목차

[SRS 2](#_Toc320103264)

[DDS 설정 4](#_Toc320103265)

[BOOST 설정 5](#_Toc320103266)

[test 환경 6](#_Toc320103267)

[테스트 환경 구성 6](#_Toc320103268)

[Test program 설정 6](#_Toc320103269)

[1. TDSR\_Sim 7](#_Toc320103270)

[2. TM 8](#_Toc320103271)

[HA algorithm 9](#_Toc320103272)

[DDS QoS 분석 9](#_Toc320103273)

[SAL build 9](#_Toc320103274)

[Windows XP 9](#_Toc320103275)

[Linux 9](#_Toc320103276)

[VxWorks 9](#_Toc320103277)

[일정 계획 10](#_Toc320103278)

# SRS



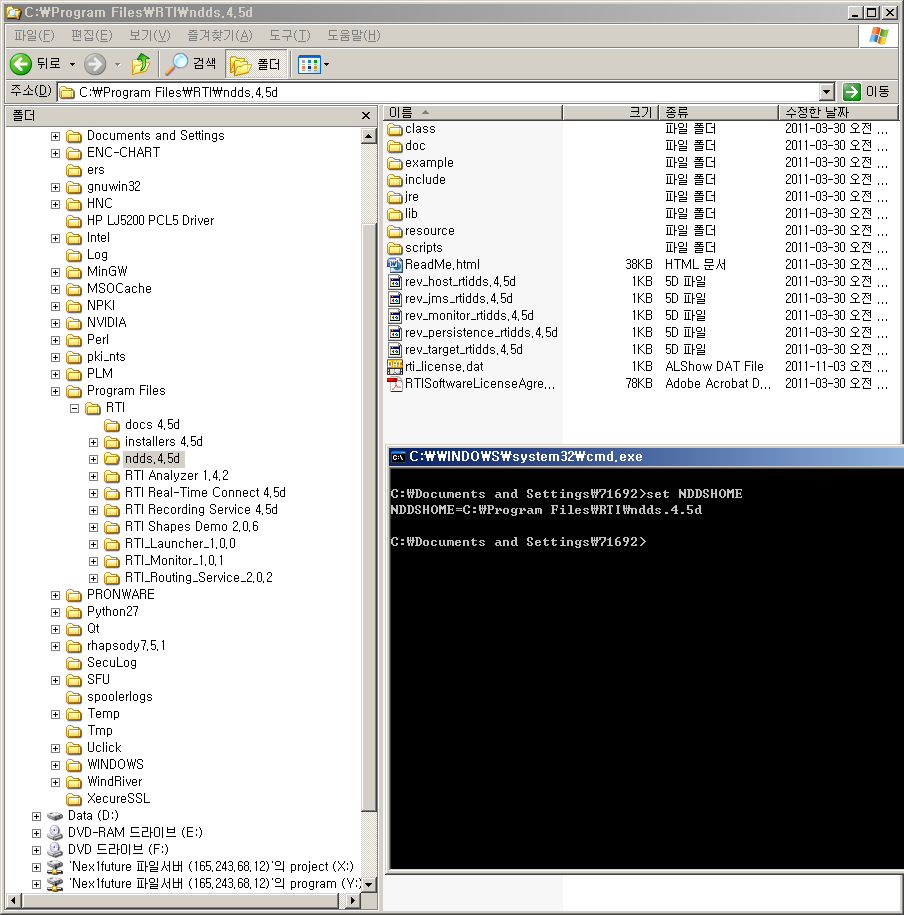
그림 . SW architecture

* OSAL은 shared library 형태로 존재 (제공 But 기능 추가 예상 됨)
* Middleware은 shared library 형태로 존재
* Agent and ~~Master Agent~~는 service(Windows) or daemon(POSIX) 형태로 존재
* DDS library is commercial product.
* ~~관련 기능은 OMG AMSM specification을 준수해야 한다.~~

개발할 SW 모듈은 다음과 같다.

* Middleware:
  + 응용 SW의 상태(active or standby)에 따라 서로 다른 동작을 수행한다. Standby인 경우 data synchronization을 위한 기능을 수행한다.
  + Agent와 heartbeat message를 주고 받는다.
  + Agent의 command에 따라 상태(active or standby)를 변경한다.
  + Configuration 정보를 담고 있는 xml 파일로부터 configuration 정보를 읽어 들인다. XML 관련 library 또한
  + *Agent와 정보는 DDS를 통하여 주고 받는다*
  + HA 라이브러리 사이의 heartbeat/ack 신호는 DDS QoS중 liveliness를 사용하면 DDS의 기능을 활용할 수 있을 듯 하다. 이런 옵션을 가져가는 것도 한 가지 방법일 듯하며, 이 옵션을 사용하는 경우와 직접 구현했을 경우의 성능 비교도 의미가 있어 보인다.
  + 다음과 같은 시나리오에 대해서 동작해야 한다.
    - 두 노드가 동일하게 시작하면서 active/standby 결정 후 정상 동작
    - Active에서 fail이 발생하여 standby가 active로 상태 변경 후 동작
    - Fail이 발생한 active가 복구 되어 standby로 동작하면서 현재 수행중인 active에 대한 standby로서 역할 수행. 이 경우 new standby는 current active와의 상태 동기화가 이루어 져야 한다(checkpoint/recovery).
* ~~Agent:~~
  + ~~응용 SW의 상태 관리~~
    - ~~Middleware와 heartbeat message를 교환하여 응용 SW의 상태를 결정한다.~~
  + ~~응용 SW의 active or standby 상태 결정~~
  + ~~Agent와 (~~*~~Middleware 또는~~*~~) Master Agent와의 통신은 DDS를 이용한다~~
  + *~~CPU usage monitoring~~*
  + *~~SNMP를 통한 network status monitoring~~*
  + Vxworks 를 사용하는 노드는 별도의 storage가 없는 경우가 대부분이다. Vxworks box에 HA publiser가 올라간 경우에는 메시지 동기를 위해서 메시지가 어딘가 저장이 되어야 하는데 이 곳을 agent라고 부르자.
* ~~Master Agent:~~
  + ~~Agent를 통하여 node 응용 SW의 상태 정보를 저장하고, 체계관리 SW에 그 정보를 제공한다.~~
* OSAL
  + 기본적인 source 제공
  + 기능 확장 필요

# DDS 설정



기본적인 설치 디렉터리는 “C:\Program Files\RTI” 임. 이 디렉터리를 기준으로 설명하면 위 그림과 같이 NDDSHOME 환경 변수를 설정해야 함. 주의할 점은 rti\_license.dat 파일의 위치도 위 그림과 같이 설정되어 있어야 함.

RTI DDS는 [http://www.rti.com/downloads/dds.html 에서](http://www.rti.com/downloads/dds.html%20에서%20) 30일짜리 시험판을 다운로드 받아 설치한다.

# BOOST 설정

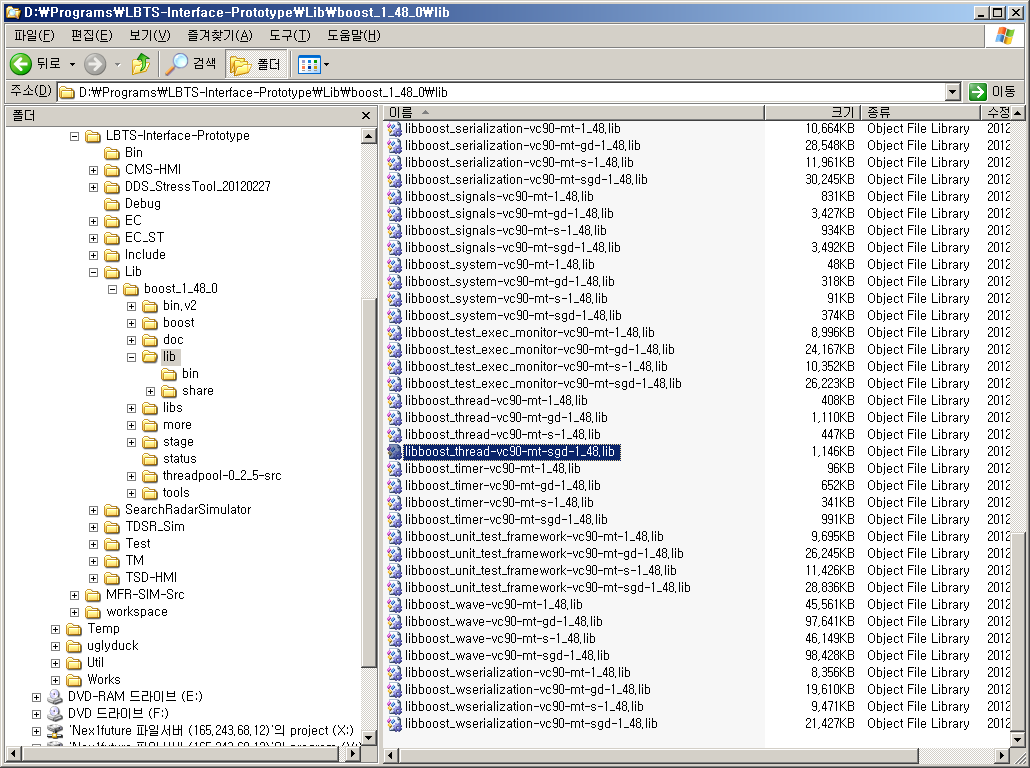
1. <http://www.boost.org/users/history/> 위치에서 1.48.0 버전을 다운로드 받는다.

2. <http://sourceforge.net/projects/threadpool/> 에서 threadpool 소스를 다운 로드 받아 “*압축해제폴더*\boost\_1\_48\_0\threadpool-0\_2\_5-src\threadpool-0\_2\_5-src\threadpool\boost” 폴더의 threadpool.hpp 파일과 threadpool 디렉터리를 “*Boost설치위치*\boost\_1\_48\_0\boost” 로 복사 한다.

3. “*Boost설치위치*\boost\_1\_48\_0” 에서 bootstrap.bat 파일을 실행 시키면 b2.exe, bjam.exe 파일이 생성된고, “*Boost설치위치*\boost\_1\_48\_0\stage\lib” 위치에 일부 컴파일된 바이너리가 생성된다. 하지만 아직 완성된 버전이 컴파일 된 것이 아니기 때문에 추가로 bjam.exe를 실행 시켜야 한다.

4. cmd.exe 에서 “*Boost설치위치*\boost\_1\_48\_0” 폴더로 이동 후

C:>bjam debug release threading=multi runtime-link=static 명령으로 추가 컴파일을 수행한다. 컴파일이 완료 되면 stage\lib 디렉터리를 “*Boost설치위치*\boost\_1\_48\_0” 로 복사한다.



완료되면 위 그림과 같이 구성되어야 한다.

# test 환경

## 테스트 환경 구성



그림 . 테스트 프로그램 구성

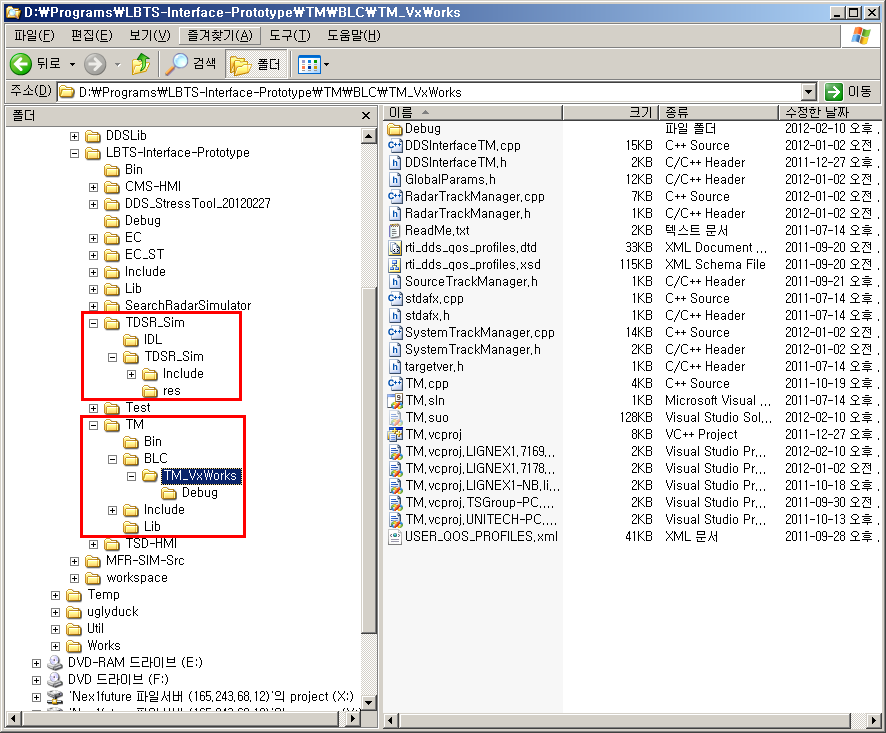
위 그림의 각 프로그램은 동시에 하나의 컴퓨터에서 실행될 수 있거나 각각 독립된 컴퓨터에서 실행될 수 있다.

## Test 시나리오

1. 초기 single node로 운영 중 standby node 운영
   1. 데이터 동기화가 정상적으로 되는지 확인
2. 초기 two node로 운영
   1. Active and standby node가 정상적으로 결정되는지 확인
   2. Active node 비정상 종료 후 재기동
      1. 데이터 동기화 무결성 확인
   3. Standby node 비정상 종료 후 재기동
      1. 데이터 동기화 무결성 확인

## Test program 설정

HA 라이브러리를 테스트하기 위한 환경은 다음과 같이 구성된다. Include와 Lib에서 NDDSHOME 및 boost 관련 설정만 기존 설정에서 변경해 주면 된다.



### 1. TDSR\_Sim

1. 컴파일 환경 설정

기존에 설정된 include 및 link 관련된 설정만 변경해 주면 됨

1. 사용법

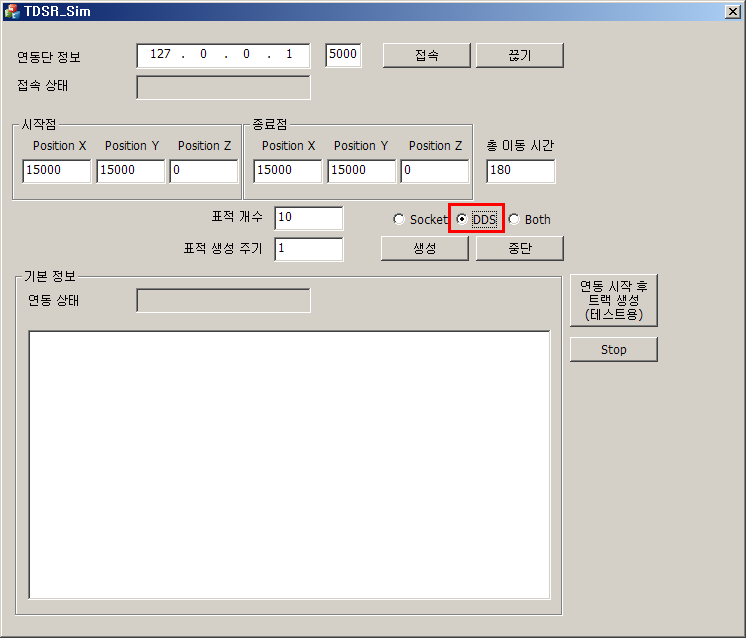


그림 . TDSR GUI

그림과 같이DDS 부분을 체크한 후 그 아래 생성 버튼을 누르면 시작점 ~ 끝점 사이의 좌표 (source track)을 생성하여 DDS로 전송(publish)을 하게됨. 여기서 전송되는 source track은 TM program에서 수신을 하게됨.

### 2. TM

1. 컴파일 환경 설정

기존에 설정된 include 및 link 관련된 설정만 변경해 주면 됨

1. 사용법

Console 프로그램으로 특별한 GUI가 없음. TDSR로부터 DDS를 통하여source track을 받아 system track 및 몇 가지 메시지를 생성한 후 전송(publish) 하게 됨. 보통은 CMSHMI에서 이 결과를 받아 사용하게 됨.

# HA algorithm

## DDS QoS 분석

* DDS의 HA는 ownership, ownership strength, liveliness, deadline QoS를 통하여 이루어 진다고 한다.
* 모든 전송 메시지를 저장할 수 없으므로 (DDS는 History QoS를 통하여 전체 or 마지막 일부를 저장한다) transaction 개념이 있어야 한다. 예를 들어, 레이더는 source track을 주기적으로 보내므로 매번 트랙을 다 가지고 있을 필요는 없을 것이다.
* DDS는 Liveliness QoS를 통하여 publisher의 fault를 detect 할 수 있다. Heartbeat message 를 DDS에서 자동으로 보내고, 그 주기 또한 조절할 수 있다. 이는 HA의 기본 기능이다.
* Ownership strength QoS를 조절하여 publisher를 active와 standby로 유지할 수 있다. 하지만 active가 장애복구가 되어 ownership strength value가 낮은 값으로 동작할 때(standby로 동작) active가 다른 subscriber and publisher와 주고 받은 메시지의 상태로 자동으로 데이터의 synch가 되는지는 확신할 수 없다.

# SAL build

## Windows XP

.Sin solution 파일 작성, 빌드 및 샘플 프로그램 테스트

## Linux

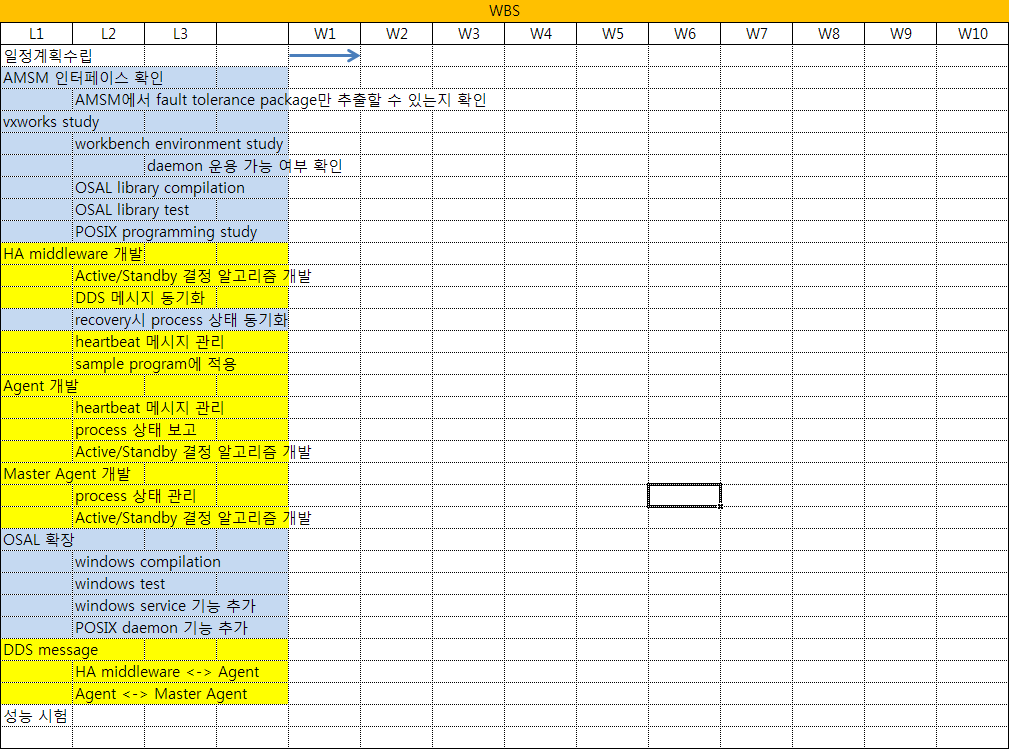
$ make –f makefile.linux 로 빌드

## VxWorks

Workbench에서 일부 소스 수정 및 빌드

소스의 구성은 makefile.linux와 같음. POSIX를 동일하게 사용하므로

# 일정 계획



노란색: HS CHOI

푸른색: YH KIM

※별도의 엑셀 문서 참조

참고 자료

1. expat xml parser library 소개 자료. <http://www.nicklib.com/bbs/board.php?bo_table=bbs_opensrc&wr_id=8>
2. Expat library는 source forge (<http://expat.sourceforge.net/>) 에 open source로 공개되어있음
3. DDS library는 <http://www.rti.com/downloads/dds.html> 30-day trial version을 활용한다.