|  |
| --- |
| **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 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[1. 시스템 개요 3](#_Toc479150191)

[2. 요구사항 4](#_Toc479150192)

[2.1. 기능적 요구사항 4](#_Toc479150193)

[2.2. 비기능적 요구사항 4](#_Toc479150194)

[2.3. 품질 속성 5](#_Toc479150195)

[부록 8](#_Toc479150198)

[A. 도메인 모델 9](#_Toc479150199)

[B. 품질 시나리오 10](#_Toc479150200)

[C. 품질 시나리오 분석 11](#_Toc479150201)

[D. 후보 구조 12](file:///C:\Users\김성후\OneDrive\문서\architect\reference\requirement\구조설계서(배포).docx#_Toc479150202)

[E. 후보 구조 평가 13](file:///C:\Users\김성후\OneDrive\문서\architect\reference\requirement\구조설계서(배포).docx#_Toc479150203)

[F. 최종 구조 설계 14](#_Toc479150204)

# 시스템 개요

* 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.

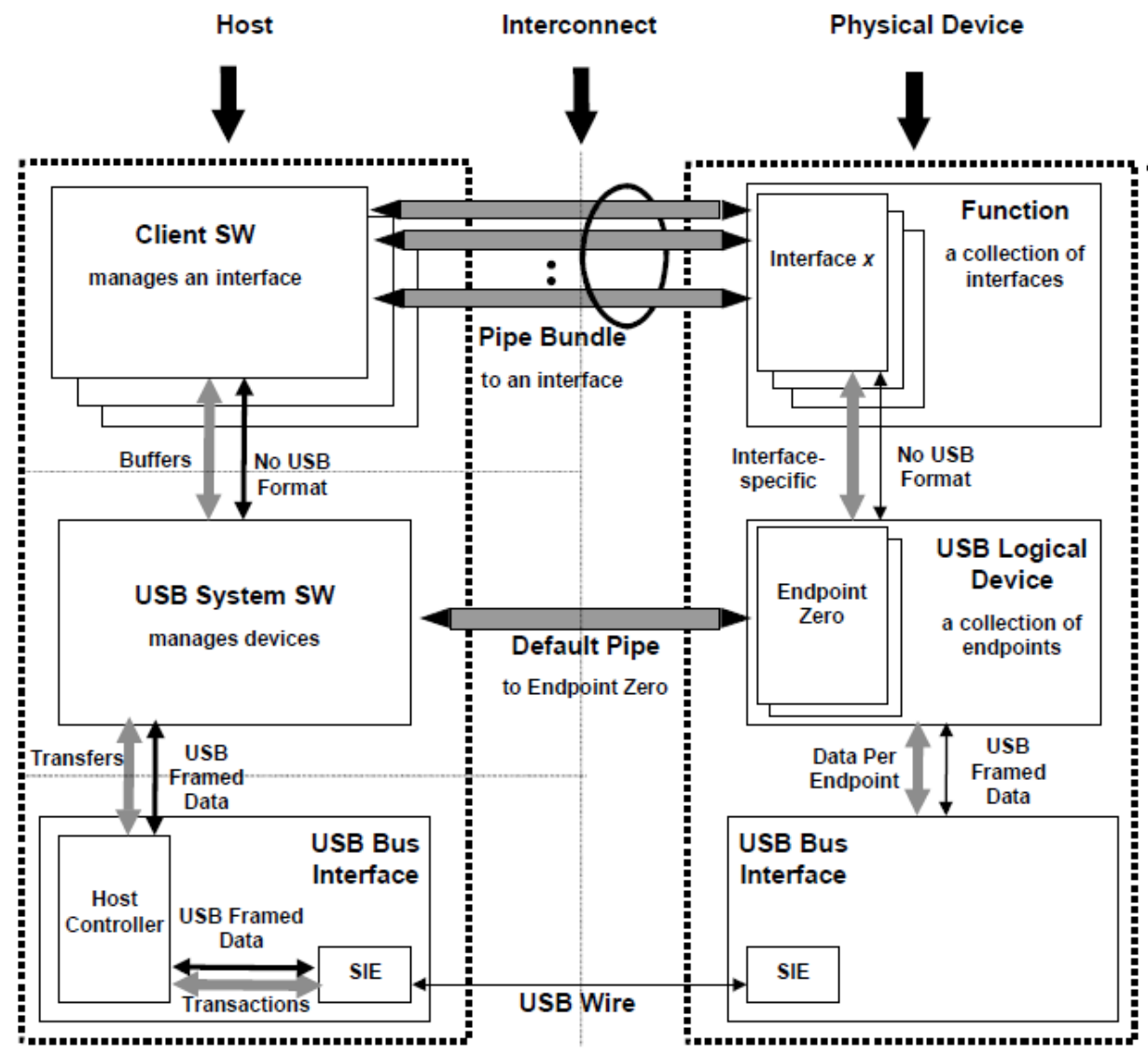


그림 - USB 시스템의 구조

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 Software와 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. **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.



그림 – Note PC 제품의 USB Port

* + 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 기대한다. 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. **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.
* 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. **시스템 정의**

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

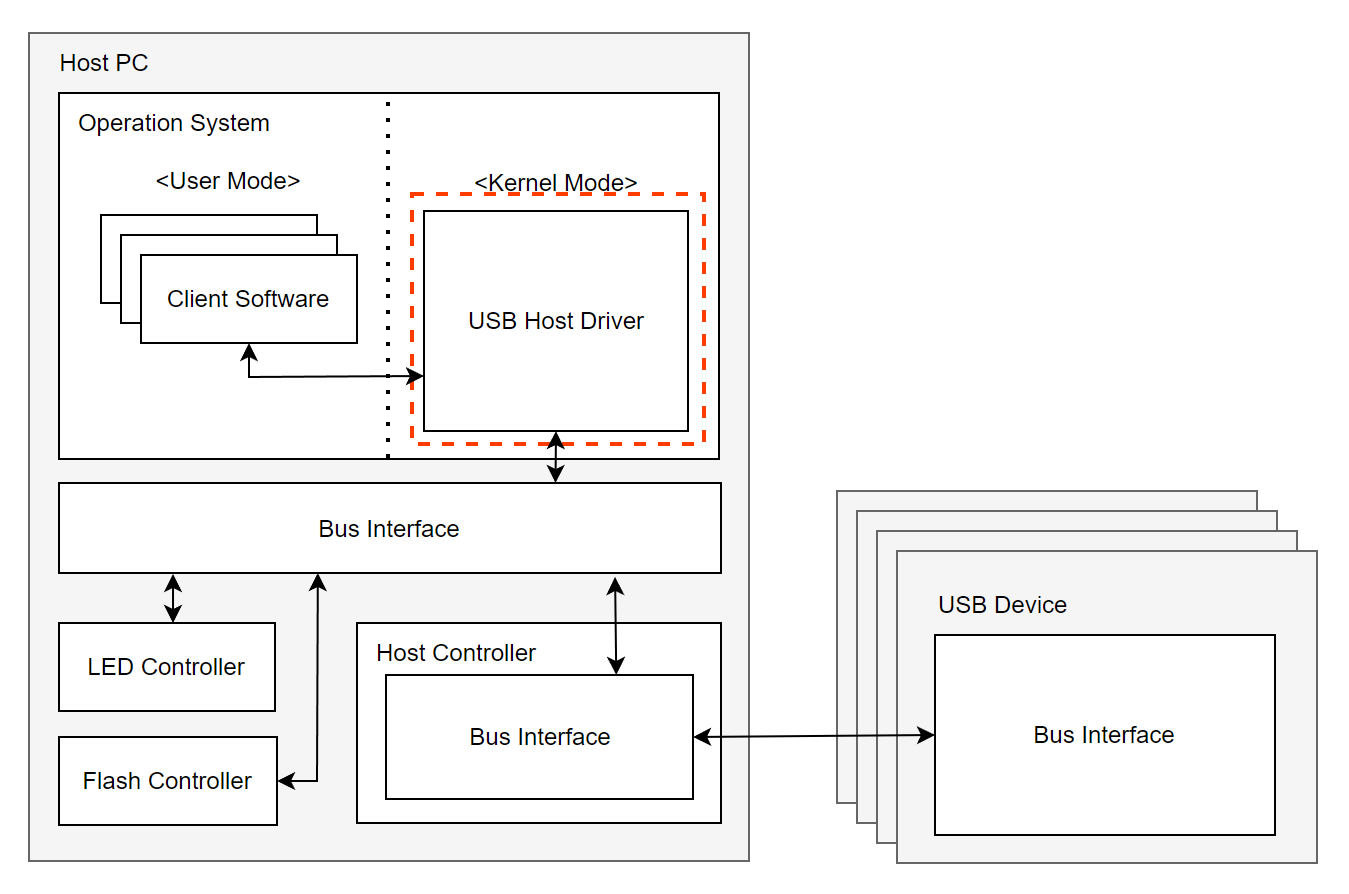


그림 3 - 시스템 전체 구조와 개발 범위

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. **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. **시스템 경계**

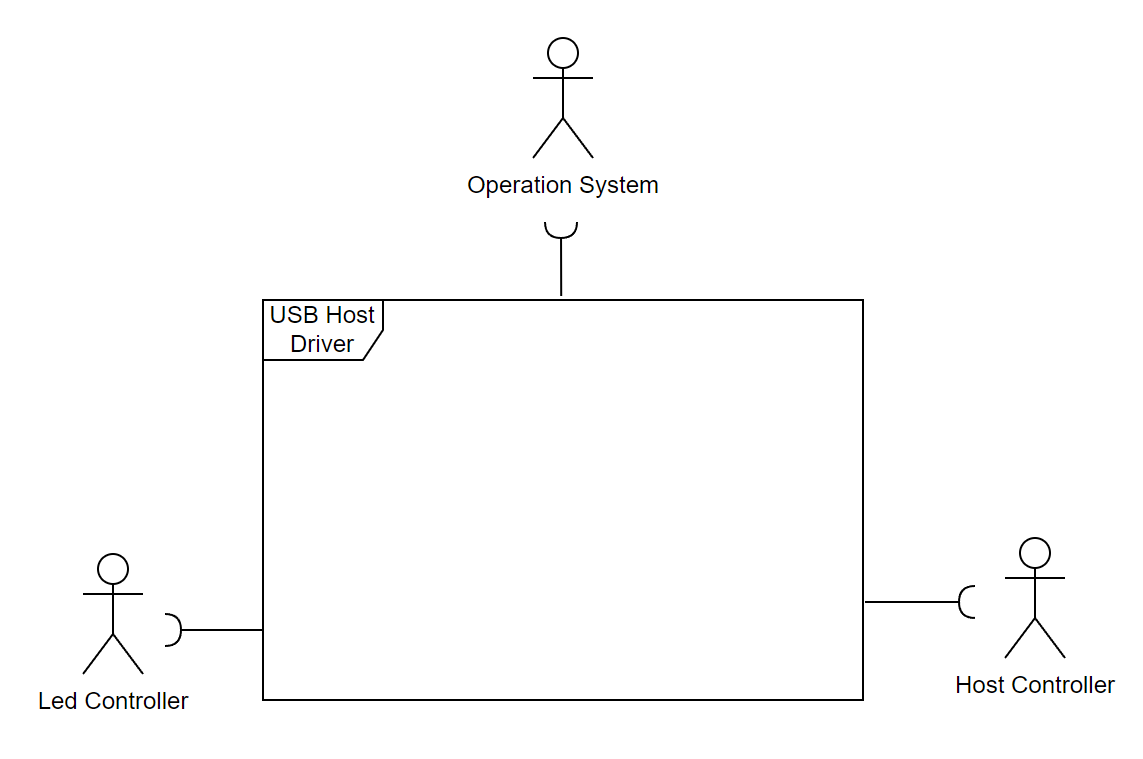


그림 4 - 시스템 경계

본 과제에서 다루고자 하는 USB Host Driver의 경계는 아래 그림 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. **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.

# 요구사항

## 기능적 요구사항

* + 1. **Use Case Diagram**

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

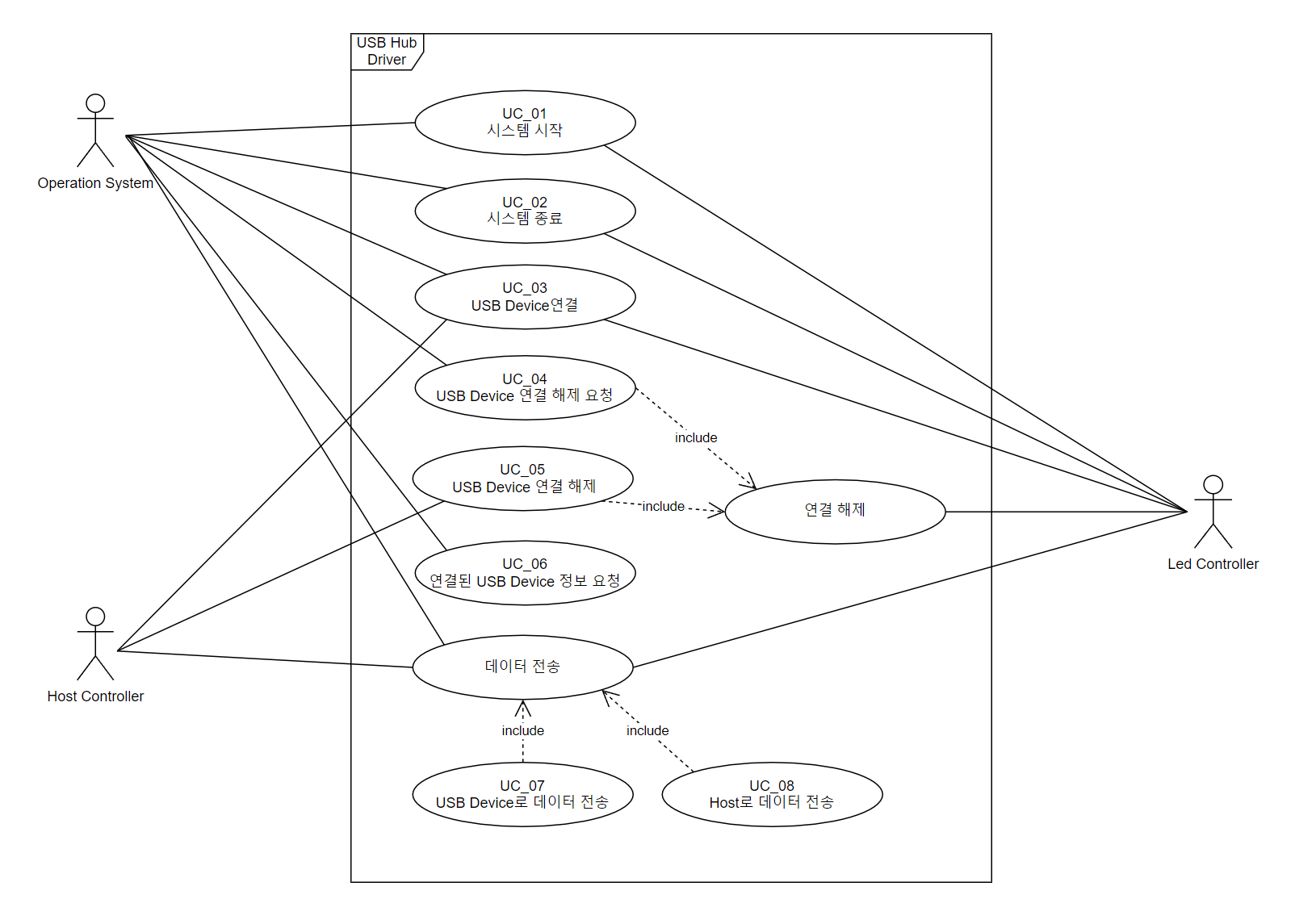


그림 5 – Use Case Diagram

* + 1. **Use Case 명세**
       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 | 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 | - |

* + - 1. **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 | 1. Operation System은 시스템에게 System Call을 통해 연결 해제를 통보한다. 2. 시스템은 모든 연결된 USB Device들에게 연결 해제를 통보하고 해당 USB Device와 Mapping된 Stream Pipe를 제거한다. 3. 시스템은 Message Pipe를 제거한다. 4. 시스템은 Led Interface에게 모든 LED 점등 해제를 명령한다. 5. 시스템은 가용 자원 정보 수집을 중단한다. |
| 추가 동작 | - |

* + - 1. **USB Device 연결**

|  |  |
| --- | --- |
| **UC\_03** | USB Device 연결 |
| 설명 | 시스템은 USB Device와 연결되면 USB Specification에 정의된 열거 절차를 진행한다. 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 | 1. Host Controller는 시스템에게 USB Device의 연결을 통보한다. 2. 시스템은 Host Controller에게 새로 연결된USB Device에 최소 전력(100mA)를 공급을 명령한다. 3. Host Controller는 시스템에게 전력의 공급을 통보한다. 4. 시스템은 Host Controller에게 USB Device와 시스템간 통신할 수 있는 주소를 할당해서 전달한다. 5. Host Controller는 시스템에게 새로 연결된 USB Device의 VID, PID를 전달한다. 6. 시스템은 새로 연결된 USB Device가 이전 연결된 적 있는지 확인한다.   6-1. 연결된 적 없는 장치라면 Host Controller에게 USB Device descriptor를 요청한다.  6-2. Host Controller는 시스템에게 새로 연결된 USB Device의 USB Device descriptor를 전달한다.  6-3. 시스템은 USB Device descriptor에서 USB Class를 확인하고 수용할 수 있는지 확인한다.   1. 시스템은 USB Device의 요구 리소스에 대한 가용 자원이 충분한지 확인한다. 충분하지 않다면 시스템은 요구 사항과 가용 자원에 대한 협상된 값을 도출한다. 2. 시스템은 Host Controller에게 7번에서 계산된 자원만큼의 전력을 새로 연결된 USB Device에 공급하도록 명령한다. 3. 시스템은 LED Controller에게 USB Device가 연결된 위치의 LED를 점등하도록 명령한다. 4. 시스템은 Host와 USB Device가 통신할 수 있는 Stream Pipe를 개설한다. 5. 시스템은 USB Device가 연결된 적 없던 장치라면 Flash Controller에게 Known Device로 저장을 요청한다. 6. 시스템은 Operation System에게 새로운 Stream Pipe가 개설되었음을 통보한다. |
| 추가 동작 | 1. (6-3에서) 수용할 수 없는 장치라면 Operation System에게 수용할 수 없는 USB Device가 연결되었음을 통보한다. 2. (9에서) USB Device가 루트 허브에 연결된 장치가 아니라면 LED 점등 절차는 생략한다. |

* + - 1. **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 운영체제의 “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 | 1. Operation System은 시스템에게 System Call을 통해 특정 USB Device 연결 해제 요청을 한다. 2. 시스템은 Host Controller에게 특정 USB Device 와의 연결 해제를 요청한다. 3. 시스템은 특정 USB Device와 Mapping된 Stream Pipe를 해제한다. 4. 시스템은 LED Interface에게 특정 USB Device와의 매핑 해제와 점등 해제를 명령한다. 5. 시스템은 특정 USB Device의 컨텍스트 정보를 삭제한다. |
| 추가 동작 | 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. |

* + - 1. **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 | 1. Host Controller가 시스템에게 특정 USB Device의 연결 해제를 통보한다. 2. 시스템은 Operation System에게 특정 USB Device의 연결 해제를 통보한다. 3. Operation System은 시스템의 Stream Pipe에 남은 데이터를 Read한 뒤, 시스템에게 디바이스 해제 요청을 한다. 4. 시스템은 특정 USB Device와 Mapping된 Stream Pipe를 해제한다. 5. 시스템은 LED Interface에게 특정 USB Device와의 매핑 해제와 점등 해제를 명령한다. 6. 시스템은 특정 USB Device의 컨텍스트 정보를 삭제한다. |
| 추가 동작 | 1. (5에서) USB Device가 루트 허브에 연결된 장치가 아니라면 LED 점등 해제 절차는 생략한다. |

* + - 1. **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 | 1. Operation System은 시스템에게 연결된 USB Device들의 정보를 요청한다. 2. 시스템은 모든 연결된 USB Device 정보를 수집한다. 3. 시스템은 Operation System에게 수집한 정보를 전달한다. |
| 추가 동작 | 1. (2에서) 연결된 USB Device가 없는 경우 바로 3번으로 넘어간다. |

* + - 1. **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 | 1. Operation System이 시스템에게 System Call을 통해 특정 USB Device의 데이터 읽기를 요청한다. 2. 시스템은 USB Device를 통해 받은 Transaction들을 묶어 IRP 형태로 응답한다. 3. 시스템은 LED Interface에게 점멸 해제를 명령한다. 4. 시스템은 내부 USB Device에 대한 상태를 “SUSPEND”로 변경한다. |
| 추가 동작 | - |

* + - 1. **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 | 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.   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.   1. 시스템은 LED Interface에게 점멸 해제를 명령한다. 2. 시스템은 내부 USB Device에 대한 상태를 “SUSPEND”으로 변경한다. |
| 추가 동작 | - |

## 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 Device를 USB 슬롯에 삽입한다. | |
| 반응 | 1. Host Controller는 시스템에게 USB Device의 연결을 통보한다. 2. 시스템은 Host Controller에게 새로 연결된USB Device에 최소 전력(100mA)를 공급을 명령한다. 3. Host Controller는 시스템에게 전력의 공급을 통보한다. 4. 시스템은 Host Controller에게 USB Device와 시스템간 통신할 수 있는 주소를 할당해서 전달한다. 5. Host Controller는 시스템에게 새로 연결된 USB Device의 VID, PID를 전달한다. 6. 시스템은 새로 연결된 USB Device가 이전 연결된 적 있는지 확인한다.   6-1. 연결된 적 없는 장치라면 Host Controller에게 USB Device descriptor를 요청한다.  6-2. Host Controller는 시스템에게 새로 연결된 USB Device의 USB Device descriptor를 전달한다.  6-3. 시스템은 USB Device descriptor에서 USB Class를 확인하고 수용할 수 있는지 확인한다.   1. 시스템은 USB Device의 요구 리소스에 대한 가용 자원이 충분한지 확인한다. 충분하지 않다면 시스템은 요구 사항과 가용 자원에 대한 협상된 값을 도출한다. 2. 시스템은 Host Controller에게 7번에서 계산된 자원만큼의 전력을 새로 연결된 USB Device에 공급하도록 명령한다. 3. 시스템은 LED Controller에게 USB Device가 연결된 위치의 LED를 점등하도록 명령한다. 4. 시스템은 Host와 USB Device가 통신할 수 있는 Stream Pipe를 개설한다. 5. 시스템은 USB Device가 연결된 적 없던 장치라면 Flash Controller에게 Known Device로 저장을 요청한다.   시스템은 Operation System에게 새로운 Stream Pipe가 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 | 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.   시스템은 가용 자원 정보를 저장한다. | |
| 측정 | [부팅 시간] = [LED 장치의 최초 점멸 시각] – [시스템에 전원이 공급되는 시각] | |
| 제약 | [부팅 시간] <= 100ms | |

## 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 Device가 데이터를 전송한다. | |
| 반응 | 1. Operation System이 시스템에게 System Call을 통해 특정 USB Device의 데이터 읽기를 요청한다. 2. 시스템은 USB Device를 통해 받은 Transaction들을 묶어 IRP 형태로 응답한다. 3. 시스템은 LED Interface에게 점멸 해제를 명령한다. 시스템은 내부 USB Device에 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 | 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. | |
| 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 | N개의 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 | N개의 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 Device를 USB 슬롯에 삽입한다. | |
| 반응 | 1. Host Controller는 시스템에게 USB Device의 연결을 통보한다. 2. 시스템은 Host Controller에게 새로 연결된USB Device에 최소 전력(100mA)를 공급을 명령한다. 3. Host Controller는 시스템에게 전력의 공급을 통보한다. 4. 시스템은 Host Controller에게 USB Device와 시스템간 통신할 수 있는 주소를 할당해서 전달한다. 5. Host Controller는 시스템에게 새로 연결된 USB Device의 VID, PID를 전달한다. 6. 시스템은 새로 연결된 USB Device가 이전 연결된 적 있는지 확인한다.   6-1. 연결된 적 없는 장치라면 Host Controller에게 USB Device descriptor를 요청한다.  6-2. Host Controller는 시스템에게 새로 연결된 USB Device의 USB Device descriptor를 전달한다.  6-3. 시스템은 USB Device descriptor에서 USB Class를 확인하고 수용할 수 있는지 확인한다.   1. 시스템은 USB Device의 요구 리소스에 대한 가용 자원이 충분한지 확인한다. 충분하지 않다면 시스템은 요구 사항과 가용 자원에 대한 협상된 값을 도출한다. 2. 시스템은 Host Controller에게 7번에서 계산된 자원만큼의 전력을 새로 연결된 USB Device에 공급하도록 명령한다. 3. 시스템은 LED Controller에게 USB Device가 연결된 위치의 LED를 점등하도록 명령한다. 4. 시스템은 Host와 USB Device가 통신할 수 있는 Stream Pipe를 개설한다. 5. 시스템은 USB Device가 연결된 적 없던 장치라면 Flash Controller에게 Known Device로 저장을 요청한다. 6. 시스템은 Operation System에게 새로운 Stream Pipe가 개설되었음을 통보한다. | |
| 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 | N개의 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 Device의 정보를 요청한다. | |
| 반응 | 1. Operation System은 시스템에게 연결된 USB Device들의 정보를 요청한다. 2. 시스템은 모든 연결된 USB Device 정보를 수집한다. 3. 시스템은 Operation System에게 수집한 정보를 전달한다. | |
| 측정 | [디바이스 정보 조회 시간] = [OS가 USB Device의 정보를 요청하는 시각] – [시스템이 USB Device의 정보를 OS에게 전달하는 시각] | |

부록

[A. 도메인 모델 9](#_Toc479150206)

[B. 품질 시나리오 10](#_Toc479150207)

[C. 품질 시나리오 분석 11](#_Toc479150208)

[D. 후보 구조 12](#_Toc479150209)

[E. 후보 구조 평가 13](#_Toc479150210)

[F. 최종 구조 설계 14](#_Toc479150211)

1. 도메인 모델

본 과제에서 USB Hub Driver에 대한 도메인 모델은 그림 6과 같다. 본 도메인 모델은 Boundary/Control/Entity 모델로 작성되었다. Boundary는 외부의 Interface와 상호 작용하고, Control은 Boundary와 Entity를 연결해주며, Entity는 시스템에 필요한 데이터를 저장한다.

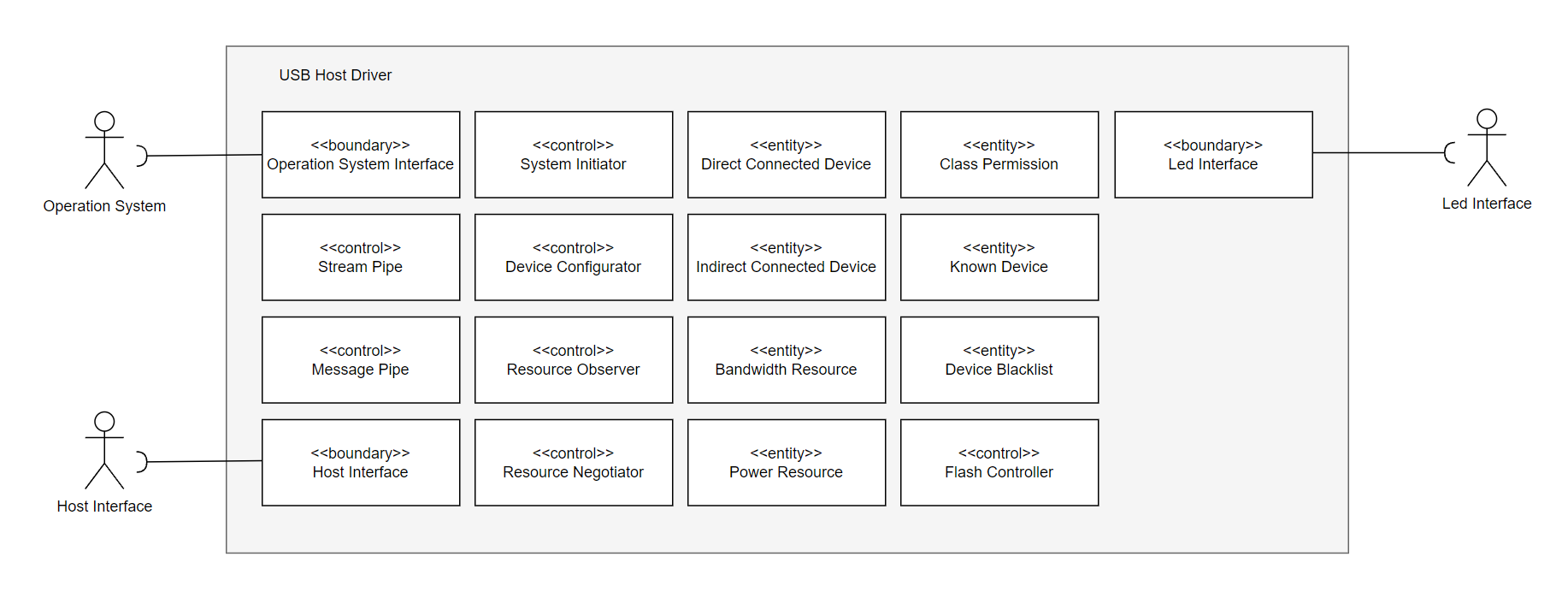


그림 - Domain Model

**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 y | 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 Controller와의 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 가용 가능한 자원에 대한 정보를 주기적으로 얻어와 저장하는 역할을 하는 도메인 모델이다. |
| Resource Negotiator | Control | 가용 가능한 자원을 조회하고 USB Device에서 요청하는 자원에 대한 가용 범위를 결정하는 도메인 모델이다. |
| Bandwidth Resource | Entity | 가용 가능한 대역폭 자원에 대한 정보를 저장하는 도메인 모델이다. |
| Power Resource | Entity | 가용 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 | 허용가능한 USB Class를 저장하는 도메인 모델이다. Host 시스템에게 질의하여 수용가능한 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**

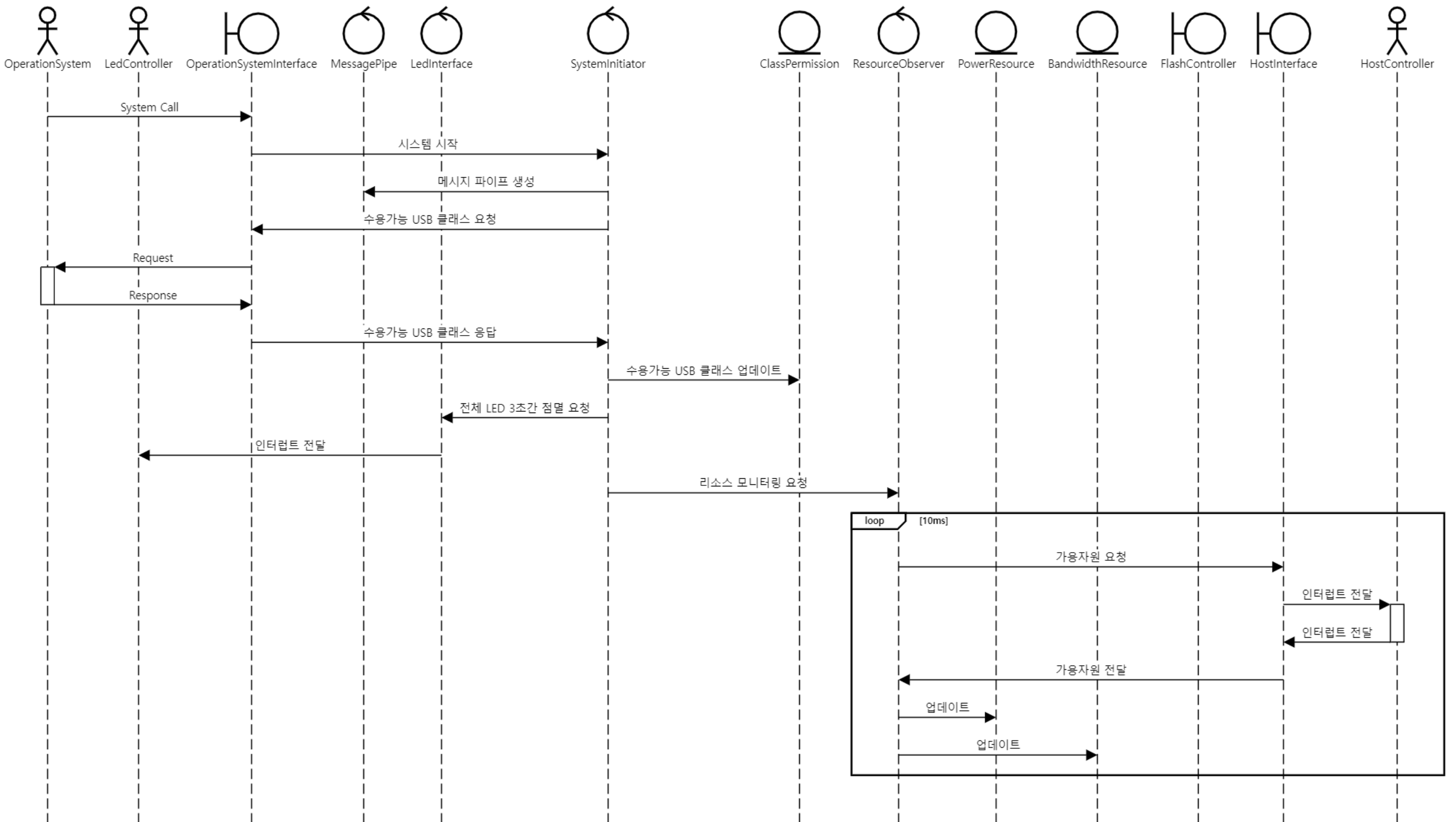


그림 – 시스템 시작

**A2.2. 시스템 종료**

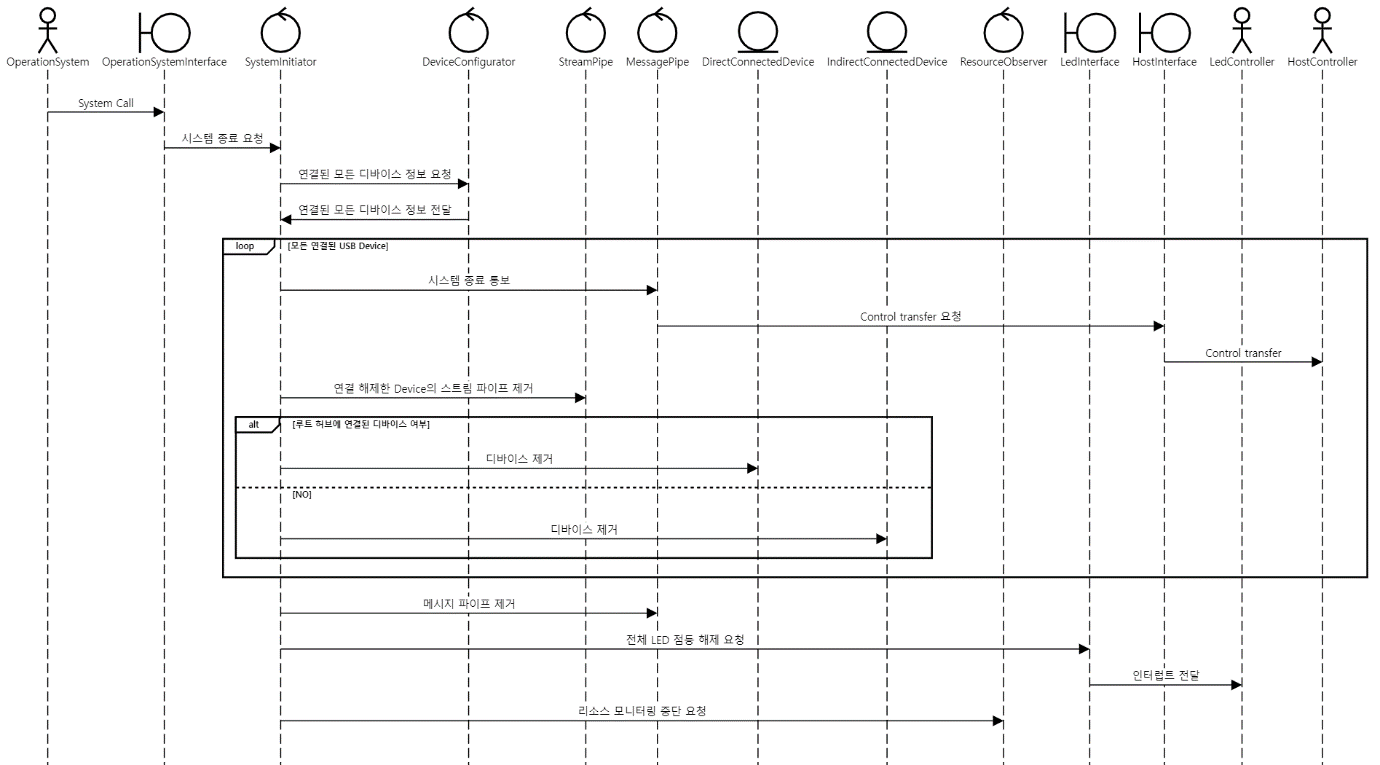
****

그림 – 시스템 종료

**A2.3. USB Device 연결**

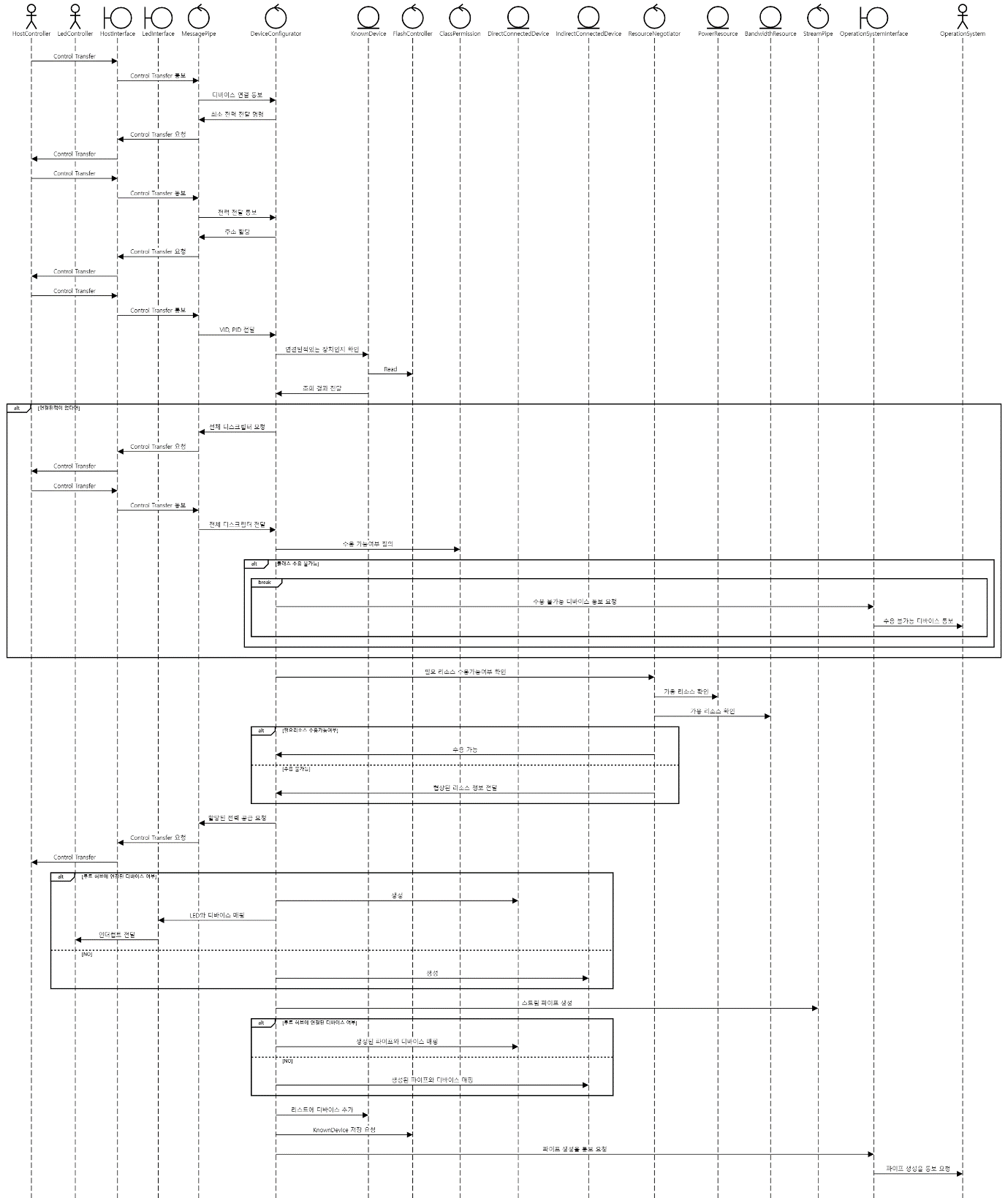
****

그림 - USB Device 연결

**A2.4. USB Device 연결 해제 요청**

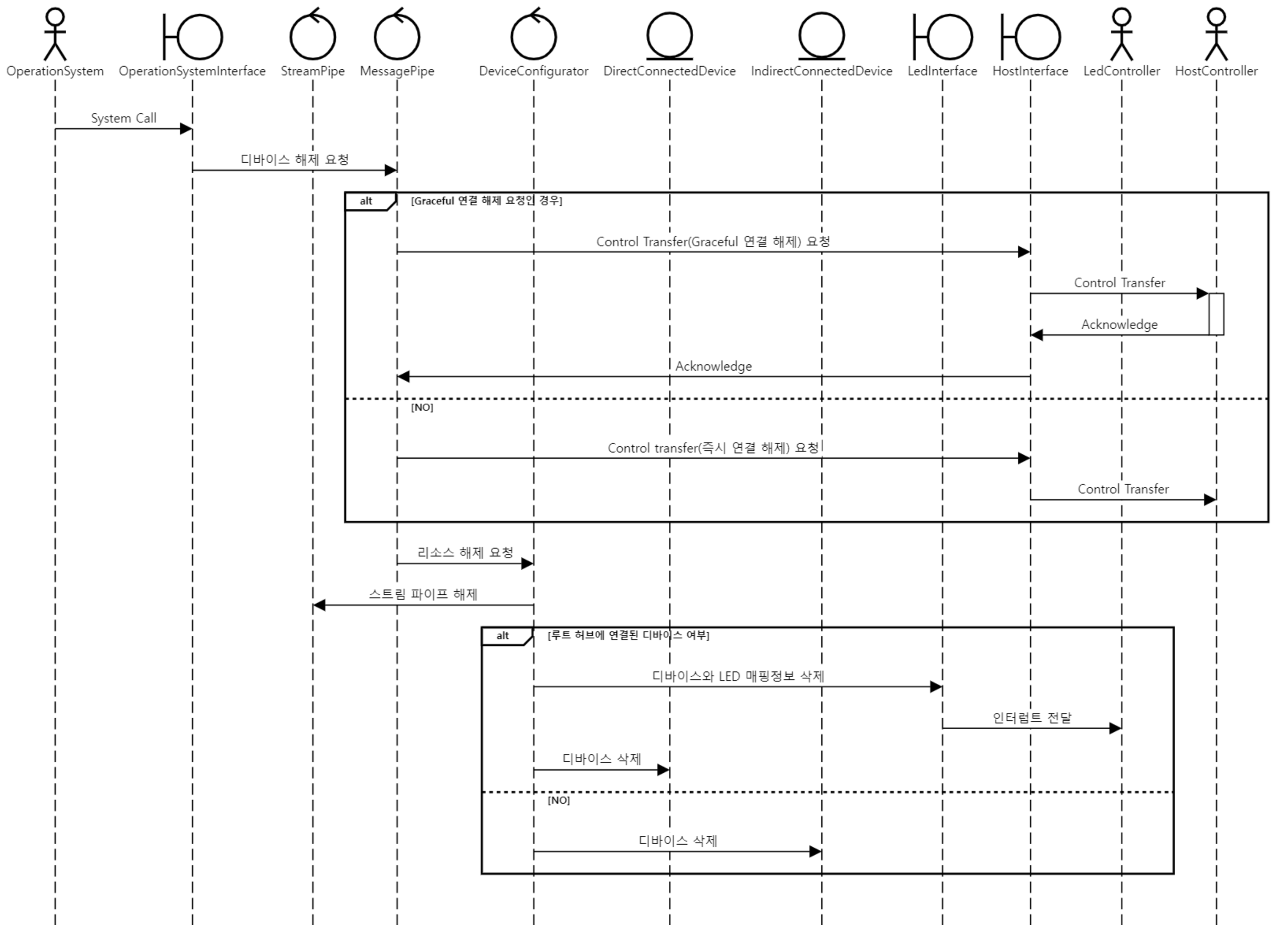


그림 - USB Device 연결 해제 요청

**A2.5. USB Device 연결 해제**

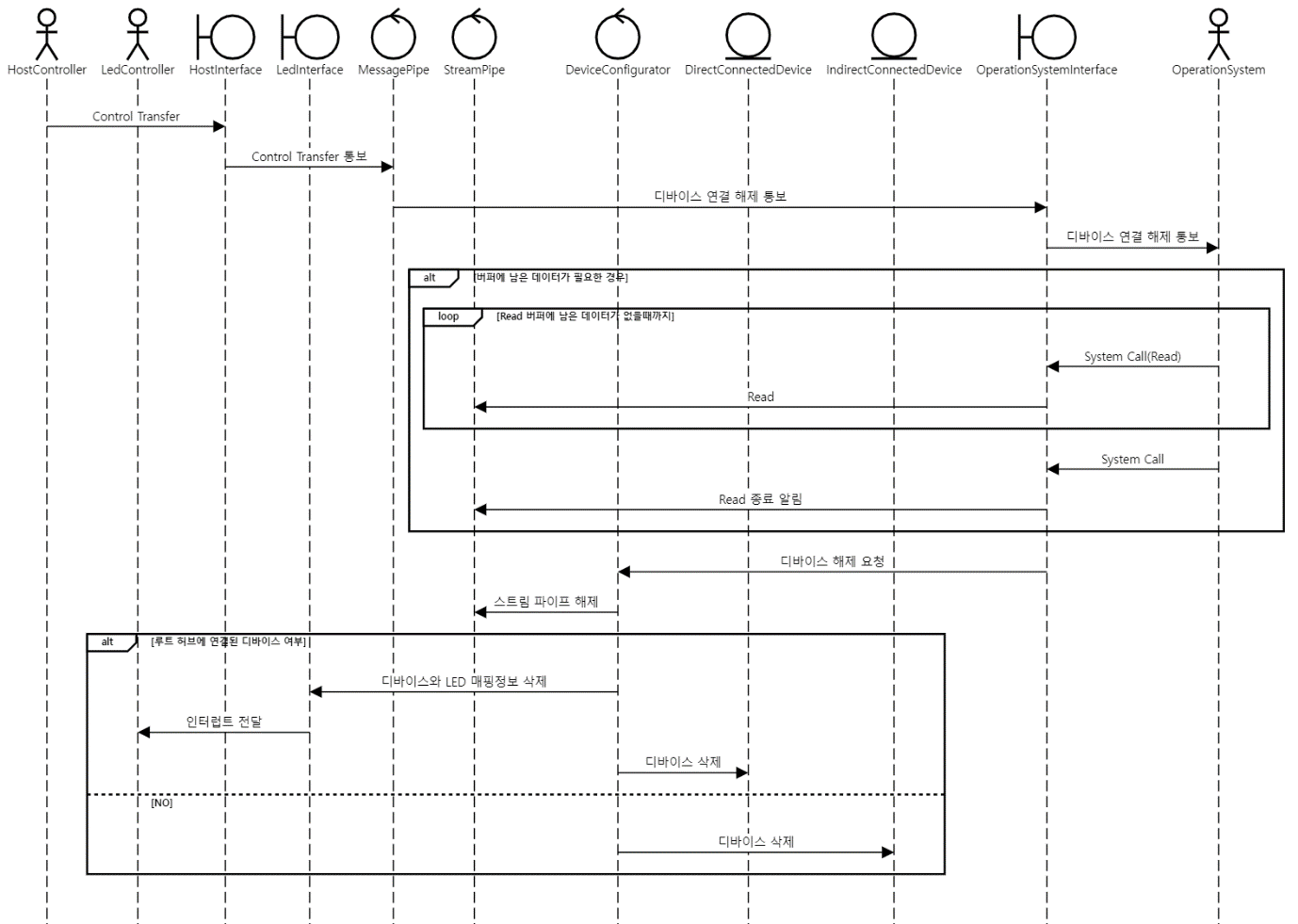


그림 - USB Device 연결 해제

**A2.6. 연결된 USB Device 정보 요청**

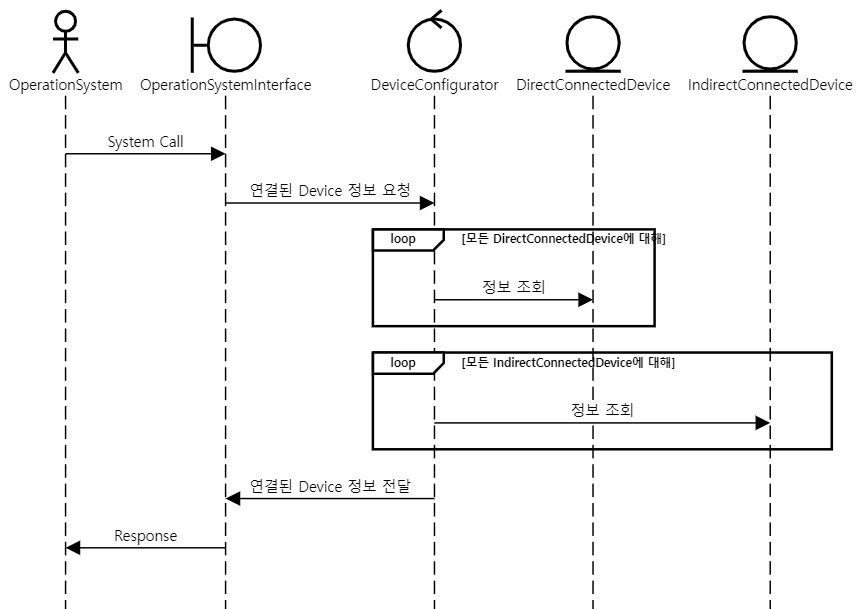


그림 - 연결된 USB Device 정보 요청

**A2.7. Host로 데이터 전송**

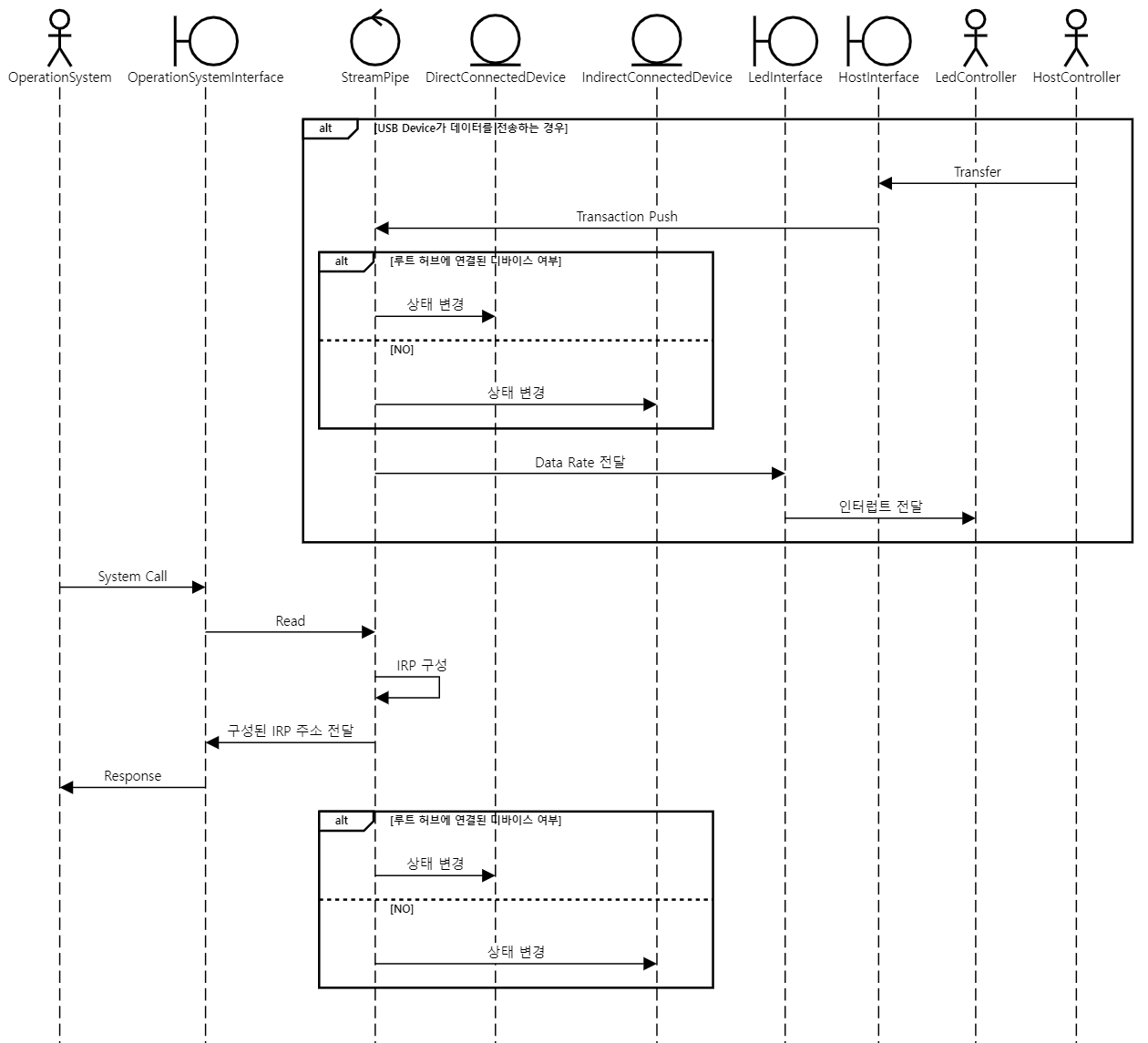
****

그림 - Host로 데이터 전송

**A2.7. USB Device로 데이터 전송**

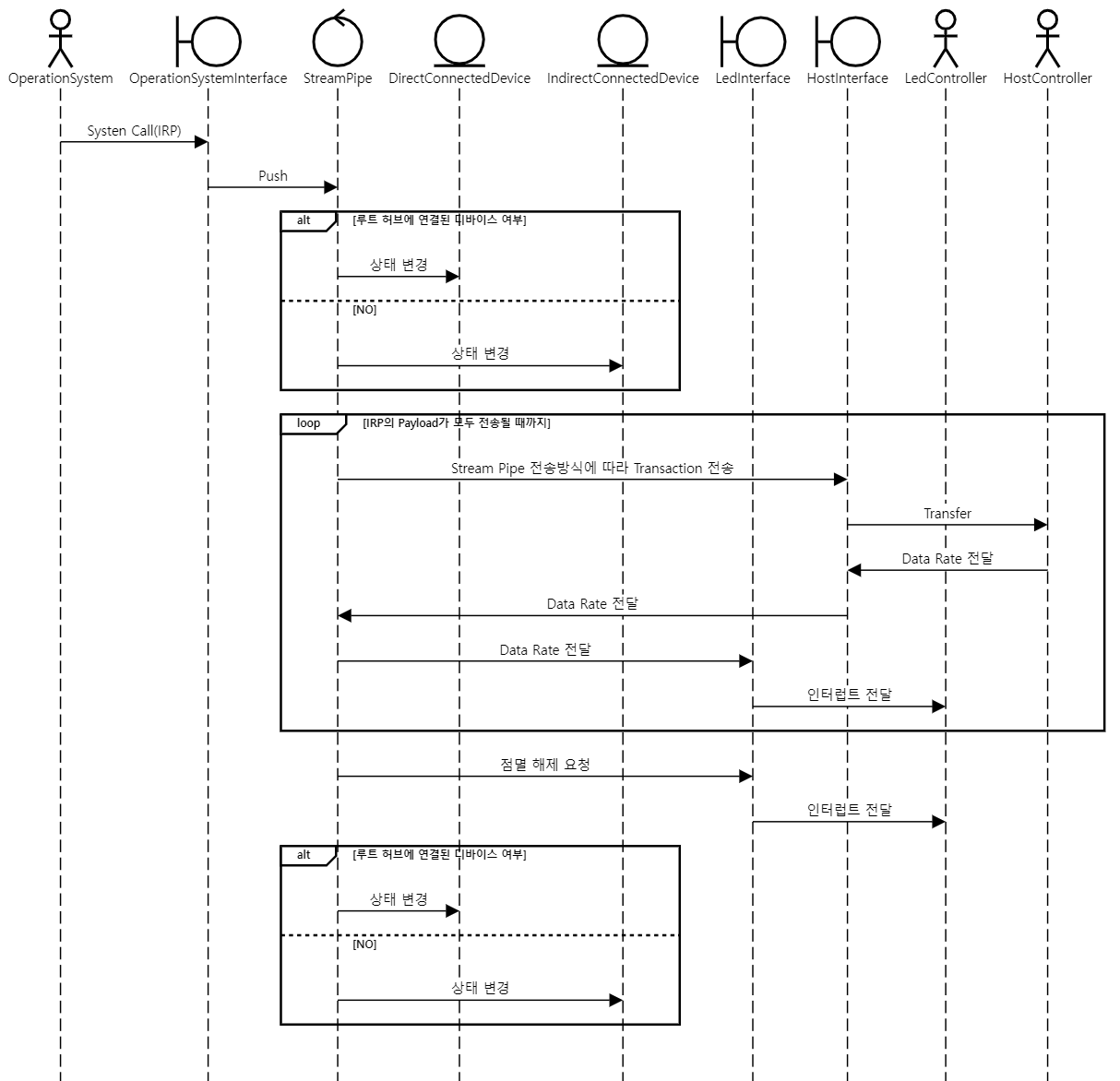
****

그림 - USB Device로 데이터 전송

1. 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. |

1. 품질 시나리오 분석

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 품질 시나리오 | | 중요도 | 복잡도 | 선정결과 |
| 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. 지원가능한 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 시나리오이므로 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 지원가능한 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 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.

1. 후보 구조

**D1. QA\_01 데이터 전송 속도**

Host와 USB Device 간의 데이터 전송 속도를 높이기 위해서 본 시스템은 세가지 주제에 대해 후보구조를 설계하였다.

1. 전송 단위에 대한 메모리 할당
2. 전송 실패시 재전송의 주체 설정
3. 전송의 병렬성

1번의 경우, 메모리의 할당 및 해제는 일반적으로 비용이 많이 발생하는 동작으로 빈번하게 일어나는 전송에 있어서 반드시 고려해야할 사항이라고 생각하였다. 2번의 경우, 전송의 실패는 어떠한 이유로든 발생할 수 있는 리스크이기에 전송의 무결성을 위하여 재전송은 반드시 필요한 동작인데 이 동작을 수행할 주체가 어디가 될 것인가에 대한 설계이다. 3번의 경우 여러 USB Device와 데이터 통신을 진행할 때의 병렬처리에 대한 고려사항이다.

이러한 특성을 고려하여 데이터 전송 속도를 높이기 위한 후보 구조를 알아본다.

**D1.1. 전송 단위에 대한 메모리 할당 후보구조**

**D1.1.1. CA\_01. IRP의 주소를 그대로 사용하여 추가적인 메모리 할당 없이 Transaction을 구성하는 방법**

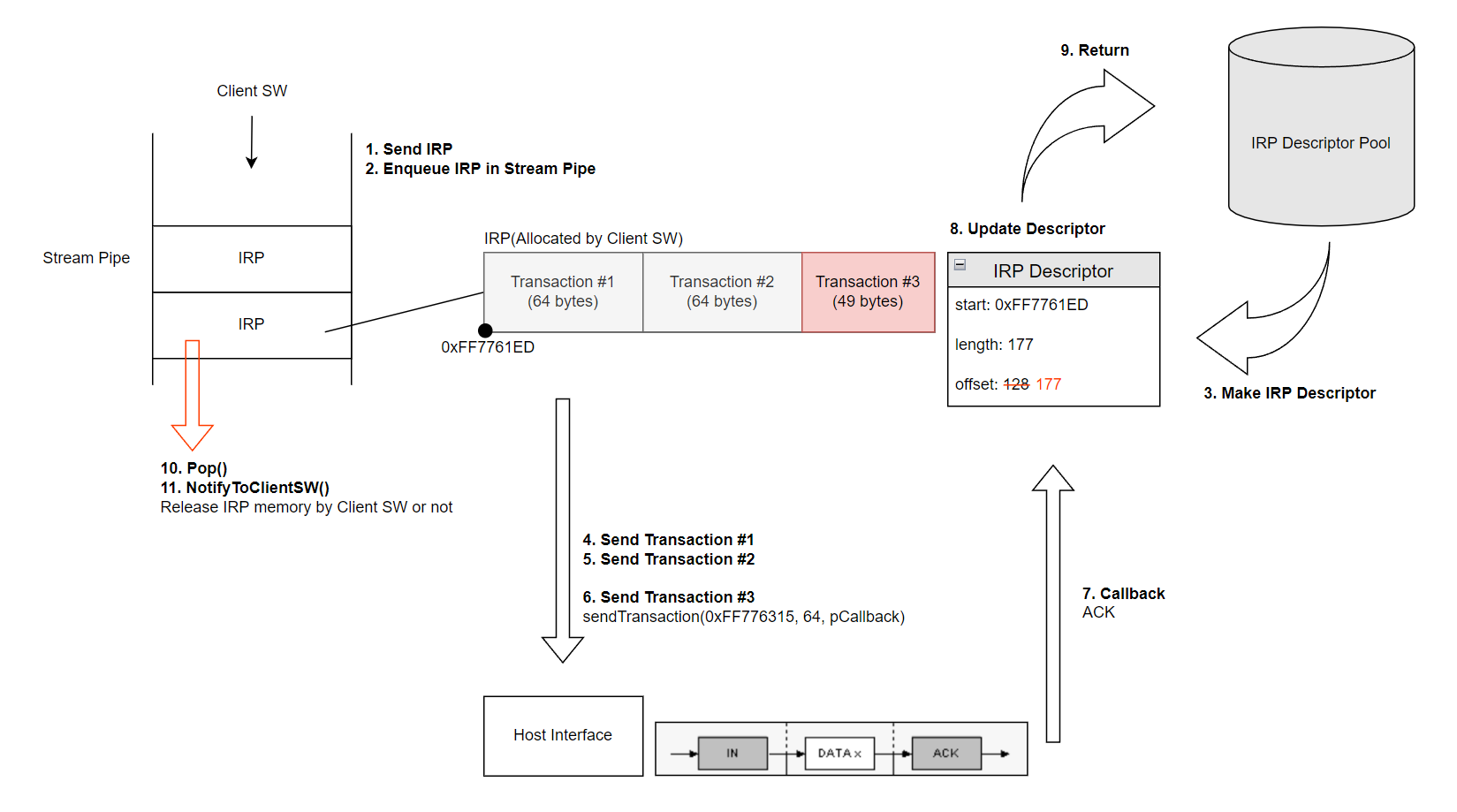


그림 – IRP의 주소를 그대로 활용하는 구조

데이터 전송시에 시스템은 Client SW로부터 받은 IRP를 segmentation해서 Transaction으로 보내야 한다. 이때 Transaction의 Payload를 위한 메모리 공간을 할당하지 않고 IRP의 주소 값을 그대로 이용하는 구조가 CA1이다.

이를 위해 시스템은 IRP를 Client SW로부터 수신하면 전송한 지점을 기록하기 위한 IRP Descriptor를 생성하고 Transaction이 이루어질 때마다 offset을 기록한다. 전송길이는 Stream Pipe에 정의된 통신방식에 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 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.

**D1.1.2. CA\_02. 메모리 Pool을 이용하여 새로운 Transaction을 구성하는 방법**

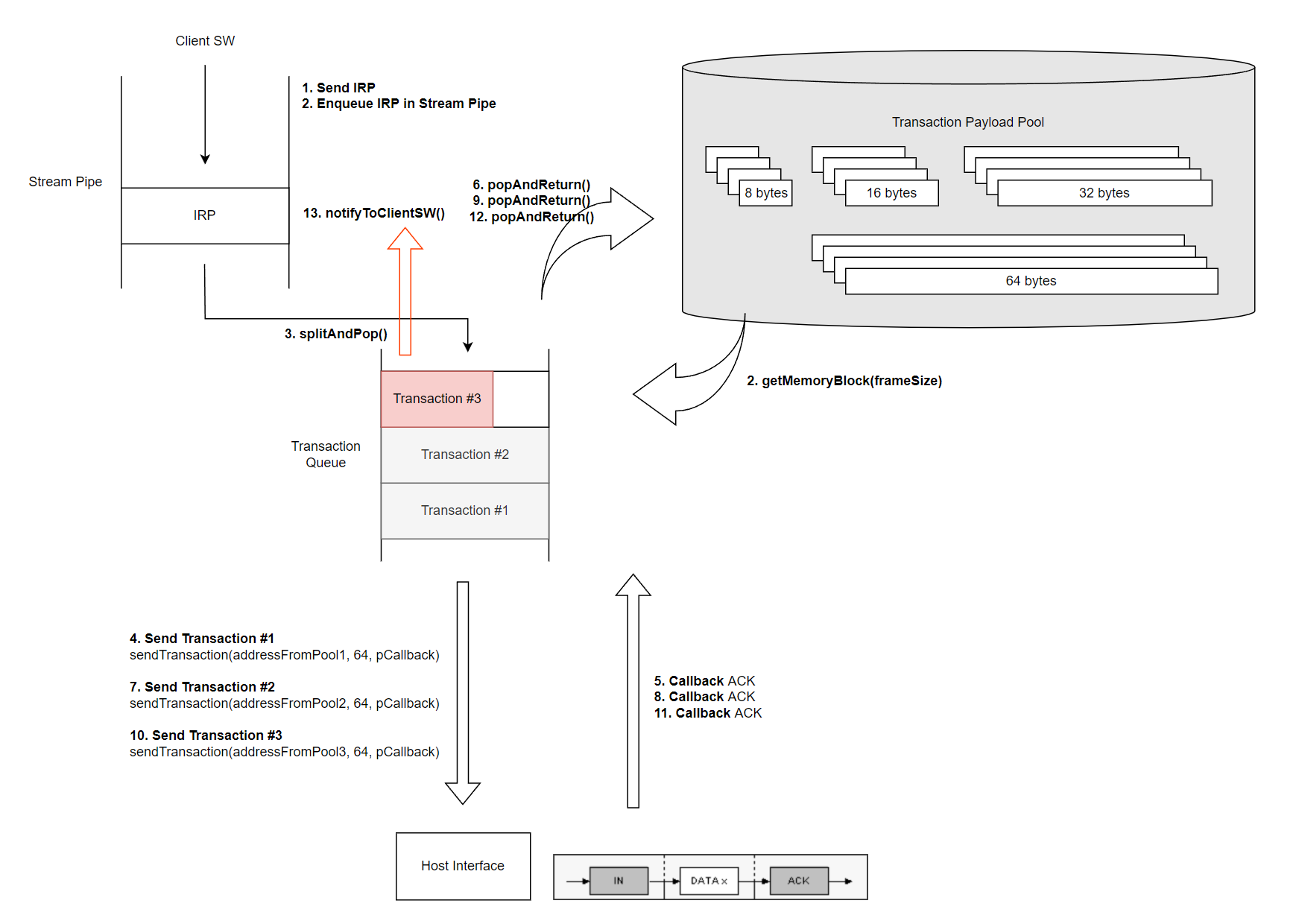


그림 - 메모리 Pool을 이용하는 방법

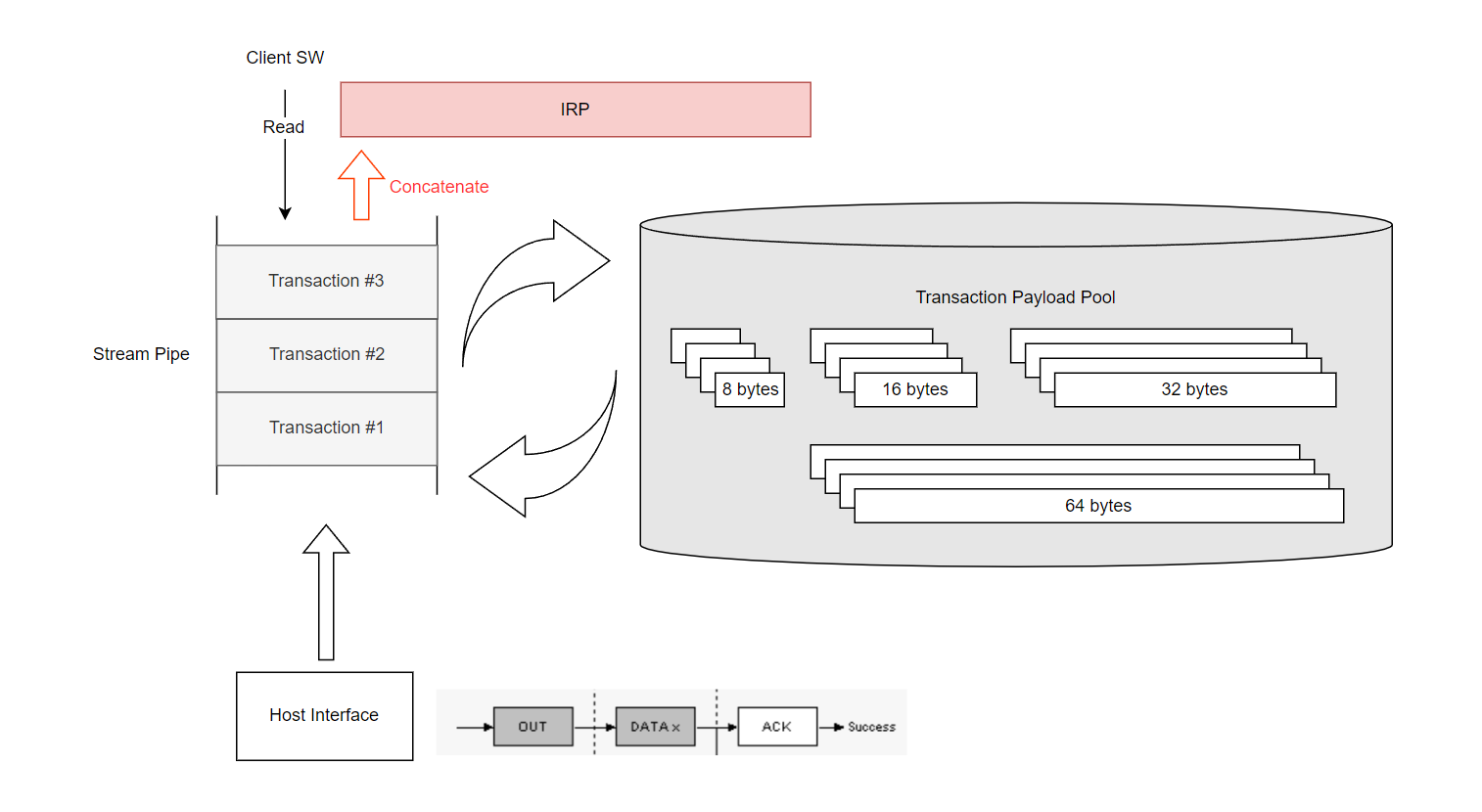
Client SW로부터 IRP를 수신하면 시스템은 Stream Pipe가 가지고 있는 전송 타입에 따라 적절한 Frame 크기로 데이터를 Segmentation해서 별도의 Transaction Queue에 넣는 구조가 CA2이다.

CA1에서 언급했듯 메모리의 할당 및 해제에는 큰 성능 비용이 발생하므로 Frame 크기에 따라 여러 memory block을 미리 할당해둔 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.

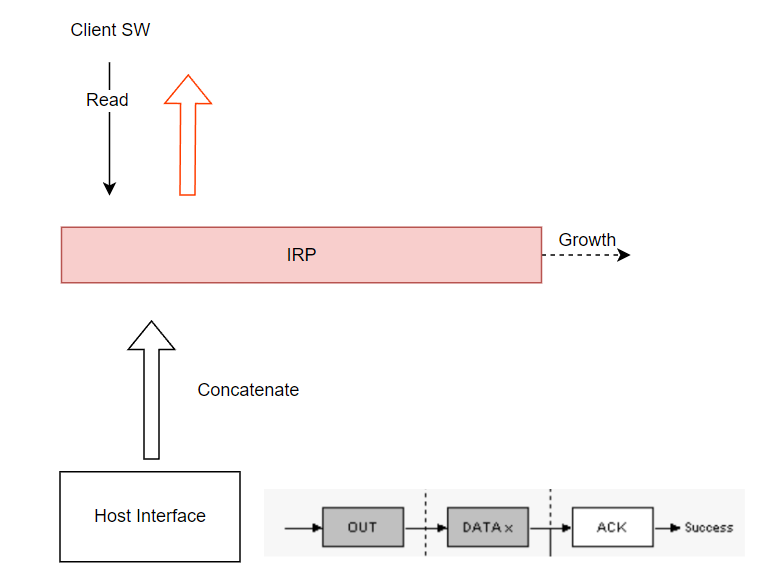
이 구조는 Payload를 복사해야 한다는 단점이 존재한다. 이에 따른 성능 저하를 보완하기 위해 DMA(Direct Memory Access) 장치를 이용한 복사를 활용할 수도 있다.

**D1.1.3. CA\_03. USB Device가 보내는 데이터를 메모리 Pool을 이용하여 IRP를 계속 구성하다가 Host에서 요청이 오면 각각 전송하는 방법**



* 리드는 내가 어쨌든 할당된 메모리를 가지고 있어야 한다. 프레임 사이즈에 맞추어 메모리 할당 풀을 가지고 있자
* 즉 프레임 사이즈의 풀에다가 들어오는데로 쌓아놓고 여러 프레임 메모리를 조합해서 패킷만들어 올려보내기
* 프로토콜떄 하던거 생각하면 분산된 메모리로 굳이 들고다니다가 최종적으로 한번 복사

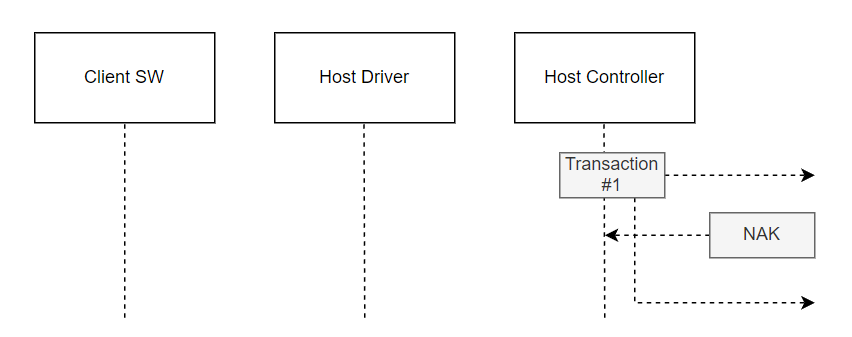
**D1.1.4. CA\_04. USB Device가 보내는 데이터를 계속 Concat하면서 이어나가다가 Host에서 요청시 하나로 보내는 방법**



* DMA로?

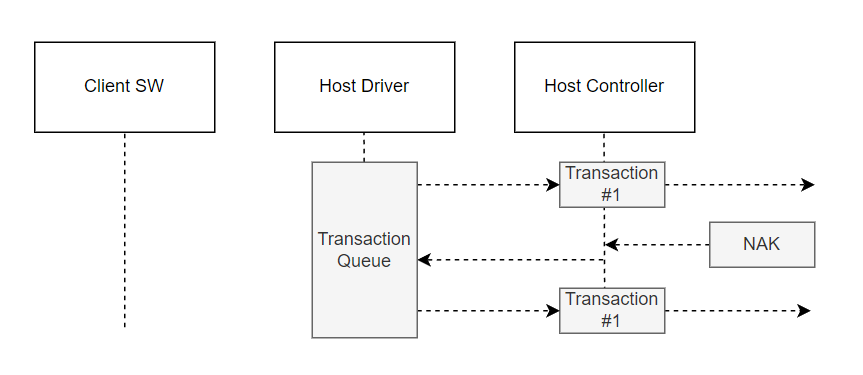
**D1.2. 전송 실패시 재전송의 주체 설정**

**D1.2.1. CA\_05. Host Interface에서 재전송 담당하여 Host Driver가 무결한 데이터를 다루는 방법**

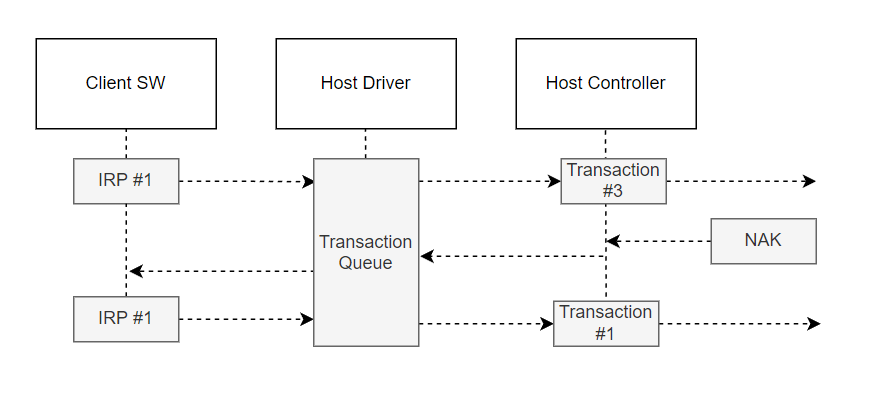


* NAK 패킷을 받으면 알아서 재전송하므로 위에서는 신경안씀

**D1.2.2. CA\_06. Host Driver가 NAK 수신시 재전송을 담당하는 방법**



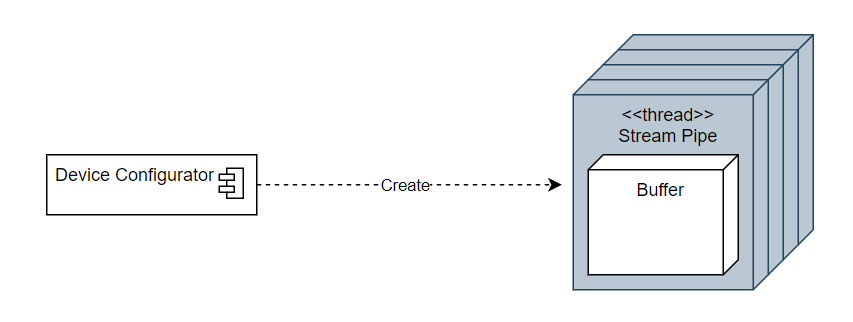
**D1.2.3. CA\_07. Client SW가 NAK을 수신하여 Host Driver에게 재전송을 지시하는 방법**



Client SW가 Transaction을 나누는게 아니니 IRP를 다시 보내야 한다. #3 #1

**D1.3. 전송의 병렬성**

**D1.3.1. CA\_08. Stream Pipe가 생성되면 별도의 쓰레드를 할당하는 방식**



**D1.3.2. CA\_09. 비동기로 처리하는 방식**



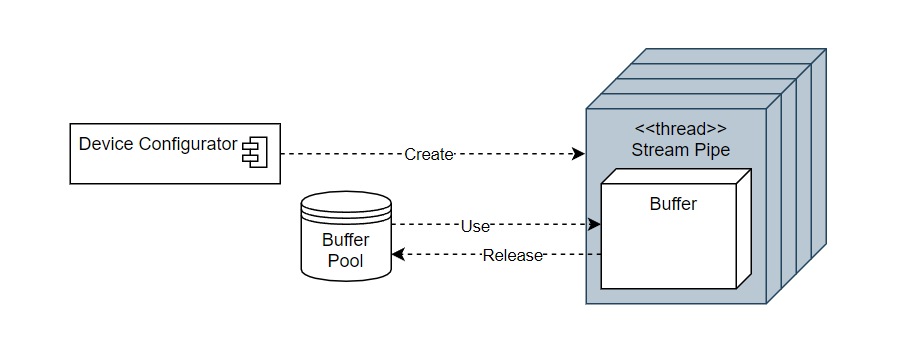
* 노드처럼

**D2. NFR\_01 디바이스 설정시간**

어쩌고저쩌고

**D2.1. 파이프의 버퍼 구조**

**D2.1.1. CA\_10. Stream Pipe를 구성하기 위한 버퍼의 Pool을 통해 즉시 개설이 가능하도록 하는 방법**



성능++ 변경용이성 -

**D2.1.2. CA\_11. Stream Pipe들을 위한 거대한 메모리 공간을 두고 필요한 리소스만큼 경계를 부여하는 방법**

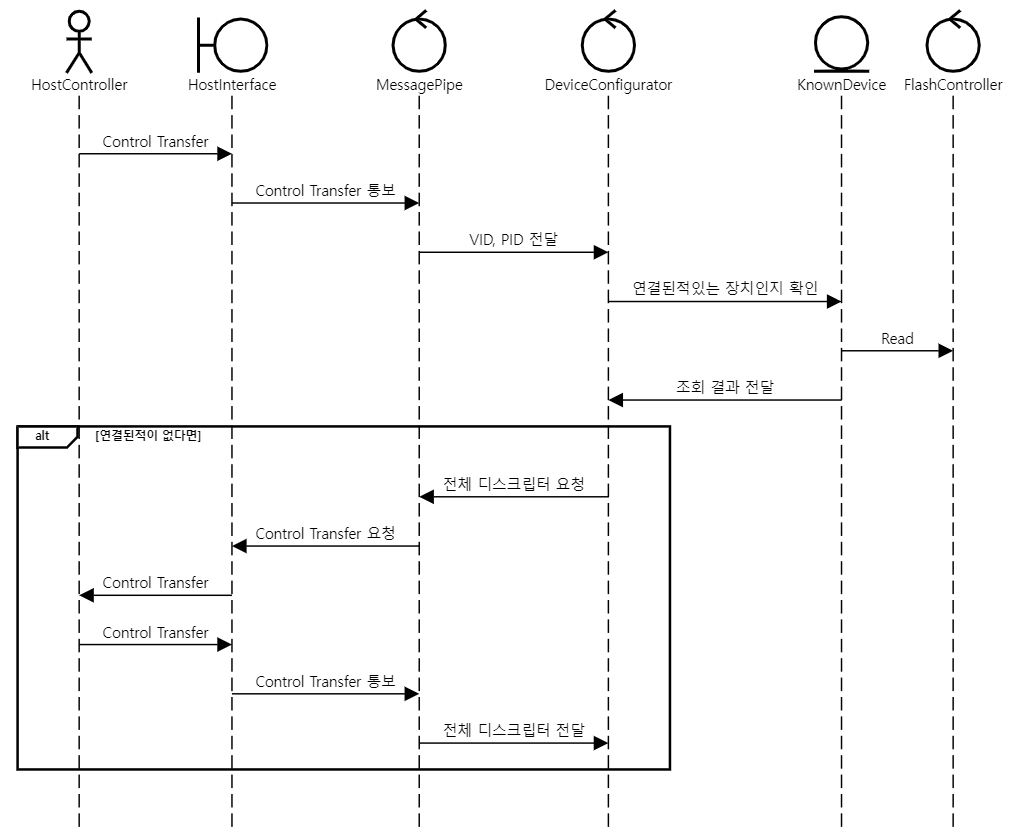
테이블이(가) 표시된 사진

자동 생성된 설명

성능 +, 변경용이성 + 안정성 -

**D2.2. 디바이스 식별 구조**

**D2.2.1. CA\_12. Known Device에 대해서 Fast Track으로 연결하는 구조**

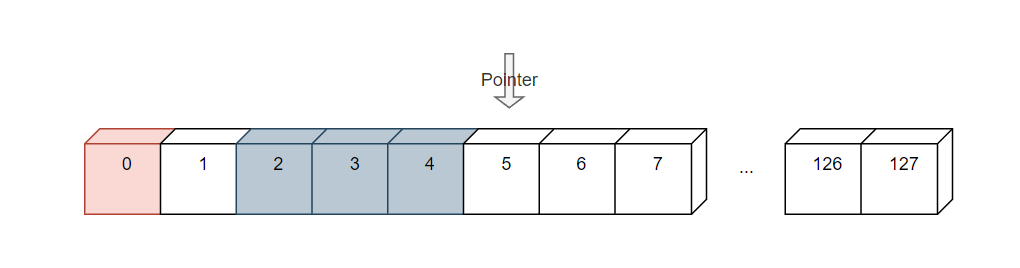
****

**D2.2.2. CA\_13. 모든 접속에 대해서 디스크립터를 확인하는 구조**

시큐리티+

**D2.3. USB Device 주소 할당 방식**

**D2.3.1. CA\_14. 주소 할당을 라운드 로빈으로 하는 방식(포인터와 127 라운드)**



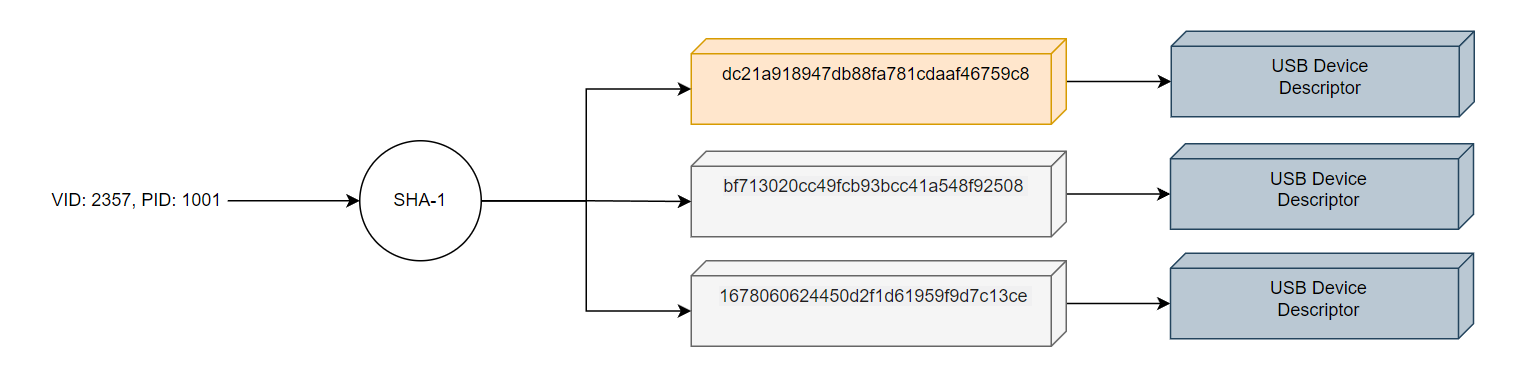
**D2.3.2. CA\_15. 1부터 빈곳을 찾아서 할당하는 구조**

가구, 테이블, 서랍장이(가) 표시된 사진

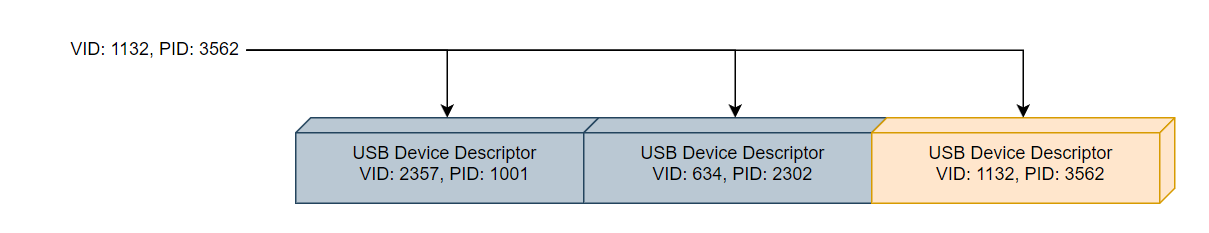
자동 생성된 설명

**D2.4. Known Device 보관 구조**

**D2.4.1. CA\_16. 해시**



**D2.4.2. CA\_17. 리스트**

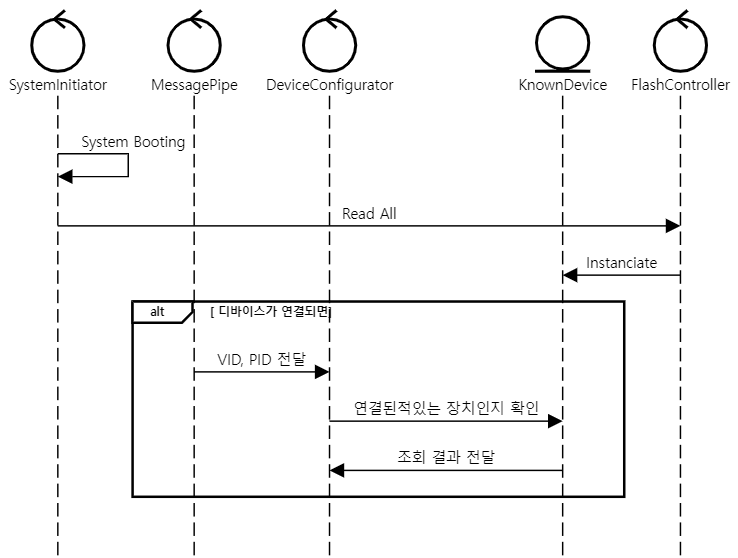


**D3. NFR\_02 시스템 부팅 시간**

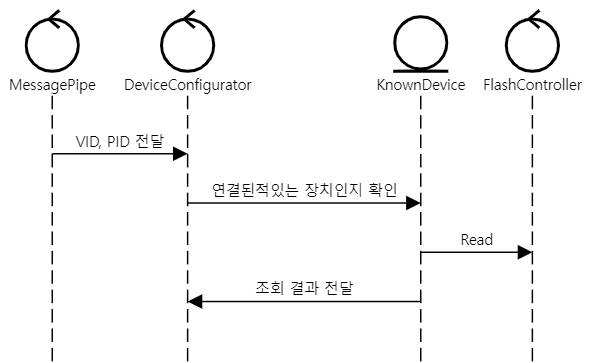
어쩌고저쩌고

**D3.1. Flash에서 데이터를 읽는 타이밍**

**D3.1.1. CA\_18. Flash에서 미리 테이블을 읽고 사용하기**

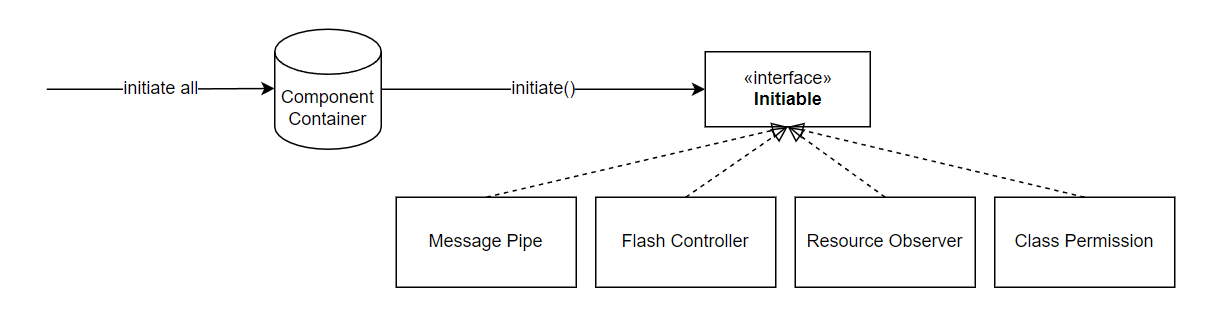
****

**D3.1.2. CA\_19. Flash에서 필요할 때 접근**

****

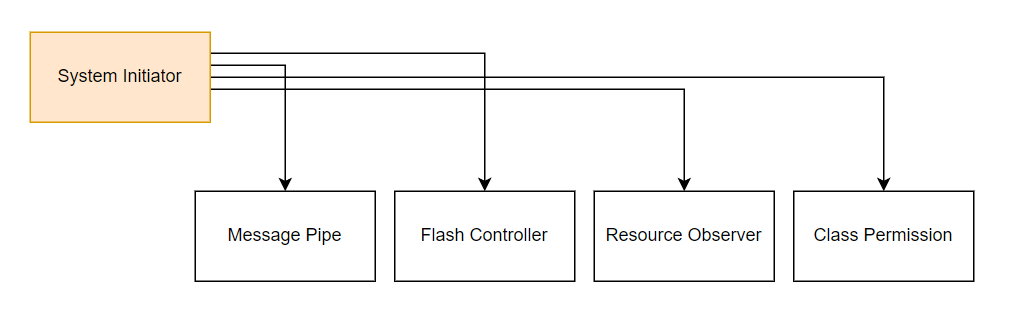
**D3.2. 부팅 시나리오를 관장하는 모듈의 유무**

**D3.2.1. CA\_20. 부팅시 모든 모듈에게 초기화 명령 전달**



* 모든 모듈에 동일한 인터페이스 제공

**D3.2.2. CA\_21. 시스템 초기화 및 종료를 위한 모듈**



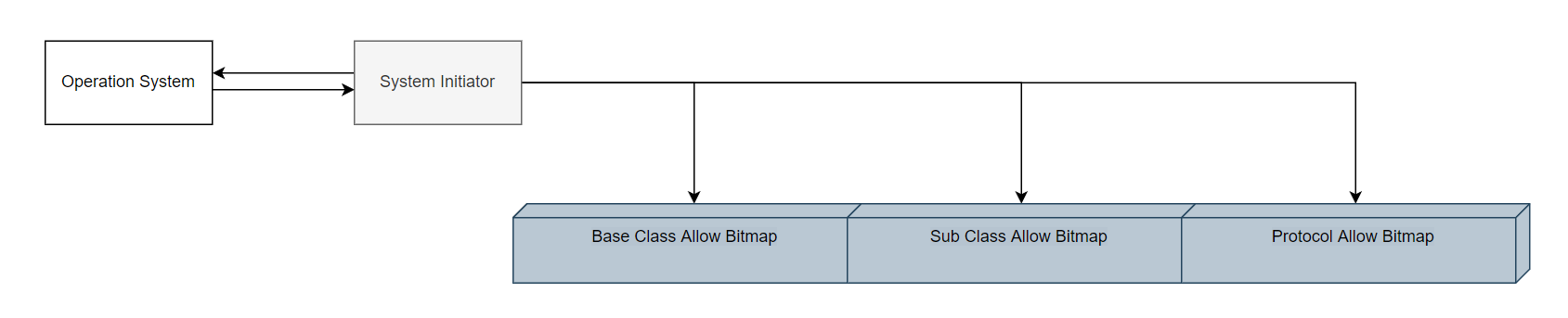
* 응집성, 시나리오 순서 등 관장이 쉽다

**D4. QA\_02 지원 가능한 USB Class 확장**

어쩌고저쩌고

**D4.1. USB 허용 클래스 취득 방법**

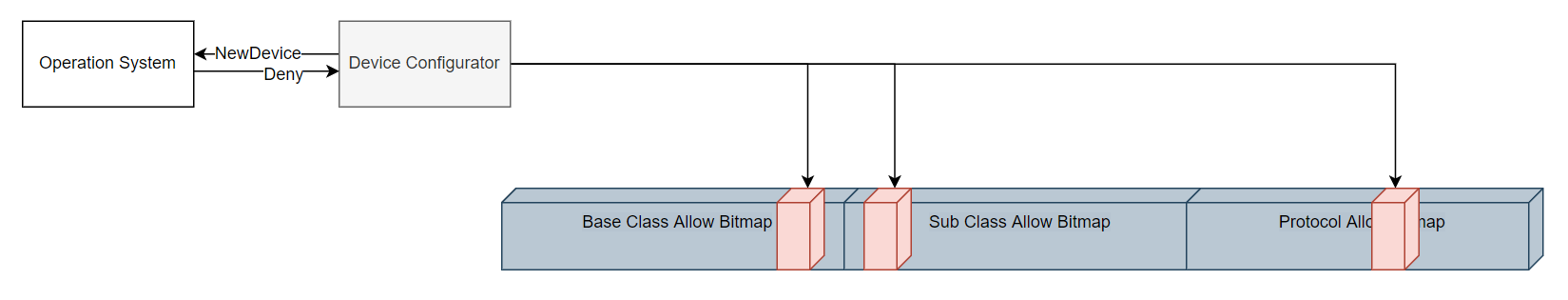
**D4.1.1. CA\_22. 허용 가능한 클래스를 os로부터 받아와서 비트맵으로 저장하는 방식**



운영체제와의 별도의 API가 필요하다.

어찌되었건 런타임에 얻어오므로 변경용이성이 좋다는걸 글로 잘 적자.

**D4.1.2. CA\_23. 모두 허용해놓고 거부되는게 있을때마다 하나씩 줄여나가는 방식**



보안성-

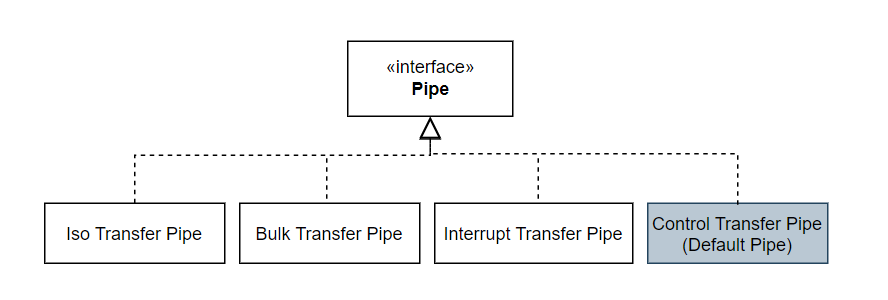
**D5. QA\_03 전송 방식 변경**

어쩌고저쩌고

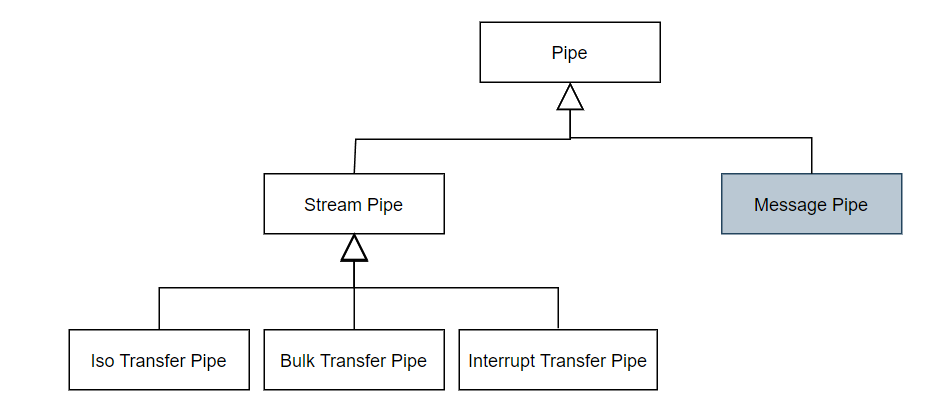
**D5.1. 제어 신호의 전송 구조**

**D5.1.1. CA\_24. Default Pipe도 제어정보를 전송하는 파이프의 상속구조로 두는 구조**

동일한 전송한다() 라는 책임을 갖는걸로 보기



**D5.1.2. CA\_25. Default Pipe를 별도의 모듈로 두는 구조(Increase Cohesion)**

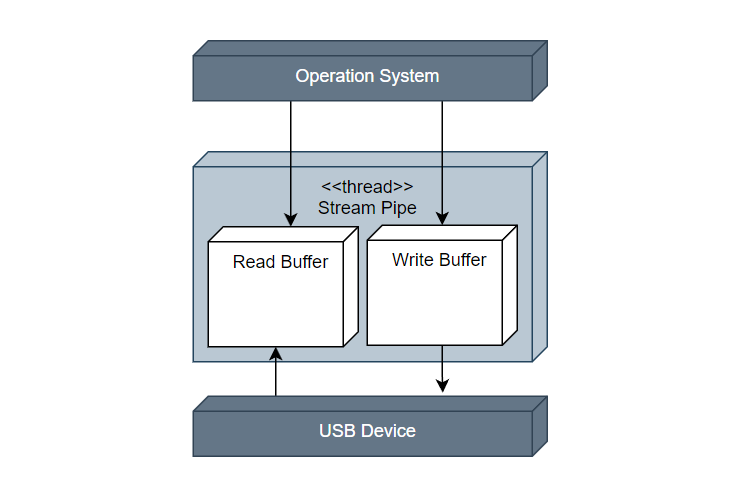


**D6. QA\_04 Host Controller의 USB 지원 버전 확장**

어쩌고저쩌고

**D6.1. 파이프의 버퍼 구조**

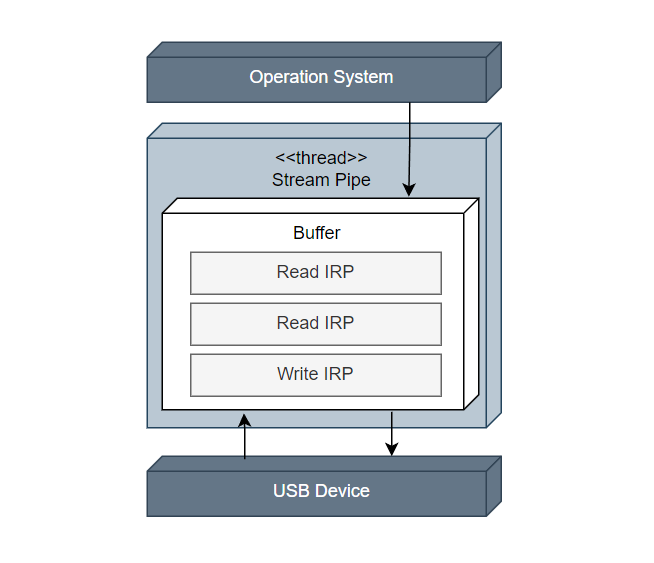
**D6.1.1. CA\_26. Full-Duflex 지원을 위해 Stream Pipe에서 읽기와 쓰기 버퍼를 분리하는 방식**



공간활용성? 리소스가용? - => 읽기하는동안 쓰기를 안쓰면

버퍼 풀을 갖자. 크기가 다양한가? 리소스 환경에 따라 달라질지도

**D6.1.2. CA\_27. 하나의 버퍼에서 읽기와 쓰기를 동시에 진행하며 IRP 마다 읽기인지 쓰기인지 구분하는 방식**

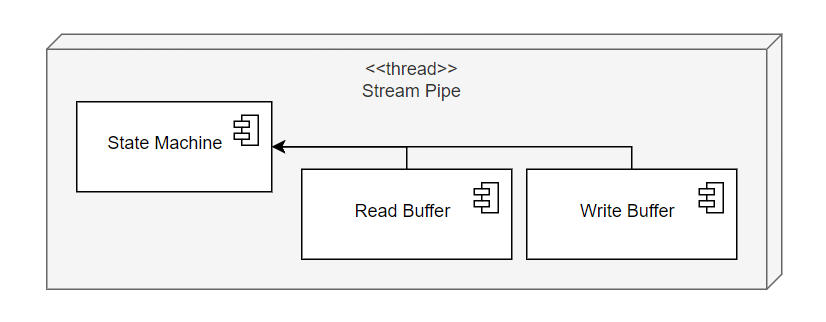


변경용이성-

하프 듀플렉스는 안될거같은데? 그래서 미채택인가

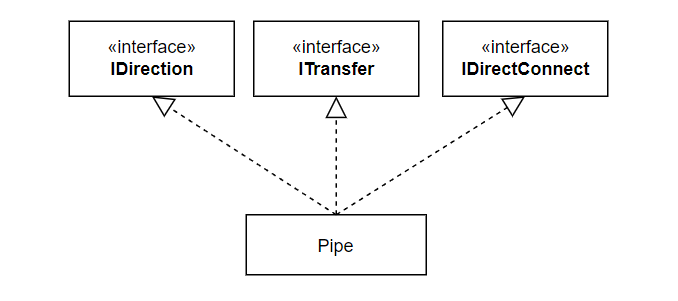
**D6.2. Stream Pipe 추상화**

**D6.2.1. CA\_28. Stream Pipe의 State Pattern 적용(DeviceAttached, Powered, Bus Reset)**



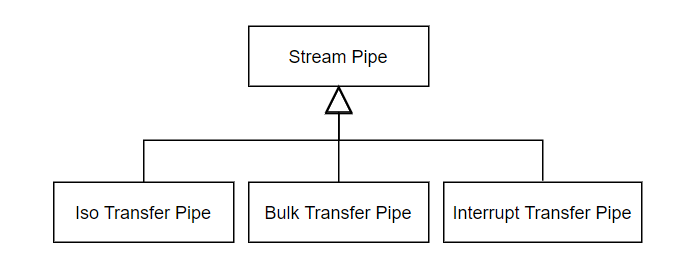
스테이트 추가용이

**D6.2.2. CA\_29. ISP을 준수하는 Pipe의 다양한 인터페이스 제공**



IDirection, IHalfDuflex => 요건 방향이네 설명하자, Itransfer, IDirectConnect

**D6.2.3. CA\_30. 전송 방법에 따라 다형성으로 표현하는 방법**



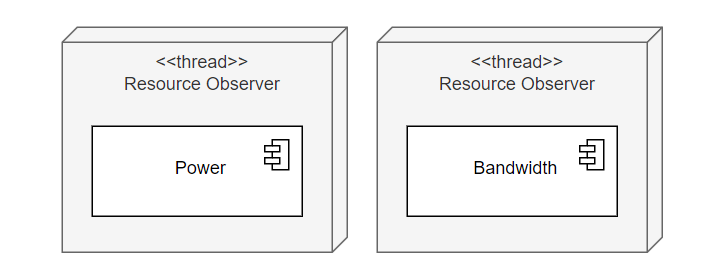
스펙에 정의된 여러 전송방식이 있지만 전송한다는 동일한 책임

**D7. QA\_05 동시에 연결 가능한 USB Device의 수**

어쩌고저쩌고

**D7.1. USB Device 연결에 대한 리소스 할당**

**D7.1.1. CA\_31. ResourceObserver를 둬서 리소스 현황을 주기적으로 파악하고 최적의 리소스 할당하는 방식**



가용성+

**D7.1.2. CA\_32. 최대 가용자원만 파악한 뒤 요청이 올때마다 Best Effort로 할당하고 더 이상 연결을 허용하지 않는 방식**

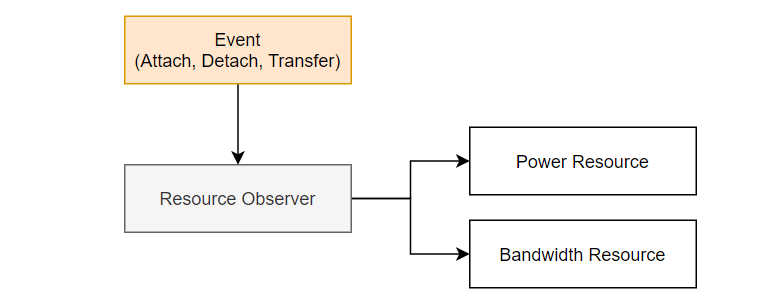
성능+

**D7.2. 리소스 확인 주기에 대한 구조**

**D7.2.1. CA\_33. Watchdog 형식으로 주기적으로 가용자원을 확인하는 방식(별도의 쓰레드)**

위에 그림 번호 나오면 말로 때우면 될거같다

**D7.2.2. CA\_34. 특정 이벤트(디바이스 연결 해제 등) 발생시마다 가용자원을 확인하는 방식**



성능+, 가용성-

**D8. QA\_06 외부 모듈 교환**

어쩌고저쩌고

**D8.1. 외부와의 인터페이스 추상화 구조**

**D8.1.1. CA\_35. Host와의 유니폼한 인터페이스를 추상화하는 방식**

* 오퍼레이션이 다양하지 않아서 시스템콜을 추상화하면 여러 운영체제에 대해 변경용이성

변경용이성류는 도메인모델에서 변경되는거 색칠하는걸로 하자

**D8.1.2. CA\_36. 주요 운영체제에 대한 여러가지 인터페이스 제공**

변경용이성-, 스페시픽한 오퍼레이션을 구현할 수 있다.

**D8.1.3. CA\_37. LED와의 인터페이스를 위한 프록시**

**D8.1.4. CA\_38. HostController와의 인터페이스를 위한 프록시**

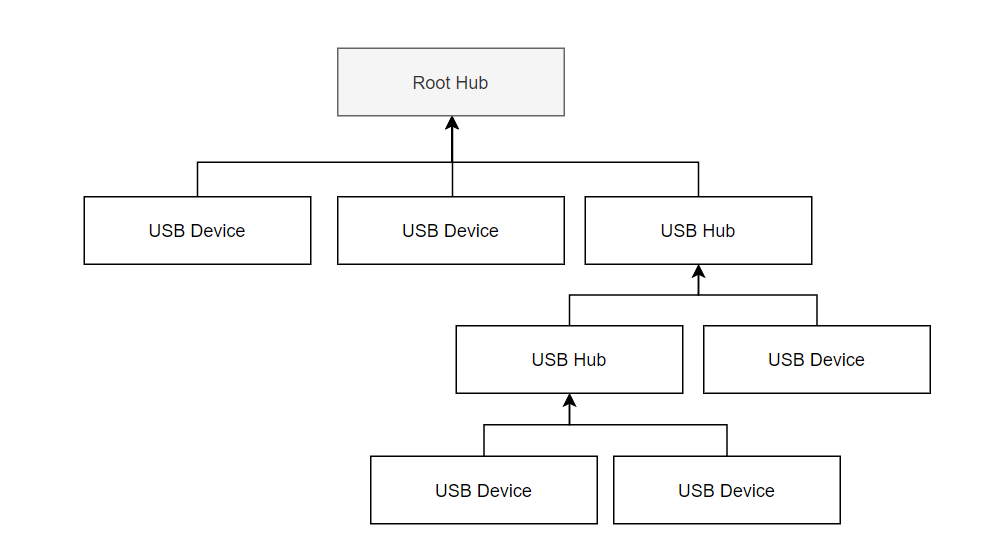
* 셋업보내고 데이터 받고 액보내고 없으면 넥보내고 이러한 동작은 스펙에 정의되어 있으므로 추상화된 인터페이스를 가진다. Send(), receive(), isData()이런걸로 추상화 => 변경용이성

**D9. QA\_07 디바이스 정보 조회 시간**

어쩌고저쩌고

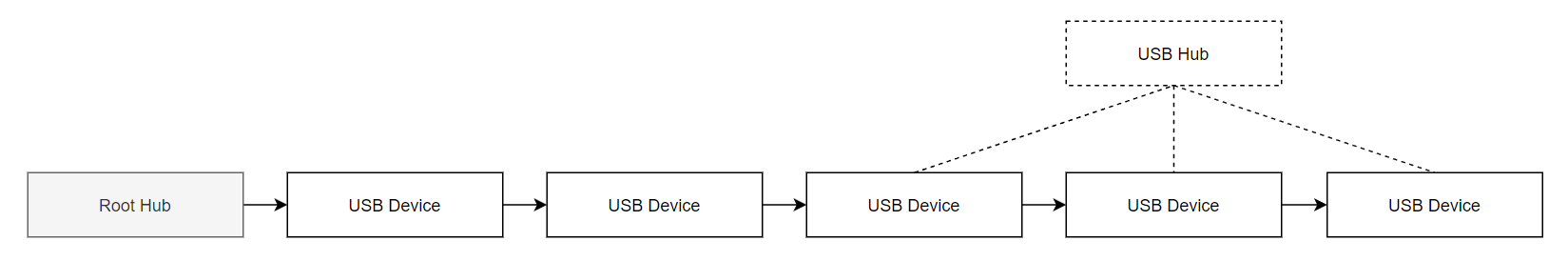
**D9.1. 디바이스 정보 저장 구조**

**D9.1.1. CA\_39. Device 연결구조를 트리로 표현하는 구조**



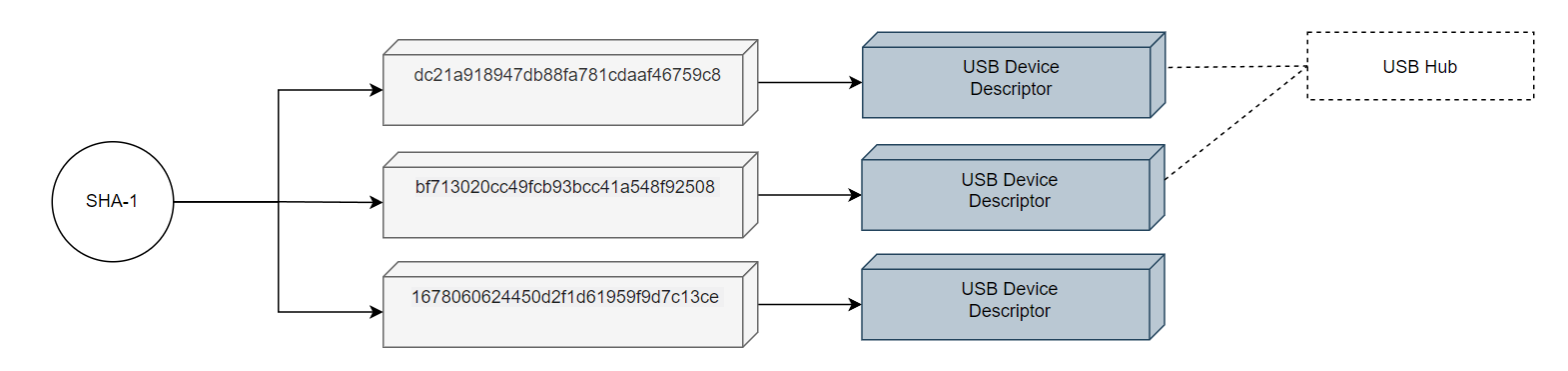
직관적이다.

**D9.1.2. CA\_40. 리스트로 표현하는 구조**



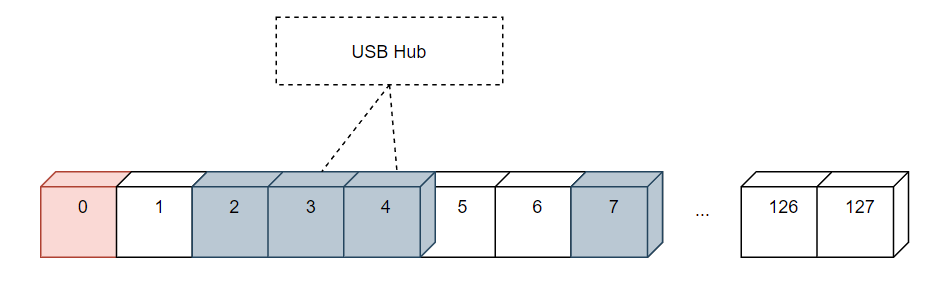
논리적으로 동급이다.

**D9.1.3. CA\_41. 고유 아이디를 가지고 해시 구성**



허브 해제등을 위해 순회가 필요한데 어렵다. 배열보다 특정 디바이스로 조회가 빠르다.

**D9.1.4. CA\_42. 구조체 배열로 구성하는 방법**



USB Hub 연결정보는 있어서 허브가 끊어지면 연결은 종료된다. 결국 어떤 형태로든 트리일수밖에 없긴 하다. 4개가 마찬가지다. 장단점을 따지자

127개로 정해져있음

1. 후보 구조 평가

**E1. QA\_01 데이터 전송 속도**

**CA\_01. IRP의 주소를 그대로 사용하여 추가적인 메모리 할당 없이 Transaction을 구성하는 방법**. **[채택]**

(퍼포먼스 ++, 가용성 +)

**CA\_02. 메모리 Pool을 이용하여 새로운 Transaction을 구성하는 방법 [미채택]**

(퍼포먼스-, 가용성- 메모리 공간 미리 많이 잡으므로, DMA 사용시 특정 하드웨어로 인해 변경용이성 -)

**CA\_03. USB Device가 보내는 데이터를 메모리 Pool을 이용하여 IRP를 계속 구성하다가 Host에서 요청이 오면 각각 전송하는 방법 [채택]**

* 리드는 내가 어쨌든 할당된 메모리를 가지고 있어야 한다. 프레임 사이즈에 맞추어 메모리 할당 풀을 가지고 있자
* 즉 프레임 사이즈의 풀에다가 들어오는데로 쌓아놓고 여러 프레임 메모리를 조합해서 패킷만들어 올려보내기
* 프로토콜떄 하던거 생각하면 분산된 메모리로 굳이 들고다니다가 최종적으로 한번 복사

**CA\_04. USB Device가 보내는 데이터를 계속 Concat하면서 이어나가다가 Host에서 요청시 하나로 보내는 방법 [미채택]**

* DMA로?

**CA\_05. Host Interface에서 재전송 담당하여 Host Driver가 무결한 데이터를 다루는 방법 [채택]**

* NAK 패킷을 받으면 알아서 재전송하므로 위에서는 신경안씀

**CA\_06. Host Driver가 NAK 수신시 재전송을 담당하는 방법 [미채택]**

**CA\_07. Client SW가 NAK을 수신하여 Host Driver에게 재전송을 지시하는 방법 [미채택]**

**CA\_08. Stream Pipe가 생성되면 별도의 쓰레드를 할당하는 방식 [채택]**

**CA\_09. 비동기로 처리하는 방식 [미채택]**

* 노드처럼

**E2. NFR\_01 디바이스 설정시간**

**CA\_10. Stream Pipe를 구성하기 위한 버퍼의 Pool을 통해 즉시 개설이 가능하도록 하는 방법 [채택]**

성능++ 변경용이성 -

**CA\_11. Stream Pipe들을 위한 거대한 메모리 공간을 두고 필요한 리소스만큼 경계를 부여하는 방법 [미채택]**

성능 +, 변경용이성 + 안정성 -

**CA\_12. Known Device에 대해서 Fast Track으로 연결하는 구조 [채택]**

**CA\_13. 모든 접속에 대해서 디스크립터를 확인하는 구조 [미채택]**

시큐리티+끼링끼링까랑까랑빵상아쥼마

**CA\_14. 주소 할당을 라운드 로빈으로 하는 방식(포인터와 127 라운드) [채택]**

**CA\_15. 1부터 빈곳을 찾아서 할당하는 구조 [미채택]**

**CA\_16. 해시 [채택]**

**CA\_17. 리스트 [미채택]**

**E3. NFR\_02 시스템 부팅 시간**

**CA\_18. Flash에서 미리 테이블을 읽고 사용하기 [채택]**

**CA\_19. Flash에서 필요할 때 접근 [미채택]**

**CA\_20. 부팅시 모든 모듈에게 초기화 명령 전달 [미채택]**

* 모든 모듈에 동일한 인터페이스 제공

**CA\_21. 시스템 초기화 및 종료를 위한 모듈 [채택]**

* 응집성?

**E4. QA\_02 지원 가능한 USB Class 확장**

**CA\_22. 허용 가능한 클래스를 os로부터 받아와서 비트맵으로 저장하는 방식 [채택]**

운영체제와의 별도의 API가 필요하다.

**CA\_23. 모두 허용해놓고 거부되는게 있을때마다 하나씩 줄여나가는 방식 [미채택]**

보안성-

**E5. QA\_03 전송 방식 변경**

**CA\_24. Default Pipe도 제어정보를 전송하는 파이프의 상속구조로 두는 구조 [미채택]**

**CA\_25. Default Pipe를 별도의 모듈로 두는 구조(Increase Cohesion) [채택]**

**E6. QA\_04 Host Controller의 USB 지원 버전 확장**

**CA\_26. Full-Duflex 지원을 위해 Stream Pipe에서 읽기와 쓰기 버퍼를 분리하는 방식 [채택]**

공간활용성? 리소스가용? - => 읽기하는동안 쓰기를 안쓰면

버퍼 풀을 갖자. 크기가 다양한가? 리소스 환경에 따라 달라질지도

**CA\_27. 하나의 버퍼에서 읽기와 쓰기를 동시에 진행하며 IRP 마다 읽기인지 쓰기인지 구분하는 방식 [미채택]**

변경용이성-

– 경계를 넘지 않도록 안전장치가 있어야 한다.ㄱ 가드

**CA\_28. Stream Pipe의 State Pattern 적용(DeviceAttached, Powered, Bus Reset) [채택]**

스테이트 추가용이

**CA\_29. ISP을 준수하는 Pipe의 다양한 인터페이스 제공 [채택]**

IDirection, IHalfDuflex, Itransfer, IDirectConnect

**CA\_30. 전송 방법에 따라 다형성으로 표현하는 방법 [채택]**

스펙에 정의된 여러 전송방식이 있지만 전송한다는 동일한 책임

**E7. QA\_05 동시에 연결 가능한 USB Device의 수**

**CA\_31. ResourceObserver를 둬서 리소스 현황을 주기적으로 파악하고 최적의 리소스 할당하는 방식 [채택]**

가용성+

**CA\_32. 최대 가용자원만 파악한 뒤 요청이 올때마다 Best Effort로 할당하고 더 이상 연결을 허용하지 않는 방식 [미채택]**

성능+

**D7.2.1. CA\_33. Watchdog 형식으로 주기적으로 가용자원을 확인하는 방식(별도의 쓰레드) [채택]**

**D7.2.2. CA\_34. 특정 이벤트(디바이스 연결 해제 등) 발생시마다 가용자원을 확인하는 방식 [미채택]**

성능+, 가용성-

**E8. QA\_06 외부 모듈 교환**

**CA\_35. Host와의 유니폼한 인터페이스를 추상화하는 방식 [채택]**

* 오퍼레이션이 다양하지 않아서 시스템콜을 추상화하면 여러 운영체제에 대해 변경용이성

**CA\_36. 주요 운영체제에 대한 여러가지 인터페이스 제공 [미채택]**

변경용이성-, 스페시픽한 오퍼레이션을 구현할 수 있다.

**CA\_37. LED와의 인터페이스를 위한 프록시 [채택]**

**CA\_38. HostController와의 인터페이스를 위한 프록시 [채택]**

* 셋업보내고 데이터 받고 액보내고 없으면 넥보내고 이러한 동작은 스펙에 정의되어 있으므로 추상화된 인터페이스를 가진다. Send(), receive(), isData()이런걸로 추상화 => 변경용이성

**E9. QA\_07 디바이스 정보 조회 시간**

**CA\_39. Device 연결구조를 트리로 표현하는 구조 [미채택]**

직관적이다.

**D9.1.2. CA\_40. 리스트로 표현하는 구조 [미채택]**

논리적으로 동급이다.

**D9.1.3. CA\_41. 고유 아이디를 가지고 해시 구성 [채택]**

허브 해제등을 위해 순회가 필요한데 어렵다. 배열보다 특정 디바이스로 조회가 빠르다.

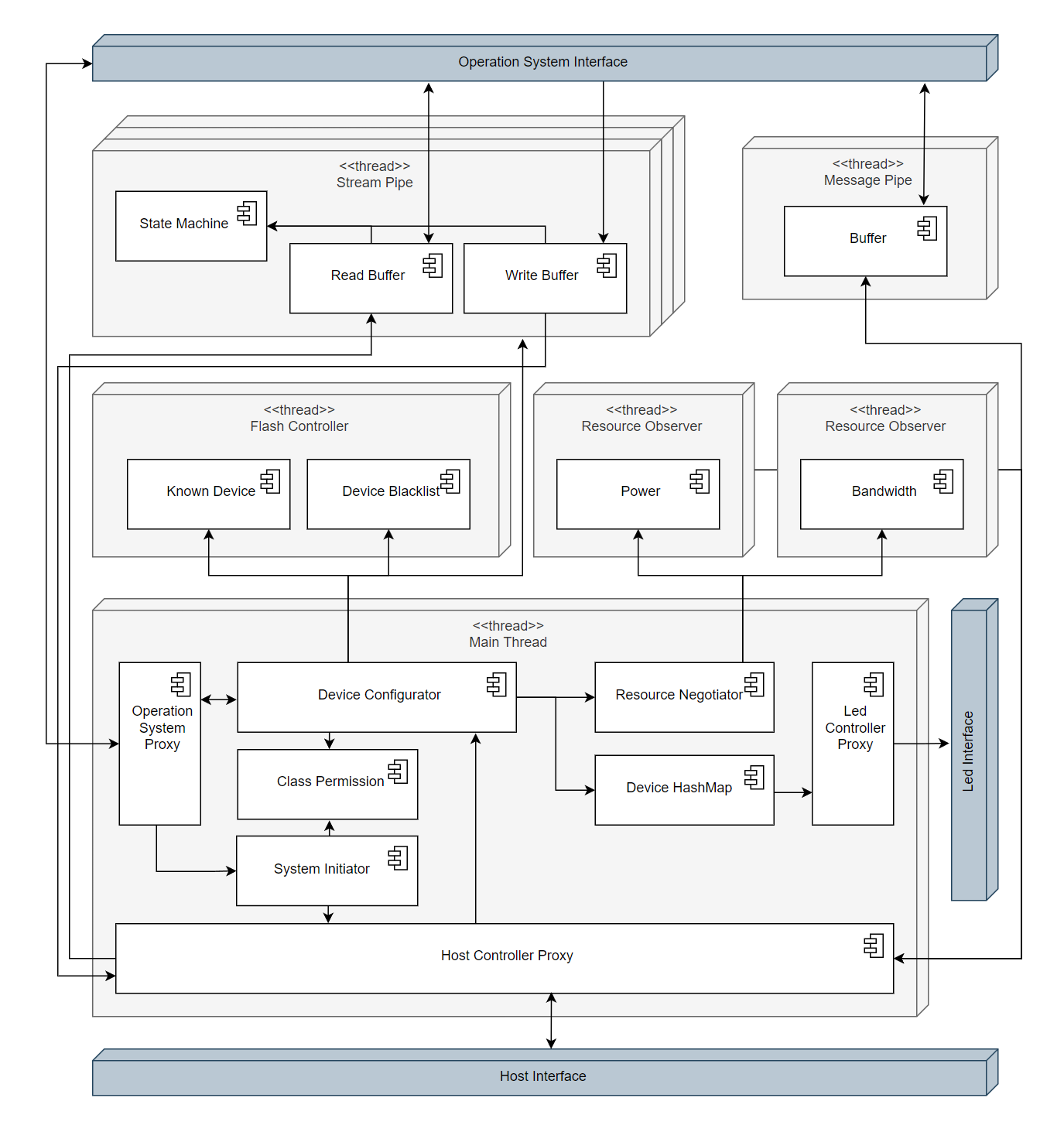
**D9.1.4. CA\_42. 구조체 배열로 구성하는 방법 [미채택]**

1. 최종 구조 설계

// 활동7. 최종 구조 설계

// 점검7-3. 최종 구조로의 통합 과정에 대한 설명이 적절한가?

// 점검7-4. 최종 구조의 단점/RISK 관리가 적절한가? (개선)



약점 보완등에 DMA 사용이런것도 괜찮을거같고 쓰레드가 여러 개니 락걸어야하는데 락프리 자료구조를 이용한다던지 이런저런거 쓰자

**CA\_01. IRP의 주소를 그대로 사용하여 추가적인 메모리 할당 없이 Transaction을 구성하는 방법**. **[채택]**

(퍼포먼스 ++, 가용성 +)

**CA\_03. USB Device가 보내는 데이터를 메모리 Pool을 이용하여 IRP를 계속 구성하다가 Host에서 요청이 오면 각각 전송하는 방법 [채택]**

* 리드는 내가 어쨌든 할당된 메모리를 가지고 있어야 한다. 프레임 사이즈에 맞추어 메모리 할당 풀을 가지고 있자
* 즉 프레임 사이즈의 풀에다가 들어오는데로 쌓아놓고 여러 프레임 메모리를 조합해서 패킷만들어 올려보내기
* 프로토콜떄 하던거 생각하면 분산된 메모리로 굳이 들고다니다가 최종적으로 한번 복사

**CA\_05. Host Interface에서 재전송 담당하여 Host Driver가 무결한 데이터를 다루는 방법 [채택]**

* NAK 패킷을 받으면 알아서 재전송하므로 위에서는 신경안씀

**CA\_08. Stream Pipe가 생성되면 별도의 쓰레드를 할당하는 방식 [채택]**

**CA\_10. Stream Pipe를 구성하기 위한 버퍼의 Pool을 통해 즉시 개설이 가능하도록 하는 방법 [채택]**

성능++ 변경용이성 -

**CA\_12. Known Device에 대해서 Fast Track으로 연결하는 구조 [채택]**

**CA\_14. 주소 할당을 라운드 로빈으로 하는 방식(포인터와 127 라운드) [채택]**

**CA\_16. 해시 [채택]**

**CA\_18. Flash에서 미리 테이블을 읽고 사용하기 [채택]**

**CA\_21. 시스템 초기화 및 종료를 위한 모듈 [채택]**

* 응집성?

**CA\_22. 허용 가능한 클래스를 os로부터 받아와서 비트맵으로 저장하는 방식 [채택]**

운영체제와의 별도의 API가 필요하다.

**CA\_25. Default Pipe를 별도의 모듈로 두는 구조(Increase Cohesion) [채택]**

**CA\_26. Full-Duflex 지원을 위해 Stream Pipe에서 읽기와 쓰기 버퍼를 분리하는 방식 [채택]**

공간활용성? 리소스가용? - => 읽기하는동안 쓰기를 안쓰면

버퍼 풀을 갖자. 크기가 다양한가? 리소스 환경에 따라 달라질지도

**CA\_28. Stream Pipe의 State Pattern 적용(DeviceAttached, Powered, Bus Reset) [채택]**

스테이트 추가용이

**CA\_29. ISP을 준수하는 Pipe의 다양한 인터페이스 제공 [채택]**

IDirection, IHalfDuflex, Itransfer, IDirectConnect

**CA\_30. 전송 방법에 따라 다형성으로 표현하는 방법 [채택]**

스펙에 정의된 여러 전송방식이 있지만 전송한다는 동일한 책임

**CA\_31. ResourceObserver를 둬서 리소스 현황을 주기적으로 파악하고 최적의 리소스 할당하는 방식 [채택]**

가용성+

**D7.2.1. CA\_33. Watchdog 형식으로 주기적으로 가용자원을 확인하는 방식(별도의 쓰레드) [채택]**

**CA\_35. Host와의 유니폼한 인터페이스를 추상화하는 방식 [채택]**

* 오퍼레이션이 다양하지 않아서 시스템콜을 추상화하면 여러 운영체제에 대해 변경용이성

**CA\_37. LED와의 인터페이스를 위한 프록시 [채택]**

**CA\_38. HostController와의 인터페이스를 위한 프록시 [채택]**

* 셋업보내고 데이터 받고 액보내고 없으면 넥보내고 이러한 동작은 스펙에 정의되어 있으므로 추상화된 인터페이스를 가진다. Send(), receive(), isData()이런걸로 추상화 => 변경용이성

**D9.1.3. CA\_41. 고유 아이디를 가지고 해시 구성 [채택]**

허브 해제등을 위해 순회가 필요한데 어렵다. 배열보다 특정 디바이스로 조회가 빠르다.