5-2020

Teaching Distributed Application Design Using Drones

Cameron Frandsen
Utah State University

TEACHING DISTRIBUTED APPLICATION DESIGN USING DRONES

by

Cameron Frandsen

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Stephen Clyde, Ph.D. Major Professor	Dean Mathias, Ph.D. Committee Member
Dan Watson, Ph.D. Committee Member	Richard S. Inouye, Ph.D. Vice Provost for Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2020

ABSTRACT

Teaching Distributed Application Design using Drones

by

Cameron Frandsen, Master of Science

Utah State University, 2020

Major Professor: Stephen Clyde, Ph.D.
Department: Computer Science

Distributed applications have become common for mobile, web, and even desktop applications. Consequentially, computer science students must learn how to program reliable distributed applications by the time they graduate. Designing distributed applications is more complicated than designing stand-alone applications. Some concepts, like secure and reliable communication, pertain to distributed applications and not to stand-alone systems.

There are several different pedagogies that instructors can use to teach distributed applications. A popular pedagogy for teaching distributed systems is project-based learning. The assignments used in this pedagogy must reinforce course concepts while providing an opportunity for the students to gain the essential skills and enhance their abilities. The instructor should have control over the concepts the students are learning, while the students need the flexibility to design their system.

This thesis introduces a software platform that instructors can use in a project-based learning environment. This platform gives the students the freedom to design the application-level components of the systems they build while eliminating the need for them to deals with extraneous details. At the same time, this platform gives the instructor control over the concepts that are being taught through various assignments.

The platform uses drones to help keep the interest of the students, so they are more fully engaged in the learning that the platform provides.

This thesis also presents an adaptation of decision matrices as a method for compare assignments in distributed-application courses. This method breaks down each assignment and provides a quantifiable way to compare the assignments. The platform presented in this thesis will be compared to two other frameworks using the technique. This research will help an instructor know what assignment is best to use for a specific distributed systems class. With this method of comparing assignments, different instructors with different priorities can compare assignments how they wish.

(89 pages)

PUBLIC ABSTRACT

Teaching Distributed Application Design using Drones

Cameron Frandsen

This thesis provides a new platform that instructors can use to create a learning environment for students to learn how to design and implement distributed applications. One possible way to use the platform uses milestones to test the student's understanding of different concepts. Milestones contain pretests that students can use to test their code. Each milestone focuses on a different concept, including interprocess communication, reliability, and security. A monitor process can be created by the students to provide a way for the instructor to see how well the students are learning the concepts by using a monitor process. The platform gives control to the instructor to choose what skills, knowledge areas, and abilities are covered by an assignment. There are many valid solutions to the problem this platform provides, giving students the flexibility to design their system. The platform uses drones to create interest in the subject to more fully engage the students and better help them understand distributed systems. This thesis provides a sample implementation of the assignment to validate the platform. This thesis then compares this platform to other assignments in a project-based pedagogy for distributed systems using a new technique. This technique quantifies the effectiveness of each assignment dynamically, allowing instructors with different priorities to compare the assignments in different ways.

ACKNOWLEDGMENTS

I want to express gratitude for the countless hours Dr. Stephen Clyde has taken to help me with this research. He has been there every step of the way, and this would not have been possible without him. I am very grateful for the opportunity he gave me to do a thesis, as well as everything he has taught me on this journey.

I am also very grateful to my committee members, Dr. Dean Mathias and Dr. Dan Watson. It has been beneficial to be able to ask them questions and get their opinions. I appreciate the time they have taken to help me with this thesis, as well.

It would be a mistake, not to mention my wife and child. They have been there supporting me the whole time. They have been very patient with me and my late nights working on this thesis.

Cameron Frandsen

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

API	Application Programming Interface
CS	Computer Science
DA	Distributed Application
DALP	Distributed Application Learning Platform
DJI	Dà-Jiāng Innovations
GUI	Graphical Application Interface
IDA	Instructor Directed Assignment
IP	Internet Protocol
KSA	knowledge, skills, and abilities
HiSAP	Highly interactive simulation of algorithms and Protocols
JSON	JavaScript Object Notation
PBL	Project Based Learning
SDK	Software Development Kit
STEAM	Science Technology Engineering Art Mathematics
SDA	Student Directed Assignment
R&D	Research and Development
RSA	Rivest, Shamir, and Adelman
PKCSA	Public Key Cryptography Standards
SSID	Service Set Identifier
TCP	Transmission Control Protocol
TUI	Text-Based User Interface
UAS	Unmanned Aerial System
UDP	User Datagram Protocol
UUID	Universally Unique Identifier
XML	Extensible Markup Language

CHAPTER 1

INTRODUCTION

Distributed Applications (DA's) are software applications that consist of multiple communicating processes that coordinate their work through message passing. DA's are not only becoming popular; they are becoming pervasive. For example, many commercial web-based and mobile apps are actually DA's. The shift from stand-alone software applications to DA's means that Computer Science (CS) students must learn how to develop quality DA's to meet the needs of an ever-increasingly sophisticated and connected user community. In other words, the more popular DA's become, the more critical it is to teach DA development to burgeoning software engineers.

However, teaching DA development effectively is a difficult pedagogical challenge. One reason is that engineering DA's relies more on design, testing, and implementation concepts that go beyond those required for creating stand-alone software systems. Some of these concepts include a) inter- and intra-process concurrency, b) the handling of partial failures caused by either disrupted communications or host-computer failure, c) managing multiple concurrent communication channels, d) synchronizing tasks without a shared global clock, and e) establishing efficient communication protocols. See Sections 2.1-2.3 for more details on DA's and DA development, as well as what concepts DA developers need to understanding.

Businesses and government agencies often define job requirements in terms of knowledge, skills, and abilities (KSA) [1]. For DA development, properly trained software engineers should be **able to apply** their **knowledge of core DA concepts** and their **design, implementation, and testing skills** to maximize DA software-quality goals, such as security, reliability, scalability, extensibility, and maintainability. For educators, the challenge is to provide the means by which students can learn all the necessary concepts, gain first-hand experience with the related skills, and have opportunities to hone

their natural abilities.

One aspect of this challenge is the number and diversity of underlying concepts. Naturally, there are many different curricula for DA courses, each favoring a specific subset of concepts associated with a textbook, an instructor's background, certain types of DA's, or a particular industry segment. However, regardless of a course's curriculum, students need to become familiar with as many core concepts of DA's as they can, so they can be as versatile as possible.

Another aspect of the overall pedagogical challenge is providing effective software-engineering assignments that are engaging and rich with opportunities for hands-on experience with DA concepts. It is this aspect of teaching DA development that is the focus of this research. Specifically, this research addresses the sub-problem of creating a platform that can allow instructors to design effective assignments which target core DA concepts while minimizing distractions caused by extraneous details. As a means of making sure assignments are engaging and based on real-world problems, I decided to have the platform tied to the domain of drones and autonomous control. Section 2.4 provides some background on drones and drone SDK's (Software Development Kits).

Developing a platform as a foundation for software-engineering assignments and using drones in teaching are not new ideas. Chapter 3 discusses some publications related to these ideas, as well as teaching DA development and teaching in general.

Depending on the course and instructor preferences, DA software-engineering assignments may be course-long or short independent assignments, team-based or individual, or any variation of these basic formats. Even though this research and its contributions are not inherently limited to any particular format, for simplicity, the focus here will be on an individual course-long assignment that is broken up into multiple stages.

A key issue for an assignment's design is how much control to give the students over the software's architecture. If the instructor does all the design work and only lets the students implement and test code, then the students may miss out on some valuable experiences related to DA design. On the other hand, this approach can help instructors

ensure that the majority of the students achieve a pre-determined set of learning objectives. At the other end of the spectrum, the instructor could give students complete freedom in choosing their own projects and developing their own designs, as well as responsibility for implementing and testing all of their own code. This approach allows for a broader range of learning experiences, including those that come from failure, but at the cost of less uniformity and predictability. It is not easy for students to choose a project that will focus on concepts taught in class, as the students do not know what those concepts are yet. Students may not learn some important DA concepts that end up not being required in their chosen projects. This approach also requires assignments to be smaller and less sophisticated, since the student are responsible for whole development life cycle. Both approaches, as well as several hybrid approaches, have been tried at USU in the introductory class to distributed systems. See Section 2.3 for more details.

Another key issues for an assignment's design is size or scope. The instructor must keep the size of the assignment at a level that is doable within the amount of time students are expected to dedicate to the assignment. In the CS Department at USU, for a 4-credit senior-level class, e.g., its Introduction to Distributed Systems class, students are expected to spend 12-16 hours/per week outside of class. Therefore, if an assignment has a two-week duration and there are no other out-of-class expectations during those two weeks, then the assignment should be doable within 24-32 hours.

Chapter 6 discusses the above issues for DA assignments in more detail and presents thoughts on DA-assignment goals, in terms of both helping students achieve certain KSA levels and helping instructors create assignments that are flexible, adaptable, and minimize extraneous details. Based on the literature review summarized in Chapter 3, to date no author has compiled a list of such issues and DA-assignment goals. The ideas presented in Chapter 6 represent one of the contributions of this research.

As mentioned above, the main contribution of this research is a platform that instructors can use to provide effective DA assignments. This platform, called *Distributed Application Learning Platform* (DALP), allows instructors to maintain control over

learning objectives while giving students significant (but not total) control over their own designs, implementation, and testing. It supports short assignments and course-long assignments as well as individual and group assignments.

DALP includes three out-of-the-box programs and two built-in test suites: referee, drones, drone simulator, a test suite for referee communications and a test suite for drone communications (see Figure 1.1). The drones currently supported by DALP are the Tello Edu drones made by Dà-Jiāng Innovations (DJI). The drones have a simply UDP-based application programming interface (API), can capture images, transmit a video and status message streams, and can automatically identify special targets (10x10 cards with unique patterns). The drone simulator mimics the communications and behavior of a real drone and logs useful information about its communications and state over time. Thus, it can help the students develop, testing, and debug their system without the use of a physical drone. The referee can provide goals, i.e., targets to be found, and can verify expected behaviors for the drones. The first test suite is packaged with the referee and can verify that student-written components are communicating with the referee according to its protocol. The second test suite is packaged with the drone simulator and does the same for drone or drone-simulator communications. Student enable one or both test suites by changing parameters in a configuration file.

With DALP, students may build referee clients, drone clients, monitors, and other functionality necessary to complete a DA assignment. Of course, dependent of the DA course and the learning objectives of the assignment, the instructor may provide designs, partially implementations, or even full implementations for some of this pieces. Also, although these pieces of functionality are shown as individual components in Figure 1.1 (i.e., the light blue boxes), they can be organized into many different designs, with different abstractions, modularity, and encapsulation. A monitor is an optional piece of function that provides an overview of the system and provides insight into how all of the processes are running. Students can use a monitor as a debugging tool.

Beside describing the architectural details of DALP, Chapter 4 introduces a sample

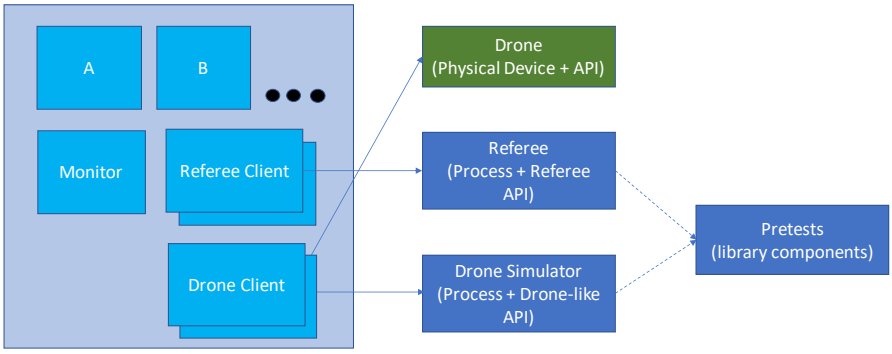

Fig. 1.1: Referee class diagram

semester-long DA assignment that takes the form of a game, where the objective is to have multiple drones hit targets given by the referee and to do so as fast as possible. More specifically, in this assignment, students implement a distributed player that receives information about targets from the referee at irregular intervals and controls multiple drones to obtains these targets as quickly as possible. A successful solution will need to consist of multiple processes or concurrent threads that coordinate tasks regularly and efficiently via inter-process communications.

Sections 5.1 and 5.2 describe the verification and validation of DALP. The verification ensures that DALP functioning as intended and is based traditional unit, integration, and system testing. The validation (see Section), which checks to see intended functionality is what instructors and students truly need, involved through role-playing use cases. Specially, it involved playing the role of an instructor who wants to design DALP-supported assignments and the role of a student who then needs to complete those assignments.

In addition to creating, verifying, and validating DALP, this research developed a unique method for comparing the effectiveness of DA assignments using decision matrices. Section 6.1 describes this method and Section 6.3 show the results of an initial comparison

between a DALP-supported assignment and two others. The method quantifies the effectiveness of each assignment in terms of the opportunities it affords students in specific KSA areas and then combines scores through a weighted decision matrix.

Finally, 7 summarizes the contributions of this research, namely those providing by DALP and a method for evaluating the effectiveness of DA assignments. It also discusses future research ideas that involve a) enhancing DALP to support other kinds of drones, b) applying the evaluation method to a boarder range of situations, and c) conducting empirical studies that attempt to measure student outcomes against learning objectives, with and without DALP-supported assignments.

CHAPTER 2

BACKGROUND

2.1 What are Distributed Systems and Distributed Applications

Distributed Applications (DA's) are communicating processes that work together to accomplish a particular set of goals or to complete a task [2]. These processes can be on the same machine or different machines. Each process has a specific job related to the overall goal of the system. Some processes perform the same functions, while others may work on different tasks. In a DA, processes coordinate their actions using inter-process communication, typically network messages.

The term *Distributed System* is often informally used when talking about a DA, but this can be confusing because it also has more precise definition that refers to the environment on which a DA runs. In this context, a distributed system consists of the communications and resources in the bottom four layers of the 7-layer Open System Interconnection (OSI) model, i.e., the hardware layer up through transport layer [3] (see Figure 2.1). A DA, on the other hand, involves communications and resources in the top three layers, i.e., the session layer through the application layer. Since most software engineers will be involved in building software for the top three layers instead of the bottom four, the biggest need is helping students improve their DA-related knowledge, skills, and ability.

Distributed systems and applications can offer many benefits, including improved throughput, scalability, and fault tolerance. A DA can run on several machines to take advantage of parallel processing across those machines. The processing load can be divided up by the system that processing does not bottleneck on one process or device. For example, many mathematical problems run on large distributed systems to speed up computation time. As needs increase, software engineers can enhance the system onto

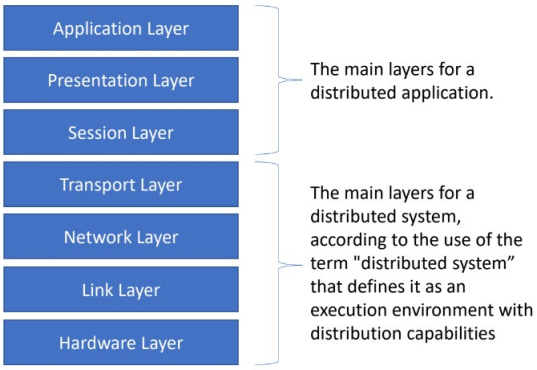

Fig. 2.1: 7-Level Open System Interconnection model

more machines and processes. This is one type of scalability that DA's provide that stand-along applications do not provide. DA's have better fault tolerance than stand-alone applications because if one or more processes go down, then the system as a whole may remain functioning, depending on its design.

Other potential benefits of a DA are heterogeneity and transparency. Heterogeneity exists when diverse hardware is involved. This can simplify hardware acquisition and maintenance. DA's can also be transparent in several ways, including access, location, and mobility transparency [2].

The processes of a DA must use message passing to communicate with each other since their memory address spaces are independent. Note that if a systems consists of multiple processes that share memory in some way, then it is classified as a parallel system or application, not a distributed system or application [4]. Without message passing, each process would be on its own; there would be no resource sharing and no coordination – the system would be disjoint.

As DA's are more complex than stand-alone applications, there are inherently more challenges in designing a DA than a stand-alone application. Every challenge that needs to be solved provides an opportunity to make important design decisions. Some common challenges are delayed or faulty communication, geographic distance, and differences in the behaviors of various implementations of underlying software libraries.

2.2 Core Concepts for Distributed-Application Development

Below is a list of 10 key concepts that software engineers need to be aware of when designing and programming DA's.

1. Networking and internetworking
2. The TCP/IP protocol suite
3. Inter- and intra-process communication
4. Handling failure
5. Security issues
6. Concurrency controls
7. Distributed objects
8. Distributed file systems
9. Time services
10. Client-server and peer-to-peer architectures

Networking and Internetworking

As DA's can run on multiple devices, students must understand how those devices connect to communicate with each other. There are different kinds of networks, and each type has a different purpose. Students must not only understand IP(Internet Protocol) addresses and ports, but networking in general to design an efficient DA.

The TCP/IP protocol suite

In addition, students must understand what communication protocols are and which existing protocols to use, and which to custom design. While it is impossible to cover every protocol, there are several protocols that the industry widely uses, including TCP and UDP (transport-layer protocols), and HTTP and HTTPS (application-layer

protocols). Students must be able to decide which protocol to use in different situations, design new protocols, and implement and test communications.

Inter- and intra-process communication

Next, students must understand how to send data in an exchange between processes, i.e., in a conversation, according to a given protocol. This typically involves serializing and deserializing objects. There are many ways to serialize and deserialize information, but students should learn at least one common mechanism, such as JSON, XML, or Protobuf. Some DA's need to compress data exchanged in conversations, which can take longer. Students need to understand the trade-offs and design accordingly.

Sharing resources and coordinating activities in a DA while using inter-process concurrency presents some non-trivial problems, like distributed mutual exclusion [2]. Software engineers should understand these problems, theories, principles, practices, and patterns associated with their solutions in the context of any given DA.

Security issues

There are two area for security considerations in DA's: 1) data in motion and 2) data at rest [5]. Data in motion, i.e., data exchanged via inter-process communications, needs to be secure when 1) the data is confidential, 2) when there is a threat of replay attacks, 3) when there may be a threat of man-in-middle attacks, or 4) when a process needs to authenticate the other process. Also, when data is confidential, it may need to be secured when at rest, i.e., stored on some kind of secondary storage device. Students need to understand general security considerations and know when and how to encrypt and decrypt data.

Handling failure

There are two types of failures that need to be considered in DA's, communication failure and process failure. Sometimes the communication channels can be unreliable. Messages can be lost, late, duplicated, or out-of-order. Inability to overcome unreliable

communications can lead to undesired behavior in DA's and can make them hard to use. Students must be able to design, implement, and test methods for handling unreliable communications.

Individual processes in a DA can also fail because they run independently of each other, often on different host machines. Having a subset of processes fail is called partial failure and is something that DA should be able to handle gracefully. Students should understand the causes and concerns surrounding partial failure and be able to design a DA to handle failure at some required level.

Concurrency controls

In a DA, processes share resources and those resources may be accessed by other processes. If multiple processes try to access the same resource at the same time and that concurrency violates the business rules of the DA or could lead to an integrity problem, then the DA needs to implement some kind of concurrency control [6]. There are a number of standard concurrency controls, such semaphores and monitors. Students need to first understand the issues surrounding concurrency controls and when they are needed, as well as at least several techniques or mechanisms for implementing them.

Distributed Objects

As mentioned, DA's include shared resource. One way to implement a shared resource is with a distributed object that hosted in one process but accessible to others through a global object reference of some kind. Students should be familiar with the concept, implementation, and use of distributed objects as a way of provided shared resources.

Distributed file systems

Some DA's contain distributed file systems or object stores, where the files do not all reside in the same place but can be on many different machines. Common examples of a distributed file system include Google Drive, Microsoft Azure, and Amazon S3. There are

many benefits to a distributed file system that students can use to overcome challenges in different DA's.

Time services

DA's do not have a shared global clock; instead, each process has a different local clock and that must communicate to sync they clocks, if needed. Without a shared global clock, it is difficult to know what order events happen. Software engineers must be aware of this and know how to use logical time instead of real time.

Client-server and peer-to-peer architectures

There are several common architectures for distributed systems, including client-server, n-tier, and peer-to-peer. The client-server model consists of one process where all the processes connect to the server, which controls access to all the shared resources. A client-server architecture can simplify coordination but can also create a single point of failure and a bottleneck for communications. The n-tier architecture tries to some the latter problem by splitting the functionality of the server into multiple layers (e.g., a web tier, business-logic tier, and data tier) and by allowing the DA to run the various process in each. Typically, systems n-tier architectures still rely on some front-end or load-balancing process that still represents a single point of failure. Still, the risk of that process failing is often much less than in a system with a pure client-server architecture. The downside of the n-tier architecture is its complexity. With a peer-to-peer architecture, there is no single server responsible for control access to all shared resources. Each process may hold and control access to a subset of the shared resources. Therefore, if one process fails, the others may still be able to complete some or all of their tasks. DA's based on a peer-to-peer architecture, however, come with their challenges, including resource identification and location, distributed mutual exclusion, and even an occasional need to perform a distributed election [2]. All these architectures are useful under certain situations, and therefore, software engineers need to know about them and when to use them.

2.3 Introductory Class for Distributed Application Development

CS5200 is the introductory course at Utah State University that focuses on teaching DA development. Its intended audience consists of seniors working on a B.S. degree in Computer Science, as well as first-year graduate students who are going into distributed systems as a research area. The students in the class work in groups of three or four to build non-trivial distributed systems. For the past two years, each group has come up with a project idea and constructed it from beginning to end. Before that, the CS5200 students used an instructor directed assignment to build three programs for a distributed system that consisted of seven different programs.

There are several advantages and disadvantages to each approach. For the most recent approach, the most significant benefit is it gives the students flexibility and experience of making architectural design choices. The most significant drawback is the lack of control over the learning outcomes — some projects end up not offering opportunities for students to address particular learning objectives. For example, one project may allow students to master the Transmission Control Protocol (TCP), but not User Datagram Protocol (UDP). The earlier approach provided more uniform student outcomes but required the students to understand, use, and extend instructor-provide software components. Although these are essential skills, they are outside the scope of CS5200, as would be the case for most classes focusing on distributed-system design, implementation, and testing.

It would be helpful to both students and instructors to achieve the benefits of both approaches without incurring their disadvantages. Instructors need to have predictable outcomes for the students across a defined set of learning objects. Students need the flexibility to design distributed systems to gain experience with architectural design decisions. A framework could enable students to build non-trivial distributed systems with design decision with a predictable outcome of learning objects.

2.4 Drone SDKs

A Software Development Kit (SDK) allows developers to control a drone programmatically. Each SDK only works with certain drones. Some popular SDKs are

Tello, DJI SDK, Parrot, DroneKit, DroneCode, and CoDrone. Each is designed specifically for certain drones because each drone has different sensors, motors, and protocols. An SDK allows a developer to control the drone programmatically. SDKs can control flight paths, sensors, and drone-specific gimbals.

Drones have different kinds of sensors. Standard sensors are an altimeter, gyroscope, thermal, orientation, accelerometer, and more. Using the SDK's API(Application Programming Interface), you can query for flight time telemetry data. Many SDKs have queries to see how much battery is left. A query to know how much battery is left allows developers to know how much flight time is left, allowing the drone to have enough battery life to return. Some SDKs even provide methods for drones to do flips.

CHAPTER 3

RELATED WORK

This research is based on several research papers that include teaching DA's with a framework, using drones in the classroom, teaching computer science using game design, and decision matrices.

Frameworks are a popular way to create assignments for teaching distributed systems development. In "A Framework to Support Teaching in Distributed Systems", Burger created a framework for teaching DA's called HiSAP [7]. The framework takes scripts from the user and provides a way to show the script functionally. The system outputs an applet to demonstrate different concepts. Some of the concepts that were taught using the HiSAP framework were distributed systems, distributed multimedia systems, and computer networks.

Instructors can teach DA development in many ways, but research papers in this area suggest that project-based learning should be used. Schummer researched how important it is to use project-based learning [8] and breaks down assignments into five different sections: administration, purposing, group building and planning, execution, and judgment. The administration is where the students choose to take the class, and the teacher decides to teach the students. Purposing is where the teacher gives the students a set of requirements, and the students come up with projects to fulfill those requirements. Group building and planning is where the students take all of their ideas and select which one they will pursue. Execution is where the students complete their chosen project. Judgment is when the students can compare how they did and think of ways they could have done it better. It is essential that at the end of the project, there is some reflection so that students can be more prepared for creating distributed systems in their careers. DALP provides an environment where this can take place. The instructor presents the requirements, and the students have the opportunity to design a project to complete those

requirements. At the end of the project, the students will evaluate how they did.

Drones can be used in education to further engage students in the subject. In "New Perspectives on Education: Drones in the Classroom", Carnahan says, "It is critical to incorporate these technologies into instructional practices so that students are college and career ready." [9] Carnahan talks about a correlation between student engagement and overall student success. The more that students are engaged in what they are doing, the better the academic performance. Carnahan says, "The use of digital technologies in the classroom connects to students' interest and assesses the learning through a variety of methods." This research aims to further engage the students in DA development by using drone technologies to enhance learning.

Sattar, in the research paper "Droning the Pedagogy: Future Prospect of Teaching and Learning" outlines how drones need to be used in education [10]. Fahra goes on to say that, "The use of drones in education is opening new trends in teaching and learning practices in an innovative and engaging way." There are many things that drones can teach; examples include sequencing, repetition, events, conditional logic, problem-solving, and debugging. There are so many different kinds of drones that you can use to teach a wide variety of skills. This thesis shows another way how drones can teach skills and hone abilities. That by using drones, students will be more interested and be more engaged in the learning.

Drones can be especially useful in teaching STEAM topics. STEAM stands for science, technology, engineering, art, and mathematics. In "Exploring the Learning Effectiveness of "The STEAM Education of Flying and Assembly of Drone", Chen says that drones can integrate imagination, creativity, and innovation. Drones can also cultivate students' innovative thinking, practical assembly ability, and active learning attitude [11]. Not only do students learn how to fly drones, but they learn how drones work. They learn about the components of drones and how those components interact with each other. By using drones in this research, it cultivates imagination and creation to create an autonomous system.

Drones can also be used by instructors to teach systems engineering. In "Using Multirotor Drones in Engineering of Systems Curricula", Yakimenko says that courses that use drones help students through the four metaphases of systems engineering: conceiving, designing, implementing, and operating [12]. Drones provide hands-on experience. It is an opportunity for students to use the language, terminology, concepts, methods, and tools needed to be a system engineer. This thesis builds on this paper by enabling the students to conceive, design, implement and operate their system.

One way to teach DA development is by using game design. In "Teaching Computer Science Through Game Design", Overmars says that game design can help explain many parts of computer science, including artificial intelligence, multiplayer games, graphics, and physics simulations [13]. He talks about how game design helps create interest in many aspects of computer science. Most multiplayer games are distributed applications since they involve multiple processes that share resources and coordinate on a single, i.e., playing the game. That interest in game design can be used by instructors to teach distributed systems. By using a game to teach distributed systems, the students should be more interested and involved, leading to a better understanding of the concepts that the instructor is teaching.

In "A Design Methodology for Choosing an Optimal Pedagogy: The Pedagogy Decision Matrix.", Malicky talks about using a decision matrix to choose which pedagogy to use for any given class [14]. An instructor will place several factors in four decision matrices. Those matrices include the students, the teacher, the institution, and the course. Each one is broken down into sub-factors to help decide which pedagogy is best. The pedagogies used by Malicky are subject-based learning, subject and project, subject and cooperative, problem-based learning, and project-based learning. The research in this thesis builds on Malicky's research. However, instead of using decision matrices to choose a pedagogy, it uses decision matrices to compare assignments, allowing an instructor to select the best assignment in a project-based pedagogy.

CHAPTER 4

DALP

DALP is a software development platform that aims to facilitate hands-on learning of DA design, implementation, and testing by supporting non-trivial assignments involving the coordinated use of multiple quadcopter drones. DALP creates a controlled environment where instructors have control over learning objectives while giving students the freedom to design their project.

To enable the teacher to verify the quality of the students' players, DALP includes instrumentation that measures metrics for inter-process communication, such as the number of messages received, targets completed, and targets missed. The assignment is broken up into three milestones, each focusing on different core concepts and skills.

There are four main components in DALP: referee, drones, drone simulator, and test suites that are referred to as pretests. See Figure 1.1 for more details. The pretests verify that the student project communicates correctly with the drones, drone simulators, and referee, as well as give the instructor a view of how well the student projects are working. The drone simulator allows the students to develop the system without the use of the physical drone, but it is not a replacement for the physical drone. The drone simulator gives students the flexibility to program anywhere in a controlled environment instead of only programming in a large enough area for the drones to fly. However, the students still need to test their project with the physical drones. The drones used in this implementation are the Tello drones.

A sample use case for DALP creates a game where the user wants the drones to hit specific targets and specific locations. The drones can detect the targets and report which target was hit by detecting a target identifier. The referee gives the target locations to the system and will verify the correct target identifier for each target location. Upon receiving a correct target identifier, the referee can add or remove targets from the system. The

system completes the game once the drones have hit all of the targets.

With this use case, students would design and implement a system to retrieve the targets from the referee and distribute them to the drones. The drones will discover the target identifier at each target location and send the detected target identifiers to the referee. A monitor process is created to give insight into how the system is running. The monitor is a helpful tool to see what events are happening in the project. Students can use this to debug their DA, and instructors can use it to see what is happening internally in the DA.

One architecture that students can use in this scenario is a ground station to handle the communication with the referee, a drone controller to communicate with the drones, and a status manager to listen to the status. Another example is a peer to peer system, where several applications of the same type communicate together, with the referee, and the drones. One possible solution is the use of a single process to connect the referee to the drones. This research recommends that students do not use this solution. Using only one executable to connect the drones to the referee will make the system less cohesive, and therefore less testable and maintainable. The students will not only have a harder time getting the system to work, but will also have fewer opportunities for learning.

If necessary, a program can end the game prematurely by sending a *Finish Message* to the referee. The *Finish Message* allows a system to finish the game even though the system has not successfully completed the game. This is useful if the system has encountered a critical and knows it cannot finish successfully. See Section 4.1 for more detail about the *Finish Message*.

4.1 Referee

The first part of DALP is called the referee. The referee's job is to supervise the system and to enforce the rules of the game. One or more processes can connect to the referee. Each instance of a game will have a game ID. The referee can supervise multiple games at a time. All communication with the referee except the start message must contain a game ID. The game ID will come from the response to the start message.

If a student would like more than one process to connect with the referee in the same game, they can. The referee creates a game ID for each game. The students must make sure that all of the processes that communicate with the referee know the game ID. The game ID will be a shared resource amongst those processes. The referee can manage multiple TCP communication channels at a time.

The main difficulty that students will have communicating with the referee is understanding how to send each message. Each message must start with the sync byte, 0x01. Since the data comes as a stream, it is not always clear where one message starts and one message ends. The sync byte will signal the beginning of a message. Following the sync byte is five bytes that represent the size of the message, not including the sync byte and size bytes. The five bytes will help the parsing of the message.

Every message contains three fields, a message ID, a type, and a message string. The message ID provides a way to tie the sending and receiving of messages. The type field represents what message type is represented by the message string. The message type makes it easy for the deserialization of the messages. The message string is the data that needs to be sent to a different process.

The class diagram for the referee is Figure 4.1. There is one class called the *Referee Controller* that controls the referee at a high level. It contains a *TCP Server* and a *Game Manager*. When the *Referee Controller* receives a connection from the *TCP Server*, it gives the connection to the *Game Manager*.

In the referee, the game manager will have access to all of the TCP connections. When another process sends a *Ready Message*, the *Game Manager* will start a timer and get all the available targets from the *Graph*. The *Game Manager* will also start keeping track of how many messages it has received. The *TCP Connection* then sends the next available targets to the process that sent the ready message. The *Ready Response* contains the game ID so that it can be distributed to any process that needs to communicate with the referee for that game. The *Ready Response* class diagram is Figure 4.3. The full sequence diagram for the *Ready Response* is in Figure 4.2.

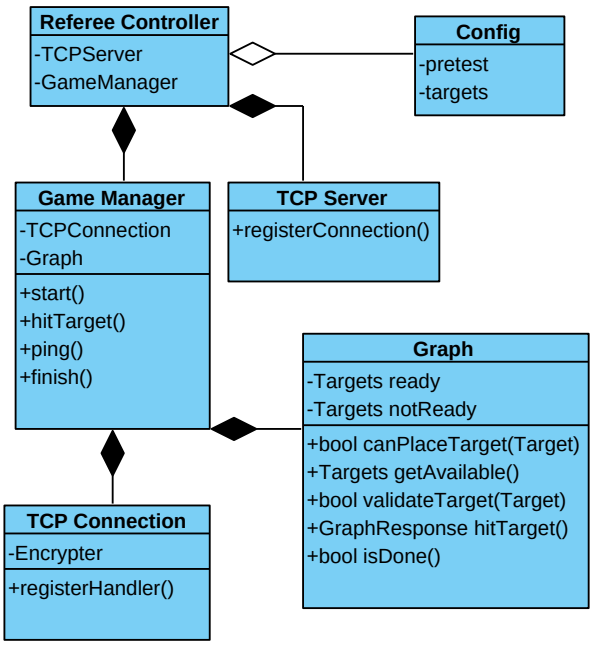

Fig. 4.1: Referee class diagram

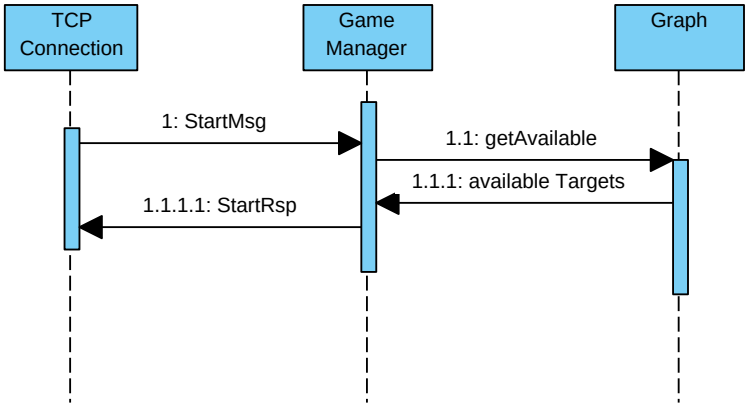

Fig. 4.2: Ready message sequence diagram

The referee will send a *Ping Message* every five seconds to every process that connects to it. The *Ping Message* verifies that processes that are connected to the referee are still responsive. Ping messages are essential in any DA. While a TCP connection guarantees delivery and will notify a process when the connection breaks, a broken link is not the only thing that can prevent a process from communicating. The process could not

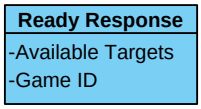

Fig. 4.3: Ready response class diagram

be responding because it is stuck in a while loop, or very far behind. The *Ping Message* has a sequence diagram like Figure 4.4.

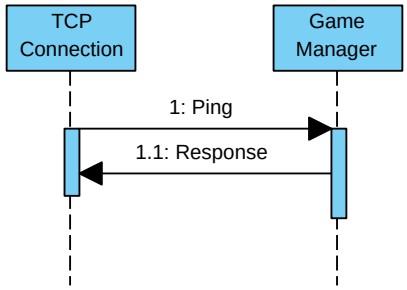

Fig. 4.4: Ping message

One of the messages that the referee can receive during a game is the *Hit Target* message. The *Hit Target* message communicates to the referee that a drone hit a target. Contained in the *Hit Target* message are the target ID and location. The target ID has to match the location and must be ready according to the *Graph*. If the target is not ready, it follows the sequence diagram in Figure 4.5. If it is ready, it follows the sequence diagram in Figure 4.6.

Each time a target is successfully hit, the system gets closer to finishing. The referee will respond with one of three situations. The first situation is that the *Graph* has no new ready targets, and the *Graph* did not remove any targets. This situation means the system can continue functioning as it was. The second situation is when the referee responds with new targets. The system must adapt to be able to hit those new targets.

The last situation removes required targets. Targets that have been removed by the *Graph* do not need to be hit by a drone anymore.

Once the drones have hit all of the targets, the *Hit Target* response will let the process know that it has successfully hit all of the targets. At this point, the referee will print out the final statistics. The statistics show how long the system took to hit all of the targets. It also displays how many of each message was received and what targets were hit by drones during the game.

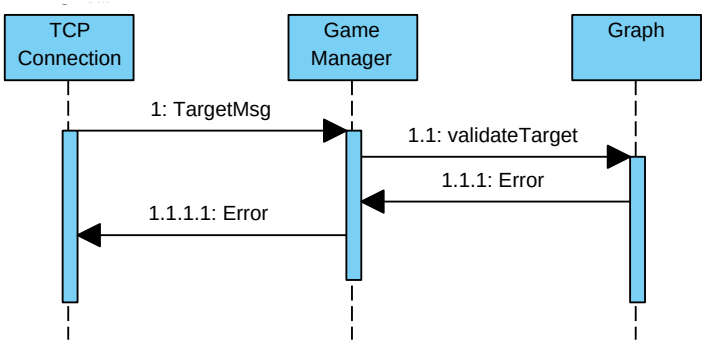

Fig. 4.5: Hit target message error

If the drones have not hit all of the targets, but the system does not know how to continue, it can give up. A process can send a *Finish Message* to the referee. If the referee receives the *Finish Message*, it will get all of the remaining targets that have not been hit and send them in the response. The referee will then print out the statistics and stop the game. It will print out the same statistics as if it had succeeded but will add information about which targets had been hit by a drone and which were not. This message is to a student to see how close they were to finishing. This sequence diagram is in Figure 4.7

4.2 Drones

The drones currently used in DALP are Tello drones. Tello drones have a UDP interface that can be used by another process to control the drone. Specific pre-defined

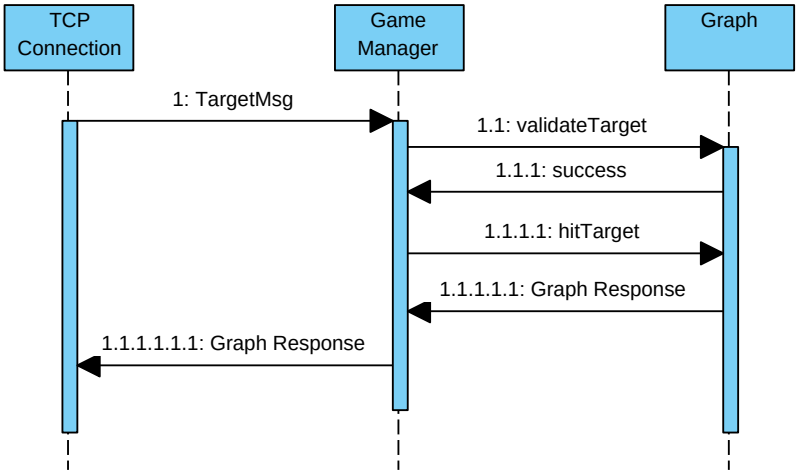

Fig. 4.6: Hit target message success

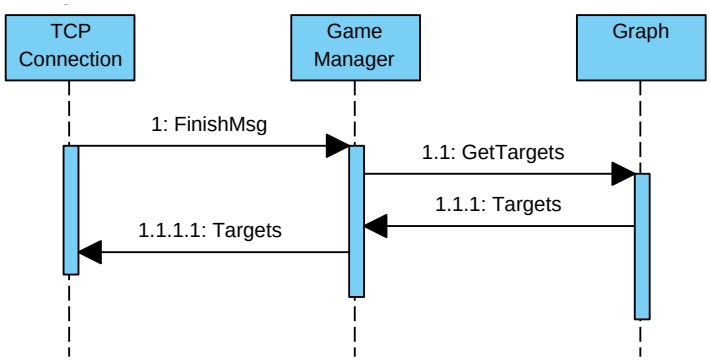

Fig. 4.7: Finish message

commands can be used by another process to control the drone. The full drone API can be found online at https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf.

There are a few issues with the API that students need to be aware of while using Tello drones. Sometimes the drone will respond with random bytes or not respond at all. The miscommunication provides the students with an opportunity to overcome unreliable communication. There are also a few messages that the drone supports that are too unreliable that the students should not use, specifically the stop message.

It is not always possible to program with the physical drone. A drone simulator comes as part of DALP. With the drone simulator, students can program their system without always having to use the physical drone. However, the simulator does not support all of the commands that the physical drone supports. Only the necessary commands are implemented in the simulator. Any message that is not in the simulator is not required to complete the system. The simulator is great for development, but the students still need to test and use the physical drones.

Another benefit of using Tello drones is the use of mission pads. The mission pads are mats that can be placed on the ground by the students that the drone can detect. Each one was an identifier between one and eight. These mission pads are the targets, and the identifier on each target is the target ID. A mission pad will be at each target location. The referee will know that a drone has hit a target if the system sends the correct target ID and coordinates to the referee. The coordinates do not have to be the exact coordinates but can have a margin of error of 20cm. This margin of error means that no targets can be within 40cm of each other as not to confuse two targets.

There is also a camera on the Tello drone. In the future, the platform can be modified by another researcher to accept pictures. The system could then send different images to the referee. The Tello drone provides multiple ways that it can be used to teach DA's.

The drone simulator uses a message hierarchy to do all of the serialization and deserialization. The base class is called *Message*. Several messages inherit from that class, such as *No Param Message*, *One Param Message*, *Flip Message*, and *Go Message*. Several messages inherit from *No Param Message* and *OneParamMessage*. There is a message factory that will parse any string and return the correct message. This is shown in Figures 4.8, 4.9, and 4.10.

Not all messages that the drone supports are supported by the simulator, some messages are considered irrelevant or too unreliable. The drone simulator will keep students from using unreliable messages by not supporting those messages. The messages that the simulator can use are: *command*, *takeoff*, *land*, *up*, *down*, *left*, *right*, *forward*, *back*,

cw, *ccw*, *flip*, *go* (without mid), *speed*, *mon*, *moff*, and *mdirection*. The *ap* message is required to set up the system, but will not be supported by the simulator.

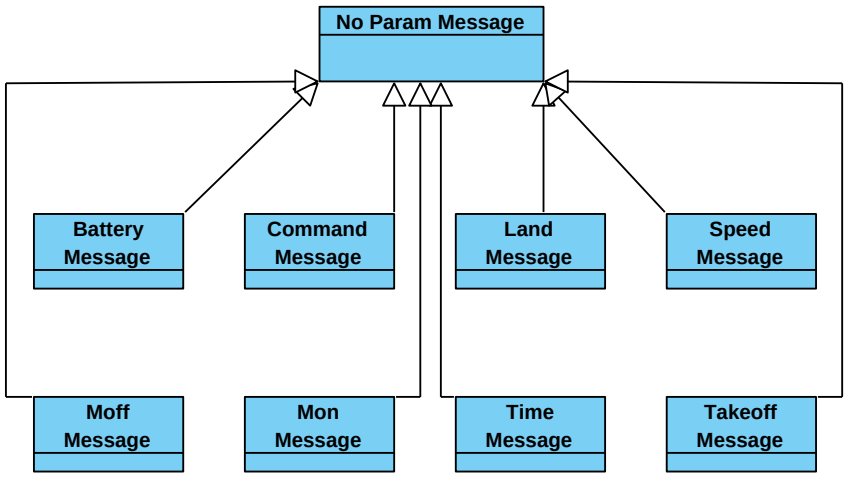

Fig. 4.8: No parameters messages

The drone simulator uses the state pattern (see [2]). The drone simulator contains a class called *Drone Simulator State*. That *Drone Simulator State* can either be a *drone simulator idle state*, *Drone Simulator SDK State*, or *Drone Pretest State*. This is shown in Figures 4.11 and 4.12.

The *Drone Simulator State* starts as a *Drone Simulator Idle State*. In this state, every message will return an error except the command message. The command message must be the first message to be sent to the drone or the drone simulator. If the *Drone Simulator Idle State* receives a command message, it will change state based on the pretest value in the configuration file. For more information about the pretests, see Section 4.3.

The *Drone Simulator SDK State* is the most complicated of the three states. In this state, the drone simulator will act like it is the drone. The class diagram for the *Drone Simulator SDK State* can be found in 4.13. It contains a *Drone Simulator State Impl* that splits all of the status and configuration values into a configuration object and a location object. When a message is received that takes some time to complete, the *Drone*

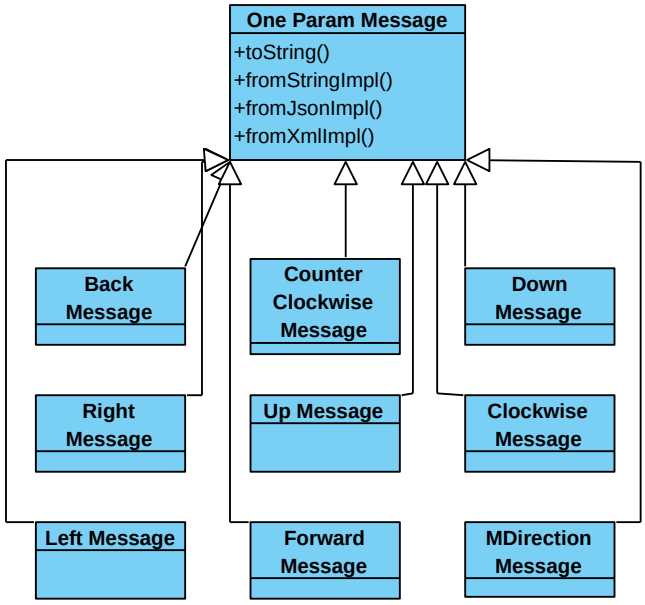

Fig. 4.9: One parameters messages

Simulator State Impl creates a drone updater. The drone updater will then update the *Drone Simulator State Impl* periodically.

The *Drone Simulator State Impl* runs in a game loop, and runs ten times per second. The game loop does not take long to run, but since the status message is sent only ten times per second, there is no benefit to running the game loop faster. The game loop is in Figure 4.14. The game loop runs the updater on the location object and will update the configuration if necessary. Once the updater has finished, the *Drone Simulator State Impl* deconstructs the updater sends back a response. This can be seen in Figure 4.15.

4.3 Assignment Milestones and Pretests

Assignment based on DALP will typically contain at least the three milestones: achieving basic communications, achieving reliable communications, and encrypting communications. To help student (and instructors) check their implementations, DALP contains some built-in test cases, called pretests, that can verify completeness and

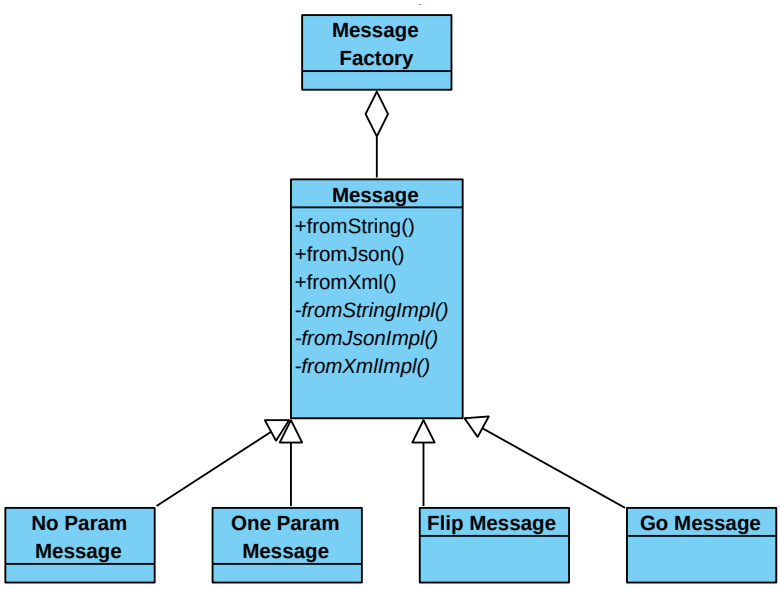

Fig. 4.10: Message factory

correctness relative to each of these milestones.

The first milestone, which involves the successful sending and receiving of messages, is supported by pretests in both the referee and the drone simulator, since the students code will need to communicate with both. The pretest in the referee will test TCP communications, while the drone simulator will test UDP communications. The first pretest in the referee is a standard TCP test to verify that a process can send all of the messages to the referee. All of the messages with the referee can be serialized with either Google Protobufs, XML(Extensible Markup Language), or JSON(JavaScript Object Notation). This allows the students to decide which serialization protocol best fits their system. To see the API for the referee, refer to appendix A. The first pretest in the drone simulator is a standard UDP test to verify all of the messages to the drones can be serialized.

The second set of pretests pertains to reliable communication and are, like the first set, in both the simulator and the referee. The second set of pretest is similar, but it does not always respond normally. There is a change it will either respond twice, wait to

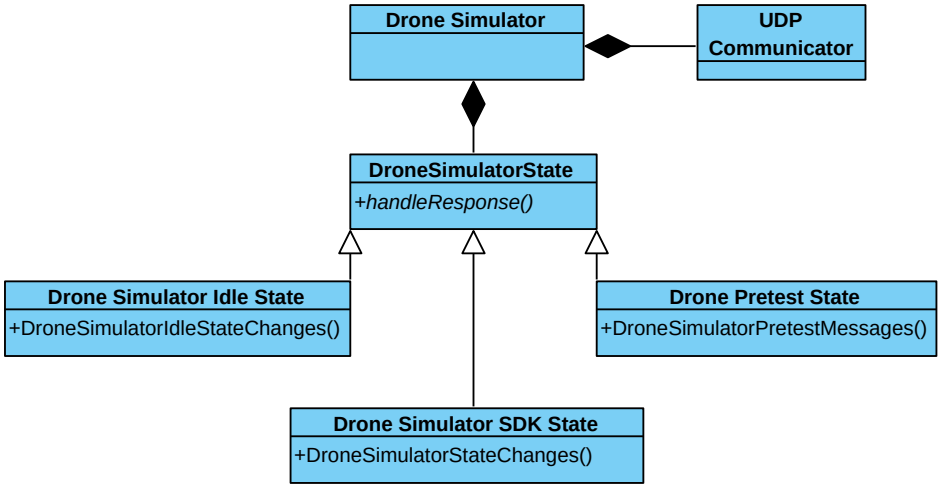

Fig. 4.11: Drone simulator class Diagram

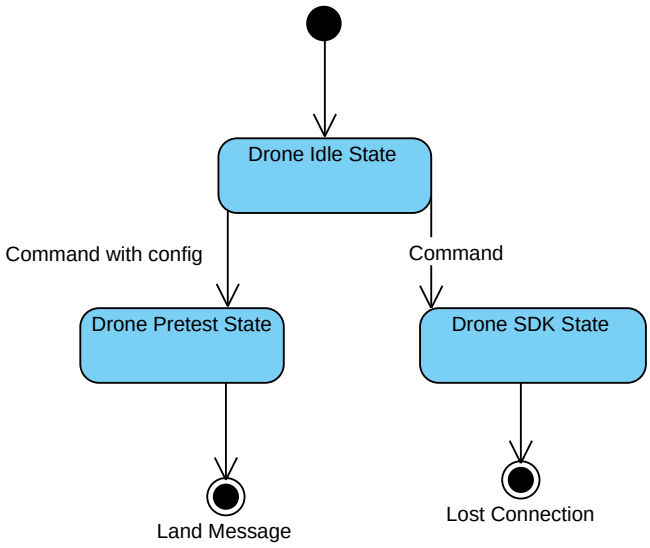

Fig. 4.12: Drone simulator state diagram

respond, or not respond at all. If the referee does not respond, it expects the message to be sent a second time. If the referee does not receive the message a second time, that part of the pretest will fail. If a process does not receive the response to a message, it should assume that the first message failed and retry. After receiving the finish message, the

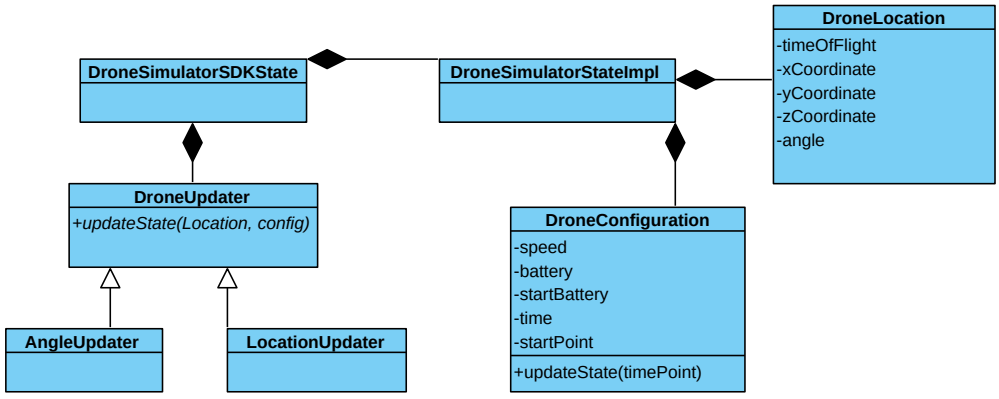

Fig. 4.13: Drone simulator SDK class diagram

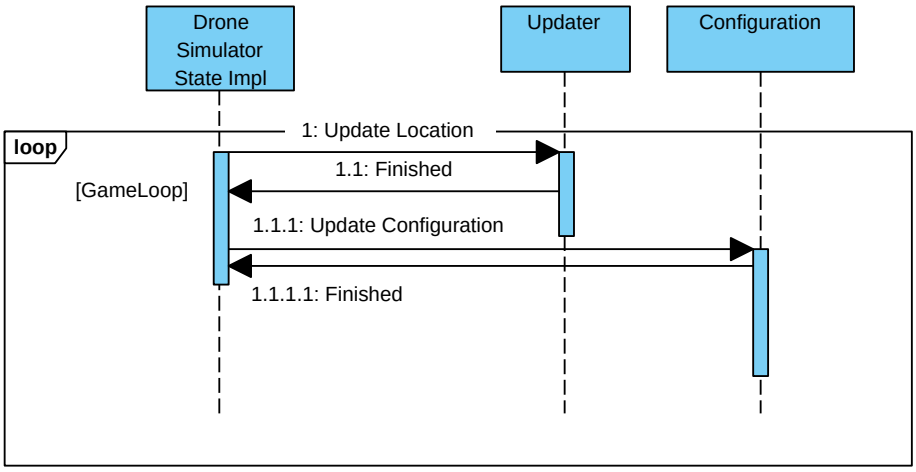

Fig. 4.14: Drone simulator game loop

referee will print out the same status as the first pretest.

The third set of pretests is just built into the referee, since only the referee supports encrypted communications. Specifically, the referee will encrypt all communication use RSA (Rivest, Shamir, and Adelman) and PKCS (Public Key Cryptography Standards) #1 v1.5 padding. Since communication with the drone is not encrypted, no messages sent to the drone simulator are encrypted. Encrypting messages with the referee is optional, as

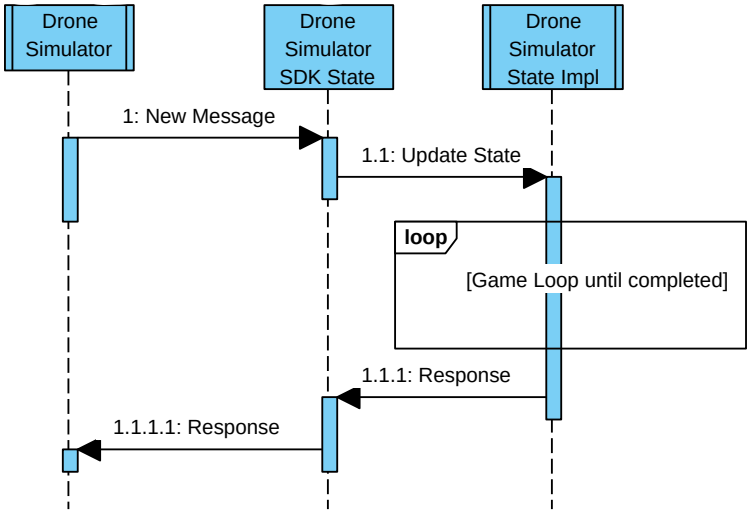

Fig. 4.15: Drone response interaction diagram

students may choose to encrypt data to the monitor. This allows students to choose which encryption algorithm to use. The third pretest builds on the first pretest as well. For this pretest, all messages need to be encrypted. Upon connection, the referee will give the public key to the connecting process. If a message was received but was not encrypted correctly, it will show that the message was not received, since it could not decrypt it to see what kind of message it was. The third pretest is optional. The students have the option between encrypting messages with the monitor or the referee. If the students chooses to encrypt the messages that go to the monitor, they do not need to pass this pretest but still need to demonstrate their encryption.

If a student want to execute one of three sets of pretests, the student simply needs to set the pretest parameter in the configuration file to 1, 2, or 3. If the students does not set the configuration parameter or set it to 0, no pretests will be run. Both the drone simulator and the referee will execute normally. The referee contains three pretests, while the drone simulator contains two. A full explanation of the configuration file can be found in appendix B.

In order to create consistency between the three pretests in the referee, a single class

managed all three called the *Pretest state* (see 4.11) For every message it receives, it will send back a response, or purposefully not respond. The *Pretest State* uses the sequence diagram in Figure 4.16. The referee will wait till it receives the *Finish Message*. Once the *Finish Message* is received, the referee will print out a status of which messages it received and which it did not.

The drone simulator contains two pretests. To enable a pretest, the pretest setting in the configuration needs to set. If the configuration value has been set to one or two, the simulator will go into the pretest mode once it has received a *command* message. The *Drone Simulator Idle State* checks the configuration when it receives a *command* message and will start the appropriate pretest.

The first pretest verifies that all of the valid messages can be received. It starts with the *command* message and ends with the *land* message. It will wait for any message and pass that message to the *Drone Simulator Pretest Messages* class. The *Drone Simulator Pretest Messages* class keeps track of what messages the *Pretest State* has received. After the *land* message is received, the pretest prints out which messages the *Pretest State* had received and which it had not. This is shown in Figure 4.17

The second pretest builds on the first. However, with the second pretest, the drone simulator does not always respond. There is a one in ten chance that the drone will respond with random bytes. Instead of replicating completely random bytes like the drone, it responds with the words "random bytes" in Leet (See [15] for more information on Leet). There is a one in ten chance that the drone simulator waits a few seconds to respond. There is also a one in ten chance that the drone simulator does not respond at all. If the drone simulator does not respond or it sends back random bytes, the simulator will expect the messages again in order to pass this pretest.

The third milestone also contains a meeting with the instructor. The students must demonstrate their system to show how well it worked. Perhaps more important, though, is the self-reflection of the project. Did they make the right design decisions, or were there better solutions? The self-reflection enables learning that cannot replicated through only

programming.

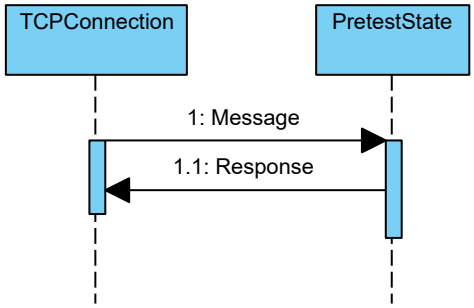

Fig. 4.16: Pretest sequence diagram

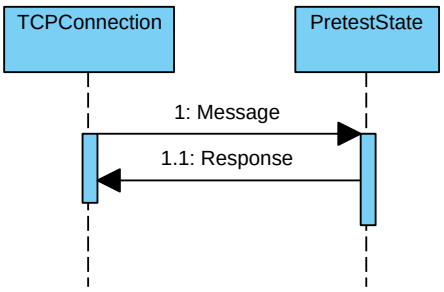

Fig. 4.17: Drone pretest interaction diagram

4.4 Monitor

The monitor is written entirely by the students. There is only one requirement for the monitor, it needs to show the current status of the system. This will help the students debug their system as well as give the instructor a view into their system. If the students choose not to encrypt communications with the referee, the communications with the

monitor need to be encrypted.

The monitor can be simple or complex, it is entirely up to the students. They will design the interface and the messaging for the monitor. Multiple processes can connect to it or a single process. The monitor is a place where the students get to have freedom and flexibility. It is recommended this is started early on in the semester. The earlier that the students make it, the more useful of a tool it will be to them.

4.5 Student Processes

For this sample use case of DALP, the students will have to design more than just the monitor, they must also design one or more programs that will combine the entire system. One or more processes must be in charge of talking with the referee. One or more processes must also communicate with the drones. It is recommended that students use more than one process to combine the entire system.

The students will need to define any messages that go between their processes. The students will be able to choose what protocol and serialization to use to send those messages. That protocol can be the same or different than what they used to communicate with the monitor, referee, and drones.

There are many design decisions that students need to make. If they are not going to use the referee's encryption, they must determine what encryption algorithm to use. It is recommended that they do not write their own encryption algorithm, but use an encryption algorithm that the industry uses today. Most students will use a third-party library to do this, which will also give the students experience with third-party libraries.

Because students will be writing more than one executable, they do not have to use only one programming language. If the students want to use one language for the monitor and another language for the other process, they can. This allows more flexibility for the students.

CHAPTER 5
VERIFICATION AND VALIDATION

5.1 Verification

This research uses three different methods to verify the correct behavior in DALP: unit tests, integration tests, and system tests. The unit tests in DALP take a path-based approach to get the best coverage. Every class in DALP has an associated test file that verifies correct behavior in that class. The unit tests have above 80% line and function coverage.

The pretests provided by DALP perform the integration tests. There are two UDP pretests and three TCP pretests. The first UDP pretest verifies correct serialization, deserialization, and sending of each message. It creates a minimum standard for message passing over a UDP interface. The second pretest builds on top of the first and verifies reliable UDP communication. If a message does not get an answer, it will send the message again.

The first two TCP pretests are similar to the UDP pretests. One pretest creates a minimum standard to verify that the DA can communicate with DALP, while the second pretest verifies reliable communication. The third verifies encrypted communication using RSA encryption. A different private and public key are generated each time, and the public key is sent to the TCP client.

To get more system tests, I created a mock ground station. It enables the testing between the drone controllers, drones, status manager, and the monitor. The mock ground station can load up targets in a pre-defined way and distribute the targets. The mock ground station enables the testing of half the system in a controlled environment. This system is performed with either: one simulator, one drone, two simulators, or two drones.

An end-to-end test is performed with the entire DA and can be shown to others to

verify the correct behavior of the system. The referee loads targets from a configuration file, and the ground station tells the referee when to start. Different configurations of targets can be loaded to perform different tests on the system.

5.2 Validation

As part of DALP's validation, I created a sample implementation of two sample assignments that use. The first sample implementation uses a sample assignment that uses the entire system, while the second implementation only uses a sample assignment that uses the suite of pretests. The first sample implementation consists of the monitor, drone manager, ground station, and status manager. The second implementation consists of two python files, one verifies UDP communication, while the second file verifies TCP communication using JSON to serialize the data.

5.2.1 Sample Implementation C++/C#: Full System

This sample implementation included the drone manager, ground station, and status manager, all implemented using C++, plus a monitor implemented in C#. The build process used cmake. 5.1 show a high design for the sample implementation.

C# Monitor: Full System

The monitor is a simple program that has a TCP server. It can communicate with as many processes as needed. For every process that connects to the monitor, a status brick appears on the monitor. Anything that the connecting process sends to the monitor appears in the status brick. The monitor uses WPF to display information to the user. The class diagram is in figure 5.2.

In this implementation, the communication with the monitor is encrypted. It uses a TCP socket to send logs to the monitor. All messages to the monitor use Google Protobufs to serialize and deserialize. The interaction diagram is figure 5.3.

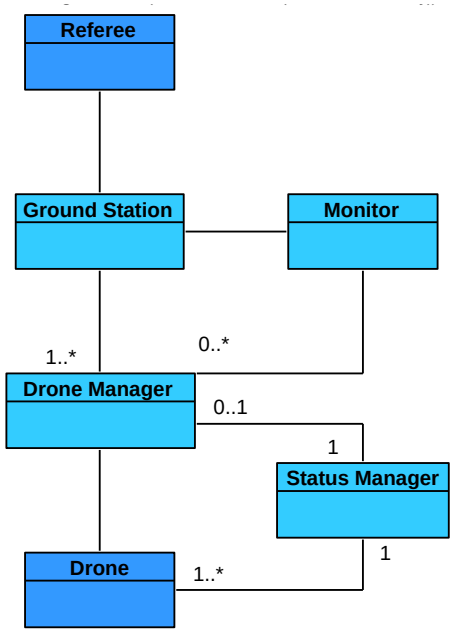

Fig. 5.1: High-level design for sample implementations
10

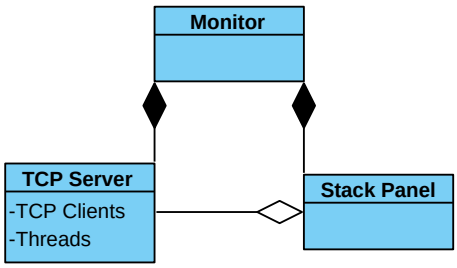

Fig. 5.2: Monitor class diagram

C++ Functionality: Full System

The first executable is the drone manager. One drone manager is used for each drone. The job of the drone manager is to handle all of the communication with a specific drone. The drone manager's class diagram is in Figure 5.4.

The interaction diagram for the drone manager is in Figure 5.5. The drone manager will turn targets into a flight path of messages that will be sent to the drone. Since the drone can only receive messages with a distance between 20cm and 500cm, special care

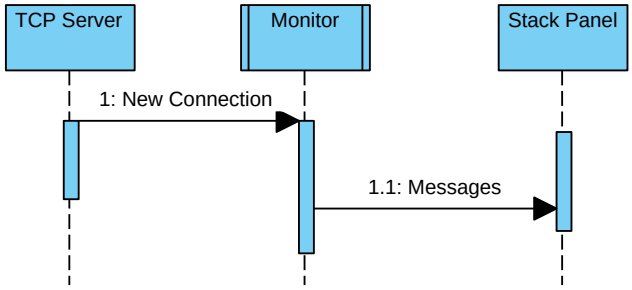

Fig. 5.3: Monitor interaction diagram

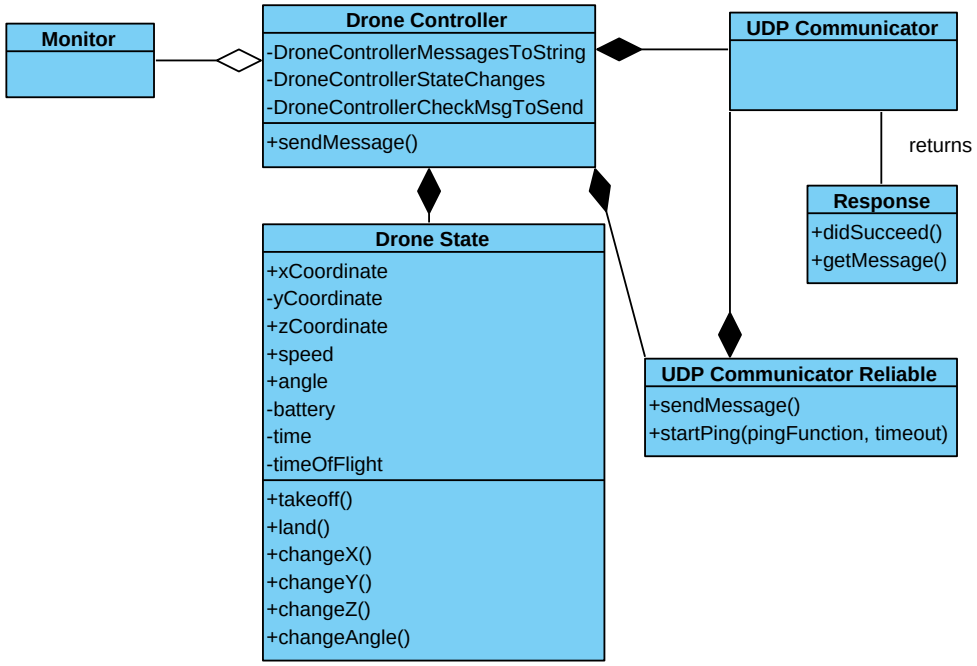

Fig. 5.4: Drone manager class diagram

needs is taken when creating the flight path. Every message that the drone manager needs to send to the drone is sent via the drone controller.

The drone controller then uses a visitor called the *Drone Controller Check Msg To Send*. The *Drone Controller Check Msg To Send* will check the drone state to see if the message can work. This way, the drone controller catches messages that are guaranteed to

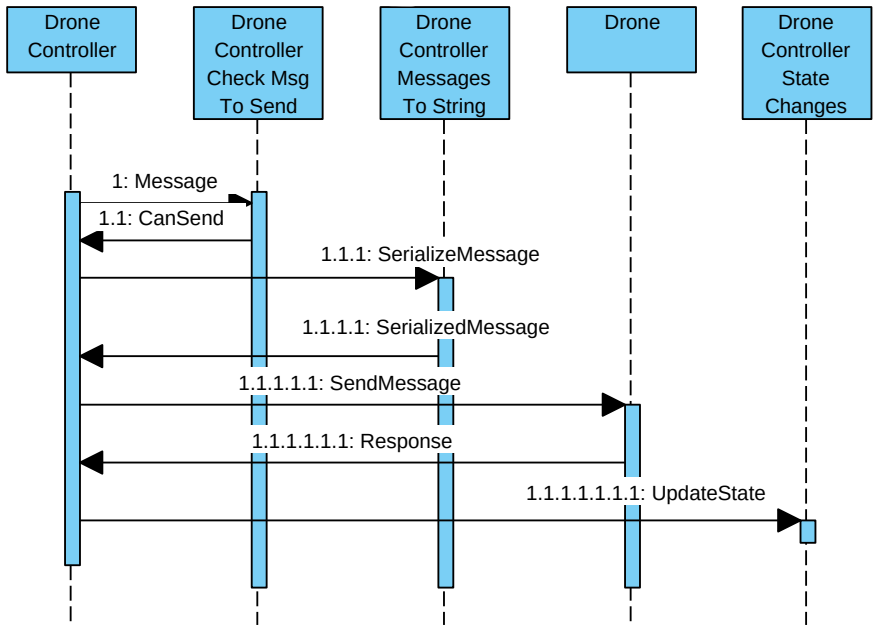

Fig. 5.5: Drone manager interaction diagram

fail even before they leave the drone manager. Once it knows that the message can succeed, another visitor called the *Drone Controller Messages To String* serializes the message. This visitor adds special rules for message serialization.

The drone controller then sends the message to the drone. The drone can respond with an "ok" or an error message. The drone controller gives the response and the original message to a third visitor called the *Drone Controller State Changes*. The *Drone Controller State Changes* will update the internal state of the drone manager to allow verification of future messages.

The second executable is the ground station. There is only ever one ground station running at a time. All of the drone managers will connect to the ground station. The job of the ground station is to communicate with the referee and coordinate all of the drone managers. It will coordinate each drone to fly at different altitudes and go for different targets. The class diagram of the ground station is found in figure 5.6.

The ground station contains a TCP server and two TCP clients. One of the clients

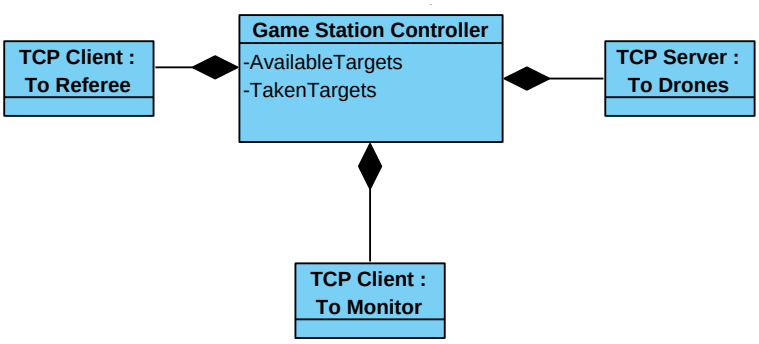

Fig. 5.6: Ground station class diagram

connects to the referee, while the other connects to the monitor. All of the communication with the referee in the system will go over this connection to the ground station. The TCP server connects to all of the drone managers.

Once the system is all set up and ready to start, the ground station will send the *Ready Message* to the referee. The interaction diagram can be found in Figure 5.7. The referee will respond with the *Ready Targets Message*. The ground station will then tell each of the drones what altitude to fly at with a *Z Config message*. Each drone manager will then receive a flight path.

The drones send out a status message ten times every second to 0.0.0.0:8890. Having two drones listen to the same port creates an issue when running two drone managers on the same system. One manager will be able to bind to port 8890, and the second drone manager will fail to bind to it since the port is already bound. Two processes may bind to the same port by setting reuse_addr on the socket, but only one process will receive the messages. The sample implementation gets around this issue by using a status manager. The status manager listens to all of the traffic on port 8890 and will send the appropriate status message to the appropriate drone manager. The status manager allows the drone manager to receive the necessary information and does not need to be aware of any other drone managers or drones other than the drone it manages.

When the drone detects a target by listening to the status message, it follows the

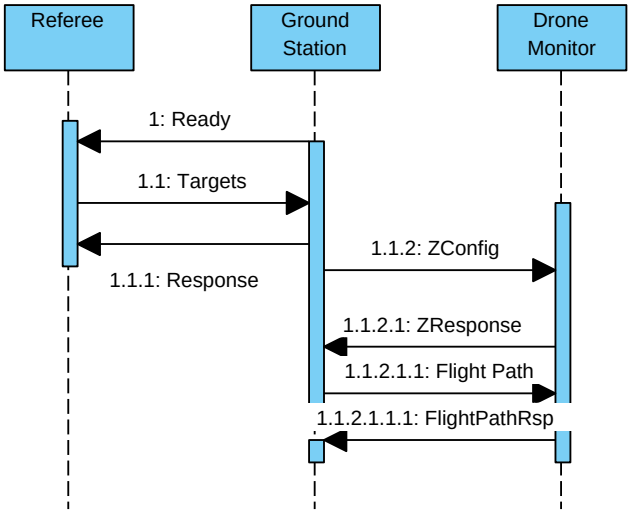

Fig. 5.7: Ground station ready interaction diagram

interaction diagram in figure 5.8. The drone manager controlling the drone will be informed when a target has been hit. The drone manager will verify the target is in the correct location. If it is, it will tell the ground station. The ground station will then forward that message to the referee. The ground station will then send the next target to the drone manager.

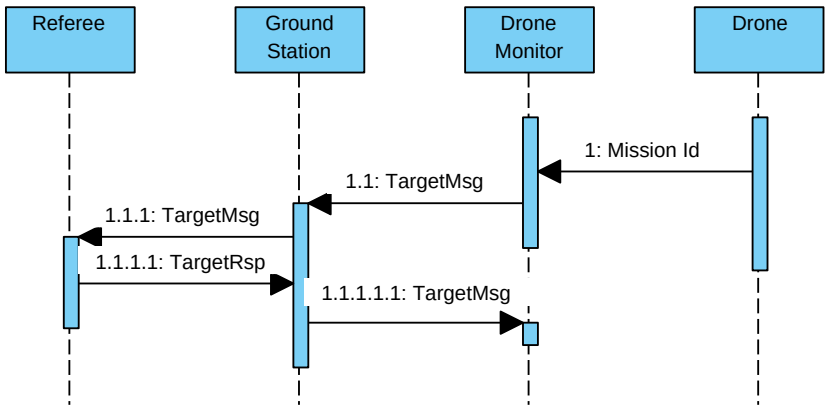

Fig. 5.8: Target interaction diagram

The status manager's job is to send the status message of each drone to the appropriate drone manager. Upon startup, it receives a string of IP addresses to ports. If the drone manager receives a message from one of the IP addresses, it forwards it to the port that is tied to that IP address.

5.2.2 Sample Implementation Python: Pretests

Another sample implementation using DALP was created using python to verify correct behavior of the pretests. For the UDP pretests, the python code sends the necessary information to the pretest and verifies reliable communication. This validates that communication can be established, and that the pretests can be used with different languages. For the TCP pretests, the python script serializes and deserializes using JSON. It also passes the third pretest to verify the encryption with the referee is correct.

CHAPTER 6
EVALUATION

6.1 How to Evaluate

A popular way to teach distributed applications is by using a project-based pedagogy. The assignment used needs to teach distributed applications in a way that students will learn how to design and build a distributed application. The programming involved in the project needs to give students proper experience to solidify distributed application concepts. This chapter explains how to quantify the effectiveness of assignments used to teach distributed applications. The more effective the assignment, the more students will learn and understand how to design, build, test, and maintain distributed applications.

A popular tool to quantify different solutions to a given problem is decision matrices. These matrices help put different variables into one chart to help compare everything. Decision matrices break down a decision into smaller and easier pieces. Each smaller decision is given a value between one and five. The better it is, the more points it is given, and in the end, the points are all added together. The higher the total, the better the option.

Certain variables may be more important than others. For those cases, a weighted decision matrix can be used. Each variable has a weight associated with it. The number of points given to each variable is multiplied by the weight associated with that variable. The total value is then divided by the total of the weights. The division results in a value between 1 and 5. By using weights, certain variables have more of an effect on the end value. One crucial decision does not get outweighed by one smaller decision.

Some decision matrices have sub-decision matrices. The sub-decision matrices behave just like a decision matrix, where the number of points is added up and then divided by the total of the weights. This value is then put into the main decision matrix.

This research uses decision matrices to quantify the effectiveness of assignments. In "A Design Methodology for Choosing an Optimal Pedagogy: The Pedagogy Decision Matrix.", Malicky explains the benefits of using a decision matrix to choose which pedagogy to use [14]. This thesis aims to build on that and use a decision matrices to select which assignment to use in a project-based pedagogy.

An instructor needs to take into account four main factors when choosing an assignment for a distributed applications course. Those factors include the flexibility a student has to design their project, the topics taught by the assignment, how well the assignment can adapt to different needs, and how frustrating is the assignment for the students.

Each factor has a different priority. An instructor can put the factors into a weighted decision matrix, so the more essential factors do not get overshadowed. Those priorities can change based on different needs, at which point an instructor can change the weights of the decision matrix, and the decision matrix will calculate the new result.

This thesis breaks each factor down into a weighted decision matrix and then evaluates each sub-factor. One of the most significant benefits of using a decision matrix is that when an instructor creates a new assignment, it can be added to the decision matrices and compared to previous assignments. It creates a single way to compare assignments in a project-based pedagogy. To see how DALP compares to other assignments using decision matrices, see Section 6.3.

6.1.1 Flexibility

The first factor of project effectiveness is flexibility. A teacher wants their student to have the freedom to design their projects. Not only does this encourage imagination from the students, but it gives the students experience designing a DA. If the project is too rigid, there will be no design work in what the students are creating. In order to give the students as much experience designing as possible, there must be as many design decisions as possible.

The bigger the design decisions are, the wider variety of solutions there will be. In

order to have a valid design decision, there must be multiple valid solutions. If there is a design decision with only one logical solution, it does not inspire creativity from the student as much as two valid solutions would.

Another aspect of flexibility is the timeline students have to finish their projects. Does the project need to be created in a specific order, or can the students work on it in any order? It is helpful to the students if they can work on the project in an order they prefer, and in a timeline that works best for them. A flexible schedule does not mean the students should procrastinate. While flexibility may encourage procrastination, milestone deadlines will mitigate procrastination. This way, students cannot delay the entire semester to start on the project.

6.1.2 Knowledge, Skills, and Abilities

The most critical factor is teaching the most valuable knowledge and skills while letting the students develop and hone their abilities while maintaining a reasonable workload. Each knowledge area and skill is assessed in this research to see how well the assignment teaches it while providing an environment to develop abilities. It is essential to know how well each knowledge area, skill, and ability is covered. The skills, knowledge areas, and abilities that are used in this research to evaluate are:

1. Characteristics of distributed systems
2. Opportunities and challenges for distributed systems
3. Core design issues
4. Common system models
5. Networking and internetworking
6. Protocols, protocol families, and protocol design principles
7. The TCP/IP protocol suite
8. Inter- and intra-process communication

9. Partial failure
10. Security issues
11. Concurrency controls
12. Distributed objects
13. Distributed file systems
14. Time services
15. Client-server and peer-to-peer architectures

For more information on the skills, knowledge areas, and abilities listed in the table above, see Chapter 2.

6.1.3 Adaptability

Assignments need to be adaptable to remain current in today's fast moving world. Can students add features to the project with effort that is commensurate to the complexity of the feature, or does the assignment's design artificial restrict the student or the extensibility of the program being built? Adaptability is essential because, as technology progresses, that technology will change how DA's will work. If the assignment does not adapt as technology advances, it will become obsolete. It is also essential to add new features for new technologies, as well as if the needs of the class change. If there is a concept that needs to be taught more in-depth for one class, can it be added to the assignment easily? It is also crucial that instructors can remove obsolete technologies in the assignment. The assignment should not teach technologies that the industry does not use.

6.1.4 Frustration Minimization

Frustration minimization is another factor that needs to be measured. The more frustrated a student gets with a project, the less likely the student will succeed with that

project. While it is important to be able to work through frustration, an assignment should not be frustrating just for the sake of being frustrating. Frustration in DA projects is broken down by this research into sub-factors. The first sub-factor is how much extraneous work does the assignment give the students. If there is a lot of irrelevant work that does not pertain to the subject matter, it creates resentment in the students towards the teacher or the course. When measuring extraneous work, it is only fair that only required extraneous work counts. Any optional self-inflicted extraneous work should not be measured since the students have elected to do it.

Some API's can be a source of frustration. Since programs in DA's communicate with each other, there will be some form of an API. Is the API in the system intuitive and usable? If an API is not well-documented, how are the students going to use it well? Another question is, how simple is the API? If the API is not straight forward, the students will get frustrated by it. It also needs to be reliable and consistent. If the API returns a wrong message every ten messages, students will become frustrated.

Another source of frustration can come from the hardware or software that a assignment requires. In the case of DALP, a source of frustration is the drones. Since it can be hard to always develop with a physical drone, DALP provides a drone simulator that can help students build their systems without the use of the physical drone. If the simulator does not act the same as the physical drone, the drone simulator will create more frustration for the students. If a student's system works with the simulator but not the physical drone, the students will get very frustrated. It is important to note that testing with only the simulator is not sufficient, and the students must test with the physical drones.

6.2 Decision Vectors

Another benefit of using a decision matrix to compare different assignments is that they are adaptable. If a new factor arises, a researcher can add that factor to the matrices. The researcher will then evaluate that factor against all of the assignments. The decision matrix will then populate the final decision.

Not all professors teaching distributed systems design focus on the same concepts. This difference is another reason why a decision matrix is suitable for comparing assignments. If someone has a different priority in their decision making, they can use different weights to see which assignment will work best for their situation.

Each design matrix uses a scale from one to five. One means that a concept was not present at all while five means the concept was very prominent. Two through four provide a scale of how prominent the concept was.

Tables 6.1, 6.2, 6.3, 6.4, and 6.5 describe the weighted decision vectors that this thesis will use to compare assignments. Each weighted decision vector contains the weight to each factor, and will be expanded into weighted decision matrices in chapter 7.

Factor	Weight
Flexibility	1.5
Skills and Knowledge areas	2
Adaptability	1
Frustration Minimization	1.5

Table 6.1: Main factors together in one weighted decision vector

Factor	Weight
Key Design Decision Opportunities	1
Timeline Flexibility	1

Table 6.2: Flexibility factors weighted decision vector

6.3 Results

It is essential to evaluate the results of how effective DALP is at creating assignments that enable students to learn the core concepts for DA's. This section quantifies the effectiveness of DALP based on the topics found in Section 6.1. It also quantifies two other assignments to show how DALP compares to previous assignment and shows that it advances the pedagogy of project-based learning in DA's. Those two assignments are an

Factor	Weight
Characteristics of distributed systems	1
Opportunities and challenges	1
Core design issues	1
Common system models	1
Networking and internetworking	2
Protocols and protocol families	1
TCP/IP protocol suite	2
Inter- and intra-process communication	2
Partial failure	1
Security issues	2
Concurrency controls	1
Distributed objects	1
Distributed file systems	1
Time services	1
Client-server and peer-to-peer	1

Table 6.3: Skills and knowledge areas factors weighted decision vector

Factor	Weight
Add new features	1
Remove features	1

Table 6.4: Adaptability factors weighted decision vector

Factor	Weight
Reliable API	1
Reliability of hardware/software	1
Minimizes extraneous work	1

Table 6.5: Frustration minimization weighted decision vector

instructor-directed assignment (IDA) and a student-directed assignment (SDA).

6.3.1 Flexibility

The first factor in the decision matrices is flexibility. The sub-factors key design decision and timeline flexibility both have a weight of one, meaning they both have the same amount of influence on the result. The IDA was given a one in key decisions opportunities because the instructor had already designed the system. The students were also restricted in their timeline because each process had to come in a particular order.

The SDA was given a five in key decision opportunities because none of the system

was designed for them. The students were able to design and create the entire system themselves. It received a four for the timeline flexibility as well because they could choose the order of implementation. It did not receive a five because certain parts had to be covered in class before the students could implement them in the project.

DALP received a three for key decision opportunities. While there are several key decisions that students can decide, there are not as many as the SDA. However, there are enough key decisions that each group project should be different from each other. DALP received a four for timeline because the students can work on most parts of the project at any time. The same components that had to wait in the SDA have to wait in DALP as well.

The weighted decision matrix for flexibility can be found in Figure 6.6. The SDA is the most flexible, whereas The IDA is the least flexible.

Factor	Weight	IDA	SDA	DALP
Key design decision opportunities	1	1	5	3
Timeline flexibility	1	2	4	4
Total	2	3	9	7
Total Average		1.5	4.5	3.5

Table 6.6: Flexibility factors weighted decision matrix

6.3.2 Skills and Knowledge Areas

The IDA consisted of applications all running on the same type of hardware so the programs could be all ran on the same machine. It wasn't very heterogeneous. You could add more programs, though, so it was scalable. The interfaces between each program were well defined, making the system more transparent. The SDA could have heterogeneity, scalability, and transparency, but it depended entirely on what project the students chose. Even if the students chose a suitable project, it was hard to enforce that their design was heterogeneous, scalable, or transparent.

DALP forces heterogeneity because of the drone. The system must run on different

types of hardware. The system is also scalable because the only limit to how many drones can be in a single system is the software the students write. The system is also transparent because there are two APIs that are static and will not change. If something on either side of the API changes, it will be easy to replace.

The challenges for creating distributed systems that this research uses to evaluate the projects are faulty communication, distance, and different programming languages. For the IDA, the data transfer was reliable. There was a chance of data loss, but it is not high. However, the application could take advantage of separate hardware for each application. However, each application can be created by the students with a different language. The SDA could experience data loss, hardware dispersal, and various programming languages, but it depended on the assignment. There was no guarantee on what the student would learn.

DALP uses a drone that has an unreliable UDP interface. While UDP has a small chance of data loss, there is a higher chance that the drone does not respond at all because of the drones itself. While having an unreliable interface in the real world is a bad thing, it is helpful in teaching. The drone provides an excellent interface for students to learn how to overcome data loss. It also can take advantage of splitting the applications across different hardware. In the future, it could use targets and drones in different rooms. Because part of the system is written in C++, there is a higher chance that the system will not consist of a single language.

The IDA had a hard time teaching core design issues because the system was already fully designed. The students could see the system design, but the students did not get the opportunity to design the system. In the SDA, the students came across a wide variety of core design issues. However, the SDA gave the teacher no control over which core design issues will be covered. DALP contains several core design decisions that the students will have to decide.

Since the IDA was already designed, the students did not cover as many common system models. The SDA helps teach a variety of common system models. However, the

learning was not spread evenly across the class. Some groups would learn some systems, while other groups could learn very different systems. Since half of the DALP project has already been designed, there are not as many models that the students can use; however, it does cover some of the common models.

There can be a lot of networking concepts that need to be understood when creating a distributed system. The IDA taught some of the concepts since part of the system could run in the cloud. The SDA depended on the project, some could have a lot of networking, while others could have close to none. DALP teaches several networking concepts as well. Each drone comes with a wifi SSID. To communicate with the drone, you must connect to the drone's wifi. Multiple drones can connect to the same network to communicate with more than one drone. The students need to set up a network, and the drones need to receive proper messages to connect to the network.

It is helpful for students to know many different protocols and protocol families. All three assignments do not cover protocol families very well. The only protocols that are covered are UDP and TCP. The SDA has the potential to cover more, but only if the students make the right design choice.

It is crucial, though, that the IP protocol family is covered. The IDA helped teach both UDP and TCP. Both were required to complete the system. The SDA tended to have one or the other, but not both. A SDA has the potential to use any protocol, not just UDP and TCP, but most ended up with only one. DALP requires both UDP and TCP. The students must make a TCP connection to the referee, and a UDP connection to the drone. The communication with the monitor can be any protocol and is not limited to just UDP and TCP.

Students must understand how to use interprocess communication. The students must be able to serialize and deserialize data to send to another process. There are many different ways to serialize data. The IDA had specific data types that had to be serialized. The SDA could either have a lot to serialize, or very little, but could use many ways to serialize the data. DALP requires serialization of both primitive and compound objects.

Data sent to the referee must be either JSON, protobuf, or XML.

An area of learning that is becoming more important in today's world is security. The students must know how to send encrypted data. In the IDA, one communication channel encrypted the data. The students chose the encryption algorithm. In the SDA, it was entirely up to the students; they just needed to encrypt something. They were also able to choose which encryption algorithm to use. In DALP, the referee has an encrypted mode where the students must learn how to interface with encrypted API. They can also choose to encrypt another channel of communication. The referee contains a pretest with encrypted traffic if the students decide to encrypt the data with the referee. However, only the SDA could encrypt data that is stored.

It was hard in the IDA to create an unreliable interface to create a learning environment for handling failure. The SDA could have an unreliable interface, but few did. DALP uses a drone that has an unreliable interface, so students must know how to overcome missed, delayed, and misconstructed messages.

It is helpful if the assignment can teach concurrency controls. The processes must coordinate over shared resources, otherwise the DA could have an integrity problem. The IDA had several shared resources that had to be managed. The SDA normally had a shared resource, but the students were not always aware of them. DALP has several shared resources, and if not shared correctly, the system will fail.

Distributed objects can exist in distributed systems in order to manage shared objects. In an IDA, there were several shared resources that needed to be managed by a distributed object. The SDA does not necessarily have one, but could also have a lot. DALP also has the potential of having several distributed objects, depending on how the students design their system.

Both the IDA and DALP do not contain any shared file system. The SDA could have one, depending on the assignment chosen. The same goes for a time service. There is no support in DALP or the IDA for a time service, but there could be in a SDA.

The IDA helped teach client-server architecture but did not teach peer-to-peer. A

SDA could be one architecture or the other, giving students experience designing both. DALP can also be both, giving the students the experience in choosing which kind of architecture would be best.

The final row of Table 6.7 shows 50 for the IDA, 62 for DALP, and 44 for SDA. This is the final result, showing that DALP creates a controlled, flexible assignments that teach necessary skills, knowledge areas, and abilities while minimizing frustration.

Factor	Weight	IDA	SDA	DALP
Characteristics of distributed systems	1	4	2	4
Opportunities and challenges	1	2	2	3
Core design issues	1	1	4	3
Common system models	1	1	4	2
Networking and internetworking	2	2	3	4
Protocols and protocol families	1	1	1	1
TCP/IP protocol suite	2	4	2	4
Interprocess communication	2	3	2	4
Security issues	2	3	2	3
Partial failure	2	4	3	5
Concurrency controls	1	4	2	4
Distributed objects	1	4	2	3
Distributed file systems	1	1	2	1
Time services	1	4	3	4
Client-server and peer-to-peer	1	2	2	3
Total	20	56	48[1]	68
Total Average		2.8	2.4	3.4

Table 6.7: Skills and knowledge areas factors weighted decision matrix

6.3.3 Adaptability

Table 6.8 shows how all three assignments line up concerning adaptability. The IDA can add and remove new features. While it is not the easiest thing to do so, it can be done. For a SDA, it is straightforward to remove an element. However, adding features can be difficult. It is hard to force a concept into an area where it does not fit. If the concept is forced into the project, the student's reaction will not be as positive as it could be.

[1] SDA's are hard to evaluate because each one will be different. It gives very little control to the teacher as to what the students are learning.

Factor	Weight	IDA	SDA	DALP
Add new features	1	3	2	4
Remove features	1	3	5	4
Total	2	6	7	8
Total Average		3	3.5	4

Table 6.8: Adaptability factors weighted decision matrix

DALP is easy to add and remove features. The most significant redesign is if a new drone is used with different features. The drone simulator will need to be replaced by someone. The rest of the system remains the same. If the new drone uses new technology for communication, UDP communication could be enforced somewhere else if it is still necessary. If a feature is removed, it can be easily replaced by a new feature.

6.3.4 Frustration

Factor	Weight	IDA	SDA	DALP
Reliable API	1	5	2	2
Reliability of Hardware/Software	1	5	2	3
Minimizes Extraneous Work	1	4	3	4
Total	3	14	7	9
Total Average		4.7	2.3	3

Table 6.9: Frustration factors weighted decision matrix

The frustration of students can have a devastating effect on a class. If the class gets frustrated trying to build the project, the project will not be as effective. Table 6.9 shows how well all three assignments try to minimize frustration levels.

The IDA has a predefined API that has been tried and tested. The API works and is reliable, leading to less frustration for the students and the teacher. The SDA may not have a straightforward API. The students can choose from a wide variety of software that has very different APIs. The DALP API has been unit tested and used in a sample implementation but has not been extensively tested. The API to the drone can be frustrating because the API is unreliable.

The software used in the IDA is reliable, it has been tested, and it works. A SDA is not guaranteed to have that. DALP's hardware has not been extensively tested. One implementation tested the system, but it may not be as reliable as the IDA. The reliability of the drone could get frustrating since it is not always reliable.

One of the biggest complaints in classes is the quantity of extraneous work. Extraneous work is work that is not related to the subject matter, but it is necessary to do. The more extraneous work given to the students, the more frustrated the students get. The IDA had little extraneous work. A SDA could have a lot of extraneous work. DALP contains some extraneous work. In the DALP project, there must be a mechanism to give targets to each drone. This mechanism can be an elaborate graph or a simple queue.

6.3.5 Results Summary

Factor	Weight	IDA	SDA	DALP
Flexibility	1.5	1.5	4.5	3.5
Skills and Knowledge areas	2	2.8	2.4	3.4
Adaptability	1	3	3.5	4
Frustration Minimization	1.5	4.7	2.3	3
Total	6	17.9	18.5	20.6
Total Average		3	3.1	3.4

Table 6.10: Main factors together in one weighted decision matrix

Table 6.10 goes over the final result of all four decision matrices combined. It is easy to tell which assignment will have the best effect using the decision matrix. If a teacher has different priorities than those expressed in this thesis, the weights can be modified, and a new result will be given specifically to that teacher's preferences.

CHAPTER 7

CONCLUSION

This thesis presents a new platform for teaching distributed systems called DALP. DALP creates a learning environment for students to practice designing, building, and testing distributed systems. DALP gives the students flexibility in designing and creating their own DA while giving instructors control over what the students are learning.

One use of DALP provides a semester-long assignment that consists of milestones. Each milestone allows the instructor to see how well the students understand a course objective by using a set of pretests. Milestones can also create deadlines to keep the students from procrastinating the entire semester before starting their projects.

The first set of pretests verifies that students can serialize and deserialize all of the messages used in the interprocess communication. This pretest will help the students find issues in their messaging code, as well as demonstrate to the instructor their messaging works correctly. The second set of pretests tests the reliability of their communication. Dropped messages or misconstructed messages may need a message to be resent. The last pretest verifies interprocess communication can be encrypted successfully.

By creating an engaging DA for the students, DALP enables students to focus on learning core concepts and developing meaningful skills through hands-on experience, without too many of the unnecessary distractions often present in non-trivial DA's. DALP advances the learning environment for teaching DA design by providing a platform that offers flexibility for the instructor in assignment design and control over learning objectives. It also minimizes incidental frustrations that a student may have working with third-party hardware and software.

This research presents a way to compare different assignments in a project-based pedagogy using decision matrices. Each assignment is broken down and evaluated. This thesis gives a theoretical argument that DALP advances the project-based pedagogy of

teaching distributed systems.

DALP can create a learning environment anywhere to teach DA design, not just in CS5200 at Utah State University. Any university can use the Tello drone and DALP in a distributed systems introductory course. If DALP does not fully meet a class's learning objectives, DALP can be easily modified to include the features necessary to fulfill those objectives.

DALP can also be used in the industry to teach distributed programming. While the most effective use of DALP will be creating the entire system, engineers in the industry may not have time to create a complete DA. The pretests in DALP provide an easy way to teach and test the understanding of serialization, deserialization, reliable communication, and encryption.

Unit tests have been created to verify the correct functionality of DALP, and the pretests validate the ability to learn the skills, knowledge areas, and abilities. This thesis provides a sample implementation of a system to evaluate DALP.

There is a lot of future work that can build upon this research. An empirical study of DALP would be beneficial to verify that DALP meets all of its intended goals. An empirical study will also test DALP on a bigger scale and provide more assurance that DALP is reliable. While this paper gives theoretical arguments about how DALP will improve the teaching of distributed systems, it is essential to test DALP with a real class.

DALP is not all-encompassing, and further researchers can add more features to DALP. These features will help teach some of the concepts that are not currently covered by DALP. New researchers might find that other communication protocols are needed. Instructors can also add different communication protocols to DALP. There are many ways that DALP can be changed to meet the needs of anyone who wants to learn about DA design.

REFERENCES

[1] (2020, Feb.) Knowledge, skills, and abilities. Wikipedia. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Knowledge,_Skills,_and_Abilities&oldid=940994951

[2] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems Concepts and Design*, 4th ed. Addison Wesley, 2005.

[3] (2020, Feb.) Knowledge, skills, and abilities. Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/OSI_model

[4] e. a. Garcia, João, "Parallel operating systems." in *Handbook on Parallel and Distributed Processing*, 2000, pp. 228–62.

[5] R. S. Jamgekar and G. S. Joshi, "File encryption and decryption using secure rsa," *International Journal of Emerging Science and Engineering*, vol. 11, p. 11–14, 2013.

[6] P. A. Bernstein and N. Goodman, "Concurrency control in distributed database systems," *ACM Computing Surveys*, vol. 13, p. 185–221, 1981.

[7] C. Burger and K. Rothermel, "A framework to support teaching in distributed systems," *J. Educ. Resour. Comput.*, vol. 1, pp. 114–119, 2001.

[8] T. Schümmer, S. Lukosch, and J. Haake, "Teaching distributed software development with the project method," in *Conference on Computer Support for Collaborative Learning*, 2005.

[9] C. Carnahan, K. Crowley, L. Hummel, and L. Sheehy, "New perspectives on education: Drones in the classroom," in *Society for Information Technology & Teacher Education International Conference 2016*, 2016, pp. 1920–1924.

[10] F. Sattar, L. Tamatea, and M. Nawaz, "Droning the pedagogy: Future prospect of teaching and learning," *International Journal of Educational and Pedagogical Sciences*, vol. 11, pp. 81–83, 2017.

[11] C. Chen, Y. Huang, C. Chang, and Y. Liu, "Exploring the learning effectiveness of 'the steam education of flying and assembly of drone," *Seventh International Conference of Educational Innovation through Technology*, pp. 63–67, 2018.

[12] O. Yakimenko, "Using multirotor drones in engineering of systems curricula," *Workshop on Research, Education and Development of Unmanned Aerial Systems*, pp. 114–119, 2017.

[13] M. Overmars, "Teaching computer science through game design," *Computer*, vol. 37, pp. 81–83, 2004.

[14] e. a. Malicky, David, "A design methodology for choosing an optimal pedagogy: The pedagogy decision matrix." *International Journal of Engineering Education*, vol. 23, 2007.

[15] (2020, Feb.) Leet. Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Leet

APPENDICES

APPENDIX A

Referee API

DALF

API for DALF

Base

Base

All messages must be wrapped in a BaseMsg, Must be prefaced with 6 bytes. The first byte is a sync byte 0x01 Next 5 bytes are the size of the message

```
BaseMsg
```

- Example-json: [#examples-Base-BaseMsg-1_0_0-0]
- Example-xml: [#examples-Base-BaseMsg-1_0_0-1]
- Example-proto: [#examples-Base-BaseMsg-1_0_0-2]

```
0x0100171{
  "msgId": "1Referee",
  "type": "HitTargetMsg",
  "msg": {
    "success": true,
    "complete": true,
    "error": "",
    "newTargets": [{
      "x": 100,
      "y": 100
    }],
    "badTargets": [{
      "x": 150,
      "y": 150
    }],
  }
}
```

```
0x01000CB<msgId>1Referee</msgId>
<type>HitTargetMsg</type>
<msg>
  <success>true</success>
  <complete>true</complete>
  </error>
  <newTargets>
```

```
    <x>100</x>
    <y>100</y>
  <newTargets>
  <badTargets>
    <x>150</x>
    <y>150</y>
  <badTargets>
</msg>
```

```
string msgId = 1;
string type = 2;
bytes msg = 3;
```

Success 200

Field	Type	Description
`msgId`	String	Message Id.
`type`	String	Message type of msg
`msg`	String	Message to be deserialized

Finish

Finish

Sent to Referee when the ground station gives up

```
FinishMsg
```

- Example-json: [#examples-Finish-FinishMsg-1_0_0-0]
- Example-xml: [#examples-Finish-FinishMsg-1_0_0-1]
- Example-proto: [#examples-Finish-FinishMsg-1_0_0-2]

```
{
   "gameId": 3
```

```
}
```

```
<gameId>3</gameId>
```

```
int32 id = 1;
```

Success 200

Field	Type	Description
gameId	Number	Game Id.

Finish

Response message to FinishMsg

```
ReadyRsp
```

- Example-json: [#examples-Finish-ReadyRsp-1_0_0-0]
- Example-xml: [#examples-Finish-ReadyRsp-1_0_0-1]
- Example-proto: [#examples-Finish-ReadyRsp-1_0_0-2]

```
{
  "Targets": [{
     "x": 50,
     "y": 50
   }]
}
```

```
<targets>
  <x>50</x>
  <y>50</y>
<targets>
```

```
repeated targets = 1;
```

Success 200

Field	Type	Description

Field	Type	Description
`id`	Number	Game Id.
`targets`	Targets	list of targets that are left.

Finish

Response message to FinishMsg

```
ReadyRsp
```

- Example-json: [#examples-Finish-ReadyRsp-1_0_0-0]
- Example-xml: [#examples-Finish-ReadyRsp-1_0_0-1]
- Example-proto: [#examples-Finish-ReadyRsp-1_0_0-2]

```
{
   "gameId": 3
}
```

```
<gameId>3</gameId>
```

```
int32 id = 1;
```

Success 200

Field	Type	Description
`id`	Number	Game Id.

HitTarget

HitTarget

Sent to Referee when a target has been hit

```
HitTargetMsg
```

- Example-json: [#examples-HitTarget-HitTargetMsg-1_0_0-0]
- Example-xml: [#examples-HitTarget-HitTargetMsg-1_0_0-1]
- Example-proto: [#examples-HitTarget-HitTargetMsg-1_0_0-2]

```
{
   "gameId": 3,
   "id": 4,
   "x": 50,
   "y": 50
}
```

```
<gameId>3</gameId>
<id>4</id>
<x>50</x>
<y>50</y>
```

```
int32 gameId = 1;
uint32 id = 2;
int32 x = 3;
int32 y = 4;
```

Success 200

Field	Type	Description
gameId	Number	Game Id.
id	Number	Target ID
x	Number	X Coordinate of target
y	Number	Y Coordinate of target

HitTarget

Response message to HitTargetMsg

```
HitTargetRsp
```

- Example-json: [#examples-HitTarget-HitTargetRsp-1_0_0-0]
- Example-xml: [#examples-HitTarget-HitTargetRsp-1_0_0-1]
- Example-proto: [#examples-HitTarget-HitTargetRsp-1_0_0-2]

```
{
  "success": true,
  "complete": true,
  "error": "",
  "newTargets": [{
     "x": 100,
     "y": 100
   }],
 "badTargets": [{
     "x": 150,
     "y": 150
   }],
}
```

```
<success>true</success>
<complete>true</complete>
</error>
<newTargets>
  <x>100</x>
  <y>100</y>
<newTargets>
<badTargets>
  <x>150</x>
  <y>150</y>
<badTargets>
```

```
bool success = 1;
bool complete = 2;
string error = 3;
repeated Target newTargets = 4;
repeated Target badTargets = 5;
```

Success 200

Field	Type	Description

Field	Type	Description
`success`	Bool	Whether the target was hit
`complete`	Bool	Whether all the targets have been hit
`error`	String	Error message
`newTargets`	Target[]	List of new targets to hit
`badTargets`	Target[]	List of targets to not hit anymore

Ping

Ping

Sent to from referee to maintain connection

```
PingMsg
```

- Example-json: [#examples-Ping-PingMsg-1_0_0-0]
- Example-xml: [#examples-Ping-PingMsg-1_0_0-1]
- Example-proto: [#examples-Ping-PingMsg-1_0_0-2]

```
<empty>
```

```
<empty>
```

```
<empty>
```

Ready

Ready

Sent to Referee when ready to start game

```
ReadyMsg
```

- Example-json: [#examples-Ready-ReadyMsg-1_0_0-0]
- Example-xml: [#examples-Ready-ReadyMsg-1_0_0-1]
- Example-proto: [#examples-Ready-ReadyMsg-1_0_0-2]

```
<empty>
```

```
<empty>
```

```
<empty>
```

Ready

Response message to ReadyMsg

```
ReadyRsp
```

- Example-json: [#examples-Ready-ReadyRsp-1_0_0-0]
- Example-xml: [#examples-Ready-ReadyRsp-1_0_0-1]
- Example-proto: [#examples-Ready-ReadyRsp-1_0_0-2]

```
{
  "gameid": 5,
  "Targets": [{
     "x": 50,
     "y": 50
   }]
}
```

```
<gameid>5</gameid>
<targets>
  <x>50</x>
  <y>50</y>
<targets>
```

```
int32 gameid = 1;
repeated targets = 2;
```

Success 200

Field	Type	Description
gameid	Number	Game Id.
targets	Targets	list of targets that are available from the start.

Ready

Response message to ReadyRsp

```
ReadyRspRsp
```

- Example-json: [#examples-Ready-ReadyRspRsp-1_0_0-0]
- Example-xml: [#examples-Ready-ReadyRspRsp-1_0_0-1]
- Example-proto: [#examples-Ready-ReadyRspRsp-1_0_0-2]

```
{
  "gameid": 5
}
```

```
<gameid>5</gameid>
```

```
int32 gameid = 1;
```

Success 200

Field	Type	Description

Field	Type	Description
gameid	Number	Game Id.
targets	Targets	list of targets that are available from the start.

APPENDIX B

Configuration File

```
The following is an example of a config file that can be passed into the
Referee and DroneSimulator

It must be a json object

{
  "ThreadCount": 1,
  "Speed": 2,
  "BatteryDecaySpeed": 1,
  "Targets": [{
    "x": 150,
    "y": 150,
    "id": 5,
    "Dependents": [3],
    "FalseAfter": 2
  ]},
  "SkipLog": false,
  "Pretest": 0,
  "PrintToConsole": true,
  "Format": "proto"
}

Configuration Options:

Options for both Referee and DroneSimulator

Targets:
  Array of targets for the Referee and DroneSimulator to know about.
  Constructed of an x, y, id, Dependents, and FalseAfter.
    x is the X Coordinate
    y is the Y Coordinate
    id is the id of the target
    Dependents is what this target is dependent on in order to be ready.
      If no dependent, then the target is available at the start of the
game.
      This only effects the Referee. The DroneSimulator ignores this
field.
    FalseAfter is after what target does this target become unnecessary.
      If not set, the target will always be necessary.
      This only effects the Referee. The DroneSimulator ignores this
field.

SkipLog
  If the log file should be produced or not. Defaults to false
  Possible values are false and true

Pretest
  Which pretest to run if any.
  Possible values for DroneSimulator are 0, 1, 2
  Possible values for Referee are 0, 1, 2, 3
  If not set, defaults to 0.
  If set to 0, no pretest is ran. Normal execution occurs
  If set to 1, pretest 1 is ran
  If set to 2, pretest 2 is ran
```

```
  If set to 3, pretest 3 is ran

PrintToConsole
  If the logs should be printed to standard out. Defaults to false
  Possible values are false and true

Referee only options:

ThreadCount:
  The Referee runs using a thread pool. Any tasks given to it are given to
the thread pool.
  The default value is 1. If a value less than one is given, 1 is used

Format:
  Format used for serialization and deserialization.
  Possible values are proto, json, and xml.
  Defaults to proto

DroneSimulator only options:

Speed:
  The DroneSimulator will wait the correct amount of time before
responding to a message.
  Changing the speed changes how fast the DroneSimulator flies. This can
be useful for unit testing.
  Just because the system works with this set, does not mean the system
will work without it set.
  Use this field with caution
  Default value is 1

BatteryDecaySpeed:
  The DroneSimulator will decrement the battery every once and a while.
That time difference can be
    changed my setting this field.
  If the field is 2, the battery will decrement every two seconds
  Default value is 1
  Any value less than 1 will be 1
```

APPENDIX C

Command Line Arguments

Referee Command Line Arguments:

Command	Description	Required
-h,--help	Provides help text	false
-c,--config	Location of the configuration file	true

Drone Simulator Command Line Arguments

Command	Description	Required
-h,--help	Provides help text	false
-c,--config	Location of the configuration file	true
-p,--port	Port to receive command messages status messages will be the next port	true
-y,--y	Starting y position	false - defaults to 0