# CONCURRENCY USING ACTIVE OBJECTS ARCHITECTURAL PATTERN

# ARCHITECTURAL PATTERN VS DESIGN PATTERNS

## ARCHITECTURAL PATTERNS[1]

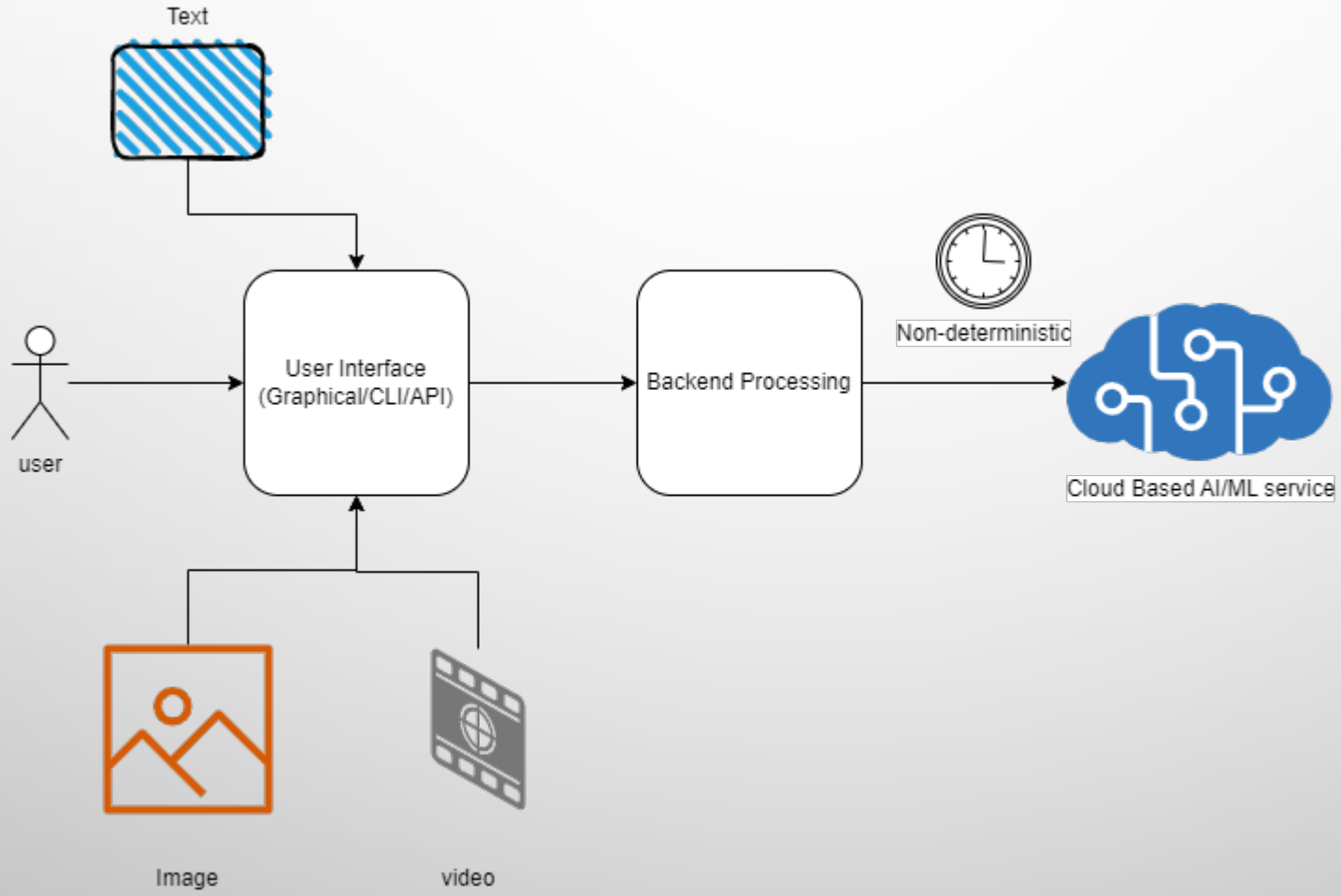
- AN ARCHITECTURAL PATTERN EXPRESSES A FU
SC
A
TH
AN
RE
- EG- MVC, LAYERS PATTERN, MICROKERNEL PATTERN

## DESIGN PATTERNS[1]

- A DESIGN PATTERN PROVIDES A SCHEME FOR REFINING THE SUBSYSTEMS OR COMPONENTS OF
GN
- EG- OBSERVER, COMMAND, FACTORY, SINGLETON

**Active Object is an Architectural Pattern**

# Let Us Do A System Design

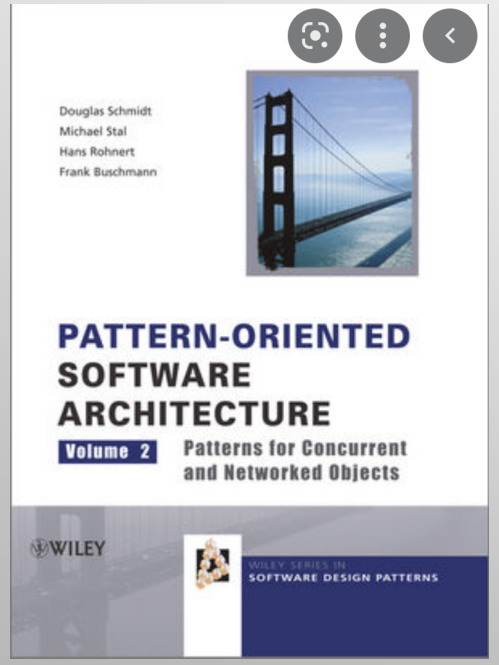

Text
user
User Interface
(Graphical/CLI/API)
Backend Processing
Non-deterministic
Cloud Based AI/ML service
Image
video

# DESIGN CHALLENGE

- UI FACING THREAD CANNOT BLOCK
- THE AI/ML SERVICE RESPONSE TIMES ARE UNPREDICTABLE
- BACKEND AND THE UI THREADS CAN HAVE MULTIPLE THREADS
- EASY TO PROGRAM AND UNDERSTAND THE DESIGN
- ADHERE TO SOLID PRINCIPLES

# ACTIVE OBJECT ARCHITECTURAL PATTERN

The Active Object design pattern decouples method execution from method invocation to enhance concurrency and simplify synchronized access to objects that reside in their own threads of control.

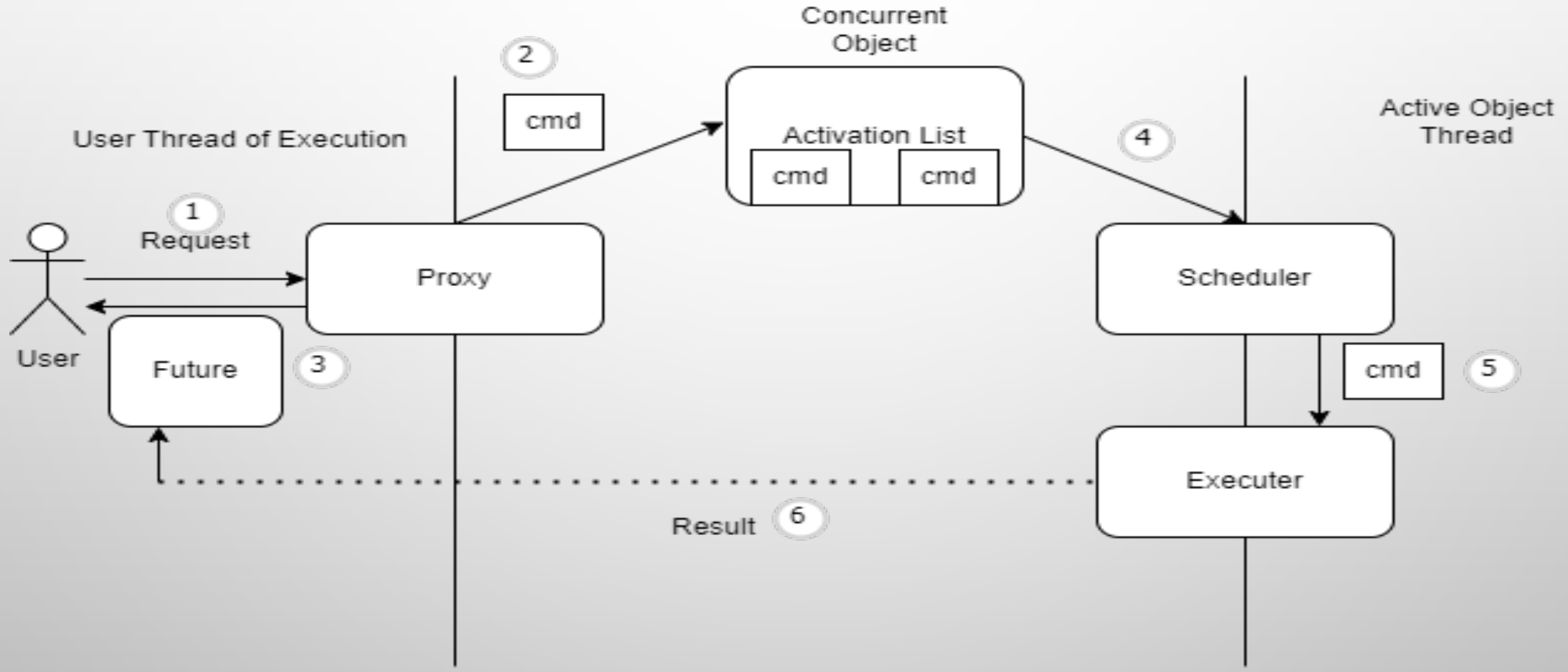


Method invocation should occur in the client's thread of control, whereas method execution should occur in a separate thread. Moreover, design the decoupling so the client thread appears to invoke an ordinary method.

# COMPONENTS INVOLVED IN ACTIVE OBJECT DESIGN

| Proxy | Command Request Interface | Concrete Command | Activation List | Scheduler | Executer | Future |
|---|---|---|---|---|---|---|
| • Provides Interface to users<br>• Create Command Request | • Defines methods, which commands should implement | • Implements the specific command | • Maintains Command requests which are pending for execution | • Examines activation list and schedules the commands ready for execution | • Implements and Runs the Command request Object. | • Holds the result of execution.<br>• Provides rendezvous point to the client |

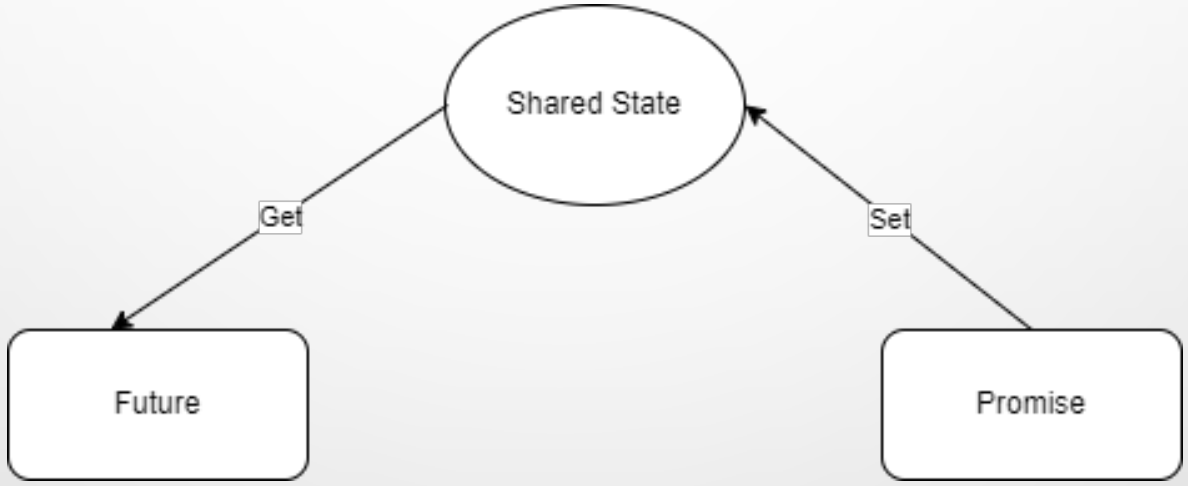
# ACTIVE OBJECT DYNAMICS

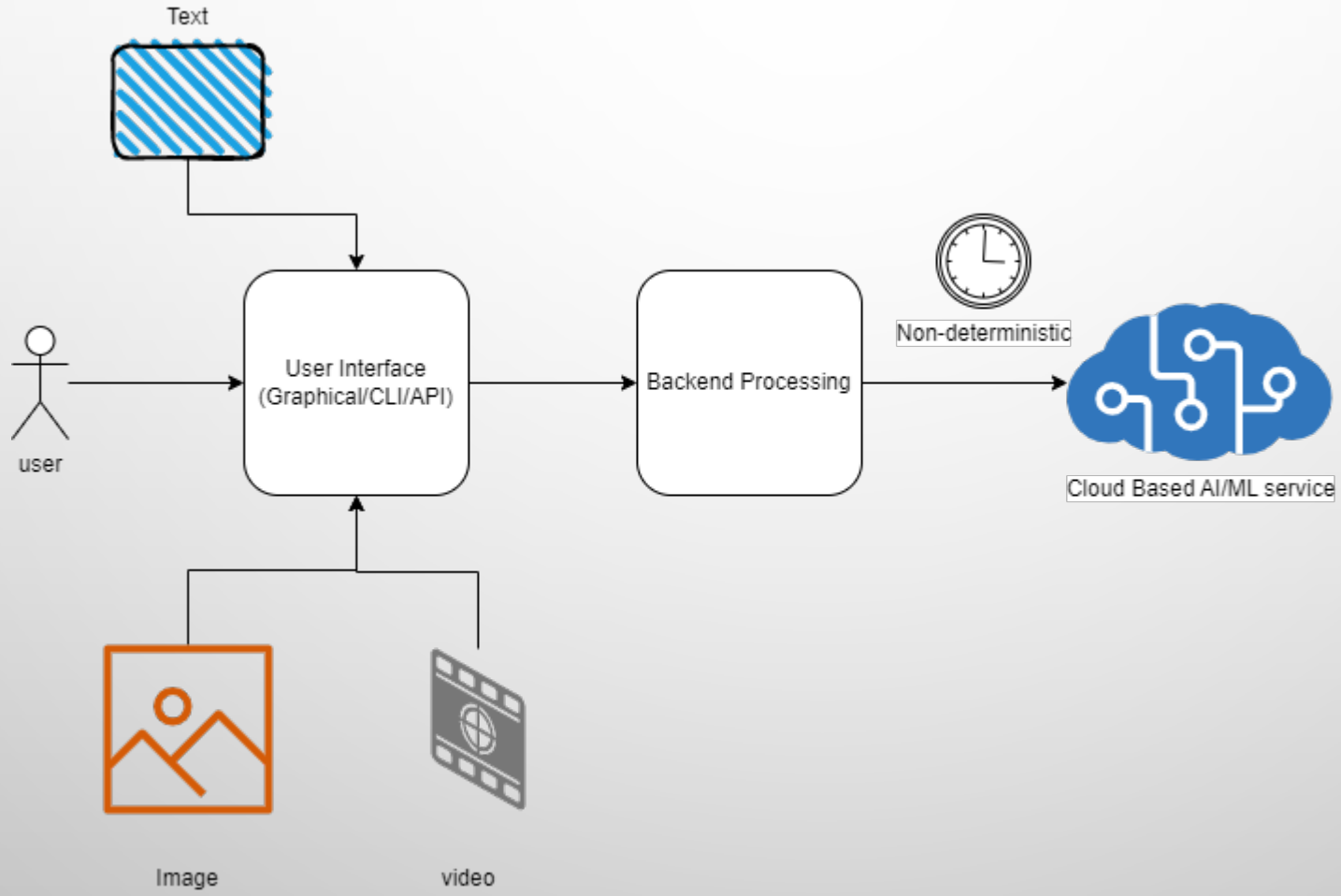


# OBSERVATIONS

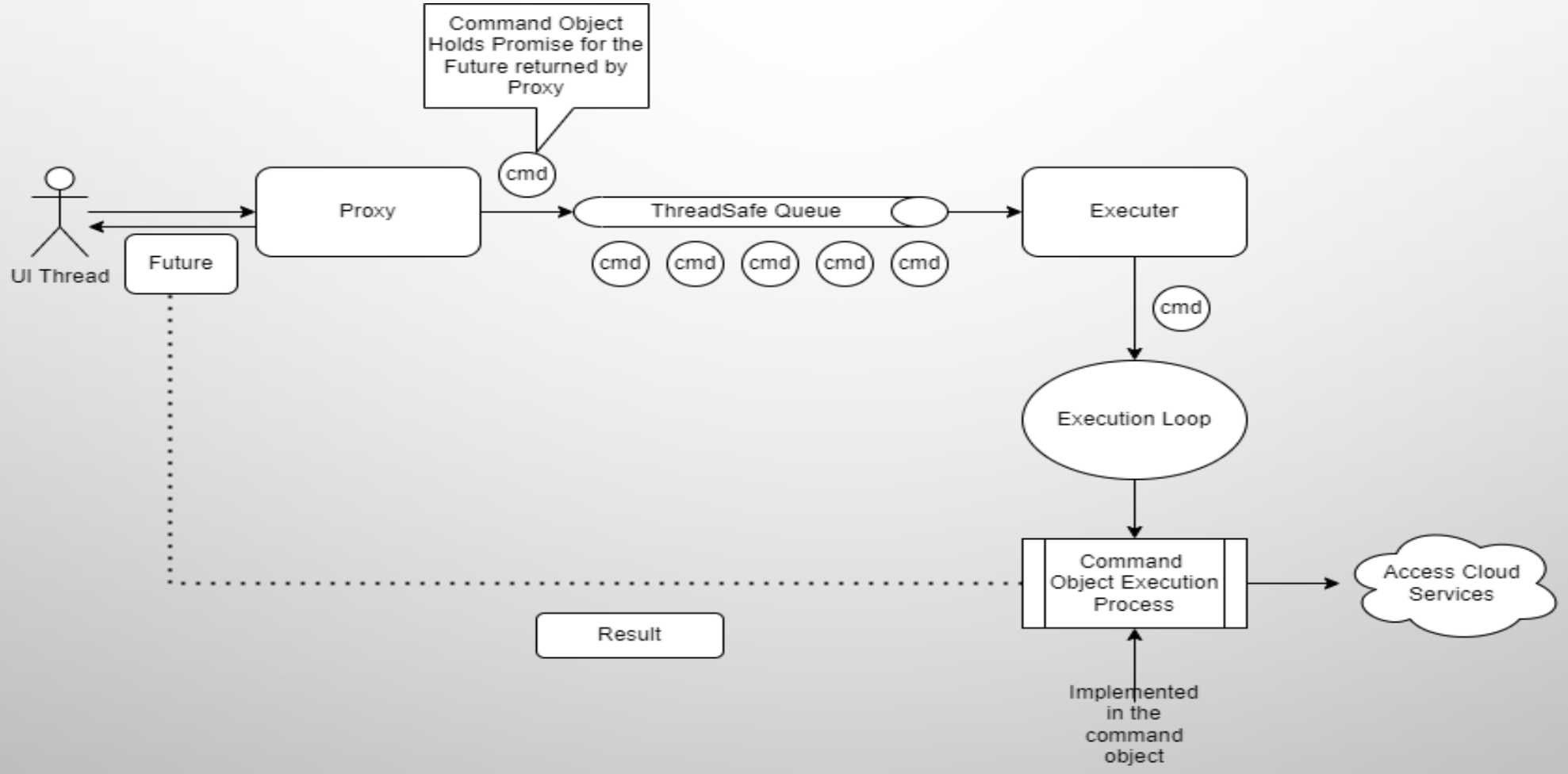
- ONLY ONE CONCURRENT OBJECT – ACTIVATION LIST
- CLEAR SEPARATION OF CONCERNS- SINGLE RESPONSIBILITY PRINCIPLE
- EASY TO UNDERSTAND AND MAINTAIN
- CAN BE EXTENDED AND CUSTOMIZED FOR DIFFERENT NEEDS.

# PROMISE AND FUTURE




Text
user
User Interface
(Graphical/CLI/API)
Backend Processing
Non-deterministic
Cloud Based AI/ML service
Image
video

# OUR SYSTEM DESIGN



```
struct CommandInterface {
    virtual std::future<Result> GetFuture()
= 0;
    virtual void Execute() = 0;
    virtual ~CommandInterface() = default;
};
```

```
 struct Result {
     std::string data;
     bool valid = false;
     std::string err_reason;
 };
```

```
struct CommandInterface {
    virtual std::future<Result> GetFuture()
= 0;
    virtual void Execute() = 0;
    virtual ~CommandInterface() = default;
};

class RecommendationCmd : public
CommandInterface {
public:
    explicit RecommendationCmd(std::string
inp) :
        input(std::move(inp)){}
    std::future<Result> GetFuture() override;
    void Execute() override;

private:
    // Some other methods and data to help
generate the results.
    std::promise<Result> async_result;
    std::string input;
};
```

```cpp
template <typename T>
class ConcurrentQueue {
public:
    void enqueue(T data) noexcept
    {
        std::lock_guard<std::mutex> guard(mtx);
        queue.push(std::move(data));
    }
    std::optional<T> dequeue() noexcept
    {
        std::lock_guard<std::mutex> guard(mtx);
        if (queue.empty()) {
            return std::nullopt;
        }
        auto data = std::move(queue.front());
        queue.pop();
        return data;
    }

private:
    std::queue<T> queue;
    std::mutex mtx;
};
```

```
using CommandQ =
ConcurrentQueue<std::unique_ptr<CommandInterface>>;
using AsyncResult = std::future<Result>;
using CommandExecutor =
Executor<std::unique_ptr<CommandInterface>>;

    class Proxy
    {
        public:
            Proxy();
            AsyncResult GetPrediction(std::string inp_data);
            AsyncResult GetRecommendation(std::string
inp_data);
            ~Proxy();

        private:
            CommandQ cmd_q;
            std::atomic<bool> running;
            CommandExecutor executor;
            std::thread executor_task;

    };
```

```
 Proxy::Proxy() :
        running(true),
        executor(cmd_q, running)
    {
       executor_task = std::thread(&CommandExecutor::Run,
&executor);
    }
   AsyncResult Proxy::GetPrediction(std::string inp_data)
    {
        auto prediction_cmd =
std::make_unique<PredictionCmd>(std::move(inp_data));
        AsyncResult result = prediction_cmd->GetFuture();
        cmd_q.enqueue(std::move(prediction_cmd));
        return result;
    }
   Proxy::~Proxy()
       {
           running = false;
           if (executor_task.joinable())
           {
               executor_task.join();
           }
       }
```

```
template<typename T>
class Executor
{
  void Run() {
    while(keep_running) {
      std::optional<T> cmd_opt = queue_ref.dequeue();
      if (!cmd_opt.has_value()) {
        using namespace std::chrono_literals;
        std::this_thread::sleep_for(5ms);
        continue;
      }
      auto cmd = std::move(cmd_opt.value());
      if constexpr (std::is_pointer<T>::value  || is_smart_ptr<T>::value) {
        cmd->Execute();
      } else {
        cmd.Execute();
      }
    }
  }
    private:
        ConcurrentQueue<T>& queue_ref;
        std::atomic<bool>& keep_running;
  };
```

```
int main()
{
    active_object::Proxy client_proxy;
    auto result_future = client_proxy.GetPrediction("Some
Input");
    auto result = result_future.get(); // This Blocks
    std::cout << "Result = " << result.data << '\n';
    return 0;
}
```

# SOME IMPROVEMENTS

- CAN HAVE A BETTER CONCURRENT QUEUE
- USE THREAD POOL FOR EXECUTION IF NEEDED.
- USE ACTIVATION LIST AND SCHEDULER IF NEEDED
- USE CRTP OR TYPE ERASURE FOR COMMAND INTERFACE

# EXTENDING THE DESIGN

- ADD NEW COMMANDS
- ADD API IN PROXY FOR THE NEW COMMANDS

# LINKS

1. HTTPS://WWW.OREILLY.COM/LIBRARY/VIEW/PATTERN-ORIENTED-SOFTWARE-ARCHITECTURE/9781118725177/
2. HTTPS://GITHUB.COM/SELVAKUMARJAWAHAR/ACTIVE_OBJECT

# THANK YOU