ChatGPT

EnergyPlus 기반 실시간 예측 제어 시뮬레이션 시스템 구축 연구

1. EnergyPlus 모델의 FMU(Co-simulation) 생성 방법 및 제약 조건

EnergyPlus 모델을 **FMU (Functional Mock-up Unit)**로 내보내면 다른 프로그램과 공동 시뮬레이션을 수행할 수 있습니다 [1]. 이를 위해 **EnergyPlusToFMU**라는 LBNL에서 개발한 오픈소스 툴을 사용합니다. 이 툴은 EnergyPlus v8.0 이상 버전의 건물 에너지 모델(IDF 파일)을 **FMI 표준**(1.0 또는 2.0)의 Co-simulation FMU로 변환해 줍니다 [2]. FMU 생성 시 EnergyPlus 모델과 필요한 리소스(예: weather 파일 등)가 패키징되며, FMU 내부에 EnergyPlus 실행 파일과 IDF/EPW 등이 포함됩니다 [3].

FMU로 내보내기 전에 **IDF 모델에 외부 인터페이스 객체들을 설정**해야 합니다. EnergyPlusToFMU는 IDF 내 특별한 객체들을 통해 **FMU 입출력 변수**를 정의하도록 되어 있습니다 [4]. 예를 들어 IDF에 다음과 같은 객체들을 추가합니다:

- `ExternalInterface, FunctionalMockupUnitExport;` – 외부 인터페이스(FMU 내보내기)를 활성화 [5]
- `ExternalInterface:FunctionalMockupUnitExport:To:Schedule` / `...:To:Actuator` / `...:To:Variabl` – **FMU의 입력**을 EnergyPlus의 Schedule 값, EMS Actuator 또는 EMS 변수에 매핑 [4]
- `ExternalInterface:FunctionalMockupUnitExport:From:Variable` – **FMU의 출력**을 EnergyPlus 출력변수에 매핑 [4]

즉, **EnergyPlus 모델에서 제어하거나 읽어오고 싶은 지점**(예: 일정(Schedule), 사람 수, 설비 제어신호 등)을 위 객체로 지정하여 FMU 입출력으로 노출합니다. 아래는 일정 값을 외부에서 입력받고, 실내 온도를 출력하는 예시입니다 [6] [7]:

```
ExternalInterface, FunctionalMockupUnitExport;

ExternalInterface:FunctionalMockupUnitExport:To:Schedule,
    MySchedule, Any Number, ExtInput1, 0;  ! 외부 입력 1 -> EnergyPlus Schedule
'MySchedule'

ExternalInterface:FunctionalMockupUnitExport:From:Variable,
    ZONE1, Zone Mean Air Temperature, RoomTemp;  ! EnergyPlus 출력 -> 외부 출력
'RoomTemp'

Output:Variable, ZONE1, Zone Mean Air Temperature, TimeStep;
```

위 설정에서는 FMU 입력 변수 `ExtInput1`이 EnergyPlus의 `MySchedule` 값을 실시간으로 대체하며, FMU 출력 `RoomTemp`는 특정 존의 평균온도 값을 나타냅니다. 이처럼 **FMU로 내보낼 때 IDF에 정의된 RunPeriod(시뮬레이션 기간)와 날씨 파일(EPW)**도 함께 포함해야 하며, 누락될 경우 FMU 실행 시 초기화 오류가 발생합니다 [3]. EnergyPlusToFMU 사용 시 명령행 인자로 IDF, EPW, 시작/종료 시간 등을 지정하여 FMU를 생성하게 됩니다 [8].

제약 조건: EnergyPlus FMU는 **Co-simulation 방식**으로 동작하며 EnergyPlus 엔진 자체가 시간 적분을 수행합니다. 따라서 모델 교환(Model Exchange) FMU와 달리 외부에서 미분방정식 계산을 하지 않고, **일정 시간 스텝으로**

EnergyPlus를 진행시키는 형태입니다. Co-sim FMU는 **do_step**과 같은 함수로 매 시간 스텝을 진행하며, EnergyPlus 내부의 최소 시간 간격(Zone/HVAC TimeStep 등)으로 시뮬레이션됩니다. 예를 들어 IDF에서 `Timesteps per Hour`를 6으로 했다면 10분 간격이 최소 스텝이며, FMU를 1분 간격으로 호출하려면 해당 설정을 더 세분화(예: 60으로 설정하여 1분 timestep) 해야 합니다. 또한 현재 공개된 EnergyPlusToFMU는 Windows/Linux에서 사용 가능하며, 생성된 FMU는 플랫폼 종속적인 바이너리를 포함합니다. 마지막으로, FMU 내보낸 EnergyPlus는 다수의 외부 입력/출력을 가질 수 있지만, **초당 수백 개 이상의 신호**를 실시간으로 주고받는 등 극단적인 경우에는 성능 및 동기화에 유의해야 합니다.

요약하면, EnergyPlus 모델을 FMU로 만들기 위해 **EnergyPlusToFMU 툴과 IDF 설정**이 필요하며, 외부 데이터 연동을 위해 입력/출력 지점을 미리 정의해야 합니다 [4]. 이러한 준비가 갖춰지면 건물 에너지 모델을 FMU로 내보내 다양한 시뮬레이션 플랫폼이나 제어 시스템과 연동할 수 있습니다.

2. 생성된 FMU에 실시간 외부 데이터를 입력하는 방법 (PyFMI, FMI++ 등)

FMU로 변환된 EnergyPlus 모델은 **외부 프로그램에서 호출 및 제어**할 수 있습니다. 일반적으로 **마스터 알고리즘**(master program)이 FMU를 불러와서 시뮬레이션을 진행하며, 이 마스터에서 **실시간 센서/환경 데이터를 입력**으로 설정합니다. Python 환경에서는 **PyFMI** (JModelica.org에서 제공)나 **FMPy** 등을 이용해 FMU를 다룰 수 있고, C++에서는 **FMI++ 라이브러리** 등을 활용할 수 있습니다 [9]. PyFMI를 예로 들면, FMU 파일을 불러온 뒤 `initialize()`로 초기화하고, `do_step()` 함수를 매 시간 스텝마다 호출하면서 그 전에 `set()` 함수를 통해 **실시간 데이터 값을 FMU 입력 변수에 주입**합니다 [10]. 아래는 Python(PyFMI)을 사용한 간단한 예시 코드입니다:

```
from pyfmi import load_fmu

# EnergyPlus로 생성된 FMU 불러오기
model = load_fmu("BuildingModel.fmu")
model.initialize()  # FMU 초기화 (시뮬레이션 시작 시간 설정)

time_step = 60  # 60초 (1분) 간격으로 시뮬레이션
current_time = 0
while current_time < 3600:  # 예: 1시간(3600초) 동안 반복
    # 1) 센서/외부로부터 실시간 데이터 읽기 (예: Python 코드로 API 호출 또는 장비 데이터 수집)
    outdoor_temp = sensor_api.get_outdoor_temperature()        # 실시간 외기 온도 (예시 함수)
    occ_count = vision_system.get_current_occupancy_count()    # 비전 시스템으로 감지된 현재 재실자 수

    # 2) FMU 입력 변수 업데이트 (사전에 IDF에서 ExternalInterface로 정의된 변수명)
    model.set("OutdoorAirTemp", outdoor_temp)      # 외기온도 값을 FMU에 전달
    model.set("Zone1_Occupancy", occ_count)        # 재실자 수 값을 FMU에 전달

    # 3) 지정된 스텝만큼 시뮬레이션 진행 (60초 진행)
    model.do_step(current_t=current_time, step_size=time_step, new_step=True)
    current_time += time_step

    # 4) FMU 출력 변수 읽기 (예: 존 온도, HVAC 에너지 소비 등)
    zone_temp = model.get("Zone1_Temperature")
    hvac_power = model.get("HVAC_Power")
    # (필요하면 여기서 제어 계산을 수행하거나 결과를 저장)
```

위의 **마스터 루프**에서는 1분마다 실시간 데이터를 가져와(FMU 입력 설정) 한 스텝 시뮬레이션하고, 결과를 읽어오는 과정을 반복합니다. 이러한 과정은 **FMI의 표준 함수** 호출로 수행되며, 본질적으로 `fmiSetReal`(입력설정) → `fmiDoStep`(진행) → `fmiGetReal`(출력읽기) 순서로 이루어집니다 [10]. PyFMI는 이를 고수준으로 제공하며, FMI++ 라이브러리도 C++에서 유사한 기능(`sendInput()`, `doStep()`, `getOutput()` 등)을 제공합니다 [9].

실시간 데이터를 FMU에 입력하기 위해서는 **외부 데이터 수집 모듈**이 필요합니다. 예를 들어, Python에서 센서의 REST API를 호출하거나, 장비의 BACnet/Modbus 데이터를 읽어오는 코드를 작성할 수 있습니다. 그런 다음 해당 값을 위 코드처럼 `model.set("변수명", 값)`으로 FMU에 전달합니다. **PyFMI** 라이브러리를 쓰면 FMI 2.0 Co-simulation FMU의 경우 `do_step()` 메서드로 한 스텝씩 진행할 수 있고, FMI 1.0이거나 Model Exchange의 경우 `simulate()`나 적절한 함수를 사용해야 합니다. (EnergyPlusToFMU로 생성한 FMU는 기본적으로 Co-simulation 타입입니다 [11].)

한편, C++ 환경에서는 **FMI++** 또는 FMIL(FMI Library)을 사용해 FMU를 다룰 수 있습니다. FMI++은 FMI 표준의 저수준 C API를 보다 쉽게 다룰 수 있게 한 C++ 래퍼로, FMU를 로드하고 시간 스텝을 진행시키는 기능을 제공합니다 [9]. 또한 FMI++에는 Python, Java 바인딩도 존재하여 다양한 환경에서 FMU를 연동할 수 있습니다.

요약하면, **PyFMI (Python)**나 **FMI++ (C++ 등)**을 통해 EnergyPlus FMU에 실시간 데이터를 주입할 수 있습니다. 마스터 프로그램이 주기적으로 센서값을 읽어와 FMU의 입력 변수에 설정한 뒤 `do_step`으로 시뮬레이션을 진행하고, 그 결과를 `get`으로 받아 활용하는 방식입니다. 이러한 구조를 통해 **실시간 데이터 연동 Co-simulation**을 구현할 수 있습니다 [10]. (과거에는 BCVTB 같은 툴로 EnergyPlus와 실시간 데이터 또는 제어시스템을 연결하기도 했습니다 [12]. 하지만 FMI 기반 방법(PyFMI 등)이 더 최신 표준으로 선호됩니다.)

3. 실시간 Vision 데이터(예: 카메라 기반 재실자 정보) FMU 연동 방법

카메라 기반의 비전 데이터를 활용하여 실내 인원수나 활동량 등의 정보를 얻고 이를 EnergyPlus FMU에 연동할 수 있습니다. 이를 구현하려면 크게 **(a) Vision 데이터로부터 유의미한 수치 정보를 추출**하고 **(b) 그 정보를 FMU의 입력으로 사용**하는 두 단계가 필요합니다.

먼저 (a) 카메라 영상에서 **재실자 수, 위치, 활동량, 조명 On/Off** 등을 추론하는 컴퓨터 비전 모듈이 필요합니다. 예를 들어, 딥러닝 기반 객체 탐지나 **영상 분석 알고리즘**을 통해 실시간으로 사람 수를 카운트하거나 사람의 움직임으로부터 활동량(발열량 추정)을 산정할 수 있습니다 [13]. 연구 사례에서는 CO_2, 조도, PIR 센서, 카메라를 종합해 숨은 마르코프 모델(HMM)로 실내 인원을 추정하거나 [14], 직접적으로 영상처리를 통해 97% 정확도로 **점유자 수와 활동 수준(정적/활동)**을 분류한 예시도 있습니다 [15] [16]. 이처럼 Vision 시스템이 **재실자 관련 정보를 실시간 산출**하면, 이제 이를 EnergyPlus 모델에 반영해야 합니다.

(b) EnergyPlus FMU에 **재실자 관련 입력**을 연결하는 방법은, IDF에 정의된 ExternalInterface 입력을 어떻게 구성했는지에 따라 다릅니다. 일반적인 방법은 **People 객체의 Occupancy Schedule**을 외부 입력으로 두는 것입니다 [17]. 예를 들어, IDF의 `People` 객체는 인원수나 밀도를 정의하고 일정(Schedule)을 통해 시간별 변화를 주도록 되어 있는데, 이 일정 데이터를 ExternalInterface:...:To:Schedule로 설정하면 외부에서 현재 재실자 수를 직접 넣어줄 수 있습니다 [17]. 실제로 UnmetHours 질의에서도 “People 객체의 Occupancy schedule을 Schedule:File 등으로 입력받아 사용할 수 있다”는 언급이 있습니다 [17]. FMU 생성 시 해당 일정 이름을 ExternalInterface의 To:Schedule로 매핑하고 FMU 변수명으로 노출했다면, 마스터 프로그램에서 vision 모듈이 제공한 **현재 재실자 수 (또는 점유율)** 값을 `model.set("OccupancyScheduleVar", count)` 형태로 입력해주면 됩니다.

비전 데이터로 조명을 판별하여 **조명 상태**를 EnergyPlus에 반영하는 것도 가능합니다. 예를 들어, 카메라 영상에서 공간의 조도나 램프 상태를 분석해 **불이 켜져있음**을 감지하면, EnergyPlus의 Lighting Schedule 입력값을 1(ON)으로 설정하고 꺼지면 0(OFF)으로 설정할 수 있습니다. 마찬가지로 IDF에서 해당 조명 스케줄을 ExternalInterface 입력으로 지정해두면, 외부 vision 시스템의 결과 (조명 on/off 신호)를 FMU로 전달해 시뮬레이션에 반영할 수 있습니다. 혹

은 조명으로 인한 **발열(조명 발열량)**을 OtherEquipment 등의 내부발열로 간주하여, 그 발열량을 외부 입력으로 설정하는 방법도 있습니다. 예를 들어 Maria Alonso 등의 코멘트에 따르면, **습기나 추가 발열을 People 외에 OtherEquipment 등을 통해 스케줄로 넣는 방법**이 논의되고 있습니다 [18]. 이를 응용하면, Vision으로 감지된 **재실자 활동 수준**(예: "Active"한 상태이면 대사열 발산 증가)을 People 객체의 **Activity Level(대사량) Schedule**에 반영하는 등 세밀한 연동도 가능합니다.

또 하나의 접근은 **EMS (Energy Management System)**을 이용하는 것입니다. EnergyPlus의 EMS를 사용하면 시뮬레이션 중에 내부 변수나 Actuator를 제어할 수 있는데, ExternalInterface:...:To:Variable로 **EMS 변수**에 카메라 데이터를 넣고, EMS 프로그램에서 해당 변수 값에 따라 창 개폐나 HVAC 설정을 바꾸는 것도 가능합니다 [19] [20]. 그러나 이 경우에도 기본 원리는 외부 Vision 데이터 → FMU 입력으로 수치 변환 입니다. (LBNL의 obFMU 사례에서는, EnergyPlus와 연동된 **Occupant Behavior FMU**가 시간별로 창문 여닫기, 온도 설정 등을 결정해 EnergyPlus에 영향을 주었는데, 이 obFMU도 4개의 센서 입력(온도, 조도, CO_2, 조명전력)을 받아 4개의 제어출력(HVAC on/off, 조명 on/off, 침투환기, occupancy 스케줄)을 EnergyPlus로 보내는 구조였습니다 [21] [22]. 이는 실제 센서 대신 **에이전트 기반 가상 센서**를 쓴 경우지만, 원리는 실세계 센서/비전 데이터를 쓰는 것과 동일합니다.)

정리하면, **카메라 비전으로부터 얻은 재실 정보**를 활용하려면 먼저 영상을 분석해 **정량화된 데이터(예: 현재 인원=5명, 활동지수=활동 중, 조명=ON)**로 변환하고, 이를 **EnergyPlus FMU의 입력 변수**로 넣어줘야 합니다. 기술적으로 Python OpenCV나 딥러닝 모델로 사람을 세고, PyFMI로 FMU 입력을 설정하는 식으로 파이프라인을 구성할 수 있습니다. EnergyPlus 쪽에서는 **해당 입력이 반영되도록 People 스케줄이나 EMS를 구성**해야 하며, 앞서 설명한 ExternalInterface 매핑을 통해 이것이 가능합니다 [17]. 이러한 Vision+시뮬레이션 연동은 연구 프로토타입으로 시도된 바 있으며 (예: 카메라 기반 occupancy 패턴 인식을 EnergyPlus 예측 제어에 적용한 연구 [14] 등), 오픈소스로는 LBNL의 obFMU 툴이 유사 사례로 볼 수 있습니다. 다만 카메라를 통한 실시간 인원 계수는 **프라이버시 이슈**와 정확도 문제가 실무 과제로 남아 있으므로, 시스템 구축 시 딥러닝 모델의 정확도, 오인식에 따른 오류 대응, 데이터 프라이버시 보호 방안도 고려해야 합니다.

4. 실시간 기상 데이터(Forecast/API)를 FMU 입력으로 활용하는 방법

EnergyPlus 모델에서 **외부 기상 조건**(Outdoor weather)은 통상 .epw 파일에 정의된 값(시간별 기상 데이터)을 사용합니다. 그러나 **실시간 기상청 API**나 기타 날씨 데이터 소스로부터 얻은 최신 정보를 시뮬레이션에 반영하려면, 몇 가지 기법을 활용해야 합니다.

(a) Weather 파일 갱신: 한 방법은 EnergyPlus 실행 전에 **EPW 날씨 파일을 최신 예보/실측 데이터로 동적으로 생성하거나 수정**하는 것입니다. 예를 들어 기상청 API로 현재 시각부터 향후 24시간의 예보를 받아 .epw 형식으로 저장한 뒤, FMU를 생성하거나 실행할 때 그 파일을 사용하도록 할 수 있습니다. EnergyPlusToFMU로 FMU 생성 시 특정 기간의 날씨 데이터를 포함할 수 있으므로, 매일 갱신된 EPW로 FMU를 다시 만들어 사용하거나, FMU 리소스의 EPW를 교체하는 방법도 생각해볼 수 있습니다. 그러나 FMU 내부의 EPW를 동적으로 바꾸는 것은 표준 절차는 아니므로, 대신 (b) 방법을 사용하는 편이 일반적입니다.

(b) EMS를 통한 외부 날씨 덮어쓰기: EnergyPlus에는 EMS 기능으로 **외부 기상값을 런타임에 override**하는 방법이 있습니다. 예를 들어, **Outdoor Dry-bulb Temperature**나 **Outdoor Humidity** 등에 대응하는 EMS **Actuator**가 제공되며, 이를 통해 EnergyPlus가 날씨 파일에서 읽은 값 대신 사용자가 지정한 값을 사용할 수 있습니다 [23]. 구체적으로, IDF에 `EnergyManagementSystem:Actuator` 객체로 Weather Data의 Outdoor Dry Bulb를 액추에이터로 정의하고, `EnergyManagementSystem:Program`에서 해당 값을 외부 입력으로 설정된 EMS 변수로 갱신하는 스크립트를 작성합니다. ExternalInterface:FunctionalMockupUnitExport:To:Variable로 **실시간 외기온도 값을 받을 EMS 변수**(예: `ExtDryBulbTemp`)를 하나 만들고, EMS 프로그램에서 매 시간마다 `SET WeatherData:OutdoorDryBulb = ExtDryBulbTemp;` 와 같이 현재 외기온도를 업데이트하는 식입니다. 그러면 EnergyPlus는 다음 시간 스텝의 계산에 이 외기온도를 사용하게 됩니다 [23]. 이때 습도나 천공량, 운량 등 관련 기상 요소도 함께 보정해주는 것이 일관성에 좋습니다. Aaron Boranian의 설명에 따르면, 외기온도는 EMS로 덮어쓰되, **정확성을 위해서는 상대습도 등도 같이 업데이트**해주는 것이 바람직하다고 합니다 [23]. (한편, EnergyPlus 9.x 버전부터는 Python Plugin을 통해 외부 데이터를 직접 호출하여 날씨를 세팅하는 것도 가능해졌습니다 [24].)

(c) 마스터 알고리즘에서 주기적 입력: FMU를 실행하는 마스터 쪽에서, **주기적으로 기상 API를 호출**하여 새로운 데이터를 가져오고, 이를 FMU에 입력으로 넣는 접근도 자연스럽습니다. 예를 들어 Python 코드에서 기상청의 REST API를 일정 주기(예: 10분마다) 호출하여 현재 실외 온도, 습도, 일사량 등을 얻은 뒤, 해당 값을 FMU의 `OutdoorTemp`, `OutdoorHumidity`, `SolarRadiation` 등의 입력 변수로 `model.set()` 해줍니다. EnergyPlus 모델이 EMS나 Schedule 등을 통해 이 입력을 참조하도록 설정되어 있으면, 시뮬레이션에 즉시 반영됩니다. 이러한 구조를 사용하면, **실시간 기상 변화나 돌발적인 날씨 이벤트**(갑작스런 기온 강하 등)도 모사할 수 있고, 예측 제어에서는 최신 예보를 지속 반영하여 미래 에너지 수요를 좀 더 정확히 예측할 수 있습니다.

기술적으로, 기상청의 OpenAPI는 Python의 `requests` 라이브러리 등을 통해 호출해 JSON 또는 XML 응답으로 데이터를 받을 수 있습니다. 예를 들어 "단기예보" API를 사용하면 향후 수시간의 기온, 습도, 강수확률 등을 받을 수 있고, "현재날씨" API로는 실시간 관측값을 얻을 수 있습니다. 이러한 값을 얻은 후 단위를 EnergyPlus가 기대하는 단위(예: °C, %RH 등)로 변환하여 FMU에 투입하면 됩니다.

주의점: EnergyPlus 시뮬레이션 시간과 **실제 시간대(Timezone)**를 맞추는 것이 중요합니다. 기상청 API의 시간표준(예: KST)을 EnergyPlus의 시뮬레이션 로컬 시간과 일치시켜야 하며, 써머타임 등의 보정도 고려해야 합니다. 또한, 시뮬레이션이 과거 시간을 가리키고 있다면 실제 과거의 날씨 데이터(기상청 과거자료 API)를 넣어야 의미가 있으므로, **시뮬레이션 진행 방식**에 따라 적절한 시점의 데이터를 넣는 로직이 필요합니다.

결론적으로, **실시간 외부 날씨 데이터**는 EnergyPlus FMU의 입력으로 충분히 활용 가능합니다. 일반적으로는 **EMS Actuator를 통한 override 기법**이 사용되며, 이를 위해 ExternalInterface 입력을 EMS와 연계합니다 [23]. 대안으로, **EnergyPlus Python API (Plugin)**를 사용하면 FMU 없이도 EnergyPlus 내부에서 웹 API를 호출해 날씨를 설정할 수 있는데 [24], FMU 접근과 유사한 개념입니다. 어떤 방법이든 **실시간 날씨 API→시뮬레이션 입력** 파이프라인을 구축함으로써, 보다 정확한 실시간 예측 및 제어가 가능해집니다.

5. 실시간 Co-simulation 구조의 응답 속도와 동기화 방법

실시간 Co-simulation에서는 **시뮬레이션 시간**과 **현실 시간**의 진행을 효과적으로 동기화하는 것이 중요합니다. EnergyPlus FMU와 외부 데이터를 연동한 시스템에서, 일반적으로 **마스터 알고리즘**이 일정한 주기로 FMU를 스텝 진행시키고 센서 데이터를 갱신합니다. 이때 **응답 속도**란 센서 데이터 변화가 시뮬레이션 결과에 반영되는 데 걸리는 시간, 그리고 제어 명령 산출까지의 지연을 의미합니다. 목표는 이 지연을 최소화하고, 시뮬레이션이 **실시간성을 유지**하도록 하는 것입니다.

동기화 (Synchronization): 마스터 프로그램은 **고정된 시뮬레이션 스텝 간격**(time step)을 설정해 운영하는 경우가 많습니다. 예를 들어 60초마다 센서값을 읽어 FMU에 입력하고, FMU를 60초 앞으로 진행(do_step)시킨 후 출력을 얻는 식입니다 [10]. 이렇게 하면 시뮬레이션 시간 60초당 실제 60초가 흐르는 것이 이상적이지만, 실제로는 EnergyPlus 연산에 시간이 걸립니다. 만약 EnergyPlus 모델이 작고 계산이 빠르면 60초 시뮬레이션을 1초만에 끝낼 수도 있고, 반대로 모델이 크고 복잡하면 60초 시뮬레이션에 실제 120초가 걸릴 수도 있습니다. **실시간 동기화**를 위해 두 가지 전략이 있습니다:

- **동기 실행:** 시뮬레이션을 실제 시간에 맞춰 **속도 조절**을 합니다. 예를 들어 1분 스텝 시뮬레이션이 0.5초만에 끝나면, 다음 스텝 시작 전에 59.5초 대기(sleep)하여 실제 시간과 맞추고, 반대로 1분 시뮬레이션에 70초가 걸렸다면 (실시간성 위배) 이를 알려주고 처리하는 방식입니다. BCVTB와 EnergyPlus를 연동했던 과거 연구에서는 **실시간 모드**로 동작을 맞추기도 했습니다 [12]. 다만 이 방식은 EnergyPlus 계산이 반드시 실시간보다 빠르거나 같을 때만 유효합니다.
- **비동기 실행:** 시뮬레이션을 가능한 빠르게 돌리고, 새로운 센서 데이터가 들어올 때마다 현재 시각 이후로 보정(sync)하는 방법입니다. 예를 들어 1분마다 센서값이 업데이트되는데, 시뮬레이터가 늦게 따라오면 중간 상태를 건너뛰고 현재 시각에 맞춰 재설정(예: state reset)하거나, 다음 주기 제어계획을 수정합니다. **에뮬레이션 플랫폼**(예: BOPTEST 등)은 일반적으로 시뮬레이션 시간을 자유롭게 가속/감속하며 제어 알고리즘을 시험하지만, 실제 장비와 연결된 HIL(hardware-in-loop) 상황에서는 동기 실행이 요구됩니다.

EnergyPlus FMU의 경우 **고정 시간 스텝** 진행이 일반적이며, 마스터가 `do_step(step_size=t)`로 호출하면 EnergyPlus 엔진은 그 기간을 내부 세분 시간(step)으로 계산하고 완료 시 반환합니다 [10]. 따라서 마스터는 `do_step` 함수의 **반환 시점**을 기준으로 시뮬레이션이 해당 스텝을 완료했다고 간주하고 다음 작업을 합니다. 마스터 알고리즘의 의사코드는 앞서 제시한 바와 같이 다음과 같습니다 [10]:

```
loop:
    - 실세계 시각 읽기 (또는 이전에 계산된 현재 simulation time)
    - 새로운 센서/비전/날씨 데이터 수집
     - fmiSetReal() 로 FMU 입력 갱신 [25]
     - fmiDoStep() 로 FMU 한 주기 진행 [25]
     - fmiGetReal() 로 FMU 출력 읽음 [25]
    - 출력 이용 (제어연산 등) 후 loop 반복
```

동기화를 위해, **센서 데이터 갱신 주기**와 **FMU 스텝 크기**를 동일하게 설정해야 합니다. 예를 들어 센서값이 1분마다 들어오는데 FMU를 5분 단위로 건너뛰면, 중간 4분은 실제 데이터 반영 없이 추정하게 되므로 오차가 커집니다. 반대로 센서 주기가 5분인데 FMU를 1분씩 잘게 돌린다면, 5번의 시뮬레이션 스텝 동안 입력이 변하지 않아 계단 형태로 작동하지만 큰 문제는 없습니다 (오히려 안정적일 수 있음). **권장되는 접근**은 **최소 필요한 주기**로 시뮬레이션을 도는 것입니다. HVAC 제어에서는 1~5분 간격이 일반적이고, 그보다 빠른 제어(예: 초단위)는 건물 열관성 상 드물기 때문에 EnergyPlus도 분단위 정도로 충분한 경우가 많습니다. EnergyPlus 모델에서 `System Step` (HVAC 시스템 시뮬레이션)은 기본 6회/시간(10분)이나, 필요시 EMS로 1분 이하 제어도 가능하므로, **1분 미만 주기** 제어를 목표로 하면 EnergyPlus의 내부 시간해상도 설정을 매우 세밀하게 해야 합니다.

응답 속도: 앞서 언급했듯, EnergyPlus 계산 시간에 따라 실시간 응답에 지연이 생길 수 있습니다. 예컨대 복잡한 전체 빌딩 모델을 상세 HVAC까지 포함해 시뮬레이션하면 1분을 계산하는데 실제 2분이 소요될 수 있습니다. 이런 경우 실시간 제어에는 지장이 생기므로, **모델 경량화** 또는 **병렬 처리**가 필요할 수 있습니다. 모델 경량화는 중요하지 않은 디테일(예: 복잡한 일사모델이나 미세한 자연환기 해석 등)을 줄이고, 코어 부분(실내온도, 주요 장비 성능)에 집중하는 것입니다. 또한 하드웨어 성능을 높이거나, 여러 FMU(예: 건물 열부하 FMU + HVAC Modelica FMU 등)로 분할하여 병렬 실행하는 방안도 있을 수 있습니다.

마스터 알고리즘은 일반적으로 단일 스레드로 FMU와 통신하지만, **멀티스레딩**으로 센서 수집과 시뮬레이션을 병행할 수도 있습니다. 예를 들어 다음 시간 스텝의 센서값을 미리 받아오는 동안 이전 스텝 시뮬레이션을 돌리는 식입니다. 다만 이런 최적화는 구현이 복잡해질 수 있어, 기본적으로는 **순차 루프**로 충분히 빠른지 확인한 후 시도합니다.

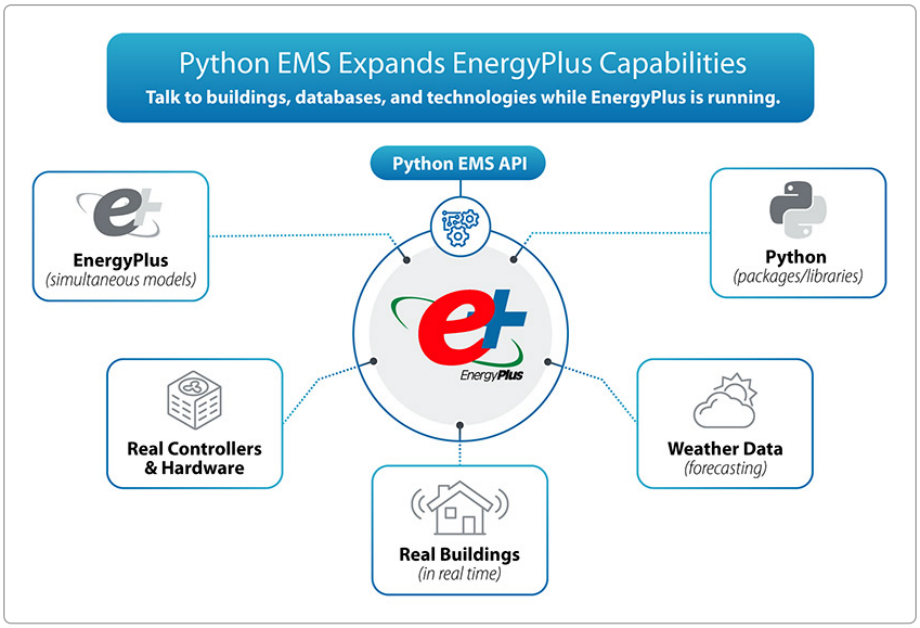
마지막으로, **Event 동기화**의 측면도 있습니다. 현실에서는 이벤트(예: 사람이 갑자기 들어옴, 장비 고장)가 불연속적으로 발생하는데, 시뮬레이션은 고정 시간 간격으로만 입력을 바꾸면 이를 즉각 반영 못할 수 있습니다. 이 경우 **이벤트 트리거**를 설계하여, 필요한 경우 시뮬레이션 스텝 중간이라도 FMU를 일시 중지하고 새로운 입력을 반영하도록 할 수 있습니다. FMI 2.0에는 이를 위한 Early Return이나 Event Mode 등이 있지만, EnergyPlus FMU는 기본적으로 일정 시간 간격 추진이므로 이벤트 대응은 입력 주기를 충분히 짧게 설정하는 방식으로 커버합니다.

정리하면, **실시간 Co-simulation**에서는 **주기 설정**과 **마스터 알고리즘의 시간 관리**가 핵심입니다. 일반적으로 **동일 주기의 폴링(polling)**으로 입력/출력을 주고받으며 FMU를 진행시키고 [10], 필요 시 실제 시간과 동기화를 맞추는 로직을 추가합니다. 이러한 구조는 EnergyPlus를 포함 여러 FMU를 동시에 돌릴 때에도 동일하게 적용되며, FMI를 이용한 Co-sim 인터페이스의 표준 절차입니다. LBNL의 BCVTB처럼 과거에는 실시간 동기화를 돕는 미들웨어도 있었지만 [12], 현재는 Python 같은 고수준 언어로 직접 타이밍을 제어하는 방식이 많습니다. **마스터 알고리즘 설계**에 따라 충분히 실시간 제어에 가까운 응답 속도를 달성할 수 있으며, 실제 구현에서 병목이 되는 부분(계산 시간, 통신 지연 등)을 모니터링하면서 튜닝하는 것이 중요합니다.

6. 구현 타당성 및 실제 사례 분석

질문에서 제시한 **FMU + 센서 + Vision 조합의 고해상도 공간 열환경 모델** 기반 **초개인화 예측 제어 시스템**은 **기술적으로 구현 가능**한 것으로 평가됩니다. 다만 여러 현실적인 고려사항이 있습니다. 이를 순차적으로 살펴보면 다음과 같습니다:

① **기술적 구성의 성숙도:** EnergyPlus를 FMU로 활용하는 기술은 이미 LBNL, NREL 등을 통해 연구되고 도구화되었습니다. EnergyPlusToFMU를 통한 FMU 내보내기 [2], PyFMI/FMI++를 통한 외부 제어 [10] 등은 확립된 방법입니다. 또한 최근 EnergyPlus 자체에 **Python EMS API**가 도입되어, 굳이 FMU로 감싸지 않아도 Python으로 실시간 데이터 연동이 가능해지는 등 (아래 그림 참고) 에코시스템이 발전하고 있습니다 [24]. 이는 EnergyPlus 개발 커뮤니티가 실시간 시뮬레이션 및 외부 연동의 중요성을 인지하고 적극적으로 대응하고 있음을 보여줍니다. 다시 말해, **건물 에너지 시뮬레이션을 실시간 예측 제어에 활용**하려는 시도는 최신 흐름에 부합하며, 필요한 표준과 라이브러리들이 갖춰져 있습니다.



EnergyPlus를 외부 시스템과 연동하는 개념도. (Python EMS API를 활용한 예로, 실시간 날씨 데이터, 실제 건물/장비, 머신러닝 라이브러리 등과 EnergyPlus 엔진을 연결할 수 있음을 보여준다 [26].) FMU를 활용하는 경우도 이와 유사하게, EnergyPlus 모델을 외부 프로그램과 연결하여 센서/예보 데이터를 주고받고 제어에 활용할 수 있다.

② **모델 정확도와 해상도:** 고해상도 공간 열환경 모델이라 하면, 공간을 세밀하게 구획하거나 CFD에 가까운 세밀한 해석을 포함한 모델일 수 있습니다. 이러한 **고정밀 모델은 계산 부하 증가**를 동반합니다. 실시간 제어에 투입하려면 **모델 단순화와 보정(calibration)**이 필요합니다. 예를 들어 한 공간을 격자 여러 개로 세분한 상세 모델은 실시간 계산이 어려울 수 있으므로, 구역을 통합하거나 중요하지 않은 세부현상(예: 상세 복사교환)을 단순화해야 할 수 있습니다. 또한 모델이 현실과 괴리가 없도록 충분한 실측데이터로 보정되어야, 예측 제어가 효과를 발휘할 수 있습니다. 구현 단계에서 **센서 데이터를 이용한 동적 보정**(on-line calibration or state estimation) 기법을 병행하면 모델 정확도를 높일 수 있습니다. 예컨대 칼만 필터 등을 이용해 EnergyPlus의 몇 가지 상태(존 온도 등)를 센서값과 맞추는 기법도 연구되고 있습니다.

③ **데이터 수집 및 통합:** 센서 (온도, 습도, CO_2 등 환경센서)와 Vision (카메라)로부터 데이터를 수집하여 통합하는 작업은 IoT/IBS 측면의 공학입니다. 카메라 영상 → 점유정보 변환을 위해 딥러닝 모델이 필요하고, 이는 별도의 AI 모듈 개발을 의미합니다. 다행히도 최근 **Vision 기반 점유센싱**에 대한 연구가 활발하여 공개된 알고리즘이나 플랫폼이 있습니다 [27]. 예를 들어 Dynamic HVAC Operations with Real-Time Vision-Based System 연구에서는 카메라와 각종 센서로 점유 패턴을 인식하고 EnergyPlus+obFMU로 통합 시뮬레이션을 한 사례가 보고되었습니다 [28]. 이러한 사례

는 **구현의 실증적 근거**가 됩니다. 또한 LBNL의 obFMU 툴은 사람이 창문 열기/닫기, 조명 제어, 온도세팅 변경 등의 행위를 모사하여 EnergyPlus와 교차 시뮬레이션하는데 사용되었습니다 [21]. 이는 실시간 카메라가 아닌 **가상 에이전트** 기반이지만, EnergyPlus와 **동등한 속도로 상호작용**하며 제어변수를 바꾼 점에서 일종의 실시간 Co-simulation 사례입니다.

④ **제어 알고리즘의 복잡도:** 예측 제어(MPC 등)를 위해서는 시뮬레이션 모델을 활용해 **미래 예측**을 하고 최적화를 수행해야 합니다. 이 과정은 단순한 피드백 제어보다 계산이 많이 들 수 있습니다. 실시간으로 이뤄지려면, FMU 시뮬레이션을 다수 차례 반복해서 다양한 제어입력 시나리오를 평가하거나, **미분 계산** 등을 해야 할 수 있습니다. EnergyPlus는 본질적으로 건물물리 시뮬레이션 툴이라, 그 자체로 최적화 알고리즘을 제공하지는 않습니다. 따라서 **외부에서 최적화 루프**를 구성해야 하며, 이때 FMU를 여러 번 호출해야 할 수도 있습니다. 예를 들어 1시간 앞을 내다보는 MPC를 10분 간격 제어로 운용한다고 하면, 6회분의 제어신호를 결정하기 위해 수십~수백 회 EnergyPlus 시뮬레이션 평가가 필요할 수 있습니다. 이렇게까지 복잡한 MPC를 실시간 구현하려면 병렬 클라우드 컴퓨팅이나, surrogate model 병용 등 추가 기술이 필요합니다. 다행히, 사용자가 원하는 것은 "초개인화" 제어이므로, **각 개인 주변의 미세한 환경제어** (예: 개인용 에어컨이나 퍼스널 팬)를 최적화하는 것일 가능성이 큽니다. 이 경우 중앙 HVAC 최적제어와는 다른 분산제어 개념도 고려해야 할 수 있습니다. 요컨대, **제어 전략에 따른 계산 요구**를 사전에 평가하고 시스템 설계를 해야 합니다.

⑤ **실무 적용 사례:** 현재까지 완전히 동일한 구성(에너지 시뮬레이터+실시간 센서+비전)을 상용 빌딩에 적용한 사례는 많지 않으나, 부분적인 구현 예는 존재합니다. 몇 가지 예를 들면: - Lawrence Berkeley NL은 2011년에 **실시간 EnergyPlus 시뮬레이션으로 빌딩 HVAC를 제어**하는 개념을 BCVTB로 시연했고, 2016년에는 실시간 에너지시뮬레이션을 활용한 MPC 연구를 발표했습니다 [29]. 이 연구들에서 EnergyPlus의 결과를 실시간으로 활용하는 가능성과 한계(모델링 오차, 계산 시간 등)가 논의되었습니다 [30]. - IBPSA Project 1의 **BOPTEST** (Building Optimization Performance Test) 프레임워크는 건물 및 HVAC 모델을 FMU로 제공하여 제어 알고리즘 개발자가 실시간으로 테스트할 수 있는 환경입니다 [31] [32]. BOPTEST의 FMU들은 Modelica나 EnergyPlus 기반 모델이며, 이것을 REST API로 제어·모니터링할 수 있게 한 것으로, **가상 건물 테스트베드**라고 볼 수 있습니다. 이는 질문의 시스템과 매우 유사한 개념으로, 오픈소스로 공개되어 있어 참조 가능하겠습니다. - 국내에서도 스마트빌딩이나 지능형 BEMS의 일환으로 **디지털 트윈**을 구축하는 시도가 일부 있으며, EnergyPlus와 IoT 데이터를 연동하는 PoC 수준의 프로젝트들이 보고되고 있습니다. 다만, 아직 연구단계인 경우가 많고 상용 솔루션에 적용되었다기보다는 실험적인 구현입니다.

이상의 사례와 기술을 종합해 보면, **제안된 시스템은 충분히 구현 가능**하지만, **통합 엔지니어링 노력이 많이 필요한 프로젝트**임을 알 수 있습니다. EnergyPlus FMU 자체는 검증된 시뮬레이터이지만, **실시간 데이터 파이프라인**, **비전 AI 통합**, **예측 제어 알고리즘** 각각이 전문 분야입니다. 따라서 전체 시스템의 안정성을 확보하려면 각 모듈의 오류가 전체에 미치지 않도록 **예외 처리와 폴백 전략**도 고민해야 합니다. 예를 들어 카메라 인식이 일시적으로 실패하면 occupancy를 최근 값으로 유지하거나 별도 센서로 보완하고, 실시간 데이터가 끊기면 스케줄 기반으로 임시 동작하도록 하는 등 **실무적인 예외 시나리오**를 대비해야 합니다.

타당성을 요약하면: 이 시스템은 **이론적으로 타당**하며, 필요한 구성 요소들이 (EnergyPlus-FMU, PyFMI, Computer Vision, IoT 센서, Weather API 등) 개별적으로 존재하고 일부는 성공 사례가 있습니다 [12] [33]. 그러나 **통합 구현의 복잡성**과 **성능상의 이슈**가 도전과제입니다. 적절한 모델 단순화로 실시간성을 확보하고, 데이터 흐름을 견고하게 구성하며, 단계적으로 테스트(예: 먼저 센서+EnergyPlus, 다음 Vision 추가, 마지막으로 제어 적용)하는 접근이 현실적입니다. 결국 충분한 시간과 리소스를 투입하면 **실제 건물에 적용 가능한 디지털 트윈 기반 예측 제어**로 완성될 수 있을 것으로 보입니다. 이는 향후 **개인 맞춤형 실내환경 제어**(온도 선호도, 환기 요구에 맞춘 제어)와 **에너지 효율 개선**을 동시에 달성할 유망한 방향이며, 연구/산업적으로 가치가 높다고 판단됩니다.

참고자료 및 추가 정보: EnergyPlusToFMU 사용자 가이드 [4] [6], UnmetHours Q&A 모음 [3] [23], LBNL/FMUs 관련 논문 [12], NREL의 Python EMS 소개글 [24] 등이 이 주제에 도움이 됩니다. 또한 "Real-time Building Energy Simulation using EnergyPlus and BCVTB (2011)" 논문, Energy Conversion and Management (2016)에 실린 실시간 MPC 연구 [34], Building and Environment 등에 실린 **Vision 기반 occupancy 제어** 논문 [13] 등을 참고하면 구체적인 구현 사례와 성능 결과를 살펴볼 수 있습니다. 이러한 선행 연구들을 기반으로 하면, 질문의 시스템 구현에 대한 **실증적 통찰**을 얻고 시행착오를 줄이는 데 도움이 될 것입니다.

---

1 Exporting EnergyPlus as a Functional Mock-up Unit for co-simulation: External Interface(s) Application Guide — EnergyPlus 8.2
https://bigladdersoftware.com/epx/docs/8-2/external-interfaces-application-guide/exporting-energyplus-as-a-functional-mock-up-unit.html

2 Introduction — FMU Export User Guide
https://simulationresearch.lbl.gov/fmu/EnergyPlus/export/index.html

3 8 Energyplus FMU in Pyfmi - Unmet Hours
https://unmethours.com/question/37197/energyplus-fmu-in-pyfmi/

4 5 6 7 19 20 7. Best Practice — FMU Export of EnergyPlus User Guide
https://simulationresearch.lbl.gov/fmu/EnergyPlus/export/userGuide/bestPractice.html

9 The FMI++ Library — The FMI++ Library documentation
https://fmipp.readthedocs.io/

10 21 22 25 OB Modeling Tool – obFMU | Occupant Behavior Research at LBNL BTUS
https://behavior.lbl.gov/?q=node/4

11 how to run EnergyPlus FMU through PyFMI - Unmet Hours
https://unmethours.com/question/39951/how-to-run-energyplus-fmu-through-pyfmi/

12 17 18 23 Is it possible to use real time sensor data (of temperature in- and outside, CO2 levels, occupancy) in EnergyPlus instead of Schedules? - Unmet Hours
https://unmethours.com/question/37845/is-it-possible-to-use-real-time-sensor-data-of-temperature-in-and-outside-co2-levels-occupancy-in-energyplus-instead-of-schedules/

13 14 15 16 27 Dynamic HVAC Operations Based on Occupancy Patterns With Real-Time Vision- Based System
https://s3-eu-west-1.amazonaws.com/pstorage-cmu-348901238291901/12258146/DynamicHVACOperationsBasedonOccupancyPatternsWithRealTim.pdf

24 26 Python Opens Up New Applications for EnergyPlus Building Energy Simulation | NREL
https://www.nrel.gov/news/detail/features/2022/python-opens-up-new-applications-for-energyplus-building-energy-simulation

28 33 Dynamic HVAC Operations with Real-Time Vision-Based Occupant ...
https://www.researchgate.net/publication/327903259_Dynamic_HVAC_Operations_with_Real-Time_Vision-Based_Occupant_Recognition_System

29 Development of a method of real-time building energy simulation for ...
https://www.sciencedirect.com/science/article/pii/S0196890416000856

30 Development of a method of real-time building energy simulation for ...
https://www.researchgate.net/publication/293649861_Development_of_a_method_of_real-time_building_energy_simulation_for_efficient_predictive_control

31 [PDF] Building optimization testing framework (BOPTEST) for simulation ...
https://www.nrel.gov/docs/fy22osti/81588.pdf

32 Prototyping the BOPTEST Framework for Simulation-Based Testing ...
https://www.researchgate.net/publication/334307128_Prototyping_the_BOPTEST_Framework_for_Simulation-Based_Testing_of_Advanced_Control_Strategies_in_Buildings

34 Experimental Long-Term Investigation of Model Predictive Heat ...
https://www.mdpi.com/1996-1073/13/22/6016