a new sentence is build using synonyms of the sentence’s words. This new word should be classified as positive in the newer version of the system. Similarly, if the new sentence is build using antonyms instead, then the expected outcome should be negative. However, how to specify and define the MR’s for every system is an extremely arduous and error-prone task. Also, defining how many MR’s are enough is also a tricky point [17].

In [3] we explore FVS (Feather Weight Visual Scenarios) [5–7] as an specification language to model and specify MR’s. FVS is a very simple, graphical and declarative language which holds a very flexible and expressive power notation. We now build on the top of that work expanding the horizon for MR’s in the formal verification phase. In this work we applied MR’s in meaningful domains: Adaptive Systems and Temporal Planning for UAV’s (Unmanned Aerial Vehicle) [25, 38–40], and Operating Systems for OnBoard Computers (OBC) for nano satellites such as CubeSats [32]. This shows the potential of MR’s as a powerful technique to formally verify a system and the applicability of FVS as an speci-

fication language for MR's. We employed MR's in two orthogonal styles: in the usual manner as a way to build a verification oracle but also a way to relate two different goals and configuration for UAV's, which is a very useful feature when dealing with Adaptive Systems. Finally, we inspected complementary behavior as a valuable source to define MR's. This is a very reachable feature in FVS since it can be automatically generated [8, 13].

1.1 Summary

The main contributions of this work can be outlined as:

- Showing the potential of FVS to describe MR's for UAV's Adaptive Systems.
- Showing the potential of FVS to describe MR's for UAV's Temporal Planning.
- Showing the potential of FVS to describe MR's for nano satellites' OBC.
- Exploring the possibility to analyze different goals and UAV's configurations employing MR's
- Automatic generation of MR's reasoning about complementary behavior.

The rest of this work is organized as follows. Section 2 presents FVS, and some basic notions of the techniques employed such as Adaptive Systems, Temporal Plannings and OBC's. The next three Sections describe all the case of studies: Sect. 3 is focused on FVS and MR's for Adaptive Systems, Sect. 4 deals with Temporal Planning and Sect. 5 handles MR's for a particular OBC operating systems. Section 6 presents some observations and final remarks regarding the analyzed examples. Finally, Sect. 7 outlines related work and future lines of research that may continue this work whereas Sect. 8 concludes.

2 Background

In this section we give some preliminaries notions about FVS, Adaptive Systems, Temporal Planning and OBC.

2.1 FVS

We now briefly introduce FVS main features. The reader is referred to [5] for a formal characterization of the language. FVS is a graphical language based on scenarios. Scenarios are partial order of events, consisting of points, which are labeled with a logic formula expressing the possible events occurring at that point, and arrows connecting them. An arrow connecting two points simply indicates a precedence relationship. The events occurring at the origin of arrow occur before the events occurring at the destiny point of the arrow. For instance, in Fig. 1-(a) *A*-event precedes *B*-event. Sometimes is necessary to refer to the very next or previous occurrence of an event. This is shown graphically with a double ended arrow (see Figs. 1-(b) and 1-(c)). Precedence relationship can be labeled to restrict behavior between events. For example, consider the FVS scenario in Fig. 1-d. This scenario is stating that an *A*-event precedes *B*-event

such that C-event does not occur between them. FVS features aliasing between points. Scenario in Fig. 1-e indicates that a point labeled with A is also labeled with $A \wedge B$. It is worth noticing that A -event is repeated on the labeling of the second point just because of FVS formal syntax. Finally, two special points are introduced as delimiters of the language: a big black circle representing the beginning of the execution of the system and rounded version standing for the end of an execution (see Fig. 1-f).

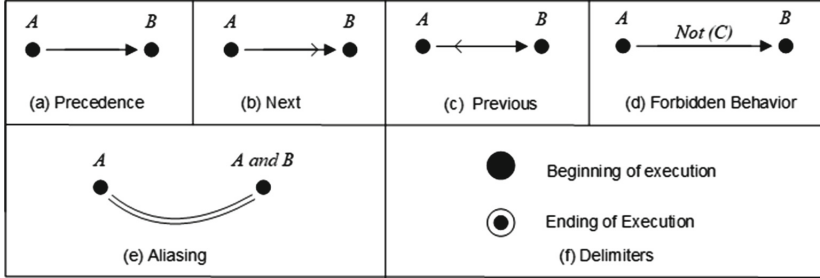


Fig. 1. Basic Elements in FVS

We now introduce the concept of FVS Rules, a fundamental constructor of the language. FVS Rules take the form of an implication: an antecedent scenario and one or more consequent scenarios. They work following this concept: whenever a trace “matches” a given antecedent scenario, then it must also match at least one of the consequent ones. The antecedent is a common substructure of all consequents, enabling complex relationship between points in antecedent and consequents: our rules are not limited, like most triggered scenario notions, to feature antecedent as a pre-chart where events should precede consequent events. Thus, rules can state expected behavior happening in the past or in the middle of a bunch of events. Graphically, the antecedent is shown in black, and consequent ones in grey. Since a rule can feature more than one consequent, elements which do not belong to the antecedent scenario are numbered to identify the consequent they belong to. An example is shown in Fig. 2. The rule describes a requirement for the FreeRTOS¹ operating system used in OBC: the system must start in the *Boot* phase of the operating system, and not in the *Scheduling* or the *ProcessingTasks* phase.

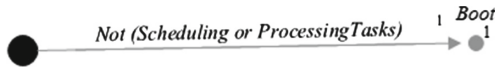


Fig. 2. An FVS rule example

¹ <https://www.freertos.org/>.

We now introduce the concept of anti-scenarios. Anti-scenarios can be automatically generated from rule scenarios. Roughly speaking, they provide valuable information for the developer since they represent a sketch of how things could go wrong and violate the rule. The complete procedure is detailed in [8, 13], but informally the algorithm obtains all the situations where the antecedent is found but none of the consequents is feasible. For example, a possible anti scenario for the previous rule in Fig. 2 is shown in Fig. 3. In this anti scenario a *Scheduling* event or an *ProcessingTasks* event occur prior the occurrence of a *Boot* event, thus violating the rule in Fig. 2.

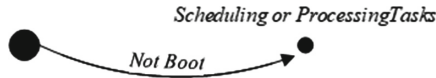


Fig. 3. An anti-scenario example

2.2 Adaptive Systems

Adaptive Systems can be defined as systems that can change their behaviour to account for conditions that were not foreseen at design time [40]. There is a tremendous challenge involved in these kind of systems: goals and configuration of the system must be able to change dynamically. In order to fulfill this objective a meticulous and rigorous architectural approach must be carefully designed. Different layers of abstraction are usually introduced to produced satisfactory results. Lower layers deal with ordinary events while upper layers handle the non-trivial transition going from the old specification to the new specification. A clear example of such approaches is described in [40], presenting a hybrid control architecture that supports runtime swapping of plans, and runtime binding and unbinding of hybrid components.

In that work a motivating example is given. Suppose a UAV is initially on a photographic mission over a certain region. Goals and configuration in the form of temporal logics are provided so that the controller of the UAV can carry on with its mission successfully. While the mission is on course, a missing person alert is issued. If the UAV was designed as an adaptive system, its mission objectives can be dynamically modified without the need for the UAV to return to the central base to be re started.

2.3 Temporal Planning

Temporal planning [39] guarantees that a certain automated device such as robot or an UAV will succeed in its task as long as certain explicit and implicit assumptions about the robot's operating environment, sensors, and capabilities hold. A

given plan can fail if any of the assumptions are violated at runtime. An illustrative example is facilitated in [39]. The system is in charge of an UAV taking photographs in a certain region. A possible planning assumption is that requesting a photograph upon flying into a discrete region will be actually taken before exiting that region. Another possible assumption is that a photograph will completely cover a discrete region if the plane's roll and pitch are within certain bounds. These conditions are dynamically verified in [39] employing a so called monitor that checks that the specified conditions are met during execution.

2.4 OBC and Nano Satellites

In the past ten years the nanosatellite industry grow by a factor of 10x, from as few as 20 satellites in 2011 to nearly 200 in 2019, and it is estimated that 1800 to 2400 nano/microsatellites will need to be launched over the next 5 years [18, 38]. In particular, CubeSats, have turned out lately as one of the most popular microsatellites [32]. The CubeSat program was initiated at Stanford University in 1999 for the purpose of building a low-cost and low-weight satellite.

There are a plethora of different domains where CubeSats satellites are directly involved: communications, earth remote sensing, space tethering, biology, astronomy, space weather exploration and planetary science [32], just to mention a few.

On BOARD Computers (OBC) must be carefully designed, planned, developed and verified. For example, battery consumption and other resources are extremely important since the services the satellites provide heavily rely on them. An OBC is in charge of the all the satellites services, it monitors them to ensure their health, and also monitors the status of all the subsystems. An OBC also interacts with the ground station to send the required telemetry data and satellite status [19].

One of the most important aspects to decide when developing a OBC is the operating system to be installed on it. One of the most widely operating systems used is FreeRTOS. As described on its website, FreeRTOS, originally created as a real time operating system for micro controllers, is distributed freely under the MIT open source license, and includes a kernel and a growing set of IoT libraries suitable for use across all industry sectors.

In [4] the formal verification of the FreeRTOS operating system using FVS as the specification language is fully provided.

3 FVS and Metamorphic Relationships(MRs) for UAV's Missions Adaptation

In this section we define MRs for different planning goals and behavior for UAV's missions. The analyzed system is based on the example fully described in [40]. In that work, several goals and UAV's configurations are properly defined so that the artifact can adapt to new missions in a dynamic way. In this example, the UAV serves as a delivery vehicle transporting packages among some locations. In

particular, the mission for the UAV is the following. There are three locations, namely A , B and C ; and three packages to be delivered: $P1$, $P2$ and $P3$. Package $P1$ must be delivered to C from A , $P2$ to A from C and finally, $P3$ to C from B . Additionally, it is required that the UAV must not move between locations without a package to preserve a minimum weight requirement. An initial plan obtained to achieve UAV's goals is the following: Starting from location A , the it grabs $P1$ and goes to location B , it grabs $P3$ and then carrying $P1$ and $P3$ goes to location C and releases $P1$ and $P3$ there. Finally, the UAV grabs $P2$, goes to location A and releases $P2$ there.

We first model this initial behavior in FVS and then we define MR's relating old and new plans later in the next subsection. Following the vocabulary introduced in [40] we employ some of the followings events in the specification: *atLocation* (the UAV is the mentioned location), *grab P_i* (the UAV grabs package P_i) or *release P_i* (the UAV releases package P_i).

The first two rules (depicted in Fig. 4) defines the expected behavior for transporting package $P1$. These FVS rules say that packaged $P1$ must be grabbed by the UAV when arriving at location A , and must be released when arriving to location C .

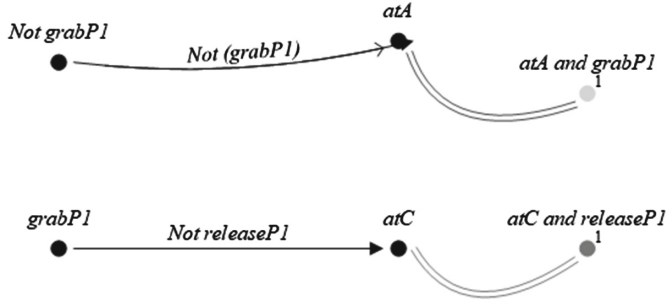


Fig. 4. FVS rules modeling UAV delivery mission for $P1$ packages

Similarly, rules for packages $P2$ and $P3$ are show in Fig. 5.

Now we specify the requirement that establish that the UAV must be always transporting at least one package. The rule in Fig. 6 shapes the behavior starting from location A and ending in location B or C . We use the meta event P_i standing for any type of packages $P1$, $P2$ or $P3$. In few words, the rule says that in every flight starting from location A carrying an package, the UAV must arrive in a different location without releasing it. Two analogues rules using location B and location C as the origin point should be added to complete the requirement specification.

3.1 MRs for UAV's Adaptation to New Plans

When new requirements are introduced in the UAV's missions' new properties, new planning strategies and new UAV configurations may be needed. We now

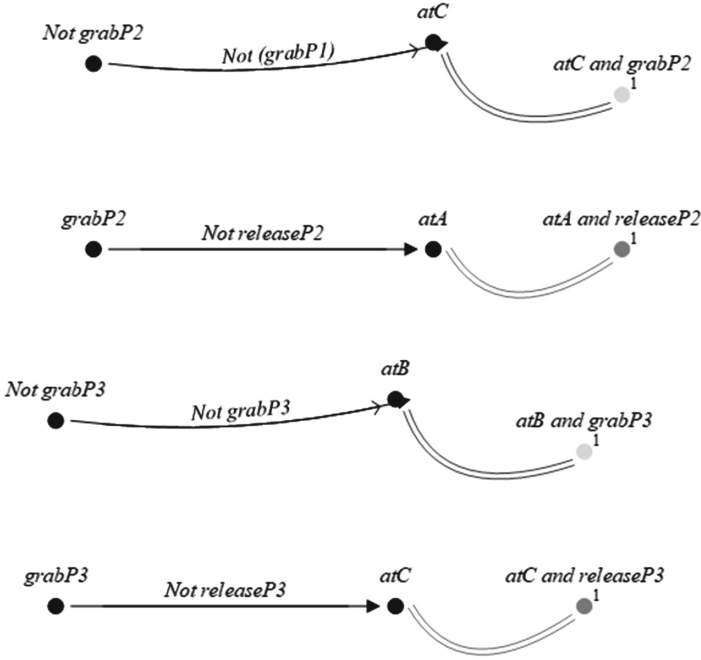


Fig. 5. FVS rules modeling UAV delivery mission for P2 and P3 packages

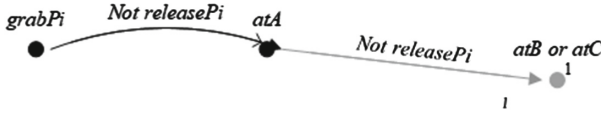


Fig. 6. The UAV must be always carrying a package

define MR's establishing well founded relationships between different plans and behavior. Following the terminology introduced in [40] we use events as *OldSpec* and *NewSpec*, representing the old specification (prior the introduction of the new requirements) and new specification (the new configuration for the UAV).

For example, a new location D can be introduced in the system described earlier, with new delivering instructions: package $P1$ must be carried from location A to location D and package $P3$ from B to D . Instructions for packages $P2$ remain unchanged. The new planning for the UAV is the following: Starting from location A , grabs $P1$ and goes to location B , grabs $P3$, then with $P1$ and $P3$ goes to location C , it grabs $P3$ and flying with the three packages arrives at D where $P1$ and $P3$ are released. Finally, the UAV grabs $P2$, goes to location A and releases $P2$ there.

These changes imply that some rules shaping the behavior for translating packages must be modified while others should not be altered. In particular, the new specification implies that new rules for packages $P1$ and $P3$ must be

introduced while rules for $P2$ suffer no changes. We introduce MR's relating old and new specification. Roughly speaking, if a behavioral property P is changed introducing a new property P' , then only P' is valid in the new specification. If P is not modified, then P is valid in the new specification. This general scheme is described in Fig. 7.

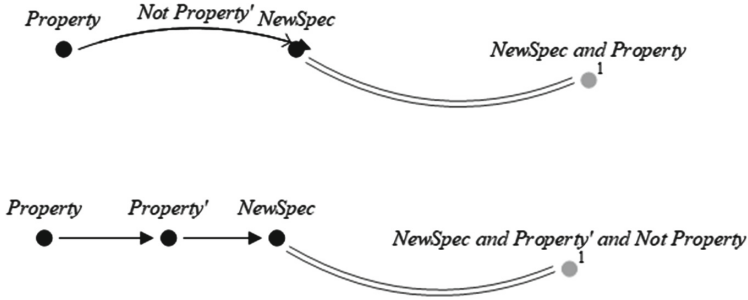


Fig. 7. MRs relating old and new specifications

As explained in [40] in some cases the new specification can not be started immediately due to consistency issues between old and new goals. For example, a statement saying that the UAV artifact can not carry more than two packages due to weight requirements can affect the new planning since three packages are carried out together at some point in the new specification. This is solved introducing a delay between the moment the old specification is dropped and the new specification takes charge. In this delay status the conditions shaping the behavior of the UAV's flight are relaxed so that, for example, a package is released in the next location, therefore waiting until the new specification can be loaded. This can be modeled in FVS as a MRs between specifications as shown in Fig. 8.

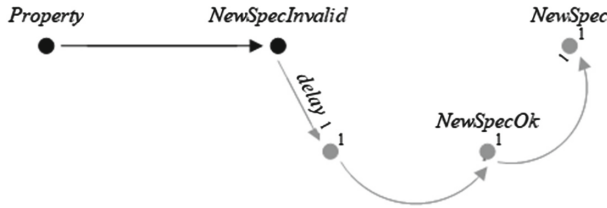


Fig. 8. MRs relating delays between specifications

4 FVS and Metamorphic Relationships(MRs) for UAV's Temporal Planning

In this section we define MRs in a more traditional way, employing FVS as a BIG DATA oracle for traces verification. The considered system is based also in defining specifications for UAV's flights, but in this case we focus on temporal planning rather than dynamically switching between specifications like it was shown in the previous Section. As detailed in [39], temporal task planning guarantees a robot will succeed in its task as long as certain explicit and implicit assumptions about the robot's operating environment, sensors, and capabilities hold. The analyzed system in this Section is based on the example introduced in [39]. An UAV artifact must provide surveillance and search and rescue services within a certain region, but considering a non fly zone. Two conditions are monitored in [39]: 1) avoid entering the non fly zone, and, if entered, the degree of violation as the distance inside the no-fly zone must be measured and 2): to assure that the area claimed to be covered by the UAV is actually covered, giving no space to measuring errors to occur. Given the specification given in [39] for the UAV's expected behavior we introduce two simple MR's. The objective is very simple: if a target is classified as being in a non fly zone, we change its coordinates so that its included in the fly zone. Similarly, if a person is detected we change the conditions so that the person is not included in the next version of the system. This follows a classic technique to obtain MR's: cases with similar conditions should belong in the same category and cases with opposite conditions should belong in opposite categories. In this Section we focus only in opposite behavior but analogous rules can be introduced considering also similar behavior. Two FVS rules are shown in Fig. 9 modeling two MR's for the UAV's temporal specification. The rule in the top of Fig. 9 says when the position of the UAV is altered (changing not being in the non fly zone to being in the fly zone) then the new result should be ruled out as a fly zone violation. Similarly, rule in the bottom of the Fig. 9 alters an image to that a person is no longer detected.

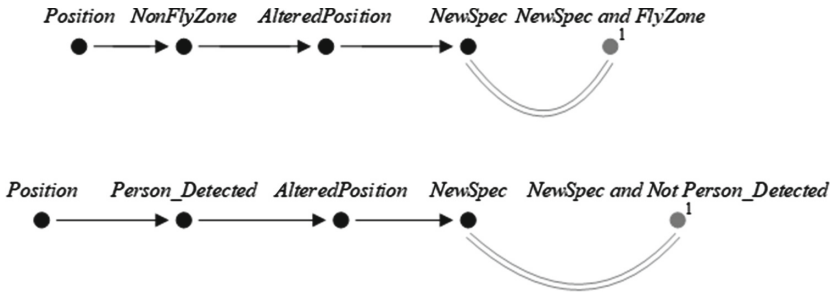


Fig. 9. MRs for temporal planning

5 FVS and Metamorphic Relationships(MRs) for OBC Operating Systems

In this section we build some MR's in a totally different domain: operating systems for OBC designed for nano satellites missions. The behavior of the FreeRTOS Operating System employing FVS as the specification language is fully detailed in [4]. In this case we build MR's considering the exact opposite behavior of the rules. Since FVS can automatically generate anti-scenarios for every rule, MR's can be automatically generated. This clearly constitute an attractive feature. We took some of the rules specified in [4] and automatically obtain MR's by generating anti-scenarios for every rule. For example, the FVS scenario in the top of Fig. 10 exhibits a behavior that should be classified as an error: the operating system does not start in the boot phase. This MR is an anti scenario for the rule demanding that the system should indeed starts in the boot phase. The scenario in the middle of Fig. 10 is another anti-scenario and should be considered as an error: the operating systems starts to process tasks without entering the scheduling phase. Finally, the scenario in the bottom of Fig. 10 illustrates an scenario which violates the priority-based scheduling policy of the operating system.

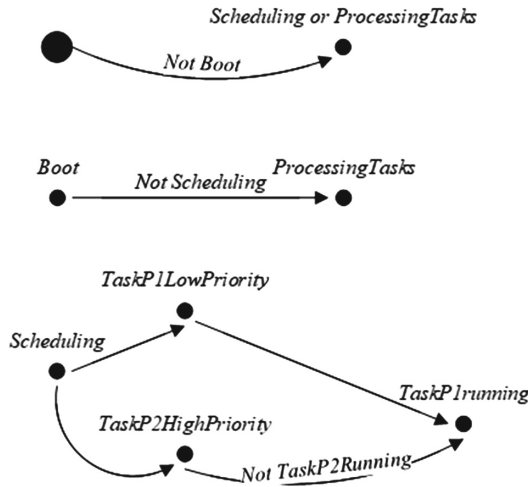


Fig. 10. MRs as FVS anti-scenarios

6 Discussion

In Sects. 3, 4 and 5 we illustrate how FVS can be used as the specification language to successfully denote metamorphic behavior in diverse, complex and

meaningful domains such as UAV's flying missions and operating systems for On Board Computers in nano satellites design. We also employed MR's in two flavours: in Sect. 3 we specify MR's to relate different goals and UAV's configurations in the context of a challenging area such as the dynamic adaption of AUV's missions due to changes in the requirements.

In Sect. 4 we continue exploring UAV based system, but we built MR's in a more traditional way: to validate behavior in a BIG-DATA context, where possible a large amount of data and information seen as traces must be checked.

Finally, in Sect. 5 we moved to validate OBC software. In particular we focused on one popular operating called FreeRTOS. We took advantage of FVS's possibility to automatically generate anti-scenarios to build MR's that behave in the exact opposite way to the rules defining the expected behavior of the operating systems. Exploring the opposite behavior of a requirement is an useful way to obtain MR's.

Based on these experiments we can conclude that FVS is an appealing framework to specify metamorphic behavior. FVS expressive power and flexibility is shown in significative domains, from UAV's flying mission to OBC operating systems. In addition, we showed how to denote MR's in two ways: to relate different UAV's configurations and to validate outputs where the vast amount of information make it hard to define. MR's can be automatically generated as anti-scenarios. When manually defining MR's as FVS rules all the desirable characteristics of a declarative and graphical notation shine: behavior can be intuitively explored and analyzed by gathering the desired information from the graphical scenarios. Finally, consistency checking of requirements and MR's can be easily checked combining FVS with model checkers. For example, employing tools like GOAL [37] or [26] we can perform behavioral synthesis [12] and classic behavioral verification [5].

7 Related and Future Work

A very appealing technique to define MR's for BIG DATA systems is presented in [17]. The foundation of this technique is to specify an initial set of MR's and start an iterative process on which MR's are refined. The process is automated, sound and correct. The experimentation of the research is focused on a BIG DATA service implementation. This service, called Cell Morphology Assay (CMA) was designed for modeling and analyzing 3D cell morphology and mining morphology patterns extracted from diffraction images of biology cells. The implementation is able to categorize a huge amount of biology cells images. We believe we can replicate this procedure using FVS, since our language support the refinement operation [3], which constitutes one possible future research line to continue our work. Contrary to FVS, the possibility to check consistency between MRs is not available in this approach.

Other approaches focused on metamorphic properties are [2, 16, 27]. These options are mostly focused on implementation details such as the actual framework tool to deploy the tests. Our approach is addressing a previous phase which

is the exploration and specification of MRs. However, one possible combination is to graphically denote tests as FVS scenarios and trying to explore automatic generation of test cases.

Adaptive System in UAV's control missions is tackled on [40]. This work exhibits how controller synthesis can be used both to guarantee correct transitioning from the old to the new mission goals including architectural reconfiguration to handle new software actuators and sensors if necessary. Adaptation can occur at various levels of abstraction to respond to many different kinds of changes. In particular, [40] deals with mobile robot adaptation that include dynamic responses to changes in the high-level goals that the robot must achieve. Goals and configurations are specified using temporal logics. A very interesting framework based on hybrid components is in charge of performing all the necessary architectural changes in the system to adapt to the new requirements. We were able to reproduce and specify the behavioral specification for some examples of adaptive systems described in [40]. We denote a set of MR's that could be used to relate, explore, and verify goals and requirements from the old and the new specification. We believe this can be a plus in this research line.

Temporal planning for UAV's is addressed on [39]. A monitoring process is implemented to verify that implicit and explicit assumptions of the environment are not violated. If some violation of the requirements occur, it is shown that under certain conditions some corrective actions can be taken to re enter in safe state. The approach can be classified under the Runtime Verification label, a formal methods technique that studies how to synthesize monitors from high-level formal specifications of complex properties [21,39]. In our experimentation we showed how FVS can be used to model MR's for temporal planning and describe how them can be used as a trace oracle and validation in BIG DATA systems.

The specification in [39] is based on the HLOLA specification language [15, 20]. HLOLA is a very rich, expressive and powerful language with the possibility to realize automatic validation tasks. It would be interesting trying to combine this language with FVS's graphical flavour.

Among other future lines of research to continue this work we would like to take one step further FVS's specifications and put them in action in some simulated or real scenarios as described in [39,40]. For example, a possible tool to consider is ArduPilot Software-In-The-Loop (SITL) UAV simulator as in [9,35].

We would also like to explore the interaction between FVS and other tools. For example, [2] extracts MR in runtime checking the different paths in the execution tree. In this context, FVS scenarios could be used as a monitor in the sense given by the model checking techniques. Finally, we are exploring the possibility to interact with automatic code and test generators like [29].

8 Conclusions

In this work we deepened the impact of MR's for formal verification. Besides employing them as a trace oracle for formal validation in BIG DATA we showed

how they can be used to explore and reason about different specifications. This is a powerful feature in order to produce the most accurate possible specification of the expected behavior of the system under analysis.

We showed the potential of FVS as a specification language to denote MR's analyzing non trivial examples in cutting-edge and heterogeneous domains such as operating systems for OBC for nano satellites and behavioral specifications for UAV's. In addition, the possibility to automatically generate complementary behavior is a distinctive feature of our approach.

The satisfactory results delineated in this work aim to further blend the interaction between MR's, formal verification and FVS as the underlying specification language and framework.

References

1. Agarwal, R., Dhar, V.: Big data, data science, and analytics: the opportunity and challenge for is research (2014)
2. Asrafi, M., Liu, H., Kuo, F.C.: On testing effectiveness of metamorphic relations: a case study. In: 2011 Fifth International Conference on Secure Software Integration and Reliability Improvement, pp. 147–156. IEEE (2011)
3. Asteasuain, F.: A flexible and expressive formalism to specify metamorphic properties for big data systems validation. In: CACIC, pp 282–291 (2022). ISBN 978-987-1364-31-2
4. Asteasuain, F.: Formalizing operating systems for nano satellites on board computers. In: CONAIIISI (2022)
5. Asteasuain, F., Braberman, V.: Declaratively building behavior by means of scenario clauses. *Requirements Eng.* **22**(2), 239–274 (2016). <https://doi.org/10.1007/s00766-015-0242-2>
6. Asteasuain, F., Caldeira, L.R.: A sound and correct formalism to specify, verify and synthesize behavior in BIG DATA systems. In: Pesado, P., Gil, G. (eds) *Computer Science. CACIC 2021. CCIS*, vol. 1584, pp. 109–123. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-05903-2_8
7. Asteasuain, F., Calonge, F., Dubinsky, M., Gamboa, P.: Open and branching behavioral synthesis with scenario clauses. *CLEI E-J.* **24**(3), 1–20 (2021)
8. Asteasuain, F., Calonge, F., Gamboa, P.: Exploiting anti-scenarios for the non realizability problem. In: Pesado, P., Arroyo, M. (eds.) *CACIC 2019. CCIS*, vol. 1184, pp. 157–171. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-48325-8_11
9. Baidya, S., Shaikh, Z., Levorato, M.: Flynetsim: an open source synchronized UAV network simulator based on ns-3 and ardupilot. In: *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 37–45 (2018)
10. Beck, K., et al.: Manifesto for agile software development (2001)
11. Bellettini, C., Camilli, M., Capra, L., Monga, M.: Distributed CTL model checking using Mapreduce: theory and practice. *CCPE* **28**(11), 3025–3041 (2016)
12. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'Ar, Y.: Synthesis of reactive (1) designs (2011)
13. Braberman, V., Garbervestky, D., Kicillof, N., Monteverde, D., Olivero, A.: Speeding up model checking of timed-models by combining scenario specialization and

- live component analysis. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 58–72. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04368-0_7
14. Camilli, M.: Formal verification problems in a big data world: towards a mighty synergy. In: ICSE, pp. 638–641 (2014)
 15. Ceresa, M., Gorostiaga, F., Sánchez, C.: Declarative stream runtime verification (hLola). In: Oliveira, B.C.S. (ed.) APLAS 2020. LNCS, vol. 12470, pp. 25–43. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64437-6_2
 16. Chen, T.Y., Cheung, S.C., Yiu, S.M.: Metamorphic testing: a new approach for generating next test cases. arXiv preprint [arXiv:2002.12543](https://arxiv.org/abs/2002.12543) (2020)
 17. Ding, J., Zhang, D., Hu, X.H.: A framework for ensuring the quality of a big data service. In: 2016 SCC, pp. 82–89. IEEE (2016)
 18. Doncaster, B., Williams, C., Shulman, J.: 2017 nano/microsatellite market forecast. SpaceWorks Enterprises Inc., Atlanta, GA, Technical report (2017)
 19. Fernandez, L., Ruiz-De-Azua, J.A., Calveras, A., Camps, A.: Assessing Lora for satellite-to-earth communications considering the impact of ionospheric scintillation. IEEE access **8**, 165570–165582 (2020)
 20. Gorostiaga, F., Sánchez, C.: HLola: a very functional tool for extensible stream runtime verification. In: TACAS 2021. LNCS, vol. 12652, pp. 349–356. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72013-1_18
 21. Havelund, K., Roşu, G.: Synthesizing monitors for safety properties. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 342–356. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46002-0_24
 22. Hummel, O., Eichelberger, H., Giloj, A., Werle, D., Schmid, K.: A collection of software engineering challenges for big data system development. In: SEAA, pp. 362–369. IEEE (2018)
 23. Kumar, V.D., Alencar, P.: Software engineering for big data projects: domains, methodologies and gaps. In: IEEE BIGDATA, pp. 2886–2895. IEEE (2016)
 24. Laigner, R., Kalinowski, M., Lifschitz, S., Monteiro, R.S., de Oliveira, D.: A systematic mapping of software engineering approaches to develop big data systems. In: SEAA, pp. 446–453. IEEE (2018)
 25. Luckcuck, M., Farrell, M., Dennis, L.A., Dixon, C., Fisher, M.: Formal specification and verification of autonomous robotic systems: a survey. ACM Comput. Surv. (CSUR) **52**(5), 1–41 (2019)
 26. Magee, J., Kramer, J.: State models and Java Programs. Wiley, Chichester (1999)
 27. Mayer, J., Guderlei, R.: An empirical study on the selection of good metamorphic relations. In: 30th Annual International Computer Software and Applications Conference (COMPSAC 2006), vol. 1, pp. 475–484. IEEE (2006)
 28. Mitchell, T.M., Mitchell, T.M.: Machine Learning, vol. 1. McGraw-hill New York (1997)
 29. Niaz, I.A., Tanaka, J.: Code generation from UML statecharts. In: Proceedings of the 7th IASTED International Conference on Software Engineering and Application (SEA 2003), Marina Del Rey, pp. 315–321 (2003)
 30. Otero, C.E., Peter, A.: Research directions for engineering big data analytics software. IEEE Intell. Syst. **30**(1), 13–19 (2014)
 31. Pramanik, S., Bandyopadhyay, S.K.: Analysis of big data. In: Encyclopedia of Data Science and Machine Learning, pp. 97–115. IGI Global (2023)
 32. Saeed, N., Elzanaty, A., Almorad, H., Dahrouj, H., Al-Naffouri, T.Y., Alouini, M.S.: Cubesat communications: recent advances and future challenges. IEEE Commun. Surv. Tutor. **22**(3), 1839–1862 (2020)

33. Segura, S., Fraser, G., Sanchez, A.B., Ruiz-Cortés, A.: A survey on metamorphic testing. *IEEE Trans. Software Eng.* **42**(9), 805–824 (2016)
34. Shaw, M., Garlan, D.: *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Inc., Upper Saddle River (1996)
35. de Sousa Barros, J., Oliveira, T., Nigam, V., Brito, A.V.: A framework for the analysis of UAV strategies using co-simulation. In: 2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC), pp. 9–15. IEEE (2016)
36. Sri, P.A., Anusha, M.: Big data-survey. *Indonesian J. Electr. Eng. Inform. (IJEEL)* **4**(1), 74–80 (2016)
37. Tsay, Y.-K., Chen, Y.-F., Tsai, M.-H., Wu, K.-N., Chan, W.-C.: GOAL: a graphical tool for manipulating büchi automata and temporal formulae. In: Grumberg, O., Huth, M. (eds.) *TACAS 2007*. LNCS, vol. 4424, pp. 466–471. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71209-1_35
38. Ziegert, S., Wehrheim, H.: Temporal plans for software architecture reconfiguration. *Comput. Sci.-Res. Dev.* **30**, 303–320 (2015)
39. Zudaire, S., Gorostiaga, F., Sánchez, C., Schneider, G., Uchitel, S.: Assumption monitoring using runtime verification for UAV temporal task plan executions. In: 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 6824–6830. IEEE (2021)
40. Zudaire, S.A., Nahabedian, L., Uchitel, S.: Assured mission adaptation of UAVs. *ACM Trans. Auton. Adapt. Syst. (TAAS)* **16**(3–4), 1–27 (2022)