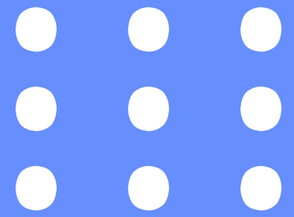
ROS2 C++ 프로그래밍

오로카 판교 온라인 스터디

오현준
ohj_980918@naver.com
010-7674-2599



목 차

Python 과 C++ 당신의 선택은?

01

ROS2의 대표적인 프로그래밍 언어는 파이썬과 C++입니다.
이 두 언어는 각각의 장점과 단점이 있습니다.

C++ 프로그래밍의 장점

속도

빠른 처리속도를 원하는 프로젝트에서 이점을 가집니다.

오류

컴파일링 과정에서 더 많은 오류를 발견할 수 있다. 이는 나중에 혹시모를 에러를 방지해준다.

활용성

로보틱스 산업에서 일하고 싶다면 선택이 아닌 필수입니다.

C++ 프로그래밍의 단점

개발속도

작은 사이즈의 프로토타입을 설계하는데도 많은 코드가 들어갑니다.

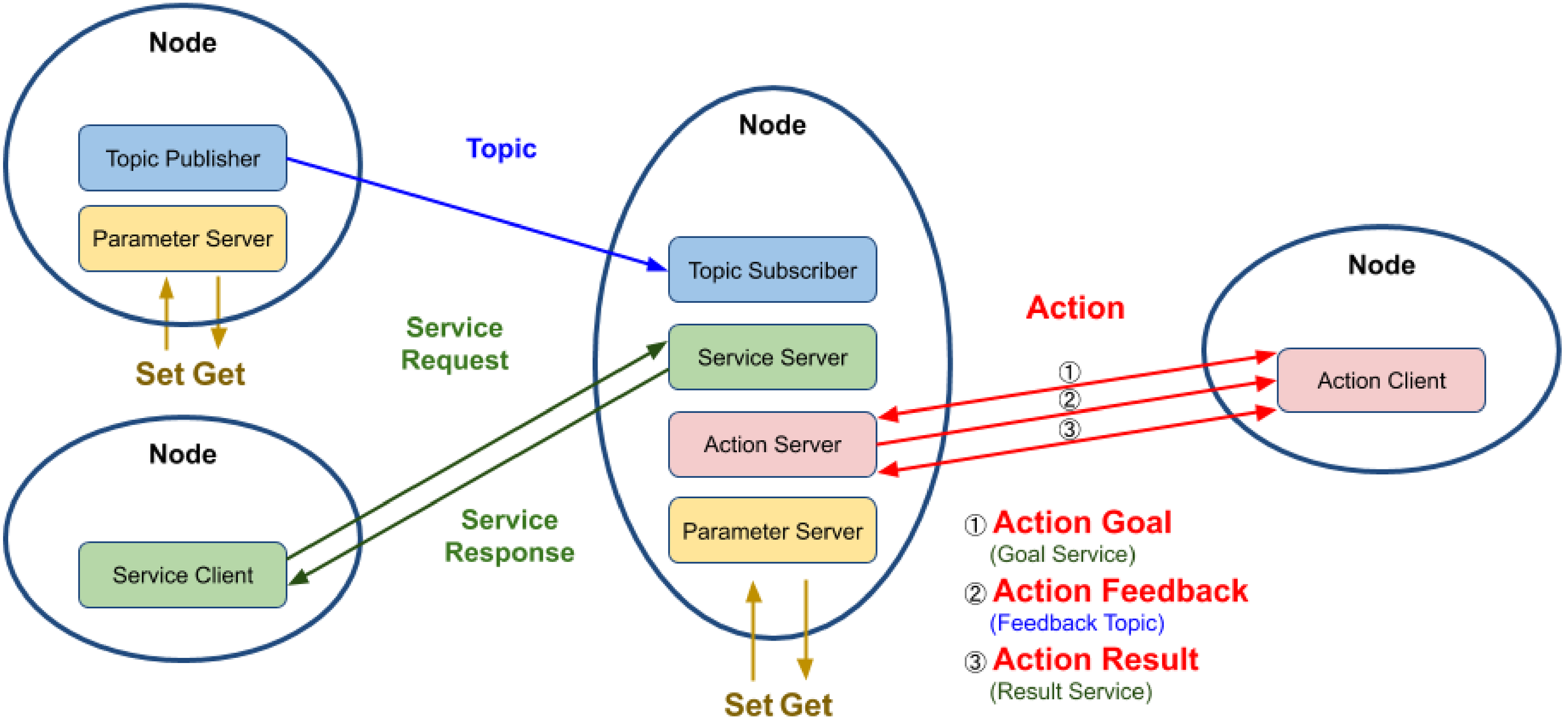
이해도

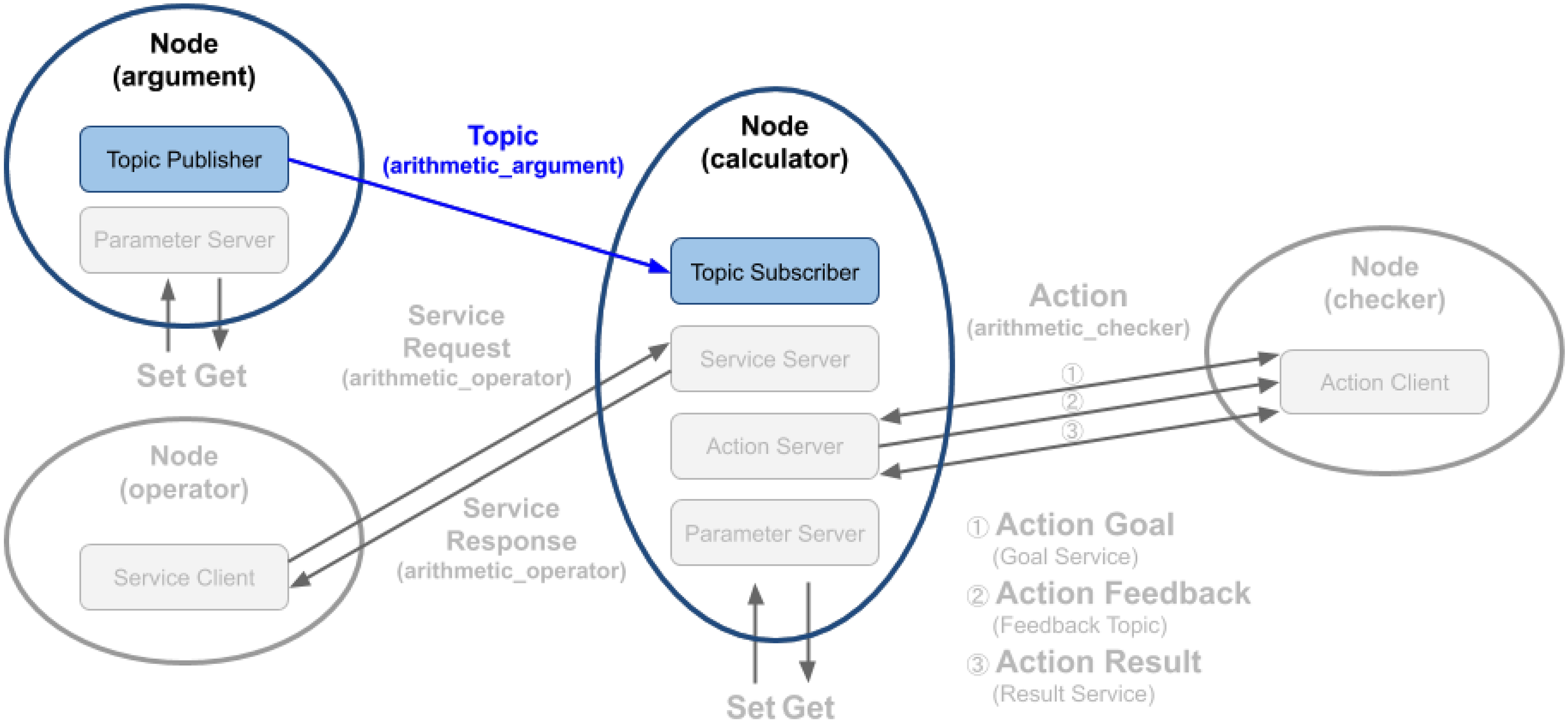
C++기반의 지식이 부족하다면, 소스코드가 어떻게 작동하는지 이해하는데 시간이 오래 걸립니다.

토픽 프로그래밍 C++

02

C++을 이용하여 토픽 프로그래밍 하기


Node
Topic Publisher
Parameter Server
Set Get
Topic
Node
Topic Subscriber
Service Server
Action Server
Parameter Server
Set Get
Service Request
Service Response
Node
Service Client
Action
①
②
③
Node
Action Client
① Action Goal
(Goal Service)
② Action Feedback
(Feedback Topic)
③ Action Result
(Result Service)


Node
(argument)
Topic Publisher
Parameter Server
Set Get
Topic
(arithmetic_argument)
Node
(calculator)
Topic Subscriber
Service Server
Action Server
Parameter Server
Set Get
Service
Request
(arithmetic_operator)
Service
Response
(arithmetic_operator)
Node
(operator)
Service Client
Action
(arithmetic_checker)
①
②
③
Node
(checker)
Action Client
① Action Goal
(Goal Service)
② Action Feedback
(Feedback Topic)
③ Action Result
(Result Service)

토픽 퍼블리셔

1) Node 설정
2) QoS 설정
3) create_publisher 설정 (+ timer 설정)
4) 퍼블리시 함수 작성

토픽 서브스크라이버

1) Node 설정
2) QoS 설정
3) create_subscription 설정
4) 서브스크라이브 함수 작성

C++ 토픽 프로그래밍

노드 설정

```
class Argument : public rclcpp::Node
{
public:
  using ArithmeticArgument = msg_srv_action_interface_example::msg::ArithmeticArgument;

  explicit Argument(const rclcpp::NodeOptions & node_options = rclcpp::NodeOptions());
  virtual ~Argument();

private:
  void publish_random_arithmetic_arguments();
  void update_parameter();

  float min_random_num_;
  float max_random_num_;

  rclcpp::Publisher<ArithmeticArgument>::SharedPtr arithmetic_argument_publisher_;
  rclcpp::TimerBase::SharedPtr timer_;
  rclcpp::Subscription<rcl_interfaces::msg::ParameterEvent>::SharedPtr parameter_event_sub_;
  rclcpp::AsyncParametersClient::SharedPtr parameters_client_;
};
```

C++ 토픽 프로그래밍

```cpp
Argument::Argument(const rclcpp::NodeOptions & node_options)
: Node("argument", node_options),
  min_random_num_(0.0),
  max_random_num_(0.0)
{
  this->declare_parameter("qos_depth", 10);
  int8_t qos_depth = this->get_parameter("qos_depth").get_value<int8_t>();
  this->declare_parameter("min_random_num", 0.0);
  min_random_num_ = this->get_parameter("min_random_num").get_value<float>();
  this->declare_parameter("max_random_num", 9.0);
  max_random_num_ = this->get_parameter("max_random_num").get_value<float>();
  this->update_parameter();

  const auto QOS_RKL10V =
    rclcpp::QoS(rclcpp::KeepLast(qos_depth)).reliable().durability_volatile();

  arithmetic_argument_publisher_ =
    this->create_publisher<ArithmeticArgument>("arithmetic_argument", QOS_RKL10V);

  timer_ =
    this->create_wall_timer(1s, std::bind(&Argument::publish_random_arithmetic_arguments, this));
}
```

QOS설정

Publisher노드 초기화의
두 번째 인자로 QOS

1초에 한 번씩 호출

C++ 토픽 프로그래밍

```cpp
void Argument::publish_random_arithmetic_arguments()
{
  std::random_device rd;
  std::mt19937 gen(rd());
  std::uniform_real_distribution<float> distribution(min_random_num_, max_random_num_);

  msg_srv_action_interface_example::msg::ArithmeticArgument msg;
  msg.stamp = this->now();
  msg.argument_a = distribution(gen);
  msg.argument_b = distribution(gen);
  arithmetic_argument_publisher_->publish(msg);

  RCLCPP_INFO(this->get_logger(), "Published argument_a %.2f", msg.argument_a);
  RCLCPP_INFO(this->get_logger(), "Published argument_b %.2f", msg.argument_b);
}
```

토픽 메세지 통신에 사용하는 msg 인터페이스

msg로 송신받은 숫자를 CLI에 로그로 표시해준다

C++ 토픽 프로그래밍

토픽 서브스크라이버 설계

```
const auto QOS_RKL10V =
  rclcpp::QoS(rclcpp::KeepLast(qos_depth)).reliable().durability_volatile();

arithmetic_argument_subscriber_ = this->create_subscription<ArithmeticArgument>(
  "arithmetic_argument",
  QOS_RKL10V,
  [this](const ArithmeticArgument::SharedPtr msg) -> void
  {
    argument_a_ = msg->argument_a;
    argument_b_ = msg->argument_b;

    RCLCPP_INFO(
      this->get_logger(),
      "Subscribed at: sec %ld nanosec %ld",
      msg->stamp.sec,
      msg->stamp.nanosec);

    RCLCPP_INFO(this->get_logger(), "Subscribed argument a : %.2f", argument_a_);
    RCLCPP_INFO(this->get_logger(), "Subscribed argument b : %.2f", argument_b_);
  }
);
```

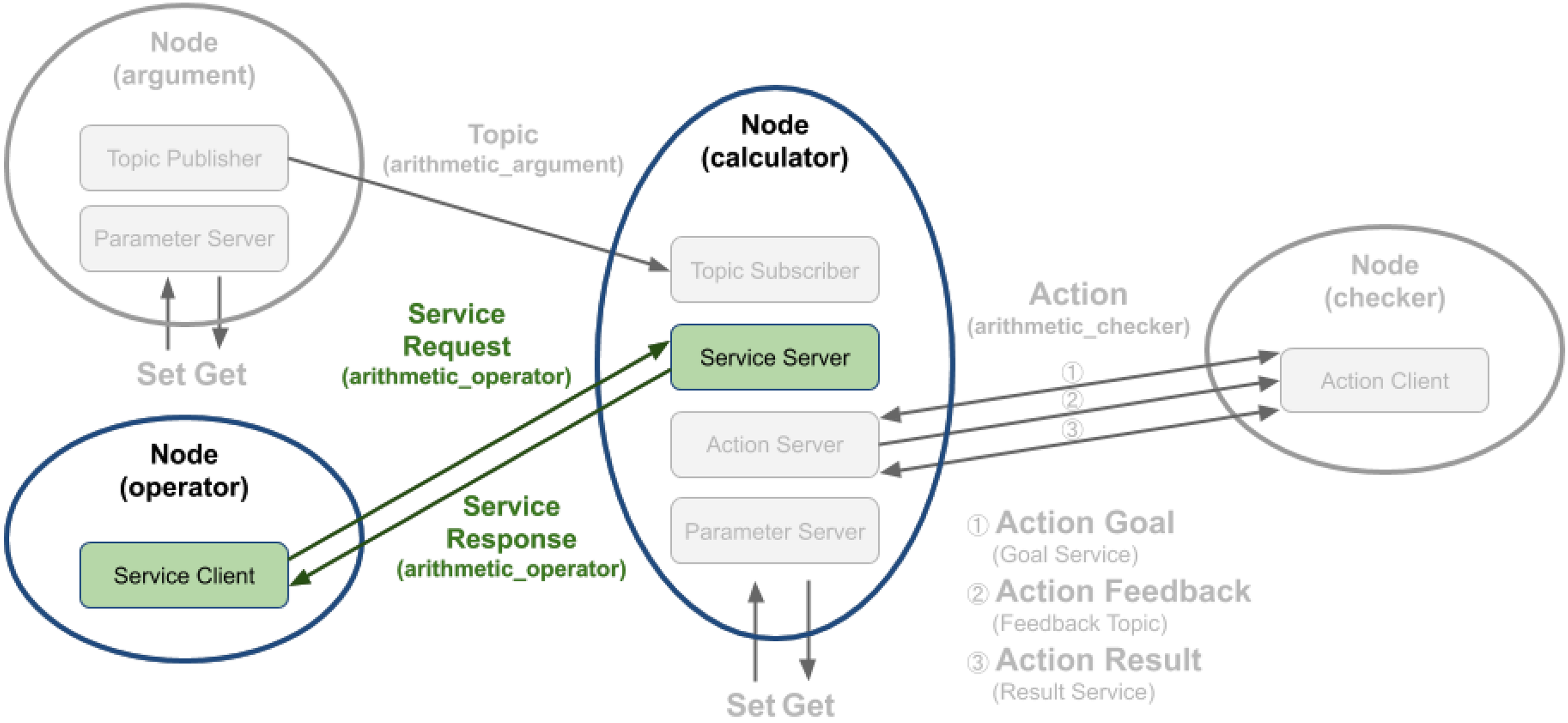
람다 표현식 사용

퍼블리시한 랜덤 숫자와 시간을 받아오고 CLI에 출력함

서비스 프로그래밍 C++

03

C++을 이용하여 서비스 프로그래밍 하기


Node
(argument)
Topic Publisher
Parameter Server
Set Get
Topic
(arithmetic_argument)
Node
(calculator)
Topic Subscriber
Service Server
Action Server
Parameter Server
Set Get
Service
Request
(arithmetic_operator)
Service
Response
(arithmetic_operator)
Node
(operator)
Service Client
Action
(arithmetic_checker)
①
②
③
Node
(checker)
Action Client
① Action Goal
(Goal Service)
② Action Feedback
(Feedback Topic)
③ Action Result
(Result Service)

서비스 서버

1) Node 설정
2) create_server 설정
3) 콜백함수 설정

서비스 클라이언트

1) Node 설정
2) create_client 설정
3) 요청함수 설정

서비스 서버 설계

```
auto get_arithmetic_operator =
  [this](
  const std::shared_ptr<ArithmeticOperator::Request> request,
  std::shared_ptr<ArithmeticOperator::Response> response) -> void
  {
    argument_operator_ = request->arithmetic_operator;
    argument_result_ =
      this->calculate_given_formula(argument_a_, argument_b_, argument_operator_);
    response->arithmetic_result = argument_result_;

    std::ostringstream oss;
    oss << std::to_string(argument_a_) << ' ' <<
      operator_[argument_operator_ - 1] << ' ' <<
      std::to_string(argument_b_) << " = " <<
      argument_result_ << std::endl;
    argument_formula_ = oss.str();

    RCLCPP_INFO(this->get_logger(), "%s", argument_formula_.c_str());
  };

arithmetic_argument_server_ =
  create_service<ArithmeticOperator>("arithmetic_operator", get_arithmetic_operator);
```

3) 콜백함수 설정

request와 response 인자를 이용한다.
미리 작성해둔 함수에 인자를 전달하고 그 결과를 리턴받는다.

C++ 서비스 프로그래밍

서비스 클라이언트 설계

```
class Operator : public rclcpp::Node
{
public:
  using ArithmeticOperator = msg_srv_action_interface_example::srv::ArithmeticOperator;

  explicit Operator(const rclcpp::NodeOptions & node_options = rclcpp::NodeOptions());
  virtual ~Operator();

  void send_request();

private:
  rclcpp::Client<ArithmeticOperator>::SharedPtr arithmetic_service_client_;
};
```

1) 노드설정

서비스 요청을 위한
send_request
함수가 있다.

C++ 서비스 프로그래밍

```cpp
Operator::Operator(const rclcpp::NodeOptions & node_options)
: Node("operator", node_options)
{
  arithmetic_service_client_ = this->create_client<ArithmeticOperator>("arithmetic_operator");
  while (!arithmetic_service_client_->wait_for_service(1s)) {
    if (!rclcpp::ok()) {
      RCLCPP_ERROR(this->get_logger(), "Interrupted while waiting for the service.");
      return;
    }
    RCLCPP_INFO(this->get_logger(), "Service not available, waiting again...");
  }
}
```

2) Create_client 설정

서비스명을 인자로 받아 rclcpp::Client 를 실체화 시켜준다

서비스 서버가 없다면 실행하지 않고 기다리는 코드가 있다.

C++ 서비스 프로그래밍

```cpp
void Operator::send_request()
{
  std::random_device rd;
  std::mt19937 gen(rd());
  std::uniform_int_distribution<int> distribution(1, 4);

  auto request = std::make_shared<ArithmeticOperator::Request>();
  request->arithmetic_operator = distribution(gen);

  using ServiceResponseFuture = rclcpp::Client<ArithmeticOperator>::SharedFuture;
  auto response_received_callback = [this](ServiceResponseFuture future) {
      auto response = future.get();
      RCLCPP_INFO(this->get_logger(), "Result %.2f", response->arithmetic_result);
      return;
    };

  auto future_result =
    arithmetic_service_client_->async_send_request(request, response_received_callback);
}
```

3)요청함수 설정

response_received_callback

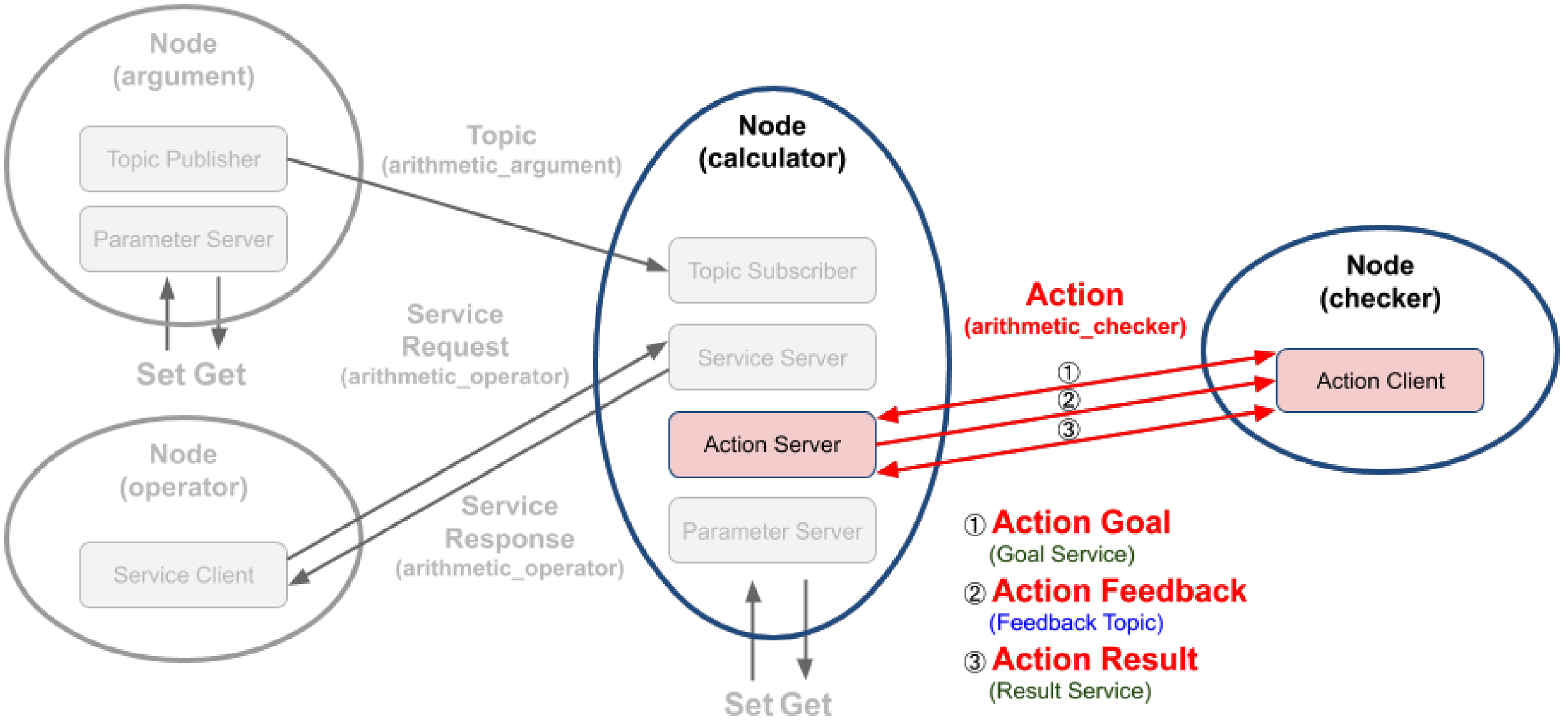
request와 response 콜백함수를 이용하여 로그로 출력한다.

async_send_request 함수를 통해 비동기식으로 서비스 요청을 보낸다.

액션 프로그래밍 C++

04

C++을 이용하여 액션 프로그래밍 하기


Node
(argument)
Topic Publisher
Parameter Server
Set Get
Topic
(arithmetic_argument)
Node
(calculator)
Topic Subscriber
Service Server
Action Server
Parameter Server
Set Get
Service
Request
(arithmetic_operator)
Service
Response
(arithmetic_operator)
Node
(operator)
Service Client
Action
(arithmetic_checker)
①
②
③
Node
(checker)
Action Client
① Action Goal
(Goal Service)
② Action Feedback
(Feedback Topic)
③ Action Result
(Result Service)

액션 서버

1) Node 설정
2) create_server 설정
3) goal, cancel, accepted 콜백 함수 설정

액션 클라이언트

1) Node 설정
2) create_client 설정
3) goal_response, feedback, result 콜백 함수 설정

액션 서버

```
arithmetic_action_server_ = rclcpp_action::create_server<ArithmeticChecker>(
  this->get_node_base_interface(),
  this->get_node_clock_interface(),
  this->get_node_logging_interface(),
  this->get_node_waitables_interface(),
  "arithmetic_checker",
  std::bind(&Calculator::handle_goal, this, _1, _2),
  std::bind(&Calculator::handle_cancel, this, _1),
  std::bind(&Calculator::execute_checker, this, _1)
);
```

토픽과 서비스 통신을 위한 멤버 변수들은 rclcpp 네임스페이스를 가지는데 반해
액션 통신을 위한 멤버 변수들은 rclcpp_action네임스페이스를 가진다.

C++ 액션 프로그래밍

```
rclcpp_action::GoalResponse Calculator::handle_goal(
  const rclcpp_action::GoalUUID & uuid,
  std::shared_ptr<const ArithmeticChecker::Goal> goal)
{
  (void)uuid;
  (void)goal;
  return rclcpp_action::GoalResponse::ACCEPT_AND_EXECUTE;
}
```

다양한 콜백함수의 설정

```
rclcpp_action::CancelResponse Calculator::handle_cancel(
  const std::shared_ptr<GoalHandleArithmeticChecker> goal_handle)
{
  RCLCPP_INFO(this->get_logger(), "Received request to cancel goal");
  (void)goal_handle;
  return rclcpp_action::CancelResponse::ACCEPT;
}
```

C++ 액션 프로그래밍

```
class Checker : public rclcpp::Node
{
public:
  using ArithmeticChecker = msg_srv_action_interface_example::action::ArithmeticChecker;
  using GoalHandleArithmeticChecker = rclcpp_action::ClientGoalHandle<ArithmeticChecker>;

  explicit Checker(
    float goal_sum,
    const rclcpp::NodeOptions & node_options = rclcpp::NodeOptions());
  virtual ~Checker();

private:
  void send_goal_total_sum(float goal_sum);

  void get_arithmetic_action_goal(
    std::shared_future<rclcpp_action::ClientGoalHandle<ArithmeticChecker>::SharedPtr> future);

  void get_arithmetic_action_feedback(
    GoalHandleArithmeticChecker::SharedPtr,
    const std::shared_ptr<const ArithmeticChecker::Feedback> feedback);

  void get_arithmetic_action_result(
    const GoalHandleArithmeticChecker::WrappedResult & result);

  rclcpp_action::Client<ArithmeticChecker>::SharedPtr arithmetic_action_client_;
};
```

액션 클라이언트

1) 노드 설정

2) Create client 설정

```
Checker::Checker(float goal_sum, const rclcpp::NodeOptions & node_options)
: Node("checker", node_options)
{
  arithmetic_action_client_ = rclcpp_action::create_client<ArithmeticChecker>(
    this->get_node_base_interface(),
    this->get_node_graph_interface(),
    this->get_node_logging_interface(),
    this->get_node_waitables_interface(),
    "arithmetic_checker");

  send_goal_total_sum(goal_sum);
}
```

부모 클래스인 rclcpp::Node를 노드명과 node_options 인자로 먼저 초기화해 준다.
그리고rclcpp_action::create_client 함수를 통해 해당 노드의 인터페이스들과 액션명을 인자로 받아
rclcpp_action::Client를 실체화 시켜준다.

C++ 액션 프로그래밍

```cpp
void Checker::send_goal_total_sum(float goal_sum)
{
  using namespace std::placeholders;

  if (!this->arithmetic_action_client_) {
    RCLCPP_WARN(this->get_logger(), "Action client not initialized");
  }

  if (!this->arithmetic_action_client_->wait_for_action_server(std::chrono::seconds(10))) {
    RCLCPP_WARN(this->get_logger(), "Arithmetic action server is not available.");
    return;
  }

  auto goal_msg = ArithmeticChecker::Goal();
  goal_msg.goal_sum = goal_sum;

  auto send_goal_options = rclcpp_action::Client<ArithmeticChecker>::SendGoalOptions();
  send_goal_options.goal_response_callback =
    std::bind(&Checker::get_arithmetic_action_goal, this, _1);
  send_goal_options.feedback_callback =
    std::bind(&Checker::get_arithmetic_action_feedback, this, _1, _2);
  send_goal_options.result_callback =
    std::bind(&Checker::get_arithmetic_action_result, this, _1);
  this->arithmetic_action_client_->async_send_goal(goal_msg, send_goal_options);
}
```

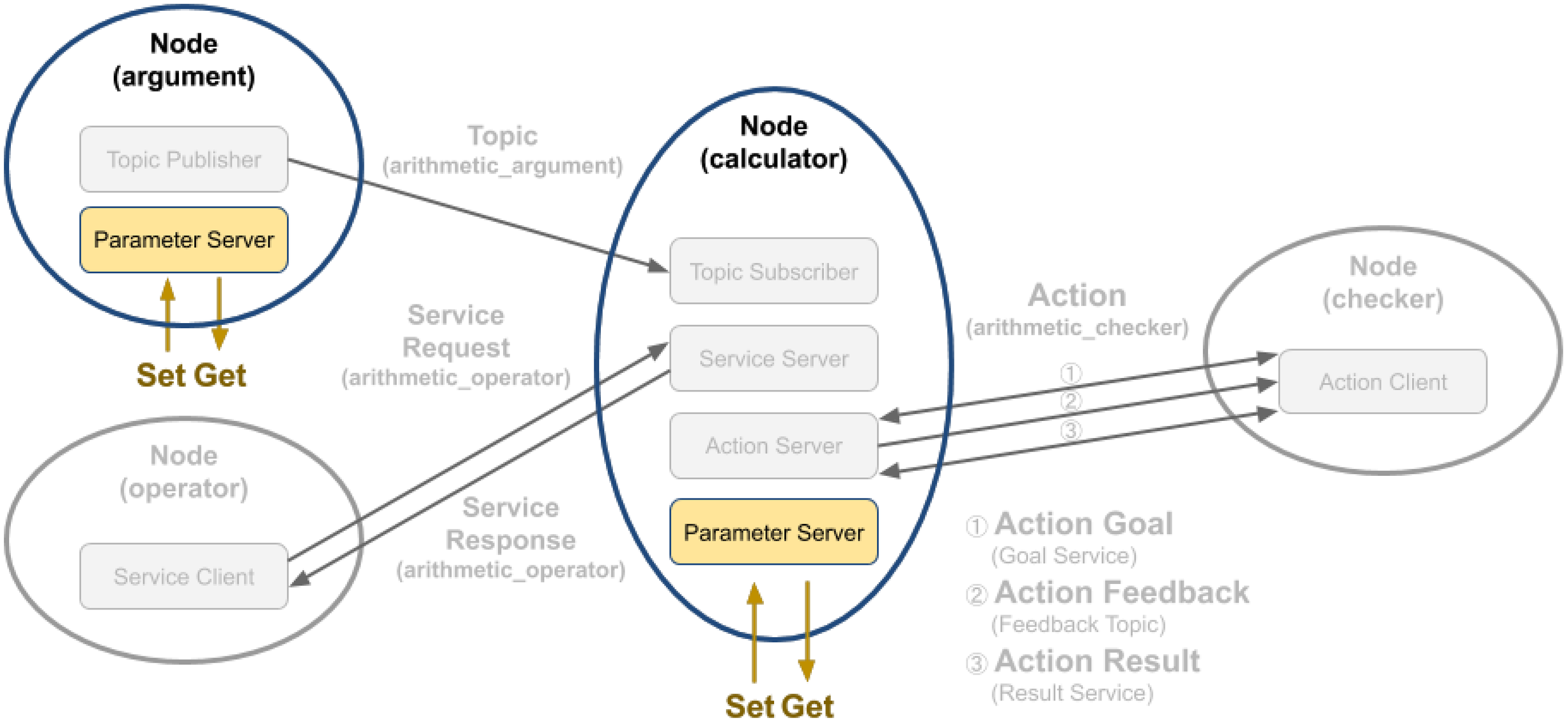
3) Create client 설정

rclcpp_action::Client의 SendGoalOptions 구조체를 통해 goal_response_callback, feedback_callback, result_callback 함수를 초기화 시켜준다.

파라미터 프로그래밍 C++

05

C++을 이용하여 파라미터 프로그래밍 하기


Node
(argument)
Topic Publisher
Parameter Server
Set Get
Topic
(arithmetic_argument)
Node
(calculator)
Topic Subscriber
Service Server
Action Server
Parameter Server
Set Get
Service
Request
(arithmetic_operator)
Service
Response
(arithmetic_operator)
Node
(operator)
Service Client
Action
(arithmetic_checker)
①
②
③
Node
(checker)
Action Client
① Action Goal
(Goal Service)
② Action Feedback
(Feedback Topic)
③ Action Result
(Result Service)

C++ 파라미터 프로그래밍

파라미터 서버

1) parameter.yaml 설정
2) launch 파일 설정

파라미터 클라이언트

1) declare_parameter 함수로 사용할 파라미터 등록
2) get_parameter 함수로 파라미터 값 회수
3) parameter_event 콜백 함수 설정

C++ 파라미터 프로그래밍

1) parameter.yaml 설정

```yaml
/**: # namespace and node name
  ros__parameters:
    qos_depth: 30
    min_random_num: 0.0
    max_random_num: 9.0
```

2) launch 파일 설정

```python
def generate_launch_description():
    param_dir = LaunchConfiguration(
        'param_dir',
        default=os.path.join(
            get_package_share_directory('topic_service_action_rclcpp_example'),
            'param',
            'arithmetic_config.yaml'))

    return LaunchDescription([
        DeclareLaunchArgument(
            'param_dir',
            default_value=param_dir,
            description='Full path of parameter file'),
```

C++ 파라미터 프로그래밍

1) declare_parameter 함수로 사용할 파라미터 등록

2) get_parameter 함수로 파라미터 값 회수

```
this->declare_parameter("qos_depth", 10);
int8_t qos_depth = this->get_parameter("qos_depth").get_value<int8_t>();
this->declare_parameter("min_random_num", 0.0);
min_random_num_ = this->get_parameter("min_random_num").get_value<float>();
this->declare_parameter("max_random_num", 9.0);
max_random_num_ = this->get_parameter("max_random_num").get_value<float>();
this->update_parameter();
```

C++ 파라미터 프로그래밍

```cpp
void Argument::update_parameter()
{
  parameters_client_ = std::make_shared<rclcpp::AsyncParametersClient>(this);
  while (!parameters_client_->wait_for_service(1s)) {
    if (!rclcpp::ok()) {
      RCLCPP_ERROR(this->get_logger(), "Interrupted while waiting for the service. Exiting.");
      return;
    }
    RCLCPP_INFO(this->get_logger(), "service not available, waiting again...");
  }

  auto param_event_callback =
    [this](const rcl_interfaces::msg::ParameterEvent::SharedPtr event) -> void
    {
      for (auto & changed_parameter : event->changed_parameters) {
        if (changed_parameter.name == "min_random_num") {
          auto value = rclcpp::Parameter::from_parameter_msg(changed_parameter).as_double();
          min_random_num_ = value;
        } else if (changed_parameter.name == "max_random_num") {
          auto value = rclcpp::Parameter::from_parameter_msg(changed_parameter).as_double();
          max_random_num_ = value;
        }
      }
    };

  parameter_event_sub_ = parameters_client_->on_parameter_event(param_event_callback);
}
```

3) parameter_event 콜백 함수 설정

런타임에서 파라미터 서버에 이벤트(등록, 변경, 삭제)가 있을 때 콜백되는 함수를 등록할 수 있고, 이를 통해 파라미터 값이 변경되었을 때를 확인할 수 있다.

실행인자 프로그래밍 C++

06

C++을 이용하여 실행인자 프로그래밍 하기

C++ 실행인자 프로그래밍

```
if (rcutils_cli_option_exist(argv, argv + argc, "-h")) {
  print_help();
  return 0;
}
```

'-h' 인자가 있는지 검사한다. 만약 '-h'가 검출되면 print_help 함수를 출력하고 main함수를 빠져나간다.
이를 통해 사용자가 해당 노드를 처음 사용하게 될때 필요한 정보를 제공해준다.

```
float goal_total_sum = 50.0;
char * cli_option = rcutils_cli_get_option(argv, argv + argc, "-g");
if (nullptr != cli_option) {
  goal_total_sum = std::stof(cli_option);
}
printf("goal_total_sum : %2.f\n", goal_total_sum);


auto checker = std::make_shared<Checker>(goal_total_sum);
```

rcutils_cli_get_option 함수는 실행 인자를 확인하고 그 값을 문자열 포인터로 반환해주는 역할을 한다.
사용자는 쉽게 여러개의 실행 인자를 파싱할 수 있고, 문자열 포인터를 원하는 변수 타입으로 변경하여 노드의 생성 인자로 넘겨줄 수 있다.

런치 프로그래밍 C++

07

C++을 이용하여 런치 프로그래밍 하기

런치 프로그래밍의 기본 매소드

```
def generate_launch_description():

    xxx = LaunchConfiguration(yyy)

    return LaunchDescription([
        DeclareLaunchArgument(aaa),
        Node(bbb),
        Node(ccc),
    ])
```

remappings 기능

내부 코드 변경없이 토픽, 서비스, 액션 등의 고유 이름을 변경할 수 있는 유용한 기능

```
        Node(
            package='topic_service_action_rclpy_example',
            executable='argument',
            name='argument',
            remappings=[
                ('/arithmetic_argument', '/argument'),
            ]
```

RCLCPP 패키지 계열 빌드

C++ 언어를 사용하는 경우 하기와 같이 빌드 설정 파일(CMakeLists.txt)의
install 구문에 launch 라는 폴더명만 기재하면 된다.

```
install(DIRECTORY
  launch
  DESTINATION share/${PROJECT_NAME}/
)
```

감사합니다