An Architectural Model of a Software Module for Piloting UAV Constellations

Marko Mijač, Boris Tomaš, Zlatko Stapić

University of Zagreb, Faculty of Organization and Informatics, Varaždin, Croatia

***Abstract* – Technology behind unmanned aircraft vehicles (UAVs) has been rapidly advancing in the recent years. This propelled the adoption rate and widened the range of UAV uses, such as in sport activities, cartography, surveillance, transportation, and military. While using single, individually controlled UAVs is already very common, using UAV constellations is becoming increasingly popular due to advantages in flexibility, scalability and sensing power. However, the use of UAV constellations remains limited to very specific applications due to number of challenges involved. One of the challenges is the lack of generic software system that would allow controlling UAV constellations in different types of flight missions.**

In order to address this problem, we propose an architectural model of a software module for UAV constellation piloting.

***Keywords* – UAV, drone, piloting software.**

1. Introduction

In the last couple of years technology behind unmanned aircraft vehicles (UAVs) has significantly improved, making UAVs not only more capable and versatile, but also easily accessible even for personal use.

DOI: 10.18421/TEM114-08
https://doi.org/10.18421/TEM114-08

Corresponding author: Marko Mijač,
University of Zagreb, Faculty of Organization and Informatics, Varaždin, Croatia.
Email: mmijac@foi.hr

Received: 07 August 2022.
Revised: 14 September 2022.
Accepted: 20 September 2022.
Published: 25 November 2022.

This, as expected, resulted in dramatic increase in use of UAVs for wide range of purposes. Some of these include multimedia, sport activity, cartography, surveying, asset inspections, payload carrying and transportation, agriculture, search and rescue, firefighting, military reconnaissance [1] and combat [2]. While in all these situations a single, individually controlled UAV can be of use, multiple UAVs organized in a constellation can ensure scalability, flexibility, and more sensing power.

In case of single UAV, unmanned aircraft system (UAS) usually consists of aircraft itself, control equipment for that particular aircraft (provided by manufacturer), and the human pilot. However, in case of UAV constellation we deal with multiple (possibly different) UAVs which all have to be piloted as a single (albeit rather complex) aircraft by a single pilot. In this case, a number of significant challenges arise. For example, if UAV localization relies solely on GNSS, satellite localization error (around 2 m) [3] makes it difficult to provide precise coordination and collision avoidance between individual UAVs in a swarm. To compensate for GNSS inaccuracy, UAVs often use optical localization using ground reference points [4], or the so-called Real-Time Kinematic [5] which augments accuracy using RF station as a radio reference point. Such approaches can reduce localization error to a level of centimetres, but a reference point has to exist. Additional problem is a significant level of RF noise that multiple UAVs in a constellation generate. This can be mitigated using various technologies already in use in wireless networking, as well as with direct UAV-to-UAV communication (with adaptive output power) [6]. Energy consumption is also recognized as one of the challenges, and it can be optimized among other techniques, by proper constellation control and management [7]. Even when these low level challenges are addressed, there still remains a challenge of executing flight missions with UAV constellations acting as a single entity. For example, maintaining formation and avoiding collisions during such flight missions requires performing complex in-flight calculations which coordinate and synchronize UAVs within constellation [8]. Because this is very challenging to do, current software systems for controlling UAV constellations are limited to specific uses, such as surveillance (e.g. monitor and measure an area) and artistic/entertainment applications (e.g. as a replacement for traditional fireworks where each UAV acts as a single pixel) [9]. Not only we have no generic software solution for piloting UAV constellation, but implementations of specific software solutions are done from scratch or with very little systematic reuse. However, despite evident variability in specific software solutions and use cases, we believe there is possibility (and need) to extract commonalities shared by all of them. This can be achieved by capturing design-level commonalities and embedding them into architectural model of a software module for UAV constellation piloting. The proposed architecture can then be used as a high-level blueprint for detailed design and implementation of a software module.

In order to focus our efforts and be unambiguous in communicating our ideas, we formalize our paper's goal: *Facilitate the process of designing and implementing software module for piloting UAV constellation by designing architectural model.* We will attempt to answer following research questions during the very process of creating UAV architecture:

RQ1 - *What functional requirements should software module meet to provide basic UAV constellation piloting capabilities?*

RQ2 – *Which design decisions should be considered when designing UAV constellation piloting solution?*

The remainder of the paper starts with elaboration of methodological approach taken to systematically devise a solution (section 2). The problem-solving process and the solution itself are described in section 3. The challenges met and the results achieved are discussed in section 4. Finally, in section 5 conclusions and possibilities for future work were offered.

2. Methodology

Since the research in this paper is directed at providing innovative solution to a practical problem, it can be characterized as a solution-seeking type of research [10]. Such type of research requires conducting systematic and scientifically rigorous problem-solving process. One of the established ways in software engineering to achieve that is design science [11]. The main outputs of design science research are a solution artifact itself, new knowledge generated during artifact construction and use, and knowledge embedded into the artifact. To ensure proper implementation of design science process we followed methodological framework for design science research [12] (see Figure 1).

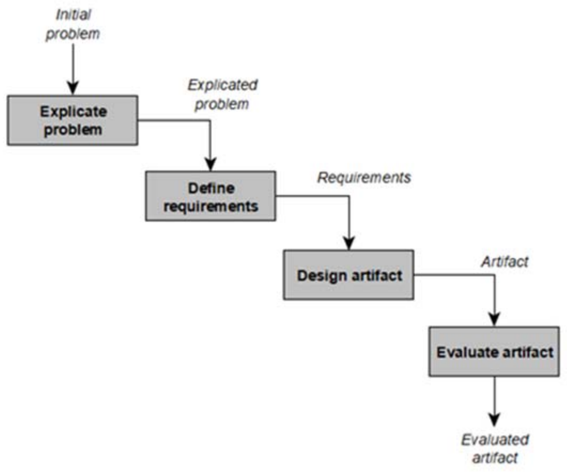

Figure 1. Design science process

The first activity (Explicate problem) is focused on defining the problem of managing flight missions for UAV constellations, putting it into a broader context, and demonstrating its relevance in both scientific and practical sense. This activity relies on conducted literature review reported in introductory section.

When properly explicated, problem acts as an input to the second activity (Define requirements). The activity starts with brief description of the solution artifact in terms of its fundamental concepts and characteristics, which is followed by specification of system requirements. This activity again relies on conducted literature review, but also on problem domain experience that the authors and other ORKAN project members have.

Specified system requirements are input to third activity (Design artifact). In this activity, a solution artifact is designed in a form of prescriptive model for a flight mission management system. Building such model involves multiple iterations of brainstorming and trying out ideas, reasoning about them, making design decisions and documenting them. This activity primarily depends on the skills and experience of authors in terms of system modelling.

The last activity (Evaluate artifact) is reserved for conducting artifact evaluation. Evaluation has important role in design science as it demonstrates that the artifact is indeed a working solution, and that it has its scientific and practical worth. For this purpose, technical feasibility, and efficacy of the solution artifact (prescriptive model) will be evaluated. These are some of the most frequently evaluated artifact properties in design science research [13]. Technical feasibility will demonstrate that it is possible to build a prescriptive model for stated system. On the other hand, efficacy will show that the prescriptive model is indeed applicable when developing concrete flight management systems. Evaluation of the prescriptive model will be conducted by building a working prototype (instantiation) of a flight management system. Indeed, by building instantiations we operationalize models they are based on, and in this way demonstrate feasibility and efficacy of these models [14].

3. A solution Proposa

3.1. Explicate Problem

As indicated in introductory section, challenges involving UAV constellation piloting are numerous. They range from hardware related challenges (e.g. GNSS inaccuracy, battery and power limitations) and communication challenges (e.g. RF noise, connection availability) to software/algorithmic challenges (e.g. maintaining formation, avoiding collision, and piloting entire constellation as a single entity). In this paper we deal with the lack of generic solution (either at design or implementation level) for constellation piloting that would be applicable for different purposes and use cases (e.g. surveillance missions, search-and-rescue missions, scientific and data gathering missions, etc.). For this reason, software modules for constellation piloting are usually designed, implemented and tested from scratch, leading to reinventing the wheel effect, unnecessary spending resources and time. This problem certainly affects practitioners who rapidly seek to make the use of UAV constellations in wide variety of contexts. However, it also affects researchers, as piloting UAV constellations is still an ongoing research challenge. This proves that the stated problem holds both practical and scientific relevance.

3.2. Define Requirements

While in previous section we remained within the problem space, in this section we describe initial steps made in solution space. This is done by first describing fundamental characteristics of the solution artifact, which is then followed by a specification of system requirements.

From the problem explication it was evident that there is a need to provide way to properly express and manage concepts related to individual UAVs, UAV constellations, as well as devices used to control them. Due to a wide variety of target domains in which these concepts are present, a potential solution should be abstract and flexible enough to accommodate variabilities in these domains and uses. For example, a solution could be an artifact that contains abstractions for domain concepts, that relates them, and structures them into a coherent unit. In this way, an artifact captures prescriptive design knowledge that can be used as a starting point for building specific UAV constellation piloting module. Software architectures are typical examples of such larger-scale, design-only reuse artifacts. They also fit well with the notion of prescriptive model - one of the four common types of design science artifacts [11]. Therefore, in this paper we propose a prescriptive model of a software architecture for UAV Constellation Piloting. In the rest of the paper we will refer to it as UCP model.

It is important to note that while UCP model can be used independently, in our case it represents an architecture of a partial solution, i.e. a single module within larger system called UCCC (short for UAV Constellation Control Center). In addition to interacting with the rest of the UCCC modules, UCP is likely to interact with external software components (such as drivers for hardware devices, and SDKs for particular UAV technology) and hardware (joysticks, UAVs, single-board computers such as Raspberry PI, etc.) using various modes of communication technology (RF, UDP, TCP).

The target users of UCP model are primarily software designers and developers interested in building specific software solutions for piloting UAV constellations. In order to be clearly communicated with target population, UCP model has to be documented in a modelling language familiar to software designers and developers (e.g. UML). Additionally, UCP model has to be customizable and allow elements of specific software solutions to be easily fitted into architecture.

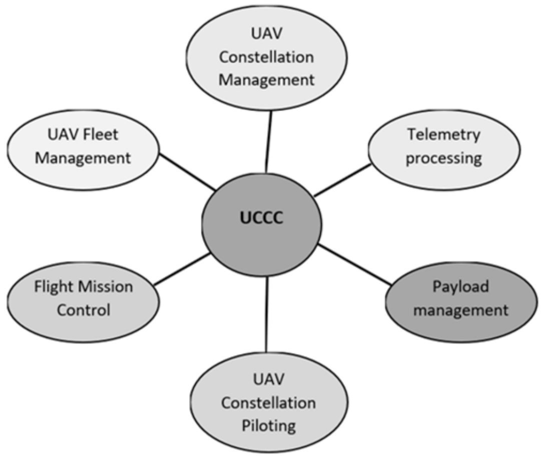

Figure 2. Modules of UAV Constellation Control Center (UCCC)

Final results of *define requirements* activity were reached through a number of brainstorming and prototyping sessions attended by paper authors and other ORKAN project members. Since model artifacts in design science are built from related construct artifacts, we first deliver base constructs that are necessary to understand the set of requirements.

- UAV (unmanned aerial vehicle) – also known as drone, is an aircraft without human pilot on board.
- UAV type – a particular model of UAV built by certain manufacturer (e.g. DJI Mavic Mini).
- UAV constellation – a group of UAVs set to conduct a joint flight mission.
- Flight mission – a planned and managed flight with specified goals and tasks to perform.
- Flight controller – physical device (e.g. joystick or keyboard), or a software component used to control individual UAV or UAV constellation.

Finally, we identified and elaborated following six requirements that UCP model should reflect and support. For each requirement we have documented its base requirement statement, rationale behind it, and how it can be evaluated.

Table 1. UCP Requirement #1

#	**UCP-1**
Statement	System shall allow controlling UAV constellation using single flight controller.
Rationale	Single human pilot with single device should be able to control entire constellation as if it was an individual UAV.
Evaluation	Commands issued by a single flight controller are propagated to all UAVs within constellation.

Table 2. UCP Requirement #2

#	**UCP-2**
Statement	System shall support the use of different types of flight controllers.
Rationale	Users may possess or prefer to use different equipment for controlling UAV constellation.
Evaluation	More than one type of flight controllers is available to choose from.

Table 3. UCP Requirement #3

#	**UCP-3**
Statement	System shall allow switching between flight controllers dynamically (even during flight)
Rationale	Due to equipment malfunction or in order to carry out some specific tasks, users may want to switch from one flight controller to another during flight.
Evaluation	A flight is started using one flight controller, and during the flight the control is switched to another flight controller.

Table 4. UCP Requirement #4

#	**UCP-4**
Statement	System shall allow switching between controlling individual UAVs and controlling UAV constellation dynamically.
Rationale	During flight an individual UAV could go astray, or could be needed for conducting a specific task, so individual control is needed.
Evaluation	A flight is started by controlling entire constellation. During a flight control is switched to a single UAV, and then back again to constellation.

Table 5. UCP Requirement #5

#	**UCP-5**
Statement	System shall control constellations formed from different types of UAVs.
Rationale	Users may have different types of UAVs with different capabilities in their possession. Also, the flight mission itself may require UAVs with different capabilities.
Evaluation	Commands issued by single flight controller result in appropriate reaction of different UAV models.

Table 6. UCP Requirement #6

#	**UCP-6**
Statement	System shall receive telemetry data from UAVs in constellation.
Rationale	In order to pilot UAV constellation, we have to know position and state of UAVs in constellation.
Evaluation	UAVs successfully report back their telemetry data.

3.3. Design Artifact

This section documents ideas and decisions made while designing constellation piloting module of UCCC system. The ending result is an artifact - prescriptive model, which can be used as a basis for designing similar models or as a blueprint for implementation of concrete software systems.

An architectural backbone of the constellation piloting module consists of two core components: (1) **flight controller** and a (2) **pilot**. Flight controller is represented by *IFlightController* interface with the goal of abstracting any physical device or software component that can be used to issue commands prior, during or post flight. Flight controller component has three responsibilities: (1) To detect inputs (e.g. key press on a keyboard), (2) to respond to an input (e.g. send turn on engine message to pilot), and (3) to provide custom user interface to visualize controller specifics. Depending on concrete situation, flight controller could receive inputs from an operating system (e.g. keyboard key pressed), device drivers (e.g. joystick axis moved), or other software components (e.g. autopilot component, on-screen commands, etc.). Responding to received inputs usually involves interpreting input and sending an appropriate message to pilot component (*IFlightController* holds a reference to an instance of pilot component). Finally, depending on particular controller, different data and available options should be presented to user. Therefore, within *IFlightController* we can also provide custom user interface that will be dynamically inserted into application.

Pilot component, as a second core component, has the responsibility of instructing a flyable entity (individual UAV or a UAV constellation) what to do. This must be done in a way particular flyable entities are capable of understanding (e.g. using supported communication technology) and responding (e.g. invoking only available UAV behaviour). *IPilot* interface is used here to prescribe common behaviour that the pilot component could request from any UAV, regardless of the underlying technology. This includes being able to turn UAV's engine on and off, to take off or land, and steer by setting pitch, roll and yaw parameters. In addition to common behaviour, our module also must be able to invoke wide variety of specific UAV capabilities. For example, some UAVs may have onboard sensors, cameras and other equipment that should be accessible through our module. For this reason, a generic ExecuteCommand is included into *IPilot* interface as a way to execute specific commands. Finally, the role of IPilot interface is not only to abstract away the underlying technological aspects of UAVs, but also to disguise what we are piloting - individual UAVs or entire constellation.

Rather than only promoting design reuse through *IFlightController* and *IPilot* interfaces, our architectural backbone also promotes code reuse. For example, abstract *FlightController* class contains programming code common to all flight controller components, while in abstract classes *ConstellationPilot* and *UAVPilot* we placed common code for all pilot components operating constellations and individual UAVs respectively. A *ConstellationPilot* must be able to instruct any number of UAVs, therefore it contains a collection of *UAVPilot* instances. Furthermore, being dependent only on abstract class *UAVPilot* allows constellation pilot component to fly different models of UAVs from different manufacturers at the same time.

One of the goals of proposed architecture is loose coupling between flight controller and pilot components. For example, pilot component itself is completely unaware of flight controller component that is sending commands. This implies possibility to switch between different flight controllers in runtime (even during the flight mission). The rest of the module also depends only on *IFlightController* interface, which means that we can easily implement additional types of flight controllers without affecting core architecture of the module. Flight controller component does hold a reference to a pilot component that it sends commands to (i.e. target). However, this dependency again is realized through *IPilot* interface. Consequently, flight controller is unaware of whether it is sending commands to single UAV pilot or entire constellation, which allows us to switch between these during a flight. Also, as UAV fleet becomes larger and more diverse, we can easily add new concrete pilot components for different UAV models without affecting core architecture.

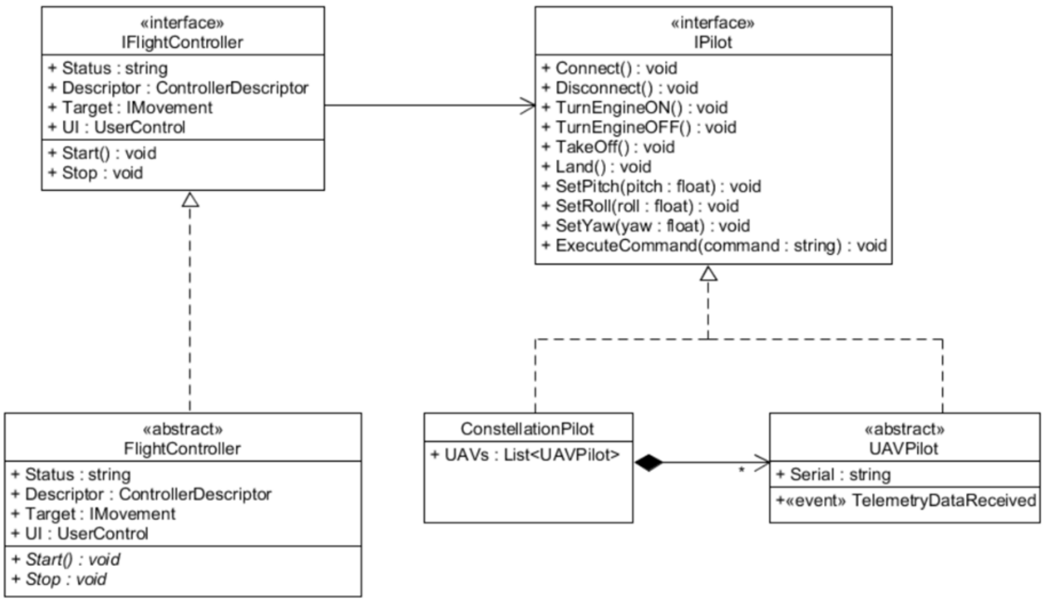

Figure 3. Architectural model for UAV constellation piloting module (UCP model)

3.4. Evaluate Artifact

As promised in methodology section, as a part of evaluation we built an instantiation artifact, i.e. we made a proof-of-concept implementation of constellation piloting module on the basis of previously developed model. Creating instantiation essentially boiled down to providing concrete implementations for abstract classes representing flight controller, constellation pilot and individual UAV pilot.

To demonstrate possibility of using different types of controllers we implemented two controllers, namely *KeyboardController* and *JoystickController*, both of which inherited abstract *FlightController* class. When implementing *KeyboardController*, the first controller responsibility (detect inputs) involved capturing messages related to pressing (key down) and releasing (key up) particular keyboard keys.

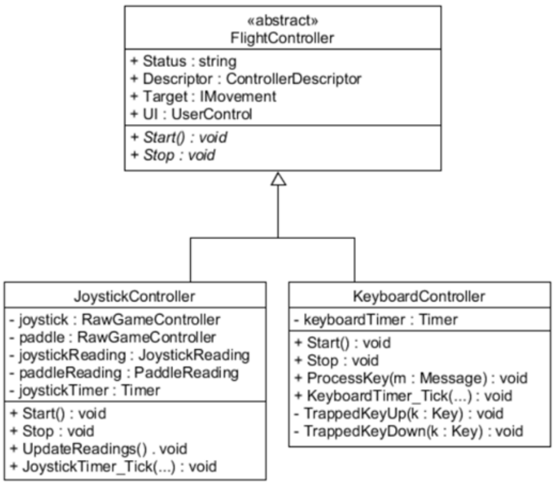
«abstract»
FlightController
+ Status : string
+ Descriptor : ControllerDescriptor
+ Target : IMovement
+ UI : UserControl
+ Start() : void
+ Stop : void
JoystickController
- joystick : RawGameController
- paddle : RawGameController
- joystickReading : JoystickReading
- paddleReading : PaddleReading
- joystickTimer : Timer
+ Start() : void
+ Stop : void
+ UpdateReadings() . void
+ JoystickTimer_Tick(...) : void
KeyboardController
- keyboardTimer : Timer
+ Start() : void
+ Stop : void
+ ProcessKey(m : Message) : void
+ KeyboardTimer_Tick(...) : void
- TrappedKeyUp(k : Key) : void
- TrappedKeyDown(k : Key) : void

Figure 4. Concrete classes for keyboard and joystick controllers

Using underlying event system to detect keyboard message would suffice when invoking discrete behaviour of UAVs (e.g. pressing turn on/off engine key). However, cases where holding (possibly multiple) keys in a pressed state should repeatedly (e.g. when steering) invoke some behaviour proved to be problematic with this implementation.

Anyone using keyboard would expect UAVs to continue with requested movement as long as the appropriate key was pressed. This forced us to introduce timers to create loop (common strategy in computer games) to continuously check if some key is newly pressed, released, or remained being pressed. When key action is detected during loop iteration an appropriate method is invoked on a *IPilot* target (second controller responsibility). Finally, a custom user interface was created to visualize keyboard activity and configure mappings between keys and target methods (third controller responsibility).

When implementing *JoystickController* we relied on *RawGameController* class from Windows Runtime API to detect inputs from joystick devices. In our case the test equipment included two cooperating devices - joystick and the paddle, so we had to maintain two *RawGameController* instances. Timer loop was used here as well to continuously update readings coming from joystick and paddle. These readings allowed us to determine pressed buttons, switch positions, and axis position in each loop iteration. Once joystick and paddle inputs were known, target (*IPilot*) methods were invoked. Finally, *JoystickController* also provided a custom user interface to not only visualize type of joystick actions, but also the intensity (e.g. how much the axis is pulled right).

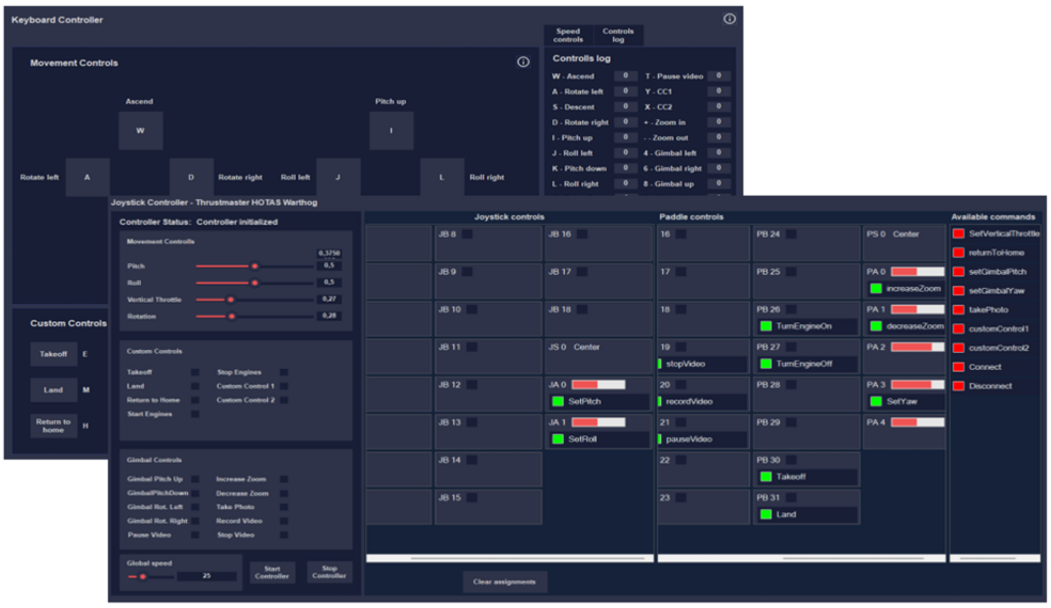

Figure 5. Custom user interfaces for keyboard and joystick controllers

ConstellationPilotBasic as a concrete implementation of *ConstellationPilot* abstract class was fairly simple in this proof-of-concept. It merely redirected the method invocations to individual *UAVPilot* instances. So, for example, when *ConstellationPilot* received message to invoke *TurnEngineOn* method, it proceeded to invoke *TurnEngineOn* method of all *UAVPilot* instances it contained.

Ensuring this communication was not a trivial task. The main issue was the fact that DJI's SDK did not provide APIs to directly access and maneuver drones from a central (desktop) application. Rather, a sole option to instruct drone to do something was to use original DJI joysticks and radio frequency communication. Luckily, DJI's joystick can be connected to Android device via USB cable, and there is an actively maintained DJI SDK for Android. This gave us an opportunity to programmatically control DJI drones after all. For this purpose, an Android application was developed which was in charge of receiving commands from the desktop application, and then invoking appropriate methods exposed in DJI's SDK. In order for DJIPilot component (desktop application) and Android application to communicate, we implemented a small TCP server as a part of desktop application.

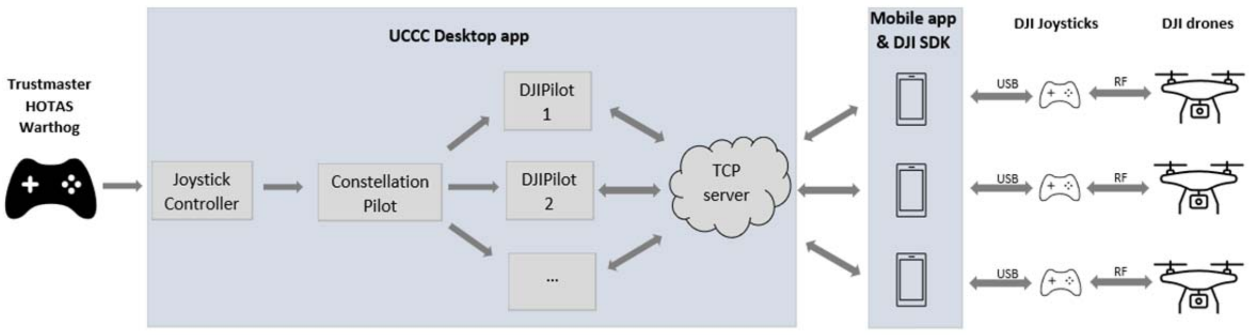

Figure 6. Proof-of-concept architecture

4. Discussion

In order to tackle the problem of lacking general solutions for piloting UAV constellations in a systematic and rigorous way, in this paper we performed a design science process. As a problem-solving process, design science results in an artifact which needs to have both scientific and practical relevance. One frequent type of design science artifacts that can capture commonalities across various domains, but also offer variability points for customization and extension are model artifacts. Therefore, our attempted solution included proposing an architectural model that is general and flexible enough to serve as a blueprint for wide variety of UAV constellation piloting software (UCP model).

Before diving into design activities and determining *how* exactly should our solution work, we first had to identify *what* should our solution do. This coincides with the first research question (RQ1) aimed at identifying functional requirements that the constellation piloting module based on the proposed model should meet. Through a series of brainstorming and prototyping sessions attended by paper authors and other project members, we identified and reported six core requirements, and thus answered RQ1. While these requirements are not final output of our research they can be used as a basis for devising new model for constellation piloting software, independently of our proposed model. In this way, stated requirements can be valuable for both practitioners (during requirements phase of traditional software development process) and researchers (during problem-solving research processes such as design science).

Of course, an architectural model of UAV constellation piloting module that is based on identified requirements is a primary output of design science process. It provides valuable information on how UAV constellation piloting module could look like, i.e. what structural and behavioural properties it possesses. In order to easily depict these properties and communicate them with target audience (software engineering researchers and practitioners), we used UML class diagrams as a modelling technique.

Architectural model such as the proposed one again offers valuable contributions to both practitioners and researchers. Practitioners get design-level artifact ready to reuse and customize during design activity of their software process. Scientific contribution of the proposed model is reflected in a prescriptive knowledge that informs researcher about necessary ingredients for the solution. This includes identified components that participate in solution, characteristics of these components, distribution of responsibilities between components, and specifics of their cooperation.

Creating UCP model involved making a number of design decisions which are reported in Design artifact section. This, for example, includes identifying two core components of the solution, namely, flight controller and a pilot. Representation of these components in our model is highly abstract, i.e. any interaction with flight controller or a pilot is done through their respective interfaces. As a consequence, complete implementations can vary without affecting the core structure of the solution. So, for example, we can easily introduce additional specific controllers (joysticks, keyboards, software modules…) or specific UAV models. We could even switch between these during runtime (e.g. start piloting constellation with keyboard, and the switch to joystick).

During implementation of proof-of-concept solution (reported in Evaluate artifact section) we faced some unexpected but interesting design and implementation decisions. For example, one of the largest issues we faced was how to establish communication between desktop application and a UAV. Limitations in UAV APIs forced us to devise entire communication infrastructure, including TCP server and Android application. Because restrictions in UAV APIs are not a characteristic of only UAVs we had at our disposal, even these design decisions can prove valuable to other researchers and practitioners. Design decisions reported in both design and evaluation sections constitute the answer to second research question (RQ2).

5. Conclusion

In this paper we addressed the problem of lacking general solutions for piloting UAV constellations. In order to conduct the problem-solving process and report its results in a systematic and rigorous way, we followed a design science approach. The first explicit result of a design science process was identification of architectural model for a UAV Constellation Piloting (UCP) module as a proper solution artifact for addressing the stated problem. This was followed by identifying and specifying six requirements that the UCP module should fulfil to be considered a solution. On that basis an architectural model was designed to act as a blueprint for software module implementation. Finally, in order to evaluate proposed model we implemented proof-of-concept software module which proved feasibility and efficacy of the model.

Since it reports on pragmatic, problem-solving research, contributions of this paper are twofold. On one hand, from scientific perspective, the paper offers novel prescriptive knowledge in a form of software requirements specification (answer to first research question) and architectural model (answer to second research question). On the other hand, these outputs can also be directly applied by practitioners to speed up and facilitate development of specific constellation piloting solutions.

While it mitigates the problem of creating software solutions for piloting UAV constellations, proposed solution does not represent the final and complete solution. A number of challenges and improvement possibilities remain for future work. For example, we plan to go beyond a mere design reuse, and offer implementation reuse by building framelet (small modular framework) on top of proposed model. Also, as depicted in Define requirements section, constellation piloting is only a part of a larger system which is currently under construction. Therefore, we plan to report on other problems and solutions closely related to constellation piloting that arise from our experience.

Acknowledgements

This work has been fully supported by the Croatian Science Foundation under the project IP-2019-04-4864 (ORKAN – Unmanned Aerial Vehicle Control and Surveillance Framework).

References

[1]. Iscold, P., Pereira, G. A., & Torres, L. A. (2010). Development of a hand-launched small UAV for ground reconnaissance. *IEEE Transactions on Aerospace and Electronic Systems*, *46*(1), 335-348. https://doi.org/10.1109/TAES.2010.5417166

[2]. Oubbati, O. S., Atiquzzaman, M., Ahanger, T. A., & Ibrahim, A. (2020). Softwarization of UAV networks: A survey of applications and future trends. *IEEE Access*, *8*, 98073-98125. https://doi.org/10.1109/ACCESS.2020.2994494

[3]. Montenbruck, O., Steigenberger, P., & Hauschild, A. (2018). Multi-GNSS signal-in-space range error assessment–Methodology and results. *Advances in Space Research*, *61*(12), 3020-3038. https://doi.org/10.1016/j.asr.2018.03.041

[4]. Propeller. (2022). "Ground Control Points for Drone Mapping," PropellerAero. Retrieved from: https://www.propelleraero.com/aeropoints/ [accessed: 20 March 2022].

[5]. Stempfhuber, W., & Buchholz, M. (2011). a Precise, Low-Cost Rtk Gnss System for Uav Applications. *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *3822*, 289-293.

[6]. Campion, M., Ranganathan, P., & Faruque, S. (2018, May). A review and future directions of UAV swarm communication architectures. In *2018 IEEE international conference on electro/information technology (EIT)* (pp. 0903-0908). IEEE. doi: 10.1109/EIT.2018.8500274.

[7]. Wan, X., Ghazzai, H., Massoud, Y., & Menouar, H. (2019, April). Optimal collision-free navigation for multi-rotor UAV swarms in urban areas. In *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)* (pp. 1-5). IEEE. doi: 10.1109/VTCSpring.2019.8746332.

[8]. Yasin, J. N., Mohamed, S. A., Haghbayan, M. H., Heikkonen, J., Tenhunen, H., & Plosila, J. (2020, October). Navigation of autonomous swarm of drones using translational coordinates. In *International Conference on Practical Applications of Agents and Multi-Agent Systems* (pp. 353-362). Springer, Cham. doi: 10.1007/978-3-030-49778-1_28.

[9]. Jackman, A. (2019). Consumer drone evolutions: Trends, spaces, temporalities, threats. *Defense & Security Analysis*, *35*(4), 362-383. doi: 10.1080/14751798.2019.1675934.

[10]. Stol, K. J., & Fitzgerald, B. (2020). Guidelines for conducting software engineering research. In *Contemporary Empirical Methods in Software Engineering* (pp. 27-62). Springer, Cham.

[11]. Hevner, AR, March, ST, Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly* , *28* (1), 75-105.

[12]. Johannesson, P., & Perjons, E. (2014). *An introduction to design science* (Vol. 10, pp. 978-3). Cham: Springer.

[13]. Prat, N., Comyn-Wattiau, I., & Akoka, J. (2015). A taxonomy of evaluation methods for information systems artifacts. *Journal of Management Information Systems*, *32*(3), 229-267. doi: 10.1080/07421222.2015.1099390.

[14]. March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision support systems*, *15*(4), 251-266. doi: 10.1016/0167-9236(94)00041-2.