Week2

5장 ROS 2의 중요 콘셉트

ROS 2의 중요 콘셉트를 10가지로 압축하여 정리하였다.

1. 시장 출시 시간 단축
2. 생산을 위한 설계
3. 멀티 플랫폼
4. 다중 도메인
5. 벤더 선택가능
6. 공개 표준 기반
7. 자유 재량 허용 범위가 넓은 오픈소스 라이센스 채택
8. 글로벌 커뮤니티
9. 산업 지원
10. ROS 1과의 상호 운용성 확보

6장 ROS 1과 2의 차이점으로 알아보는 ROS 2특징

2007년 개발이 시작된 ROS 1은 다음과 같은 제한 사항이 있었다.

- 단일 로봇
- 워크 스테이션급 컴퓨터
- 리눅스 환경
- 실시간 제어 지원하지 않음
- 안정된 네트워크 환경 요구
- 대학이나 연구소의 아카데믹 연구 목적 사용

이러한 ROS 1의 제한 사항들을 보완하기 위해 새로운 로봇 개발 환경의 필요성이 대두 되었고 요구되는 기능을 정리하면 다음과 같다.

- 두대 이상의 로봇
- 임베디드 시스템에서의 ROS 사용
- 실시간 제어
- 불안정한 네트워크 환경에서도 동작할 수 있는 유연함
- 멀티 플랫폼(리눅스, 윈도우, macOS)
- 최신 기술 지원(Zeroconf, Protocol Buffers, ZeroMQ, WebSockets, DDS 등)
- 상업용 제품 지원

ROS 1에서 새로운 요구에 적합한 기능을 제공하기 위해선 대규모의 API의 변경이 필요하기 때문에 차세대 기능을 도입한 버전을 ROS 2로 명명하여 ROS 1과 분리 개발되었다.

7장 ROS 2와 DDS

DDS는 데이터 분산 서비스(Data Distribution Service)의 약자로 개념을 제쳐두고 그 실체는 결국 데이터 통신을 위한 미들웨어이다. 이 미들웨어는 운영체제와 사용자 애플리케이션 사이에서 통신하면서 데이터를 공유 할 수 있게 해준다.

DDS가 ROS 2의 미들웨어로 사용되면서 보이는 특징은 다음과 같다.

- Industry Standards (산업 표준)
- OS Independent (운영체제 독립)
- Language Independent (언어 독립)
- Transport on UDP/IP (UDP 기반의 전송 방식)
- Data Centricity (데이터 중심적 기능)
- Dynamic Discovery (동적 검색)
- Scalable Architecture (확장 가능한 아키텍쳐)
- Interoperability (상호 운용성)
- Quality of Service(QoS) (서비스 품질)
- Security (보안)

ROS에서의 사용법

데모 패키지인 demo_nodes_cpp에서 제공하는 간단한 Publiher&Subscriber를 이용하여 DDS의 설정값을 변경하는 테스트를 진행 해 보았다.

```
$ ros2 run demo_nodes_cpp talker
[INFO] [1689142828.055060335] [talker]: Publishing: 'Hello World: 1'
[INFO] [1689142829.055160318] [talker]: Publishing: 'Hello World: 2'
[INFO] [1689142830.055340116] [talker]: Publishing: 'Hello World: 3'
[INFO] [1689142831.055177983] [talker]: Publishing: 'Hello World: 4'
[INFO] [1689142832.055087728] [talker]: Publishing: 'Hello World: 5'
```

```
$ ros2 run demo_nodes_cpp listener
[INFO] [1689142879.056848458] [listener]: I heard: [Hello World: 52]
[INFO] [1689142880.056751515] [listener]: I heard: [Hello World: 53]
[INFO] [1689142881.056680753] [listener]: I heard: [Hello World: 54]
[INFO] [1689142882.056798971] [listener]: I heard: [Hello World: 55]
[INFO] [1689142883.056759539] [listener]: I heard: [Hello World: 56]
[INFO] [1689142884.056810280] [listener]: I heard: [Hello World: 57]
[INFO] [1689142885.056697356] [listener]: I heard: [Hello World: 58]
```

```
$ rqt_graph
```

- **RMW 변경 방법**

RMW에 아무 설정도 하지 않으면 기본인 rmw_fastrtps_cpp가 사용된다. 만약 RMW를 변경해 사용하려면 다음중 하나를 RMW_IMPLEMENTATION 환경변수로 지정한 후 노드를 실행하면 된다.

- rmw_connext_cpp
- rmw_cyclonedds_cpp
- rmw_fastrtps_cpp
- rmw_gurumdds_cpp

```
$ export RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
$ ros2 run demo_nodes_cpp talker
1689143658.033458 [0]     talker: using network interface wlp0s20f3 (udp/192.168.0.1) selected arbitrarily from: wlp0s20f3, br-75a99bc
[INFO] [1689143659.035874708] [talker]: Publishing: 'Hello World: 1'
[INFO] [1689143660.035847999] [talker]: Publishing: 'Hello World: 2'
[INFO] [1689143661.036038406] [talker]: Publishing: 'Hello World: 3'
[INFO] [1689143662.035902417] [talker]: Publishing: 'Hello World: 4'
[INFO] [1689143663.035993514] [talker]: Publishing: 'Hello World: 5'
```

```
$ ros2 run demo_nodes_cpp listener
[INFO] [1689143690.038594038] [listener]: I heard: [Hello World: 32]
[INFO] [1689143691.038611882] [listener]: I heard: [Hello World: 33]
[INFO] [1689143692.038666775] [listener]: I heard: [Hello World: 34]
[INFO] [1689143693.038633554] [listener]: I heard: [Hello World: 35]
[INFO] [1689143694.038909460] [listener]: I heard: [Hello World: 36]
```

위와 같이 서로 다른 두 RMW를 사용해도 정상적으로 통신이 되는 것을 볼 수 있다. 이러한 DDS의 특성이 Interoperability (상호 운용성)이다.

- **Domain 변경 방법**

ROS2에서는 UDP 멀티캐스트로 통신이 이루어지기 때문에 별도의 설정을 하지 않으면 동일 네트워크상에 있는 모든 노드가 연결되게 된다. 예를 들어 같은 연구실에서 동일 네트워크를 사용하는 다른 연구원이 사용하는 노드에도 접근가능하게 된다. 이를 방지하는 방법으로 노드를 실행하기 전 DDS의 Domain을 변경하는 것으로 해결 가능하다. ROS의 RMW에서는 ROS_DOMAIN_ID라는 환경변수로 설정하게 되어있다.

```
$ export ROS_DOMAIN_ID=11
$ ros2 run demo_nodes_cpp talker
[INFO] [1689144187.110068217] [talker]: Publishing: 'Hello World: 1'
[INFO] [1689144188.110081011] [talker]: Publishing: 'Hello World: 2'
[INFO] [1689144189.110012544] [talker]: Publishing: 'Hello World: 3'
[INFO] [1689144190.110035765] [talker]: Publishing: 'Hello World: 4'
[INFO] [1689144191.110172184] [talker]: Publishing: 'Hello World: 5'
```

```
$ export ROS_DOMAIN_ID=12
$ ros2 run demo_nodes_cpp listener
(다른 도메인을 사용하여 아무런 토픽도 Subscribe 되지 않는다.)

$ export ROS_DOMAIN_ID=11
$ ros2 run demo_nodes_cpp listener
[INFO] [1689144222.111563102] [listener]: I heard: [Hello World: 36]
[INFO] [1689144223.111564744] [listener]: I heard: [Hello World: 37]
[INFO] [1689144224.111523446] [listener]: I heard: [Hello World: 38]
[INFO] [1689144225.111292263] [listener]: I heard: [Hello World: 39]
```

위와 같이 서로 같은 Domain에서만 노드 간의 통신이 가능하게 된다.

8장 DDS의 QoS

QoS는 간단히 말해서 “데이터 통신 옵션”으로 데이터 통신을 위한 Publisher, Subscriber를 생성할 때 TCP처럼 신뢰성을 중시 여기는 통신 방식과 UDP처럼 통신 속도에 초점을 둔 통신 방식을 선택적으로 사용할 수 있다.

ROS 2에서는 DDS에서 제공되는 QoS 중에서 History, Reliability, Durability, Deadline, Lifespan, Liveliness를 지원한다.

ROS 2에서 사용되는 QoS 옵션

- **History**

History	데이터를 몇개나 보관할지를 결정하는 QoS 옵션
KEEP_LAST	정해진 메시지 큐크기만큼의 데이터를 보관
KEEP_ALL	모든 데이터를 보관

```
[RCLPY]
qos_profile = QoSProfile(history=QoSHistoryPolicy.KEEP_LAST, depth=10)
```

- **Reliability**

Reliability	데이터 전송에 있어 속도를 우선시 하는지 신뢰성을 우선시 하는지를 결정하는 QoS옵션
BEST_EFFORT	데이터 송신에 집중, 전송 속도를 중시하고 네트워크 상태에 따라 유실이 발생 가능
RELIABLE	데이터 수신에 집중, 신뢰성을 중시하며 유실이 발생하면 재전송을 통해 수신을 보장함

RxO

Pub\Sub	BEST_EFFORT	RELIABLE
BEST_EFFORT	BEST_EFFORT	**불가**
RELIABLE	BEST_EFFORT	RELIABLE

```
[RCLPY]
qos_profile = QoSProfile(reliability=QoSReliabilityPolicy.BEST_EFFORT)
```

- **Durability**

Durability	데이터를 수신하는 subscriber가 생성되기 전의 데이터를 사용할 것인지에 대한 Qos 옵션
TRANSIENT_LOCAL	Subscription이 생성되기 전의 데이터도 보관
VOLATILE	Subscription이 생성되기 전의 데이터는 무효

RxO

Pub\Sub	TRANSIENT_LOCAL	VOLATILE
TRANSIENT_LOCAL	TRANSIENT_LOCAL	VOLATILE
VOLATILE	**불가**	VOLATILE

```
[RCLPY]
qos_profile = QoSProfile(durability=QoSDurabilityPolicy.TRANSIENT_LOCAL)
```

- **Deadline**

Deadline	정해진 주기한에 데이터가 수/발신되지 않을 경우 EventCallback을 실행시키는 Qos 옵션
deadline_duration	Deadline을 확인하는 주기

RxO

Pub\Sub	1000ms	2000ms
1000ms	가능	가능
2000ms	**불가**	가능

```
[RCLPY]
qos_profile = QoSProfile(depth=10, deadline=Duration(0.1))
```

- Lifespan

Lifespan	정해진 주기 안에서 수신되는 데이터만 유효 판정하고 그렇지 않은 데이터는 삭제하는 Qos 옵션
lifespan_duration	lifespan을 확인하는 주기

```
[RCLPY]
qos_profile = QoSProfile(lifespan=Duration(0.01))
```

- Liveness

Liveness	정해진 주기 안에서 노드/토픽의 생사를 확인하는 Qos 옵션
liveness	자동/메뉴얼 로 확인할지를 지정하는 옵션
lease_duration	Liveness을 확인하는 주기

RxO

Pub\Sub	AUTOMATIC	MANUAL_BY_HAND	MANUAL_BY_TOPIC
AUTOMATIC	가능	**불가**	**불가**
MANUAL_BY_HAND	가능	가능	**불가**
MANUAL_BY_TOPIC	가능	가능	가능

```
[RCLPY]
qos_profile = QoSProfile(
  liveness=AUTOMATIC,
  liveness_lease_duration=Duration(1.0))
```

rmw_qos_profile

ROS 2의 RMW에서 QoS 설정을 쉽게 사용할 수 있게 RMW QoS Profile을 지정해 두었다. 그 목적에 따라 Default, Sensor Data, Service, Action Status, Parameters, Parameter Events와 같이 6가지로 구분하였고, Reliability, History, Depth, Durability를 설정하게 된다.

각각의 자세한 설정 값은 아래의 링크를 참조하면 된다.

https://github.com/ros2/rmw/blob/foxy/rmw/include/rmw/qos_profiles.h

이 중에서 Sensor Data옵션을 실제 코드에 적용하는 법은 다음과 같다. 먼저 qos_profile_sensor_data 을 import를 수행하고 publisher를 선언하는 create_publisher 함수를 사용할 때 qos_profile_sensor_data를 매개변수로 사용하면 된다.

```
[RCLPY]
from rclpy.qos import qos_profile_sensor_data
...
self.sensor_publisher = self.create_publisher(Int8MultiArray, 'sensor', qos_profile_sensor_data
```

ROS QoS 테스트

데모 패키지인 demo_nodes_py로 간단한 QoS 테스트를 진행하였다. demo_nodes_py 내부에 QoS 옵션을 변경할 수 있는 talker_qos와 listener_qos가 있다. 해당 노드들을 실행시켜 Reliability옵션을 테스트해보았다.

```
$ ros2 run demo_nodes_py talker_qos
[INFO] [1689145311.754858990] [talker_qos]: Best effort talker
[INFO] [1689145312.756676949] [talker_qos]: Publishing: "Hello World: 0"
[INFO] [1689145313.756659466] [talker_qos]: Publishing: "Hello World: 1"
[INFO] [1689145314.756665776] [talker_qos]: Publishing: "Hello World: 2"
[INFO] [1689145315.756660669] [talker_qos]: Publishing: "Hello World: 3"
[INFO] [1689145316.756712329] [talker_qos]: Publishing: "Hello World: 4"
[INFO] [1689145317.756778270] [talker_qos]: Publishing: "Hello World: 5"
```

```
ros2 run demo_nodes_py listener_qos --reliable
[INFO] [1689145224.891536025] [listener_qos]: Reliable listener
(RxO가 맞지 않아 아무런 반응이 없음을 확인 할 수 있다.)
```

Pub를 BEST_EFFORT, Sub를 RELIABLE로 하였을 때 RxO가 맞지 않아 송수신이 원활하지 않음을 볼 수 있다.

```
$ ros2 run demo_nodes_py talker_qos --reliable
[INFO] [1689145680.931980686] [talker_qos]: Reliable talker
[INFO] [1689145681.934007814] [talker_qos]: Publishing: "Hello World: 0"
[INFO] [1689145682.933878452] [talker_qos]: Publishing: "Hello World: 1"
[INFO] [1689145683.933876974] [talker_qos]: Publishing: "Hello World: 2"
[INFO] [1689145684.934001846] [talker_qos]: Publishing: "Hello World: 3"
[INFO] [1689145685.933956080] [talker_qos]: Publishing: "Hello World: 4"
[INFO] [1689145686.933878898] [talker_qos]: Publishing: "Hello World: 5"
```

```
$ ros2 run demo_nodes_py listener_qos --reliable
[INFO] [1689145224.891536025] [listener_qos]: Reliable listener
[INFO] [1689145299.244413125] [listener_qos]: I heard: [Hello World: 0]
[INFO] [1689145300.236254742] [listener_qos]: I heard: [Hello World: 1]
[INFO] [1689145301.236084735] [listener_qos]: I heard: [Hello World: 2]
[INFO] [1689145302.236156145] [listener_qos]: I heard: [Hello World: 3]
[INFO] [1689145303.236159003] [listener_qos]: I heard: [Hello World: 4]
[INFO] [1689145304.236407278] [listener_qos]: I heard: [Hello World: 5]
```

Pub & Sub를 둘다 RELIABLE로 하였을 때 송수신이 원활함을 볼 수 있다.