DDS 응용 개발 시 통합개발지원 환경의 효율적 활용을 위한 기법 개발 연구

**1차년도 보고서**

통합개발지원 환경을 활용한 DDS 응용 개발을 위한

설계 방법론 수립

2014.06.25

고려대학교 기술경영전문대학원 강석중 교수

목차

[I. 서론 3](#_Toc390833246)

[II. DDS 응용프로그램 개발을 위한 설계방법 4](#_Toc390833248)

[1. DIDE 컴포넌트 소프트웨어 4](#_Toc390833249)

[2. DIDE 개발 환경 4](#_Toc390833250)

[3. 설계 방법 6](#_Toc390833251)

[A. 토픽 설계 방법 9](#_Toc390833252)

[B. QoS 설정 방법 10](#_Toc390833253)

[C. 어플리케이션 설계 방법 11](#_Toc390833254)

[III. 결론 15](#_Toc390833257)

[부록: OMG QoS 요약 16](#_Toc390833258)

# 서론

본 보고서에서는 DIDE를 이용하여 DDS 응용프로그램을 개발하고자 하는 경우에 DIDE를 구성하는 각 컴포넌트를 이용하여 개발 및 시험하고자 하는 절차에 대하여 설명하고, 각 절차에 따라 지켜야 할 주의 사항에 대하여 언급하고자 한다.

*DIDE에 대하여 간략하게 소개한다.*



# DDS 응용프로그램 개발을 위한 설계방법

## DIDE 컴포넌트 소프트웨어

DIDE를 구성하는 컴포넌트 소프트웨어를 간단하게 정리하면 아래 표와 같다.

표 . DIDE 컴포넌트 기능 요약

|  |  |  |
| --- | --- | --- |
| **컴포넌트 명칭** | **기능** | **사용 단계** |
| 응용 모델러, | * 프로젝트 및 소스 코드 생성 * 어플리케이션 모델링 * 토픽 모델링 * QoS 관리 | 설계 및 구현 |
| 토픽 레포지토리 | * 토픽 정의, 생성, 등록, 알림, 버전관리 * IDL 파일 생성 | 설계 및 구현 |
| 토픽 분석기 | * 실시간 DDS 엔티티 속성, QoS 등의 정보 수집 및 비교 | 모니터링 및 분석 |
| 실시간 모니터 | * CPU/Memory usage * DDS 엔티티의 QoS 체킹 * DDS 토픽 통신 통계정보 수집 | 모니터링 및 분석 |
| 로그 관리자 | * 로그 수집 및 저장 * 로그 데이터 분석 | 모니터링 및 분석 |
| 토픽 기록기 | * 토픽 및 샘플 정보 수집 및 저장 * 토픽 재생 | 시험 및 개선 |
| 토픽 시뮬레이터 | * 시나리오 작성 및 관리 * 시나리오 실행 | 시험 및 개선 |
| 토픽 튜너 | * 로컬 및 리모트 DDS 엔티티 속성 조회 * 로컬 및 리모트 DDS 엔티티 QoS 설정 | 시험 및 개선 |
| DDS-DB 연동 | * DB 연동 기능 제공 | 지원도구 |
| 고가용성 지원 | * 프로세스 모니터링을 통하여 고가용성 기능 제공 | 지원도구 |
| 형상관리 도구 | * 산출물 버전 관리 기능 제공 | 지원도구 |
| DDS 엔진 연동 | * 다양한 DDS 엔진에 대하여 단일 인터페이스 제공 기능 | 지원 도구 |

## DIDE 개발 환경

DIDE 전체 도구의 구조도는 아래 그림과 같다. 이 도구들은 설계, 구현, 시험 등의 일반적인 SW개발 과정과 크게 다르지 않으며, 각 개발 단계에서 효율적으로 사용될 수 있도록 개발되어 있다.

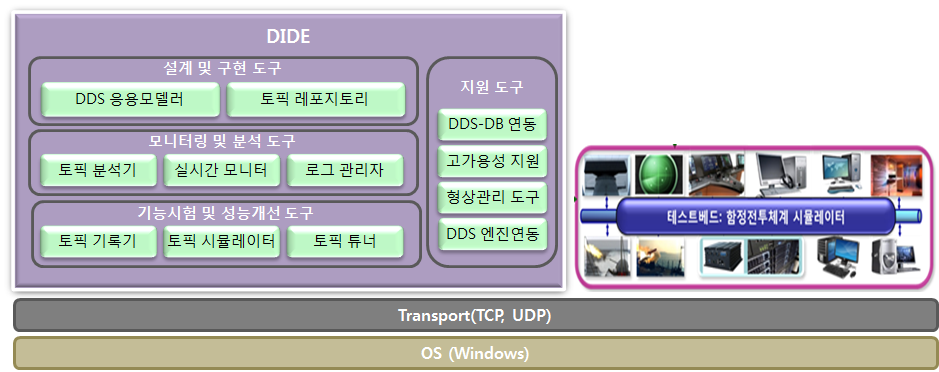


그림 . DIDE 구조도

위의 구조도를 다음 그림과 같이 연관 관계를 살펴볼 수 있다.

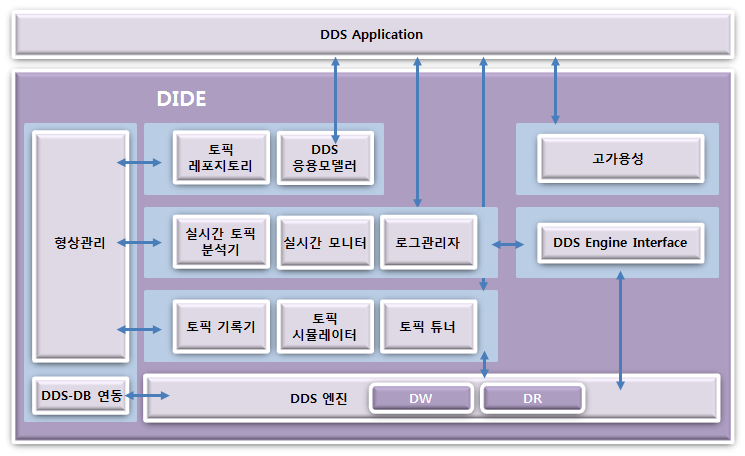


그림 . DIDE 컴포넌트 소프트웨어 구조도

그림 . 컴포넌트간 연관 관계

## 설계 방법

각 컴포넌트를 이용하여 DDS 응용프로그램을 개발하고 시험하는 일반적인 절차는 아래 그림과 같이 정리할 수 있다.



그림 . DDS 응용프로그램 개발 및 시험 절차

DIDE 환경에서 DDS 어플리케이션을 개발하기 위한 절차는 위 그림과 같다. 위와 같은 단계에서 설게방법론이 적용되는 부분은 응용모델러를 이용하여 DDS 어플리케이션을 개발하는 단계에 집중된다고 할 수 있다. 응용모델러 사용절차는 다음 그림과 같다.

그림 . DDS 응용모델러 기능 수행 절차

위 절차를 간략히 설명하면 아래와 같다.

* 이클립스의 프로젝트 생성 메뉴에서 DDS 응용프로그램 타입의 프로젝트를 생성한다.
* 생성된 프로젝트에 DDS 엔티티를 갖기 위한 Application 모듈을 생성한다. 이 Application 모듈은 향후 실행 파일로 변환된다.
* 생성된 Application 모듈 내에 GUI Editor를 이용하여 Domain participant를 생성한다.
* Domain participant가 생성된 후 Publisher 또는 Subscriber를 Domain participant내에 생성한다.
* 토픽을 주고 받을 DataReader 또는 DataWriter를 Publisher 또는 Subscriber내에 생성한다.
* 생성된 DDS 엔티티(DomainParticipant, Publisher, Subscriber, DataWriter, DataReader, Topic 등)에 대하여 QoS를 설정한다.
* GUI Editor를 이용하여 생성된 DDS 엔티티 사이의 관계를 설정한다.
* 소스 코드를 생성한다.

위와 같은 절차 중 설계 방법론이 적용되는 부분을 크게 3 가지 정도로 분류하여 보았다.

### 토픽 설계 방법

#### 데이터 모델링

DDS 토픽은 데이터의 송수신 단위로 일반적인 TCP/IP 또는 시리얼통신 등을 위하여 패킷을 정의할 때와 유사한 방법으로 데이터 모델을 정의할 필요가 있으며, 이렇게 정의된 데이터 모델을 도메인, 데이터 타입 및 토픽으로 매핑을 시켜야 한다. 토픽으로 매핑 시에는 토픽의 상태 정보를 고려해야 한다. 예를 들어 화물 운송 회사의 경우 트럭 및 화물에 대한 정보를 송수신한다고 가정하자. 이 경우 트럭의 위치를 나타내는 GPS 정보, 트럭의 기름의 양, 화물 목록, 정비관련 정보 등등이 있을 수 있다. 이러한 정보들은 주기적인지 비주기적인지 또는 전송 주기 등등이 다르다. 따라서 이러한 정보 들은 각 정보의 특징을 감안하여 토픽으로 매핑을 해야 한다.

이러한 방법은 통하여 전체 시스템을 도메인 단위로 분리할 수 있으며, DDS 통신에서 도메인 단위로 분리된 시스템은 통신이 통신 오버로드가 분산될 수 있으므로 성능상의 이점도 갖게 된다.

데이터 모델 형식으로 정의를 한 후 UML 클래스 다이어그램과 같은 방법을 이용하여 각 토픽의 관계성을 검증할 수 있다. 데이터 모델이 잘 정의되어 있는지 확인하기 위하여 다음과 같은 체크 리스트를 적용할 수 있다.

* + 자료 구조와 그 상호 관계 측면에서 데이터 모델을 표현한다.
  + 이러한 데이터 모델을 통하여 프로그래밍이나 소프트웨어 공학에 대한 지식이 없더라도 문제 영역에 대하여 논의 및 이해가 좀더 쉽게 가능하다.
  + 데이터 모델은 프로그래밍 언어, 플랫폼 등에 독립적인 UML, XML 또는 IDL과 같은 모델링 언어로 표현할 수 있다.
  + 데이터 모델은 위치 투명성을 가져야 한다. 즉, 데이터 모델은 하나의 컴퓨터에 배치될 수도 있으며, 원격지의 어디에도 배치될 수 있어야 한다.
  + 시스템의 시나리오나 행위의 변경은 데이터 모델을 이루는 각 엘리먼트의 상태 변경으로 표현될 수 있어야 한다.

#### 명확한 타입 정의

토픽 정의 시 토픽의 아이템으로 사용되는 데이터 타입은 새로 정의해서 사용하는 것이 유리하다. 가능하면 string 이나 opaque byte array 타입의 사용을 지양해야 한다. 이유는 데이터의 내부 형태가 불분명하기 때문에 미들웨어가 제공하는 기능을 충분히 활용하기가 어렵기 때문이다. 예를 들어 필터링 기능이나 DBMS에 저장 기능 등 이용 시에 제약을 받을 수 있으며, marshalling 및 demarshalling 시 좀 더 복잡한 과정을 거칠 수 있다.

#### 키 필드 사용

가능한 경우에 키를 갖는 토픽을 정의하는 것이 여러 가지 경우에 유리하다. 예를 들어 토픽을 DBMS에 저장하고 조회하는 경우에 키를 가짐으로 인하여 속도를 높일 수 있으며, 어플리케이션끼리 샘플 송수신 시 콘텐츠 필터링 기능을 사용하여 관심 있는 샘플만 수신이 가능하기 때문이다.

### QoS 설정 방법

#### XML 프로파일 사용

DDS 엔티티는 자신만의 QoS 설정이 가능하다. QoS는 소스 코드에서 직접 설정도 가능하지만 가능하면 XML 파일을 이용하여 QoS설정하는 것이 QoS 관련 API를 이용하는 것보다 유연하며, 관리하기가 쉽다. 또한 QoS 값이 실행 파일 외부에 존재하기 때문에 개발이 완료된 후 재컴파일이 없이 어플리케이션 튜닝 등에 쉽게 적용할 수 있기 때문에 매우 유리하다.

일정 규모 이상의 프로젝트에서는 DDS API를 직접 사용하기 보다는 프로젝트 초기에 랩퍼(wrapper) 라이브러리를 만들어서 사용하는 경우가 있다. 이렇게 함으로 인하여 종속성이 줄어드는 단점이 있지만 이 경우에도 주의할 점은 방대한 DDS의 특장점을 잘 이해하고 있어야 한다는 점이다. 특히나 각 QoS와 관련된 파라메타는 매우 다양하기 때문에 QoS 설정과 관련된 API를 설계하는 것은 매우 복잡할 수 있다. 이를 간과하는 경우 개발 중에 이미 방대해진 랩퍼 라이브러리를 변경해야 하는 경우가 발생하며, 이 변경 또한 용이하지 않을 수 있다. QoS 설정은 DDS에 있어서 매우 중요하면서도 샘플 송수신 시 미치는 영향이 매우 크기 때문에 특히나 주의해야 한다.

#### 나) 올바른 Reliable QoS 사용

많은 개발자들이 모든 데이터가 항상 반듯이 전송이 되어야 한다고 생각하지만 반듯이 그렇지만은 아닌 경우도 있다. 예를 들어, 1초 마다 주기적으로 발생하는 센서의 상태 정보는 GPS 위치 정보와 같이 중요하지 않은 정보일 수도 있으며, 센서의 상태 정보는 급박하게 변경되지 않으며, 또한 주기적으로 송신되기 때문에 UDP 프로토콜의 특징에 따라 분실이 되더라도 마지막에 수신된 정보는 유지되기 때문이다. DDS 미들웨어는 Reliable QoS를 사용하는 경우에 DataWriter 내부에 큐를 유지하며 DataReader가 데이터를 수신하지 못하는 경우 재전송을 하게 된다. 이런 과정에서 내부 큐가 꽉 차는 경우에 구현 방법에 따라 블록되는 경우도 있게 된다.

#### 다) 전송 버퍼는 최대 크기 사용

UDP v4의 경우에 헤더를 감안하면 그 크기는 다소 줄어들지만 일반적으로 최대 전송 데이터 크기는 64K 정도이다. 전송 처리율을 최대로 하고자 하는 경우에 가장 먼저 고려해야 할 부분은 바로 이 전송 패킷의 크기를 최대로 늘리는 것이다.

#### 라) 멀티캐스트 사용

송신은 한 곳에서, 수신은 여러 곳에서 하는 경우에 가능하다면 멀티캐스트를 사용하면 데이터 송수신 시 대역폭(bandwidth)을 절약할 수 있을 뿐만 아니라, DataWriter가 한번 만 write 오퍼레이션을 수행하고, 나머지는 스위치가 대신해주므로 CPU 사이클도 절약할 수 있다.

#### 마) IGMP 스누핑(snooping) 사용

많은 네트워크 스위치가 멀티캐스트를 브로드캐스트와 같이 취급하고 있다. 즉, DataWriter가 멀티캐스트 주소를 데이터를 전송하면 스위치는 이 데이터를 모든 어플리케이션으로 브로드캐스트한다. 따라서 스위치의 IGMP 스누핑(snooping) 옵션을 활성화 시켜서 스위치가 정상적으로 멀티캐스트로 동작하도록 하여 네트워크 혼잡 방지를 할 수 있다.

### 어플리케이션 설계 방법

#### DomainParticipant의 수

가능한 적은 domain participant를 생성하는 것이 좋다. DomainParticipant는 내부적으로 다음과 같은 작업을 수행한다.

* + Discovery를 수행한다. 각 어플리케이션은 디스커버리(discovery) 프로토콜을 이용하여 네트워크내의 다른 어플리케이션을 discovery 하게되는데 이때 기준의 DomainParticipant 기준이다.
  + 따라서 발견된 타 DomainParticipant에 대한 내부 정보를 유지하게된다.
  + DomainParticipant 기준으로 쓰레드를 생성하여 데이터의 송수신과 DDS 미들웨어 내부의 이벤트를 처리한다.

따라서 DomainParticipant가 사용될수록 보다 많은 리소스(메모리 및 쓰레드)가 사용된다. DomainParticipant가 생성될 때마다 네트워크 내의 모든 어플리케이션은 서로의 정보에 대하여 이를 디스커버리 해야 함으로 네트워크 대역폭(bandwidth) 또한 소비된다. 따라서 Domain 당 하나의 DomainParticipant 만 생성하는 것이 좋다.

#### DDS 엔티티 생성

DDS entity를 가능한 어플리케이션 실행 초기에 생성한다. DDS 미들웨어는 DDS 엔티티 생성 시마다 디스커버리 작업을 수행하며, 디스커버리 작업 중에 로컬 어플리케이션의 DataReader 또는 DataWriter와 원격지 어플리케이션의 DataReader와 DataWriter를 매칭시키게 된다. 이러한 디스커버리 프로세스가 완료되기 전에는 일반적인 QoS(Reliable QoS 포함하여)인 경우에도 데이터의 전송이 이루어지지 않는다. 즉, 개발자는 일반적으로 DataWriter를 생성하자 마자 정보를 전송하고자 하지만 매칭되는 DataReader 쪽에서의 디스커버리 절차가 완료되지 않는 이상 전송이 이루어지지 않게 된다. 따라서 디스커버리 절차가 일찍 완료될 수 있도록 DDS 엔티티는 가능한 초기에 만드는 것이 좋다.

#### 콜백(Callback) 함수 내에서 블록 되지 않도록 한다

DDS Listener 클래스는 특정 이벤트가 발생할 때마다 호출되는 콜백 함수를 제공하고 있다. DataReader, DataWriter, DomainParticipant, Publisher 또는 Subscriber와 같은 DDS 엔티티는 이러한 Listener 클래스의 기능을 이용하여 다양한 상태 변화에 대응하게된다. 예를 들어, DataReader는 Listener 클래스가 제공하는 콜백 함수를 통하여 수신된 데이터를 사용하거나, 수신이 지연되거나 또는 데이터가 분실 됐음을 알 수 있다.

이런 Listener 클래스의 콜백 함수는 DDS 미들웨어 쓰레드에 의해서 호출되며, DDS 미들웨어 쓰레드는 내부적으로 socket으로부터 데이터를 수신하고, 이를 전송하거나 이벤트가 발생했는지를 확인하는 등의 다양한 작업을 수행한다. 만약 Listener 클래스의 콜백 함수 내에서 블럭이 발생하는 경우 DDS 미들웨어 쓰레드 역시 블럭 되기 때문에 위에 언급한 DDS 미들웨어의 다양한 작업이 방해를 받게 된다. 따라서 Listener 콜백 함수 내에서 sleep 이나 긴 시간이 소요되는 작업을 하지 않는 것이 좋다.

예를 들어, DataReader의 on\_data\_available 콜백 함수는 수신된 토픽 샘플이 내부 버퍼에 존재하는 경우에 DDS 미들웨어에 의해서 호출되는 콜백 함수이다. 일반적으로 이 콜백 함수에서 블럭되는 일이 없도록 하고 또한 다음 수신되는 데이터의 처리를 위하여 가능한 빠르게 리턴하는 것이 좋기 때문에 수신된 데이터를 큐(queue)에 저장하여 다른 사용자 쓰레드에 의해서 처리하도록 하고 즉시 리턴하게 된다.

#### 라) Listener 대신 WaitSet을 사용한다

일반적으로 극단적인 성능이 필요한 경우가 아니면 listener 보다는 WaitSet을 사용하는 것이 보다 안전하다. 이유는 WaitSet의 경우 사용자 쓰레드를 이용하여 특정 이벤트의 발생을 기다리기 때문에 DDS 미들웨어의 쓰레드에 의해서 호출되는 Listener 콜백 함수와 달리 DDS 미들웨어와 독립적으로 실행되기 때문이다.

WaitSet의 단점은 사용자 쓰레드를 이용하기 때문에 DDS 미들웨어 쓰레드로부터 컨텍스트 스위칭(context switching)에 따른 수신 지연이 발생한다는 점 정도이다.

#### 마) OS 튜닝을 이용한 성능 향상

만약 시스템이 높은 처리량의 데이터를 가지고 있다면, 운영체제의 설정을 변경하여 처리량을 높이는 방법도 있다는 것을 알아야 한다. 기본적으로 설정된 Windows는 많은 UDP 데이터 처리에 최적화되어 있지 않다. 따라서 Windows의 레지스티리나 리눅스의 시스템 설정 파라메터를 변경하여 처리량을 높일 수 있다. 예를 들어, 수신 오버로드가 큰 경우에 UDP 어플리케이션은 데이터 수신 시 오류가 발생할 수 있다. 콘솔 박스에서 netstat –s로 통계 데이터를 조회하면 아래와 같은 값들을 볼 수 있다.

|  |
| --- |
| IP Statistics  Packets Received = 687750865  Received Header Errors = 0  Received Address Errors = 1  Datagrams Forwarded = 0  Unknown Protocols Received = 0  Received Packets Discarded = 0  Received Packets Delivered = 574556088  Output Requests = 596445524  Routing Discards = 0  Discarded Output Packets = 0  Output Packet No Route = 0  Reassembly Required = 118445100  Reassembly Successful = 5234431  **Reassembly Failures = 107976783**  Datagrams Successfully Fragmented = 7642664  Datagrams Failing Fragmentation = 0  Fragments Created = 15285321 |

위의 내용을 살펴 보면 IP 헤더 리어셈블리 오류가 매우 큰 값을 나타내고 있음을 볼 수 있다. 이런 오류를 방지하기 위하여 Windows 레지스트리를 변경하여 이런 오류를 개선할 수 있다.

|  |
| --- |
| 1. **시작**을 누른 다음 **실행** 을 클릭하십시오. 2. **열기** 상자에 **regedit** 를 입력한 다음 **확인** 을 누릅니다. 3. 찾은 후 다음 레지스트리 키를 누릅니다.   HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters   1. 오른쪽 창에서 **MaximumReassemblyHeaders** 두 번 클릭하십시오. 2. **값 데이터** 상자에 IP 리어셈블리 헤더의 최대 수를 설정할 숫자를 입력한 다음 **확인** 을 누릅니다. 100-65535 사이의 임의의 정수를 입력할 수 있습니다. 100은 기본값이며 성능 향상을 위하여 가능한 큰 값을 설정합니다. 3. 레지스트리 편집기를 종료하고 시스템을 재부팅합니다. |

리눅스의 경우에는 커널의 설정을 변경할 필요가 있다. 예를 들어, 커널의 소켓 버퍼의 크기는 늘리면 성능 향상에 많은 도움이 된다.

|  |
| --- |
| [root]# /sbin/sysctl –a | grep core  net.core.rmem\_default = 109568  net.core.wmem\_default = 109568  net.core.rmem\_max = 131071 <- receive socket buffer size  net.core.wmem\_max = 131071 <- send socket buffer size |

위와 같이 설정된 값을 아래와 같이 늘려준다.

|  |
| --- |
| # Increase max. send/recv socket buffer limits  # for higher network performance  net.core.rmem\_default = 65536  net.core.rmem\_max = 2097152  net.core.wmem\_default = 65536  net.core.wmem\_max = 1048576 |

아래와 같이 소켓 백로그 값이나 IP 프래그먼트 리어셈블 값을 늘려 줄 수도 있다. 일반적으로 기본값은 약 300 정도이다. 1Gbit 이상의 네트워크 환경에서는 이 값을 늘려주는 것이 좋다. 이 값이 커지면 패킷 분실이 줄어들게 된다.

|  |
| --- |
| # Increase max. recv packets queued at the interface  # for higher network performance  net.core.netdev\_max\_backlog = 30000  # Increase memory size used to reassemble IP fragments  # for higher network performance  net.ipv4.ipfrag\_high\_threshold = 8388608 |

이 외에도 NIC의 MTU 크기를 늘리거나 인터럽트 요청 시간 간격을 늘려서 하나의 인터럽트 시 처리 가능한 패킷의 크기를 늘리는 방법(Interrupt coalescing) 등도 생각해 볼 수 있다.



# 결론

이상과 같이 DIDE를 이용하여 DDS 어플리케이션을 개발하는 절차를 간략하게 살펴보고, 설계 방법론이 적용되는 응용 모델러를 이용하여 DDS 어플리케이션 개발 시 적용 가능한 방법 3 가지에 대하여 언급해 보았다.

DDS 어플리케이션 개발 시 가장 중요한 부분은 역시 토픽을 용도에 맞게 설계하고 각 토픽의 특성에 맞는 QoS를 이용하는 부분이라고 볼 수 있다. 특히나 DDS는 QoS 설정을 통하여 통신 품질 및 성능에 다양한 변화를 줄 수 있다. 또한 OMG에서 제시한 QoS 가 다양하고, 각 DDS 미들웨어가 제공하는 QoS는 벤더에 따라 추가적인 QoS를 제공하는 경우도 많으므로 이에 대한 유의가 필요하며, QoS의 특징을 사전에 충분히 파악하여 설계 및 구현에 적용하는 것이 디버깅이나 유지 보수 시 노력을 절감하는 방법이 될 수 있을 것이다.

# 부록: OMG QoS 요약

|  |  |  |  |
| --- | --- | --- | --- |
| QoS 명칭 | 설명 | RxO[[1]](#footnote-1) | 대상  엔티티[[2]](#footnote-2) |
| **HistoryQosPolicy** | 송신할 데이터의 크기와 저장된 데이터의 관리 방법을 제어할 수 있다. | No | T,R,W |
| **LifespanQosPolicyLifespanQosPolicy** | 어플리케이션이 전송하는 데이터를 얼마 동안 유효한 상태로 유지할 것인지 설정한다. | N/A | T,W |
| **OwnershipQosPolicy** | 복수의 DataWriter가 전송하는 샘플들을 동시에 수신했을 때 DataReader의 수신 방법을 지정할 수 있다. 이 설정이 “shared”인 경우 DataReader는 매칭되는 DataWriter가 수신하는 모든 토픽을 수신하게 된다. 만약 이 설정을 “exclusive”로 변경하는 경우 DataReader는 한번에 하나의 DataWriter가 전송하는 토픽만 수신하게된다. | Yes | T,R,W |
| **OwnershipStrengthQosPolicy** | 복수의 DataWriter가 전송하는 인스턴스의 강도(strength) 값을 설정할 수 있다. **OwnershipQosPolicy** 이 “exclusive”의 경우 이 값이 높은 샘플에 대해서 DataReader는 수신한다. | N/A | W |
| **ReaderDataLifecycleQosPolicy** | 수신된 데이터를 어떻게 관리할 것인지 DataReader 를 설정할 때 사용된다. 일반적으로 DataReader는 데이터를 수신하고 이를 큐에 보관한다. 어플리케이션은 이 큐에 저장된 데이터를 읽게 된다. 이 QoS를 이용하여 큐내의 특정 데이터를 제거할 수 있다. 예를 들어, DataWriter가 더 이상 살아 있지 않은 데이터를 큐에서 자동으로 삭제할 수 있다. | N/A | R |
| **WriterDataLifecycleQosPolicy** | 이 QoS는 키를 갖는 토픽의 인스턴스에 적용된다. DataWriter가 인스턴스의 등록을 해제하는 경우 해당 인스턴스를 모두 제거하는 것이 DataWriter의 기본 동작이다.  토픽의 소유권이 독점(exclusive)적인 경우, DataWriter는 이 토픽의 특정 인스턴스의 소유권을 양도하여 다른 DataWriter가 **OwnershipStrengthQosPolicy** 의 값과는 상관없이 데이터를 업데이트할 수 있도록 한다. | N/A | W |
| **PresentationQosPolicy** | 특정 데이터 셋의 일부가 아니라 전부를 한번에 수신하고자 하는 등을 조절하고자 할 때 사용된다. 예를 들어 키가 있는 데이터 샘플의 경우 동일한 키를 갖는 데이터를 한꺼번에 수신하고자 하는 경우 사용될 수 있다. | Yes | P,S |
| **TransportPriorityQosPolicy** | 일부 트랜스포트 프로토콜의 경우에 운영체제나 스위칭하드웨어에서 사용 가능한 사용자가 설정 가능한 “우선순위” 개념을 가지고 있다. 이런 경우에만 사용 가능한 QoS 이다. | N/A | T,W |
| **UserDataQosPolicy** | 이 QoS를 이용하여 디스커버리중에 부가적으로 보내고자 하는 데이터를 첨부할 수 있다. **UserDataQosPolicy** 는 DataReader/DataWriter 수준에서, **GroupDataQosPolicy** 는 Publishe/Subscriber 수준에서, **TopicDataQosPolicy** 는 토픽 수준에서 디스커버리 메타 데이터를 첨부한다. 이 QoS를 이용하여 보안 인증서를 디스커버리 시 보낼 수 있다. | No | D,R,W |
| **TopicDataQosPolicy** | T |
| **GroupDataQosPolicy** | P,S |
| **LatencyBudgetQosPolicy** | 데이터 전송에 걸리는 지연 시간을 설정한다. 보통의 경우에 0 이 사용된다. 0의 의미는 지연 시간을 최소화 하라는 의미이다. | Yes | T,R,W |
| **DeadlineQosPolicy** | 데이터 샘플간의 최대 시간 간격을 조절할 수 있다. DataReader의 경우 이 시간을 벗어난 샘플에 대해서는 DDS 미들웨어는 사용자 콜백을 호출한다. DataWriter의 경우 이 시간을 넘어서는 데이터는 전송하지 않는다. | Yes | T,R,W |
| **TimeBasedFilterQosPolicy** | 새로운 데이터의 최소 도착 시간을 설정한다. 최소 시간을 벗어다는 데이터는 전송하지 않는다. 이 QoS를 이용하여 너무 많은 데이터가 수신되지 않도록 조절할 수 있다. 예를 들어, GUI에 1초 마다 업데이트만 되는 데이터를 DataWriter가 너무 빠르게 보내는 경우, 이 QoS를 사용하여 전송 빈도를 조절할 수 있다. 동일한 토픽을 수신을 하는 다양한 DataReader에 대해서도 각각 이 QoS를 설정하여 데이터 수신 빈도를 조절할 수 있다. | N/A | R |
| **EntityFactoryQosPolicy** | DDS 엔티티의 행위를 제어할 수 있다. 예를 들어, DataReader가 enabled 상태로 생성되는 경우, DataReader는 생성 즉시 디스커버리가 수행되고, 즉시 데이터를 수신할 수 있게된다. 반대로 disabled 상태로 생성되는 경우 모든 초기화 절차가 완료되기 전에는 데이터 수신을 하지 못하도록 설정할 수 있다. | No | F,D,P,S |
| **PartitionQosPolicy** | 동일한 토픽에 대하여 DataWriter와 DataReader를 매칭하기 위한 string 식별자를 추가할 수 있다. 일반적으로 DataWriter와 DataReader는 토픽을 기준으로 매칭된다. 하지만 이 QoS가 적용되면 동일한 string을 갖는 Publisher와 Subscriber내의 DataWriter와 DataReader가 매칭되게 된다. 따라서 이 QoS를 이용하여 DataWriter와 DataReader의 매칭 범위를 임의로 조절할 수 있다. | No | P,S |
| **ReliabilityQosPolicy** | 네트워크에 의해서 분실된 샘플에 대하여 DDS 미들웨어가 재전송을 할 것인지 설정할 수 있다. 이 설정을 통하여 샘플이 항상 송신 순서대로 수신되도록 할 수 있다. BEST\_EFFORT로 설정되는 경우, DDS 미들웨어는 송수신 보장을 위하여 아무런 조치를 취하지 않으나. 이 설정의 장점은 가장 빠르며, 효율적이다. 주로 센서들이 주기적으로 보내는 상태 메시지에 많이 사용된다. 이 QoS와 더불어 **HistoryQosPolicy** 와 **ResourceLimitsQosPolicy** 사용함으로써 Reliable 수준을 정할 수 있다. 예를 들어, **HistoryQosPolicy** 의 설정이 KEEP\_ALL로 설정되는 경우 DataWriter는 모든 샘플이 수신될 때까지 내부 큐에 보관하며, DataReader는 어플리케이션이 데이터를 읽을 때까지 내부 큐에 보관하게된다. 이러한 내부 큐의 크기는 **ResourceLimitsQosPolicy** 를 이용하여 설정할 수 있다. | Yes | T,R,W |
| **LivelinessQosPolicy** | DataReader와 매칭되는 DataWriter가 네트워크와 단절되거나 삭제될 경우를 검출할 수 있는 방법을 제공한다. | Yes | T,R,W |
| **DurabilityQosPolicy** | 새롭게 또는 나중에 조인되는 DataReader에게 조인 이 전 시점의 데이터를 전송할 것인지를 지정할 수 있다. | Yes | T,R,W |
| **DurabilityServiceQosPolicy** | PERSISTENT 또는 TRANSIENT Durability QoS를 갖는 DataWriter가 외부 저장 서비스를 설정할 수 있다. 대부분의 경우 이 QoS는 **DurabilityQosPolicy** 와 함께 사용된다. | No | T,W |
| **ResourceLimitsQosPolicy** | 데이터 캐슁을 위하여 메모리 할당 크기를 조정할 수 있다. | No | T,R,W |
| **DestinationOrderQosPolicy** | 복수의 DataWriter가 전송하는 동일한 토픽에 대하여 어떻게 다룰 것인지 설정할 수 있다. “by source timestamp” 기준인 경우 DataReader는 DataWriter가 전송한 순서대로 데이터를 수신할 수 있으며, “by reception timestamp” 기준인 경우 DataReader는 수신된 시간 기준으로 데이터를 수신한다. 이는 UDP 프로토콜이 데이터의 도달 순서를 보증하지 않기 때문에 도착 순서를 변경할 때 유용한 기능이다. | Yes | T,R,W |

1. RxO 가 “Yes”인 경우 publishing측과 subscribing 측은 반듯이 QoS가 호환성이 있어야 함을 의미한다. “No”인 경우 QoS 가 publishing측과 subscribing 양측에 설정은 되어야 하나 그 값은 서로 독립적임을 의미한다. “N/A”인 경우 이 QoS 는 publishing측과 subscribing측 중 어느 한 쪽에만 정의되어 됨을 의미하며 호환성과 관계가 없음을 나타낸다. [↑](#footnote-ref-1)
2. F=DomainParticipantFactory, D=DomainParticipant, T=Topic, P=Publisher, W=DataWriter, S=Subscriber, R=DataReader [↑](#footnote-ref-2)