UDC 004.042

A. Akhaladze, O. Lisovychenko

INCREASING THE RELIABILITY OF THE COMMUNICATION CHANNEL BETWEEN AGENTS IN A MULTI-AGENT ENVIRONMENT BASED ON AZURE IOT SERVICES

Abstract: An architectural approach is considered and proposed for increasing the reliability of the communication channel, which allows guaranteeing the delivery of messages in the middle of the system from each member of the system, provided that there is an active communication channel. At the same time, the used policy of restoring the communication channel on the side of each agent implements the "circuit breaker" pattern, which allows for minimizing the time of each agent offline and avoids excessive analysis of irrelevant data of agents operating in real-time.

Keywords: multi-agent environment, IoT, service, agent, circuit breaker, retry policy

Introduction

IoT architecture[1], used to build a drone swarm control system, provides a number of advantages, such as an unlimited number of swarm members, ease of adding a new drone (provided there is an implemented proxy for communication[1]), ease of maintenance and deployment of the architecture.

The problem that requires a solution is the instability of the communication channel with the agents that the drones and the operator's console act as in the system. Implementing communication channels between agents in a multi-agent environment requires the use of a complex solution that guarantees the fulfillment of requirements for speed, security, reliability, resistance to failures, logging, and monitoring requirements. The numerical values of the critical indicators of these requirements depend on the configuration environment and conditions for the applied task. The environment configuration includes

1. degree of certainty of agents (formally described configurations of each agent and their actions) and relations between agents. In this system, a formal model of behavior is implemented (commands for execution and the format of drone data are described when registering the device in the swarm)

2. the topology of the communication architecture between agents: at this stage of the research, the "pyramid" topology is implemented

3. mechanisms for ensuring infrastructure requirements

a. security: agents' role access mechanisms, authorization, authentication, and encrypted data transmission channels

b. reliability: guarantee of data delivery between agents

c. failure of stability: mechanisms to guarantee the restoration of the communication channel between agents (retry, circuit breaker, etc.)

d. logging and monitoring: real-time monitoring of agent status, failure detection, and analysis mechanism

Separate requirements for the speed of transport channels between agents and the feedback system is made taking into account the requirements of a specific applied problem being solved. In this work, we will propose a set of measures and approaches to increase the resistance of the communication channel to failures and interruptions for the communication system between agents in a multi-agent environment based on Azure IoT services[1] to solve the problem of drone swarm management.

Problem statement

It is necessary to consider and propose architectural approaches and software patterns for the operation of the drone control system (agents) in conditions of interruptions and loss of communication with the decision-making center (hub). The main requirement for the system is the implementation of the agent-side communication recovery mechanism in case of loss (for sending telemetry and video data) and the implementation of the data resending mechanism that implements the “circuit breaker” pattern.

Solving the tasks

To fully formalize requirements, it is necessary to define critical indicators (ASR - architecture significant requirements) and formalize the representation of agents.

Based on the requirements of the given task, we will include the following requirements in the ASR:

1. Guaranteed delivery of commands to each agent in an active state
2. Monitoring the status of each agent in real-time
3. The mechanism for obtaining telemetry data of each agent
4. Feedback mechanism

According to the conditions of the task, we have three types of agents:

1. a control agent whose task is to initiate the execution of commands
2. an executive with 2 degrees of freedom ("car") fig. 4
3. executive with 6 degrees of freedom ("drone") fig. 5

We implement communication using Azure IoT Central (architectural diagram Fig. 1) and formalize the capabilities of each type of agent. According to the conditions of the given task, we have a controlling agent, the model of which is shown in fig. 2, which sends commands to which all other connected agents respond. Executive agents, in turn, send telemetry data for analysis of the correction of the control team.

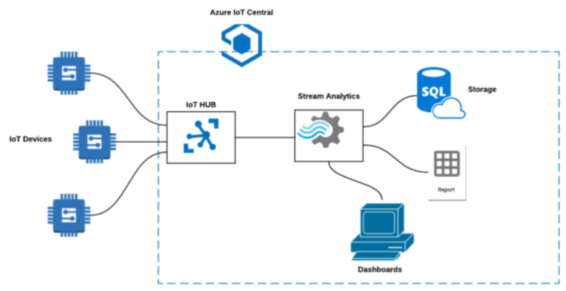

Figure 1. Connection diagram to Azure IoT central

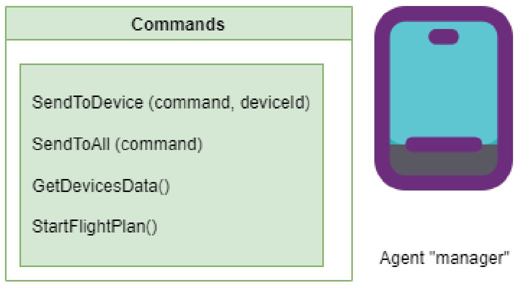

Figure 2. Formal presentation of the "manager" agent

We formalize the representation of executive agents. Based on the requirements of the applied problem, we have the following agents.

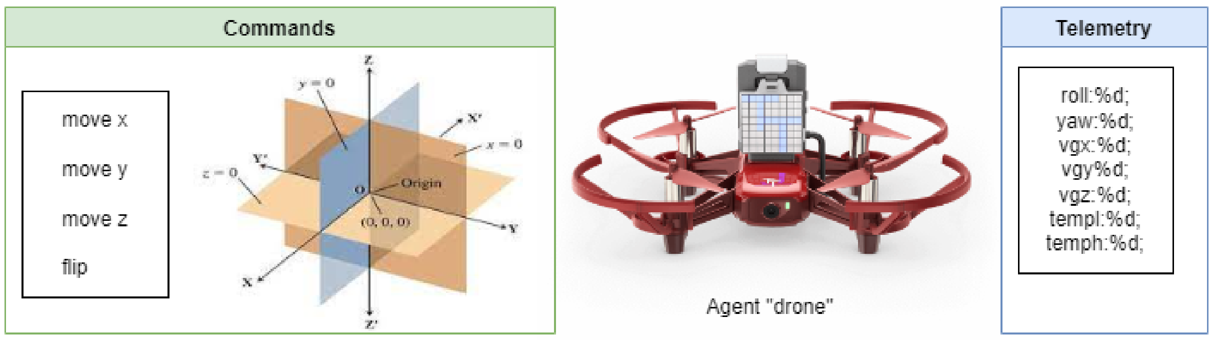

Figure 3. Formalized representation of the "drone" agent

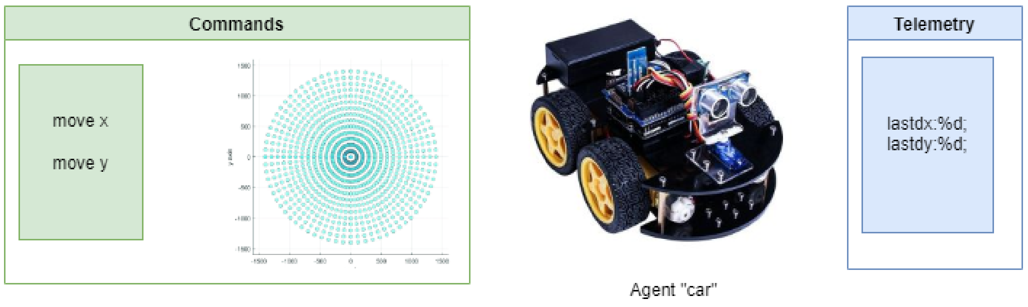

Figure 4. Formalized representation of the “car” agent

By combining agents in one infrastructure using Azure IoT Central, we have the architectural scheme of connecting the agents of fig. 5.

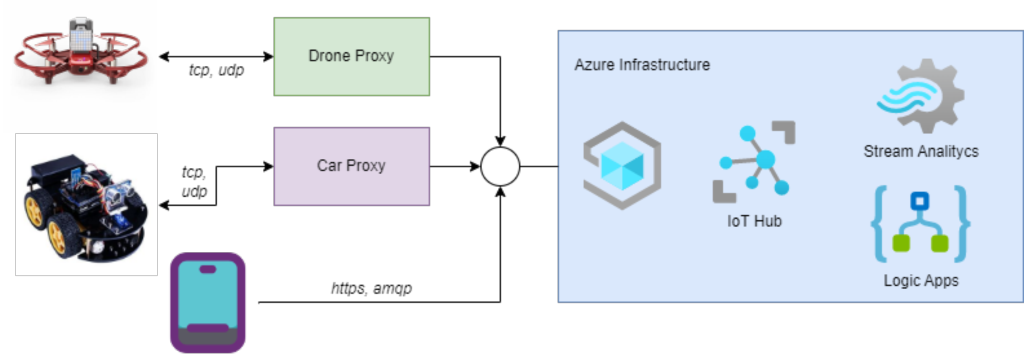

Figure 5. The architectural scheme of connecting agents by type to Azure

Azure IoT Central[1] provides extensive integration opportunities with ready-made solutions that are already used to solve applied problems, for example, the analysis of video streams. At the same time, the service architecture allows you to build flexible logic based on serverless functions, limited only by speed requirements, implements a diagnostic and monitoring system, and also ensures the fulfillment of security, availability, and reliability conditions, including inter-regional DR (disaster recovery). A typical modular diagram of a solution based on Azure IoT Central is shown in Fig. 6.

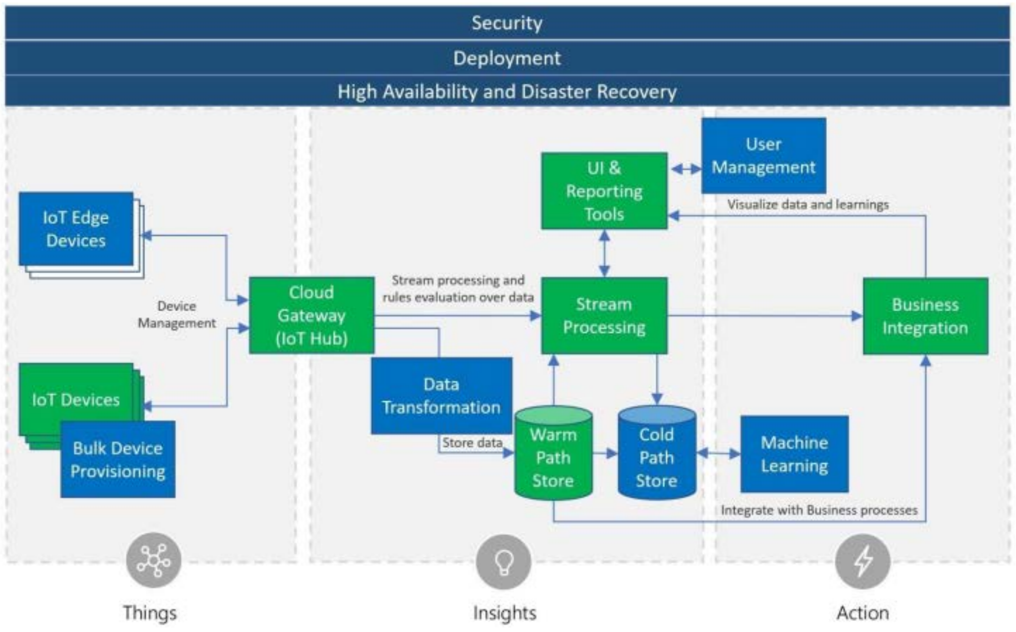

Figure 6. Modular scheme of the solution is built on the basis of Azure IoT Central

The service scheme component, shown in Fig. 7, allows you to guarantee the delivery of each message at least once from each sender, under the conditions of message registration in the system, provides a simplified scheme for registering new agents, and allows you to use existing integration solutions with other services, which allows, on based on received messages from agents, to build a flexible logic of swarm behavior. However, the key condition for guaranteed message delivery is its registration in the system, the solution of which is the responsibility of the protocol level, that is, the use of architectural approaches is only a partial solution. Next, we will consider the protocol level.

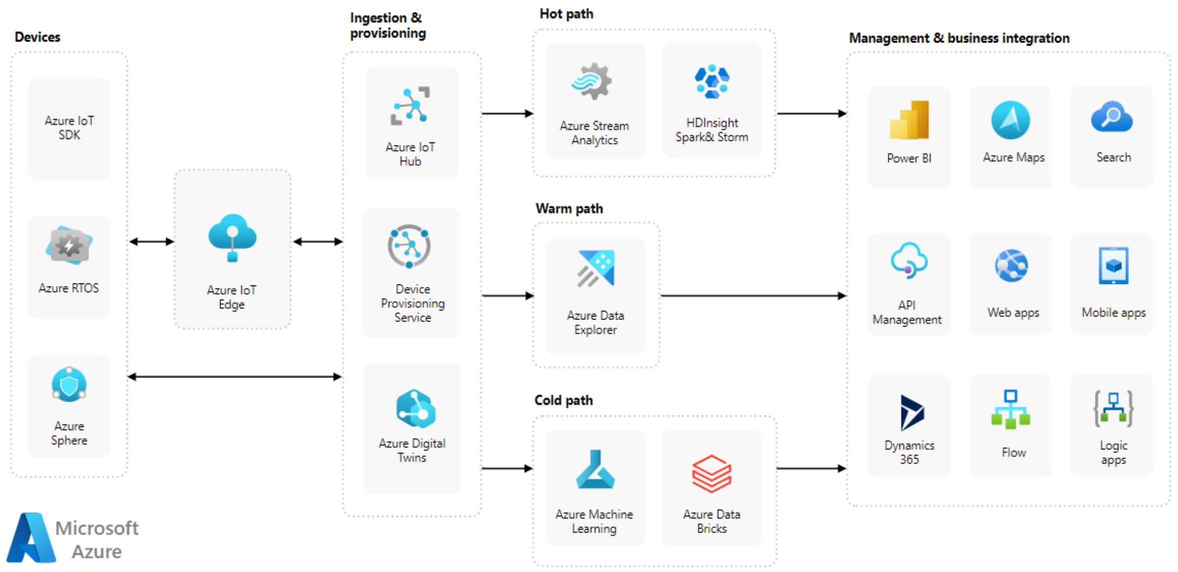

Figure 7. Component diagram of services that allows implementation

The component diagram shown in Fig. 7. is redundant for solving the given problem. The following services were used during the implementation of this system:

1. Azure IoT SKD[2] to connect the “drone” agent proxy application

2. Azure RTOS for testing the use of the "manager" agent as a proxy (IoT Edge Device) for executive agents

3. Azure Sphere is declared redundant

4. Azure IoT Hub to configure message routing

5. Device Provisioning Service for organizing access via the Internet

6. Azure Digital Twins for testing and deploying a proxy, opium, for connecting data mining

7. Azure Stream Analytics for streaming data from agents

8. Azure Logic Apps[4] for fast math operations and hot telemetry analysis

All other services add data analytics capabilities to the system but are redundant at this stage of the research.

Let's consider the methods of the protocol level of applications. It is impossible to avoid gaps in communication between agents, therefore, we minimize the time of an agent offline when executing a flight plan by a swarm, by implementing a mechanism for retrying to send all messages. Since the system operates in real-time, the loss and restoration of the communication channel will lead to the sending of data that is no longer relevant for data analysis (telemetry and video), so we will use the "circuit breaker" template as the default retry policy. This will reduce the load on the communication channel and avoid the analysis of irrelevant data. In fig. 8 shows the curve of the ratio of the number of retries to the delay time between them when receiving constant failures of the communication channel.

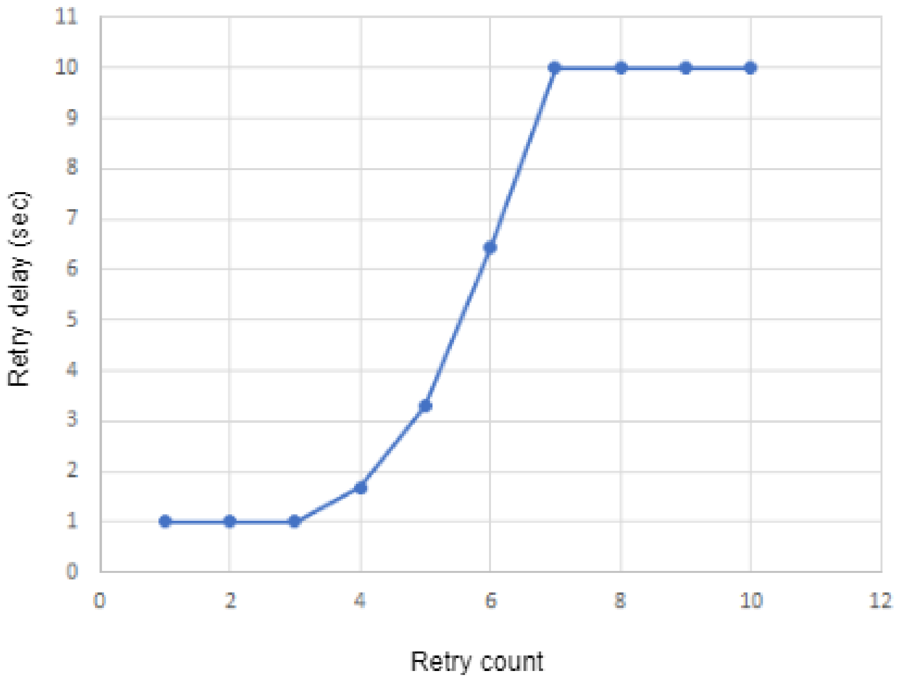

Figure 8. The curve of repeated attempts to send a message

That is, when the communication channel is lost, the first three attempts to send a message from the agent to the hub will be made without delay, and the delay between the 7th and 8th attempts is 10 seconds. When sending telemetry data and a video frame for analysis, real-time data loss is allowed in case of connection breaks, and this is not critical for the system, since the last received snapshot of the state will reflect the most up-to-date available data of the agent's state, and intermediate data obtained during the period of lost communication already are not relevant and have no value for the analysis, so they should be omitted for reduction to the system.

The data is received from the sensors and the camera and is ready to be sent, on the second unsuccessful attempt it is replaced by a new cast, i.e. when the communication channel is restored, which is a necessary condition for exiting the cycle of repeated sending attempts, the data of the last two seconds of the agent's work will be sent to the system.

In fig. 9 is a diagram of state transitions of the "circuit breaker" template, which demonstrates the general behavior of the retries mechanism. If the message is successfully sent, the status "closed" is set, which means that the message was sent successfully. At the first unsuccessful attempt, the state is set to "open" and the message is sent again. In case of unavailability of the hub, the state is changed to “half-open” and a new delay value is set before the next attempt. The sending mechanism is in this state until the first successful sending. At the same time, for each new attempt, up-to-date data from the drone is obtained, which is replaced by an out-of-date snapshot of the state, thus we reduce the load on the system by analyzing out-of-date data.

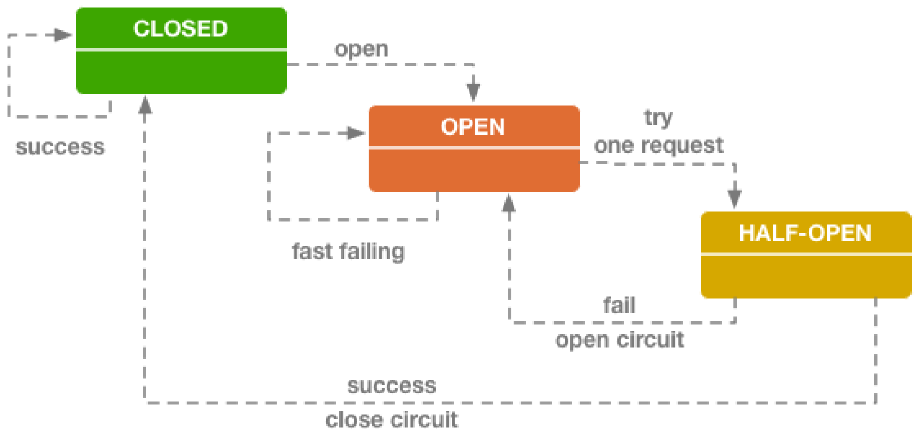

Figure 9. State diagram of the message-sending module that implements the “circuit breaker” template

Conclusions

The implemented swarm management system (where agents are drones) is vulnerable to the loss of communication between agents and leads to a redundant analysis of irrelevant data received after communication is restored since the system works in real-time.

The proposed architectural approach for building an agent management system in a multi-agent environment guarantees the delivery of a message from agents to the system at least once. At the same time, the "circuit breaker" software template used to send each message through the system minimizes the drone's offline time when executing commands and restoring the communication channel.

An additional benefit of using this template is the ability to avoid analysis of out-of-date data from a real-time agent.

The use of Azure software and services enables the configuration of transport channels at the protocol level (MQTT, AMQP, HTTPS, TCP), implements the security perimeter, and expands the analytical capabilities of the system due to integration with existing services (AI, FaceId API, recognition services, partial analytics solutions, etc.)

REFERENCES

1. A. Akhaladze Using IoT to synchronize drone flight paths 2021. № 39. C.20-26 URL: http://asac.kpi.ua/article/view/247381 https://doi.org/10.20535/1560-8956.39.2021.247381

2. Clint B. (2019)Will serverless computing be the future of cloud computing. Computer world, 2019-03-25(005).

3. Brandon B. (2017)Serverless Computing: Next Generation Cloud Infrastructure. Computer world, 2017-05-15(003).

4. Carter G. (2018) Serverless Architecture. Mechanical Industry Press. Beijing.

5. Zhi Yun. (2018)Read the advantages and disadvantages of serverless architecture, and use cases. http://www.sohu.com/a/234002113_100159565.

6. Lingming Xia. (2018) Deep understanding of serverless architecture. https://blog.csdn.net/xialingming/article/details/81369624.