논문 2012-07-38

CPS의 점진적인 개발 과정을 지원하는 실시간 시뮬레이션 프레임워크

(A Real-Time Simulation Framework for Incremental Development of Cyber-Physical Systems)

한 재 화, 위 경 수, 이 창 건*
(Jae-Hwa Han, Kyoung-Soo We, Chang-Gun Lee*)

Abstract : When developing a CPS, since it is nature of CPS to interact with a physical system, CPS should be verified during its development process by real-time simulation supporting timely interactions between the simulator and existing implemented hardwares. Furthermore, when a part of a simulated system is implemented to real hardwares, i.e., incremental development, the simulator should aware changes of the simulated system and apply it automatically without manual description of the changes for effective development. For this, we suggest a real-time simulation framework including the concept of 'port' which abstracts communication details between the tasks, and a scheduling algorithm for guaranteeing 'real-time correctness' of the simulator.

Keywords : Cyber-Physical Systems, Real-Time Systems, Real-Time Simulation, Incremental Development, HW Migration, Event Scheduling

Ⅰ. 서 론

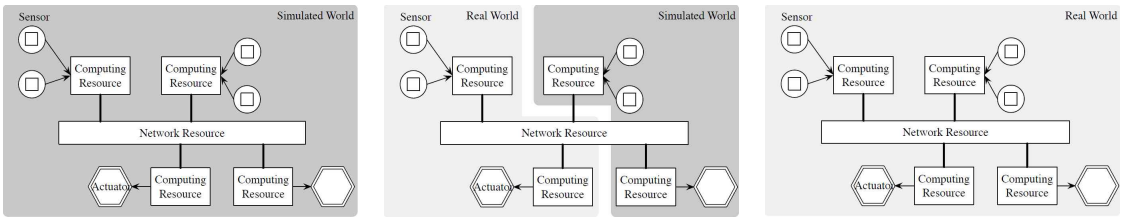
CPS (Cyber-Physical System)는 네트워크화된 컴퓨터들(Cyber System)이 물리적 환경(Physical System)과의 상호교류를 통하여 물리적 환경을 실시간으로 감지하고 제어하는 시스템이다[1]. 대표적인 CPS로는 자율주행 차량용 제어 시스템, 원격 수술 로봇 제어 시스템, 스마트 그리드 제어 시스템 등을 들 수 있다.

CPS는 다양한 센서와 액추에이터, 프로세서 같은 이기종 하드웨어(Hardware, HW)들로 구성되어 있으며, 하드웨어들이 네트워크를 통해 서로 통신하면서 시스템을 유지하므로 기존의 임베디드 시스템보다 훨씬 거대하고 복잡하다. 그러므로 개발 과정에서 검증 및 확인(Verification and Validation, V&V) 과정이 차지하는 비중이 크며, 결국 검증 및 확인 과정에서 소모되는 노력 및 비용이 전체 개발 과정의 효율성을 대변한다. 이러한 검증 및 확인 과정에서 가장 널리 사용되는 기법은 시뮬레이션 기법이다. 이는 실제 하드웨어 자원이나 실제적인 환경 없이 다양한 상황에서의 동작을 실험적으로 검증할 수 있기 때문으로, 특히 다양한 하드웨어로 구성되고, 복잡한 물리적 환경과 상호교류를 해야 하는 CPS는 개발 과정에서 시뮬레이션이 차지하는 비중이 클 수밖에 없다.

CPS의 개발 과정에서 아직 구현되지 않은 하드웨어 자원은 시뮬레이션으로 검증될 수 있다. 이러한 하드웨어 자원을 가상 하드웨어라고 하며 이들이 있는 환경을 시뮬레이션 되는 환경(Simulated World)이라고 하자. 반면에 실제 구현되어 시뮬레이터와 연동되는 하드웨어를 실제 하드웨어라고 부르며 이들이 있는 환경을 실제 환경(Real World)이

* Corresponding Author (cglee@snu.ac.kr)
Received: 31 July 2012, Accepted: 06 Sep. 2012.
J.H. Han, K.S. We, C.G. Lee: Seoul National Univ.
※ 본 연구는 지식경제부 및 한국산업기술평가관리원의 산업원천기술개발사업(정보통신)의 일환으로 수행하였음. [10035243, CPS를 위한 콤포넌트 기반 설계이론 및 제어커널 개발]
※ 이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다.

ISSN : 1975-5066



(a) 개발 초기 시점
(a) At the initial stage of the development process

(b) 개발 중간 시점
(b) In the middle of the development process

(c) 개발 완료 시점
(c) At the end of the development process

그림 1. CPS의 개발 과정
Fig. 1 Development process of CPS

라 하자. 다양한 이기종 하드웨어로 구성된 CPS는 개발 과정 중 필연적으로 시뮬레이션 되는 환경에 있던 가상 하드웨어가 개발이 진행됨에 따라 실제로 구현되어 실제 환경으로 옮겨가게 된다. 가상 하드웨어가 실제 하드웨어가 되면서 실제 환경으로 옮겨가는 것을 하드웨어 이동(HW Migration)이라 하며 이때 시뮬레이터가 시뮬레이션하려는 환경이 바뀌게 된다.

그림 1은 CPS의 개발 과정에 따른 하드웨어 이동을 나타내고 있다. 그림 1(a)는 개발 초기에 어떤 하드웨어도 실제로 구현되지 않은 상황을 나타낸다. 이때는 모든 하드웨어가 가상 하드웨어로 표현된다. 그림 1(b)는 어느 정도 개발이 진행되어 일부의 하드웨어 자원이 실제로 구현된 상황을 표현하고 있다. 이때는 시스템을 이루는 하드웨어 중 일부가 가상 하드웨어이며, 시뮬레이터는 가상 하드웨어와 실제 하드웨어 간의 연동을 지원해야만 한다. 마지막으로 그림 1(c)는 개발이 완료된 상황을 나타내고 있다. 이때는 시스템을 구성하는 모든 하드웨어가 실제로 구현된 상황이며, 더 이상 시뮬레이션이 필요하지 않다.

이렇듯 CPS의 개발 과정에서는 지속적으로 시뮬레이션 되는 환경이 변화하고 실제 하드웨어 자원과 시뮬레이터가 긴밀하게 연계되어야 하는 특성을 보인다. 그러므로 시뮬레이터가 CPS의 개발 과정을 제대로 지원하기 위해서는 전통적인 시뮬레이션 기법보다 더 고도화된 기능들이 요구된다. 그 요구사항으로는 다음과 같은 것들이 있다.

유연한 하드웨어 자원 이동(HW Migration) 지원 : CPS의 개발 과정 중에는 그림 1에서처럼 시뮬레이션 되는 환경이 지속적으로 변화한다. 이때, 시스템 개발자는 시뮬레이션 되는 환경이 바뀔 때마다 시뮬레이터에 시뮬레이션 될 환경을 새로 기술해주어야 하며 이러한 상황은 CPS의 개발에서 비일비재하게 발생할 수밖에 없다. 그러므로 개발 과정의 효율성을 높이기 위해, 시뮬레이터는 하드웨어 자원이 이동하여 시뮬레이션 되는 환경이 변화하는 상황에서 시스템 개발자의 수고를 최소화하며 이를 유연하게 지원할 수 있어야 한다.

하드웨어 자원 연동을 고려한 실시간 정확성(Real-Time Correctness) 보장 : CPS의 시스템 전반을 검증 및 확인하기 위해서 시뮬레이터는 아직 개발되지 않은 하드웨어 자원을 정확히 시뮬레이션 할 수 있어야 하며, 동시에 이미 개발이 완료된 하드웨어와의 연동을 지원해야 한다. 즉, 이미 개발된 하드웨어 자원의 입장에서 보자면, 시뮬레이터는 마치 실제 하드웨어들의 집합인 것처럼 동작하여야 한다. 이를 위해서 시뮬레이터에서는 아직 개발되지 않은 하드웨어 자원들의 논리적 정확성(Logical Correctness) 뿐만 아니라 시간적 정확성(Temporal Correctness)을 보장할 수 있어야 한다.

우리는 이미 [2]와 [3]에서 위에서 설명한 두 가지 요구사항을 언급한 바 있다. 본고에서는 [2]와 [3]에서 소개한 개념을 확장하여 CPS를 정확하게 시뮬레이션 하기 위한 시뮬레이션 프레임워크를 제안한다. 우리는 하드웨어 이동을 지원하기 위한 시뮬레이션 구조를 명세하며, 단일 코어(Single-Core) 컴퓨터에서 CPS의 실시간 정확성을 보장하면서 효율적으로 시뮬레이션 하기 위한 이벤트 스케줄링 알고리즘을 제안한다.

본고의 이후는 다음과 같이 구성되어 있다: 다음 장에서는 우리가 대상으로 하고 있는 시스템을 기술하며 문제를 정형화한다. 3장에서는 시뮬레이션 구조를 명세하고 이벤트 스케줄링 알고리즘을 제안한다. 4장에서는 관련된 연구에 대해 언급하며, 마

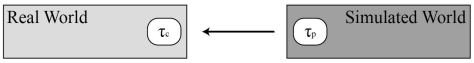


그림 2. 실제 환경과 시뮬레이션 환경의 연동
Fig. 2 Interaction between Real World and Simulated World

지막으로 5장에서는 결론을 맺는다.

II. 문제 기술

CPS에서는 다양한 하드웨어 자원을 이용하는데, 여기에서는 하드웨어 자원을 컴퓨팅 자원(Computing Resource)과 네트워크 자원(Network Resource)의 두 분류로 구분하도록 하자. 그리고 두 컴퓨팅 자원 간에 데이터가 교환된다면, 두 컴퓨팅 자원 사이에는 항상 네트워크 자원이 존재한다고 가정한다. 네트워크 자원은 공유 버스(Shared Bus)일 수도 있고 두 컴퓨팅 자원만 연결하는 전용선(Dedicated Line)일 수도 있다. 이때, 실제 환경과 시뮬레이션 되는 환경을 연결하는 하드웨어 자원은 무조건 네트워크 자원이어야 하며, 결국 그림 1(b)에서 볼 수 있듯이 일부의 네트워크 자원은 실제 환경과 시뮬레이션 되는 환경의 경계에 존재하여 시뮬레이터와 실제 하드웨어 자원을 연결한다. 시뮬레이터는 이 네트워크 자원을 이용하여 실제 환경과 데이터를 주고받는다.

이러한 하드웨어 자원 위에서 각 태스크 τ_i는 주기(Period) P_i, 최악 수행 시간(Worst-Case Execution Time) C_i, 위상(Phase) ϕ_i로 동작한다고 가정하며, 시뮬레이터를 구동시키는 컴퓨터(시뮬레이션 호스트 컴퓨터)와 실제 태스크가 구동되는 하드웨어 자원 간의 계산속도 차이도 주어져 있다고 가정한다. 이때, 컴퓨팅 자원에서 수행되는 태스크의 수행 시간은 소스 코드가 해당 컴퓨팅 자원에서 수행되는 시간일 것이며, 네트워크 자원에 있는 태스크의 수행 시간은, 네트워크 자원의 컨트롤러가 메모리에 있는 데이터를 패킷화하는 시간과 그 패킷이 실제 네트워크 채널에 실린 후 전달되는 전파 딜레이(Propagation Delay)의 합이 될 것이다.

CPS에서는 각 태스크 간에 데이터를 주고받으면서 물리적 환경을 실시간으로 감지, 제어한다. 그러므로 각 태스크 간에는 데이터 의존관계(Data Dependency)가 있다고 가정하며, 태스크 τ_i의 출력 데이터(Output Data)를 태스크 τ_j가 입력 데이터(Input Data)로 사용한다면 $\tau_i \rightarrow \tau_j$와 같이 표현한다. 또한, $\tau_i \rightarrow \tau_j$ 관계에 있는 두 태스크는 서로 독립적으로 동작한다고 가정한다. 즉, τ_j는 τ_i에 의해 수행이 시작되는 것(Triggering)이 아니라 자신의 주기에 의해 수행을 시작하는 주기적 태스크이다. 이때, τ_j는 자신이 수행을 시작하는 시점에서 τ_i가 출력으로 내놓은 가장 최근의 데이터를 입력으로 사용하며, τ_j의 수행 도중에 τ_i에서 새로운 데이터가 출력으로 나온다고 해도 그 데이터는 τ_j의 이번 수행에 반영되지 않는다고 가정한다. 즉, 태스크는 자신이 시작할 때만 가장 최근의 데이터를 읽어 들여 자신의 계산에 반영한다고 가정한다. 또한 자신의 계산이 끝난 이후에야 자신의 출력 데이터를 내놓는다고 가정한다.

위에서 소개한 가정들을 이용하여, CPS를 정확하게 시뮬레이션 하기 위한 입력 정보를 다음과 같이 정의한다.

- 전체 시스템 구조 및 목적 시스템 : 개발하고자 하는 CPS 전체의 하드웨어 연결 구조 및 그 중 시뮬레이션 되는 환경의 범위.
- 태스크-하드웨어 자원 매핑 및 데이터 의존관계 : 각 하드웨어에서 수행되는 태스크와 각 태스크 간의 데이터 의존관계.
- 각 하드웨어 자원 및 태스크의 타이밍 정보 : 각 하드웨어 자원의 컴퓨팅 성능, 태스크 스케줄링 정책, 각 태스크 τ_i의 P_i, C_i, ϕ_i 등의 타이밍 정보.
- 각 태스크의 소스코드 : 시뮬레이션 호스트 컴퓨터에서 수행가능한 각 태스크의 소스코드.

위의 정보들을 시뮬레이션을 위한 시스템 환경(System Configuration)이라 부르자. 본고에서는 시스템 환경이 주어졌을 때, 주어진 시스템을 단일 코어 컴퓨터에서 정확하고 효율적으로 시뮬레이션 하기 위한 시뮬레이션 프레임워크를 제안하고자 한다. 이를 위해, 시뮬레이션이 제대로 되는지 판단의 척도가 되는 실시간 정확성에 대해 정의한다.

시뮬레이션의 실시간 정확성은 단순하게 다음과 같이 정의할 수 있다: 실제 환경(Real World)에 있는 실제 하드웨어 자원의 입장에서 봤을 때, 시뮬레이션 되는 환경(Simulated World)에 있는 가상 하드웨어들이 마치 실제 하드웨어들인 것처럼 동작하여야 한다. 바꿔 말하면, 시뮬레이터에서 나오는 출력들이 올바른 값(Logically Correct)이어야 하며, 출력들이 나오는 시기가 올바른 때(Temporally Correct)여야 한다. 그림 2를 보자.

그림 2는 실제 환경과 시뮬레이션 되는 환경이 연동되는 상황을 나타내고 있다. $\tau_p \rightarrow \tau_c$의 관계가 있

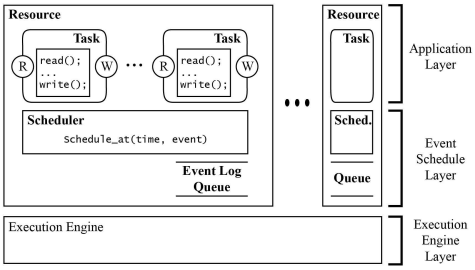


그림 3. 시뮬레이터의 계층적 구조

Fig. 3 Layered architecture of the simulator

고 τ_p가 시뮬레이션 되는 환경에, τ_c가 실제 환경에 있다면, τ_c의 입장에서 보았을 때, τ_p에서 나오는 출력 데이터가 τ_p가 실제 하드웨어에서 동작한 것과 같은 값을 가지고, 실제 하드웨어에서 동작했을 때와 같은 시점에 나온다면 논리적, 시간적으로 정확하게 시뮬레이션이 이루어졌다고 볼 수 있다. 정리하자면, 시뮬레이터에서 수행되는 태스크들이 실제 환경에서 보였을 실시간 행태를 예상 스케줄(Expected Schedule)이라고 했을 때, 시뮬레이터에서 수행되는 태스크 중 실제 환경에 출력을 보내는 모든 태스크에 대해 그 태스크의 출력이 기대되는 값을 갖고 예상 스케줄에서 계산된 시점에 실제 환경으로 나왔을 때 실시간 정확성이 보장되었다고 할 수 있다.

앞서 언급한 가정들과 소개한 개념들로 본고에서 풀고자 하는 문제를 다음과 같이 기술할 수 있다.

문제기술 :

- 유연한 하드웨어 이동을 지원하기 위하여 시뮬레이터는 어떤 구조를 갖추어야하고 하드웨어가 시뮬레이션 되는 환경에서 실제 환경으로 이동할 때 시뮬레이터는 어떤 작업을 해 주어야 하는가.
- 주어진 목적 시스템을 단일 코어 컴퓨터에서 실시간 정확성을 보장하면서도 효율적으로 시뮬레이션하기 위해서는 각 태스크들을 어떻게 스케줄링하여 수행해야하는가.

Ⅲ. CPS의 점진적인 개발 과정을 지원하는 시뮬레이션 프레임워크

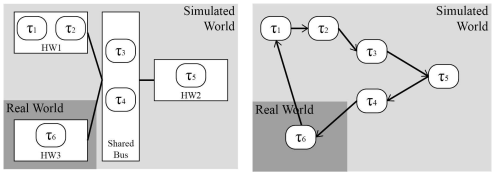
1. 하드웨어 자원 이동을 지원하는 시뮬레이터 구조

하드웨어 자원이 시뮬레이션 되는 환경에서 실제 환경으로 이동하거나 그 반대로 이동할 때, 시스템 개발자의 수고를 최소화하며 이를 유연하게 지원하기 위해서 시뮬레이터에서는 가상 하드웨어 자원을 새로 인식해야하고, 가상 하드웨어 자원 위에서 수행되는 태스크들이 주고받아야 하는 데이터의 경로를 재설정해야하며 각 태스크의 예상 스케줄을 생성하여 실시간 정확성을 보장하면서 시뮬레이션 할 수 있어야 한다. 이를 위해, 여기에서는 그림 3과 같은 시뮬레이터 구조를 제안한다.

본고에서 제안하는 시뮬레이터 구조는 응용 계층, 이벤트 스케줄 계층, 수행 엔진 계층의 3계층으로 이루어져 있다. 이 중 응용 계층은 하드웨어 자원이 이동하여 시뮬레이션을 위한 시스템 환경이 바뀌었을 때, 이를 인지하고 태스크 간 주고받아야 하는 데이터의 경로를 재설정하는 계층이다. 앞 장에서 소개한 태스크 모델에 의하면, 각 태스크들은 데이터 의존관계에 따라 서로 입력과 출력을 주고받으며 동작한다. 그런데, 시뮬레이션 되는 한 태스크에서 수행이 끝난 후 나온 출력 데이터를 다른 태스크로 보낼 때, 데이터를 받을 태스크가 시뮬레이션 되는 환경에 있는지 실제 환경에 있는지에 따라 데이터 전송이 달라진다. 예를 들어, 데이터를 받을 태스크가 실제 환경에 있다면, 시뮬레이션 호스트 컴퓨터는 실제 네트워크 자원을 이용하여 데이터를 실제 환경으로 내보내야 한다. 반대로 시뮬레이션 되는 환경에 있다면 단순히 해당 태스크의 메모리 버퍼에 데이터를 복사해주면 된다. 이러한 데이터 흐름 변경 과정은 시스템 환경이 바뀔 때마다 이루어져야 하며, 이 과정에서 시스템 개발자의 수고가 최소화되어야 한다.

이를 위하여 여기에서는 태스크의 입력과 출력을 추상화한 포트(Port)라는 개념을 소개한다. 이 포트는 각 태스크의 수행 전에 입력 데이터를 처리하는 읽기 포트(Read Port)와 태스크의 수행 완료 후에 출력 데이터를 보내는 쓰기 포트(Write Port)로 나뉜다. 그림 3의 응용 계층에 있는 태스크의 좌우에 붙은 R과 W는 각각 읽기, 쓰기 포트를 의미한다. 이제 태스크가 속한 환경과 데이터 의존관계에 따라 포트가 가져야 하는 특성을 정의한다.

이를 위해, 우리는 시뮬레이션 되는 환경에 있는 태스크의 포트를 그 위치에 따라 분류한다. 우선 $\tau_i \rightarrow \tau_j$의 관계가 있을 때, τ_i가 실제 환경에 있고 τ_j가 시뮬레이션 되는 환경에 있다면, τ_j는 자신의 입력 데이터를 실제 환경으로부터 받아야 한다. 데이터를 실제 환경으로부터 받는다는 것은 실제 환경과 시간 동기화가 이루어져야 한다는 뜻이므로, 그



(a) 하드웨어 연결 구조 및 태스크 매핑
(a) HW connection topology and task mapping

(b) 데이터 의존관계
(b) Data dependency

그림 4. CPS의 예제 시스템
Fig. 4 An example of CPS

표 1. 그림 4에서 도출된 각 태스크의 포트 종류 및 수
Table 1. Port character and number of each task derived from Figure 4

τ_i	읽기 동기화 포트	읽기 비동기화 포트	쓰기 동기화 포트	쓰기 비동기화 포트
τ_1	1	0	0	1
τ_2	0	1	0	1
τ_3	0	1	1	1
τ_4	0	1	1	0
τ_5	0	1	0	1

런 의미에서 τ_j의 읽기 포트를 읽기 동기화 포트(Read Sync. Port)라 부르고 동기화가 필요 없는 읽기 포트를 읽기 비동기화 포트(Read No-Sync. Port)라 부르자. 반대로 $\tau_i \rightarrow \tau_j$의 관계에서 τ_i가 시뮬레이션 되는 환경에 있고 τ_j가 실제 환경에 있을 때 τ_i는 자신의 출력 데이터를 실제 환경으로 노출시켜야 한다. 그러므로 τ_i의 쓰기 포트를 쓰기 동기화 포트(Write Sync. Port)라 부르고 동기화가 필요 없는 쓰기 포트를 쓰기 비동기화 포트(Write No-Sync. Port)라 부르자. 당연히 한 태스크가 둘 이상의 태스크로부터 데이터를 받거나, 둘 이상의 태스크로 데이터를 보내야 한다면 그 태스크의 읽기 또는 쓰기 포트는 역시 둘 이상이 된다.

각 포트에서 구체적으로 일어나야 하는 작업은 다음과 같이 정의된다.

- 읽기 동기화 포트 : 태스크가 수행을 시작하는 시점에서 그 태스크가 수행되는 시뮬레이션 호스트 컴퓨터의 네트워크 컨트롤러 버퍼로부터 실제 데이터를 읽어온다.
- 읽기 비동기화 포트 : 태스크가 수행을 시작할 때 그 태스크에 할당된 메모리 버퍼에서 데이터를 읽는다.
- 쓰기 동기화 포트 : 태스크가 수행을 완료하는 시점에서 그 태스크가 수행되는 시뮬레이션 호스트 컴퓨터의 네트워크 컨트롤러를 통해 외부로 데이터를 보낸다.
- 쓰기 비동기화 포트 : 태스크가 수행을 완료했을 때 출력 데이터를 의존관계에 있는 다른 태스크의 메모리 버퍼에 복사한다.

하드웨어 이동을 지원하기 위해서는 한 시스템 환경에서 다른 시스템 환경으로 변화되었을 때, 즉, 하드웨어 자원이 이동하였을 때, 이를 인지하고 올바르게 데이터를 전달하기 위하여 특정 태스크의 포트가 자동적으로 바뀌어야 한다. 데이터 의존관계가 주어지고 각 하드웨어 자원의 위치가 결정되었을 때 시뮬레이션 되는 환경에 있는 각 태스크의 포트는 다음과 같이 결정된다.

- $\tau_i \rightarrow \tau_j$ 관계가 있고 τ_i가 시뮬레이션 되는 환경, τ_j가 실제 환경에 있을 때, τ_i는 쓰기 동기화 포트를 갖는다.
- $\tau_i \rightarrow \tau_j$ 관계가 있고 τ_i가 실제 환경, τ_j가 시뮬레이션 되는 환경에 있을 때, τ_j는 읽기 동기화 포트를 갖는다.
- $\tau_i \rightarrow \tau_j$ 관계가 있고 두 태스크 모두 시뮬레이션 되는 환경에 있을 때, τ_i는 쓰기 비동기화 포트를, τ_j는 읽기 비동기화 포트를 갖는다.
- τ_i가 시뮬레이션 되는 환경에 있고, τ_i가 네트워크 자원에서 수행되며 이 네트워크 자원에 실제 환경의 하드웨어 자원이 하나 이상 연결되어 있을 때, τ_i는 쓰기 동기화 포트를 갖는다.

위에서 네 번째 항목은 데이터 의존관계와 무관하게 쓰기 동기화 포트를 가져야 하는 조건이다. 그림 4를 보자. 그림 4는 CPS의 한 예를 표시하고 있다. 그림에는 3개의 컴퓨팅 자원과 1개의 네트워크 자원으로 구성된 전체 시스템의 구조와 시뮬레이션 해야 하는 목적 시스템, 태스크-하드웨어 자원 매핑, 데이터 의존관계 등이 나타나 있다. 여기에서 τ_3를 보자. τ_3의 경우 실제로 데이터 의존관계만 보면 τ_5가 시뮬레이션 되는 환경에 있으므로 하나의 쓰기 비동기화 포트만 있으면 될 것 같다. 하지만, τ_3가 수행되는 공유 버스가 HW3와 연결되어 이미 실제 환경에 노출되었으므로 τ_3의 출력 역시 실제 환경으로 나가야 HW3 입장에서 정확한 시뮬레이션이 된다. 조금 더 부연하자면, 만일 어떤 시간 t에 τ_3가

시작되어 $t+2$에 끝난다고 가정하고 $t+1$에 HW3에서 τ_1으로 데이터를 보내려고 한다고 하자. 이때, τ_3에 쓰기 동기화 포트가 없다면 HW3는 시간 $t+1$에 그림 4의 공유 버스가 유휴상태(idle)라고 판단하여 데이터를 보낼 것이다. 하지만 실제로는 t에서 $t+2$까지 공유 버스는 τ_3가 사용하고 있으므로 HW3는 $t+2$가 되어야 데이터를 보낼 수 있다. 이를 위해서 τ_3는 실제로 시간 t에서 공유 버스에 데이터를 노출시켜야 한다. 그러므로 τ_3는 τ_4로 데이터를 복사하는 쓰기 비동기화 포트와 공유 버스를 통해 실제 환경으로 데이터를 보내는 쓰기 동기화 포트 둘을 모두 가진다. 그림 4의 시스템 환경으로부터 각 태스크들이 가져야할 포트의 종류 및 수를 구한 표는 표 1과 같다.

하드웨어 이동 시 시스템 개발자의 수고를 최소화하기 위하여 시뮬레이터에서는 앞서 설명한 읽기 포트와 쓰기 포트를 추상화한 API (Application Programming Interface)를 제공하여야 한다. read(), write()라는 포트를 위한 API가 있다고 하면, 시스템 개발자는 시뮬레이션을 위한 시스템 환경을 기술할 때 그림 3에서와 같이 이 API를 이용하여 각 태스크의 소스코드를 기술한다. 이후 시스템 개발자는 어떤 하드웨어 자원이 시뮬레이션 되는 환경에 있고 어떤 하드웨어 자원이 실제 환경에 있는지만 명세해주면, 시뮬레이터에서는 각 태스크의 포트 종류를 결정하고, 동시에 read(), write() API의 내용을 자동으로 코딩하여(Auto Coding) 각 포트에서 일어나야 하는 작업을 설정함으로써 유연한 하드웨어 이동을 지원한다.

2. 실시간 정확성을 보장하는 이벤트 스케줄링

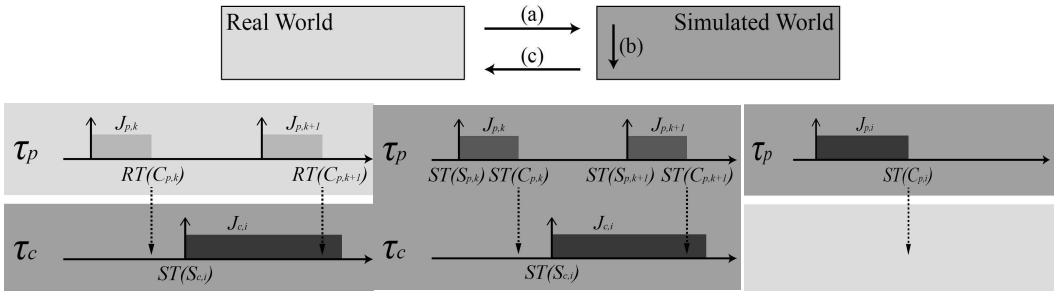
그림 3의 응용 계층에서 시뮬레이션 되는 환경이 정해지면, 두 번째 계층인 이벤트 스케줄 계층에서는 가상 하드웨어 자원 위에서 수행되는 태스크들이 실제 환경에서 보였을 실시간 행태인 예상 스케줄(Expected Schedule)을 생성한다. 이때, 각 가상 하드웨어 자원의 태스크 스케줄링 정책은 널리 사용되는 RM (Rate Monotonic)이나 EDF (Earliest Deadline First) 등 일 수도 있고, 이벤트 스케줄 계층에서 제공되는 API를 통해 시스템 개발자가 시뮬레이터에서 제공하지 않는 새로운 스케줄 정책을 구현하여 적용할 수도 있어야 한다. 이렇게 생성된 예상 스케줄은 이벤트 기록 대기열(Event Log Queue)에 저장된다. 이 단계에서는 실제로 태스크 수행이 일어나진 않으며, 단지 각 태스크의 시작과 완료 이벤트가 발생될 시간만 정해진다.

시뮬레이터의 응용 계층과 이벤트 스케줄 계층에서 시스템 환경이 결정되고 그에 따른 예상 스케줄이 생성되었을 때, 이제 그 하위 계층인 수행 엔진 계층에서는 2장에서 정의한 실시간 정확성을 만족시키면서 각 하드웨어 자원의 예상 스케줄에 맞추어 실제로 태스크의 코드를 시뮬레이션 호스트 컴퓨터에서 수행시켜야 한다. 이를 위해, 여기에서는 이벤트 스케줄 계층에서 생성된 병렬적인 (Parallel) 예상 스케줄을 실시간 정확성을 보장하면서 단일 코어 컴퓨터에서 효율적으로 수행하기 위한 이벤트 스케줄링 알고리즘을 제안한다.

우선 다음의 기호를 먼저 정의한다.

- $J_{i,j}$: 태스크 τ_i의 j번째 작업(Job)
- $S_{i,j}$: $J_{i,j}$의 시작 이벤트(Event)
- $C_{i,j}$: $J_{i,j}$의 완료 이벤트
- $ST(E)$: 예상 스케줄에 명시된 이벤트 E의 시간
- $RT(E)$: 실제 환경에서 이벤트 E가 발생하는 시간
- $A(E)$: 이벤트 E가 시뮬레이터에서 처리된 실제 시간

여기에서의 이벤트는 각 작업의 시작과 완료 시점에 발생하는 일을 의미한다. 더 구체적으로, 컴퓨팅 자원에서는 소스 코드가 수행되기 시작하고 완료되는 것을 의미하며, 네트워크 자원에서는 특정 데이터의 송신이 시작되고 완료되는 것을 의미한다. 이제 그림 5를 보자. 그림 5는 세 가지 상황을 나타내고 있다. 우선 그림 5(a)는 실제 환경에 있는 태스크 τ_p의 출력 데이터를 시뮬레이션 되는 환경에 있는 태스크 τ_c에서 사용하는 상황을 나타내고 있다. 임의의 i, k에 대해서, 작업 $J_{p,k}$의 종료 이벤트 $C_{p,k}$가 작업 $J_{c,i}$의 시작 이벤트 $S_{c,i}$ 전의 마지막 완료 이벤트이고 작업 $J_{p,k+1}$의 종료 이벤트 $C_{p,k+1}$이 $S_{c,i}$ 이후의 첫 완료 이벤트라고 하면, $J_{c,i}$는 $J_{p,k}$의 출력 데이터를 받아와야만 한다. 그러므로 $S_{c,i}$는 $C_{p,k}$와 $C_{p,k+1}$ 사이에 시뮬레이터에서 처리되어야 한다. 즉, $RT(C_{p,k}) \leqq A(S_{c,i}) < RT(C_{p,k+1})$의 조건이 만족되어야 한다. 그런데, 시뮬레이터 입장에서는 실제 환경에서의 시간인 $RT(C_{p,k})$와 $RT(C_{p,k+1})$을 알 수 없다. 그러므로 $RT(C_{p,k}) \leqq A(S_{c,i}) < RT(C_{p,k+1})$을 만족시킬 수 있는 유일한 방법은 $A(S_{c,i}) = ST(S_{c,i})$를 만족시키는 것이다. 이 조건을 실시간 정확성 보장을 위한 조건 1이라 부르자. 조건 1이 만족된다면, 시뮬레이터가 실제 환경으로부터 올바른 데이터를 받는 것이 보장된다. 그러므로 조건 1은 시뮬레이터의 논리적 정



(a) 실제 환경에서 시뮬레이션 되는 환경으로의 데이터 전달
(a) Data transmission from the real world to the simulated world

(b) 시뮬레이션 되는 환경 내부에서의 데이터 전달
(b) Data transmission in the simulated world

(c) 시뮬레이션 되는 환경에서 실제 환경으로의 데이터 전달
(c) Data transmission from the simulated world to the real world

그림 5. 데이터가 전달되는 세 가지 상황
Fig. 5 Three cases of data flows

확성을 보장하기 위한 조건이다.

그림 5(b)는 시뮬레이션 되는 환경에 있는 두 태스크 사이의 관계를 나타내고 있다. 두 태스크의 데이터 의존관계는 $\tau_p \rightarrow \tau_c$ 이므로 임의의 i, k에 대해서, 그림 5(b)에서처럼 $ST(C_{p,k}) < ST(S_{c,i})$ 인 모든 $C_{p,k}$와 $S_{c,i}$에 대해서 $A(C_{p,k}) < A(S_{c,i})$가 보장되어야 한다. 그런데 $S_{p,k}$는 무조건 $C_{p,k}$보다 시간 상 앞서므로, $A(C_{p,k}) < A(S_{c,i})$는 $A(S_{p,k}) < A(S_{c,i})$로 변환된다. 마찬가지로 $ST(S_{c,i}) < ST(C_{p,k+1})$인 모든 $S_{c,i}$와 $C_{p,k+1}$에 대해서 $A(S_{c,i}) < A(C_{p,k+1})$이 보장되어야 한다. 이때, 시뮬레이터에서 한 태스크의 수행을 시작했을 때, 수행이 종료되기 전까지는 다른 태스크가 시뮬레이션 호스트 컴퓨터의 CPU를 선점할 수 없다고 가정하면 위의 조건은 $A(S_{c,i}) < A(S_{p,k+1})$로 변환된다. 그러므로, 시뮬레이션 되는 환경에 있는 두 태스크 τ_p, τ_c에 대해서 $\tau_p \rightarrow \tau_c$일 때, $ST(C_{p,k}) < ST(S_{c,i}) < ST(C_{p,k+1})$이라면, $A(S_{p,k}) < A(S_{c,i}) < A(S_{p,k+1})$여야만 한다. 이 조건을 조건 2라 부르자. 조건 2가 만족된다면, 시뮬레이션 되는 환경에 있는 태스크들 사이에서 제대로 된 데이터가 흘러간다는 것이 보장된다. 결국, 조건 2 역시 시뮬레이터의 논리적 정확성을 위한 조건이다.

마지막으로 그림 5(c)는 시뮬레이션 되는 환경에 있는 태스크 τ_p의 출력 데이터가 실제 환경에 노출되어야 하는 경우이다. 이 경우 그림 5(a)에서와 비슷하게 예상 스케줄의 시간인 $ST(C_{p,i})$에 실제로 데이터를 내보내야 하므로, $A(C_{p,i}) = ST(C_{p,i})$의 조건이 보장되어야 한다. 이 조건을 조건 3이라 부르자. 조건 3이 만족된다면, 시뮬레이터에서 제대로 된 시간에 실제 환경으로 데이터가 나간다는 것이 보장된다. 그러므로 조건 3은 시뮬레이터의 시간적 정확성을 위한 조건이다.

지금까지 알아본 시뮬레이터의 실시간 정확성 보장을 위한 세 조건은 다음과 같다.

조건 1: τ_c가 시뮬레이션 되는 환경에 있고 실제 환경으로부터 데이터를 받는 경우, τ_c의 모든 시작 이벤트 $S_{c,i}$에 대하여 $ST(S_{c,i}) \leqq A(S_{c,i})$가 지켜져야 한다.

조건 2: $\tau_p \rightarrow \tau_c$이고 τ_p, τ_c 모두 시뮬레이션 되는 환경에 있는 경우, $ST(C_{p,k}) < ST(S_{c,i}) < ST(C_{p,k+1})$라면, $A(S_{p,k}) < A(S_{c,i}) < A(S_{p,k+1})$여야만 한다.

조건 3: τ_p가 시뮬레이션 되는 환경에 있고 τ_p의 출력이 실제 환경에 노출되어야 하는 경우, τ_p의 모든 종료 이벤트 $C_{p,i}$에 대하여 $A(C_{p,i}) \leqq ST(C_{p,i})$가 지켜져야 한다.

위의 조건들 중 조건 1이 의미하는 것은 실제 환경에서 데이터를 받는 태스크는 예상 스케줄에 명시된 시작 시간에 정확히 시작되어야 한다는 것이다. 그런데, 여기에서 가정하고 있는 시뮬레이션 호스트 컴퓨터는 단일 코어이므로, 실제 환경에서 데이터를 받는 둘 이상의 태스크가 동시에 시작하는 상황은 시뮬레이션이 불가능하다. 이 한계를 극복하기위해 시뮬레이션 되는 환경과 실제 환경에 걸쳐있는 모든 네트워크 자원에 실제 환경으로부터

표 2. 그림 4에 있는 태스크의 타이밍 정보
Table 2. Timing information of tasks on Figure 4

τ_i	P_i	C_i	ϕ_i
τ_1	20	5	0
τ_2	50	10	0
τ_3	50	2	20
τ_4	50	2	0
τ_5	80	5	30

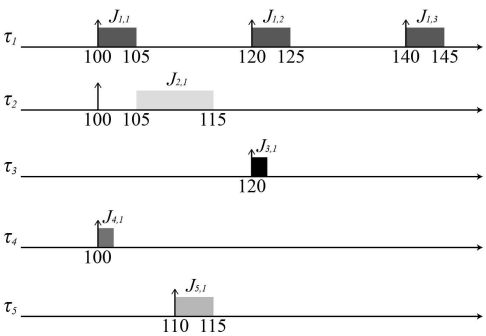


그림 6. 표 2로부터 도출된 시간 90에서 150 사이의 예상 스케줄
Fig. 6 Expected schedule derived from Table 2 at time 90 to 150

들어오는 데이터를 읽는 특수한 태스크를 둔다. 이 태스크를 읽기 데몬(Read Daemon)이라 부르자. 읽기 데몬은 읽기 동기화 포트와 쓰기 비동기화 포트를 가지며, 조건 1에 해당되는 태스크의 예상 스케줄상의 시작 시간에 깨어나서 단지 데이터를 해당 태스크에 전달하는 역할을 한다. 읽기 데몬의 수행 시간은 0에 가깝다고 가정하자. 읽기 데몬이 있다면 조건 1의 $A(S_{c,i}) = ST(S_{c,i})$는 $ST(S_{c,i}) \leqq A(S_{c,i})$로 완화될 수 있다. 왜냐하면 읽기 데몬이 실제 환경으로부터 들어오는 데이터를 처리해주므로 그 데이터를 이용하는 태스크가 예상 스케줄에 명시된 시간 이후에 수행을 시작하기만 한다면, 제대로 된 데이터가 입력으로 들어온다고 보장할 수 있다.

마찬가지로, 읽기 데몬이 있는 네트워크 자원에 쓰기 데몬(Write Daemon)을 둔다. 쓰기 데몬은 읽기 비동기화 포트와 쓰기 동기화 포트를 가지며, 조건 3에 해당되는 태스크의 예상 스케줄상의 완료 시간에 깨어나서 데이터를 실제 환경으로 전송하는 역할을 한다. 읽기 데몬과 마찬가지로 쓰기 데몬 역시 수행 시간은 0에 가깝다고 가정한다. 쓰기 데몬이 있으므로 조건 3의 $A(C_{p,i}) = ST(C_{p,i})$는 $A(C_{p,i}) \leqq ST(C_{p,i})$로 완화된다. 왜냐하면 예상 스케줄에 명시된 시간 이전에 태스크의 수행이 완료되기만 한다면 예상되는 시간에 쓰기 데몬에서 데이터를 내보낼 것이기 때문이다. 읽기 데몬과 쓰기 데몬에서 실제 환경과의 통신을 모두 담당하므로 이제 데몬들을 제외한 시뮬레이션 환경에 있는 나머지 모든 태스크는 비동기화 포트만 가지며, 정확한 데이터 전달을 위해 읽기 데몬과 쓰기 데몬만이 시뮬레이션 호스트 컴퓨터의 CPU를 선점할 수 있다.

지금까지 우리는 시뮬레이터의 실시간 정확성 보장을 위한 세 조건을 알아보았다. 이제부터는 세 조건을 만족시키면서 주어진 예상 스케줄을 단일 코어 컴퓨터에서 효율적으로 수행하기 위한 알고리즘을 설명한다. 표 2를 보자.

표 2는 그림 4의 시스템에서 각 태스크의 타이밍 정보를 표시한 것이다. 설명의 편의를 위해 그림 4의 모든 하드웨어 자원은 선점 불가능한 고정 우선순위 스케줄링 정책을 취한다고 하고 태스크의 번호가 낮으면 우선순위가 높다고 하자. 그렇다면 표 2에 의해 도출된 시간 90에서 150 사이의 예상 스케줄은 그림 6과 같이 된다. 이제 분산된 하드웨어 자원에서 도출된 예상 스케줄을 단일 코어 컴퓨터에서 모사하고자 한다.

주어진 예상 스케줄을 직렬화(Serialization)하기 위한 기본 아이디어는 단순하다. 실시간 정확성 보장을 위한 조건 2에 의하면 데이터 의존관계가 있는 태스크들은 예상 스케줄에 나타는 수행 순서대로 시뮬레이터에서 처리되어야 한다. 그런데, 이 조건에는 조건 1에서와 달리 시뮬레이터에서 태스크의 수행을 시작하는 시간에 대한 제약사항은 없다. 다시 그림 6의 예상 스케줄을 보자. 현재 시간 90에서 우리는 직렬화된 스케줄을 생성하려고 한다. 시간 90에 $J_{1,1}$은 수행될 수 없다. 왜냐하면, $J_{1,1}$은 조건 1에 의해 시간 100 이후에야 수행이 가능하기 때문이다. 하지만, $J_{4,1}$과 $J_{5,1}$은 조건 1에 해당되지도 않고, τ_4와 τ_5가 τ_1과는 아무런 데이터 의존관계가 없으므로 시간 90에도 수행이 가능하다. 결국, 예상 스케줄에 나타난 태스크의 수행 순서를 지키지 않고 수행을 해도 실시간 정확성에 문제가 없는 경우가 존재한다는 말이며, 이를 통해 기대 스케줄을 효율적으로 직렬화할 수 있게 된다.

예상 스케줄에 나타난 각 태스크들의 수행 순서를 정하기 위해 우리는 P_Count pair 표를 정의한다. 이 표의 각 행은 각 작업의 이름, P_Count

표 3. P_Count pair 표
Table 3. P_Count pair table

$J_{i,j}$	P_Count pair	$ST(S_{i,j})$	$ST(C_{i,j})$	조건 1	조건 3
$J_{1,1}$	(τ_1, 0)	100	105	O	X
$J_{1,2}$	(τ_1, 1)	120	125	O	X
$J_{1,3}$	(τ_1, 2)	140	145	O	X
$J_{2,1}$	(τ_1, 1), (τ_2, 0)	105	115	X	X
$J_{3,1}$	(τ_2, 1), (τ_3, 0)	120	122	X	O
$J_{4,1}$	(τ_4, 0), (τ_5, 0)	100	102	X	O
$J_{5,1}$	(τ_3, 0), (τ_5, 0)	110	115	X	X

pair, 예상 스케줄의 시작 시간, 예상 스케줄의 종료 시간, 조건 1 해당 여부, 조건 3 해당 여부로 구성된다. $J_{i,j}$의 P_Count pair는 τ_i를 포함하여 $\tau_k \rightarrow \tau_i$인 모든 τ_k에 대해 $J_{i,j}$보다 먼저 수행되었어야 하는 τ_k의 작업의 수를 나타낸다. 예를 들어, 시간 90에서 $J_{2,1}$의 P_Count pair는 {(τ_1, 1), (τ_2, 0)}이다.

이는 $J_{2,1}$이 수행되기 전에 τ_1이 반드시 한 번 수행되어야 한다는 의미이다. 그림 4의 데이터 의존 관계와 그림 6의 예상 스케줄로부터 우리는 표 3과 같이 P_Count pair값을 구할 수 있다.

P_Count pair 표가 추출된 이후, 이제 시뮬레이터는 어떤 작업을 먼저 수행할지 결정한다. 시뮬레이터는 P_Count pair 표를 수정해가며 다음에 수행할 작업을 선정한다. 만약 작업 $J_{i,j}$가 시뮬레이터에 의해 수행된다면, P_Count pair 표에서 작업 $J_{i,j}$를 나타내는 행은 삭제되며 남은 작업 중 P_Count pair에 τ_i를 가지고 있는 모든 작업은 P_Count pair에 있는 τ_i의 수행 횟수를 하나 줄여야 한다. 예를 들어, 표 3의 상황에서 $J_{2,1}$이 수행된다면, $J_{2,1}$이 있는 행은 삭제되며, τ_2를 P_Count pair에 가지고 있던 $J_{3,1}$의 P_Count pair는 {(τ_2, 1), (τ_3, 0)}에서 {(τ_2, 0), (τ_3, 0)}로 변경된다.

현재 시간 t에 수행할 수 있는 작업은 다음과 같은 과정으로 선택된다.

1) P_Count pair에 있는 모든 수행 횟수가 0인 작업을 선택한다.
2) 선택된 작업 $J_{i,j}$ 중 조건 1에 해당되면서 자신의 $ST(S_{i,j})$가 현재 시간 t보다 뒤인 작업을 제외한다.
3) 선택된 작업 $J_{i,j}$에 대해 자신을 제외한 다른 모든 작업의 P_Count pair에 (τ_i, 0)이 있는 작업은 제외한다.

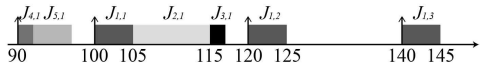


그림 7. 그림 6의 예상 스케줄을 직렬화한 수행 결과
Fig. 7 Execution result serialized from the expected schedule on Figure 6

3)의 과정은 어떤 작업 $J_{i,j}$가 수행됨으로써 아직 수행되지 않은 다른 작업의 P_Count pair에 (τ_i, -1)이 나타나는 것을 막기 위한 것이다. P_Count pair에 -1이 나타났다면, 그것은 예상 스케줄에서 의도되었던 태스크 수행 순서가 제대로 지켜지지 않았다는 것이다. 위의 과정에서 1)과 3)은 실시간 정확성 보장을 위한 조건 2를 만족시키며 2)는 조건 1을 만족시킨다.

위의 과정 1), 2), 3)에 의해 선정된 작업을 수행 후보라고 부르자. 현재 시간 t에 둘 이상의 수행 후보가 있을 때, 그 중 어떤 작업을 먼저 수행해야 할지 결정하기 위하여 여기에서는 다음의 과정을 제안한다. 이때, 앞서 소개한 읽기 데몬과 쓰기 데몬은 시뮬레이터에서 어떤 작업이 수행되고 있다고 하더라도 정확한 시간에 CPU를 선점하여 동작한다고 가정한다.

1) 수행 후보에 있는 작업 $J_{i,j}$가 쓰기 동기화 태스크인 경우 유효 데드라인을 $ST(C_{i,j})$로 한다.
2) 수행 후보에 있는 작업 $J_{i,j}$가 쓰기 동기화 태스크가 아닌 경우 유효 데드라인을 무한대로 한다.
3) 유효 데드라인이 먼저인 작업을 시뮬레이터에서 수행하며, 유효 데드라인이 같은 경우 작업 $J_{i,j}$의 $ST(S_{i,j})$가 빠른 작업을 먼저 수행한다.

수행 후보를 선별하는 과정에서 실시간 정확성 보장을 위한 조건 1과 2를 만족시켰으므로, 이제 조건 3을 만족시켜야 한다. 이를 위해, 위의 과정 1), 2), 3)에서는 쓰기 동기화 태스크를 우선하여 EDF (Earliest Deadline First) 정책으로 각 작업을 수행하는 휴리스틱(Heuristic)을 제안한다. 이는 직관적으로 실시간 정확성 보장을 위한 조건 3을 가급적 만족시키려고 한다는 것이다. 만일 시뮬레이터에서 조건 3에 해당되는 작업 $J_{i,j}$의 수행을 마쳤을 때의 실제 시간이 t일 때 $t > ST(C_{i,j})$라면 주어진 목적 시스템은 실시간 정확성을 보장하면서 시뮬레이션을 할 수 없다는 것을 의미한다. 설명의 편의를 위해 모든 하드웨어의 컴퓨팅 성능이 시뮬레이션

호스트 컴퓨터와 같다고 가정하면, 그림 6의 예상 스케줄을 위에서 설명한 과정을 거쳐 단일 코어 컴퓨터에서 수행한 결과는 그림 7과 같다. 이를 통해, 시간 90에서 150까지는 실시간 정확성을 보장하면서 시뮬레이션하는 것이 가능하다는 것을 알 수 있다.

Ⅳ. 관련 연구

시스템 개발에서 검증 및 확인 과정에서 사용되는 시뮬레이터 및 시뮬레이션 기법은 다양하다. 우선 Matlab/Simulink[4]는 여러 분야에서 매우 널리 사용되는 시뮬레이터로 모델 기반 개발 과정을 지원한다. 하지만 이 도구는 실제 하드웨어 자원과의 연동을 지원하지 않는다. 또한, 검증하고자 하는 태스크들의 수행 시간을 전혀 반영하지 못하므로 시스템의 시간적 정확성을 검증할 수 없다. [5]에서는 Matlab/Simulink의 커널 함수를 확장함으로써 이러한 문제를 극복하고자 시도하였지만, 실제 구현을 다루지는 않는다.

시뮬레이터가 실제 하드웨어와 데이터를 주고받으며 시뮬레이션을 하는 기법은 HiLS (Hardware-in-the-Loop Simulation)라 부르며 이를 지원하는 상용 시뮬레이터로는 CANoe[6], LabVIEW[7] 등이 있다. 하지만 이 도구들은 검증하고자 하는 태스크들의 타이밍 정보를 정확하게 반영하지 못하여 시스템의 시간적 정확성을 검증할 수 없다. [8]은 HiLS에서 검증된 결과와 실제 구동 단계에서 보이는 행태와의 괴리를 좁히기 위한 방법론을 다루고 있다. 하지만 이는 실제 환경의 복잡도를 시뮬레이션 과정에서 더 정확하게 반영하기 위한 연구이며, 태스크 단위의 실시간 행태나 하드웨어 자원의 이동은 전혀 고려하지 않는다.

네트워크 에뮬레이션 분야에서도 시뮬레이터에서 실제 하드웨어를 연동시키고자 하는 연구가 있다. [9]과 [10]에서는 실제 네트워크 노드와 에뮬레이션 되는 네트워크 간의 정확한 데이터 교환을 지원하는 에뮬레이션 기법을 제안함으로써 시스템의 시간적 정확성을 보장하고자 하였다. 하지만, 에뮬레이션 되는 네트워크 노드에서는 데이터 전송의 타이밍만 고려를 하고 있어서 시스템의 논리적 정확성을 검증할 수 없다.

[11]는 ns-2 [12]와 Modelica [13]를 결합함으로써 실제 액추에이터와 연동되는 시뮬레이션 플랫폼을 구성하고자 하였다. 비슷하게 [14]에서는 EPANET [15]과 Matlab을 결합하여 CPS를 위한 통합 시뮬레이션 환경을 제안하였다. 그러나 둘 모두 시뮬레이터에서 각 태스크의 수행 시간을 정확하게 반영하지 않았으며, 특히, 하드웨어 자원의 이동은 전혀 고려하지 않는다.

Ⅴ. 결 론

본고에서는 CPS를 설계하고 개발하는 과정 전반에 사용될 수 있는 실시간 시뮬레이션 프레임워크를 제안하였다. CPS와 같은 복잡한 시스템을 점진적으로 구현하면서 실제 하드웨어 자원들과 시뮬레이션 되는 시스템간의 상호작용을 통한 검증을 정확히 지원하기 위해서는, 시뮬레이터가 시뮬레이션 되는 시스템이 실제 하드웨어로 구현 되었을 때의 실시간 행태를 정확히 모사하여야 한다. 또한 하드웨어 자원이 시뮬레이션 되는 환경에서 실제 환경으로 이동할 때, 시스템 개발자의 추가적인 작업 없이 시뮬레이터에서 이를 지원해야 한다. 이를 위해 본고에서는 CPS의 점진적인 개발 과정을 지원하기 위한 실시간 시뮬레이터의 구조를 기술하고, 시스템의 실시간 정확성을 보장하는 효율적인 이벤트 스케줄링 알고리즘을 제안하였다.

본 연구의 후속 연구로 본고에서 제안된 방법론을 실제에 적용하여 실시간 시뮬레이션을 얼마나 효율적으로 수행할 수 있는지에 대한 실험적 검증을 진행 중에 있다. 더 나아가, 제안된 이벤트 스케줄링 알고리즘을 단일 코어 컴퓨터뿐만 아니라, 다중 코어(Multi-Core) 컴퓨터로 확장하여 더욱 효율적으로 실시간 시뮬레이터의 정확성을 보장하는 방법론을 연구하고 있다. 이러한 연구들을 통해 본고에서 제안된 실시간 시뮬레이션 프레임워크가 항공, 자동차와 같은 복잡한 CPS를 다루는 실제 산업에서 유용하게 사용될 수 있을 것이라 기대한다.

참 고 문 헌

[1] E. Lee, "Cyber Physical Systems: Design Challenges," Proceedings on the IEEE International Symposium on Object Oriented Real-Time Distributed Computing, pp.363-369, 2008.

[2] K.S. We, J.C. Kim, C.G. Lee, "A Novel Simulation Framework for Supporting Real-Time Cyber-Physical Interactions,"

Proceedings on the International Workshop on Large-Scale Cyber-Physical Systems, pp.1-3, 2011.
[3] J.H. Han, K.S. We, C.G. Lee, "WiP Abstract: Cyber Physical Simulation Supporting Smooth Development from All-simulated Systems to All-real Systems," Proceedings on the International Conference on Cyber-Physical Systems, pp.208, 2012.
[4] Simulink, http://www.mathworks.co.kr/products/simulink/
[5] D. Henriksson, A. Cervin, K. Arzen, "TRUETIME: Real-Time Control System Simulation with Matlab/Simulink," Proceedings on Nordic MATLAB Conference, 2003.
[6] CANoe, http://www.vector.com/vk_canoe_ko.html/
[7] LabVIEW, http://ni.com/labview/ko/
[8] C. Faure, M.B. Gaid, N. Pernet, M. Fremovici, G. Font, G. Corde, "Methods for real-time simulation of Cyber-Physical Systems: application to automotive domain," Proceedings on the International Wireless Communication and Mobile Computing Conference, pp.1105-1110, 2011.
[9] D. Mahrenholz, S. Ivanov, "Real-Time Network Emulation with ns-2," Proceedings on the IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, pp.29-36, 2004.
[10] J. Zhou, Z. Ji, M. Takai, R. Bagrodia, "Maya: a Multi-Paradigm Network Modeling Framework for Emulating Distributed Applications," Proceedings on the Workshop on Parallel and distributed simulation, pp.163-170, 2003.
[11] A. Al-Hammouri, V. Liberatore, H. Al-Omari, Z. Al-Qudah, M. S. Branicky, D. Agrawal, "A co-simulation platform for actuator networks," Proceedings on the International Conference on Embedded Networked Sensor Systems, pp.383-384, 2007.
[12] ns-2, http://isi.edu/nsnam/ns/
[13] M.M. Tiller, "Introduction to Physical Modeling with Modelica," Springer, 2001.
[14] J. Lin, S. Sedigh, A. Miller, "Towards Integrated Simulation of Cyber-Physical Systems: A Case Study on Intelligent Water Distribution," Proceedings on the IEEE International Conference on Dependable, Autonomics and Secure Computing, pp.690-695, 2009.
[15] EPANET, http://www.epa.gov/nrmrl/wswrd/dw/epanet.html/

저 자 소 개

한 재 화

2010년 서울대학교 컴퓨터공학부 학사.
현재, 서울대학교 컴퓨터공학부 석사과정.
관심분야: 실시간 시뮬레이션, 모바일 컴퓨팅.
Email: jhhan@rubis.snu.ac.kr

위 경 수

2009년 서울대학교 컴퓨터공학부 학사.
2011년 서울대학교 컴퓨터공학부 석사.
현재, 서울대학교 컴퓨터공학부 박사과정.
관심분야: 지능형 자동차, Cyber-Physical Systems.
Email: kswe@rubis.snu.ac.kr

이 창 건

1991년 서울대학교 컴퓨터공학부 학사.
1993년 서울대학교 컴퓨터공학부 석사.
1998년 서울대학교 컴퓨터공학부 박사.
현재, 서울대학교 컴퓨터공학부 부교수.
관심분야: 실시간 시스템, Cyber-Physical Systems, Wireless Sensor Networks.
Email: cglee@snu.ac.kr