

USB Host DRIVER

구조설계서

2022-01-14

김성후

이 문서는 **USB Host Driver** 개발을 위한 구조설계서이다.

REVISION HISTORY

Version	Date	Author	Description
0.1	2022-01-14	김성후	초기 문서 생성
0.2	2022-01-16	김성후	Interim 작성
0.3	2022-01-20	김성후	Interim 1차 수정
0.4	2022-01-22	김성후	Pre-final 작성
0.4	2022-02-07	김성후	Final 작성

1. 시스템 개요	3
2. 요구사항	9
2.1. 기능적 요구사항	9
2.2. 비기능적 요구사항	16
2.3. 품질 속성	18
3. 시스템 구조	22
4. 모듈 사양	322
부록	54
A. 도메인 모델	55
B. 품질 시나리오	65
C. 품질 시나리오 분석	67
D. 후보 구조	70
E. 후보 구조 평가	102
F. 최종 구조 설계	113

1. 시스템 개요

1.1 시스템 개요

USB란 Universal Serial Bus의 약어로, CTI(Computer Telephony Integration) 산업의 성장을 위해 인텔, 마이크로소프트, COMPACT, IBM, NEC, DEC 및 Nortel 등의 회사가 개발하였다. 플러그 앤 플레이를 위한 PC 주변장치의 Bus 규격으로서, 새로운 주변기기가 접속되었을 때 재부팅이나 셋업 과정 없이 자동인식으로 최대 127개의 장치를 연결할 수 있으며, 기존 흔히 사용되던 시리얼 포트에 비해 데이터 전송속도도 빠르게 향상되었다. 규격이 마련됨과 동시에 인텔과 VIA, SiS, ALi 등과 같은 메인보드용 칩셋 제작사들은 USB 포트를 지원하는 칩셋을 발표하고 그들을 주축으로 지금처럼 PC에 USB 포트가 기본적으로 장착되기 시작됐다.

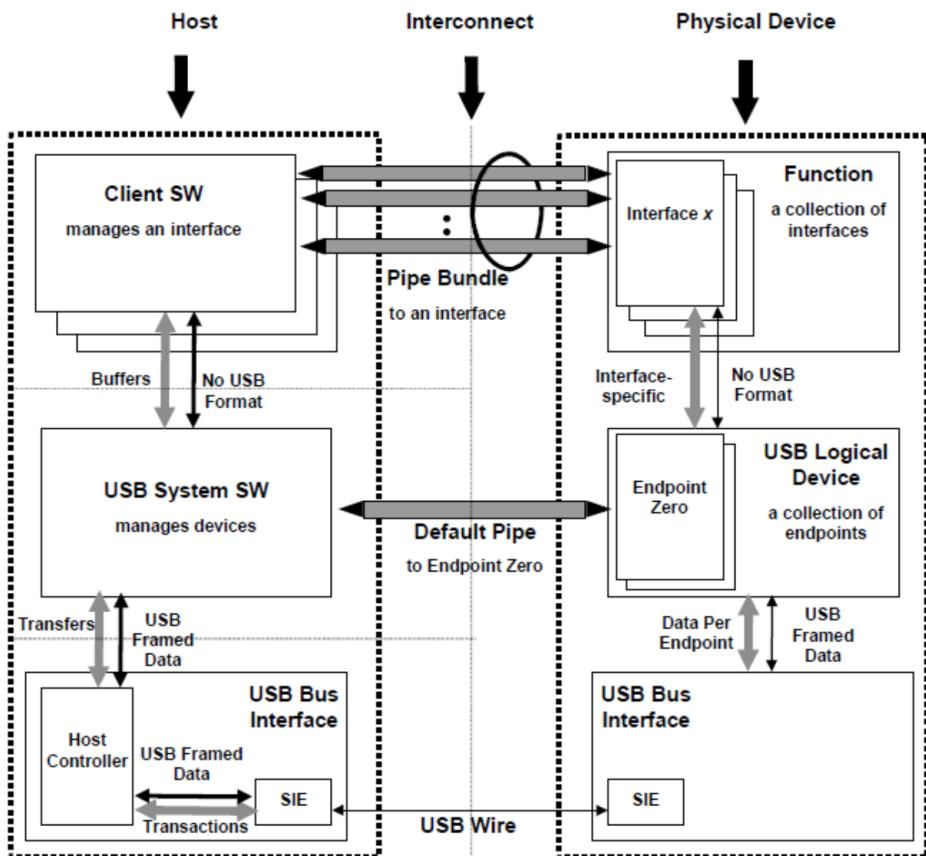


Figure 1 - USB 시스템의 구조

USB는 하나의 Host와 다수의 USB Device를 연결한다. 이들은 물리적인 USB Wire를 통해 연결되어 서로 통신하며 그 위에 논리적인 계층을 가진다. 우선 물리적 계층의 USB Bus Interface의

Host Controller는 통신을 위한 하드웨어 적인 요소를 담당한다. 그리고 그 위의 레이어의 USB System SW는 USB 장치들을 관리하기 위한 Logic을 담당하며 Peer 개념의 USB Logical Driver와 논리적인 연결을 가지고 있다. USB Logical driver의 통신을 위한 접점을 Endpoint Zero라고 부르며, 논리적 개념의 연결을 Default Pipe라고 명명하고 있다. 또 그 상위 레이어는 Client Software 와 USB Device의 Function과의 논리적인 연결을 가지게 된다. Client Software는 Host의 운영체제 내의 소프트웨어 혹은 Host에 탑재된 소프트웨어를 의미하며, USB Device의 Function은 해당 장치의 고유한 기능을 구현한 소프트웨어를 의미한다. 각각의 논리적 연결은 실제로는 물리적 계층의 USB Wire를 통해 전기적인 신호로 전달되지만 논리적 개념의 추상화를 통해 복잡도를 낮춰주는 역할을 한다.

1.2 사업 환경

USB Device가 PC 업계의 인터페이스 표준으로 확고한 지위를 가지면서, 매해 수십억 개의 새로운 USB Device들이 시장에 출시되고 있다. 그에 따라 USB 이전 시대의 보편적인 연결 방식이었던 시리얼 포트에 대한 PC 제품들의 지원이 점차 감소하고 있으며, PC에서는 다수의 USB Port를 제공하고 있다. 또한 코로나 19의 영향으로 한동안 스마트폰과 태블릿 PC에 밀려 수요가 감소하던 Note PC에 대한 수요는 재택근무와 가정 학습의 증가에 따라 2021년 최고 성장률을 기록하였다. 이러한 환경하에서 본 과제에서는 Note PC에 연결하여 5개의 USB 연결을 지원하는 USB Host Driver를 설계하려 한다.



Figure 2– Note PC 제품의 USB Port

1.2.1 Stakeholder

Stakeholder	Interest
Note PC 제조사	부품사로부터 USB Host Device를 납품 받아 자사의 Note PC에 탑재하려 한다. 얇은 제품을 지향하기 때문에 납품 받는 제품이 최소한의 하드웨어로 충분한 성능을 내기를 기대한다.
Operation System 개발사	불특정 다수의 USB Device에 대해서 최대한 정보를 파악할 수 있길 바라며, 그에 따른 최적의 데이터 전송을 기대한다.
USB Device 개발사	개발한 장치가 타겟 운영체제에서 개발의도대로 동작하기를 기대 한다. 그러기 위해 제품의 특성에 맞는 최대한의 리소스(전력, 대역폭 등)를 할당 받기를 기대한다.
LED Controller 개발사	다양한 제품에 범용적으로 사용될 수 있는 LED 제품을 개발하고 납품하고 싶어한다.
USB Device 사용자	가능한 많은 장치를 연결하여 사용하고 싶어한다. 합리적인 가격에 안정적인 속도와 충전 기능을 제공받고 싶어한다.

1.2.2 시스템 제약사항

- 제품은 5개의 USB 슬롯을 제공한다.
- 제품은 외부에 LED가 5개 존재하며 연결된 5개의 USB Device를 표현한다. 파란색 불빛을 낼 수 있으며 점멸할 수 있고 점멸 속도도 조절이 가능하다.
- 제품은 Host PC로부터 동작에 충분한 양의 전력을 공급받아 제품 자체의 전력은 고려사항이 아니다.
- 시스템은 Single Core 환경에서 동작한다.
- USB Device Driver를 가진 운영체제가 있는 PC 환경을 Host로 가정한다.
- USB의 물리적 통신 프로토콜은 USB Host Controller가 담당한다.
- USB Host Controller는 CRC(Cyclic Redundancy Check) 및 재전송 등을 통해 전송 데이터에 대한 무결성을 보장한다고 가정한다.
- 127 개의 USB Device 연결이 가능하며, 물리적으로 30미터 이내에 연결되어 있다.

1.3 시스템 정의

본 과제에서 개발하는 USB Hub System의 개요 및 개발 범위는 아래 Figure 3에서 빨간색 점선으로 표시한 부분과 같다.

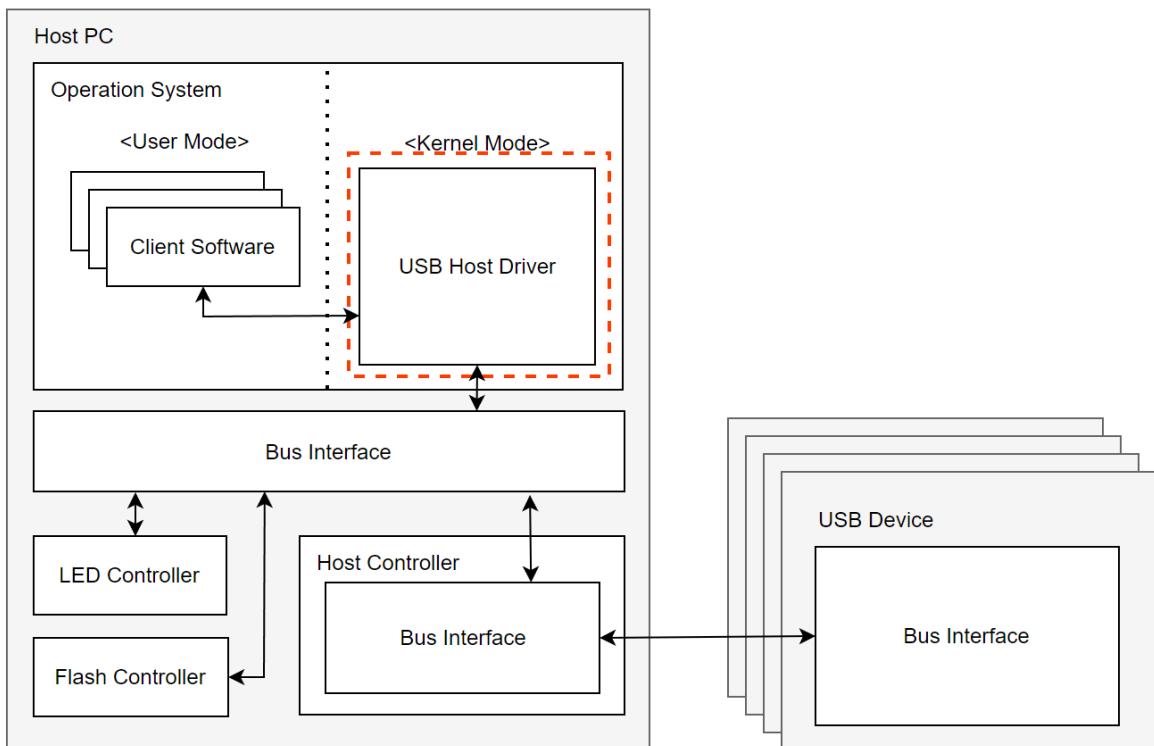


Figure 3 - 시스템 전체 구조와 개발 범위

본 과제의 설계 범위는 USB Host Driver까지이다. Host PC 내에는 USB Host System과 LED, Flash 등의 하드웨어가 존재하며 그를 제어하기 위한 각각의 Controller들이 있다. 각각의 Controller들은 Bus Interface를 통해 Operation System과 물리적 통신을 할 수 있으며 소프트웨어 적인 처리는 Device Driver가 담당을 한다. Device Driver는 System Call을 통해 Client Software와 통신한다. 본 과제에서는 USB Host Driver가 Client Software와 System Call을 통해 통신하고 Host Controller에 USB Wire로 연결된 USB Device와 물리적으로 통신하며 USB Device를 관리하고 데이터를 전송하는 기능을 설계한다.

1.3.1 External System

External System	Interface
Operation System	본 시스템은 Operation System의 System Call을 통해 Client Software와 통신한다. 또한 본 시스템을 통해 Operation System은 USB Device와 제어 정보를 전달하고 데이터를 주고받을 수 있다.
USB Device	USB Host Driver는 USB Device와 Host Controller를 통해 통신한다. USB Device는 본 시스템을 통해 Client Software의 요청에 응답한다.
LED	LED 장치는 Third-party 형태로 USB Host System에 제공되는 장치이다. USB Host Driver는 Bus Interface를 통해 인터럽트로 제어 신호를 주고받는다.

1.3.2 시스템 경계

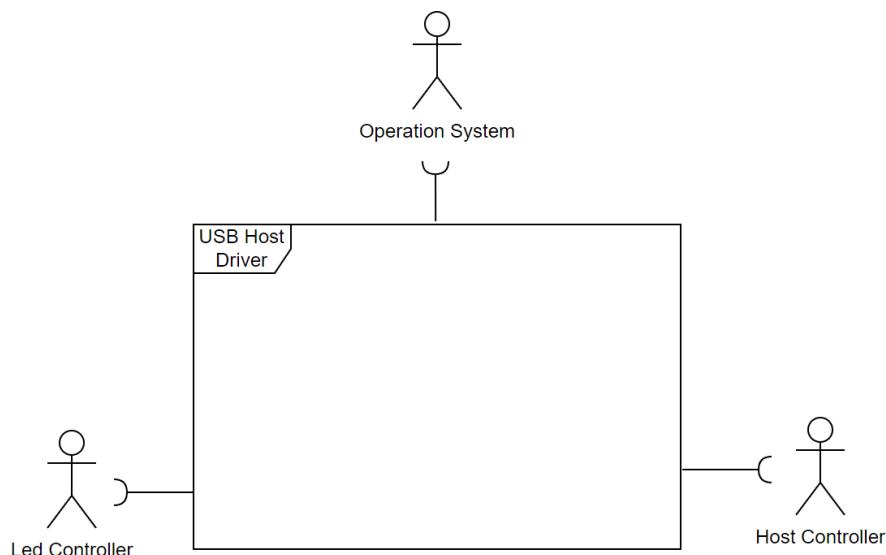


Figure 4 - 시스템 경계

본 과제에서 다루고자 하는 USB Host Driver의 경계는 아래 Figure 4와 같다. 본 시스템은 Operation System의 요청에 따라 Host Controller와 통신하며 USB Device에 대한 제어 및 데이터 전송을 수행할 수 있다. 또한 LED 하드웨어에 대한 제어를 Led Controller에게 요청할 수 있다.

1.3.3 시스템 동작

- USB Host System을 시작하고 종료하는 절차를 수행한다.
- USB Device 연결 및 해제에 대한 처리를 수행한다.
- USB Device에 대한 정보를 수집하고 요청에 따라 전달한다.
- USB Device에 대한 읽기 및 쓰기를 수행한다.

2. 요구사항

2.1. 기능적 요구사항

2.1.1 Use Case Diagram

시스템 경계에 따른 Use Case Diagram은 아래 Figure 5와 같다.

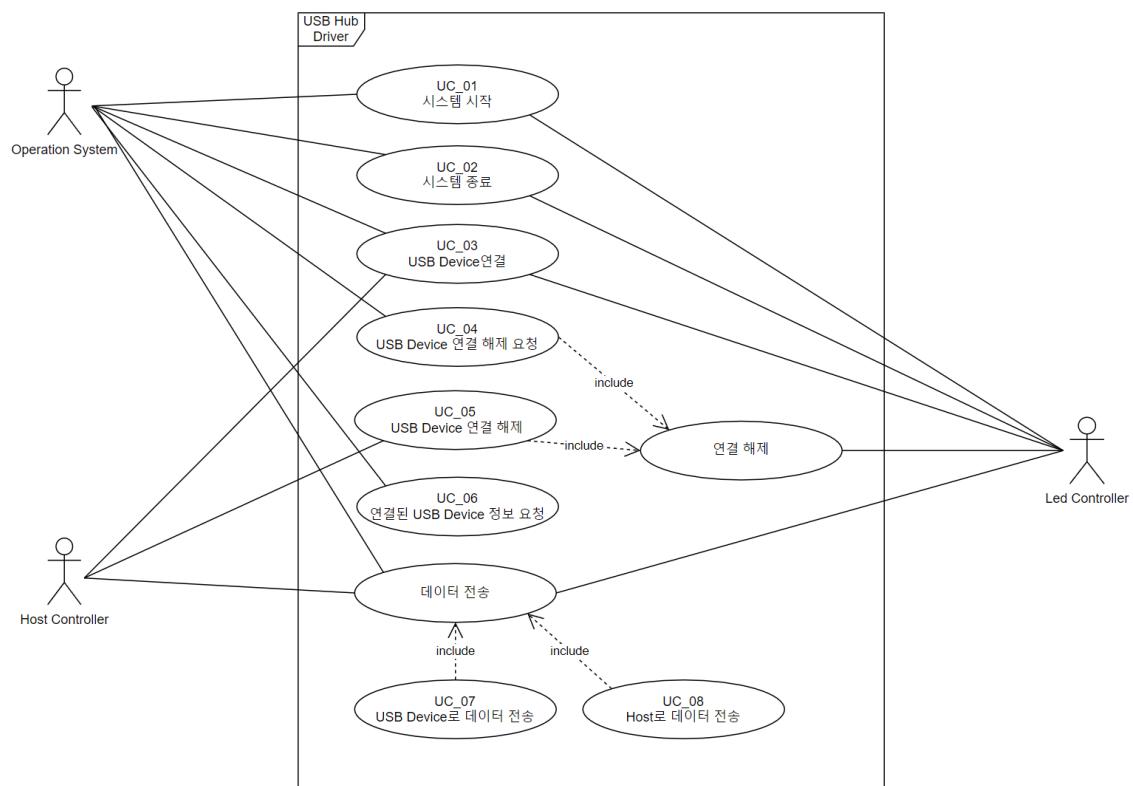


Figure 5 – Use Case Diagram

2.1.2 Use Case 명세

2.1.2.1 시스템 시작

UC_01	시스템 시작
설명	시스템은 Operation System으로부터 시스템 시작을 통보받으면 각종 리소스들을 초기화하고 가용 자원에 대한 모니터링을 시작한다.
행위자	Operation System
선행조건	Host PC로부터 시스템에 전력이 공급
후행조건	시스템은 INITIATED 상태
기본 동작	<ol style="list-style-type: none"> 1. Operation System은 시스템에게 System Call을 통해 시스템이 시작했음을 통보한다. 2. 시스템은 Message Pipe를 구성한다. 3. 시스템은 Operation System에게 수용 가능한 USB Class 목록을 요청한다. 4. Operation System은 수용 가능한 USB Class 목록을 시스템에게 전달한다. 5. 시스템은 수용 가능한 USB Class를 저장한다. 6. 시스템은 LED Interface에게 3초간 점멸하도록 명령한다. 7. 시스템은 10ms마다 Host Controller에게 가용 자원 정보를 요청한다. 8. Host Controller는 요청이 올 때마다 시스템에게 가용 자원 정보를 전달한다. 9. 시스템은 가용 자원 정보를 저장한다.
추가 동작	-

2.1.2.2 시스템 종료

UC_02	시스템 종료
설명	시스템은 Host와 연결이 해제되면 일정시간 재연결을 기다린다. 재연결이 되지 않으면 연결된 USB Device와 연결을 해제하고 LED 점등을 해제한다. 또한 시스템은 가용 자원 수집을 중단한다.
행위자	Operation System
선행조건	시스템은 INITIATED 상태

후행조건	시스템은 HALTED 상태
기본 동작	<ol style="list-style-type: none"> 1. Operation System 은 시스템에게 System Call 을 통해 연결 해제를 통보한다. 2. 시스템은 모든 연결된 USB Device 들에게 연결 해제를 통보하고 해당 USB Device 와 Mapping 된 Stream Pipe 를 제거한다. 3. 시스템은 Message Pipe 를 제거한다. 4. 시스템은 Led Interface 에게 모든 LED 점등 해제를 명령한다. 5. 시스템은 사용 자원 정보 수집을 중단한다.
추가 동작	-

2.1.2.3 USB Device 연결

UC_03	USB Device 연결
설명	시스템은 USB Device와 연결되면 USB Specification에 정의된 열거 절차를 진행 한다. 시스템은 USB Device가 요구한 리소스에 대해서 수용가능한만큼의 리소스를 제공한다. 또한 USB Device가 연결된 위치의 LED를 점등하고 Operation System에게 연결을 통보한다.
행위자	Host Controller
선행조건	시스템은 INITIATED 상태 USB 슬롯 혹은 연결된 USB Hub에 USB Device가 물리적으로 연결
후행조건	-
기본 동작	<ol style="list-style-type: none"> 1. Host Controller 는 시스템에게 USB Device 의 연결을 통보한다. 2. 시스템은 Host Controller 에게 새로 연결된 USB Device 에 최소 전력(100mA)를 공급을 명령한다. 3. Host Controller 는 시스템에게 전력의 공급을 통보한다. 4. 시스템은 Host Controller 에게 USB Device 와 시스템간 통신할 수 있는 주소를 할당해서 전달한다. 5. Host Controller 는 시스템에게 새로 연결된 USB Device 의 VID, PID 를 전달한다. 6. 시스템은 새로 연결된 USB Device 가 이전 연결된 적 있는지 확인한다. <ol style="list-style-type: none"> 6-1. 연결된 적 없는 장치라면 Host Controller 에게 USB Device descriptor 를 요청한다. 6-2. Host Controller 는 시스템에게 새로 연결된 USB Device 의 USB Device descriptor 를 전달한다.

	<p>6-3. 시스템은 USB Device descriptor에서 USB Class를 확인하고 수용할 수 있는지 확인한다.</p> <p>7. 시스템은 USB Device의 요구 리소스에 대한 가용 자원이 충분한지 확인한다. 충분하지 않다면 시스템은 요구 사항과 가용 자원에 대한 협상된 값을 도출한다.</p> <p>8. 시스템은 Host Controller에게 7 번에서 계산된 자원만큼의 전력을 새로 연결된 USB Device에 공급하도록 명령한다.</p> <p>9. 시스템은 LED Controller에게 USB Device가 연결된 위치의 LED를 점등하도록 명령한다.</p> <p>10. 시스템은 Host와 USB Device가 통신할 수 있는 Stream Pipe를 개설한다.</p> <p>11. 시스템은 USB Device가 연결된 적 없던 장치라면 Flash Controller에게 Known Device로 저장을 요청한다.</p> <p>12. 시스템은 Operation System에게 새로운 Stream Pipe가 개설되었음을 통보한다.</p>
추가 동작	<p>1. (6-3에서) 수용할 수 없는 장치라면 Operation System에게 수용할 수 없는 USB Device가 연결되었음을 통보한다.</p> <p>2. (9에서) USB Device가 루트 허브에 연결된 장치가 아니라면 LED 점등 절차는 생략한다.</p>

2.1.2.4 USB Device 연결 해제 요청

UC_04	USB Device 연결 해제 요청
설명	Host에서 특정 USB Device의 연결을 해제하는 절차이다. 물리적으로 USB 슬롯에서 USB가 해제되지는 않지만 논리적으로는 연결을 해제한다. Windows 운영 체제의 “안전하게 USB 장치제거”에 해당하는 기능이다.
행위자	Operation System
선행조건	시스템은 INITIATED 상태 USB Device가 시스템에 연결된 상태
후행조건	USB Device는 USB 슬롯에 물리적으로만 연결된 상태
기본 동작	<ol style="list-style-type: none"> Operation System은 시스템에게 System Call을 통해 특정 USB Device 연결 해제 요청을 한다. 시스템은 Host Controller에게 특정 USB Device 와의 연결 해제를 요청한다.

	<ol style="list-style-type: none"> 3. 시스템은 특정 USB Device 와 Mapping 된 Stream Pipe 를 해제한다. 4. 시스템은 LED Interface 에게 특정 USB Device 와의 매플 해제와 점등 해제를 명령한다. 5. 시스템은 특정 USB Device 의 컨텍스트 정보를 삭제한다.
추가 동작	<ol style="list-style-type: none"> 1. (2에서) Graceful 연결 해제가 요청되는 경우, Host Controller로부터 연결 해제 통보를 받을 때까지 3을 진행하지 않는다. 2. (4에서) USB Device가 루트 허브에 연결된 장치가 아니라면 LED 점등 해제 절차는 생략한다.

2.1.2.5 USB Device 연결 해제

UC_05	USB Device 연결 해제
설명	물리적으로 특정 USB Device가 USB 슬롯에서 분리되었을 때 발생하는 Use Case이다. Host에게 연결 해제를 통보하고 Stream Pipe 등 할당된 자원과 저장된 자료를 삭제한다.
행위자	Host Controller
선행조건	시스템은 INITIATED 상태 USB Device가 시스템에 물리적으로 연결된 상태
후행조건	-
기본 동작	<ol style="list-style-type: none"> 1. Host Controller 가 시스템에게 특정 USB Device 의 연결 해제를 통보한다. 2. 시스템은 Operation System 에게 특정 USB Device 의 연결 해제를 통보한다. 3. Operation System 은 시스템의 Stream Pipe 에 남은 데이터를 Read 한 뒤, 시스템에게 디바이스 해제 요청을 한다. 4. 시스템은 특정 USB Device 와 Mapping 된 Stream Pipe 를 해제한다. 5. 시스템은 LED Interface 에게 특정 USB Device 와의 매플 해제와 점등 해제를 명령한다. 6. 시스템은 특정 USB Device 의 컨텍스트 정보를 삭제한다.
추가 동작	<ol style="list-style-type: none"> 1. (5에서) USB Device가 루트 허브에 연결된 장치가 아니라면 LED 점등 해제 절차는 생략한다.

2.1.2.6 연결된 USB Device 정보 요청

UC_06	연결된 USB Device 정보 요청
설명	Host는 연결된 USB Device에 대한 정보를 요청한다. 시스템은 USB Device와 연결될 때 생성한 컨텍스트를 이용하여 Host Controller를 거치지 않고 정보를 모은 뒤 Operation System에게 전달한다.
행위자	Operation System
선행조건	시스템은 INITIATED 상태
후행조건	-
기본 동작	<ol style="list-style-type: none"> 1. Operation System 은 시스템에게 연결된 USB Device 들의 정보를 요청한다. 2. 시스템은 모든 연결된 USB Device 정보를 수집한다. 3. 시스템은 Operation System 에게 수집한 정보를 전달한다.
추가 동작	1. (2에서) 연결된 USB Device가 없는 경우 바로 3번으로 넘어간다.

2.1.2.7 Host로 데이터 전송

UC_07	Host로 데이터 전송
설명	특정 USB Device로부터 데이터를 읽는 Use Case이다.
행위자	Operation System
선행조건	시스템은 INITIATED 상태 한 개 이상의 USB Device가 시스템에 연결된 상태
후행조건	-
기본 동작	<ol style="list-style-type: none"> 1. Operation System 이 시스템에게 System Call 을 통해 특정 USB Device 의 데이터 읽기를 요청한다. 2. 시스템은 USB Device 를 통해 받은 Transaction 들을 묶어 IRP 형태로 응답한다. 3. 시스템은 LED Interface 에게 점멸 해제를 명령한다. 4. 시스템은 내부 USB Device 에 대한 상태를 "SUSPEND"로 변경한다.
추가 동작	-

2.1.2.8 USB Device로 데이터 전송

UC_08	USB Device로 데이터 전송
설명	특정 USB Device에게 데이터를 쓰는 Use Case이다.
행위자	Operation System
선행조건	시스템은 INITIATED 상태 한 개 이상의 USB Device가 시스템에 연결된 상태
후행조건	-
기본 동작	<ol style="list-style-type: none"> 1. Operation System은 시스템에게 System Call을 통해 IRP를 전송한다. 2. 시스템은 내부 USB Device에 대한 상태를 "WRITING"으로 변경한다. 3. 시스템은 Host Controller에게 받은 IRP에 대한 Transaction을 전송한다. <ol style="list-style-type: none"> 3-1. Host Controller는 전송에 대한 Data Rate를 전달한다. 3-3. 시스템은 LED Interface에게 Data Rate를 전달해서 값에 맞추어 점멸할 수 있도록 명령한다. 이 때 USB Device가 USB Hub에 연결된 상태라면 USB Hub에 연결된 조상 장치에 해당하는 LED를 점멸한다. 4. 시스템은 LED Interface에게 점멸 해제를 명령한다. 5. 시스템은 내부 USB Device에 대한 상태를 "SUSPEND"으로 변경한다.
추가 동작	-

2.2. 비기능적 요구사항

2.2.1 NFR_01 디바이스 설정 시간

NFR_01 (QS_04)	성능	디바이스 설정 시간
설명		USB Device가 삽입되었을 때, 500ms이내에 설정을 완료하여, 사용자가 빠른 시간내에 해당 디바이스를 사용할 수 있어야 한다.
환경		시스템이 정상 동작한다.
자극		USB Device 를 USB 슬롯에 삽입한다.
반응		<p>1. Host Controller 는 시스템에게 USB Device 의 연결을 통보한다.</p> <p>2. 시스템은 Host Controller 에게 새로 연결된 USB Device 에 최소 전력(100mA)를 공급을 명령한다.</p> <p>3. Host Controller 는 시스템에게 전력의 공급을 통보한다.</p> <p>4. 시스템은 Host Controller 에게 USB Device 와 시스템간 통신할 수 있는 주소를 할당해서 전달한다.</p> <p>5. Host Controller 는 시스템에게 새로 연결된 USB Device 의 VID, PID 를 전달한다.</p> <p>6. 시스템은 새로 연결된 USB Device 가 이전 연결된 적 있는지 확인한다.</p> <p>6-1. 연결된 적 없는 장치라면 Host Controller 에게 USB Device descriptor 를 요청한다.</p> <p>6-2. Host Controller 는 시스템에게 새로 연결된 USB Device 의 USB Device descriptor 를 전달한다.</p> <p>6-3. 시스템은 USB Device descriptor 에서 USB Class 를 확인하고 수용할 수 있는지 확인한다.</p> <p>7. 시스템은 USB Device 의 요구 리소스에 대한 가용 자원이 충분한지 확인한다. 충분하지 않다면 시스템은 요구 사항과 가용 자원에 대한 협상된 값을 도출한다.</p> <p>8. 시스템은 Host Controller 에게 7 번에서 계산된 자원만큼의 전력을 새로 연결된 USB Device 에 공급하도록 명령한다.</p> <p>9. 시스템은 LED Controller 에게 USB Device 가 연결된 위치의 LED 를 점등하도록 명령한다.</p> <p>10. 시스템은 Host 와 USB Device 가 통신할 수 있는 Stream Pipe 를 개설한다.</p> <p>11. 시스템은 USB Device 가 연결된 적 없던 장치라면 Flash Controller 에게 Known Device 로 저장을 요청한다.</p>

	시스템은 Operation System 에게 새로운 Stream Pipe 가 개설되었음을 통보한다.
측정	[디바이스 설정 시간] = [Pipe가 개설되는 시각] – [USB Device가 물리적으로 삽입되는 시각]
제약	[디바이스 설정 시간] <= 500ms

2.2.1 NFR_02 시스템 부팅 시간

NFR_02 (QS_01)	성능	시스템 부팅 시간
설명		시스템 부팅이 100ms 이내에 이루어져 운영체제의 전체 부팅 속도에 악영향을 주지 않도록 해야한다.
환경		시스템의 전원이 Off 상태이다.
자극		시스템에 전원을 인가한다.
반응		<ol style="list-style-type: none"> 1. Operation System 은 시스템에게 System Call 을 통해 시스템이 시작했음을 통보한다. 2. 시스템은 Message Pipe 를 구성한다. 3. 시스템은 Operation System 에게 수용 가능한 USB Class 목록을 요청한다. 4. Operation System 은 수용 가능한 USB Class 목록을 시스템에게 전달한다. 5. 시스템은 수용 가능한 USB Class 를 저장한다. 6. 시스템은 LED Interface 에게 3 초간 점멸하도록 명령한다. 7. 시스템은 10ms 마다 Host Controller 에게 가용 자원 정보를 요청한다. 8. Host Controller 는 요청이 올 때마다 시스템에게 가용 자원 정보를 전달한다. <p>시스템은 가용 자원 정보를 저장한다.</p>
측정		[부팅 시간] = [LED 장치의 최초 점멸 시각] – [시스템에 전원이 공급되는 시각]
제약		[부팅 시간] <= 100ms

2.3. 품질 속성

2.3.1 QA_01 데이터 전송 속도

QA_01 (QS_02)	성능	데이터 전송 속도
설명	데이터 전송은 빠를수록 좋다.	
환경	USB Device가 USB 슬롯에 삽입되어 있다.	
자극	USB Device 가 데이터를 전송한다.	
반응		<ol style="list-style-type: none"> Operation System 이 시스템에게 System Call 을 통해 특정 USB Device 의 데이터 읽기를 요청한다. 시스템은 USB Device 를 통해 받은 Transaction 들을 묶어 IRP 형태로 응답한다. 시스템은 LED Interface 에게 점멸 해제를 명령한다. 시스템은 내부 USB Device 에 대한 상태를 "SUSPEND"로 변경한다.
측정	[전송 속도] = [Destination으로 전송된 Bit] / 1000ms	

2.3.2 QA_02 지원가능한 USB Class 확장

QA_02 (QS_09)	변경용이성	지원가능한 USB Class 확장
설명	지원 가능한 USB Class를 추가하는데 비용이 적어야 한다.	
환경	시스템의 전원이 Off 상태이다.	
자극	시스템에 전원을 인가한다.	
반응		<ol style="list-style-type: none"> Operation System 은 시스템에게 System Call 을 통해 시스템이 시작했음을 통보한다. 시스템은 Message Pipe 를 구성한다. 시스템은 Operation System 에게 수용 가능한 USB Class 목록을 요청한다. Operation System 은 수용 가능한 USB Class 목록을 시스템에게 전달한다. 시스템은 수용 가능한 USB Class 를 저장한다.
측정	[지원가능한 USB Class 확장시 변경이 필요한 컴포넌트의 수]	

2.3.3 QA_03 전송 방식 변경

QA_03 (QS_07)	변경용이성	전송 방식 변경
설명	전송 방식을 변경하거나 추가/제거하는데 비용이 적어야 한다.	
환경	시스템이 정상 동작한다.	
자극	시스템의 전송 방식을 변경하거나 추가/제거한다.	
반응	N 개의 컴포넌트가 해당 변경으로 수정된다.	
측정	[전송 방식 변경시 변경이 필요한 컴포넌트의 수]	

2.3.4 QA_04 Host Controller의 USB 지원 버전 확장

QA_04 (QS_10)	변경용이성	Host Controller의 USB 지원 버전 확장
설명	Host Controller 하드웨어의 변경 등으로 지원하는 USB 버전이 변경되어도 비용이 적어야 한다.	
환경	시스템이 정상 동작한다.	
자극	Host Controller 하드웨어의 변경 등으로 지원하는 USB 버전이 변경된다.	
반응	N 개의 컴포넌트가 해당 변경으로 수정된다.	
측정	[USB 지원 버전 확장시 변경이 필요한 컴포넌트의 수]	

2.3.5 QA_05 동시에 연결 가능한 USB Device의 수

QA_05 (QS_11)	가용성	동시에 연결 가능한 USB Device의 수
설명	USB Hub에 연결된 Device를 포함하여 동시에 연결 가능한 USB Device의 수는 많을수록 좋다.	
환경	시스템이 정상 동작한다.	
자극	USB Device 를 USB 슬롯에 삽입한다.	
반응	1. Host Controller 는 시스템에게 USB Device 의 연결을 통보한다. 2. 시스템은 Host Controller 에게 새로 연결된 USB Device 에 최소 전력(100mA)를 공급을 명령한다.	

	<p>3. Host Controller 는 시스템에게 전력의 공급을 통보한다.</p> <p>4. 시스템은 Host Controller 에게 USB Device 와 시스템간 통신할 수 있는 주소를 할당해서 전달한다.</p> <p>5. Host Controller 는 시스템에게 새로 연결된 USB Device 의 VID, PID 를 전달한다.</p> <p>6. 시스템은 새로 연결된 USB Device 가 이전 연결된 적 있는지 확인한다.</p> <p>6-1. 연결된 적 없는 장치라면 Host Controller 에게 USB Device descriptor 를 요청한다.</p> <p>6-2. Host Controller 는 시스템에게 새로 연결된 USB Device 의 USB Device descriptor 를 전달한다.</p> <p>6-3. 시스템은 USB Device descriptor 에서 USB Class 를 확인하고 수용할 수 있는지 확인한다.</p> <p>7. 시스템은 USB Device 의 요구 리소스에 대한 가용 자원이 충분한지 확인한다. 충분하지 않다면 시스템은 요구 사항과 가용 자원에 대한 협상된 값을 도출한다.</p> <p>8. 시스템은 Host Controller 에게 7 번에서 계산된 자원만큼의 전력을 새로 연결된 USB Device 에 공급하도록 명령한다.</p> <p>9. 시스템은 LED Controller 에게 USB Device 가 연결된 위치의 LED 를 점등하도록 명령한다.</p> <p>10. 시스템은 Host 와 USB Device 가 통신할 수 있는 Stream Pipe 를 개설한다.</p> <p>11. 시스템은 USB Device 가 연결된 적 없던 장치라면 Flash Controller 에게 Known Device 로 저장을 요청한다.</p> <p>12. 시스템은 Operation System 에게 새로운 Stream Pipe 가 개설되었음을 통보한다.</p>
측정	[동시에 연결 가능한 USB Device의 수]

2.3.6 QA_06 외부 모듈 교환

QA_06 (QS_08)	변경용이성	외부 모듈 교환
설명	외부 모듈이 변경되어도 비용이 적어야 한다.	
환경	시스템이 정상 동작한다.	
자극	시스템과 통신하는 외부 모듈이 변경된다.	

반응	N 개의 컴포넌트가 해당 변경으로 수정된다.
측정	[외부 모듈 교환시 변경이 필요한 컴포넌트의 수]

2.3.7 QA_07 디바이스 정보 조회 시간

QA_07 (QS_06)	성능	디바이스 정보 조회 시간
설명	데이터 정보 조회 시간은 빠를수록 좋다.	
환경	USB Device가 USB 슬롯에 삽입되어 있다.	
자극	운영체제가 삽입되어 있는 USB Device 의 정보를 요청한다.	
반응		<ol style="list-style-type: none"> 1. Operation System 은 시스템에게 연결된 USB Device 들의 정보를 요청한다. 2. 시스템은 모든 연결된 USB Device 정보를 수집한다. 3. 시스템은 Operation System 에게 수집한 정보를 전달한다.
측정		[디바이스 정보 조회 시간] = [OS가 USB Device의 정보를 요청하는 시각] – [시스템이 USB Device의 정보를 OS에게 전달하는 시각]

3. 시스템 구조

3.1. Deployment View

본 시스템의 시스템 구조는 멀티 쓰레드 구조로 병렬성이 필요한 작업에 따라 여러 개의 쓰레드로 나누어 동작한다. 그에 따른 전체 Deployment View는 아래 그림과 같다.

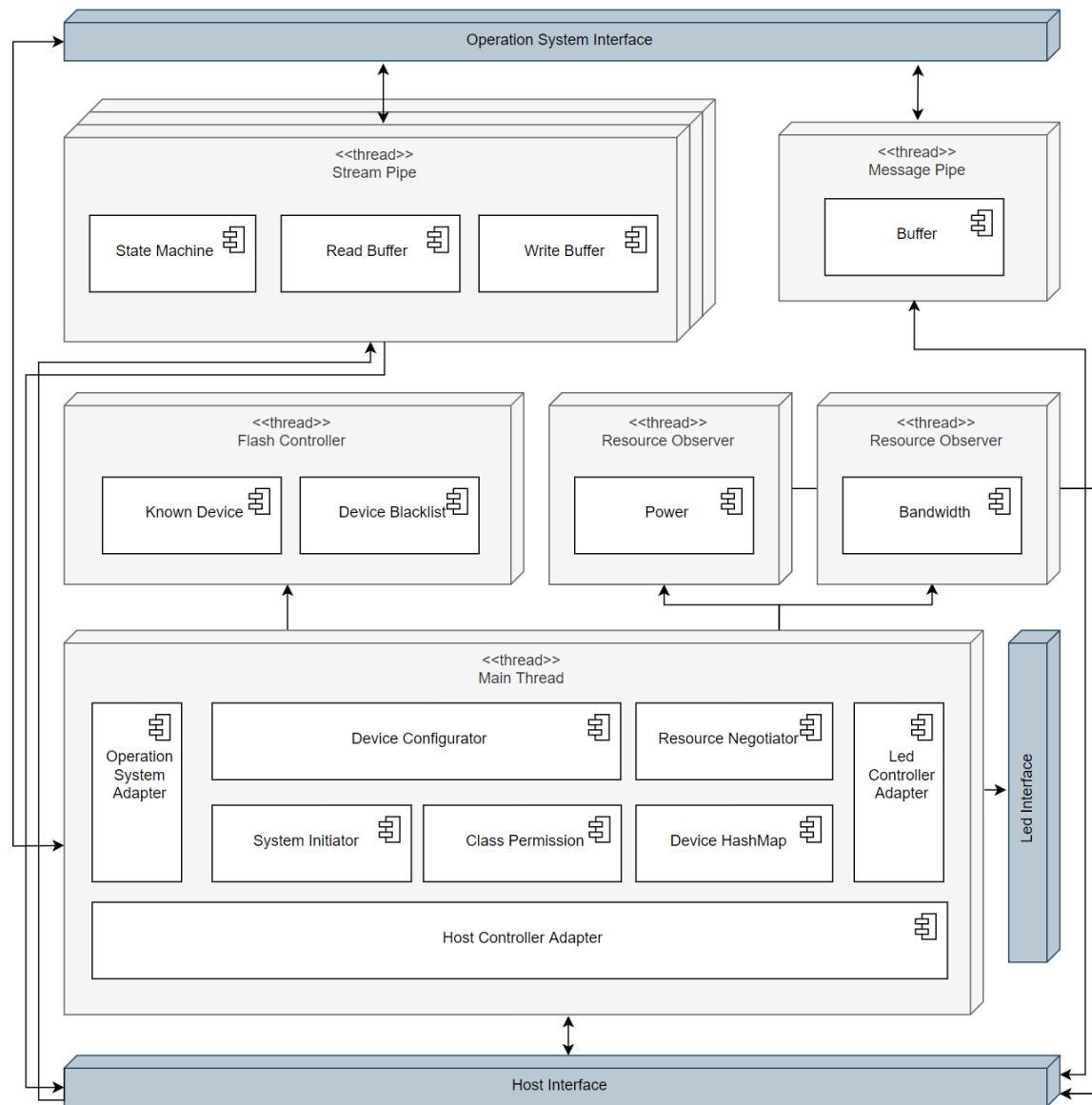


Figure 6 - Deployment View

3.1.1. Stream Pipe의 쓰레드 동작

Stream Pipe는 Read Buffer 및 Write Buffer에서 IRP와 Transaction 간의 전환 및 Operation System / Host Controller로의 전송을 담당한다. 그리고 Stream Pipe와 관련된 핵심적인 아이디어는 전송 속도를 쓰레드 우선순위에 따른 스케줄링에 따라 조절하는 것이다. Stream Pipe는 네 가지 전송 타입을 가지고 있는데 특성에 따라 전송 주기나 속도가 다르다. 이러한 전송 특성에 따라 쓰레드 우선순위를 부여하여 스케줄링된 쓰레드가 Host Controller의 Buffer에 데이터를 넣어 전송할 수 있도록 하기 위해 Stream Pipe가 개설될 때마다 별도의 쓰레드로 배치한다.

이는 근본적으로 QA_01(데이터 전송 속도) 충족을 위해 전송의 병렬성을 확보하고 최종적으로 Host Controller의 가용성을 높이기 위함이다. 공유 자원에 대한 접근은 Stream Pipe 개설 시 한번 접근하고 서로의 간섭없이 스케줄링 만으로 전송의 병렬성을 확보하여 동기화 문제를 해결하도록 한다. 쓰레드의 할당은 Thread Pool을 통해 진행하며 이를 통해 NFR_01(디바이스 설정 시간)을 달성할 수 있도록 한다.

3.1.2. Message Pipe의 쓰레드 동작

Message Pipe(Default Pipe)는 Stream Pipe와 분리되어 별도의 컴포넌트로 존재하며 제어 정보의 전송이라는 별도의 책임을 가진다. Message Pipe를 별도의 모듈로 두는 구조로 Stream Pipe가 전송 중 일지라도 제어 정보의 전송이 보장될 수 있도록 한다. 그를 위해 Message Pipe는 별도로 하나의 쓰레드로 배치한다. 이는 NFR_01(디바이스 설정 시간) 확보를 위한 최대한 빠른 제어 신호의 전송을 확보하기 위함이다.

3.1.3. Resource Observer의 쓰레드 동작

Resource Observer는 각각 Power와 Bandwidth의 가용 리소스를 감지하는 별도의 쓰레드로 구성되며 주기적으로 가용 리소스를 파악하고 질의 시 응답한다. 이를 통해 데이터 전송 중일지라도 주기적으로 가용 가능한 리소스를 파악할 수 있도록 한다. 다만 너무 짧은 주기로 리소스를 파악하여 전송을 위한 Stream Pipe의 스케줄링에 차질이 없도록 운영환경에서 주기에 대한 Threshold를 잘 찾아야 한다. 이는 최대 효율의 리소스를 할당하여 QA_05(동시에 연결 가능한 USB Device의 수)를 확보하기 위함이다.

3.1.4. Flash Controller의 쓰레드 동작

Known Device에 대해서 Fast Track으로 연결하는 방식을 위해 시스템은 연결 이력을 Flash 장치에 저장한다. 이러한 상황에서 NFR_02(시스템 부팅 시간)에 따라 시스템 부팅 시간을 최소화하기 위해서는 Flash라는 하드웨어에서 별도의 쓰레드를 통해 데이터를 병렬적으로 읽어온다.

3.1.5. Main Thread 동작

Main Thread는 시스템의 메인 스트림으로서 시스템의 주요 로직들을 담당한다. Main Thread의 동작 중 병렬성이 필요한 부분에 대해서 3.1.1 ~ 3.1.4의 쓰레드를 이용하는 방식으로 전체 시스템이 동작한다. 아래 내용은 Main Thread의 주요 로직에 대한 정리이다.

- Known Device: USB Device의 연결 이력을 저장하여 한번 이상 연결되었던 장치의 경우 빠른 연결을 지원한다. 장치의 구분자는 USB Specification에 정의된 VID, PID를 조합하여 Hash Key를 만들어 HashMap에 저장하는 방식을 사용한다.
- Device HashMap: 라운드 로빈 방식으로 USB Device에 주소를 부여하며 VID, PID를 이용한 Key를 통해 HashMap을 운영한다. Known Device는 연결 이력 만을 관리하는 기능이며 Device HashMap은 현재 연결되어 있는 USB Device를 관리하는 기능이다.
- System Initiator: 시스템 초기화 및 종료를 위한 모듈이며, 시스템 초기화 및 종료를 위한 시나리오를 관장한다.
- Class Permission: 허용 가능한 USB Class 정보를 부팅 시간에 Operation System으로부터 전달받아 저장하고 새로운 USB Device의 연결 허용여부를 결정한다.

3.2. 주요 프로세스 별 동작

3.2.1. 전송 프로세스 측면

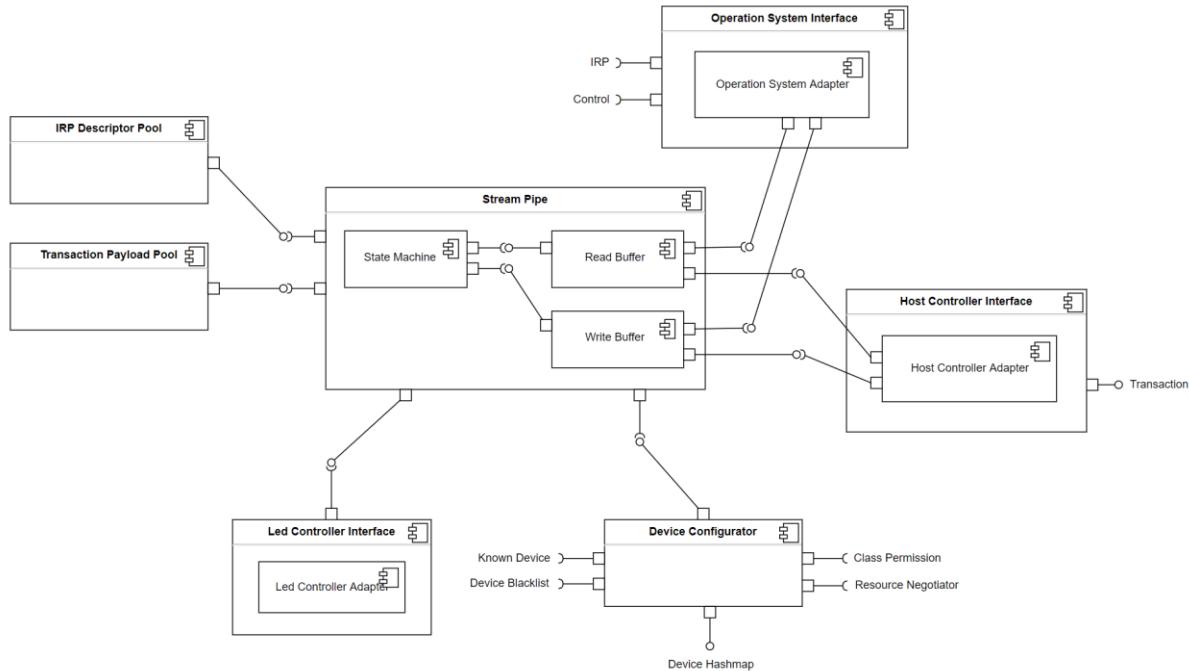


Figure 7 - 전송 프로세스 측면 C&C View

전송 측면에서 주요한 컴포넌트는 Stream Pipe이다. 복수개의 Stream Pipe가 Operation System – Stream Pipe – Host Controller로 이어지는 독립적인 Path를 갖고, 3.1에서 설명했듯 스케줄링되는 Stream Pipe가 그 Path를 통해 데이터를 전송한다. Stream Pipe는 Read Buffer, Write Buffer가 분리되어 있으며 전자가 Host Controller로부터 전송되는 데이터, 후자가 Client SW로부터 전송되는 데이터를 담게 된다. 또한 전송 상태와 유휴 상태를 구분하기 위해 Stream Pipe는 State Machine을 가지게 된다. Stream Pipe는 USB Device가 삽입되었을 때 Enumeration 과정에서 Device Configurator에 의해 생성되며 이에 관련된 구조와 동작은 3.2.3을 참고한다. 이 생성 과정에서 Write를 위한 IRP Description Pool과 Read를 위한 Transaction Pool을 이용하게 된다. 이에 대한 과정은 전송 과정을 이해하는데 중요한 과정이므로 이어서 자세히 설명한다. 마지막으로 Led Controller Adapter는 Stream Pipe에 의해 Data Rate를 전달받아 점멸하는 기능을 가진다.

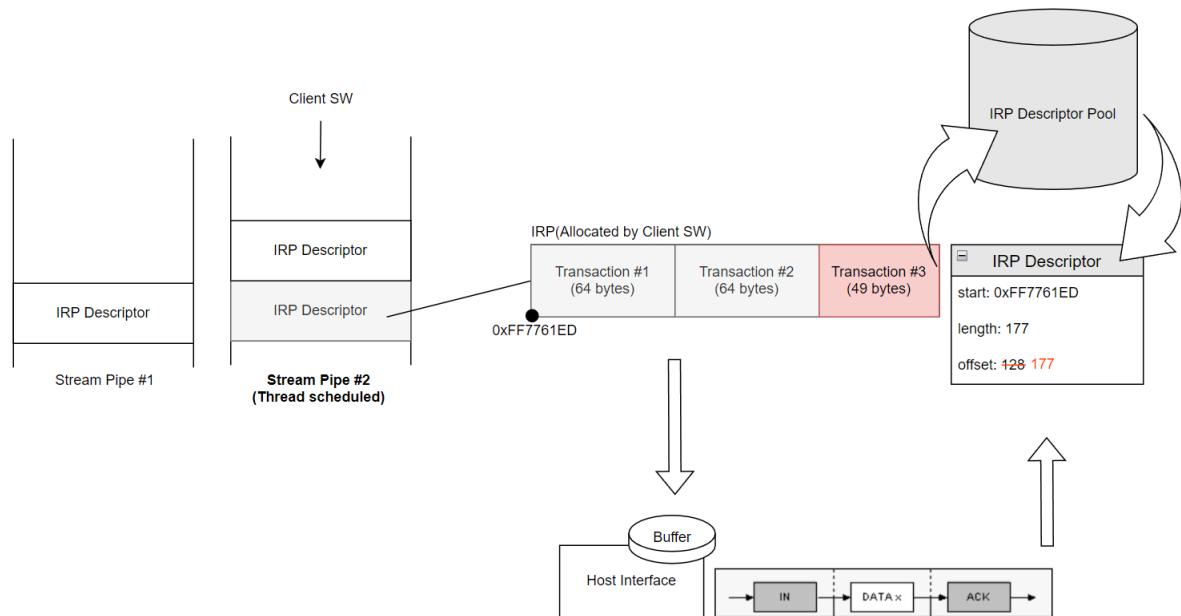


Figure 8 – USB Device로의 데이터 전송 구조

USB Device로 데이터 전송 시의 흐름에 대해 좀 더 자세히 알아본다. 데이터 전송 시에 시스템은 Client SW로부터 받은 IRP를 segmentation해서 Transaction으로 보내야 한다. 이때 Transaction의 Payload를 위한 메모리 공간을 할당하지 않고 IRP의 주소 값을 그대로 이용하는 구조를 시스템은 가지고 있다. 이를 위해 시스템은 IRP를 Client SW로부터 수신하면 전송한 지점을 기록하기 위한 IRP Descriptor를 생성하고 Transaction이 이루어질 때마다 offset을 기록한다. 전송 길이는 Stream Pipe에 정의된 통신 방식에 따라 적절한 크기의 Frame만큼 데이터를 전송한다. IRP를 구성하는 Transaction이 모두 전송되면 Queue형태인 Stream Pipe에서 IRP를 Pop하고 Client SW에게 IRP 전송이 완료되었음을 통지한다. 이 구조는 메모리의 할당 및 해제에 큰 성능 비용이 발생한다는 점에서 칙안하였다. 따라서 Transaction의 Payload를 위한 메모리를 할당하지 않는 것이 핵심이며, 할당이 필요한 것은 고정된 크기의 작은 IRP Descriptor 뿐이다. IRP는 Transaction에 비해 개수가 적으므로(Transaction이 IRP의 Segment이므로) 상대적으로 적은 비용이 발생하며, 미리 할당된 Pool을 통해 비용을 더 줄일 수 있다. 또한 시스템 내에서 IRP의 데이터를 Transaction의 Payload로 복사하는 비용을 줄일 수 있다. 위 전송의 구조에 따라 다음 그림에서 전체적인 흐름을 정리해본다.

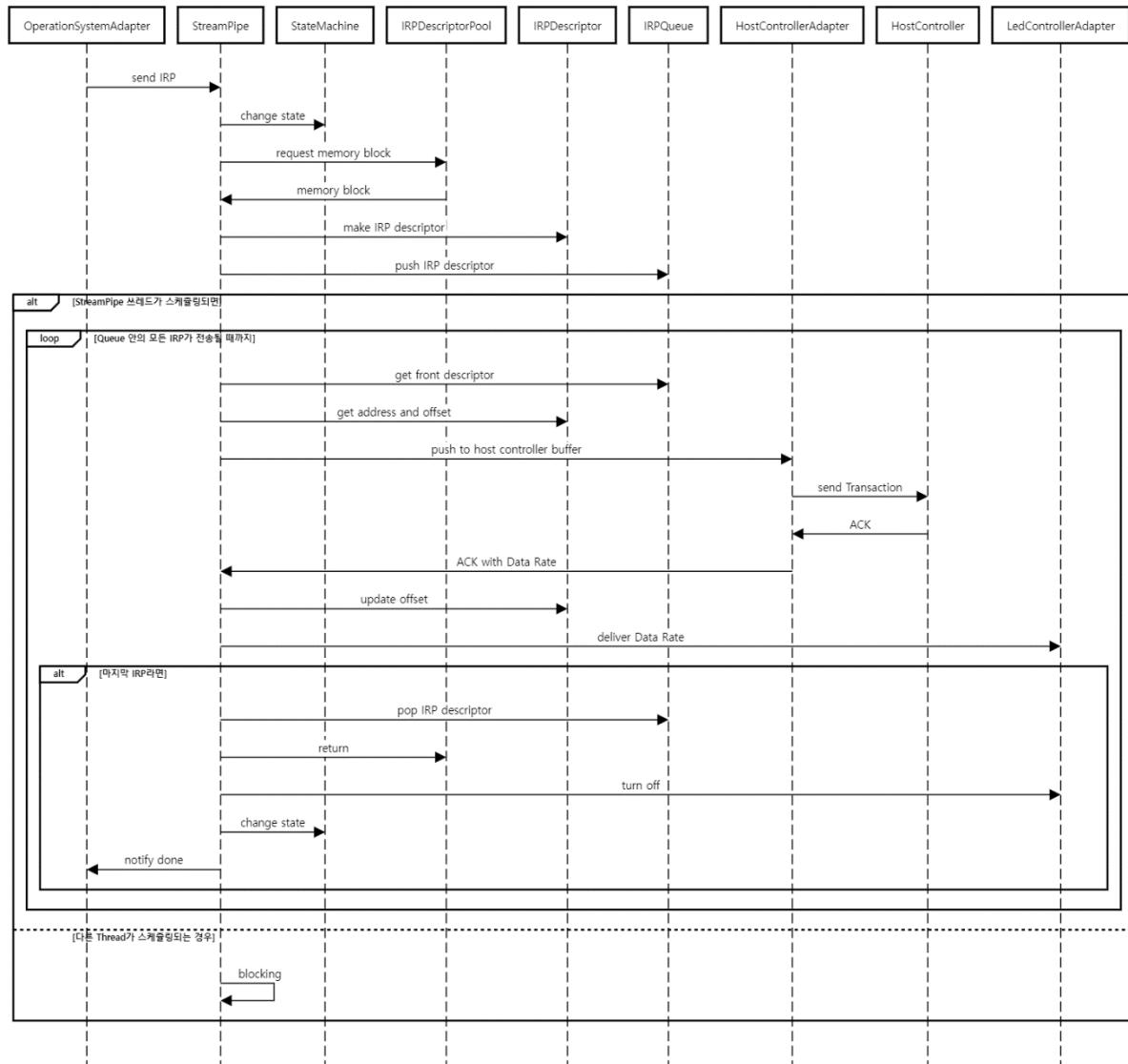


Figure 9 - USB Device로 데이터 전송 과정

Stream Pipe는 Operation System으로부터 전송된 IRP에 대한 정보로 IRP Descriptor를 생성하여 Buffer(Queue)에 넣는다. IRP Descriptor 생성을 위한 메모리 할당은 IRP Descriptor Pool을 사용한다. 해당 Stream Pipe가 스케줄링되어 전송을 위한 차례가 되면 Stream Pipe는 Buffer(Queue)에서 IRP Descriptor를 하나 꺼내서 이를 Transaction 단위만큼 Host Controller에게 전송한다. 이때 전송된 양을 IRP Descriptor에 offset 형태로 기록하고 Host Controller로부터 ACK를 받게 되면 다음 Transaction을 전송한다. Transaction을 전송할 때마다 Led Controller에게 Data Rate를 전달하여 전송속도에 맞춰 LED가 점멸할 수 있도록 한다. IRP의 모든 데이터가 전송되면 다음 IRP를 꺼내 전송한다.

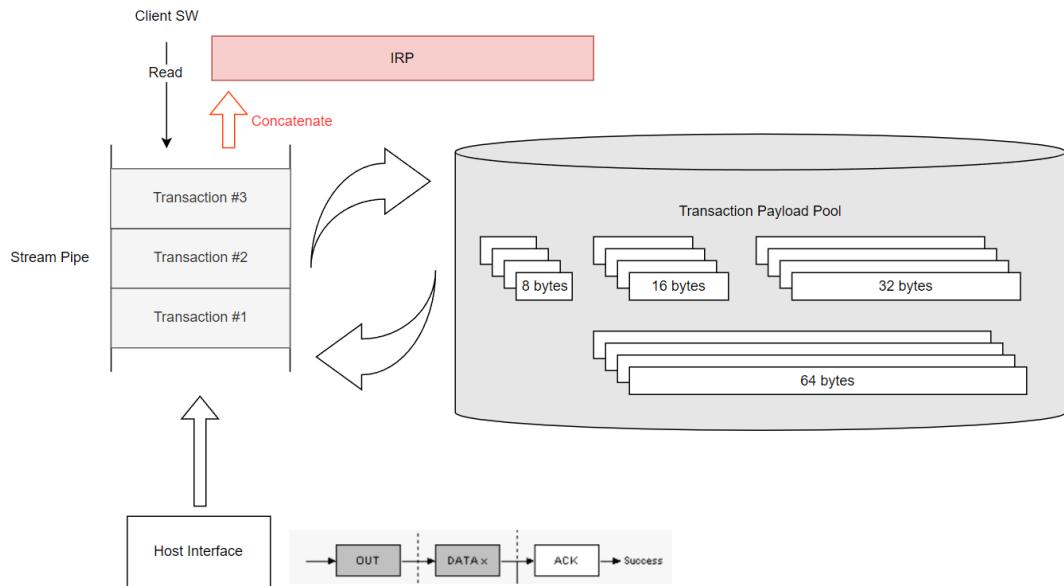


Figure 10 – Client SW로의 데이터 전송 구조

다음으로는 Host로 데이터를 전송하는 흐름에 대해 자세히 알아본다. 시스템은 메모리 Pool을 이용해서 USB Device가 보내는 Transaction을 Queueing했다가 Client SW로부터 Read 요청이 올 때 하나로 묶어 올려 보내는 방식을 가지고 있다. 시스템은 Transaction이 오면 이를 Frame 크기에 따라 Transaction Payload Pool에서 꺼낸 메모리 공간에 저장을 한다. Transaction 형태로 보관을 하는 이유는 Client SW로부터의 읽기 요청이 있을 때 쌓인 데이터가 하나의 IRP 형태로 올라가기 때문이다. Transaction들은 분산된 메모리 주소에 저장될 것이므로 하나로 Concatenate시켜서 하나의 메모리 주소로 만든 뒤 Descriptor와 함께 Client SW에게 전달한다. 이에 따른 전송의 흐름은 다음 그림을 통해 정리한다.

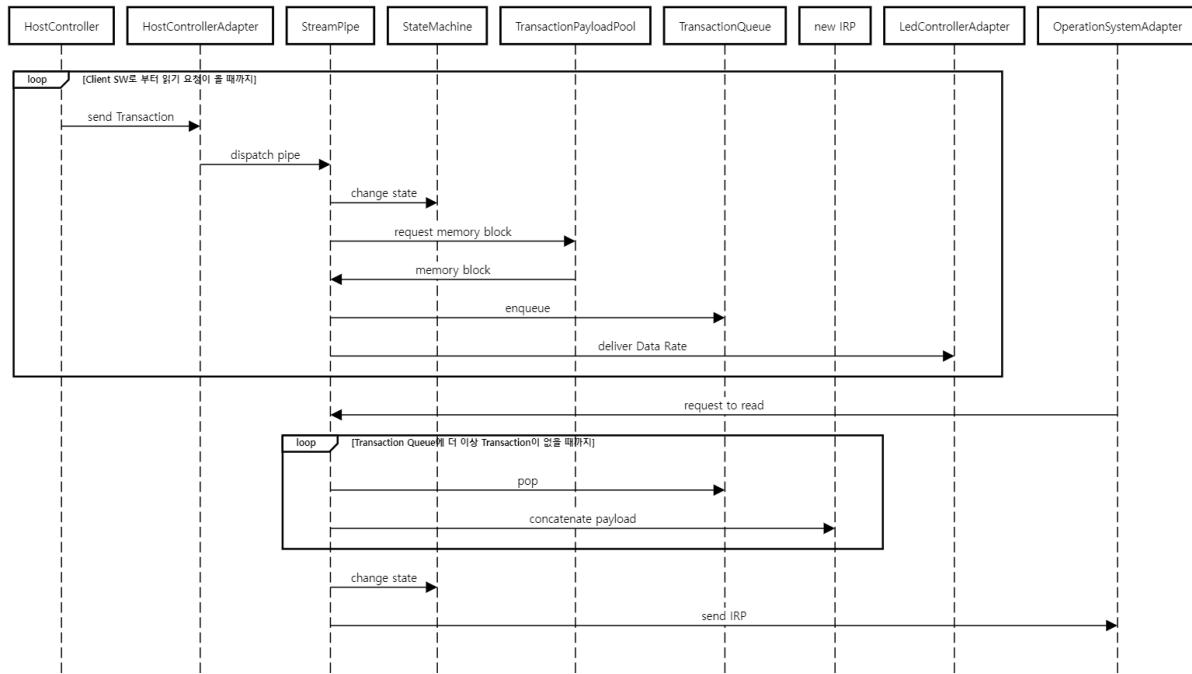


Figure 11 - Host로 데이터 전송 과정

Host Controller로부터 Transaction이 전송되면 시스템은 그 데이터의 목적지(Client SW)에 해당하는 Stream Pipe를 찾아 Buffer(Queue)에 Transaction을 넣는다. 이 과정에서 Transaction Payload를 위한 메모리 할당은 Transaction Payload Pool을 사용한다. 이 과정에서 Payload에 대한 데이터 복사가 1회 발생한다. 또한 Data Rate를 Led Controller에게 전달하여 전송속도에 맞춰 LED가 점멸할 수 있도록 한다. Operation System으로부터 데이터 읽기 신호가 오면 지금까지 쌓인 Transaction Payload를 순서대로 Concatenate하여 하나의 IRP 데이터를 구성한 뒤 Operation System에게 전달한다.

3.2.2. 시스템 초기화 프로세스 측면

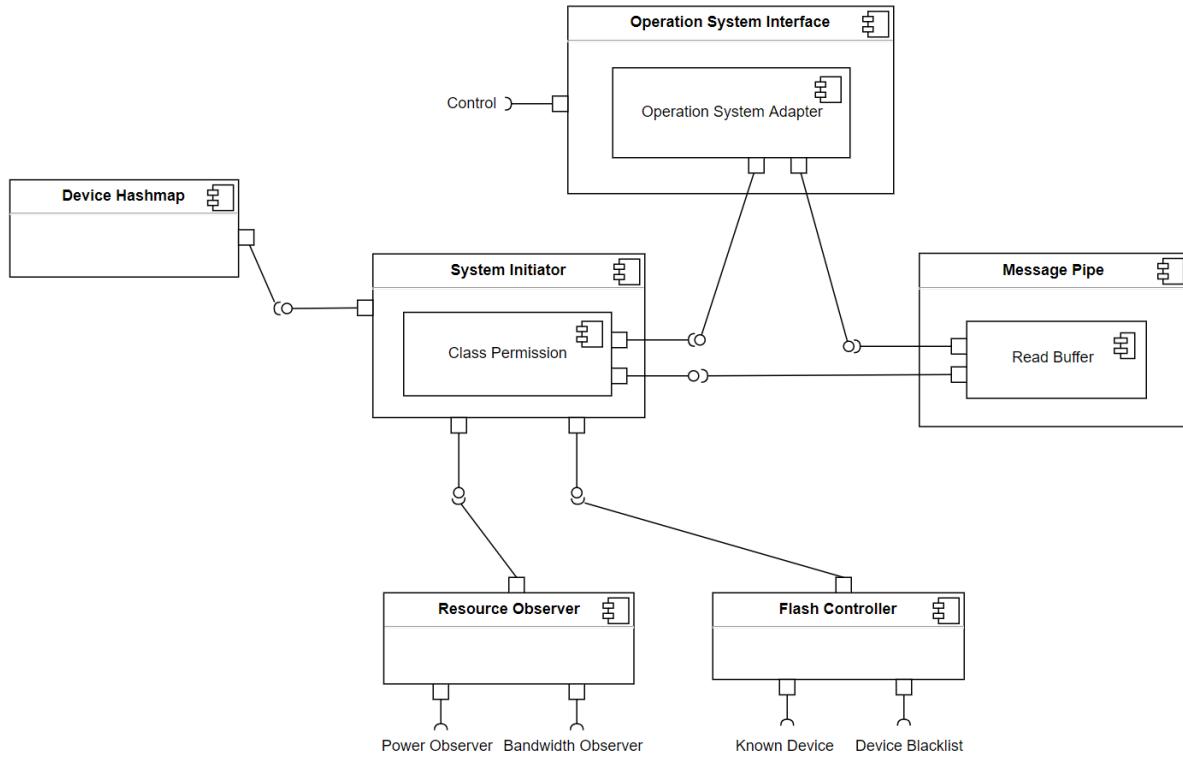


Figure 12 - 시스템 초기화 프로세스 측면 C&C View

시스템 시작 및 시스템 종료의 시나리오를 관장하는 컴포넌트는 System Initiator이다. System Initiator는 Operation System으로부터 시스템 시작을 통지 받으면 정해진 시나리오에 따라 순차적으로 주변 컴포넌트들을 초기화한다. Message Pipe를 생성하여 USB Device 삽입 시 제어 정보를 받을 수 있는 Default Pipe 역할을 담당하도록 하며, Operation System으로부터 USB Class Permission을 얻어와 비트맵 형태로 보관한다.

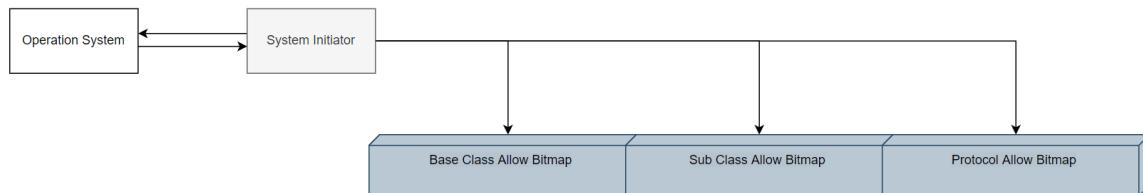


Figure 13 – 허용 가능한 USB Class 보관

또한 Resource Observer를 가동시켜 가동 리소스를 파악하도록 하며 새로운 USB Device가 삽입될 때 가용 리소스를 확인할 수 있도록 한다. 시스템 시작에 관한 시나리오는 아래 그림과 같다.

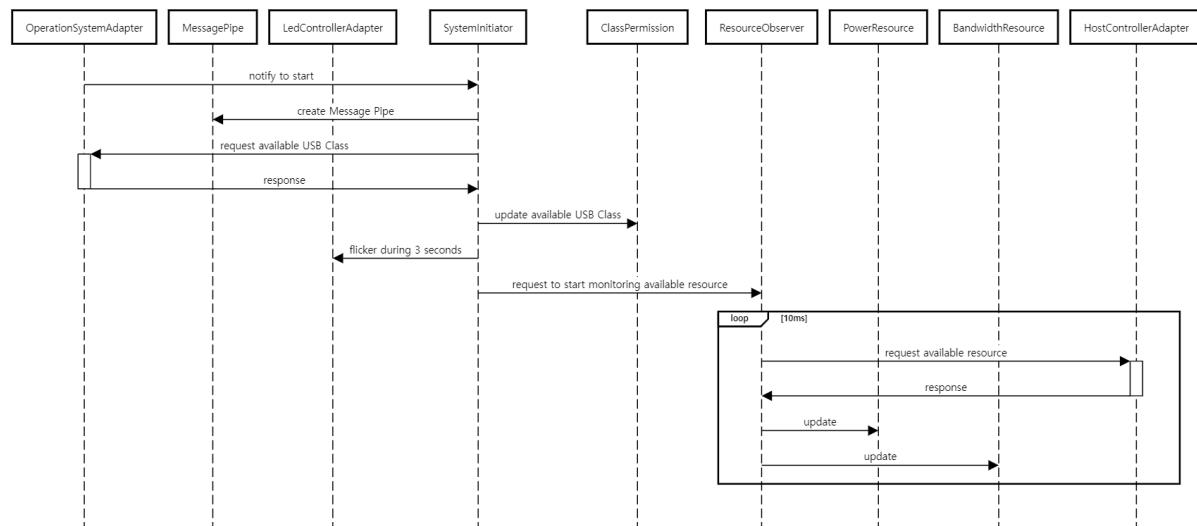


Figure 14 - 시스템 시작 과정

시스템 종료의 경우 Operation System이 시스템 종료를 통지하면 연결된 모든 USB Device에게 이를 통지하고 Device 정보 및 이와 연결된 Stream Pipe를 제거한다. 마지막으로 Message Pipe를 제거하고 모든 LED를 끈 뒤, 가용 리소스의 모니터링을 중단한다. 시스템 종료와 관련된 시나리오는 아래 그림과 같다.

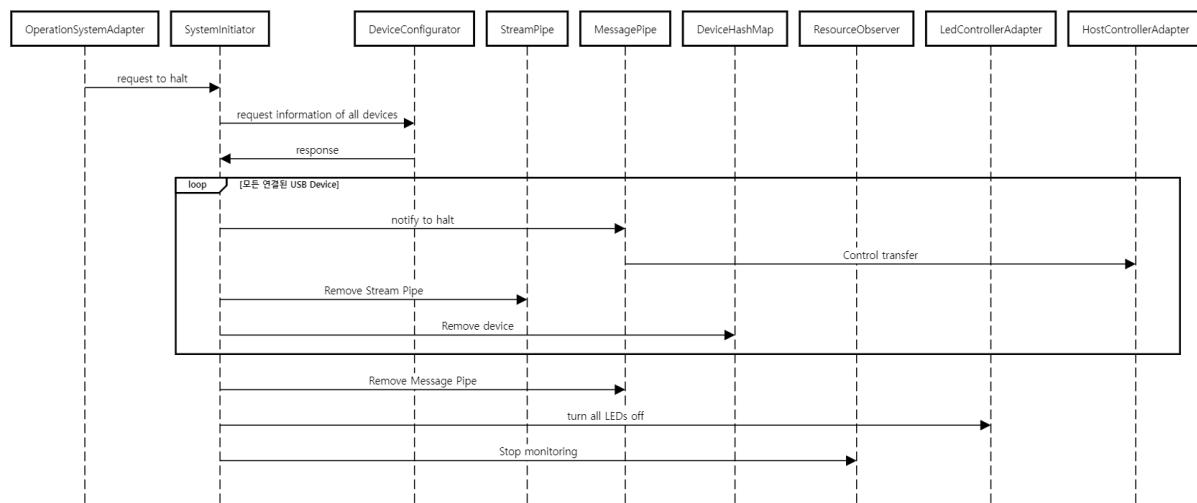


Figure 15 - 시스템 종료 과정

3.2.3. Enumeration 프로세스 측면

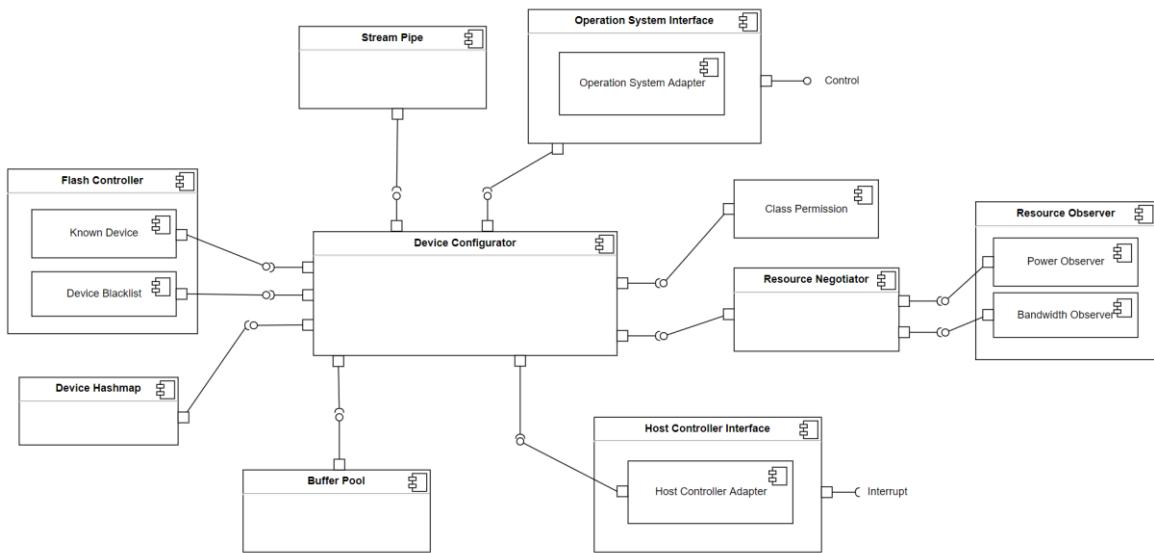


Figure 16 - Enumeration 프로세스 측면 C&C View

USB Device가 삽입되었을 때 이를 세팅하는 과정을 Enumeration 과정이라고 한다. 이에 대해 책 임을 갖고 시나리오를 관장하는 컴포넌트는 Device Configurator이다. Host Controller가 USB Device 연결을 통지하면 Device Configurator는 최소 전력을 우선 공급하고 고유의 주소를 할당한다. 주소 할당은 Default Pipe를 위한 0번을 제외하고 127개의 주소를 라운드 로빈 방식으로 할당 한다.

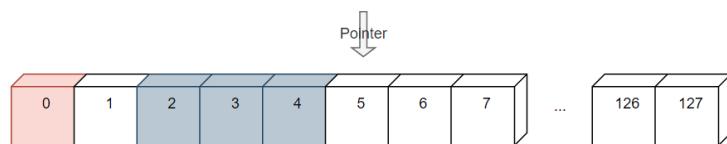


Figure 17 - USB Device 주소 할당 방식

이 주소를 통해 시스템은 우선 VID, PID를 전달받아 연결 이력이 있는지를 확인한다. 연결 이력이 있는 경우 Known Device HashMap을 통해 Descriptor를 얻을 수 있어 전체 Descriptor를 전달받는 과정을 생략할 수 있다.

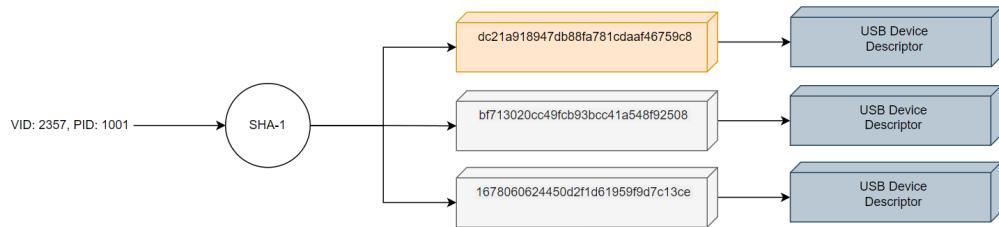


Figure 18 - 연결 이력 저장 방식

또한 보안 목적을 위해 Device Blacklist 조회를 하여 Blacklist에 있는 USB Device의 경우 연결을 허용하지 않는다. 연결 이력이 없는 USB Device의 경우 시스템은 전체 Descriptor를 전달받게 되고 USB Class 정보를 얻어서 허용 가능한지를 파악한다. 위 세 경우에서 연결이 허용되지 않는 경우 Operation System에게 허용되지 않는 USB Device가 연결 시도했음을 통지한다. 이후 시스템은 USB Device가 요구하는 리소스와 Resource Observer로부터 파악한 가용 리소스를 확인하여 리소스 협상을 진행한 뒤 그에 맞는 리소스를 할당하고 전력을 공급한다. 시스템은 새로운 USB Device를 위한 Stream Pipe를 생성한다. 과정의 시간 단축을 위하여 Stream Pipe의 Buffer는 미리 구성된 Buffer Pool을 사용한다.

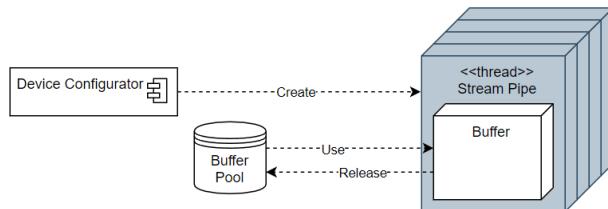


Figure 19 - Stream Pipe의 Buffer Pool 사용

또한 연결된 USB Device의 정보를 저장하는데 VID, PID 정보를 키 값으로 하여 Hash 구조로 데이터를 저장한다. Known Device로의 등록도 진행하며 이후 Operation System에게 통지한다.

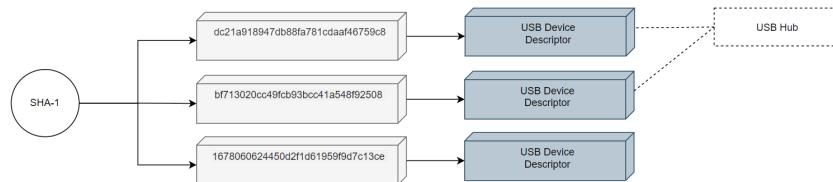


Figure 20 - 연결된 USB Device 정보 보관 방식

이러한 USB Device 연결 과정의 흐름은 아래 그림과 같다.

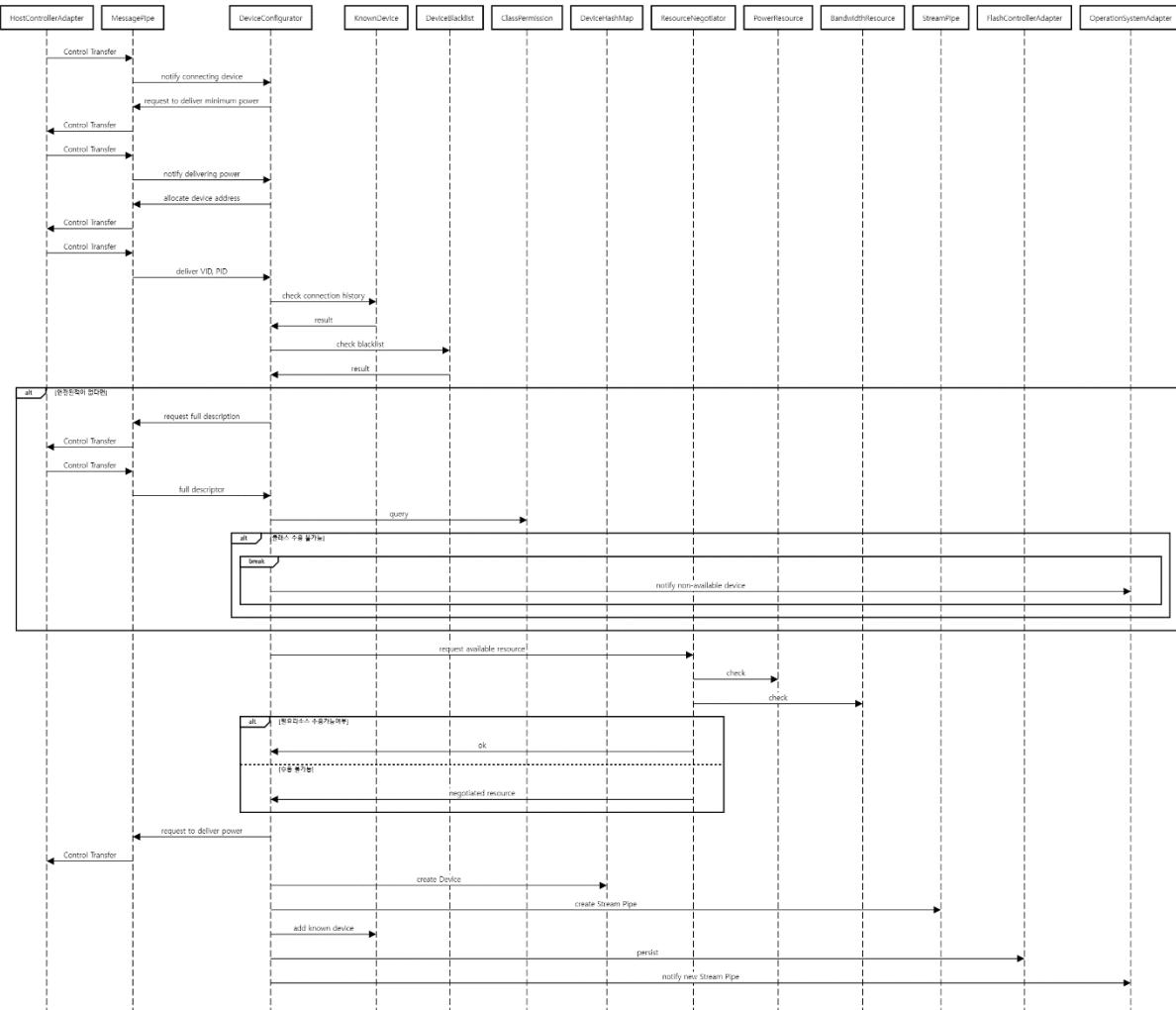


Figure 21 - USB Device 연결 과정

USB Device 연결 해제 과정은 크게 Operation System으로부터의 연결 해제와 실제 물리적인 USB Device의 연결 해제 이렇게 두 가지 과정으로 나뉜다. 전자의 경우 Windows 환경에서 “USB 장치 안전하게 제거” 기능과 유사한 기능이며 후자는 연결이 물리적으로 해제될 경우 Host Controller를 통해 그것이 통지된다.

Operation System으로부터의 연결 해제는 또 두가지로 나누게 되는데 Graceful Release의 경우 USB Device가 충분히 연결 해제 절차를 밟을 수 있도록 응답을 기다리는 형태이며, 그렇지 않은 경우 즉시 연결을 해제한다. USB Device에게 연결 해제를 통보한 뒤에는 관련된 Stream Pipe를 제거하며 연결된 USB Device 정보를 삭제하고 Operation System에게 연결 해제를 응답한다. 이 과정과 관련된 흐름은 아래 그림과 같다.

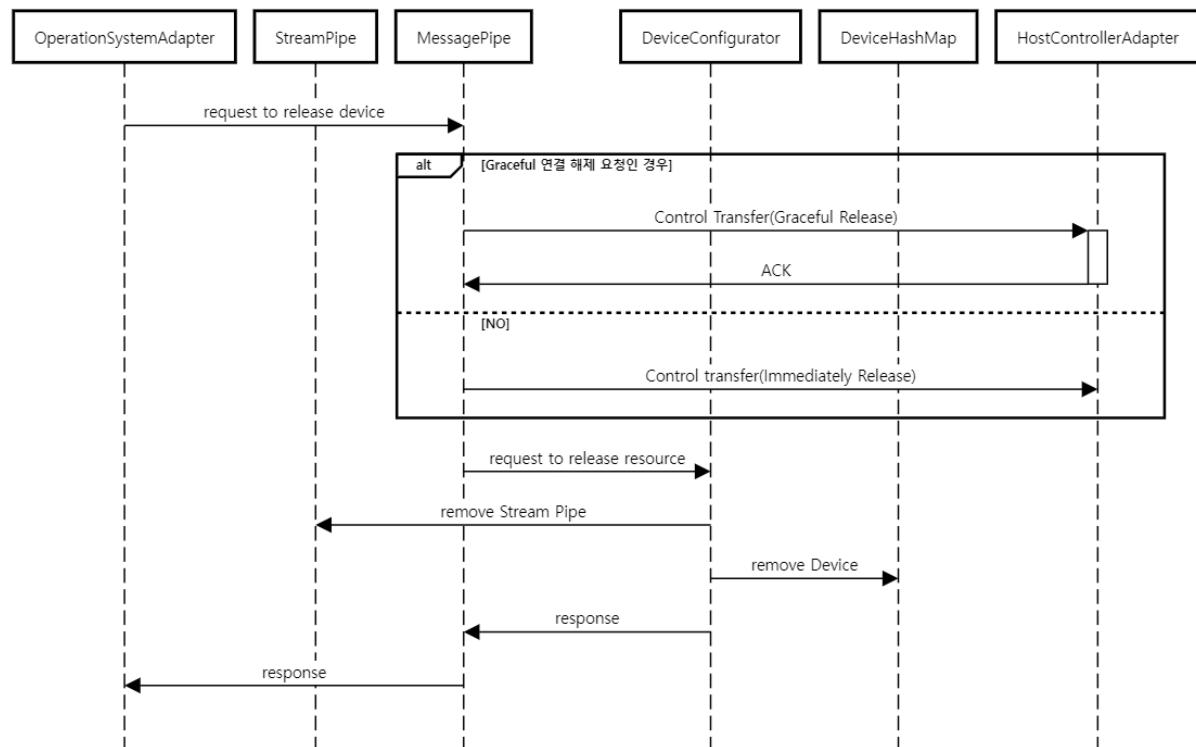


Figure 22 - UC_04. USB Device 연결 해제 요청

USB Device가 물리적으로 연결 해제되는 경우 이는 Operation System에게 즉시 통지되며, Operation System은 Stream Pipe에 남아있는 Transaction을 모두 Read한 뒤, 최종 연결 해제를 시시한다. 시스템은 관련된 Stream Pipe를 제거하며 연결된 USB Device 정보를 삭제한다. 이 과정과 관련된 흐름은 아래 그림과 같다.

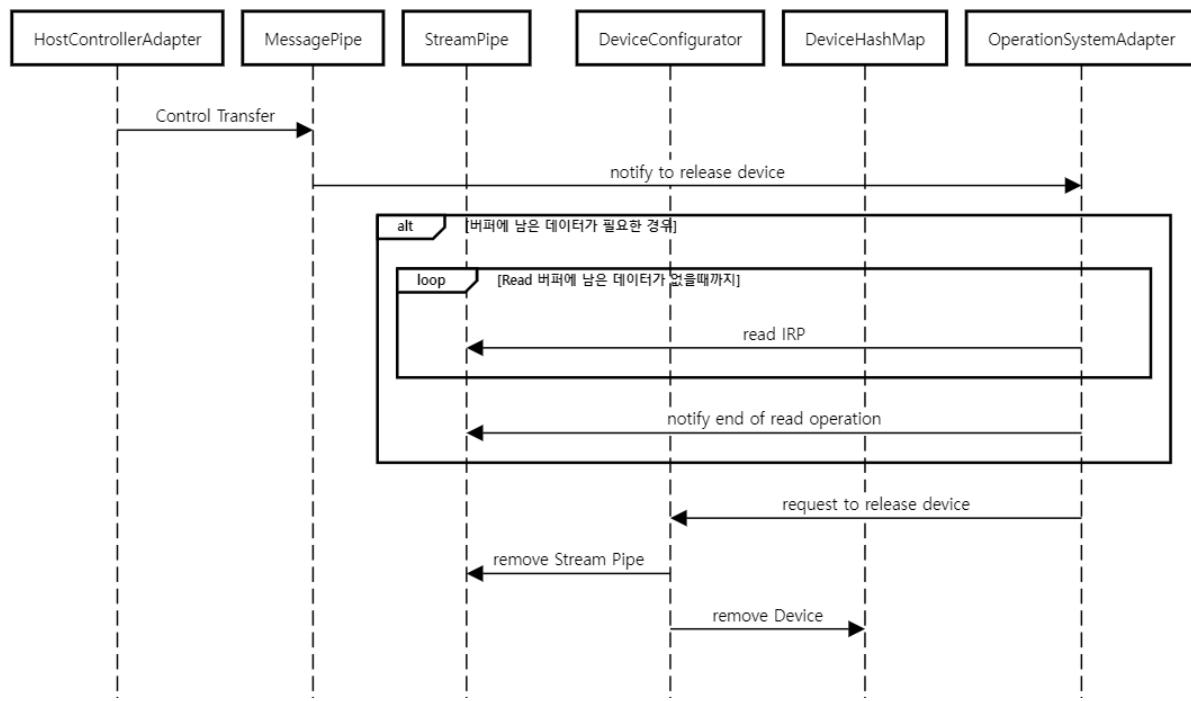


Figure 23 - UC_05. USB Device 연결 해제

4. 모듈 사양

4.1. Module View

본 시스템의 Module View는 아래 그림과 같다. 기능의 유사성과 연관도를 고려하여 External Interface, Resource Management, Device Management, Transfer Management, Scenario Management 등 다섯 개의 패키지로 구분하였으며 이후 파일 및 디렉토리 구조와 개발 조직 등의 영향을 고려하여 구분하였다.

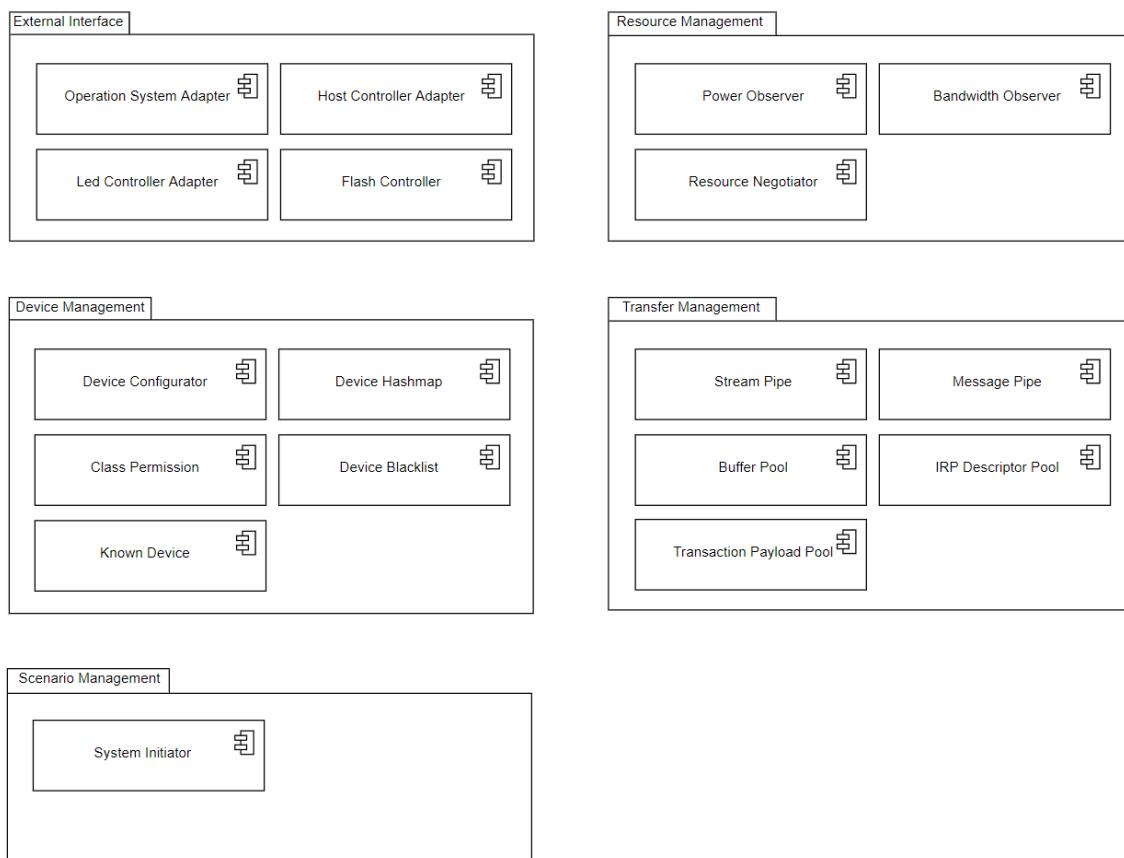


Figure 24 - Module View

External Interface는 외부 모듈(Operation System, Host Controller, Led Controller, Flash Controller)에 대한 어댑터를 모은 Package이다. Resource Management는 Resource Observer, Resource Negotiator로 나뉘며 가용 자원에 대한 감지와 할당에 관련된 기능을 모든 Package이다. Device Management는 Device 연결 이력, 연결된 USB Device 관리, Enumeration 시나리오 관리 등의 기능을 모든 Package이다. Transfer Management는 전송과 관련된 Package로 Pipe의 구현과 그를 위한 Memory Pool 기능을 모은 Package이다. 마지막으로 Scenario Management는 시나리오를 관리하는 모듈을 모든 Package로 System Initiator를 가지고 있는 Package이다. 위 패키지 구조와 소스코드의 경로를 일치시켜 개발 응집도를 높이고 협업의 편의성에 도움이 될 수 있도록 한다.

4.2. Class Diagram

4.2.1. 전체 디자인

시스템의 전체 클래스 디자인은 아래 그림과 같다. 각각의 클래스들은 별도의 IoC 컨테이너에 의해 관리되며 각각의 클래스에서 요청 시 Dependency Injection된다.

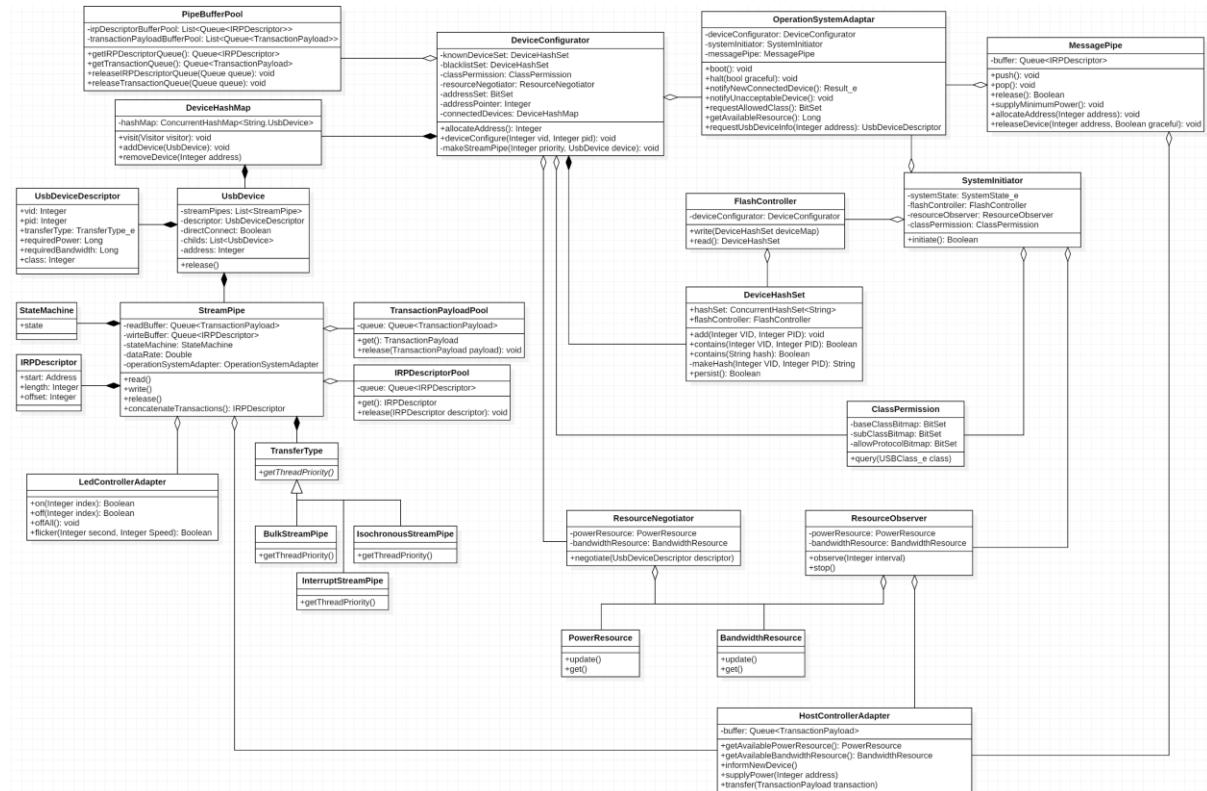


Figure 25 - Class Diagram

4.2.2. 클래스 명세

4.2.2.1. DeviceConfigurator

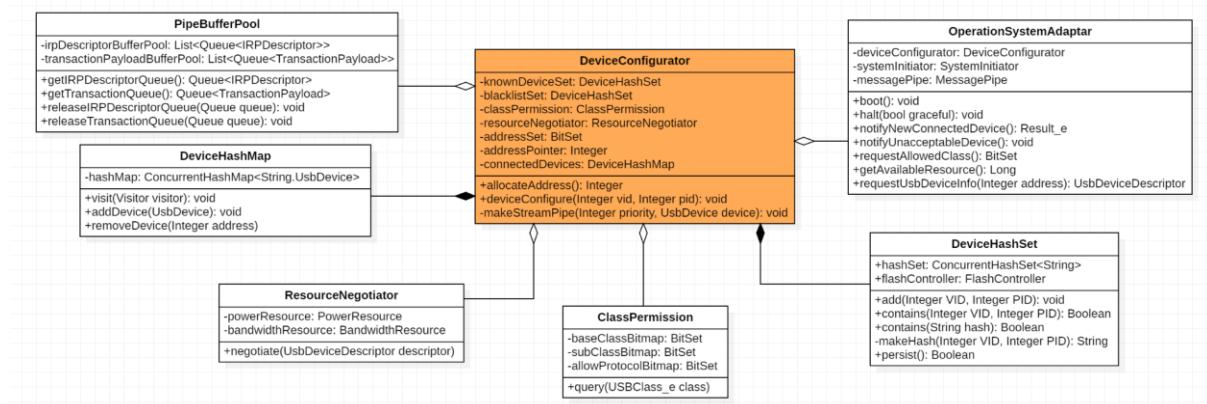


Figure 26 - DeviceConfigurator 클래스

설명	USB Device가 삽입되었을 때 장치를 설정하는 시나리오를 관리하는 클래스이다. USB Device를 식별하고 연결 하용여부를 결정하며 적절한 리소스를 할당한다. USB Device의 정보를 저장하며 해당 디바이스가 사용할 Stream Pipe를 생성한다.	
필드	knownDeviceSet	USB Device의 연결 이력을 관리하는 HashSet 자료구조를 가진 필드이다.
	blacklistSet	Operation System으로부터 안전하지 않은 연결로 인해 연결이 거부되었던 USB Device의 식별자를 관리하는 HashSet 자료구조를 가진 필드이다.
	classPermission	ClassPermission의 레퍼런스로 의존성을 주입받는다. 새로운 USB Device가 연결되었을 때 해당 장치의 USB Class가 수용 가능한지 수질의할 때 참조한다.
	resourceNegotiator	ResourceNegotiator의 레퍼런스로 의존성을 주입받는다. 새로운 USB Device에 할당 가능한 리소스를 질의할 때 참조한다.
	addressSet	USB Device 들에게 할당된 주소의 비트맵 정보를 갖는 필드이다.
	addressPointer	라운드 로빈 방식의 주소 할당을 위해 다음 번지를 가리키는 필드이다.
	connectedDevices	현재 연결되어 있는 USB Device들의 정보를 담은 HashMap 자료구조의 필드이다.
메서드	allocateAddress	새로 연결된 USB Device의 주소를 할당하는 로직을 작성하는 자리이다. addressSet, addressPointer 필드를 사용하여 라운드 로빈

	방식의 주소 할당을 구현한다.
deviceConfigure	새로운 USB Device가 연결되었을 때 호출되는 메서드로써 enumeration 과정의 시나리오가 시작되는 지점이다.
makeStreamPipe	새로운 USB Device가 연결되었을 때 데이터 전송을 위한 StreamPipe를 생성하는 로직을 작성하는 자리이다. Buffer Pool을 이용하여 생성하도록 한다.

4.2.2.2. SystemInitiator

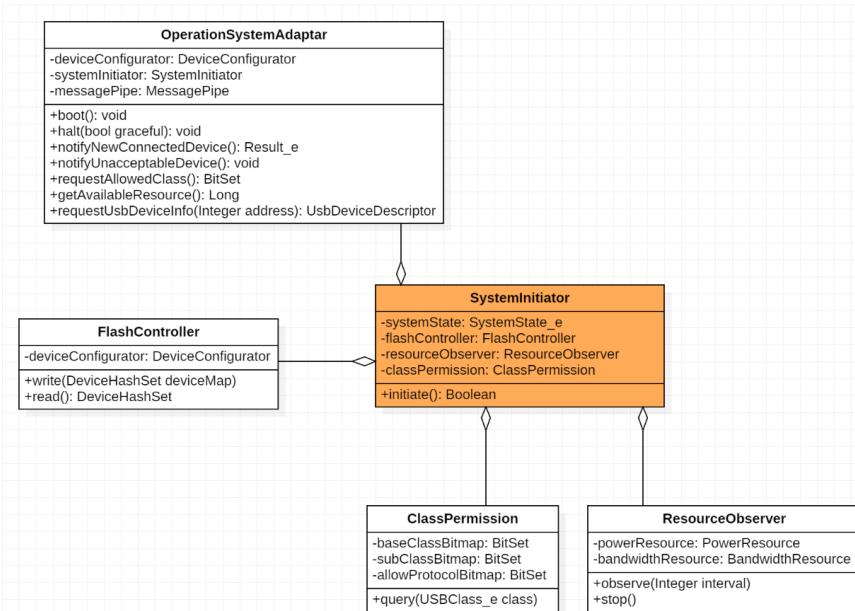


Figure 27 – SystemInitiator 클래스

설명	시스템 초기화 시나리오를 관장하는 클래스이다. OperationSystemAdapter로부터 notification을 받아 FlashController의 읽기, ClassPermission 얻어오기, ResourceObserver 가동하기 등의 책임을 가지고 있다.		
필드	systemState	시스템 전체의 상태로 INITIATE, HALTED 상태를 가진다.	
	flashController	FlashController의 레퍼런스로 의존성을 주입받는다.	
	resourceObserver	ResourceObserver의 레퍼런스로 의존성을 주입받는다.	
	classPermission.	ClassPermission의 레퍼런스로 의존성을 주입받는다.	
메서드	initiate	시스템 초기화의 시나리오를 작성하는 부분이다. OperationSystemAdapter를 통해 호출되며 필드들의 레퍼런스를 통해 초기화 시나리오를 진행한다.	

4.2.2.3. DeviceHashMap

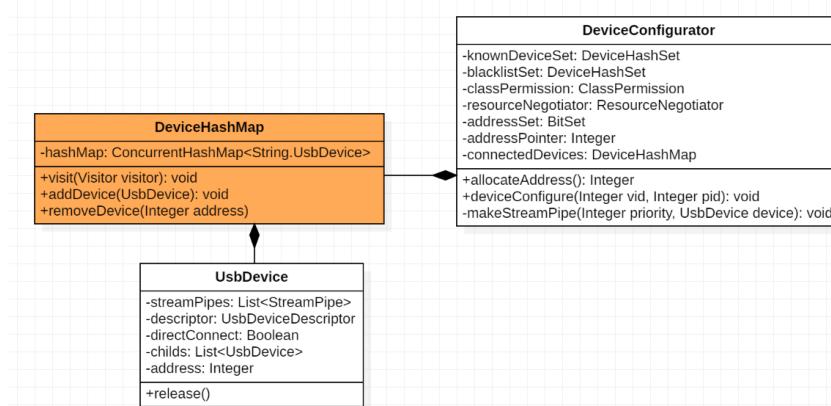


Figure 28 - DeviceHashMap 클래스

설명	연결된 USB Device의 정보를 보관하기 위한 자료구조 클래스이다.	
필드	hashMap	동시성 이슈에서 자유로운 ConcurrentHashMap 타입의 필드이다.
메서드	Visit	순회하며 USB Device 정보를 처리할 경우 visitor를 받아 순회하며 처리한다.
	addDevice	USB Device를 추가하는 로직을 작성하는 자리이다. 장치의 VID, PID를 이용해 hash의 키 값을 만들어 추가한다.
	removeDevice	USB Device를 삭제하는 로직을 작성하는 자리이다.

4.2.2.4. DeviceHashSet

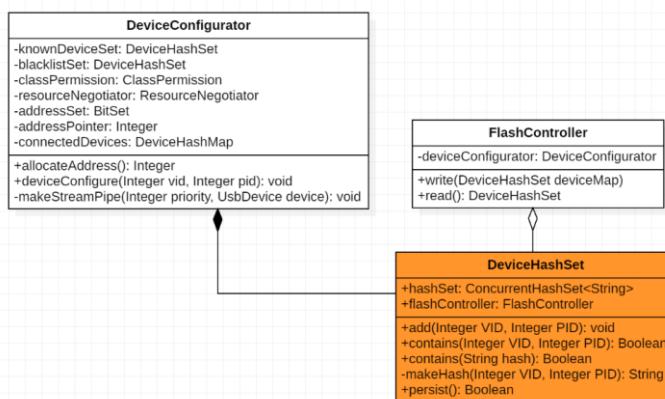


Figure 29 - DeviceHashSet 클래스

설명	Known Device와 Device Blacklist를 구현하기 위한 자료구조 클래스이다. DeviceConfigurator로부터 질의를 받으며, FlashController로부터 영속적인 데이터를 쓰고 읽을 수 있다.	
필드	hashSet	동시성 이슈에서 자유로운 ConcurrentHashSet 타입의 필드이다. 집합 개념으로서 데이터의 중복을 허용하지 않는다.
	flashController	FlashController의 레퍼런스로 의존성을 주입받는다. 영속적인 데이터를 쓰고 읽는데 참조된다.
메서드	add	USB Device의 식별자를 통한 키 값에 장치의 정보를 추가할 수 있도록 구현한다.
	contains	키 값을 전달받아 정보를 가지고 있는지 확인하는 내용을 작성한다.
	makeHash	키 값을 생성하는 로직을 작성하는 부분이다. SHA-1 알고리즘을 통해 생성하도록 구현한다.
	persist	flashController를 통해 데이터를 영속적으로 저장하는 내용을 구현한다.

4.2.2.5. FlashController

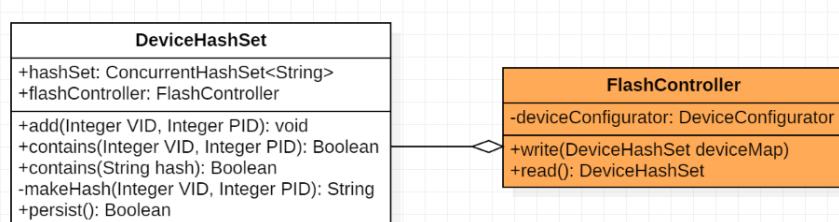


Figure 30 - FlashController 클래스

설명	Flash 디바이스를 조작하는 클래스이다.	
필드	deviceConfigurator	읽어온 데이터를 저장하기 위해 참조하는 레퍼런스이다.
메서드	write	자료구조에 있는 데이터를 영속적으로 저장하는 내용을 작성한다.
	read	저장된 내용을 읽어오는 내용을 작성한다.

4.2.2.6. StreamPipe

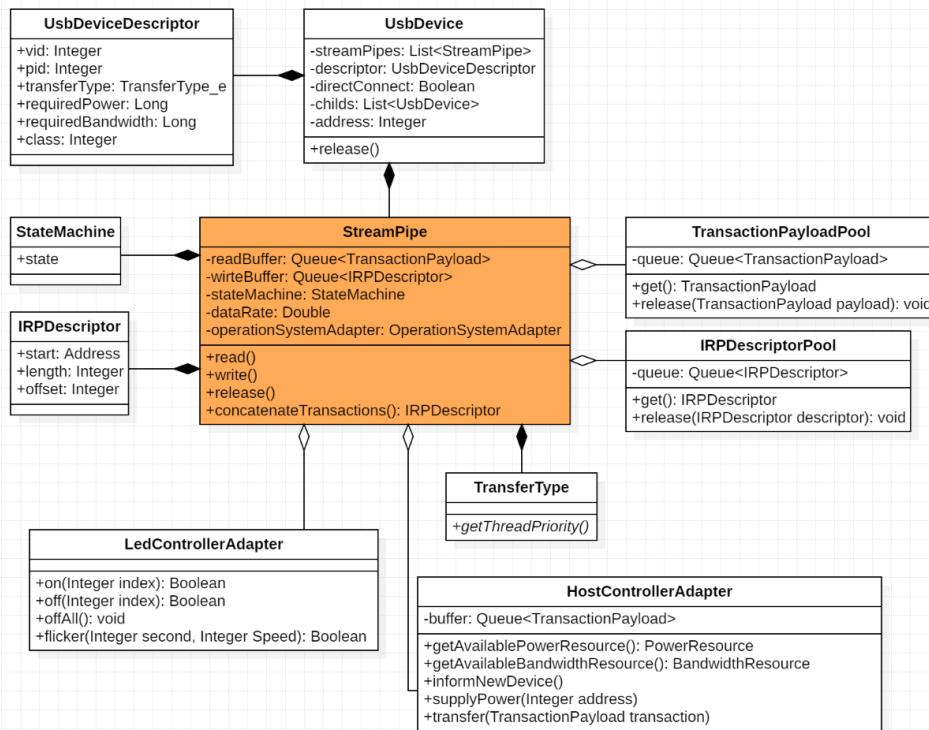


Figure 31 - StreamPipe 클래스

설명	StreamPipe는 Operation System과 Host Controller 사이에서 데이터 전송을 담당하는 클래스이다. 데이터 전송에 필요한 메모리 공간은 Memory Pool을 통해 제공받으며 전송 타입에 따라 스케줄링에 대한 우선순위를 갖는다. Led Controller와 인터페이스가 있으며 전송에 따라 점멸을 요청할 수 있다.	
필드	readBuffer	Client SW로 데이터를 전송하기 위한 Transaction Payload를 담는 queue 자료구조의 버퍼이다.
	writeBuffer	Host Controller로 데이터를 전송하기 위한 IRPDescriptor를 담는 queue 자료구조의 버퍼이다.
	stateMachine	전송 및 유회 상태를 기록한다.
	dataRate	전송 속도를 기록한다.
	operationSystemAdapter	Operation System에 접근하기 위한 래퍼런스이다.
메서드	read	Operation System으로부터 읽기 요청이 오면 concatenateTransaction 메서드를 호출하여 하나의 IRP를 얻은 뒤 Operation System에 전달하는 내용을 작성한다.

	write	스케줄링된 StreamPipe가 데이터를 HostControllerAdapter의 buffer에 IRP를 넣는 내용을 작성한다.
	release	장치의 제거 혹은 시스템 종료로 인해 StreamPipe가 해제될 때의 내용을 작성한다.
	concatenateTransactions	readBuffer에 쌓인 transaction을 순서대로 이어 붙여 하나의 IRP로 만드는 내용을 작성한다.

4.2.2.7. MessagePipe

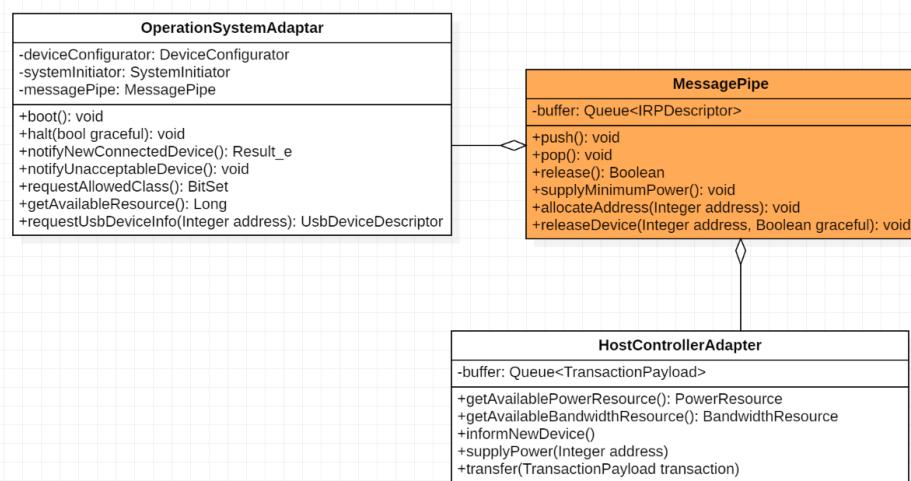


Figure 32 - MessagePipe 클래스

설명	Operation System과 Host Controller 사이에서 제어정보를 주고 받는 통로가 되는 클래스이다. 하나만 존재하며 전송은 선입선출 방식으로 이루어진다.	
필드	buffer	전송할 제어 정보를 담는 자료구조 필드이다.
메서드	push	전송할 제어 정보를 buffer에 넣는 내용을 작성한다.
	pop	전송할 제어 정보를 buffer에서 꺼내는 내용을 작성한다.
	release	시스템 종료로 MessagePipe를 제거하는 내용을 작성한다.
	supplyMinimumPower	최초에 새로운 USB Device가 연결될 때 최소 전력을 공급하는 내용을 작성한다.
	allocateAddress	새로운 USB Device에게 주소를 할당하는 내용을 작성한다.
	releaseDevice	장치가 해제될 때의 처리를 작성한다. graceful 방식인 경우 해당 장치에게 응답을 받고 나서 해제를 처리한다.

4.2.2.8. PipeBufferPool

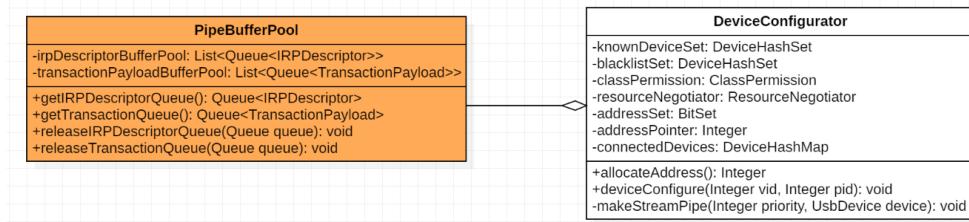


Figure 33 - PipeBufferPool 클래스

설명	Pipe 개설 시간을 단축하기 위한 메모리 풀 클래스이다. Read 동작을 위한 TransactionPayloadPool과 Write 동작을 위한 IRPDescriptorPool이 있다.	
필드	irpDescriptorBufferPool	IRPDescriptorPool의 메모리 풀 필드이다.
	transactionPayloadBufferPool	TransactionPayloadPool의 메모리 풀 필드이다.
메서드		
getIRPDescriptorQueue		IRPDescriptorPool을 하나 대여해주는 내용을 구현한다.
getTransactionQueue		TransactionPayloadPool을 하나 대여해주는 내용을 구현한다.
releaseIRPDescriptorQueue		IRPDescriptorPool을 반납받는 내용을 구현한다.
releaseTransactionQueue		TransactionPayloadPool 을 반납받는 내용을 구현한다.

4.2.2.9. OperationSystemAdapter

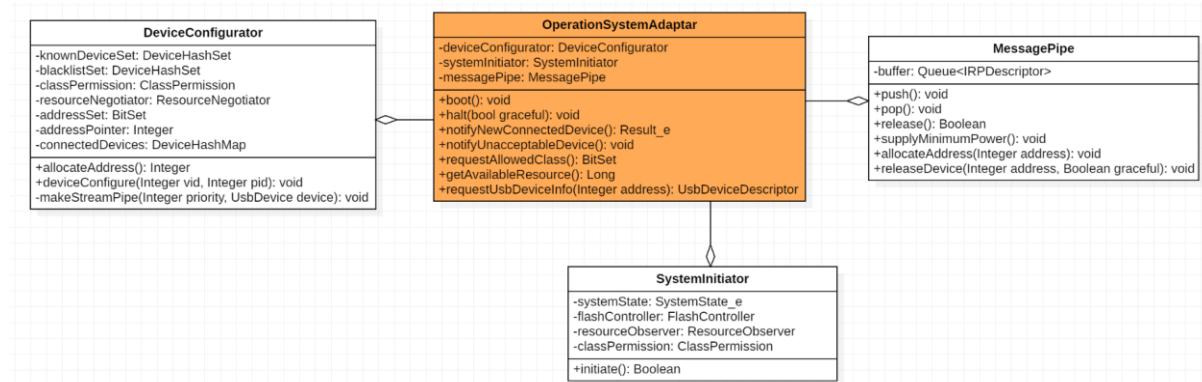


Figure 34 - OperationSystemAdapter 클래스

설명	Operation System 과의 인터페이스를 담당하는 클래스이다. 정해진 operation을 여러 Operation System에서 지원할 수 있도록 추상화한다.	
필드	deviceConfigurator	deviceConfigurator의 레퍼런스로 의존성을 주입받는다.
	systemInitiator	systemInitiator의 레퍼런스로 의존성을 주입받는다.
	messagePipe	MessagePipe의 레퍼런스로 의존성을 주입받는다.
메서드	boot	시스템이 부팅되었을 때 호출되며 systemInitiator를 참조하여 시나리오를 시작하는 내용을 구현한다.
	halt	시스템이 종료되었을 때 호출되며 systemInitiator를 참조하여 시나리오를 시작하는 내용을 구현한다.
	notifyNewConnectedDevice	새로운 USB Device의 설정이 완료되었을 때 Operation System에 공지하는 내용을 작성하는 부분이다.
	notifyUnacceptableDevice	연결이 허용되지 않는 장치가 삽입되었을 때 Operation System에 공지하는 내용을 작성하는 부분이다.
	requestAllowedClass	허용 가능한 USB Class 정보를 Operation System에게 요청하는 내용을 작성하는 부분이다.
	getAvailableResource	Host PC의 가용 리소스를 확인하는 내용을 작성한다.
	requestUsbDeviceInfo	Operation System이 연결된 USB Device 정보를 확인할 때 호출하는 메서드이다.

4.2.2.10. UsbDevice

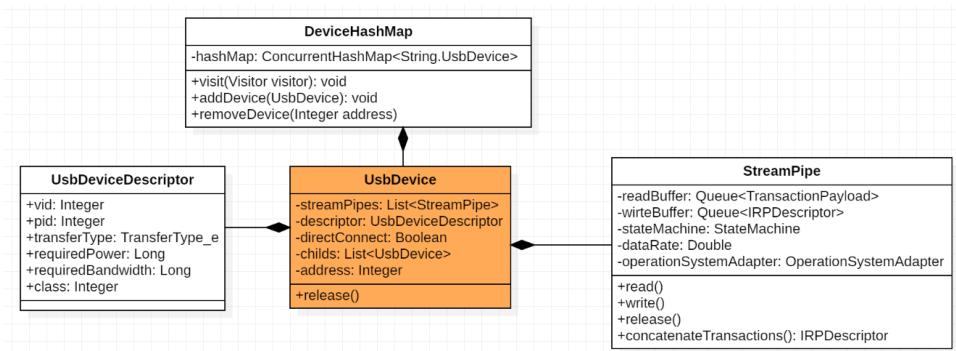


Figure 35 - UsbDevice 클래스

설명	연결된 USB Device의 정보를 갖는 클래스이다. Stream Pipe와 descriptor, 자식 및 주소에 대한 정보를 가지고 있다.	
필드	streamPipes	해당 장치가 사용하는 StreamPipe 들을 보관하는 자료구조이다.

	descriptor	해당 장치의 descriptor를 보관하는 필드이다.
	directConnect	해당 장치가 USB Hub가 아니라 Root Hub에 연결되어 있는지를 기록하는 필드이다.
	childs	해당 장치가 USB Hub이거나 Hub 기능을 가질 때 자식 장치들을 기록하는 필드이다.
	address	해당 장치가 할당받은 주소를 기록하는 필드이다.
메서드	release	해당 장치가 제거되었을 때 장치를 삭제하는 내용을 구현한다.

4.2.2.11. UsbDeviceDescriptor

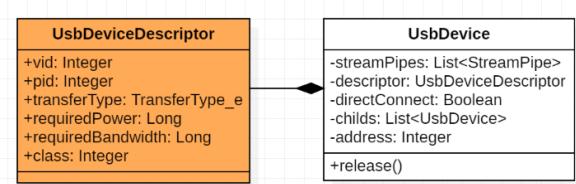


Figure 36 - UsbDeviceDescriptor 클래스

설명	USB Device로부터 전달받은 descriptor 정보를 담은 클래스이다. 구조체 형태의 데이터 클래스이다.	
필드	vid	해당 장치의 vender id를 기록하는 필드이다.
	pid	해당 장치의 product id를 기록하는 필드이다.
	transferType	해당 장치의 전송 타입을 기록하는 필드이다.
	requiredPower	해당 장치가 필요로 하는 전력을 기록하는 필드이다.
	requiredBandwidth	해당 장치가 필요로 하는 대역폭을 기록하는 필드이다.
	class	해당 장치의 USB Class를 기록하는 필드이다.

4.2.2.12. TransactionPayloadPool

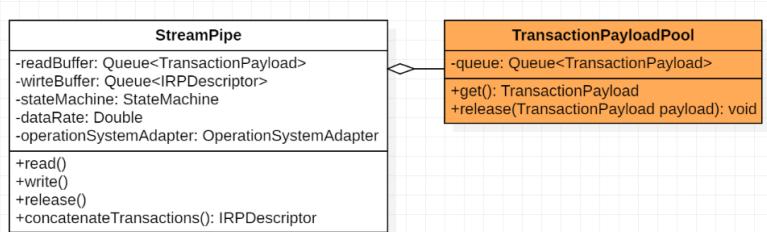


Figure 37 – TransactionPayloadPool 클래스

설명	Read 동작에서 Host Controller로부터 올라오는 데이터의 Payload를 담기 위한 메모리 공간들을 담은 메모리 풀 클래스이다.	
필드	queue	TransactionPayload 들을 미리 생성하고 보관하고 있는 메모리 풀 공간이다.
메서드	get	TransactionPayload 를 하나 내어주는 내용을 구현한다.
	release	다 쓴 TransactionPayload 를 반환받는 내용을 구현한다.

4.2.2.13. IRPDescriptorPool

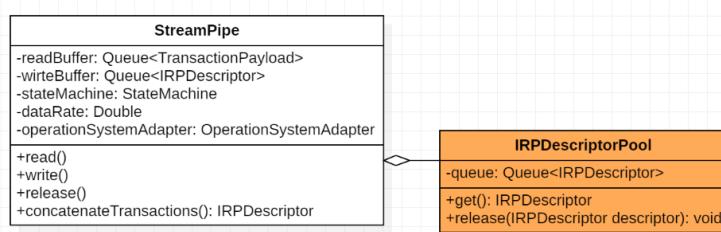


Figure 38 - IRPDescriptorPool 클래스

설명	Write 동작에서 Client SW로부터 내려오는 IRP의 정보를 담을 공간들을 담은 메모리 풀 클래스이다.	
필드	queue	IRPDescriptor 들을 미리 생성하고 보관해둔 메모리 풀이다.
메서드	get	IRPDescriptor를 하나 내어주는 내용을 구현한다.
	release	다 쓴 IRPDescriptor를 반환받는 내용을 구현한다.

4.2.2.14. IRPDescriptor

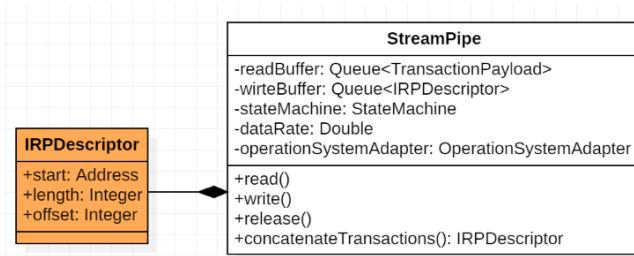


Figure 39 - IRPDescriptor 클래스

설명	Client SW로부터 전송된 IRP의 정보를 담은 클래스이다. 구조체 형태의 데이터 클래스이다. Transaction으로 나누어져 전송될 때마다 업데이트된다.	
필드	start	Client SW로부터 전달받은 IRP의 메모리 시작 주소를 기록하는 필드이다.
	length	Client SW로부터 전달받은 IRP의 데이터 길이를 기록하는 필드이다.
	offset	IRP가 Transaction으로 잘려 전송되면 이동하는 offset이다.

4.2.2.15. TransferType

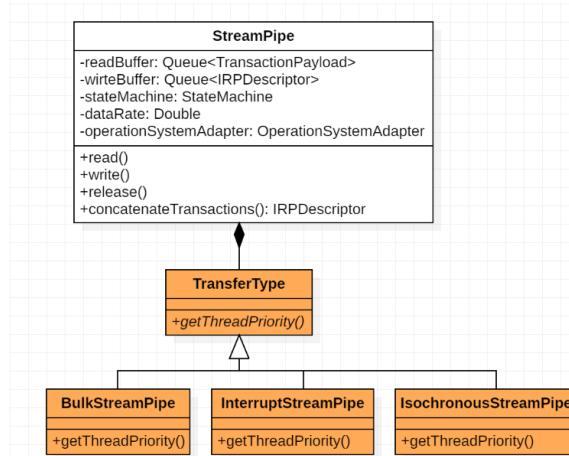


Figure 40 - TransferType 클래스

설명	StreamPipe의 전송 타입에 관한 클래스이다. 다형성을 이용해 StreamPipe의 쓰레드 priority를 도출한다.
메서드	getThreadPriority 쓰레드 priority를 도출한다.

4.2.2.16. LedControllerAdapter

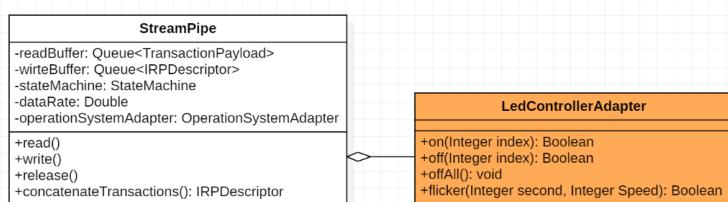


Figure 41 - LedControllerAdapter 클래스

설명	Client SW로부터 전송된 IRP의 정보를 담은 클래스이다. 구조체 형태의 데이터 클래스이다. Transaction으로 나누어져 전송될 때마다 업데이트된다.	
메소드	on	특정 인덱스의 Led를 켜는 기능을 구현한다.
	off	특정 인덱스의 Led를 끄는 기능을 구현한다.
	offAll	모든 Led를 끄는 내용을 구현한다.
	flicker	주어진 시간 동안 주어진 속도로 Led를 점멸하는 기능을 구현한다.

4.2.2.17. ClassPermission

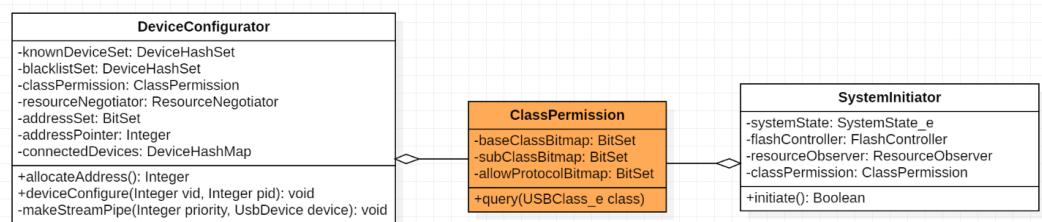


Figure 42 - ClassPermission 클래스

설명	허용 가능한 USB Class 정보를 담는 클래스이다. 비트맵 형태로 데이터를 기록하며 요청 시 정보를 제공한다.	
필드	baseClassBitmap	USB Specification에 정의된 base class의 내용이다.
	subClassBitmap	USB Specification에 정의된 sub class의 내용이다.
	allowProtocolBitmap	USB Specification에 정의된 protocol class의 내용이다.
메서드	query USB Class 정보를 받아 허용되는지를 확인하는 내용을 작성한다.	

4.2.2.18. ResourceObserver

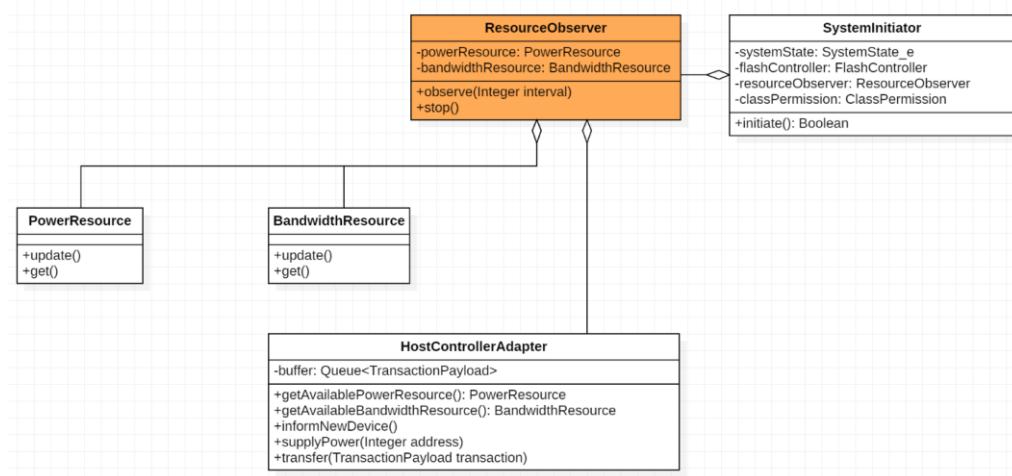


Figure 43 - ResourceObserver 클래스

설명	가용 리소스를 실시간으로 파악하기 위한 Watchdog 클래스이다.	
필드	powerResource	가용 리소스 기록을 위한 레퍼런스이다. 의존성이 주입된다.
	bandwidthResource	가용 리소스 기록을 위한 레퍼런스이다. 의존성이 주입된다.
메서드	observe	주기를 전달받아 주기마다 가용 리소스를 확인한다.
	stop	가용 리소스 확인을 중단한다.

4.2.2.19. ResourceNegotiator

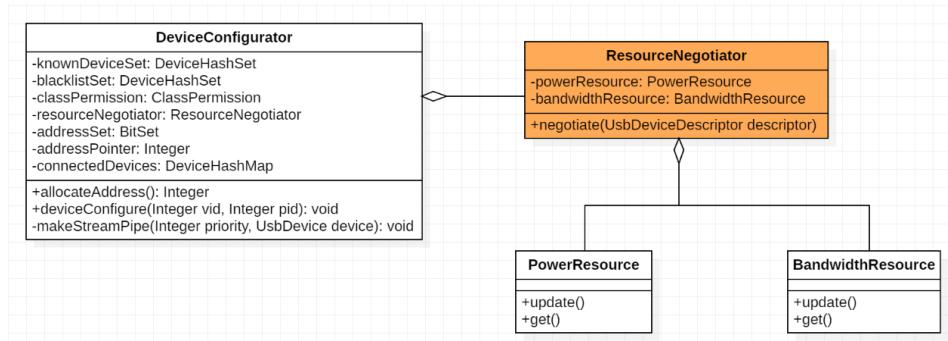


Figure 44 - ResourceNegotiator 클래스

설명	새로운 USB Device가 연결되면 그 장치가 요구하는 리소스와 가용 리소스에 대해 negotiation을 진행하고 적정 리소스를 도출하는 클래스이다.	
필드	powerResource	가용 리소스 확인을 위해 참조하는 레퍼런스이다. 의존성이 주입된다.
	bandwidthResource	가용 리소스 확인을 위해 참조하는 레퍼런스이다. 의존성이 주입된다.
메서드	negotiate	새 장치가 요구하는 리소스와 가용 리소스를 통해 적정 리소스를 도출하는 로직이 작성되는 자리이다.

4.2.2.20. HostControllerAdapter

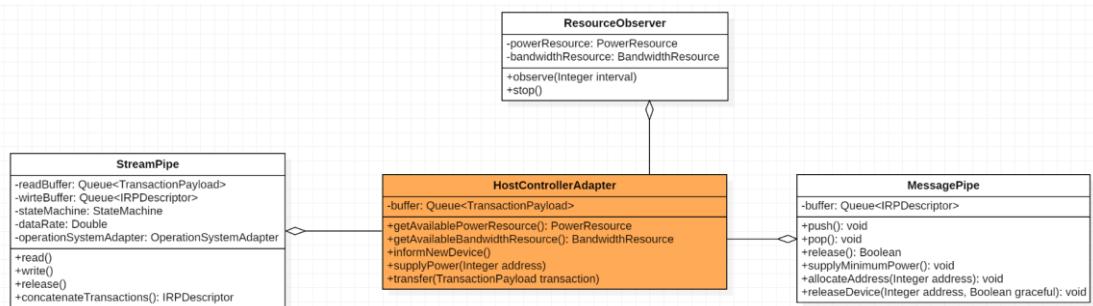


Figure 45 - HostControllerAdapter 클래스

설명	Host Controller와의 인터페이스를 담당하는 클래스이다. 정해진 operation을 여러 Host Controller에서 지원할 수 있도록 추상화한다.	
필드	buffer	실제로 Host Controller로 전송되는 Transaction을 담는 버퍼이다. 스케줄링된 StreamPipe가 이곳에 Transaction을 넣는다. Queue 형식의 자료구조이다.
메서드	getAvailablePowerResource	가용 가능한 전력을 파악하여 전달하는 내용을 구현한다.
	getAvailableBandwidthResource	가용 가능한 대역폭을 파악하여 전달하는 내용을 구현한다.
	informNewDevice	새로운 장치가 연결되면 호출되는 메서드로 DeviceConfigurator를 참조한다.
	supplyPower	최종적으로 결정된 전력을 공급하는 내용을 구현한다.
	transfer	Buffer에 쌓인 Transaction을 실제로 전송하는 내용을 구현한다.

4.3. Work Assignment View

본 시스템의 개발팀 구성은 아래 그림과 같다. Melvin E. Conway가 제안한 Conway의 법칙에 의하면 시스템의 구조는 설계하는 조직의 커뮤니케이션 구조와 닮는다고 한다. 그러한 관점에서 4.1에서 제시한 Module View의 패키지에 근거하여 조직 구조를 구성하였다. Scenario Management Package의 경우 볼륨이 상대적으로 적으므로 어느정도 연관이 있는 Interface 그룹에 포함하였다.

본 시스템을 개발하는 조직은 USB Host Driver 개발팀이며 하위 조직으로 Transfer 그룹, Interface & Scenario 그룹, Device Management 그룹, Resource Management 그룹 이렇게 4개의 그룹이 있다. Transfer 그룹은 실제 전송 로직을 담당하는 Pipe 파트와 그를 위한 Memory Pool 파트가 있다. Interface & Scenario 그룹은 외부 모듈과의 추상화 업무를 담당하는 Interface 파트와 시스템 초기화 등 시나리오 모듈을 담당하는 Scenario 파트가 있다. Device Management 그룹은 연결된 USB Device 정보를 관리하는 Device Information 파트와 연결 이력 및 권한을 관리하는 Device Permission 파트가 있다. 마지막으로 Resource Management 그룹은 단일 파트로 가용 리소스에 대한 수집 모듈 및 Negotiation 모듈을 담당한다.

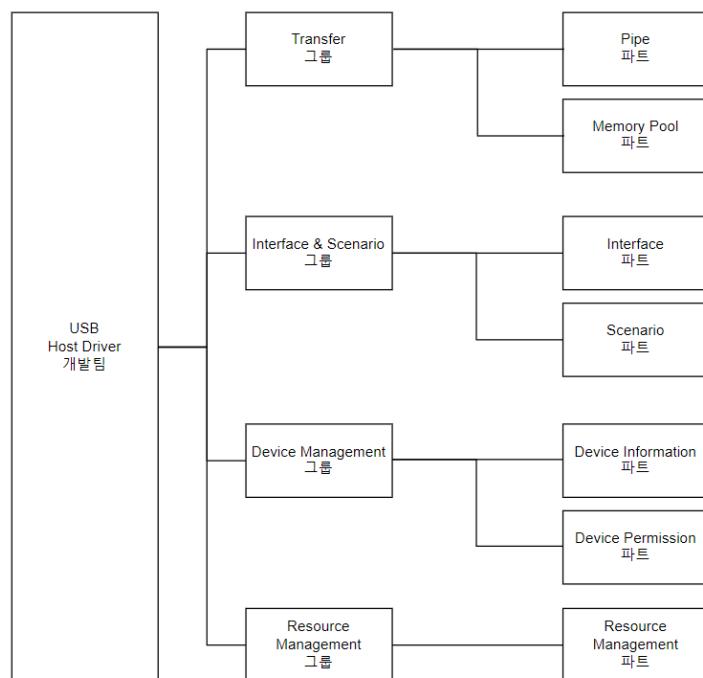


Figure 46 - Work Assignment View

부록

A. 도메인 모델.....	55
B. 품질 시나리오	65
C. 품질 시나리오 분석	67
D. 후보 구조.....	70
E. 후보 구조 평가.....	102
F. 최종 구조 설계	113

A. 도메인 모델

본 과제에서 USB Hub Driver에 대한 도메인 모델은 아래 그림과 같다. 본 도메인 모델은 Boundary/Control/Entity 모델로 작성되었다. Boundary는 외부의 Interface와 상호 작용하고, Control은 Boundary와 Entity를 연결해주며, Entity는 시스템에 필요한 데이터를 저장한다.

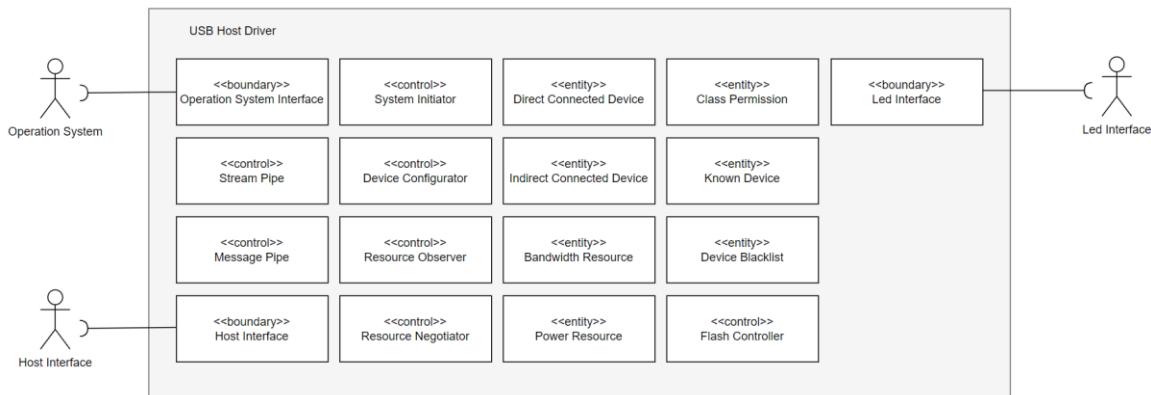


Figure 47 - Domain Model

A1. Domain Model List

Domain Model	Category	Description
Operation System Interface	Boundary	Operation System으로부터 내려오는 System Call을 받고 해석하고 전파하는 역할을 담당하는 도메인 모델이다.
Stream Pipe	Control	Host 시스템과 USB Device 간의 데이터 전송을 담당하는 도메인 모델이다. USB Specification에서 Host의 Client SW와 USB Device의 Function 간의 Endpoint에 해당하는 역할이다.
Message Pipe	Control	Host 시스템과 USB Device 간의 제어 신호를 전송하는 도메인 모델이다. USB Specification에서 Endpoint Zero에 해당하는 역할이다.
Led Interface	Boundary	LED Controller와 통신하는 LED Interface에 대한 Boundary 도메인 모델이다. LED Interface와 인터럽트를 주고받으며 명령을 해석하거나 명령으로 변환하는 역할을 한다.

Host Interface	Boundary	USB Device와 물리적인 통신을 담당하는 Host Controller 와의 인터페이스이다. Host Controller 장치에 대한 추상화가 이루어진다.
Flash Controller	Control	Host 시스템 내부의 Flash로부터 데이터를 쓰고 읽는 역할을 담당하는 도메인 모델이다.
System Initiator	Control	시스템의 시작, 종료 절차를 관리하는 도메인 모델이다.
Device Configurator	Control	USB Device에 대한 열거 절차를 담당하는 도메인 모델이다. Host와 USB Device가 서로 인지할 수 있도록 Configuration 절차를 주도하며 Device Entity를 생성하고 Pipe를 개설한다.
Resource Observer	Control	Host PC와 Host Controller의 상태를 체크하여 사용 가능한 자원에 대한 정보를 주기적으로 얻어와 저장하는 역할을 하는 도메인 모델이다.
Resource Negotiator	Control	사용 가능한 자원을 조회하고 USB Device에서 요청하는 자원에 대한 사용 범위를 결정하는 도메인 모델이다.
Bandwidth Resource	Entity	사용 가능한 대역폭 자원에 대한 정보를 저장하는 도메인 모델이다.
Power Resource	Entity	사용 가능한 전력 자원에 대한 정보를 저장하는 도메인 모델이다.
Direct Connected Device	Entity	USB 슬롯에 직접 연결된 장치를 추상화한 도메인 모델이다. 직접 연결된 장치는 LED 장치와 Mapping되므로 플로우가 달라져 간접 연결된 장치와 구분하였다. 본 시스템은 직접적으로 USB Device와 연결되지 않기 때문에 USB Device에 대한 정보를 저장하는 역할을 한다.
Indirect Connected Device	Entity	USB 슬롯에 연결된 USB Hub를 통해 연결된 장치를 추상화한 도메인 모델이다. 본 시스템은 직접적으로 USB Device와 연결되지 않기 때문에 USB Device에 대한 정보를 저장하는 역할을 한다.
Class Permission	Entity	허용 가능한 USB Class를 저장하는 도메인 모델이다. Host 시스템에게 질의하여 수용 가능한 USB Class를 얻어와 해당 Entity에 저장한다.
Known Device	Entity	연결된 적 있는 USB Device에 대한 식별자를 저장하는 도메인 모델이다. USB Device의 VID, PID를 조합하여 USB Device의 고유한 값을 만들어 저장한다.
Device Blacklist	Entity	연결 거부된 USB Device의 식별자를 저장하여 향후 연결을 거부하는데 쓰이는 도메인 모델이다.

A2. Sequence Diagram

A2.1. 시스템 시작

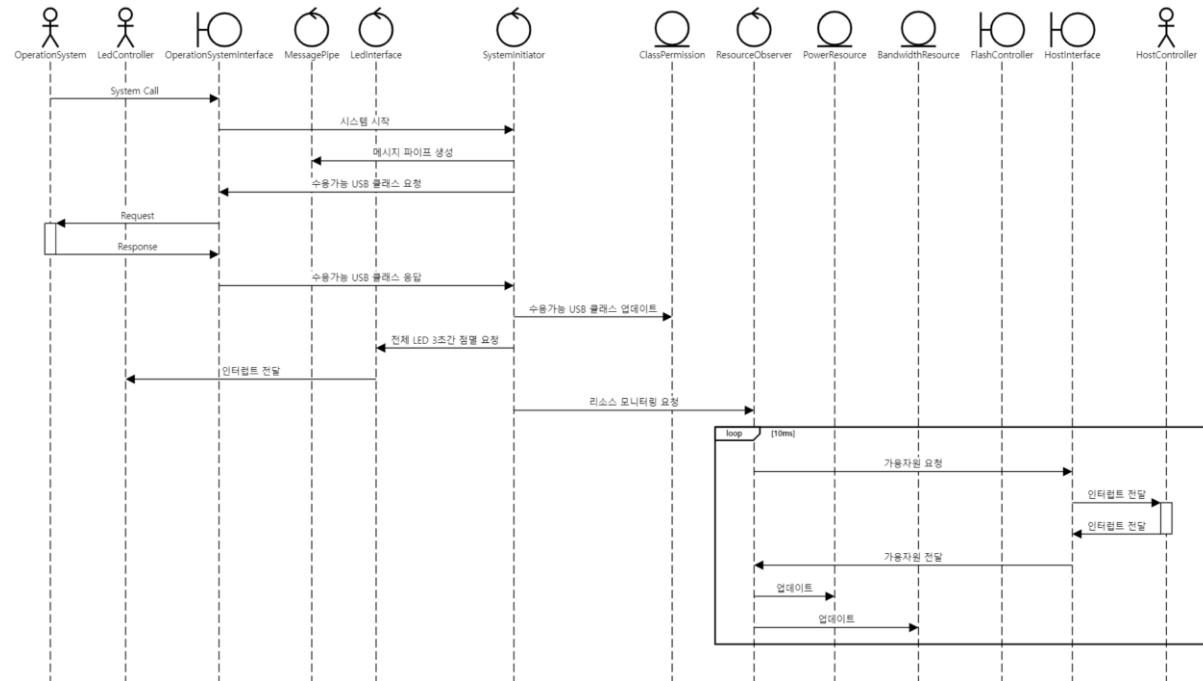


Figure 48 - 시스템 시작

A2.2. 시스템 종료

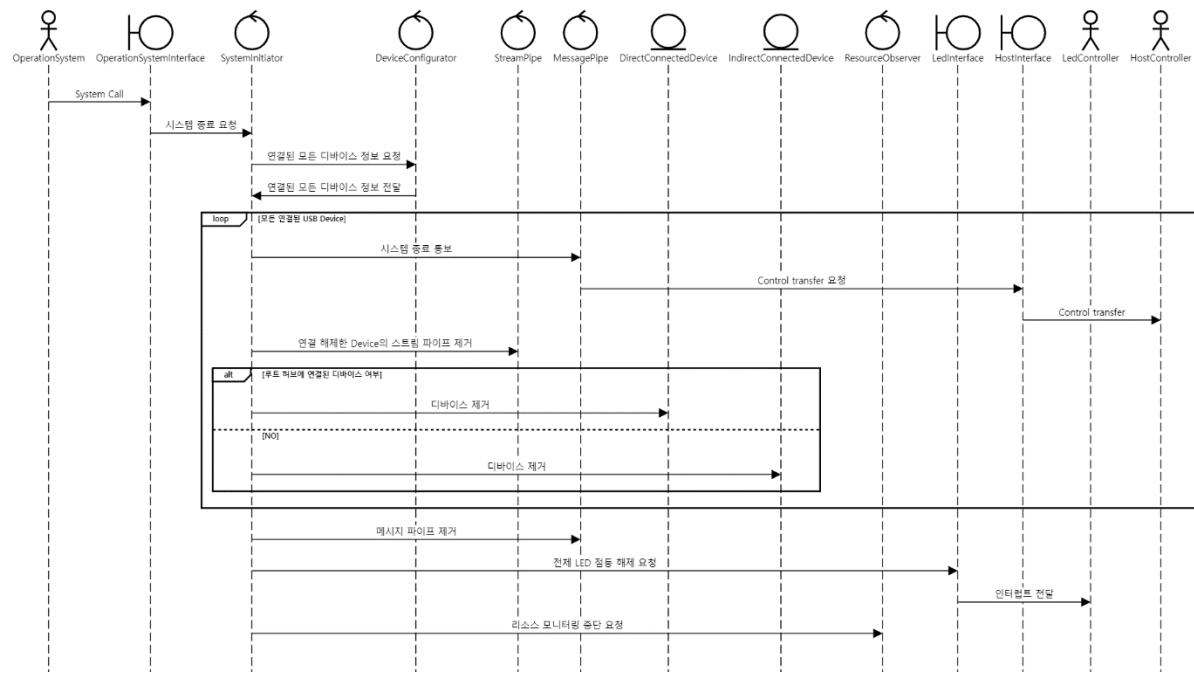


Figure 49 - 시스템 종료

A2.3. USB Device 연결

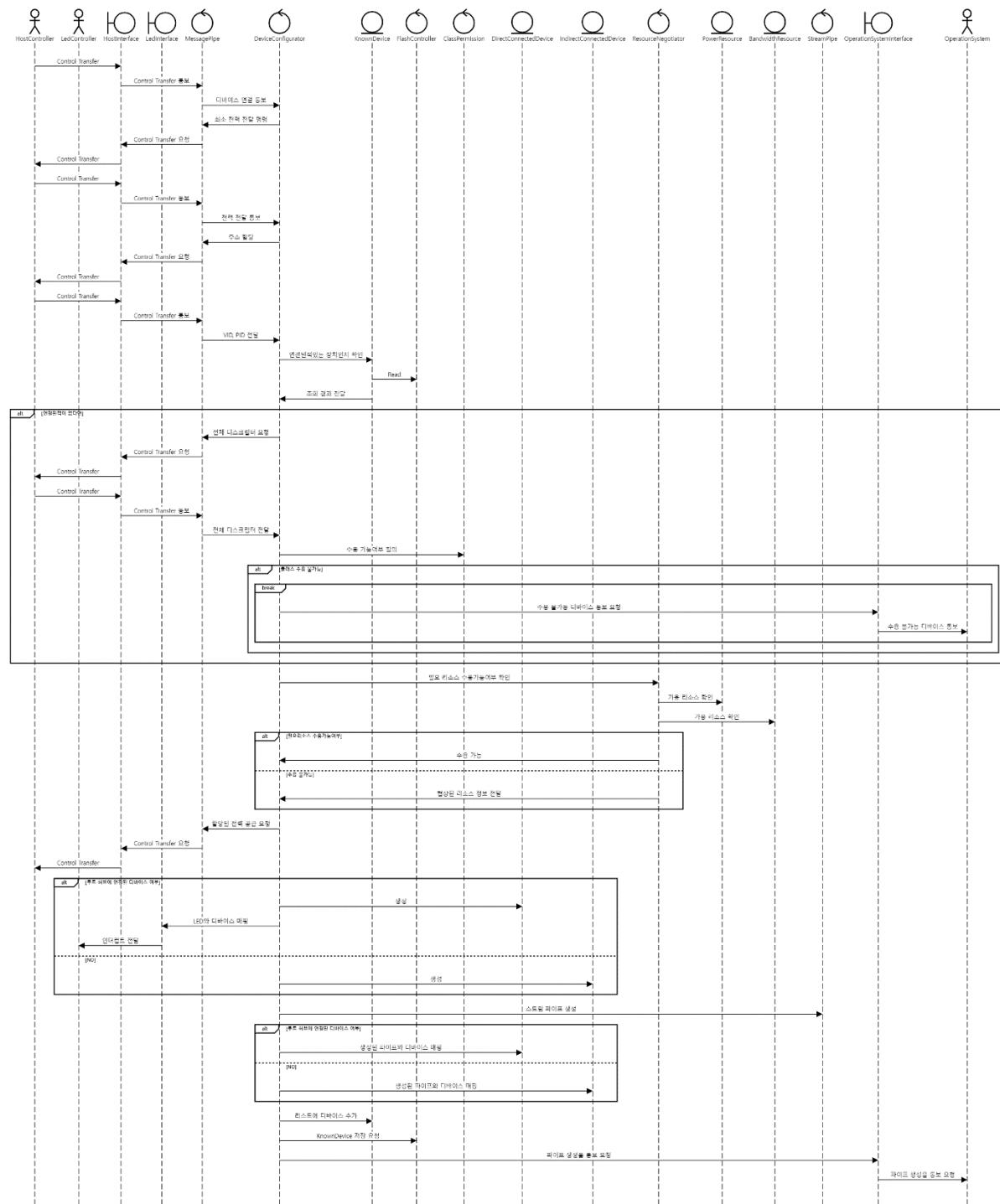


Figure 50 - USB Device 연결

A2.4. USB Device 연결 해제 요청

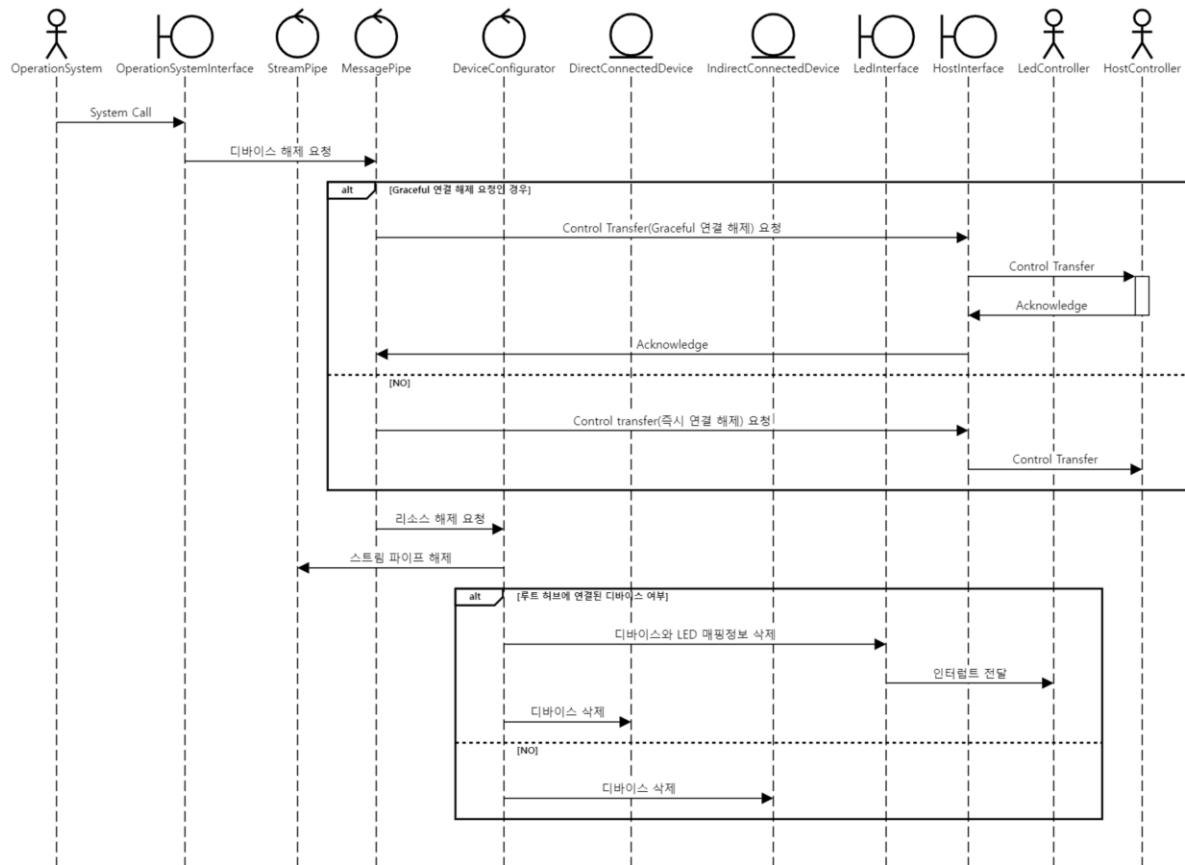


Figure 51 - USB Device 연결 해제 요청

A2.5. USB Device 연결 해제

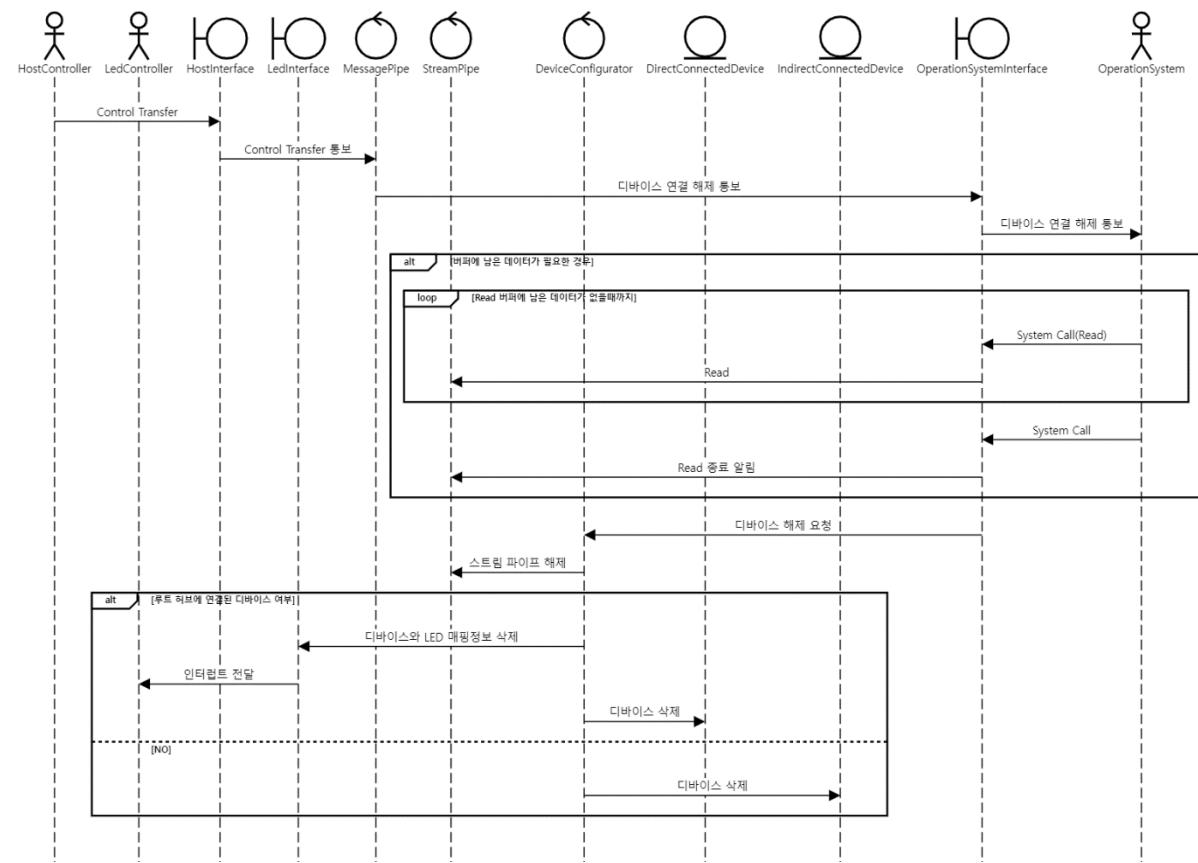


Figure 52 - USB Device 연결 해제

A2.6. 연결된 USB Device 정보 요청

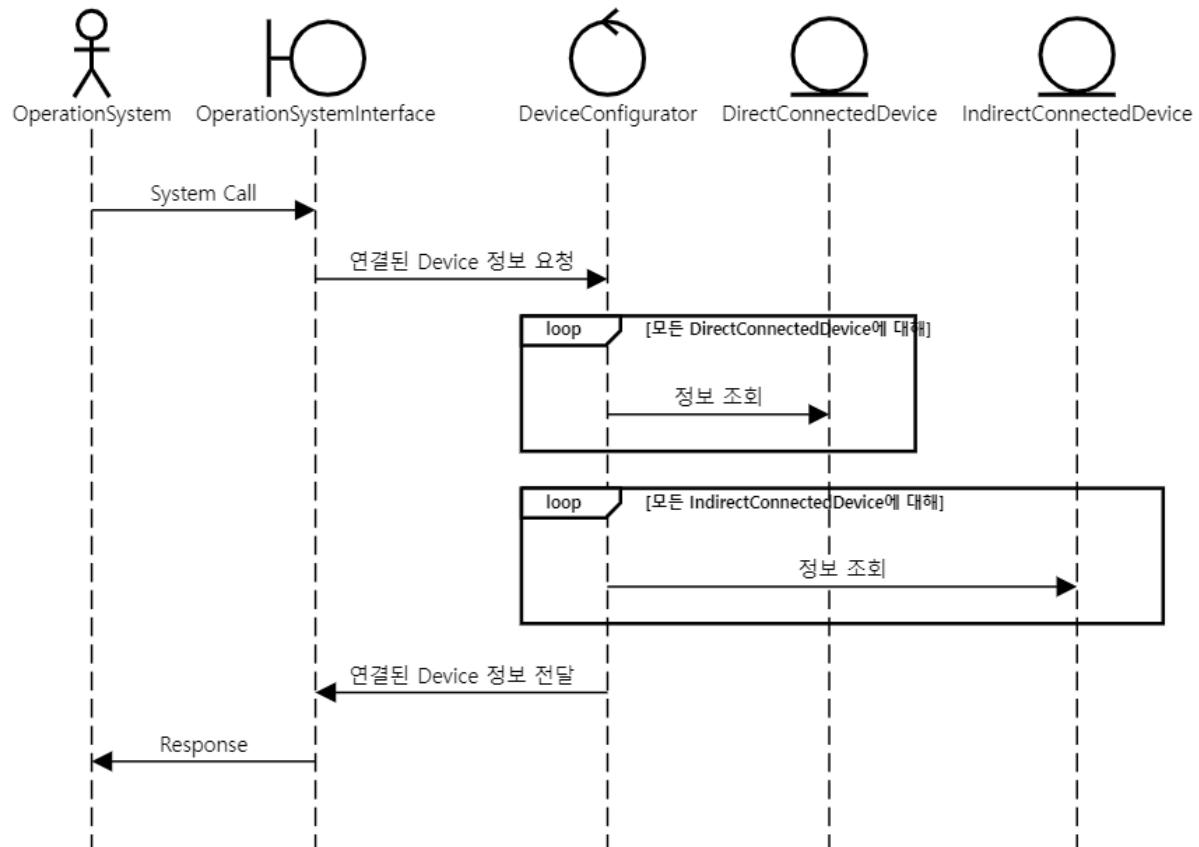


Figure 53 - 연결된 USB Device 정보 요청

A2.7. Host로 데이터 전송

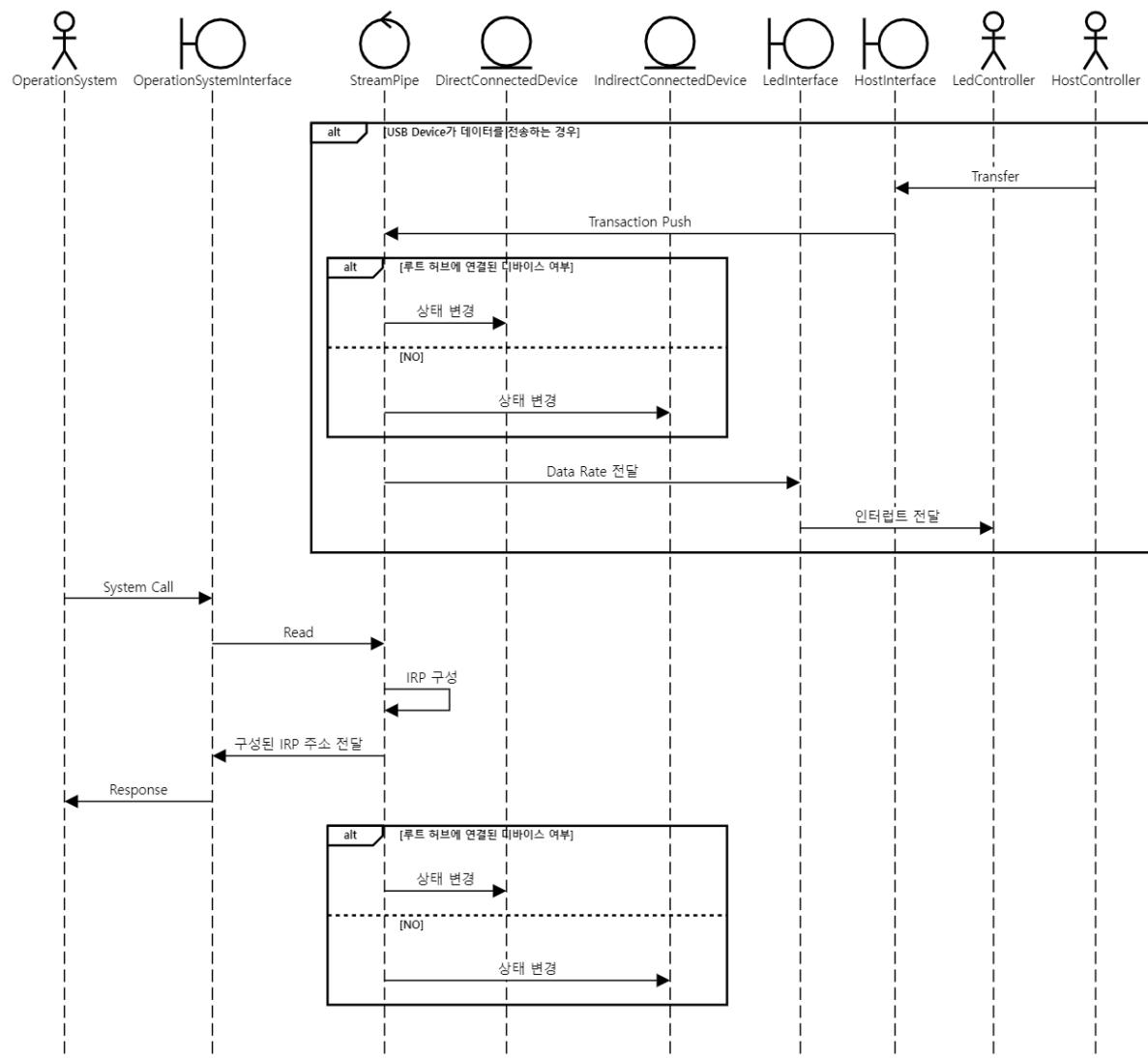


Figure 54 - Host로 데이터 전송

A2.7. USB Device로 데이터 전송

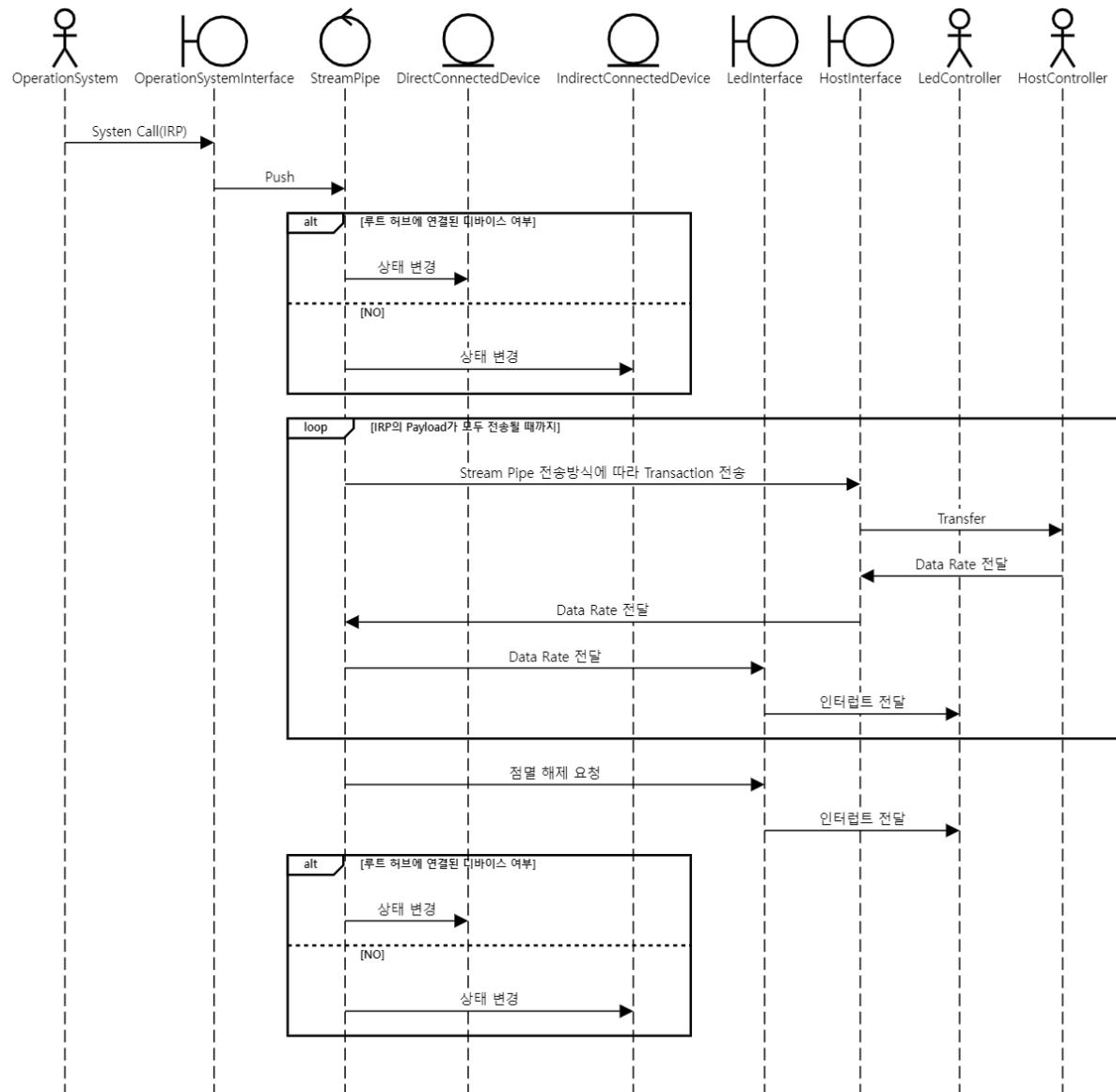


Figure 55 - USB Device로 데이터 전송

B. 품질 시나리오

QS_01	성능	시스템 부팅 시간
	측정 방법	[부팅 시간] = [LED 장치의 최초 점멸 시각] – [시스템에 전원이 공급되는 시각] [부팅 시간] <= 100ms
QS_02	성능	데이터 전송 속도
	측정방법	[전송 속도] = [Destination으로 전송된 Bit] / 1000ms [전송 속도]는 빠를수록 좋다.
QS_03	기능 정확성	Low-Speed를 사용하는 USB Class 장치의 전송 속도
	측정 방법	[전송 속도] = [Destination으로 전송된 Bit] / 1000ms [전송 속도] <= 1.5Mbps
QS_04	성능	디바이스 설정 시간
	측정방법	[디바이스 설정 시간] = [Pipe가 개설되는 시각] – [USB Device가 물리적으로 삽입되는 시각] [디바이스 설정 시간] <= 500ms
QS_05	성능	등시성(Isochronism)이 요구되는 USB Class 장치의 전송 속도 일관성
	측정방법	[전송 속도의 표준편차] = [단위 시간동안 Host Controller HW를 통해 측정되는 속도의 표준 편차] [전송 속도의 표준편차]가 작을수록 좋다.
QS_06	성능	디바이스 정보 조회 시간
	측정방법	[디바이스 정보 조회 시간] = [OS가 USB Device의 정보를 요청하는 시각] – [시스템이 USB Device의 정보를 OS에게 전달하는 시각] [디바이스 정보 조회 시간]가 작을수록 좋다.
QS_07	변경용이성	전송 방식 변경
	측정방법	[전송 방식 변경시 변경이 필요한 컴포넌트의 수] [전송 방식 변경시 변경이 필요한 컴포넌트의 수]는 적을수록 좋다.
QS_08	변경용이성	외부 모듈 교환
	측정방법	[외부 모듈 교환시 변경이 필요한 컴포넌트의 수] [외부 모듈 교환시 변경이 필요한 컴포넌트의 수]는 적을수록 좋다.
QS_09	변경용이성	지원가능한 USB Class 확장
	측정방법	[지원가능한 USB Class 확장시 변경이 필요한 컴포넌트의 수] [지원가능한 USB Class 확장시 변경이 필요한 컴포넌트의 수]는 적을수록 좋다.
QS_10	변경용이성	Host Controller의 USB 지원 버전 확장
	측정방법	[USB 지원 버전 확장시 변경이 필요한 컴포넌트의 수]

		[USB 지원 버전 확장시 변경이 필요한 컴포넌트의 수]는 적을수록 좋다.
QS_11	가용성	동시에 연결 가능한 USB Device의 수
	측정방법	[동시에 연결 가능한 USB Device의 수] [동시에 연결 가능한 USB Device의 수]는 많을수록 좋다.
QS_12	안정성	시스템의 CPU 점유율 표준편차
	측정방법	[CPU 점유율 표준편차] = [IDLE 상태에서 100MB 전송시 OS를 통해 측정되는 CPU 점유율의 표준편차] [CPU 점유율 표준편차]는 낮을수록 좋다
QS_13	안정성	USB Device가 요구한 전력 공급 비율
	측정방법	[USB Device가 요구한 자원 할당 비율] = [Resource Negotiator에서 요구한만큼 할당하는 경우] / [USB Device가 연결된 횟수] [USB Device가 요구한 자원 할당 비율]가 높을수록 좋다.

C. 품질 시나리오 분석

품질 시나리오		중요도	복잡도	선정 결과
성능	QS_01. 시스템 부팅 시간	M	H	NFR_02
	QS_02. 데이터 전송 속도	H	H	QA_01
	QS_04. 디바이스 설정 시간	H	M	NFR_01
	QS_05. 등시성(Isochronism)이 요구되는 USB Class 장치의 전송 속도 일관성	M	M	
	QS_06. 디바이스 정보 조회 시간	L	M	QA_07
	QS_03. Low-Speed를 사용하는 USB Class 장치의 전송 속도	M	L	
변경용이성	QS_07. 전송 방식 변경	H	H	QA_03
	QS_08. 외부 모듈 교환	L	M	QA_06
	QS_09. 지원 가능한 USB Class 확장	H	M	QA_02
	QS_10. Host Controller의 USB 지원 버전 확장	M	H	QA_04
가용성	QS_11. 동시에 연결 가능한 USB Device의 수	M	M	QA_05
안정성	QS_12. 시스템의 CPU 점유율 표준편차	L	H	
	QS_13. USB Device가 요구한 전력 공급 비율	H	M	

C1. 품질 시나리오별 비기능 요구사항 및 품질 속성 선정 근거

QS_01 시스템 부팅 시간은 운영체제 전체의 부팅 시간에 영향을 줘 사용자가 느끼는 성능에 영향을 줄 수 있다고 판단하여 비기능 요구사항으로 선정하였다. 부팅시 1회만 실행되는 시나리오 이므로 중요도는 M으로 정했으나 구현을 위한 복잡도는 부팅시간을 단축하는 작업이 간단하지 않아 H로 정하였다.

QS_02 데이터 전송 속도는 USB 시스템에서 가장 중요한 성능 요소라고 생각하여 품질 속성으로 선정하였다. 따라서 중요도도 H로 정하였고, 성능을 이끌어 내기 위한 별도의 아이디어나 알고리즘이 필요할 것으로 예상되어 복잡도도 H로 정하였다.

QS_03 Low-Speed를 사용하는 USB Class 장치의 전송 속도의 경우에는 이를 담당하는 주체가 Host Driver가 아닌 Client SW가 적절하다고 판단하여 선정에서 제외하였다.

QS_04 디바이스 설정 시간은 사용자 USB Device를 삽입하고 동작하기 위해 매우 중요한 성능 요소라 생각하여 비기능 요구사항으로 선정하였다. 따라서 중요도도 H로 정하였고, 데이터 전송 속도의 개선보다는 복잡도가 낮다고 생각되어 복잡도는 M으로 선정하였다.

QS_05 등시성(Isochronism)이 요구되는 USB Class 장치의 전송 속도 일관성 역시 QS_03과 유사하게 주체가 Client SW가 되어야 적절하다고 판단하여 선정에서 제외하였다.

QS_06 디바이스 정보 조회 시간의 경우 빈번하게 발생하지는 않지만 성능 측면에서 체감되는 요소라고 생각하여 우선순위가 낮은 품질 속성으로 선정하였다. 따라서 중요도는 L로 정하였고, 구현 측면에서도 복잡도가 높지 않을 것으로 예상되어 복잡도는 M으로 선정하였다.

QS_07 전송 방식 변경은 USB Specification에서 정의된 전송방식이 Spec의 버전업으로 추가되는 경우 변경에 대응할 수 있어야 하므로 변경용이성을 고려하여 품질 속성으로 선정하였다. 전송방식은 구조에 영향을 미칠 수 있는 요소인데 적은 변경으로 이를 대응하는 일은 쉽지 않을 것으로 예상되어 중요도와 복잡도를 모두 H로 선정하였다.

QS_08 외부 모듈 교환은 하드웨어의 변경을 통해 충분히 있을 수 있는 시나리오이므로 변경용이성을 고려하여 품질속성으로 선정하였다. 하드웨어 변경과 인터페이스 변경이라는 보편적인 시나리오를 고려할 때 중요도는 L, 복잡도는 M으로 선정하였다.

QS_09 지원 가능한 USB Class 확장은 과제의 요구사항이므로 품질요소로 선정하고 중요도 또한 H로 정하였다. 또한 구현의 복잡도를 고려할 때 복잡도는 M을 선정하였다.

QS_10 Host Controller의 USB 지원 버전 확장은 역시나 발생할 수 있는 시나리오이면서 다수의 컴포넌트 수정을 야기할 수 있는 시나리오이므로 품질요소로 선정하였다. 하나의 제품에서 빈번하게 발생하는 시나리오는 아니므로 중요도는 M, 변경용이성을 지키는 구조의 복잡도를 고려할 USB Host Driver

때 복잡도는 H로 정하였다.

QS_11 동시에 연결 가능한 USB Device의 수는 사용자가 USB Hub 제품을 사용할 때 고려되어야 할 시나리오라 판단되어 품질요소로 선정하였다. 소프트웨어가 해당 요소에 미칠 영향의 정도를 고려할 때, 중요도와 복잡도를 M으로 선정하였다.

QS_12 시스템의 CPU 점유율 표준편차의 경우 시스템이 주체가 되어 컨트롤 할 수 있는 시나리오가 아니라고 판단되어 선정에서 제외하였다.

QS_13 USB Device가 요구한 전력 공급 비율의 경우 시스템이 주체가 되어 컨트롤 할 수 있는 시나리오가 아니라고 판단되어 선정에서 제외하였다.

D. 후보 구조

D1. QA_01 데이터 전송 속도

Host와 USB Device 간의 데이터 전송 속도를 높이기 위해서 본 시스템은 두가지 주제에 대해 후보 구조를 설계하였다.

1. 전송 단위에 대한 메모리 할당
2. 전송의 병렬성

Host Controller는 Transaction을 순차적으로 처리한다. 이러한 제약조건 내에서 성능 품질 요소를 위한 최적의 설계를 고려할 때, 매우 빈번하게 일어나는 동작인 전송에 대해 메모리 공간의 할당 및 해제와 같은 비용이 큰 동작이 첫번째로 고려되었다. 또한 Host Controller가 유휴 시간 없이 높은 가용성으로 계속 일할 수 있도록, Client SW로부터 병렬적으로 전송 요청을 받아 관리할 수 있는 구조가 최적 구조로서 두번째로 고려되었다. 이러한 특성을 고려하여 데이터 전송 속도를 높이기 위한 후보 구조를 알아본다.

D1.1. 전송 단위에 대한 메모리 할당 후보 구조

D1.1.1. CA_01. IRP의 주소를 그대로 사용하여 추가적인 메모리 할당 없이 Transaction을 구성하는 방법

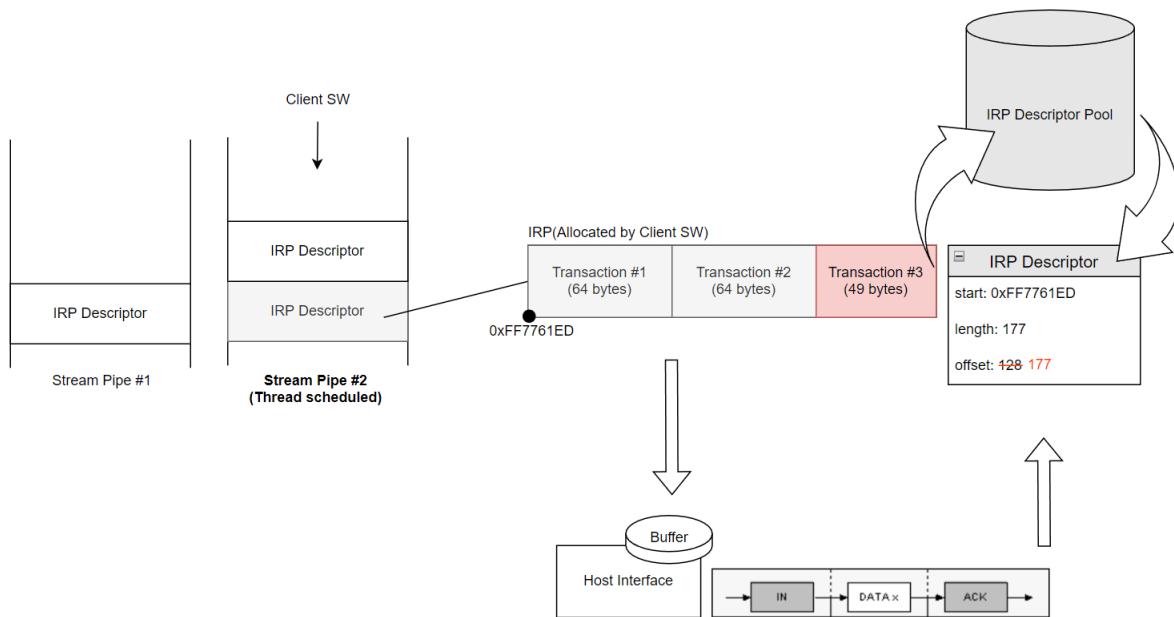


Figure 56 – IRP의 주소를 그대로 활용하는 구조

데이터 전송 시에 시스템은 Client SW로부터 받은 IRP를 segmentation해서 Transaction으로 보내야 한다. 이때 Transaction의 Payload를 위한 메모리 공간을 할당하지 않고 IRP의 주소 값을 그대로 이용하는 구조가 CA_01이다.

이를 위해 시스템은 IRP를 Client SW로부터 수신하면 전송한 지점을 기록하기 위한 IRP Descriptor를 생성하고 Transaction이 이루어질 때마다 offset을 기록한다. 전송 길이는 Stream Pipe에 정의된 통신 방식에 따라 적절한 크기의 Frame만큼 데이터를 전송한다. IRP를 구성하는 Transaction이 모두 전송되면 Queue형태인 Stream Pipe에서 IRP를 Pop하고 Client SW에게 IRP 전송이 완료되었음을 통지한다.

이 구조는 메모리의 할당 및 해제에 큰 성능 비용이 발생한다는 점에서 착안하였다. 따라서 Transaction의 Payload를 위한 메모리를 할당하지 않는 것이 핵심이며, 할당이 필요한 것은 고정된 크기의 작은 IRP Descriptor 뿐이다. IRP는 Transaction에 비해 개수가 적으므로(Transaction이 USB Host Driver

IRP의 Segment이므로) 상대적으로 적은 비용이 발생하며, 미리 할당된 Pool을 통해 비용을 더 줄일 수 있다. 또한 시스템 내에서 IRP의 데이터를 Transaction의 Payload로 복사하는 비용을 줄일 수 있다.

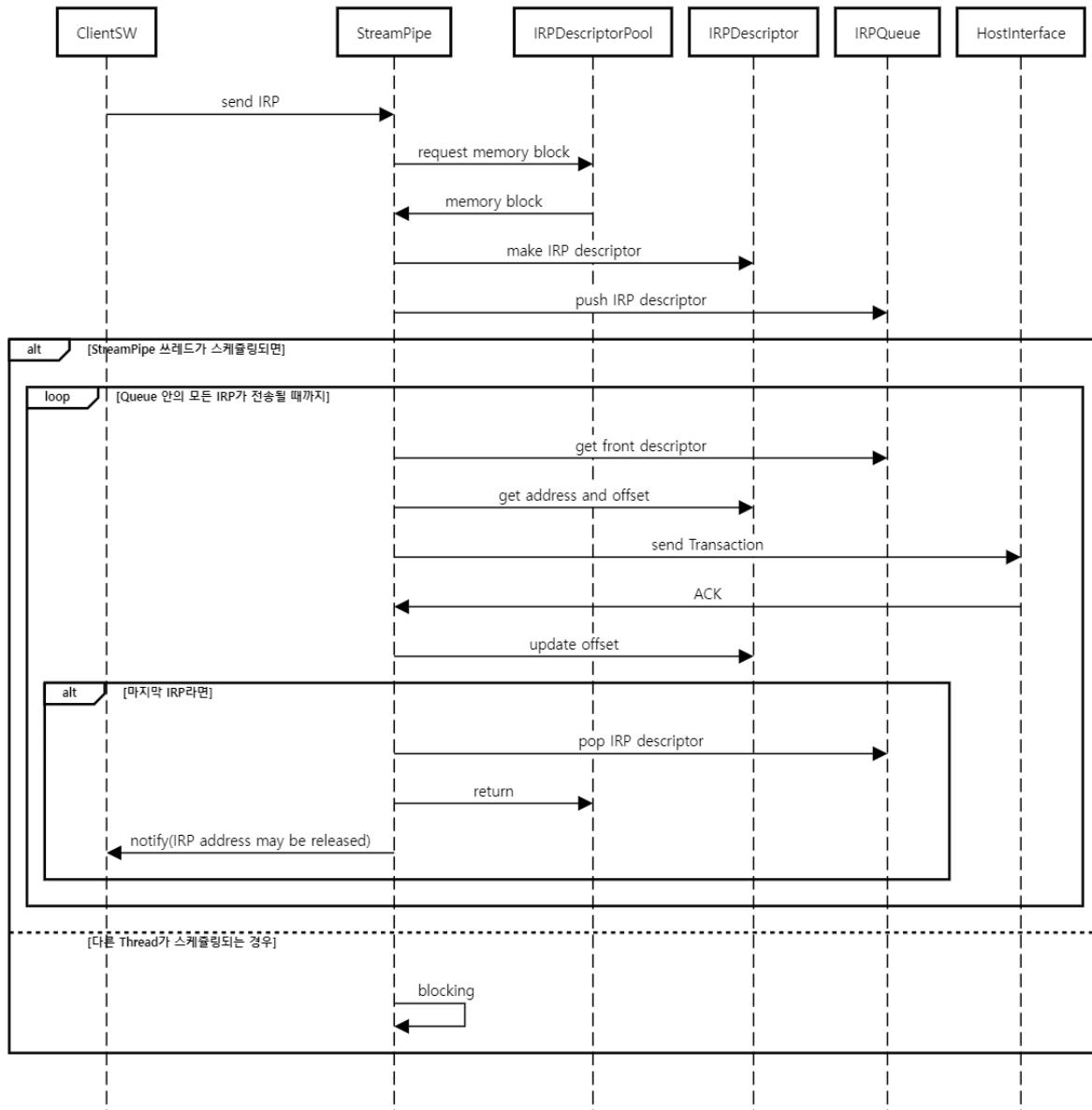


Figure 57 - CA_01의 동작

D1.1.2. CA_02. 메모리 Pool을 이용하여 새로운 Transaction을 구성하는 방법

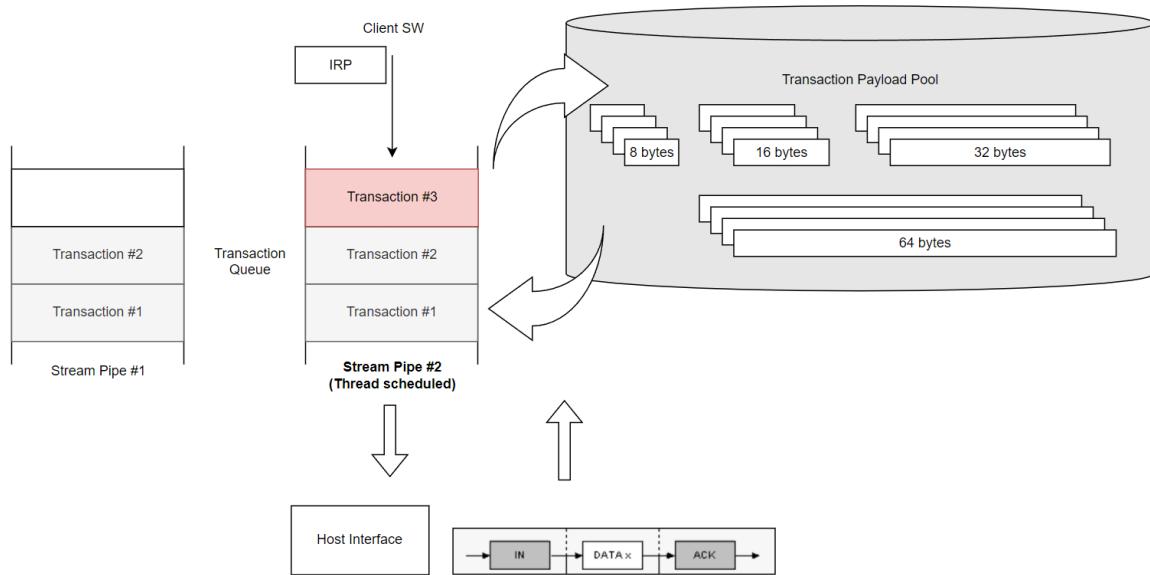


Figure 58 - 메모리 Pool을 이용하는 방법

Client SW로부터 IRP를 수신하면 시스템은 Stream Pipe가 가지고 있는 전송 타입에 따라 적절한 Frame 크기로 데이터를 Segmentation해서 별도의 Transaction Queue에 넣는 구조가 CA_02이다.

CA1에서 언급했듯 메모리의 할당 및 해제에는 큰 성능 비용이 발생하므로 Frame 크기에 따라 여러 memory block을 미리 할당해 둔 Transaction Payload Pool을 가진다. 만약 미리 할당된 Element가 모두 소진되면 그때부터는 Heap에 메모리를 할당하여 반환한다. Pool Element의 대여/반납과 Heap에서 할당된 메모리의 일관된 처리를 위해 구현단계에서 interface에 대한 추상화가 요구된다.

Transaction Queue의 요소들은 순차적으로 Host Interface를 통해 전송되며 전송이 완료되면 Pool Element를 반환한다. IRP의 마지막(Max Frame 사이즈가 아닌 Transaction)을 전송한 후에는 반납을 진행한 후 Client SW에게 전송이 완료되었음을 통지한다.

이 구조는 Payload를 복사해야 한다는 단점이 존재한다. 이에 따른 성능 저하를 보완하기 위해 DMA(Direct Memory Access) 장치를 이용한 복사를 활용할 수도 있다.

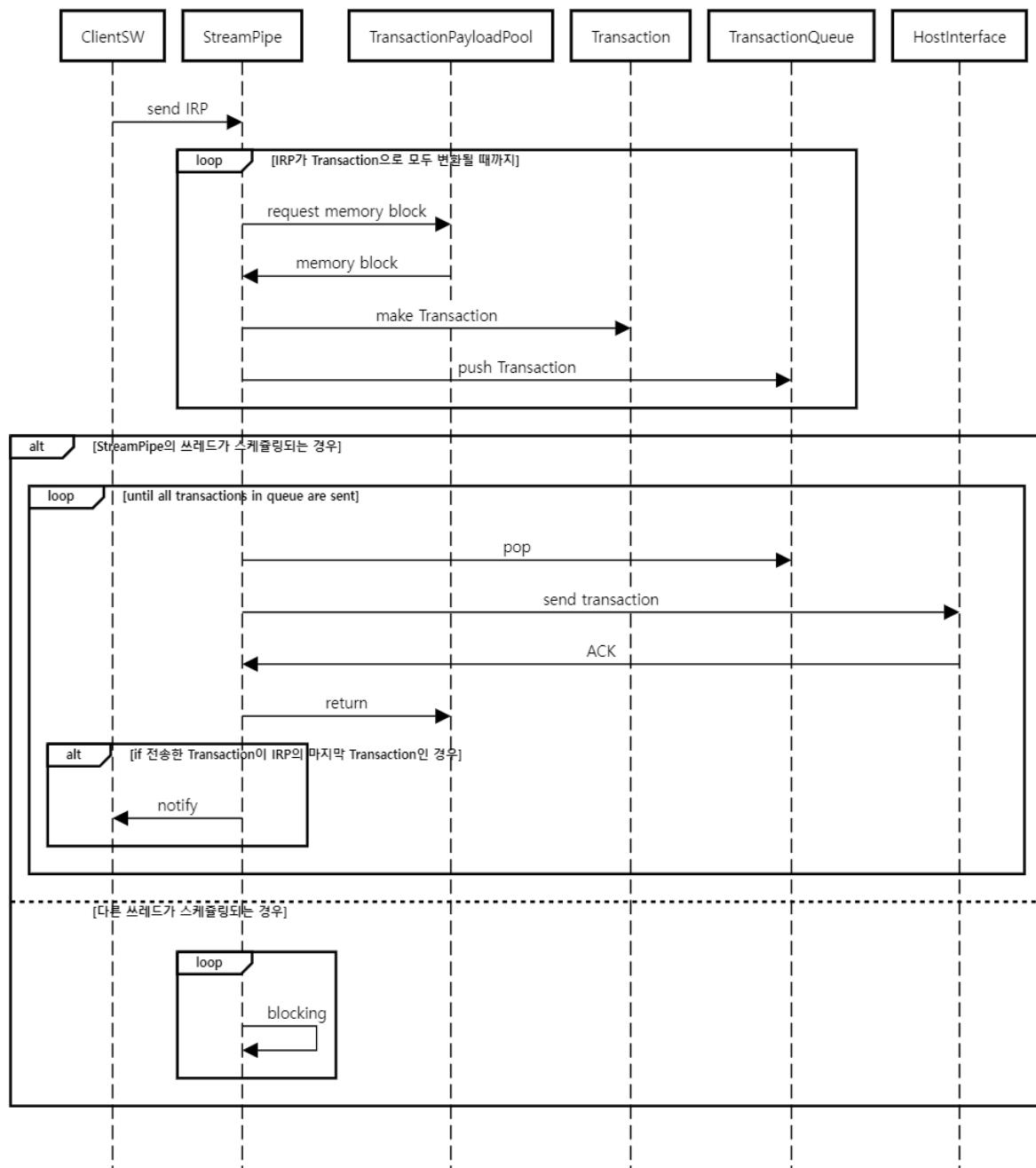


Figure 59 - CA_02의 동작

D1.1.3. CA_03. USB Device가 보내는 데이터를 메모리 Pool을 이용하여 Queueing하다가 Host에서 요청이 오면 Concatenate해서 전송하는 방법

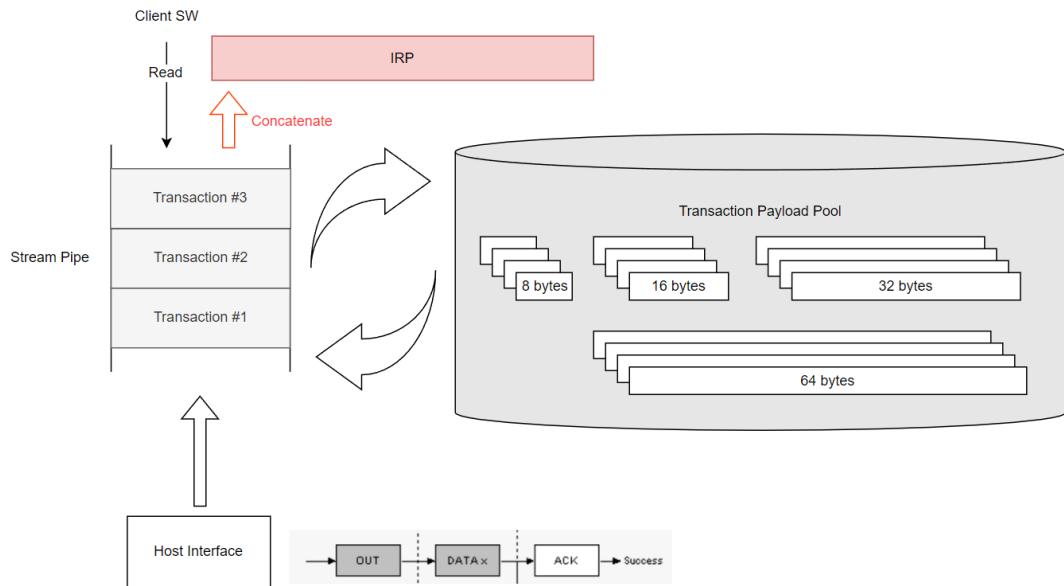


Figure 60 - 최종적으로 한번 Concatenate하는 방법

CA_02처럼 메모리 Pool을 이용해서 USB Device가 보내는 Transaction을 Queueing했다가 Client SW로부터 Read 요청이 올 때 하나로 묶어 올려 보내는 방식이 CA_03이다.

USB Device가 보내는 데이터의 경우 데이터를 저장할 메모리 공간은 Host Driver가 가지고 있어야 하는 것으로 판단하였다. 따라서 Transaction이 오면 이를 Frame 크기에 따라 Transaction Payload Pool에서 꺼낸 메모리 공간에 저장을 한다.

Transaction 형태로 보관을 하는 이유는 Client SW로부터의 읽기 요청이 있을 때 쌓인 데이터가 하나의 IRP 형태로 올라가기 때문이다. Transaction들은 분산된 메모리 주소에 저장될 것이므로 하나로 Concatenate시켜서 하나의 메모리 주소로 만든 뒤 Descriptor와 함께 Client SW에게 전달한다.

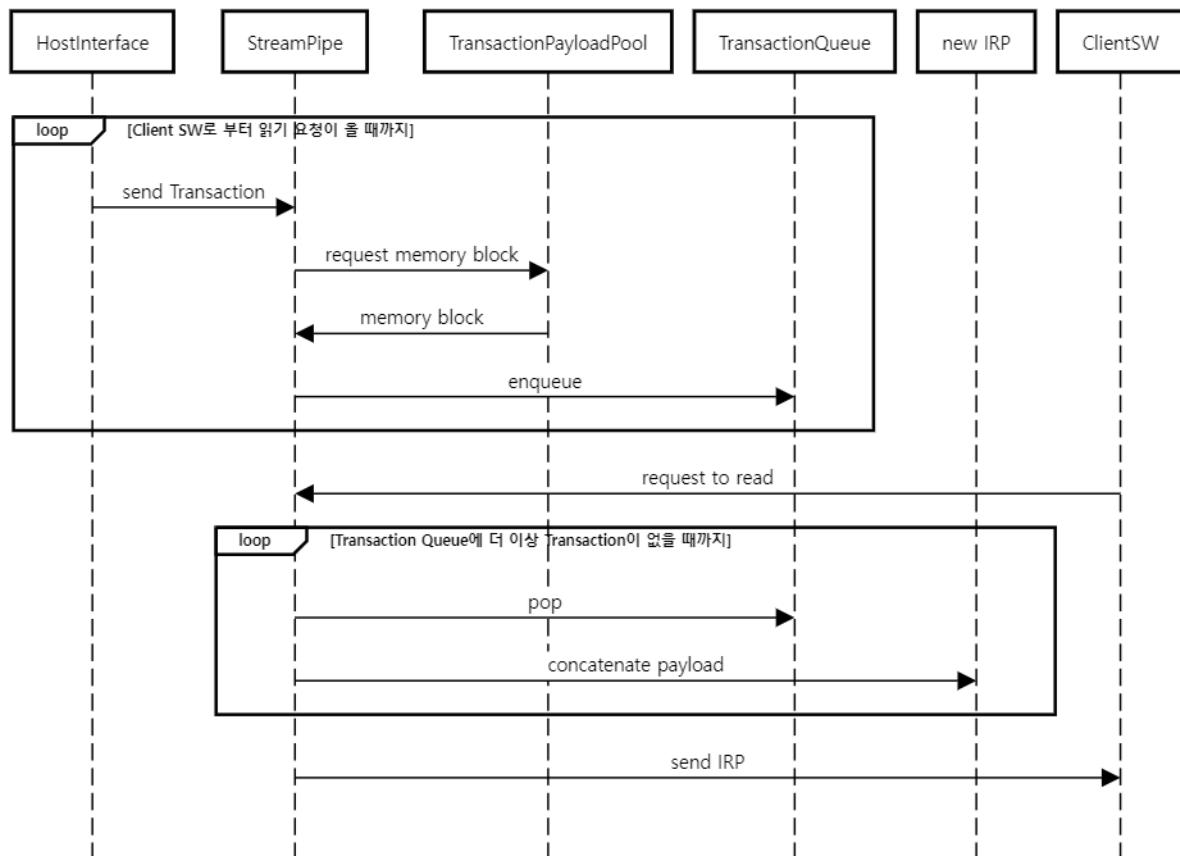


Figure 61 – CA_03의 동작

D1.1.4. CA_04. USB Device가 보내는 데이터를 계속 Concatenate하면서 이어 나가다가 Host에서 요청시 하나로 보내는 방법

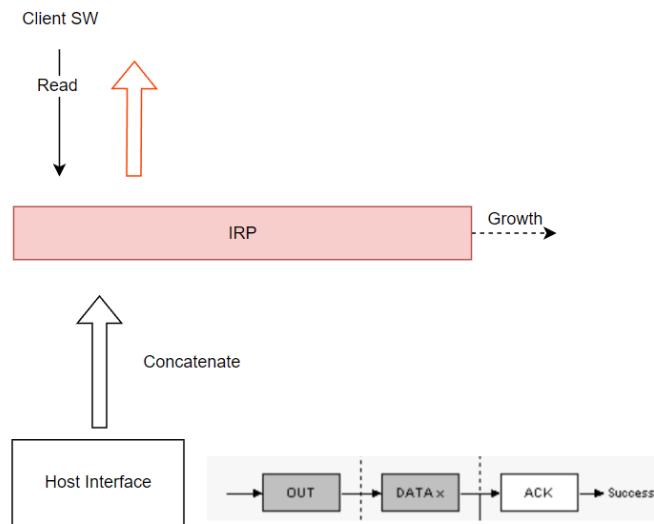


Figure 62 - 매번 Concatenate하는 방법

CA_04는 CA_03과는 달리 매번 USB Device로부터 전송되는 Transaction의 Payload들을 IRP의 Tail에 Concatenate해서 계속해서 하나의 IRP를 키워 나가는 방식이다. Client SW로부터의 Read 요청이 있는 경우 바로 구성한 IRP를 전송한다.

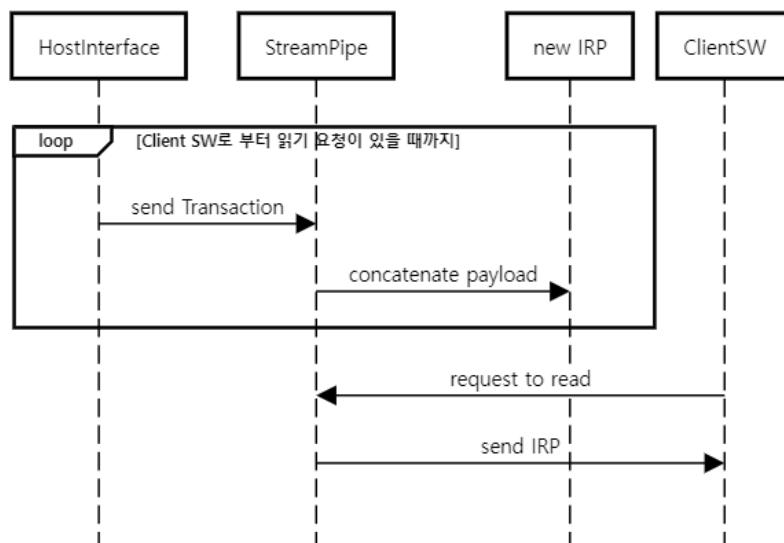


Figure 63 - CA_04의 동작

D1.2. 전송의 병렬성

D1.2.1. CA_05. Stream Pipe가 생성되면 별도의 쓰레드를 할당하는 방식

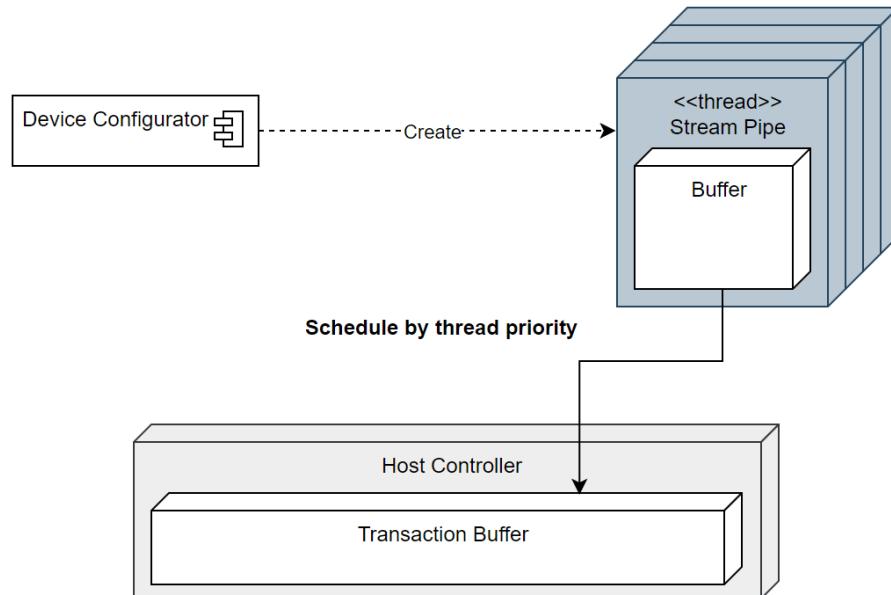


Figure 64 – 멀티 쓰레드로 Pipe를 구성하는 구조

CA_05은 Stream Pipe가 개설될 때마다 별도의 쓰레드를 생성하여 병렬적인 데이터 처리가 가능하도록 구성한 구조이다. 이는 Java의 Servlet이나 Django 등에서 웹 서비스를 구성하는 방식과 유사하다.

이 후보 구조의 핵심적인 아이디어는 전송 속도를 쓰레드 우선순위에 따른 스케줄링에 따라 조절하는 것이다. Stream Pipe는 네 가지 전송 타입을 가지고 있는데 특성에 따라 전송 주기나 속도가 다르다. 이러한 전송 특성에 따라 쓰레드 우선순위를 부여하여 스케줄링된 쓰레드가 Host Controller의 Buffer에 데이터를 넣어 전송할 수 있도록 한다.

이 구조의 경우 주의사항은 우선순위에 따른 특정 Stream Pipe의 기아 현상을 방지해야한다는 것이다. 그를 위해 어느정도 전송이 이루어지거나 일정 시간이 되면 다른 특성을 가진 Stream Pipe의 전송이 이루어질 수 있도록 스스로 yield하는 동작이 필요하다. 예를 들어 Interrupt Transfer의

경우 입력이 발생하면 우선순위를 높여 우선적으로 처리될 수 있어야 하며, Isochronous Transfer의 경우 전송 주기가 되면 우선 순위를 높여 최우선으로 처리될 수 있도록 해야하고, Control Transfer의 경우에도 주기적으로 우선 순위를 높여 제어 정보를 받을 수 있어야 한다. 그를 위한 현재 스케줄된 Stream Pipe의 양보가 필요하다. 그 외로는 멀티쓰레드 환경에서 공유 자원을 최소화하여 성능저하를 막아야 한다.

또한 본 후보 구조의 실질적인 동작 방식은 CA_01, CA_02 중 채택되는 후보 구조의 시퀀스 다이어그램을 참고하면 된다.

D1.3.2. CA_06. 비동기로 처리하는 방식

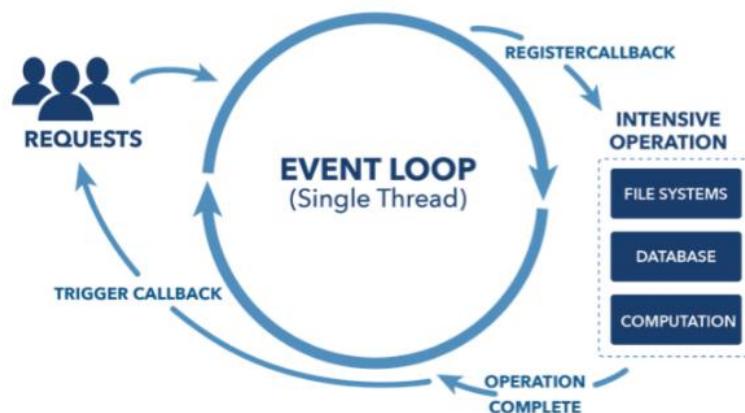


Figure 65 - nodejs의 Single Thread 기반 비동기 처리 방식

CA_06는 하나의 Single Thread로 Stream Pipe를 Scheduling해서 전송의 병렬성을 확보하는 방식의 구조이다. Blocking되는 작업이 발생하는 경우 다른 작업에게 양보하여 하나의 Thread에 대한 가용성을 극대화하는 방식이며, 멀티 쓰레드 방식과 달리 공유 자원에 대한 처리가 용이하다. 이는 nodejs와 같은 비동기 환경에서 웹서비스를 구성하는 방식과 유사하다.

단일 쓰레드 환경에서 Stream Pipe는 요청들을 Queueing하고 순차적으로 Host Controller에게 전송한다. 전송이 완료되었다는 Event Callback이 와서 전송이 완료되었거나 Blocking에 빠진 것을 확인하는 경우 다음 요청을 전송하고 대기 중인 요청은 다시 Queue에 쌓아 놓는다.

동작이 단조로운 장점이 있으나 기아 현상에 대한 고려사항으로 우선순위 Queue를 운영해야 하는 경우 그 장점이 상쇄될 수 있다. Blocking 시간을 최소화하여 Host Controller의 가용률을 높이고 공유자원으로 인한 성능 저하도 방지할 수 있는 구조이지만 단조로운 구조로 다양한 전송 방식에 대한 처리가 까다로울 수 있다.

D2. NFR_01 디바이스 설정 시간

USB Device가 삽입되었을 때 설정 시간을 최소화하기 위해 본 시스템은 네 가지 주제에 대해 후보 구조를 설계하였다.

1. 파이프의 버퍼 구조
2. 디바이스 식별 구조
3. USB Device 주소 할당 방식
4. Known Device 보관 구조

디바이스 설정 시간에 관련된 성능 품질과 관련하여 Stream Pipe 개설 시간 단축 및 Descriptor 해석 시간 단축에 주안점을 두고 최적 설계를 고민하였으며, 그 결과 1, 3번은 전자, 2, 4번은 후자에 관한 후보 구조가 선정되었다.

그를 위해 Stream Pipe 개설 시간 단축과 관련해서는 Buffer의 메모리 할당 시간 절약, 주소 할당 알고리즘의 최적화가 고려되었으며, Descriptor 해석 시간 단축을 위해서 연결 이력이 있는 USB Device의 shortcut과 그를 위한 식별자에 대한 고려사항이 후보 구조에 반영되었다.

이러한 특성을 고려하여 디바이스 설정 시간을 줄이기 위한 후보 구조를 알아본다.

D2.1. 파이프의 버퍼 구조

D2.1.1. CA_07. Stream Pipe를 구성하기 위한 버퍼의 Pool을 통해 즉시 개설이 가능하도록 하는 방법

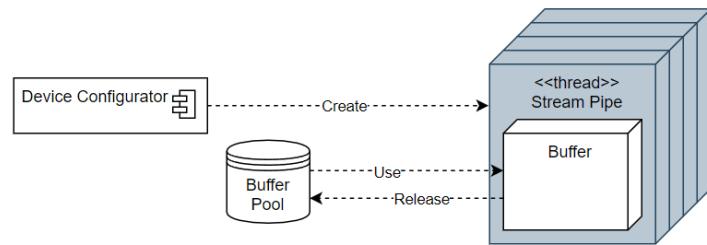


Figure 66 - Buffer Pool을 이용해 Stream Pipe를 구성하는 방법

CA_07은 Stream Pipe를 개설할 때, Buffer에 대한 메모리 할당 및 해제 시간에 대한 절약 목적으로 Buffer Pool을 구성하는 방식이다. 메모리 할당 및 해제는 비용이 큰 Operation이므로 이를 통해 성능을 높일 수 있는 구조이다.

D2.1.2. CA_08. Stream Pipe들을 위한 거대한 메모리 공간을 두고 필요한 리소스만큼 경계를 부여하는 방법

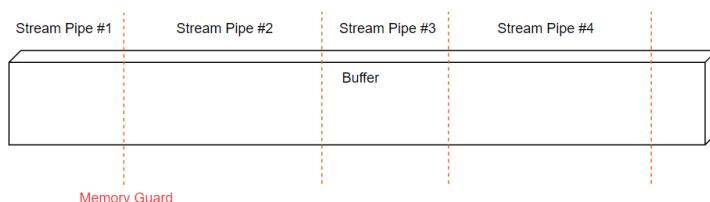


Figure 67 - 하나의 Buffer를 동적으로 잘라서 쓰는 방법

CA_08은 하나의 거대한 Buffer 공간을 Memory Guard로 구분하여 사용하는 구조이다. CA_08과 마찬가지로 메모리 할당 및 해제에 대한 비용을 줄일 수 있다. Overflow 및 Underflow에 대한 Memory Guard의 구현이 쉽지 않지만 필요한 만큼 동적으로 Buffer를 구성하는데 유리하다.

D2.2. 디바이스 식별 구조

D2.2.1. CA_09. Known Device에 대해서 Fast Track으로 연결하는 구조

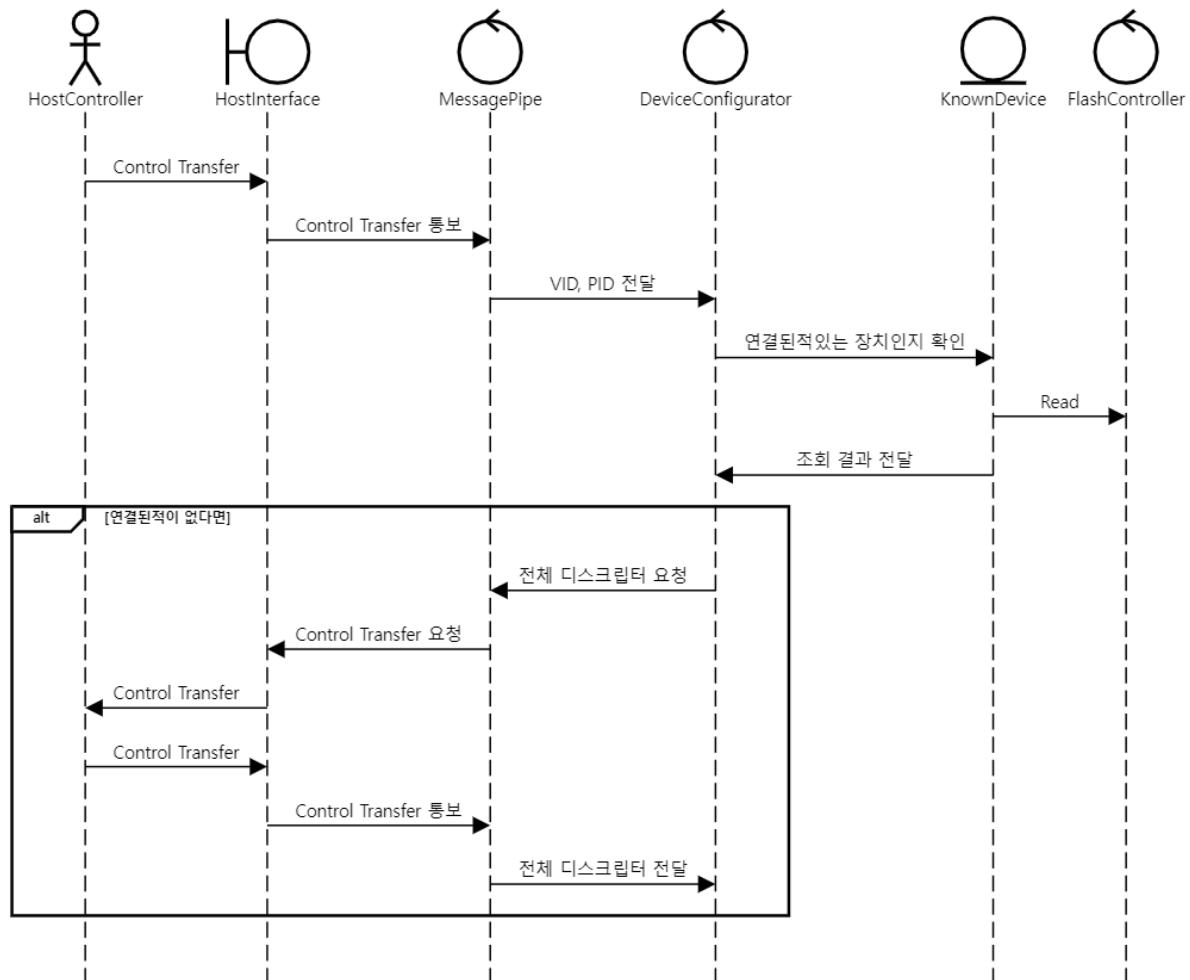


Figure 68 - Fast Track

CA_09는 USB Device의 VID와 PID를 조합하면 고유한 키가 생긴다는 점에 착안하여, 연결 이력을 저장하고, 다시 연결 시도가 있는 경우 USB Device Descriptor 전송 과정 없이 연결을 구성하는 구조이다. 데이터 보관 비용이 발생하긴 하지만 연결 속도에는 장점이 있다.

Operation System 레벨에서 한번 연결했던 블루투스 및 USB Device는 단시간에 사용이 가능하다. 이와 유사한 방식으로 Enumeration 과정을 단축한다.

D2.2.2. CA_10. 모든 접속에 대해서 디스크립터를 확인하는 구조

CA_10은 통상적인 USB Device의 연결방식이다. USB Device Descriptor를 전달받아 그에 대한 연결을 구성한다.

D2.3. USB Device 주소 할당 방식

D2.3.1. CA_11. 주소 할당을 라운드 로빈으로 하는 방식(포인터와 127 라운드)

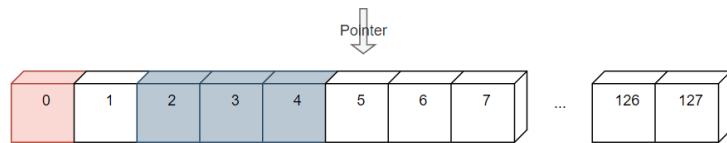


Figure 69 - 라운드로빈 방식

CA_11는 라운드 로빈 방식으로 연결된 USB Device에게 주소를 할당하는 방식이다. 할당한 위치를 가리키는 Pointer가 존재하며, 연결 해제로 인해 앞선 번호가 비더라도 계속 포인터 위치를 늘려가며 전진한다. 최대 할당 번지는 127을 넘는 경우 Default Pipe에서 사용하는 0번을 제외하고 1번으로 돌아가게 된다.

D2.3.2. CA_12. 빈 주소를 List로 가지고 있는 방식

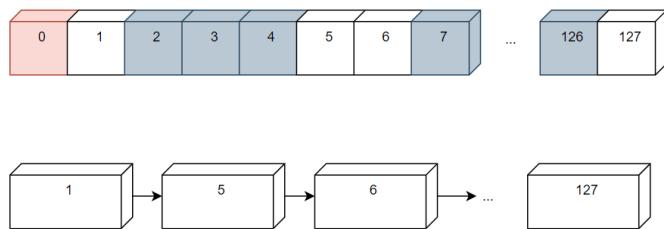


Figure 70 - 빈 주소를 리스트로 관리하는 방식

CA_12는 비어있는 주소의 번지를 List의 형태로 보관하는 방식이다. 리스트의 첫번째에 있는 주소를 사용하면 되며, 주소가 반환되면 리스트의 앞이나 끝에 연결한다.

D2.4. Known Device 보관 구조

D2.4.1. CA_13. Hash를 통해 Known Device를 저장하는 방식

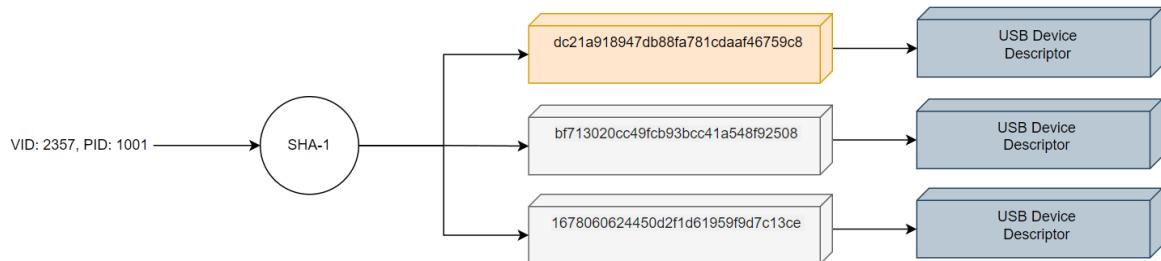


Figure 71 - Hash를 통해 Known Device를 저장하는 방식

CA_13은 VID와 PID를 조합한 고유한 값을 해시 값으로 사용하여 Known Device를 Hash에 저장하는 구조이다. Known Device 조회 시 O(1) 시간 복잡도로 데이터를 조회할 수 있다.

D2.4.2. CA_14. List를 통해 Known Device를 저장하는 방식

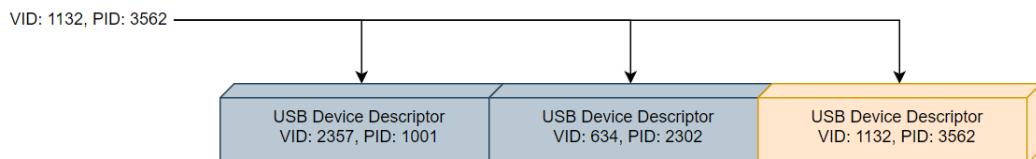


Figure 72 - List 방식으로 Known Device를 저장하는 방식

CA_14는 List 형식으로 Known Device를 저장하는 방식이다. 매치되는 USB Device를 순회하며 찾는다. 시간 복잡도는 O(N)이지만 구현이 단순하다.

D3. NFR_02 시스템 부팅 시간

시스템의 부팅 시간을 줄이기 위해 본 시스템은 두가지 주제에 대해 후보 구조를 설계하였다.

1. Flash에서 데이터를 읽는 타이밍
2. 부팅 시나리오를 관장하는 모듈의 유무

본 시스템은 영구적인 데이터 저장이 필요한 것으로 판단하였으며, 따라서 하드웨어인 Flash에 대한 접근 빈도나 데이터의 양이 성능 품질 요소를 위한 최적 설계 요소로서 고려되었다. 또한 부팅 시나리오의 최적화는 부팅 속도를 줄이는데 중요한 최적 설계 요소로서 그 주체의 유무에 대한 고려사항이 후보 구조로 설계되었다.

이러한 특성을 고려하여 시스템 부팅 시간을 줄이기 위한 후보 구조를 알아본다.

D3.1. Flash에서 데이터를 읽는 타이밍

D3.1.1. CA_15. Flash에서 미리 테이블을 읽고 사용하기

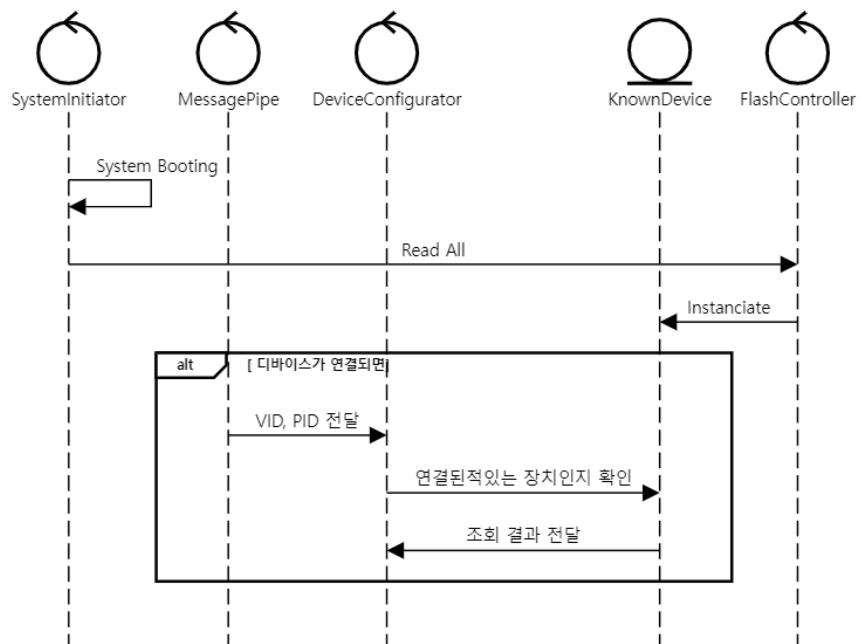


Figure 73 – 부팅 시 미리 읽는 방식

CA_15는 부팅 시간에 모든 데이터를 Flash Controller로부터 읽고, 조회가 필요할 때는 메모리에 있는 데이터를 사용하는 방식이다. Flash는 하드웨어이므로 접근 횟수를 줄여서 조회 시간을 단축 할 수 있지만, 수정이 필요한 경우 읽어온 데이터와 Flash의 데이터를 계속해서 동기화해줘야 하는 문제가 있다.

D3.1.2. CA_16. Flash에서 필요할 때 접근

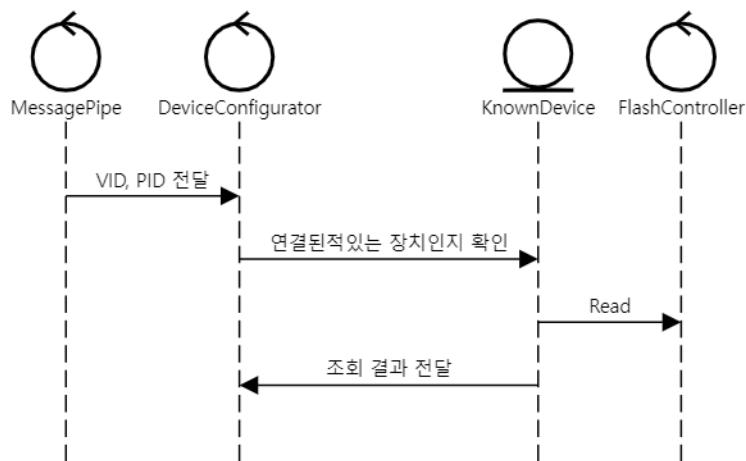


Figure 74 - 필요할 때마다 읽는 방식

CA_16는 데이터가 필요할 때마다 Flash에서 데이터를 읽는 방식이다. 하드웨어로의 접근 횟수는 늘어나지만 동기화 문제에서 자유롭다.

D3.2. 부팅 시나리오를 관장하는 모듈의 유무

D3.2.1. CA_17. 부팅 시 모든 모듈에게 초기화 명령 전달

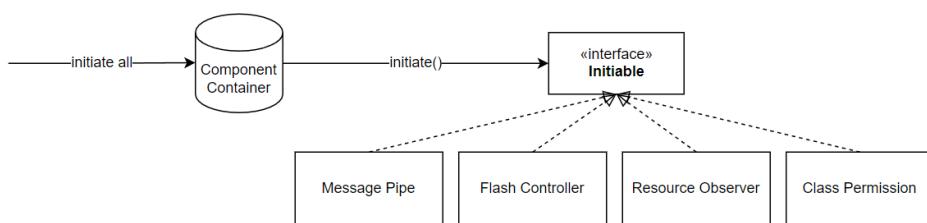


Figure 75 - 모든 모듈에게 초기화 명령을 전달하는 방식

CA_17은 모든 모듈을 보관하는 저장소가 있다고 가정하고 시작한다. 이는 스프링 프레임워크에서의 IoC 컨테이너와 유사한 개념이다. 초기화 과정이 필요한 모듈들은 초기화를 위한 인터페이스를 구현하도록 되어 있으며 컨테이너에 의해 일괄적으로 초기화 명령이 전달되게 된다.

D3.2.2. CA_18. 시스템 초기화 및 종료를 위한 모듈

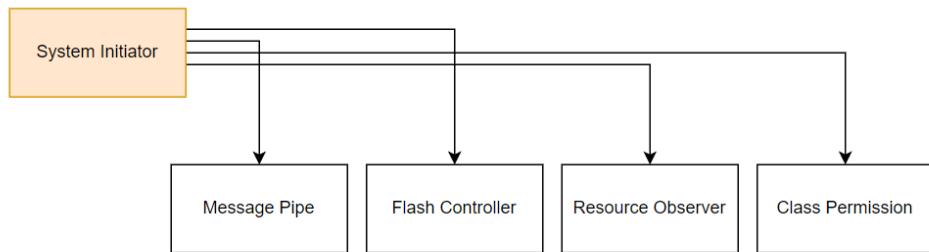


Figure 76 - 시스템 초기화를 관장하는 모듈을 두는 방식

CA_18은 시스템 초기화 및 종료를 위한 주체를 따로 두는 방식이다. 초기화라는 책임을 부여해서 응집성을 높이고 시나리오의 순서 등을 관장하기 쉬운 장점이 있다.

D4. QA_02 지원 가능한 USB Class 확장

지원 가능한 USB Class의 확장 시 변경용이성을 확보하기 위해 본 시스템은 USB 허용 클래스 취득 방법에 대해 설계하였다. 이는 런타임에 USB Class를 얻어올 수 있는지에 대한 고려사항이다. 런타임에 정보를 얻어오는 방식은 유연하여 변경 용이성에 도움이 되어 최적 설계를 위한 요소라 할 수 있다. 이러한 특성을 고려해 지원 가능한 USB Class 확장을 위한 후보 구조를 알아본다.

D4.1. USB 허용 클래스 취득 방법

D4.1.1. CA_19. 허용 가능한 클래스를 OS로부터 받아와서 비트맵으로 저장하는 방식

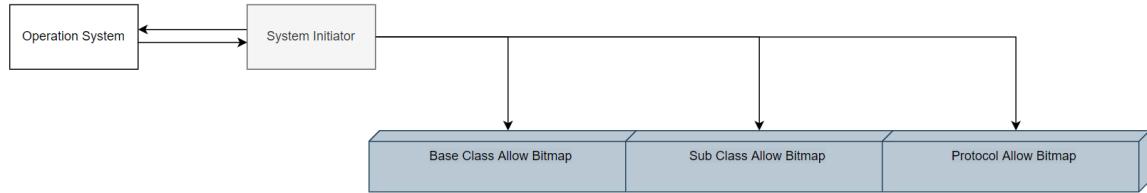


Figure 77 - OS로부터 허용 가능한 USB 클래스를 얻어오는 방식

CA_19는 시스템 부팅 시간에 Operation System으로부터 허용 가능한 USB Class를 얻어와서 비트 맵 형식으로 저장해두는 방식이다. 운영체제와의 별도의 인터페이스가 필요하다는 단점이 있지만 런타임에 이를 얻어 음으로써 변경용이성에 큰 도움을 준다.

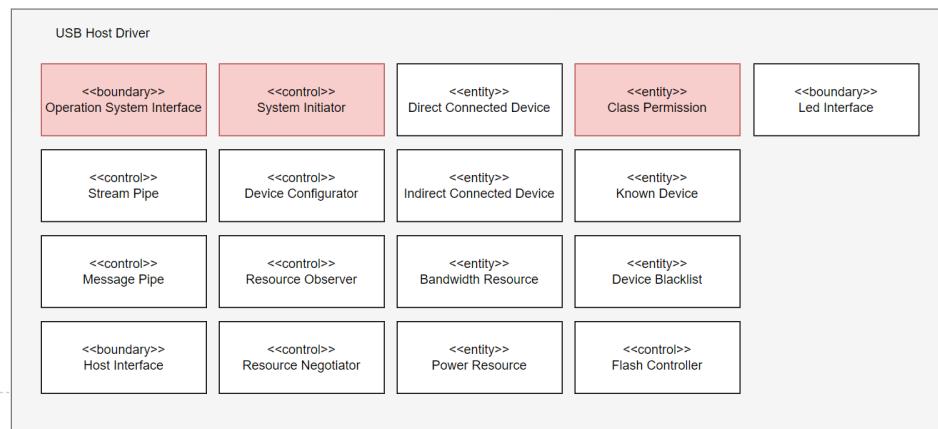


Figure 78 - 영향을 받는 도메인 요소

D4.1.2. CA_20. 모두 허용해놓고 거부되는게 있을 때마다 하나씩 줄여 나가는 방식

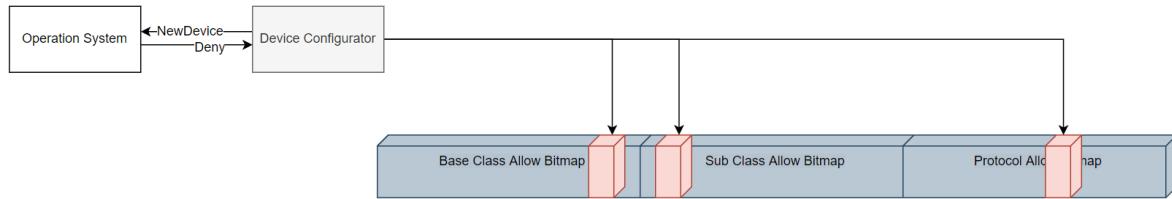


Figure 79 - 연결이 거부될 때마다 추가하는 방식

CA_20은 허용되지 않는 USB Class를 이유로 새로운 USB Device의 연결이 거부되었을 때마다 거부된 USB Class에 대한 기록을 진행하는 방식이다.

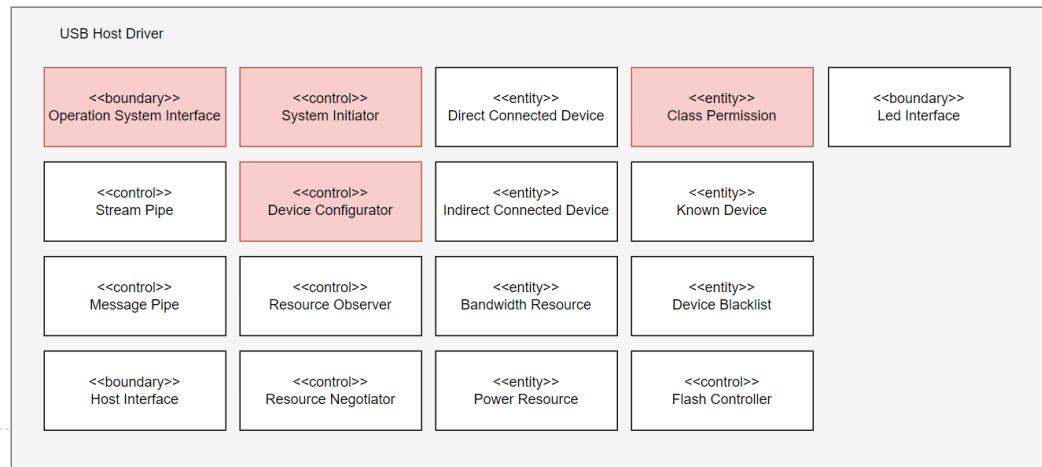


Figure 80 - 영향을 받는 도메인 요소

D5. QA_03 전송 방식 변경

전송 방식 변경 시 변경용이성을 확보하기 위해 본 시스템은 제어 신호의 전송 구조에 대해 후보 구조를 설계하였다. 이는 제어 신호 전송에 대한 다른 책임이 추가되었을 때의 최적 설계를 위한 고려사항이다. 이러한 특성을 고려하여 전송 방식 변경을 위한 후보 구조를 알아본다.

D5.1. 제어 신호의 전송 구조

D5.1.1. CA_21. Default Pipe도 제어정보를 전송하는 파이프의 상속 구조로 두는 구조

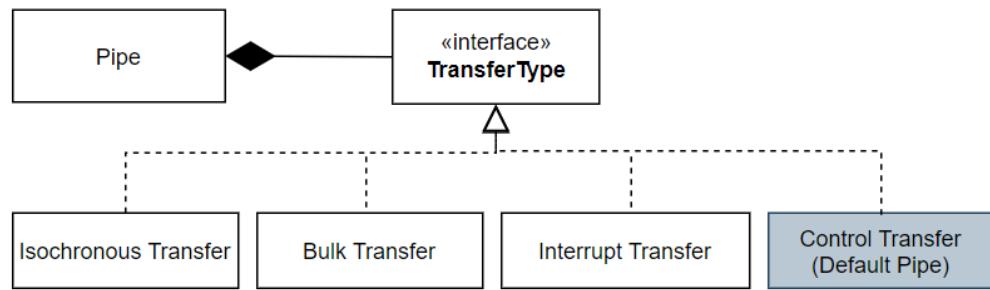


Figure 81 - 동일한 인터페이스를 가지는 구조

CA_21는 데이터든 제어 신호든 Pipe는 전송한다는 동일한 책임을 갖는 것으로 보고 동일 인터페이스를 구현하는 컴포넌트로 보는 구조이다. 전송한다는 책임하에서 제어 정보를 주고받아 원하는 목적을 달성한다.

D5.1.2. CA_22. Default Pipe를 별도의 모듈로 두는 구조(Increase Cohesion)

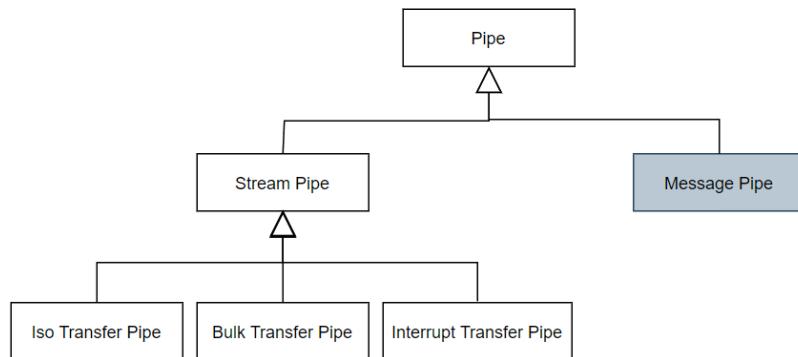


Figure 82 - Stream Pipe와 책임을 분리하는 구조

CA_22는 데이터를 전송하는 책임을 갖는 Stream Pipe와 제어하는 USB Device를 제어하는 Message Pipe의 책임을 나누는 구조이다. 일괄적인 처리는 동일한 책임을 갖는 것이 유리하지만 제어 정보 전달에 있어 좀 더 전문적인 처리가 가능해진다.

D6. QA_04 Host Controller의 USB 지원 버전 확장

Host Controller의 하드웨어 변경 등으로 USB 지원 버전 확장 시 변경용이성을 확보하기 위해 본 시스템은 두가지 주제에 대해 후보 구조를 설계하였다.

1. 파이프의 버퍼 구조
2. Stream Pipe 추상화

USB 버전 업그레이드로 인한 변경 사항 중 가장 치명적인 변경사항은 전송 방식의 변경으로, Half-duplex 방식이 제공되는 2.0에서는 차후 Full-duplex 방식으로의 업그레이드가 예상되었다. 따라서 그러한 변경사항에 대비하기 위한 최적의 후보 구조를 설계하려 했으며, 그 다음 리스크인 USB Specification에 정의되는 전송 타입의 변경에 대응하기 위한 추상화 구조도 후술할 후보 구조에서 다루게 된다.

이러한 특성을 고려하여 Host Controller의 USB 지원 버전 확장을 위한 후보 구조를 알아본다.

D6.1. 파일의 버퍼 구조

D6.1.1. CA_23. Full-Duplex 지원을 위해 Stream Pipe에서 읽기와 쓰기 버퍼를 분리하는 방식

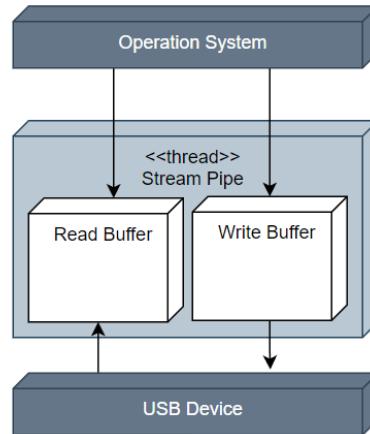


Figure 83 - Read / Write 버퍼를 분리하는 방식

CA_23은 Read Buffer와 Write Buffer를 분리하여 구현하는 방식이다. 추후 Full-Duplex 구현이 필요해지는 경우 각각의 버퍼를 다른 Thread에서 담당하도록 하여 병렬성을 확보하면 큰 로직 수정 없이 변경이 가능할 것으로 보인다.

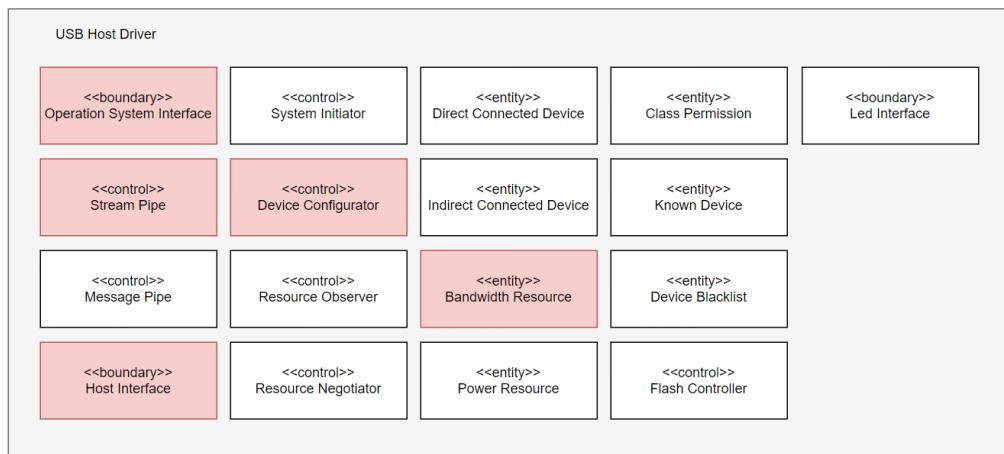


Figure 84 - 영향을 받는 도메인 요소

D6.1.2. CA_24. 하나의 버퍼에서 읽기와 쓰기를 동시에 진행하며 IRP 마다 읽기인지 쓰기인지 구분하는 방식

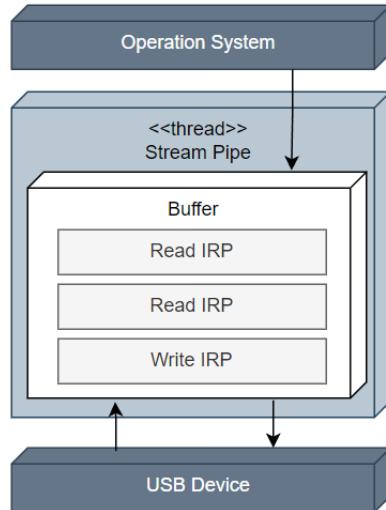


Figure 85 - 하나의 버퍼로 처리하는 방법

CA_24은 하나의 버퍼로 Stream Pipe를 구현하는 방식이다. 추후 Full-Duplex 구현이 필요한 경우 별도로 두개의 Thread가 Buffer에 접근하여 필요한 데이터를 주고받아야 한다. 영향을 받는 도메인 모델의 양상은 CA_23과 같을 수 있으나 주로 로직은 크게 수정되어야 할 것으로 보인다.

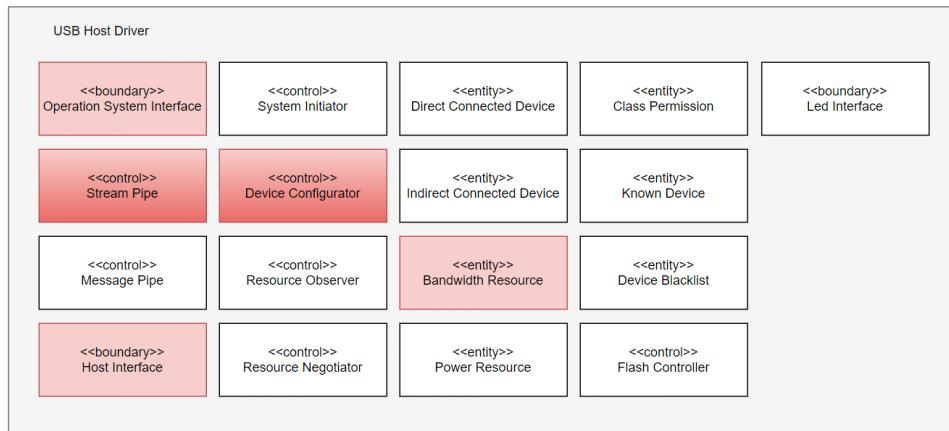


Figure 86 - 영향을 받는 도메인 요소

D6.2. Stream Pipe 추상화

D6.2.1. CA_25. Stream Pipe의 State Pattern 적용

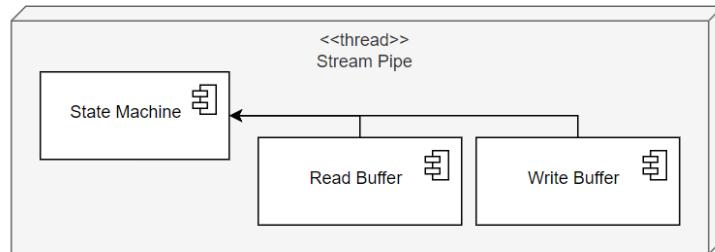


Figure 87 - Stream Pipe의 State Machine

CA_25은 Stream Pipe마다 State Machine을 두는 방식이다. USB Specification에 정의된 방식을 고려할 때, Device Attached, Powered, Bus Reset와 같은 State가 필요할 수 있으며 추후 State에 대한 추가도 용이하다. GoF의 디자인패턴 중 State Pattern을 활용할 수 있다.

D6.2.2. CA_26. ISP을 준수하는 Pipe의 다양한 인터페이스 제공

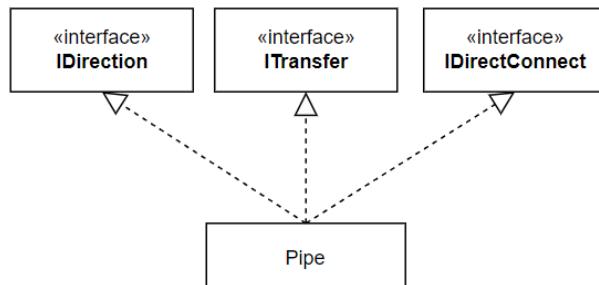


Figure 88 - 다양한 책임을 갖는 Pipe

CA_26는 Pipe가 갖는 책임에 따라 다양한 인터페이스를 구현하고 불필요한 인터페이스는 구현하지 않아 ISP를 지키는 구조이다. 양방향을 포함한 전송 방향에 따른 인터페이스, 등시성, 벌크, 인터럽트, 컨트롤 등 전송 방식에 따른 인터페이스, Sub Hub를 통해 연결된 Pipe 등 책임에 따른 인터페이스를 구현한다.

D6.2.3. CA_27. 전송 방법에 따라 다형성으로 표현하는 방법

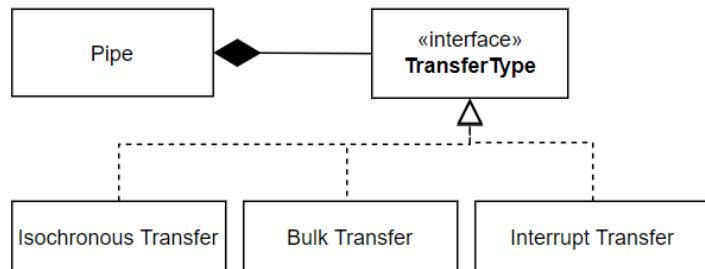


Figure 89 - 전송방식에 따른 다형성

CA_27은 다양한 전송 방식에 대해 다형성을 이용해 처리하는 구조이다. Stream Pipe는 USB Specification에 벌크, 등시성, 인터럽트 등 다양한 전송방식을 제공하고 있는데, 추후 상위 버전에서 다른 전송방식을 제공하더라도 전송하다라는 책임은 변함이 없을 것이라는 판단 하에 다형성을 통해 처리한다. Stream Pipe를 다루는 주체에서 전송방식에 대해 고민하지 않아도 된다는 장점이 있다.

D7. QA_05 동시에 연결 가능한 USB Device의 수

동시에 연결 가능한 USB Device의 수를 최대화하여 가용성을 확보하기 위해 본 시스템은 두가지 주제에 대해 후보 구조를 설계하였다.

1. USB Device 연결에 대한 리소스 할당
2. 리소스 확인 주기에 대한 구조

1번의 경우 USB Device에 리소스 요구사항에 대해 판단할 주체에 대한 고려 사항이며, 2번의 경우 가용 가능한 리소스를 파악하기 위한 고려사항이다. 이는 시나리오에 대한 최적화 및 정확한 정보에 의한 자원 할당을 통해 가용성을 달성하기 위한 최적 구조 설계 활동이다.

이러한 특성을 고려하여 동시에 연결 가능한 USB Device의 수를 위한 후보 구조를 알아본다.

D7.1. USB Device 연결에 대한 리소스 할당

D7.1.1. CA_28. Resource Observer를 둘러 리소스 현황을 주기적으로 파악하고 최적의 리소스 할당하는 방식

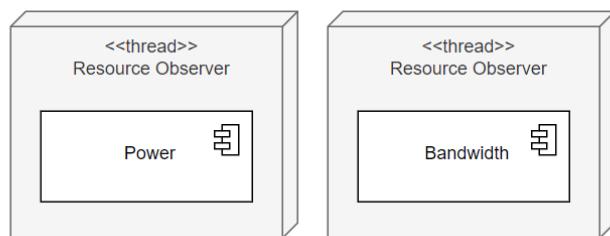


Figure 90 - Resource Observer가 가용 리소스를 파악하는 구조

CA_28은 Resource Observer라는 컴포넌트가 가용 리소스를 파악하는 구조이다. 전송 등 다른 작업과 병렬적으로 가용 리소스를 파악할 수 있기 위해 별도의 쓰레드로 구성하며 리소스 종류별로 별도의 쓰레드를 할당할 수도 있다. 이렇게 수집한 가용 리소스의 정보는 USB Device의 요구사항과의 Negotiation 과정에 활용된다.

D7.1.2. CA_29. 최대 가용 자원만 파악한 뒤 요청이 올 때마다 Best Effort로 할당하고 더 이상 연결을 허용하지 않는 방식

CA_29는 최대 가용자원을 파악한 뒤, USB Device가 요구하는 리소스에 대해 Best Effort로 맞춰서 할당한 뒤, 더 이상 가용 가능한 리소스가 없으면 연결을 허용하지 않는 구조이다.

D7.2. 리소스 확인 주기에 대한 구조

D7.2.1. CA_30. Watchdog 형식으로 주기적으로 가용자원을 확인하는 방식

CA_30은 CA_18의 구조에서 가용 리소스에 대한 확인 주기에 대한 고려사항이다. 정해진 주기를 두고 가용 리소스를 파악한다.

D7.2.2. CA_31. 특정 이벤트(디바이스 연결 해제 등) 발생 시마다 가용자원을 확인하는 방식

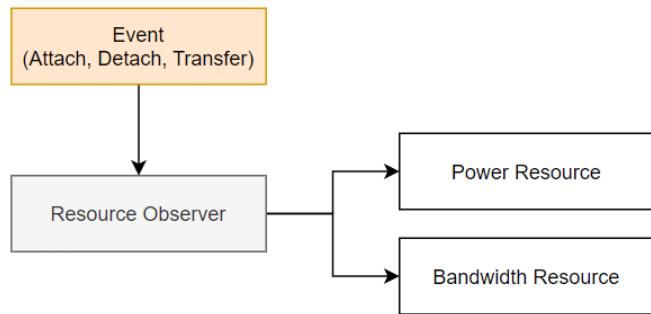


Figure 91 - 특정 이벤트마다 가용 리소스를 파악하는 방식

CA_31는 특정 주기가 아니라 특정 이벤트가 발생할 때마다 가용 리소스를 파악하는 방식이다. 주기 방식에 비해 리소스 파악을 위한 비용은 감소하지만 가용 리소스에 대한 실시간 파악은 어려울 수 있다.

D8. QA_06 외부 모듈 교환

외부 모듈 교환 시 변경 용이성을 확보하기 위해 본 시스템은 외부와의 인터페이스 추상화 구조에 대해 후보 구조를 설계하였다. 이는 외부 모듈이 변경되었을 때의 고려사항이다. 두 가지 방향이 있을 것으로 생각하였는데 시스템에서 요구하는 오퍼레이션은 정해져 있으므로 그에 따른 인터페이스를 정의하고 적용 가능한 하드웨어만 적용하는 경우와 여러 외부 모듈에 관해 적용이 가능하도록 어댑터를 제공하는 방식이 있다. 그에 따라 최적의 후보 구조를 설계해보려 한다. 이러한 특성을 고려하여 외부 모듈 교환을 위한 후보 구조를 알아본다.

D8.1. 외부와의 인터페이스 추상화 구조

D8.1.1. CA_32. Host와의 유니폼한 인터페이스를 추상화하는 방식

CA_32는 Operation System과 본 시스템 간의 동작을 미리 정의해서 인터페이스 역할의 컴포넌트를 두는 구조이다. Operation System이 변경되어도 정해진 동작을 수행할 수 있다면 변경사항이

없도록 하는 것이다. 이는 시스템에 외부 모듈이 인터페이스를 충족해야하는 방식이지만 본 시스템이 USB Specification을 충족하는 만큼 그에 맞는 기능을 제공하는 외부 모듈 만을 채택하는 방식이다. 외부 모듈 자체에 대한 변경 용이성은 떨어지나 시스템 자체의 변경 용이성은 올라간다.

D8.1.2. CA_33. 주요 운영체제에 대한 여러가지 인터페이스 제공

CA_33은 주요 운영체제(Windows, MAC, Ubuntu 등)과의 인터페이스를 개별적으로 준비하는 것이다. 제공하는 운영체제만 지원한다는 단점이 있지만 운영체제별로 세부적인 구현이 가능하다는 장점이 있다. 그에 따른 어댑터를 구현하여 동일한 인터페이스로 동작할 수 있도록 구현 노력이 필요하다.

D8.1.3. CA_34. LED와의 인터페이스를 위한 어댑터

CA_34은 Led Controller와의 인터페이스에서 어댑터 계층을 하나 둬서 Led 하드웨어가 교체되었을 때 유연하게 대처하는 구조이다.

D8.1.4. CA_35. Host Controller와의 인터페이스를 위한 어댑터

CA_35은 Host Controller와의 인터페이스에서 어댑터 계층을 하나 둬서 Host Controller가 교체되었을 때 유연하게 대처하는 구조이다. Host Controller와 연동하는 동작은 USB Specification을 통해 몇가지로 추상화 할 수 있으므로 구현에 무리가 없다고 보여진다.

다만 USB Specification의 버전 업그레이드와 그에 따른 하드웨어의 변경 등에 대처하기 위해서는 대규모의 변경이 필요할 수 있는데 그와 관련해서는 D6.1. 항목을 참고한다.

D9. QA_07 디바이스 정보 조회 시간

디바이스 정보 조회 시간을 단축하기 위해 본 시스템은 디바이스 정보 저장 구조에 대해 후보 구조를 설계하였다. 이를 위해서는 최적의 자료구조를 찾는 것이 성능 품질 요소의 최적 구조를 찾기 위한 활동이 될 것이다. 이러한 특성을 고려하여 디바이스 정보 조회 시간 단축을 위한 자료구조 측면의 최적의 후보 구조를 알아본다.

D9.1. 디바이스 정보 저장 구조

D9.1.1. CA_36. Device 연결구조를 트리로 표현하는 구조

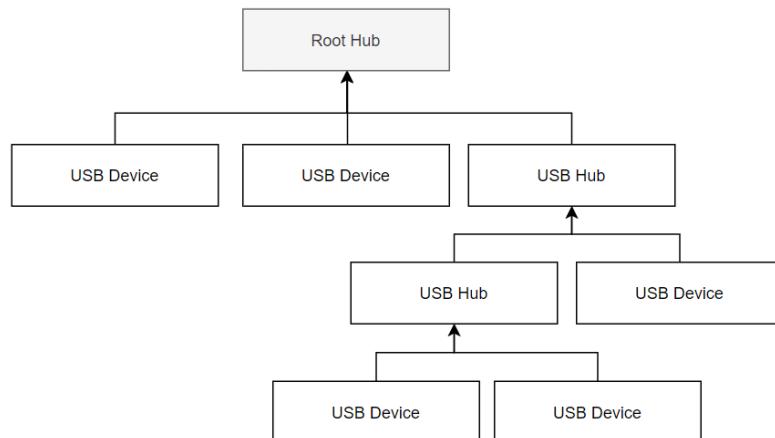


Figure 92 - Tree 구조로 표현하는 방식

CA_36는 Tree 구조를 통해 디바이스 연결을 표현하는 구조이다. USB 시스템은 Hub를 통해 계층적으로 연결되므로 직관적으로 연결 구조를 파악하는데 좋은 구조이다. Hash보다는 낮은 성능, List나 Array보다는 좋은 성능을 가질 것으로 예상된다.

D9.1.2. CA_37. 리스트로 표현하는 구조

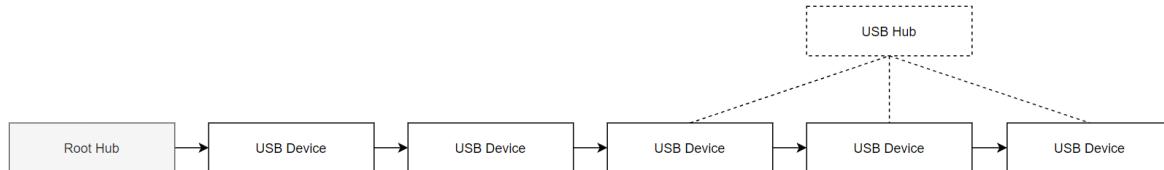


Figure 93 - List 구조로 표현하는 방식

물리적으로는 계층적으로 USB Device가 Hub를 통해 연결되어 있다고 하더라도, Operation System 입장에서는 논리적으로 동등한 디바이스라 할 수 있다. 따라서 CA_37처럼 List 형식의 표현이 가능하다. Tree 구조에 비해 입출력, 검색 방식이 단순하다는 장점이 있지만 USB Hub 분리 시 하위 장치들의 해제가 필요하므로 결국 어느정도 계층 정보를 표현할 수밖에 없는 단점이 있다.

D9.1.3. CA_38. 고유 아이디를 가지고 해시 구성

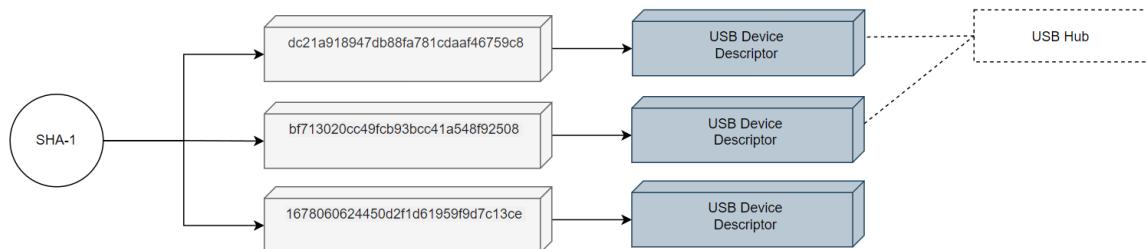


Figure 94 - Hash 구조로 표현하는 방식

CA_38은 디바이스의 연결을 Hash를 통해 표현하는 구조이다. USB Device의 VID, PID를 이용한 Key를 통해 O(1) 시간 복잡도로 필요한 디바이스를 찾을 수 있는 장점이 있다. 다만 순회에 어려움이 있어 이를 위한 별도의 트릭이 필요하며, 이 역시 USB Hub 분리 시 하위 장치들의 해제가 필요하므로 결국 어느정도 계층 정보를 표현할 수밖에 없는 단점이 있다.

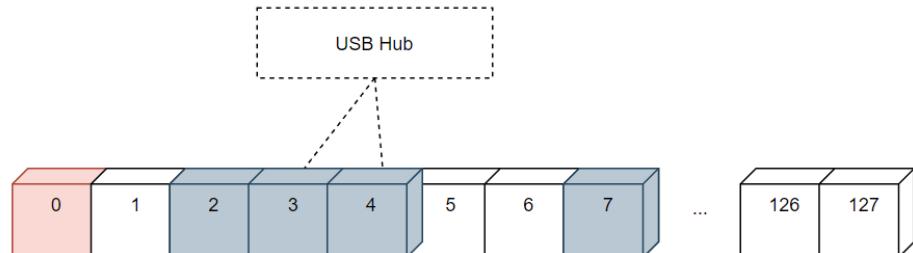
D9.1.4. CA_39. 구조체 배열로 구성하는 방법

Figure 95 - Array 구조로 표현하는 방식

List와 마찬가지로 논리적으로 동급인 USB Device를 CA_39처럼 일렬로 Array로 표현할 수 있다. 최대 연결 가능한 USB Device의 수가 정해져 있으므로 Array의 동적인 확장이 필요 없어 장점이 있다. 다만 이 역시 USB Hub 분리 시 하위 장치들의 해제가 필요하므로 결국 어느정도 계층 정보를 표현할 수밖에 없는 단점이 있다.

E. 후보 구조 평가

E1. QA_01 데이터 전송 속도

CA_01. IRP의 주소를 그대로 사용하여 추가적인 메모리 할당 없이 Transaction을 구성하는 방법.

[채택]

CA_02. 메모리 Pool을 이용하여 새로운 Transaction을 구성하는 방법 [미채택]

품질 속성	CA_01 영향	CA_02 영향
QA_01 데이터 전송 속도	(++) 추가적인 메모리 할당이 나 데이터 복사가 필요하지 않다.	(+) Memory Pool을 통해 할당에 대한 비용을 아끼더라도 결국 Payload의 복사는 필요하다.
NFR_01. 디바이스 설정 시간		(-) Transaction Queue가 추가적으로 더 구성되어야 한다.
NFR_02. 시스템 부팅 시간	(-) IRP Descriptor Pool을 구성하는 시간이 소요된다.	(-) Payload Pool을 구성하는 시간이 소요된다.
QA_03. 전송 방식 변경	(--) 운영체제와의 인터페이스에 따라 주소체계를 활용하지 못할 수 있다.	(-) 전송 방식 변경에 의해 Payload Pool의 구성이 달라질 수 있다.

CA_01과 CA_02는 데이터 Write와 관련된 후보 구조로서 하나의 구조를 선택해야 했다. 데이터 전송 속도는 우선순위가 높은 품질 속성으로서 다른 품질 속성보다 우선적으로 성능에 대해 평가 하였으며 NFR_01이나 NFR_02, QA_03 등을 고려할 때에도 CA_01이 적합하여 최종 구조로 채택하였다.

하지만 이로 인한 트레이드오프로 발생하는 NFR_02와 QA_03의 리스크에 대해서도 고려가 필요하다. NFR_02의 경우 시스템 부팅보다 데이터 전송이 더 우선순위 높은 유즈케이스이므로 다소의 부팅속도 저하에 대해 감수하고, QA_03의 경우 3대 주요 운영체제(Windows, Mac, Linux)에 대해서만이라도 지원한다면 큰 리스크가 없을 것으로 보인다.

CA_03. USB Device가 보내는 데이터를 메모리 Pool을 이용하여 Queueing하다가 Host에서 요청이 오면 Concatenate해서 전송하는 방법 [채택]

CA_04. USB Device가 보내는 데이터를 계속 Concatenate하면서 이어 나가다가 Host에서 요청시 하나로 보내는 방법 [미채택]

품질 속성	CA_03 영향	CA_04 영향
QA_01 데이터 전송 속도	(++) Concatenate 과정이 1회만 필요하여 DMA 등의 하드웨어를 1회만 사용해서 쉽게 데이터를 구성할 수 있다.	(-) IRP를 동적으로 크기를 늘려가며 Concatenate를 진행하는 단점이 있다.
NFR_01. 디바이스 설정 시간		(+) 별도의 Queue를 구성하지 않아도 된다.
NFR_02. 시스템 부팅 시간	(-) Payload Pool을 구성하는 시간이 소요된다.	
QA_03. 전송 방식 변경	(-) 전송 방식 변경에 의해 Payload Pool의 구성이 달라질 수 있다.	

CA_03과 CA_04는 데이터 Read와 관련된 후보 구조로서 하나의 구조를 선택해야 했다. 데이터 전송 속도는 우선순위가 높은 품질 속성으로서 다른 품질 속성보다 우선적으로 성능에 대해 평가하였으며 NFR_01이나 NFR_02, QA_03 등을 고려할 때 종합적인 평가는 비슷하였으나 성능에 더욱 주안점을 두고 CA_03이 적합하여 최종 구조로 채택하였다.

하지만 이로 인한 트레이드오프로 발생하는 NFR_02와 QA_03의 리스크에 대해서도 고려가 필요하다. NFR_02의 경우 시스템 부팅보다 데이터 전송이 더 우선순위 높은 유즈케이스이므로 다소의 부팅속도 저하에 대해 감수하고, QA_03의 경우 새로운 전송 타입으로 인한 새로운 frame 사이즈에 대해서 Payload 종류를 추가하는 것은 큰 틀에서 성능 저하가 미미할 것으로 예상된다.

CA_05. Stream Pipe가 생성되면 별도의 쓰레드를 할당하는 방식 [채택]**CA_06. 비동기로 처리하는 방식 [미채택]**

품질 속성	CA_05 영향	CA_06 영향
QA_01 데이터 전송 속도	(++) 다수의 CPU 코어를 활용하여 더 좋은 성능을 낼 수 있다. (-) 공유 자원 동기화로 인한 성능 저하가 있을 수 있다.	(-) 하나의 쓰레드를 사용하므로 여러 코어를 통한 성능 증가를 기대할 수 없다.
NFR_02. 시스템 부팅 시간	(-) Thread Pool을 구성하는 시간이 소요될 수 있다.	
QA_03. 전송 방식 변경	(+) 전송 방식에 알맞은 Thread 우선순위를 부여하여 스케줄링 할 수 있다.	(-) 전송 방식에 알맞은 우선순위를 부여하기 위해 로직을 구현하기가 어렵다.

CA_05와 CA_06는 데이터 전송의 병렬성에 관련된 후보 구조로서 하나의 구조를 선택해야 했다. Thread 방식에는 비동기 방식에 비해 고전적인 문제(공유자원 및 쓰레드 생성)를 가지고 있지만 최근 Host PC들이 다수의 코어를 가지고 동작함을 고려할 때 성능을 위주로 판단하여 CA_05을 채택하였다.

CA_05 채택에 대한 리스크 관리로서 Thread Pool의 구성은 큰 부팅시간 성능 영향을 주지는 않을 것으로 예상된다. 다만 공유 자원에 대한 관리는 성능 품질에 크게 영향을 줄 수 있는 요소로써 세부 구조를 설계함에 있어 공유 자원을 최소화하려는 노력을 해야 한다.

E2. NFR_01 디바이스 설정 시간

CA_07. Stream Pipe를 구성하기 위한 버퍼의 Pool을 통해 즉시 개설이 가능하도록 하는 방법

[채택]

CA_08. Stream Pipe들을 위한 거대한 메모리 공간을 두고 필요한 리소스만큼 경계를 부여하는 방법 [미채택]

품질 속성	CA_07 영향	CA_08 영향
QA_01 데이터 전송 속도	(+) Stream Pipe 구성 시 버퍼 구성을 위한 시간이 거의 들지 않는다.	(+) Stream Pipe 구성 시 버퍼 구성을 위한 시간이 거의 들지 않는다. (-) Memory Guard는 해당 후보 구조에서 가장 중요한 요소로 이를 위한 로직이 항상 동작해야 한다.
NFR_02. 시스템 부팅 시간	(-) Buffer Pool을 구성하는 시간이 소요될 수 있다.	
QA_03. 전송 방식 변경		(+) 동적인 Buffer 크기 할당이 가능한 구조이므로 장점이 있다.

CA_07와 CA_08는 Stream Pipe의 Buffer 할당에 관련된 후보 구조로서 하나의 구조를 선택해야 했다. CA_08의 동적인 버퍼 크기 할당은 장점이지만 Buffer Pool 역시 다양한 크기의 Buffer를 제공하는 방식을 취할 수도 있으며 Memory Guard에 대한 CA_08의 단점으로 인해 CA_07을 채택하였다.

CA_07이 시스템 부팅 시간에 약점이 있긴 하지만 시스템 부팅보다 데이터 전송이 더 우선순위 높은 유즈케이스이므로 다소의 부팅속도 저하에 대해 감수할 수 있을 것으로 보인다.

CA_09. Known Device에 대해서 Fast Track으로 연결하는 구조 [채택]**CA_10. 모든 접속에 대해서 디스크립터를 확인하는 구조 [미채택]**

품질 속성	CA_09 영향	CA_10 영향
NFR_01. 디바이스 설정 시간	(++) 이미 연결한 이력이 있는 USB Device의 경우 Descriptor 전달없이 빠르게 연결할 수 있으므로 성능에 장점이 있다.	
NFR_02. 시스템 부팅 시간	(-) Known Device를 Flash로부터 읽는 동작이 필요하므로 부팅 시간이 필요하다. (CA_18 채택의 경우)	

CA_09와 CA_10은 빠른 USB Device 연결에 관련된 후보 구조로서 하나의 구조를 선택해야 했다.

CA_10은 통상적인 연결 흐름으로, CA_09에 장점이 더 부각되어 이를 채택하였다.

CA_09이 시스템 부팅 시간에 약점이 있긴 하지만 시스템 부팅보다 디바이스 설정이 더 우선순위 높은 유즈케이스이므로 다소의 부팅속도 저하에 대해 감수할 수 있을 것으로 보인다.

CA_11. 주소 할당을 라운드 로빈으로 하는 방식(포인터와 127 라운드) [채택]**CA_12. 빈 주소를 List로 가지고 있는 방식 [미채택]**

품질 속성	CA_11 영향	CA_12 영향
NFR_01. 디바이스 설정 시간		(-) 라운드 로빈보다 성능 상이 이점이 없는데 구현도 복잡하다.

CA_11과 CA_12는 USB Device 주소 할당에 관련된 후보 구조로서 하나의 구조를 선택해야 했다.

CA_12는 명백한 단점이 있기에 CA_11를 채택하였다.

CA_13. Hash를 통해 Known Device를 저장하는 방식 [채택]**CA_14. List를 통해 Known Device를 저장하는 방식 [미채택]**

품질 속성	CA_13 영향	CA_14 영향
NFR_01. 디바이스 설정 시간	(+) O(1) 시간 복잡도로 디바이스를 찾을 수 있다.	(-) 매치되는 디바이스를 찾을 때까지 순회를 해야하는 단점이 있다.

CA_13와 CA_14는 Known Device에 대한 메모리 저장 구조에 관련된 후보 구조로서 하나의 구조를 선택해야 했다. CA_17는 명백한 단점이 있기에 CA_13를 채택하였다.

E3. NFR_02 시스템 부팅 시간**CA_15. Flash에서 미리 테이블을 읽고 사용하기 [채택]****CA_16. Flash에서 필요할 때 접근 [미채택]**

품질 속성	CA_15 영향	CA_16 영향
NFR_01. 디바이스 설정 시간	(++) 필요한 정보를 메모리에서 즉시 읽을 수 있으므로 장점이 있다. (-) 데이터 변경 시 메모리와 Flash 간의 동기화 작업이 필요하다.	(-) 데이터 조회가 필요할 때 마다 하드웨어인 Flash에 접근해야 한다.
NFR_02. 시스템 부팅 시간	(-) Flash로부터 읽는 동작이 필요하므로 부팅 시간이 필요하다.	

CA_15와 CA_16는 Flash에서 데이터를 읽어오는 시점에 관련된 후보 구조로서 하나의 구조를 선택해야 했다. CA_15은 부팅시간이 늘어나는 단점이 있었지만 한번의 접근으로 모든 데이터를 읽어온다면 필요한 정보가 있을 때 매번 접근하지 않아도 되는 장점이 더 큰 것으로 판단되어 CA_15을 채택하였다. 다만 CA_15는 Flash와 인메모리의 데이터 간의 동기화가 중요하고 제대로 되지 않으면 데이터 무결성이 깨지므로 구현 단계에서 각별한 주의가 요구된다.

CA_17. 부팅시 모든 모듈에게 초기화 명령 전달 [미채택]**CA_18. 시스템 초기화 및 종료를 위한 모듈 [채택]**

품질 속성	CA_17 영향	CA_18 영향
NFR_02. 시스템 부팅 시간		(+) 초기화 순서가 중요한 경우 이를 컨트롤 할 수 있는 모듈이 존재하므로 장점이 있다.

CA_17와 CA_18는 초기화 프로세스의 주체에 관련된 후보 구조로서 하나의 구조를 선택해야 했다. CA_18은 명백한 장점이 있기에 CA_18를 채택하였다.

E4. QA_02 지원 가능한 USB Class 확장**CA_19. 허용 가능한 클래스를 OS로부터 받아와서 비트맵으로 저장하는 방식 [채택]****CA_20. 모두 허용해 놓고 거부되는게 있을 때마다 하나씩 줄여 나가는 방식 [미채택]**

품질 속성	CA_19 영향	CA_20 영향
QA_02. 지원 가능한 USB Class 확장	(+) Operation System에게 정보를 얻어오므로 변경에 유리하다.	(-) 전체 USB Class에 대한 정보를 Host Driver가 가지고 있어야 한다. 변경에 불리하다.
NFR_02. 시스템 부팅 시간	(-) 부팅 시간에 OS로부터 허용 Class를 받아오는 과정이 필요하다.	

CA_19와 CA_20는 지원 가능한 USB Class 확장의 방식에 관련된 후보 구조로서 하나의 구조를 선택해야 했다. CA_19는 부팅 프로세스가 추가되어 부팅 시간에 단점이 있었지만 주체를 Operation System으로 넘겨 동적으로 변경이 용이함을 들어 최종 구조로 채택하였다.

E5. QA_03 전송 방식 변경

CA_21. Default Pipe도 제어정보를 전송하는 파이프의 상속 구조로 두는 구조 [미채택]

CA_22. Default Pipe를 별도의 모듈로 두는 구조(Increase Cohesion) [채택]

품질 속성	CA_21 영향	CA_22 영향
QA_03. 전송 방식 변경	(++) 모든 Pipe에 대해서 동일한 상속 구조를 제공해 추가 및 제거에 용이하다.	(+) 제어 정보에 대한 전송 방식이 변경되었을 때 변경이 용이하다.
QA_04. Host Controller의 USB 지원 버전 확장		(+) USB 버전업으로 제어 정보가 추가되었을 때 변경에 용이하다.

CA_21과 CA_22는 Default Pipe에 대한 구분에 관련된 후보 구조로서 하나의 구조를 선택해야 했다. CA_21에도 장점이 존재했지만 제어 정보의 특수성 등을 고려할 때 CA_22를 채택하였다.

E6. QA_04 Host Controller의 USB 지원 버전 확장

CA_23. Full-Duplex 지원을 위해 Stream Pipe에서 읽기와 쓰기 버퍼를 분리하는 방식 [채택]

CA_24. 하나의 버퍼에서 읽기와 쓰기를 동시에 진행하며 IRP 마다 읽기인지 쓰기인지 구분하는 방식 [미채택]

품질 속성	CA_23 영향	CA_24 영향
QA_01 데이터 전송 속도	(+) 읽기와 쓰기의 Thread를 분리하여 병렬적으로 성능을 높일 수 있다.	
QA_04. Host Controller의 USB 지원 버전 확장	(+) Half-Duplex에서 Full-Duplex로 확장되어도 변경이 용이하다.	(-) 버퍼의 용도를 구분할 수 없어 변경에 용이하지 않다.

CA_23과 CA_24은 읽기와 쓰기 버퍼의 분리에 관련된 후보 구조로서 하나의 구조를 선택해야 했다. 성능 측면이나 Full-Duplex 추후 지원을 고려했을 때 CA_23을 채택하였다.

CA_25. Stream Pipe의 State Pattern 적용 [채택]

CA_28의 경우 USB Specification에 정리된 State를 State Machine을 통해 관리하고, QS_04처럼 새로운 버전으로 인해 State가 추가되어도 변경이 용이하다는 점을 들어 채택하였다.

CA_26. ISP을 준수하는 Pipe의 다양한 인터페이스 제공 [채택]

CA_29의 경우 ISP를 지원하는 것은 소프트웨어의 기본 원칙이기도 하지만, QA_04처럼 새로운 버전으로 인해 또 다른 책임이 추가되더라도 인터페이스를 추가해 대처할 수 있어 채택하였다.

CA_27. 전송 방법에 따라 다형성으로 표현하는 방법 [채택]

CA_30의 경우 QA_04의 경우처럼 새로운 버전으로 인해 또 다른 전송 방식이 추가되어도 변경이 용이해 채택하였다.

E7. QA_05 동시에 연결 가능한 USB Device의 수**CA_28. Resource Observer를 둘러 리소스 현황을 주기적으로 파악하고 최적의 리소스 할당하는 방식 [채택]****CA_29. 최대 사용 자원만 파악한 뒤 요청이 올 때마다 Best Effort로 할당하고 더 이상 연결을 허용하지 않는 방식 [미채택]**

품질 속성	CA_28 영향	CA_29 영향
QA_05 동시에 연결 가능한 USB Device의 수	(+) 별도의 컴포넌트가 사용 리소스를 확인해서 최적의 리소스를 할당하고 많은 수의 디바이스를 연결할 수 있다.	

CA_28와 CA_29은 사용 리소스 활용에 관련된 후보 구조로서 하나의 구조를 선택해야 했다. CA_28에 명백한 장점이 있어 최종 구조로 채택하였다.

D7.2.1. CA_30. Watchdog 형식으로 주기적으로 가용자원을 확인하는 방식 [채택]**D7.2.2. CA_31. 특정 이벤트(디바이스 연결 등) 발생 시마다 가용자원을 확인하는 방식 [미채택]**

품질 속성	CA_30 영향	CA_31 영향
QA_05 동시에 연결 가능한 USB Device의 수		(-) 실시간으로 가용 리소스에 대해 파악하는 것이 아니라 정확하지 않은 정보로 리소스를 할당할 수 있다.

CA_30와 CA_31은 가용 리소스 파악에 관련된 후보 구조로서 하나의 구조를 선택해야 했다.

CA_31에 단점이 있어 CA_30을 최종 구조로 채택하였다.

E8. QA_06 외부 모듈 교환**CA_32. Host와의 유니폼한 인터페이스를 추상화하는 방식 [채택]****CA_33. 주요 운영체제에 대한 여러가지 인터페이스 제공 [미채택]**

품질 속성	CA_32 영향	CA_33 영향
QA_06 외부 모듈 교환	(+) OS와의 오퍼레이션에 대해 추상화된 인터페이스를 가지면 이를 지원하는 다른 Operation System에도 변경없이 적용이 가능하다.	(-) 인터페이스가 제공되는 OS하고만 통신할 수 있다.
NFR_02. 시스템 부팅 시간		(+) 특정 OS에 최적화된 부팅 프로세스로 인해 시간이 절약 될 가능성이 있다.

CA_32와 CA_33는 Operation System과의 인터페이스와 관련된 후보 구조로서 하나의 구조를 선택해야 했다. CA_32는 General한 방식이지만 그만큼 변경용이성에 장점이 있어 최종 구조로 채택하였다. 다만 주요 운영체제가 가지는 오퍼레이션을 잘 확인하고 3대 주요 운영체제(Windows, Mac, Linux)는 모두 지원이 가능한 형태로 세부 설계 해야한다.

CA_34. LED와의 인터페이스를 위한 어댑터 [채택]**CA_35. Host Controller와의 인터페이스를 위한 어댑터 [채택]**

CA_34과 CA_35은 어댑터를 사용해 QS_06 외부 모듈 교환에 있어 변경용이성을 제공하므로 최종 구조로 채택하였다.

E9. QA_07 디바이스 정보 조회 시간**CA_36. Device 연결구조를 트리로 표현하는 구조 [미채택]****CA_37. 리스트로 표현하는 구조 [미채택]****CA_38. 고유 아이디를 가지고 해시 구성 [채택]****CA_39. 구조체 배열로 구성하는 방법 [미채택]**

CA_36, CA_37, CA_38, CA_39는 디바이스의 연결 구조와 관련된 후보 구조로서 하나의 구조를 선택해야 했다. CA_38은 순회가 까다롭고 USB Hub와의 연결정보를 따로 표현해야하는 단점이 있었지만 특정 디바이스를 찾는데 있어 절대적으로 큰 우위를 가지고 있어 성능 측면에서 최종 구조로 채택하였다.

F. 최종 구조 설계

F1. Deployment View 측면

본 시스템의 시스템 구조는 멀티 쓰레드 구조로 병렬성이 필요한 작업에 따라 여러 개의 쓰레드로 나뉘어 동작한다. 그에 대한 시스템 최종 구조는 아래 그림과 같다.

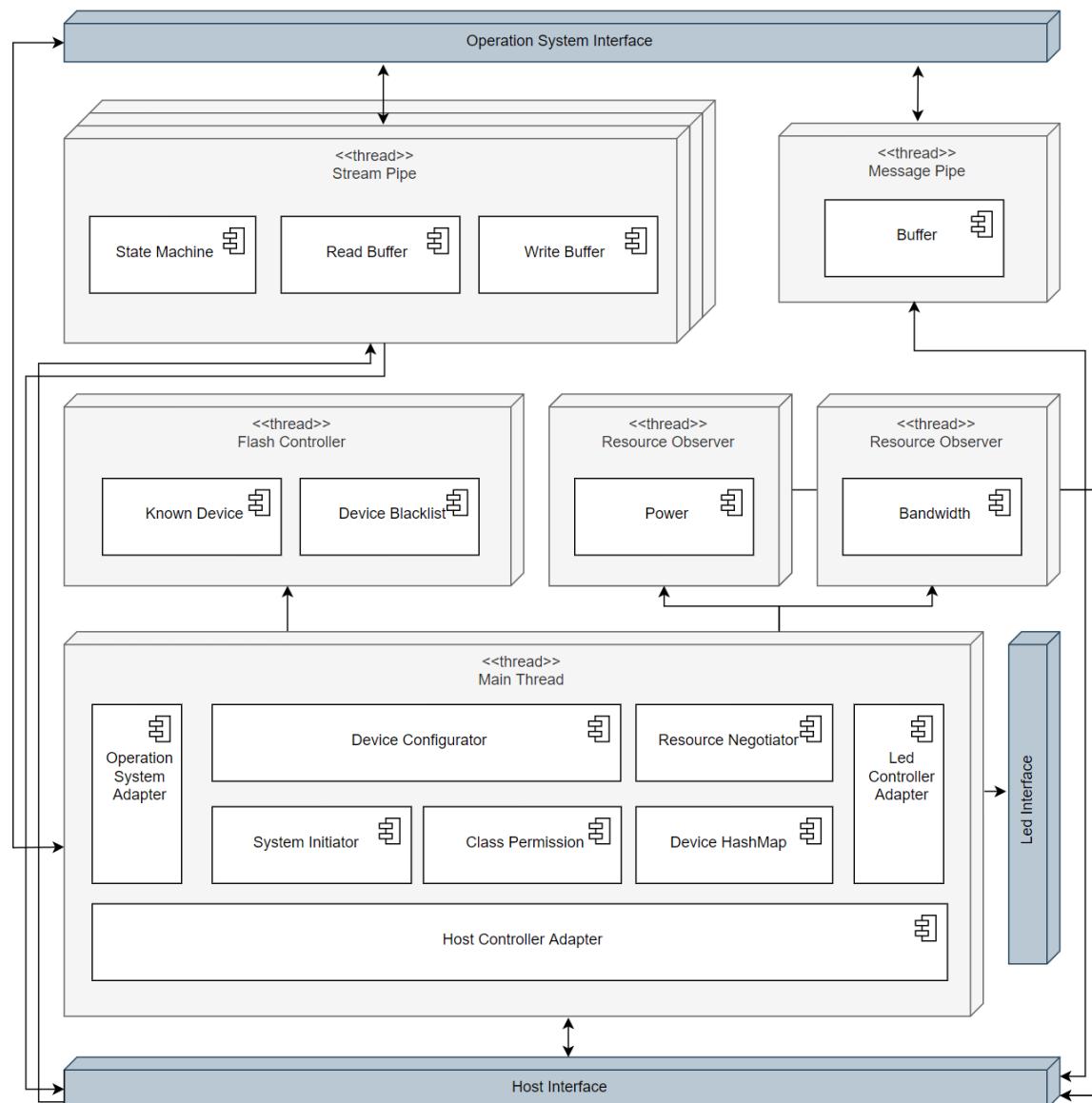


Figure 96 - Deployment View

F1.1. Stream Pipe의 쓰레드 배치 근거

CA_05(Stream Pipe가 생성되면 별도의 쓰레드를 할당하는 방식)의 핵심적인 아이디어는 전송 속도를 쓰레드 우선순위에 따른 스케줄링에 따라 조절하는 것이다. Stream Pipe는 네 가지 전송 타입을 가지고 있는데 특성에 따라 전송 주기나 속도가 다르다. 이러한 전송 특성에 따라 쓰레드 우선순위를 부여하여 스케줄링된 쓰레드가 Host Controller의 Buffer에 데이터를 넣어 전송할 수 있도록 하기 위해 Stream Pipe가 개설될 때마다 별도의 쓰레드로 배치한다. 이는 근본적으로 QA_01(데이터 전송 속도) 충족을 위해 전송의 병렬성을 확보하고 최종적으로 Host Controller의 가용성을 높이기 위함이다. 공유 자원에 대한 접근은 Stream Pipe 개설 시 한번 접근하고 서로의 간섭없이 스케줄링 만으로 전송의 병렬성을 확보하여 동기화 문제를 해결하도록 한다.

F1.2. Masseage Pipe의 쓰레드 배치 근거

CA_22(Default Pipe를 별도의 모듈로 두는 구조)를 채택하여 Stream Pipe가 전송 중 일지라도 제어 정보의 전송이 보장될 수 있도록 한다. 그를 위해 Message Pipe는 별도로 하나의 쓰레드로 배치한다. 이는 NFR_01(디바이스 설정 시간) 확보를 위한 최대한 빠른 제어 신호의 전송을 확보하기 위함이다.

F1.3. Resource Observer의 쓰레드 배치 근거

CA_28(Resource Observer를 뒤서 리소스 현황을 주기적으로 파악하고 최적의 리소스 할당하는 방식)과 CA_30(Watchdog 형식으로 주기적으로 가용자원을 확인하는 방식)을 채택하여 데이터 전송 중 일지라도 주기적으로 가용 가능한 리소스를 파악할 수 있도록 한다. 다만 너무 짧은 주기로 리소스를 파악하여 전송을 위한 Stream Pipe의 스케줄링에 차질이 없도록 운영환경에서 주기에 대한 Threshold를 잘 찾아야 한다. 이는 최대 효율의 리소스를 할당하여 QA_05(동시에 연결 가능한 USB Device의 수)를 확보하기 위함이다.

F1.4. Flash Controller의 쓰레드 배치 근거

CA_09(Known Device에 대해서 Fast Track으로 연결하는 구조)와 CA_13(Hash를 통해 Known Device를 저장하는 방식)을 채택하여 연결 이력을 Flash 장치에 저장해야 하는 니즈가 있다. 이러

한 상황에서 NFR_02(시스템 부팅 시간)에 따라 시스템 부팅 시간을 최소화하기 위해서는 Flash라는 하드웨어에서 병렬적으로 데이터를 읽어오는 방식이 필요할 것으로 판단하여 별도의 쓰레드로 배치하였다.

F2. Module View 측면

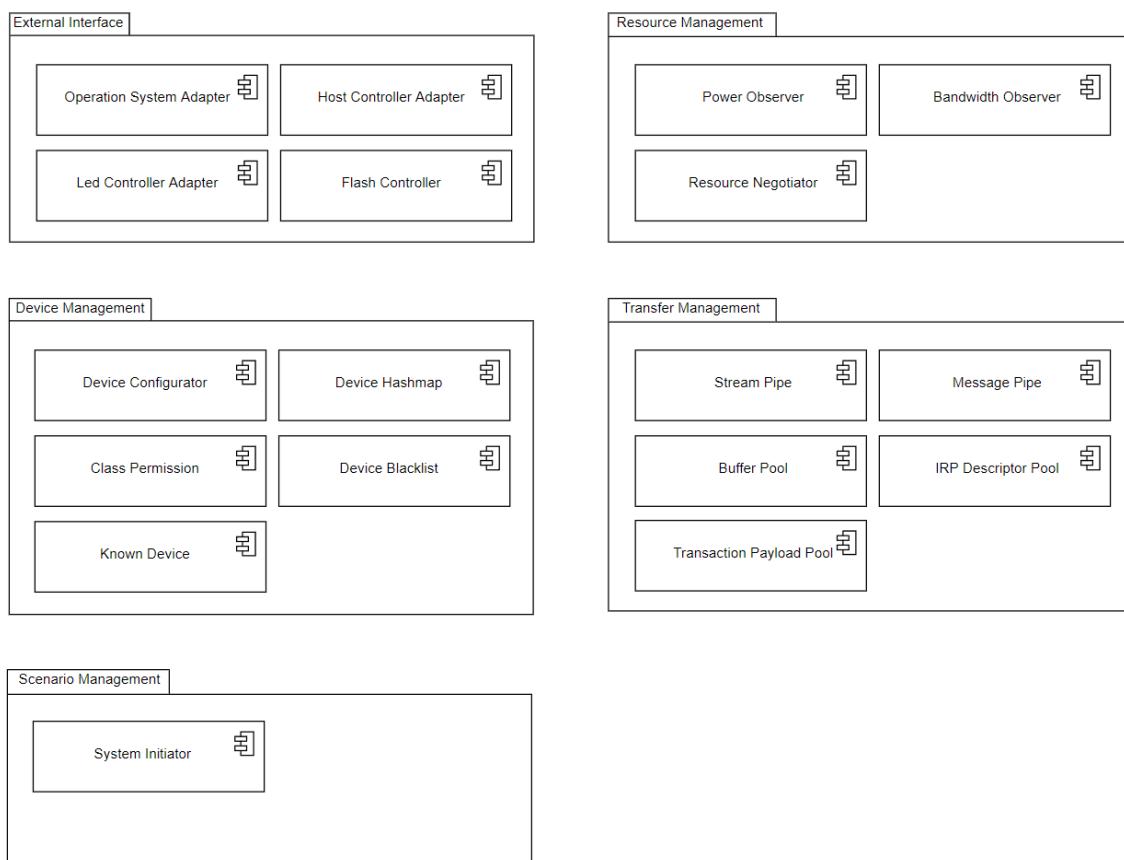


Figure 97 - Module View

최종 구조를 반영한 Module View는 위와 같다. 기능의 유사성과 연관도를 고려하여 External Interface, Resource Management, Device Management, Transfer Management, Scenario Management 등 다섯 개의 패키지로 구분하였으며 이후 파일 및 디렉토리 구조와 개발 조직 등의 영향을 고려하여 구분하였다.

F3. 최종 구조의 Risk 관리

F3.1. Payload 복사 문제

데이터 쓰기와 관련하여 CA_01을 통해 데이터가 복사가 필요하지 않은 구조를 만들었으나, 여전히 읽기와 관련해서는 Payload 복사가 필요하다는 문제가 있다. 이는 피하기 어려운 구조로 이에 대한 보완으로 DMA와 같은 하드웨어 장치를 도입하여 데이터 복사 및 이동에 대한 성능 문제를 보완한다면 이 Risk에 대비할 수 있을 것으로 보인다.

F3.2. Device HashMap의 순회 및 Hierarchy 표현 문제

Hash 구조는 원하는 자료를 즉시 꺼낼 수 있는 장점이 있지만 트레이드오프로서 순회에 적합하지 않다. 이를 보완하기 위해 Hash 요소를 가리키는 Reference의 List를 따로 구성하여 순회에 사용할 수 있다. 또한 논리적으로는 동급으로 사용되는 USB Device라고 할지라도 Hub의 제거와 같은 이벤트가 발생하면 Hierarchy를 따져 처리할 수밖에 없다. 따라서 Hash에 저장하는 데이터에 자식 관계를 표현하도록 하여 일정 부분 계층 구조를 표현하도록 할 수 있다.

F3.3. 멀티 쓰레드 환경에서 공유 자원에 대한 Race Condition 문제

Race Condition은 예기치 못한 문제를 발생시킨다는 문제도 있지만 이를 해결하기 위한 Lock이 성능 저하의 원인이 될 수 있다는 점도 가지고 있다. 이러한 치명적인 트레이드오프를 줄이기 위해 Lock-Free 자료구조의 적극적인 도입은 성능 저하를 보완할 수 있는 방법으로 보인다.