Journal of the Korean Institute of Industrial Engineers
Vol. 41, No. 2, pp. 173-184, April 2015.
ISSN 1225-0988 | EISSN 2234-6457
http://dx.doi.org/10.7232/JKIIE.2015.41.2.173

<Application Research>

콜드체인 시스템의 센서태그 운영 최적화를 위한 DEVS 기반 시뮬레이션 모델

류옥현[1] · 강용신[2] · 진희주[3] · 이용한[3†]

[1]한국산업기술대학교 경영학부/ [2]동국대학교서울 나노정보과학기술원/ [3]동국대학교서울 산업시스템공학과

DEVS-Based Simulation Model for Optimization of Sensor-Tag Operations in Cold Chain Systems

Okhyun Ryou[1] · Yong-Shin Kang[2] · Heeju Jin[3] · Yong-Han Lee[3]

[1]Dept. of Business Administration, Korea Polytechnic University
[2]Nano Information Technology Academy, Dongguk University-Seoul
[3]Dept. of Industrial and Systems Engineering, Dongguk University-Seoul

The application of radio frequency identification (RFID) sensor-tags in cold chain systems has recently received a great deal of attention. To design cold chain systems with RFID sensor-tags that minimize the initial investment and operational cost while fulfilling the functional and operational requirements, simulation study is one of the preferable and effective approaches. To simulate the possible design configurations, the individual components in a cold chain system can be extracted and implemented as a DEVS (Discrete Event System Specification) model. Based on the proposed DEVS model, a new cold chain simulation model can be efficiently created by simply connecting each DEVS model around the RFID sensor-tag of interest in sequence according to the structure of the cold chain system, and then executed (or simulated) on Java programming environments by the DEVSJAVA simulator. As a result of simulation, some key performance indexes such as reliability, accuracy or timeliness can be calculated and used to choose better components or to compare different system configurations of cold chain systems.

Keywords: Cold chain system, RFID, Simulation, DEVS

1. 서 론

부패하는 식품의 안전과 가격 손실에 가장 영향을 주는 환경요인은 온도와 시간이다. 공장이나 창고와 같이 관리되는 환경에서는 온도의 모니터링이 용이할 수 있으나 수송 구간에서는 실시간성과 정교함이 떨어지게 마련이다. 특히 차량에서의 상/하차 작업은 온도 모니터링 및 제어 측면에서 취약부분이다. 이렇게 온도에 영향을 많이 받는 상품들이 생산, 저장, 운송, 배포, 판매, 소비 단계에서 항상 적정온도로 유지할 수 있도록 하는 프로세스, 장비, 정보 관리로 구성된 체계를 콜드체인(cold chain) 시스템이라고 한다. 콜드체인은 일반적인 공급망보다 요구사항이 더 많으며, 특히 물류의 모든 단계에서 정교한 모니터링 및 컨트롤이 필요하다.

최근 콜드체인 시스템을 생산적으로 운영하기 위하여 RFID와 센서 기술이 결합된 RFID 센서태그를 사용하는 시도가 늘고 있다(Pereira, 2008; Catarinucci, 2009). RFID 센서태그를 활용하여 상품을 추적하는 콜드체인 시스템의 구축에는 고비용의 투자가 필요하다. 콜드체인 시스템 구축에 필요한 하드웨어 및

본 연구는 농림축산식품부 첨단기술개발사업(과제번호: 312068-03) 지원으로 이루어졌음.
† 연락저자 : 이용한 교수, 100-715 서울특별시 중구 필동로 1가 30 동국대학교 산업시스템공학과, Tel : 02-2260-3887, Fax : 02-2269-2212, E-mail : yonghan@dgu.edu
2014년 9월 12일 접수; 2015년 1월 9일 수정본 접수; 2015년 3월 20일 게재 확정.

소프트웨어를 적절히 구매/배치하고, 콜드체인의 요구(예를 들어 : 에너지 효율, 적시성, 신뢰성, 정합성 등)를 만족시키면서, 오래 사용할 수 있는 운영 모델을 설정하여, 전체 시스템의 수명 시간을 연장시키는 시스템 설계는 매우 중요하다(Aiello, 2011). 이와 같은 목적을 달성하기 위해서 시뮬레이션 기법은 매우 효과적으로 사용될 수 있다. 그러나 현재 RFID 센서태그 기반의 콜드체인 시스템 구축은 도입 단계이고, 이를 위해 개발된 전용 시뮬레이터가 존재하지 않는다. 물론 범용의 시뮬레이션 패키지(Kelton, 2007; Sigam, 2011)를 사용하는 것도 가능하지만, 범용 도구에 사용된 시뮬레이션의 관점, 복잡한 구성요소 및 사전 지식을 습득하는 것도 큰 부담일 수 있다. 따라서 기업이 RFID 센서태그 기반의 콜드체인 시스템을 설계할 때, 비용 및 성능과 관련된 구성요소들을 사전에 자유롭게 선택/적용하고, 콜드체인의 요구사항에 비추어 적합한 구성을 실험 해 볼 수 있는 전용 시뮬레이션 도구는 가 필요하다.

이에 본 연구에서는 콜드체인 시스템의 핵심 기술인 센서태그의 효과적인 운영전략을 분석하고 평가하기 위하여, 즉 이산사건(discrete even) 시스템을 기술하는 형식 언어인 DEVS (Discrete Event System Specification) 형식론을 활용하는 모델링 및 시뮬레이션 방법을 제시한다. 본 연구의 결과를 활용하여 직접 시뮬레이션 연구수행 결과는 Kang *et al*.(2012)에서 제시되었다.

본 연구의 구성은 다음과 같다. 먼저 제 2장에서 DEVS 형식론 및 관련 분야의 시뮬레이션 연구에 대해서 간략히 설명한다. 제 3장에서는 대상이 되는 콜드체인 시스템의 개요를 파악하고 구조적 기능적 구성을 분석한다. 제 4장에서는 시뮬레이션을 위한 구성요소 및 관점을 결정하고, 이를 DEVS 형식으로 표현하는 방안과 Java 클래스로 구현하는 방안에 대해서 설명한다. 제 5장에서는 간단한 콜드체인 시스템 예제를 대상으로 모델링 및 시뮬레이션을 수행하는 절차를 설명하고, 획득 가능한 결과를 살펴본다. 마지막으로 제 6장에서 향후 연구방향을 포함한 결론을 제시하였다.

2. 관련 연구

2.1 DEVS 형식론

(1) DEVS 모델-Atomic and Coupled

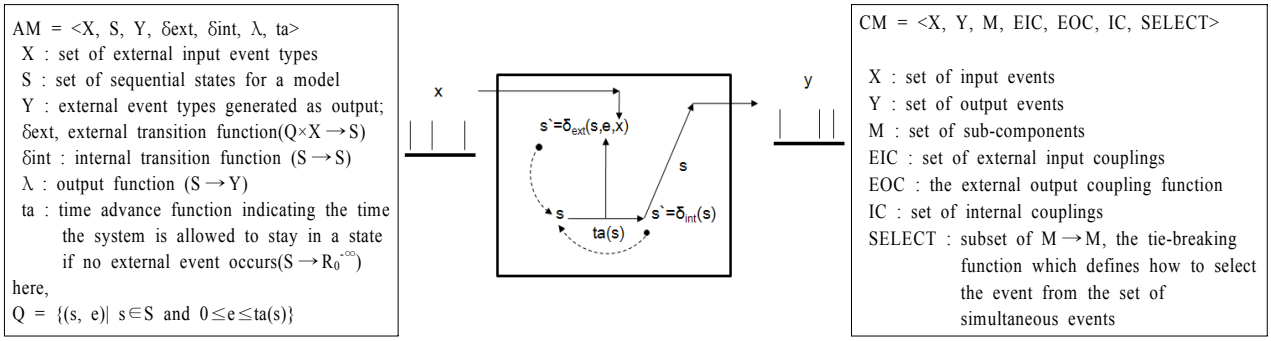
Zeigler(2000)에 의해서 제안된 Discrete Event System Specification(DEVS)은 이산사건 시스템을 계층적이고 모듈화된 형태로 기술하는 수학적인 언어로서, 콜드체인 시스템의 모델링 및 시뮬레이션에 적용할 수 있는 방법론이다. 이산사건 모델링 및 시뮬레이션에 있어서 운영 프로세스의 내부 메커니즘은 프로세스를 진전시키는 사건(event)의 발생을 기반으로 한다(North and Macal, 2007). 콜드체인 시스템은 물류 운영 프로세스를 가지고 있고, 입고, 포장, 출고 등과 같은 물류활동의 실행에 따라 그 상태가 변화한다. 개별 물류활동을 사건으로 간주하면, 콜드체인 시스템은 이산사건 시스템에 속한다. 물류 프로세스뿐만 아니라, DEVS 형식론은 컴퓨터/네크워크 시스템, 공장의 프로세스 제어 등을 포함한 다양한 분야에서 적용된다(Farooq *et al*., 2006; Pujo *et al*., 2006; Zacharewicz *et al*., 2011; Bae *et al*., 2013).

DEVS에서 개별 구성요소들은 atomic 모델로, 계층적인 구성은 coupled 모델로 표현한다. DEVS 형식론은 대상 시스템을 모듈화된 가장 작은 구성요소로 나누고, 이들 구성요소의 동작과 구조를 기술하고, 이 구성요소들을 계층적으로 연결함으로써 보다 복잡한 시스템을 모델링한다. 여기서 가장 기본이 되는 작은 구성요소를 atomic 모델이라고 한다. 이 atomic 모델은 <Figure 1>(a)에 제시하였다. atomic 모델을 구성하는 7개 요소 중 첫 3개 요소는 시스템의 입력, 상태 그리고 출력 집합이다. 그리고 다음 4개의 요소는 앞의 3개 요소의 제약 사항을 규정한다. 또, $R_0^{-\infty}$은 0을 포함한 양의 실수의 집합이고 Q는 atomic 모델 AM의 전체 상태의 집합으로 표현된다. 두 개의 상태전이 함수를 가지는데, 내부 상태전이 함수(δ_{int})는 시간이 진행됨에 따라 일어나는 상태의 변화를 나타내기 위한 함수이고, 외부 상태전이 함수(δ_{ext})는 어떤 상태에서 외부로부터 입력을 받아서 그 상태가 바뀌는 것을 표현하는 함수이다. 출력 함수(λ)는 내부 상태전이가 일어날 때 수행되어, 외부 출력을 발생하는 함수이다. 이때 발생하는 출력은 다른 모델들의 입력이 되므로, atomic 모델의 내부 상태전이는 연결된 다른 모델의 외부 상태전이를 야기하게 된다. 외부 상태전이에서는 출력이 발생되지 않으므로 다른 모델들에게 영향을 주지 않는다. <Figure 1>(b)는 기 설명한 DEVS 모델의 동적 상태변화 메카니즘을 도식을 이용하여 설명하고 있다.

Atomic 모델은 행위를 나타내는 반면, 계층적인 구조를 나타내는 coupled 모델은 복합형 구성요소로서 atomic 모델들이나 다른 coupled 모델들로 구성되고, 큰 규모의 coupled 모델의 구성요소가 될 수 있다. DEVS 형식에서는 coupled 모델을 이용해서 계층적인 구조를 갖는 복잡한 모델을 쉽게 구성할 수 있게 된다. Coupled 모델의 정의는 <Figure 1>(c)와 같다.

(2) DEVS 시뮬레이터

DEVS 형식론에서는 DEVS로 모델링된 시스템의 시뮬레이션을 위하여 추상적 시뮬레이터의 알고리즘을 제공한다. 이와 같은 시뮬레이터의 개념을 Java 기반에서 구현한 것이 DEVSJAVA (ACIMS, 2010; Sarjoughian, 1998)이다. DEVSJAVA는 DEVS 형식론을 기반으로 하는 모델링 및 시뮬레이션 환경으로, Java 프로그래밍 언어를 사용하여 개발되었다. DEVS 형식론 기반 시뮬레이션을 가능하게 하는 모델링(atomic 모델의 정의, 관계 연결) 및 시뮬레이션 실행(시뮬레이션 엔진)에 필요한 API(Application Program Interface)를 제공한다. DEVS 모델의 기본 특성을 갖는 프로그래밍 클래스가 포함되어 있으며, 이들 클래스는 객체지향언어인 Java의 특성인 상속(inheritance)에 따라서 확장(extend) 할 수 있다. 모델의 작성은 기존의 DEVS API



(a) Definition of atomic model (b) State transition of atomic model (c) Definition of coupled model

Figure 1. DEVS atomic model(Zeigler, 2000)

에 정의된 클래스의 객체를 정의하고, 이들을 연결(coupled 모델 작성)하는 소스코드를 작성하고, Eclipse(2011) 등 JAVA 프로그램 개발 환경을 통해서 컴파일하고 실행할 수 있다.

DEVSJAVA의 모델링 과정을 지원하기 위한 툴도 개발되었는데, 이 가운데 DEVS-Suite(Sarjoughian, 2010; Kim, 2009)는 DEVSJAVA가 가지고 있는 시뮬레이터를 확장한 시뮬레이터다. 기존의 상용 시뮬레이션의 애니메이션(모델 수행시의 시간의 변화에 따른 대상 시스템의 동적인 변화를 시각적으로 제공하는 그래픽 소프트웨어) 기능을 간략하게 제공한다. DEVS-Suite는 DEVSJAVA 환경에서 작성한 모델(소스코드)을 실시간 컴파일하고 시간변화에 따른 모델의 상태 변화를 시각적으로 보여준다.

3. 콜드체인 시스템 분석

3.1 콜드체인 시스템의 구성

RFID 센서태그를 활용한 식품의 콜드체인의 구성을 간단하게 <Figure 2>에 나타내었다. 낙농, 소고기 유통, 과일/채소, 약품 등에서도 유사한 물류망의 구성이 가능하다. 작업 흐름의 관점에서 보면, RFID 센서태그를 대상에 부착(팔렛 또는 포장단위)한 후, 운송을 위한 상차, 운송, 하차, 보관, 재포장을 위한 분배 등의 작업이 수행된다. RFID 센서태그의 작동 방식은 일반적으로 사용자에 의해 선택되기 보다는 RFID 센서태그 공급자에 의해 최소한의 작동 방식을 설정할 수 있는 범위에서 결정한다. RFID 리더가 설치된 곳을 센서태그가 통과하는 경우 센서태그에 저장된 시간-온도 데이터는 리더의 요구에 의해서 정보시스템으로 전달된다. 리더에서 정보시스템으로의 데이터 전송은 설치 환경에 따라서 유선/무선일 수도 있으며, WIFI, LAN, CDMA 망이 일반적으로 사용된다.

3.2 정보 처리 구성요소

센서태그에서 수집된 정보의 전달과 관련한 구성요소들과 정보전달 방식에 대한 전체적인 구조는 <Figure 3>에 표현된 바와 같다. 여기서는 RFID 센서태그와 리더 그리고 유무선 전송 장치를 이용하여 센서 데이터를 수집하고 정보시스템에 전송하는 모델을 설명한다. 센서는 온도 측정 데이터를 RFID 센서태그의 유저메모리(user memory)에 저장한다. 이후 리더-미들웨어는 RFID 센서태그의 ID(예 : Electronic Product Code : EPC)(Traub *et al.*, 2005)와 유저메모리에 저장된 센서 데이터를 읽어 정보시스템에 전달한다. 리더-미들웨어는 기능적으로는 RFID 센서태그의 유저메모리에 저장된 센서 데이터를 수집하고 필터링하여 정보시스템에 전달하는 하나의 요소로 볼 수

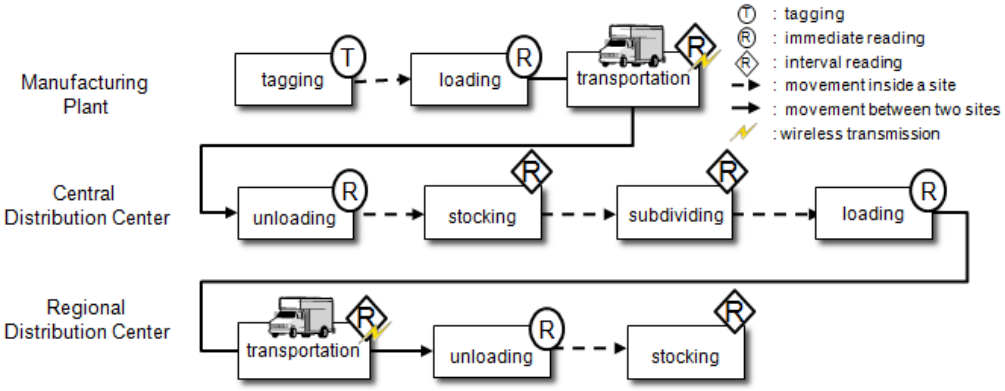


Figure 2. Typical cold chain system architecture

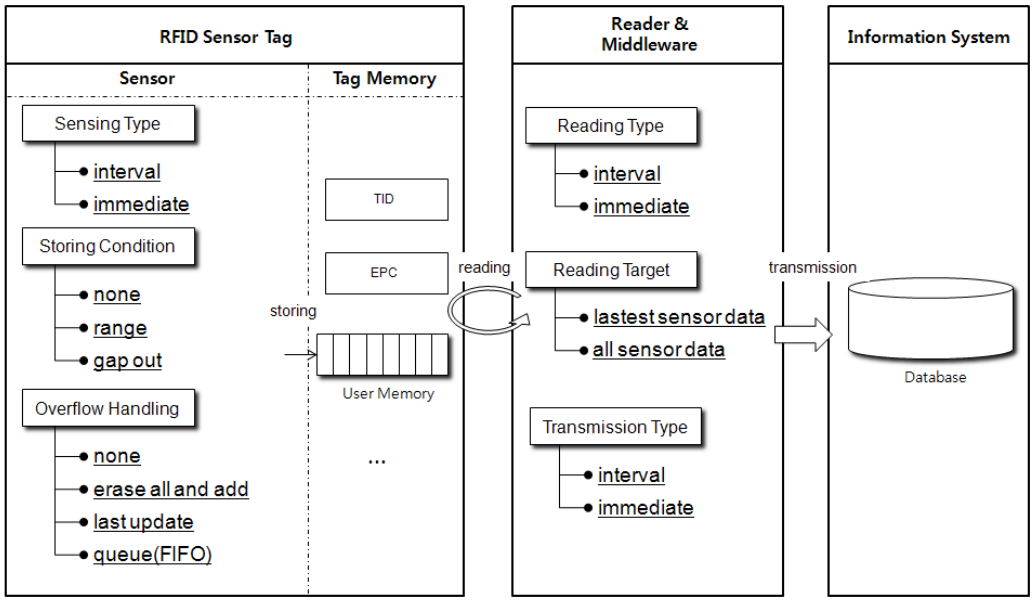


Figure 3. Cold chain system data collection and storage model

있다. 수송구간에서는 CDMA와 같은 무선 통신 장비가 차량에 구축된 경우 정보시스템에 직접 전달할 수 있다고 가정한다.

(1) RFID 센서태그

<Figure 3>의 왼쪽 상자에, 현재 가용한 RFID 센서태그의 기능 및 옵션을 정리하였는데, RFID 센서태그는 온도측정 방식, 저장 방식, 오버플로 처리 방식에 따라서 다음과 같이 분류하였다.

- **센싱 방법(Sensing Type)** : 센서의 온도 측정 방식은 크게 1) 주기적으로 인식하거나 2)이벤트(사용자나 시스템의 요청)에 의해 일시적으로 한 번 온도를 측정하는 두 가지로 분류할 수 있다.
- **저장 조건(Storing Condition)** : 센서가 태그 메모리에 센서 데이터를 저장하는 조건은 조건 없이 측정된 모든 센서 데이터를 태그에 저장하는 1) none 방식을 기본으로, 허용 범위를 결정해놓고 그 기준을 이탈(out of range)하는 시점에 저장하는 2) range 방식과 바로 이전에 측정된 데이터 값과 최근 측정 데이터 값의 차이가 정해진 기준보다 클 경우 저장하는 3) gap-out 방식의 총 세 가지로 분류할 수 있다.
- **오버플로 처리(Overflow Handling)** : 센서가 센서 데이터를 유저메모리에 저장할 때, 메모리 오버플로가 발생할 경우 적절히 처리하는 방법이 필요하다. 오버플로 처리는 총 네 가지 방식으로 분류하는 것이 가능하다. 첫 번째는 측정 온도 데이터를 센서태그에 저장하지 않는 방식 즉, 오버플로 처리를 하지 않는 1) none 방식, 두 번째는 마지막 데이터를 지우고 추가하는 2) last data update 방식, 세 번째는 가장 오래된 데이터를 지우고 마지막에 추가하는 3) queue 방식, 마지막으로 모든 데이터를 지우고 처음부터 추가하는 4) erase all and add 방식이다.

(2) 리더-미들웨어(Reader and Middleware)

RFID 리더가 센서태그로부터 ID와 센서 데이터를 수집하는 방법은 1) interval 방식과 요청에 의해 전달받는 2) immediate 방식으로 구분할 수 있다. Immediate 방식의 경우 입고, 출고 등의 물류 이벤트가 발생할 때 실행될 수 있다. RFID 리더는 태그를 인식할 때 기본적으로 ID를 인식하고, 유저메모리에 저장된 센서 데이터를 선택적으로 인식할 수 있다. 이때 1) 가장 최근의 센서 데이터 한 개만 읽어오는 방식 2) 모든 센서 데이터를 읽어오는 방식 등 두 가지로 나뉠 수 있다. 가장 최근 센서 데이터 한 개만 읽어오는 경우는 리더-미들웨어가 장소에 구애 받지 않고 항상 정보시스템에 RFID와 센서 데이터를 전달할 수 있는 시스템 환경에서 사용될 수 있으며, 모든 센서 데이터를 읽어오는 방법은 수송 차량에 무선 전송장치가 설치되지 않아 수송이 완료된 후 수송 중 발생된 센서 데이터를 수집하는데 사용될 수 있다. 또한 수송 구간에서도 정보시스템에 데이터를 전달할 수 있는 시스템 환경이지만, 장비 오류로 인해 전송이 안 될 경우를 고려하여 수송이 완료된 후 수송 중에 저장된 센서 데이터를 한 번에 인식하여 전송이 안 된 데이터를 보상하는 즉, 데이터 로깅 차원에서 사용될 수도 있다.

리더-미들웨어가 정보시스템에 전송하는 유형은 1) 일정 주기로 전송하는 방식(interval)과 2) 허용 범위를 벗어나게 될 경우 즉시 전송하는 방식(immediate) 두 가지로 나뉠 수 있다. 이 두 가지 방식은 혼용될 수도 있는 바, RFID 센서태그로부터 읽어온 센서 데이터를 허용 온도 범위와 비교한 후 정상으로 판단되면 계속해서 초기에 설정된 주기적인 방식으로 전송하다가, 허용 온도를 벗어나게 되는 경우, 즉시 정보시스템 전달하고 향후 전송 주기를 줄이는 탄력적 주기 조절 방법을 사용할 수 있다.

4. 콜드체인 시스템의 시뮬레이션 모델링

4.1 시뮬레이션의 관점-센서태그 중심의 모델링

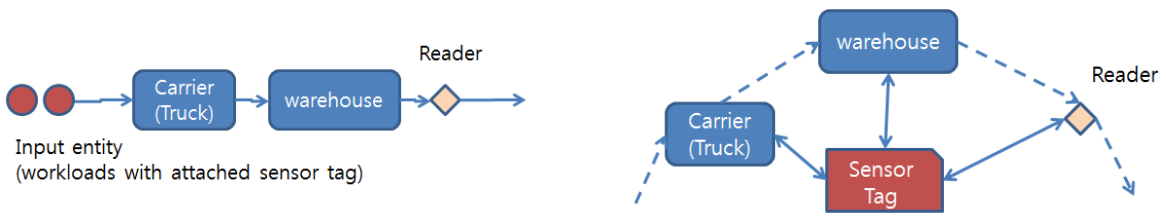
시뮬레이션이 활발하게 사용하는 제조시스템 및 공급망의 설계/성능향상에 사용하는 경우를 중심으로, 전통적인 시뮬레이션 이론 또는 상용 시뮬레이션 소프트웨어가 구성 요소들을 모델링하는 방법의 일부를 살펴보고자 한다. 대상이 되는 시스템을 구성하는 요소는 영구적/한시적(permanent/temporary entity) 두 가지 종류로 분류 할 수 있다. 장비, 기계, 운송수단과 같이 시뮬레이션 연구 기간 전체에 존재하는 요소를 영구적 요소로 모델링 하고, 작업물 또는 정보와 같이 시뮬레이션 모델 내부를 이동하면서 상태를 변화시키고, 다른 개체 및 시스템의 상태에 의해 영향을 주고받으며, 출력 수행 척도에 영향을 미치고 소멸되는 요소를 한시적 요소로 모델링한다(Kelton, 2007; Carrie, 1988). 후자는 시뮬레이션 내의 동적인 대상물로서, 생성되어 얼마간 시스템 내부를 돌아다니다가 시스템을 떠나면서 소멸된다. 가공할 부품, 운반할 제품 등이 주로 한시적 요소로 모델링 된다. 이런 관점에서 콜드체인을 살펴보면 센서태그를 부착한 제품은 한시적 요소로 모델링하고, 영구적 설비(창고, 운송트럭, RFID Reader)들을 통과하여 이동하는 것으로 모델링하는 것이 가능하다. 이와 같은 모델링은 기본적으로 개별 설비의 운용 방법과 대기행렬을 분석하고, 공급망 전체의 성과측정에 대한 측정을 용이하게 해줄 수 있는 전통적인 방법이다.

그러나 센서태그를 사용하는 콜드체인의 시뮬레이션 연구의 관심은 한시적 요소로서 센서태그를 처리하기에는 비효율적인 여러 측면을 가진다. 센서태그를 운송물(식품의 용기 등)에 부착한 개체(한시적인 요소)로 본다면 몇 가지 고려하고 극복해야 할 사항이 있다. 첫째, 경우에 따라서 과다한 수의 개체를 발생시켜야 한다. 이는 현재와 같이 컴퓨팅 리소스가 충분하고 제한적인 소규모 시스템의 시뮬레이션의 경우에는 큰 문제가 되지 않을 수 있으나, 개별 개체에 센서태그가 부착된 대규모의 실제 콜드체인 시스템의 경우에서 확장성(Scalability)을 담보 할 수 있을 만큼 컴퓨팅 리소스가 충분하다고 확신할 수는 없다. 둘째, 콜드체인 안에서의 각 시설(창고, 운송 트럭, 기타 설비)을 별도로 모델링해야 하고, 개별 시설의 운영 방식에 대해서도 심각하게 고민하고, 모델에 반영하여야 한다. 즉 작업 대상이 되는 개체에 대한 각각의 처리 방식이 고민되어야 한다. 즉 영구적 요소의 모델링에 상당한 부담이 존재한다. 셋째, 한시적인 개체로 센서태그를 간단히 모델링 하는 것은 나름의 운용 방식(Parameter, Specification, Internal Operations)을 갖고 있는 센서태그의 복잡성에 비추어 쉽지 않다.

본 연구에서는 이러한 문제를 해소하기 위하여 센서태그를 영구적인 요소로 보고, 이를 중심으로 주변 환경 또는 작업이 번갈아 활성화되어 센서태그와 상호작용하는 방식으로 대상 시스템을 해석하였다. <Figure 4>에 표현된 바와 같이, 센서태그가 물류 네트워크를 정해진 방식에 따라 이동시키고, 작업의 대상이 되는 것이 아니라, 센서태그를 중심으로 주변 시설이 순서에 따라 상호 작용하는 개념이다. 고전적인 천동설의 관점과 유사하게, 직관적으로 RFID 센서태그를 중심으로 발생되는 상황들을 관찰하는 것이 센서태그 기반의 콜드체인 분석 연구에 적합하다고 판단되었기 때문이다. 이와 같은 방식을 통하여 보다 세밀한 센서태그의 모델링과 관찰이 가능할 수 있고, 시설 및 장비를 센서태그와의 교류형식 및 내용 관점에서 파악하여 단순화하고, 유사요소를 통합화할 수 있다.

4.2 구성 요소의 추출

RFID 센서태그는 운송, 저장 또는 작업의 대상으로서 여정을 거치면서 정해진 방법에 따라 리더-미들웨어를 통해서 외부 정보시스템으로 인식정보를 보내게 된다. 센서태그의 관점에서 보면 콜드체인의 각 단계별 환경으로부터 온도 정보 측정하여 리더-미들웨어와 상호 작용하는 것이 주요 관건이다. 즉, 센서태그와 교류하는 구성요소는 1) 주변 환경(environment)과 2) RFID 리더노드(reader, antenna, middleware 포함)이다. 센서가 교류하는 시설(창고, 운송트럭, 작업장 등 모든 것)이 무엇이든 간에 센서태그의 입장에서 보면 단지 변화된 환경정보를 제공해주는 역할만을 수행한다. 한편 RFID 리더노드는 센서태그로부터 정해진 방법에 따라서 정보를 읽어오는 역할을 한다. RFID 리더노드의 뒷단에는 정보를 저장하고 처리하는 3) 정보 저장소(repository) 또는 정보시스템이 필요하다. 정보시스템에 전달된 실시간 정보가 결국은 센서태그가 실제 상황을 적절하게 검출 했는지를 판단하는 기준이 된다. 마지막으로 이 연구의 중심이 되는 4) RFID 센서태그는 가장 중요한 구성 요소이다.



(a) entity flow based simulation model (b) censor tag based simulation model

Figure 4. Two different entity modeling in cold chain system simulation modeling

4.3 DEVS Atomic 모델

제 4.2절에서 언급한 요소들을 DEVS의 atomic 모델 관점에서 보면, <Table 1>에 정리된 바와 같이 다섯 개의 atomic 모델만으로 콜드체인을 모델링하고 시뮬레이션 할 수 있다. 정의된 atomic 모델은 콜드체인의 기본적인 빌딩 블록 역할을 하며, 빌딩 블록의 조합을 통해서 다양한 구성의 콜드체인 시스템을 모델링 할 수 있다. 전술한 네 개의 구성요소에 시뮬레이션의 시작과 종료를 위한 논리적인 요소(End_Node)를 더하여 다섯 개의 구성요소를 <Table 1>에 제시하였다. <Table 1>의 세 번째 열(Atomic model)에서 DEVS atomic 모델의 in/out 포트의 수와 이름을 확인 할 수 있다.

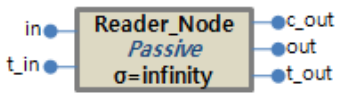
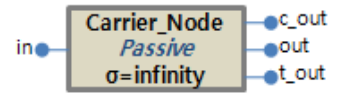
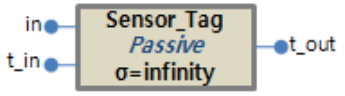
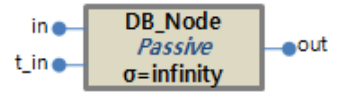
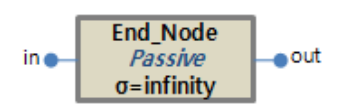
각 atomic 모델은 DEVSJAVA에서 기본 DEVS 모델로 제공되는 Atomic 클래스를 확장(extends)하여 개발하였다. 실제로 Java 프로그래밍 언어의 통합 개발환경인 Eclipse에서 DEVSJAVA API를 이용하여 제공된 Atomic 클래스를 상속하여 <Table 1>과 같은 다섯 개 클래스를 정의 하였다.

(1) Sensor_Tag

기술한 atomic 모델 중에서 가장 핵심적인 역할을 하는 Sensor_Tag 모델은, 온도 센서와 RFID 태그를 함께 모델링 한 것이다. 실제 상황에서는 온도센서와 RFID Tag가 같이 있는 경우도 있고, 단지 RFID 태그가 온도 정보의 전달자로 사용되는 경우도 있으나, 물리적인 장비의 실제 형태와 무관하게 기능적으로는 통합된 단일체 형태로 볼 수 있다. 내부적으로는 한정된 크기의 온도 정보를 저장 할 수 있는 메모리를 가지고 있으며, 앞서 언급한 여러 가지 방법들을 통해 센서에서 인식한 온도 데이터를 저장하고 전달한다.

Sensor_Tag 모델의 동적 특성을 설명하면, 기본적으로 Carrier_Node로부터 전달받은 환경 정보를 저장하고 Reader_Node의 요청에 의해 내부에 저장된 온도 및 시간을 다른 시스템 구성요소에 제공한다. Sensor_Tag는 두 개의 입력 포트(in, t_in)와 한 개의 출력 포트(t_out)가 필요하다. 'in' 포트는 모델의 활성화를 위한 'start' 또는 'stop' 신호를 End_Node로부터 받거나, Reader_Node로부터 온도 환경 정보 요청 신호를 받는데 사용된다. 또한 't_in' 포트는 Carrier_Node로 부터 변화된 온도를 전달받는데 사용된다. 't_out' 포트는 Sensor_Tag가 내부에 저장한 환경 정보(시간, 온도)를 Reader_Node로 내보낸다. DEVS 형식론에 입각하여 Sensor_Tag는 복수의 상태를 가질 수 있으며, 내부의 활동을 규정하는 복수의 매개변수(parameter), 내/외부 상태전이함수(internal/external transition)를 가질 수 있다. <Figure 5>는 Sensor_Tag가 일정시간 간격마다 외부 온도를 측정/저장할 때(When_to_Sense == INTERVAL) 동작하는 방식을 pseudo code를 통해 개략적으로 설명한 것이다. Pseudo code에는 Sensor_Tag의 상태(phase), 매개변수(parameter), 내/외부 상태전이 함수가 정의되어 있다. 기타 다양한 매개변수 값에 따른 Sensor_Tag의 행태도 유사한 방법으로 설명이 가능하나 생략하였다. 한편 <Figure 6>은 Sensor_Tag를 이벤트에 의한 상태 전이(state-transition) 관점에서 도식화 하였고, 기타 콜드체인의 구성 요소와 Sensor_Tag간의 상호작용(사건의 전달 : 입출력 포트를 통한 정보의 전달)도 간략하게 제시하였다.

Table 1. Five cold chain system atomic models

DEVS Model (component)	Descriptions	Atomic model (Icon)
Reader_Node (Reader/ Middleware)	An atomic model corresponding to RFID Readers and Middleware. This model requests and receives data such as temperature and temperature recorded time(timestamp) from Sensor_Tag_Node. It also delivers the received data to Database_Node	
Carrier_Node (environment)	An atomic model for any storage or transportation facility where the sensor-tag is stationed temporarily. Once the temperature changes, it passes the changed temperature data to Sensor_Tag_Node	
Sensor_Tag (sensor-tag)	An atomic model corresponding to the RFID sensor-tag. Environmental information like temperature that is delivered from any Carrier_Node is stored and maintained inside. In addition, the information stored is passed to Reader_Node by request	
DB_Node (information system)	An atomic model for the external information system or data storage for temperature-.timestamp information transferred from Reader_Node	
End_Node (logical component)	An atomic model to start and end a simulation run. This model generates a signal for simulation start and activates the connected atomic models such as Sensor_Tag_Node, Carrier_Node and Reader _Node. When it is used as the last component in the simulation model, it sends a 'deactivating' signal to Database_Node, and Sensor_Tag_Node	

State variables: model state can be one of elements of following set phase
phase = {"active", "passive", "requested", "closing", "responding"}

Parameters : each parameter has several options regarding the sensor-tag specifications.
When_to_Sense = {EVENT, INTERVAL} // decide when to check temperature
When_to_Store = {EVENT, INTERVAL, NOTHING} // decide when to store the temperature checked
What_to_Store = {AS_IS, RANGE_OUT, GAP_OUT} // decide the condition of storing temperature
Overflow_Treat = {NOTHING, REMOVE_ALL, QUEUE, LAST_UPDATE}
// treatments for tag memory full case

In case of (When_to_Sense == INTERVAL)
External Transition Function (δ_{ext}) :
Case : **if**(port == "in") ∧ (x == "start") ∧ (phase == "passive") **then** phase = "active", σ = 0
Case : **if**(port == "in") ∧ (x == "requested") ∧ (phase == "active") **then** phase = "requested", σ = 0
Case : **if(**port == "t_in") ∧ (x ∈ *XT*) ∧ (phase == "active") **then** set *CT* = *XT*, phase = "active" for the remaining time.
Case : **if**(port == "in") ∧ (x == "stop") ∧ (phase == "active") **then** phase = "closing", σ = 0

Internal Transition Function (δ_{int}) :
if phase = "closing" **then** phase = "passive" forever
else if phase = "requested" **then** phase = "responding" for0seconds
else if (phase = "active") ∧ (current_time == sensing_time)
then *ST* = *CT*, sensing_interval += sensing_interval, phase = "active" until any event happens.
else phase = "active" until any event happens.

Output function (λ) :
if (phase = "responding") **then** send current_temperature through port "t_out" to the connected Reader_Node
else do nothing

Figure 5. Sensor_Tag_Node pseudo code

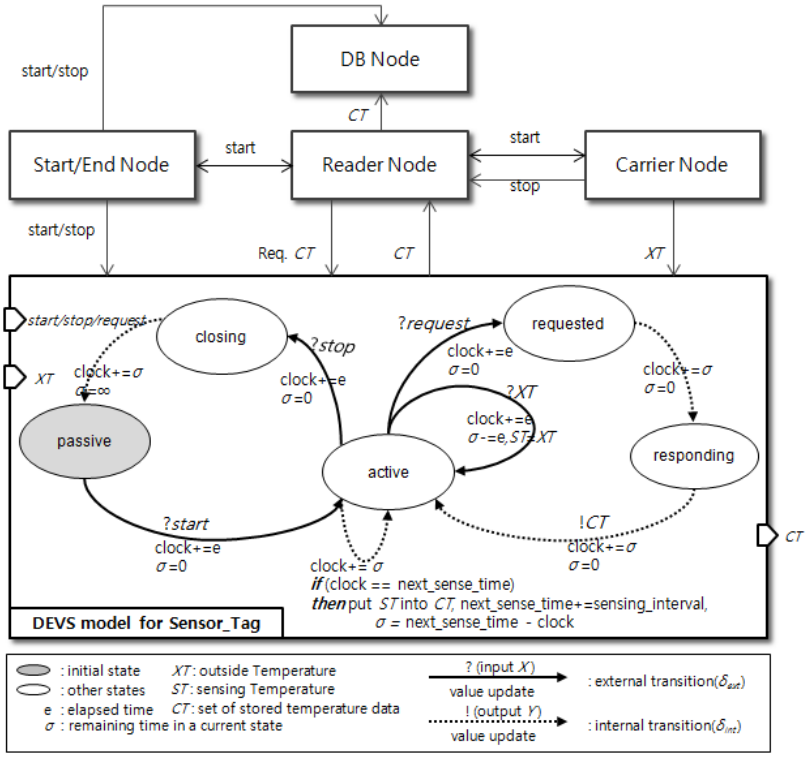


Figure 6. Sensor_Tag model

(2) Reader_Node 모델

Reader_Node는 RFID 리더/미들웨어를 추상화 한 것으로 RFID 안테나와 함께 센서태그 메모리에 저장된 정보를 읽어오는 기능을 모델링한다. 정보를 요구하는 시점과 센서태그에

메모리를 관리(삭제, 백업 방식)하는 방식 등을 파라미터 설정 방식으로 구현하였으며, 이는 실제의 동작 양식과 유사하다. 정보를 센서태그에서 요구하는 방식은 특정 시간간격에 반복적으로 요구하거나(interval), 센서태그가 리더의 안테나 범위를 지날 때(event)에 요구하는 두 가지 옵션 중에서 선택하도록 하였다. 요구하는 데이터도 Sensor_Tag에 저장된 모든 정보 또는 최근 정보 등을 선택하는 것이 가능하고, Sensor_Tag로터 전달받은 데이터를 Sensor_Tag에서 삭제 또는 유지하는 작업 명령을 전달한다.

Reader_Node 모델은 두 개의 입력 포트(in, t_in)와 3개의 출력 포트(out, c_out, t_out)가 구비되어 있다. 'in'과 'out'은 End_Node 모델에서와 같이 모델의 활성화를 위한 신호를 받거나, 연속된 모델에 활성화 신호를 보내는데 사용된다. 't_in', 't_out'은 Sensor_Tag 로부터 정보(온도, 시간)를 받아 DB_Node로 보내는데 사용된다. 'c_out'으로 정보요청/정보관리을 위한 명령을 Sensor_Tag로 보내는데 사용된다. 내부/외부 상태전이에 의한 입출력과 상태의 변화를 <Figure 7>에 제시하였다.

(3) Carrier_Node 모델

Carrier_Node는 운송용 트럭, 냉장/냉동 창고등과 같이 콜드체인 공급망에서 운송 대상을 저장하거나, 운송하는 실질적인 공간을 추상화 한 것이다. 센서태그의 입장에서 보면 특정 기간 동안 독립적인 외부 환경(온도, 습도)을 제공한다. Carrier_Node는 제공해야 할 환경 값을 항시 유지하고 있다가, 센서태그의 측정 요구에 따라 그 값을 제공하는 것이 기본 기능이다. Carrier_Node는 Reader를 포함하고 있는 경우, 즉 운송/보관 기간 동안에 고정된 Reader가 주기적으로 측정된 정보를 읽어 가는 경우와, Carrier_Node 전후에 Reader 가 설치되어 있는 경우가 있을 수 있다.

Carrier_Node는 하나의 입력 포트(in)와 3개의 출력 포트(out, c_out, t_out)를 가진다. 'in'과 'out'은 모델의 활성화를 지시하는 신호를 받거나, 연속된 모델에 활성화 신호를 보내는데 사용된다. 't_out'은 Carrier_Node 내부의 정보(즉, 온도)를 Sensor_Tag로 보내는 경우에 사용된다. 'c_out' 출력 포트는 Carrier_Node가 Reader를 포함하고 있는 경우 Reader의 작동(start, stop)을 요청하는 신호를 보내는데 사용된다. 입력포트를 통한 정보의 전달, 내부/외부 상태전이에 의한 입출력과 상태의 변화를 <Figure 8>에 제시하였다.

(4) DB_Node

DB_Node 모델은 Reader_Node로부터 전달받은 데이터를 저장하는 데이터베이스 또는 정보시스템에 대한 모델이다. 다수

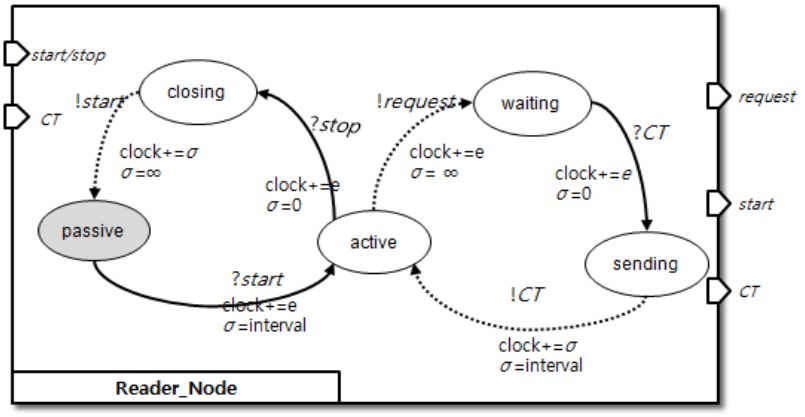


Figure 7. Reader_Node model

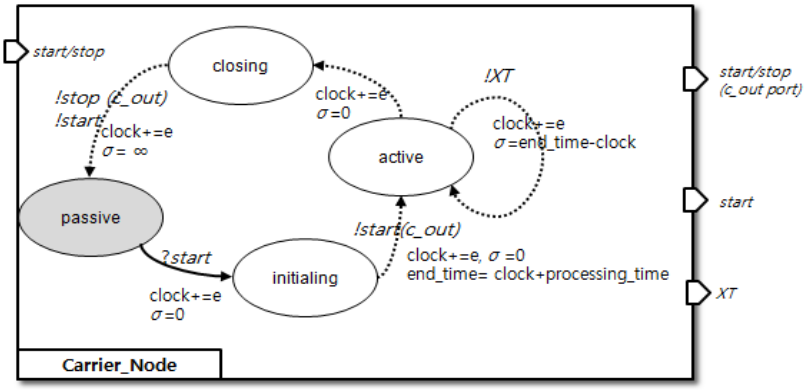


Figure 8. Carrier_Node model

의 Reader_Node 모델로부터 전달된 데이터는 Reader_Node에 시간 순서에 맞게 저장되고, 중복적인 것들이 가려내어진다. End_Node의 'stop' 명령에 따라서 시뮬레이션 결과를 준비하고 출력하는 역할도 수행한다. <Figure 9>에 DB_Node의 상태전이모델을 제시하였다.

(5) End_Node 모델

논리적인 구성요소인 End_Node 모델은 하나의 입력 포트(in)와 하나의 출력 포트(out)를 가지고 있다. 대상이 되는 콜드체인 시스템 모델의 양끝단의 시작과 종료를 위해 사용되는 논리적인 요소로서, 상태 값에 따라서 시작 또는 끝 모델로 사용된다.

시작 요소로 사용되는 경우 시뮬레이션의 초기 단계에서 'start' 신호를 연결된 Sensor_Tag, Reader_Node, Carrier_Node로 보내 활동을 수행하게 한다. 끝 요소로 사용되는 경우는 연결된 Sensor_Tag, DB_Node에 'stop' 신호를 보내서 보유하고 있는 정보를 시뮬레이션 출력으로 변환하고 시뮬레이션 수행을 종료 시키는 역할을 한다. <Figure 10>에 End_Node의 상태전이 모델을 제시하였다.

4.3 모델 구성 방법

정의된 다섯 개의 atomic 모델은 콜드체인의 기본적인 빌딩 블록의 역할을 하고, 빌딩 블록의 조합을 통해서 다양한 구성의 콜드체인 시스템을 모델링 할 수 있다. 앞서 설명한 단위 DEVS 모델을 연결하여 단위 기능을 넘어서는 복잡한 대상을 모델링하는 것이 가능하다. 시뮬레이션 모델은 콜드체인을 구성하고 있는 주요 atomic 모델을 콜드체인의 구성 순서대로 연

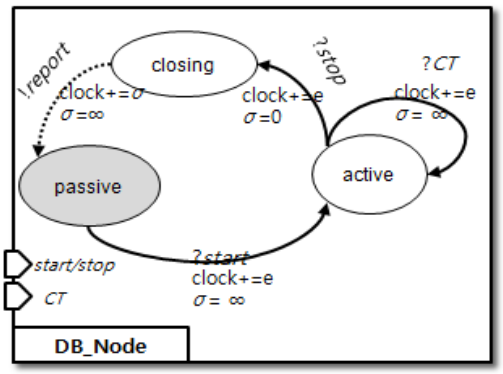


Figure 9. DB_node model

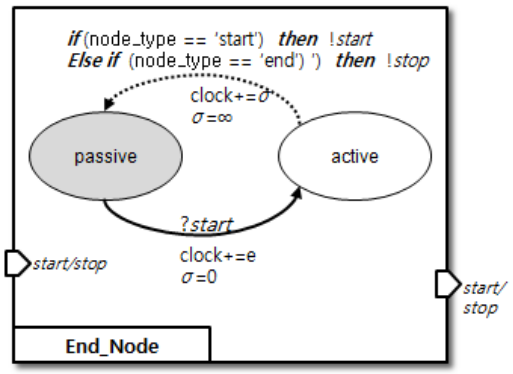
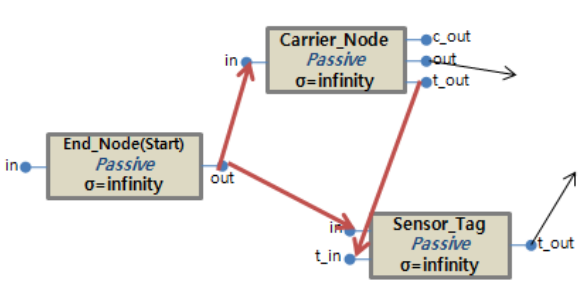
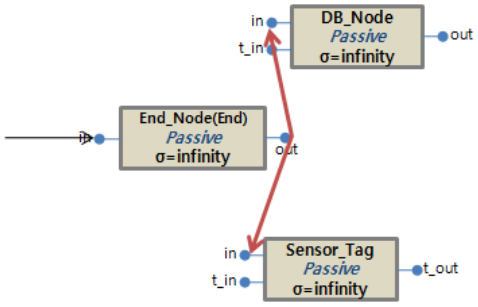


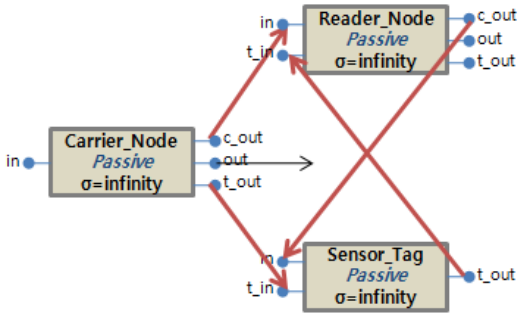
Figure 10. End_node model



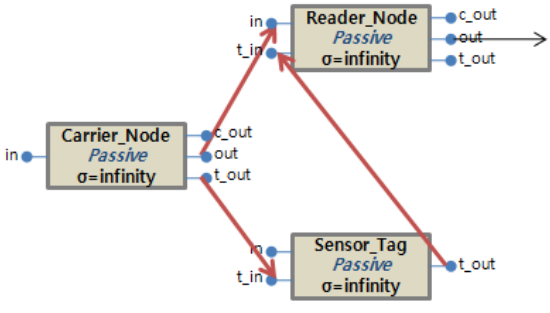
(a) End_node(start type) connection



(b) End_node(end type) connection



(c) Carrier_Node connection(installed reader_node)



(d) Carrier_Node connection(independent reader_node)

Figure 11. Model coupling

결하고, 모든 구성요소와 하나의 Sensor_Tag 모델과의 연결하는 방식을 통해서 모델을 구현할 수 있다. <Figure 11>은 단위 모델들 간의 전형적인 연결 형태들을 보여준다.

단위 모델의 연결을 <Figure 11>의 도식을 이용해서 살펴보면, <Figure 11>(a)에 End_Node가 시작점(START)으로 사용되었을 때 후속하는 Carrier_Node와 콜드체인 상에 존재하는 Sensor_Tag와의 연결을 나타낸 것이다. End_Node의 'start' 출력에 의해 Carrier_Node와 Sensor_Tag가 활동을 시작하게 된다. <Figure 11>(b)는 End_Node가 끝점(END)으로 사용되었을 때, 활성화 되어 있는 Sensor_Tag와 DB_Node를 종료시키는 명령을 보내는 연결을 표현한 것이다. <Figure 11>(c)는 Carrier_Node 내부에 Reader_Node가 설치되어 있는 경우를 표현한 것으로 Carrier_Node의 c_out과 Reader_Node의 in 포트가 연결되어, Carrier_Node가 종속적인 Reader_Node의 활성화와 비활성화를 조정할 수 있다. <Figure 11>(d)는 Carrier_Node와 Reader_Node가 독립적으로 연속되어 배치되는 경우의 연결 예로서 Carrier_Node의 out 과 Reader_Node의 in이 연결되어 연속적으로 연결된 모델을 활성화할 수 있도록 하였다. 제시된 연결 예에서 Sensor_Tag와의 연결한 방식도 주의 깊게 살펴봐야 한다. 다수의 요소로 구성되는 복잡한 콜드체인 모델링도 여기서 제시한 연결 패턴과 유사한 방법으로 단위 모델을 연결함으로 가능하다.

5. 시뮬레이션 테스트

5.1 시뮬레이션 모델링 사례

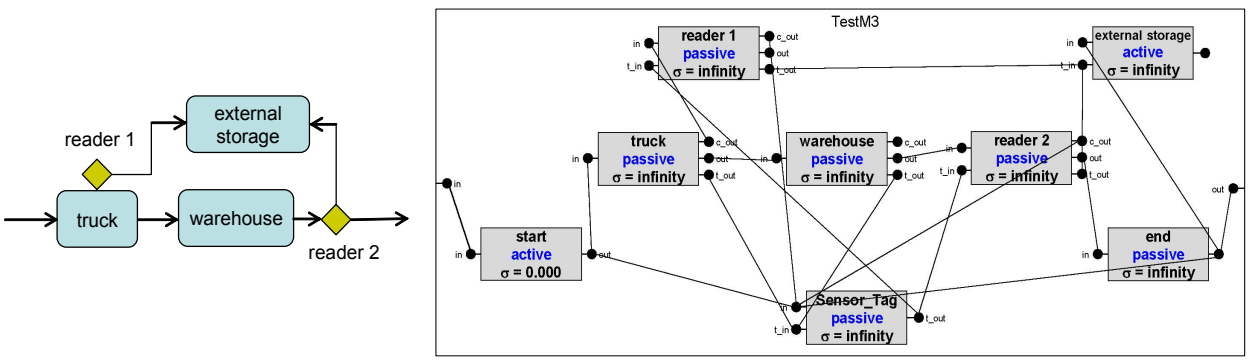
냉장 트럭과 창고로 구성된 간단한 콜드체인의 예를 <Figure 12>(a)에 도식화하였다. 트럭내부에는 리더가 설치되어 설정된 방식(예를 들어 주기적으로)으로 정보를 외부 정보시스템으로 전송한다. Sensor_Tag는 트럭을 통해 이동되어 창고에 보관되었다가, 출하되면서 다시 한 번 리더를 통과하여 온도 정보를 전달하는 시나리오이다. 이러한 콜드체인을 본 연구에서 제시한 DEVS 기반 coupled 모델로 표현한 것이 <Figure 12>(b)이다. <Figure 12>(b)는 DEVS-Suite의 atomic 모델 기반의 애니메이션 화면을 재구성한 것인데, 두 그림의 비교를 통하여 sensor-tag의 라이프사이클에 기반을 둔 DEVS 모델이 어떻게 구성되는지를 비교/파악하는 것이 가능하다. 기본적으로 두 그림의 구성요소는 동일하고, <Figure 12>(a)에 없는 Sensor_Tag가 있으며, 제 4.3절에서 설명한 구성요소간의 연결 방법을 통해서 연결 한 것을 알 수 있다. 아무리 복잡한 콜드체인의 경우라도 이와 같은 방식으로 구성요소를 파악하고 연결하면 쉽게 구성하는 것이 가능하다.

5.2 DEVSJAVA 시뮬레이션 소스코드

<Figure 13>은 Devs-Suite에 의해 <Figure 12>(b)로 표현되기 전의 실제 시뮬레이션 모델인 Java 소스코드의 일부이다. 전체 소스코드는 시뮬레이션 엔진을 구동하는 메서드를 호출하는 부분을 포함하나 지면의 제약으로 생략하였다. 구성을 살펴보면 두 부분으로 구성할 수 있는데, 첫 번째 부분은 Java의 클래스로 정의된 atomic 모델의 객체(object)를 생성하는 부분이다. 소스코드에서 살펴보면, Java의 'new' 메서드를 사용하여 <Figure 12>(b)의 구성 요소들을 하나씩 선언하였고, 또한 객체의 생성자(constructor method)의 파라미터로 구성 요소의 작동 조건을 명시하였다. 두 번째 부분은 구성 요소의 연결하는 부분인데, 소스코드에서 보듯이 "addCoupling"이라는 메서드를 통해서 객체(구성요소)들을 연결한 것을 확인 할 수 있다. 이 메서드를 호출한 수는 <Figure 12>(b)에 요소들의 연결선의 수와 정확하게 일치한다.

5.3 Output 화면

<Figure 14>는 예제 시뮬레이션 수행 결과로서, 본 콜드체인 시뮬레이션을 통하여 파악할 수 있는 결과에 대한 개략적인 아이디어를 제공하고자 한다. 현재의 결과화면은 세 부분으로 구성되어 있는데, 각각 실험조건(1), 센서태그와 관련된 정보(2), 그리고 외부 정보시스템에 전달된 정보(3)를 요약하여 출력한다. (1) 실험 조건에는 센서태그의 온도 측정 및 내부 메모리 저장에 관한 옵션 및 내부 저장 공간이 오버플로 상황 시 처



(a) Simplified cold chain system (b) DEVS models corresponding to the system left(a)

Figure 12. Simple cold chain system subject to model in the proposed DEVS simulator

```
public Test_Sim() {
    super("TestM3");
    initialize();
    addInport("in"); addOutport("out");

    ViewableAtomic sn_tag= new Sensor_Tag ("sensor-Tag",
          When_to_Sense.INTERVAL, when_to_Store.INTERVAL,
          What_to_Store.RANGE_OUT, Overflow_Treat.REMOVE_ALL,
          3, 4.3, 100, 3, 5.0, -20, 3);
    ViewableAtomic sn= new End_Node("Start",End_Type.START);
    ViewableAtomic cn_1= new Carrier_Node ("carrier
          1",20,Reader_Install.YES, temp_queue);
    ViewableAtomic rd_1= new Reader_Node ("reader 1",
          Back_Up_Method.YES_REMOVE,
          When_to_Read.INTERVAL,6);
    ViewableAtomic cn_2= new Carrier_Node ("carrier 2",
          10,Reader_Install.NO,temp_queue);
    ViewableAtomic rd_2= new Reader_Node ("reader 2",
          Back_Up_Method.YES_REMOVE, When_to_Read.EVENT,4);
    ViewableAtomic en= new End_Node ("end point",
          End_Type.END);
    ViewableAtomic dn= new DB_Node ("DB System",5.0, -20);
          add(sn_tag); add(sn); add(cn_1); add(rd_1); add(cn_2);
          add(rd_2); add(en); add(dn);

    addCoupling(this, "in", sn, "in");
    addCoupling(sn, "out", cn_1, "in");
    addCoupling(cn_1, "c_out", rd_1, "in");
    addCoupling(cn_1, "out", cn_2, "in");
    addCoupling(cn_2, "out", rd_2, "in");
    addCoupling(rd_2, "out", en, "in");
    addCoupling(en, "out", this, "out");
    addCoupling(en, "out", sn_tag, "in");
    addCoupling(en, "out", dn, "in");
    addCoupling(sn, "out", sn_tag, "in");
    addCoupling(cn_1, "t_out", sn_tag, "t_in");
    addCoupling(cn_2, "t_out", sn_tag, "t_in");
    addCoupling(rd_1, "c_out", sn_tag, "in");
    addCoupling(rd_2, "c_out", sn_tag, "in");
    addCoupling(sn_tag, "t_out", rd_1, "t_in");
    addCoupling(sn_tag, "t_out", rd_2, "t_in");
    addCoupling(rd_1, "t_out", dn, "t_in");
    addCoupling(rd_2, "t_out", dn, "t_in");

    addTestInput("in", new entity("packet"));
    initialize();
    showState();
}
```

Components declaration

Make connections (coupling)

Figure 13. Java codes for <Figure 12>(b)

```
Experimental Conditions ----------1
==========================================
When to check temperature: INTERVAL/1.0
When to store Temperature: EVENT(by Reader)
Temperature storing condition: Temperature as it is
In case of memory overflow: remove the oldest data
==========================================

Tag summary Report ----------2
==========================================
Ending Clock : 30.00
occurrences of sensing : 30
occurrences of storing : 13
occurrences of overflows : 0
max. stored data : 13
Avg. stored data: 6.02
Actual temperature bound off time: 18.40
Stored temperature bound off time: 16.10
Sensor temperature bound off time: 18.00
==========================================

DB Storage Report----------3
==========================================
Ending Clock : 30.00
No. of Data delivered : 21
No. of Distinct : 14
DB temperature bound off time: 16.10
Timeliness : 4.90 No of Data out of range: 8
==========================================
Terminated Normally at ITERATION 127 ,time: Infinity
```

Figure 14. Simulation output(Sample)

리 방식 등이 제시되어있는데, 기본적으로 실험 조건에 관련된 항목이다. (2) 센서태그 정보는 시뮬레이션 시간, 측정 횟수, 저장 횟수, 오버플로 발생 수, 최대 정보 저장 회수, 평균 저장 온도 정보 수, 실제 범위 밖의 온도 시간, 센서태그에 저장된 범위 밖의 온도 시간 등을 제시하였다. (3) 전달된 정보에는 온도 정보의 수(중복/비 중복), 저장된 정보를 바탕으로 관리범위를 벗어난 시간, 측정된 정보가 정보시스템까지의 전달에 소요된 평균시간 등을 포함한다. 향후 다양한 성능 척도가 개발된다면, 시뮬레이션 결과에 포함하는 것은 언제나 가능하다.

6. 결 론

시뮬레이션은 실제 투자를 수행하기 이전에, 비선형적인 시스템 행태를 사전에 예측하고 발생할 수 있는 문제점을 제거 할 수 있는 유용한 도구이다. 최근 적극적으로 검토되고 있는 RFID 센서태그기반의 콜드체인관리 시스템은 초기투자비가 막대하고, 운용 및 유지 보수에도 큰 비용이 요구되는 바, 센서태그의 최적 운영전략을 포함한 시스템 설계에 있어서 시뮬레이션은 유용하게 활용될 수 있다. 콜드체인콜드체인 시스템의 중요한 목표는 정교한 모니터링과 컨트롤을 통한 식품의 신선도 유지와 안전 보장이다. RFID 기반의 센서태그를 이용하여 콜드체인관리 시스템을 구성하는 경우에 사전적으로 검토해야 할 이상 상황 검출 능력, 비용(센서태그의 종류 및 온도 측정 방법, 간격을 기반으로 한 수명 분석)을 본 연구에서 제안한 방법을 통해 손쉽게 검토하는 것이 가능하다. 본 연구는 RFID-센서태그 기반 콜드체인 시스템의 최적 설계를 위한 시뮬레이션에 관한 연구이며 수행한 내용은 다음과 같다.

1. DEVS 방법론 및 DEVSJAVA를 사용하여 RFID 센서태그 기반 콜드체인 관리시스템을 위한 전용 시뮬레이션 도구를 개발하였다. 콜드체인의 구성요소를 DEVS 방법론을 사용하여 모델링하고, DEVSJAVA를 통해 개발/실현하여 시뮬레이션 연구를 수행할 수 있다. 연구 범위에 적절한 시뮬레이션 도구로 범용 시뮬레이션 도구에 대한 사전 지식 없이 손쉽게 콜드체인관리 시스템을 시뮬레이션 연구를 수행할 수 있다.
2. 시뮬레이션의 모델링 입장에서 보면 센서태그를 일반적인 작업물과 같은 개체의 흐름으로 보는 전통적인 관점 대신, 센서태그를 중심으로 관련 시설이 차례로 교류하는 관점을 택하였으며, 이를 통해 구성요소의 간략한 추상화를 통하여 최소화하여, 불필요한 모델링 노력을 제거하려고 하였다. 다섯 개 구성요소의 연결만으로 어떤 콜드체인 관리시스템도 연구 대상으로 하는 것이 가능해졌다. 모델의 구성을 손

쉽게 할 수 있어 시뮬레이션을 위한 노력 보다 콜드체인 관리 시스템에 집중 할 수 있도록 하였다.

3. 개발된 DEVS 기반의 시뮬레이션 모델은 콜드체인 네트워크를 설계하는 전반에 사용하기에는 부족한 면이 있지만, 기존의 물류 네트워크에 센서태그 기반의 시스템을 도입하고자 하는 초기 단계에 센서태그의 다양한 파라미터 조정 및 RFID 및 안테나의 위치 /수량 등을 결정하는데 효율적으로 사용될 수 있으며, 추가적인 계산을 통하여 정보물량 및 태그 수명 예측 등에 효과적으로 사용될 수 있을 것으로 기대된다.

이와 같은 기여에도 불구하고 실제 콜드체인 시뮬레이션을 현업에서 효과적으로 실행하기 위해서는 추가적인 개발이 요구 된다. 즉, 간단하게 Java 소스코드의 작성으로 모델의 구축과 실행이 가능하지만, 비전문가에 의해서 수행되기에는 무리가 있으므로, 추후 Java 소스코드의 생성을 지원하는 그래픽 에디터를 통한 네트워크 모델링 및 속성 값 설정 등이 가능해지면, 보다 효과적으로 사용될 수 있을 것이다. 아울러 본 연구에서 제안된 센서태그 중심의 모델링 방법론은 센서태그 운영 전략의 최적화에는 효과적으로 사용될 수 있으나, 콜드체인 설계의 또 다른 측면인 네트워크의 혼잡 및 정체 구성요소의 용량 선정과 같은 전통적인 분석설계에 활용될 경우에는 센서태그를 개체화하는 전통적인 시뮬레이션 방법론이 더 효과적으로 활용될 수 있을 것이다.

참고문헌

ACIMS (2010), *DEVSJAVA Modeling and Simulation Tool*, http://www.acims.arizona.edu/ SOFTWARE.

Aiello, G., Scalia, G. L., and Micale, R. (2011), *Simulation analysis of cold chain performance based on time-temperature data*, Production Planning and Control, **21**, 1-9.

Bae, J. W., Lee, K., Kim, H., Lee, J. S., Goh, B., Nam, B., Moon, I., Kim, K., Park, J. (2013), Modeling Combat Entity with POMDP and DEVS, *Journal of the Korean Institute of Industrial Engineers*, **39**(6), 498-516.

Carrie, A. (1988), *Simulation of Manufacturing Systems*, Jonh Wiley and Sons(New York).

Catarinucci, L., Colella, R., Esposito, A., Tarricone, L., and Zappatore, M. (2009), *A Context-Aware Smart Infrastructure based on RFID sensor-tags and its Application to the Health-Care Domain*, In Proceedings of IEEE Conference on Emerging Technologies and Factory Automation(ETFA), Mallorca, Spain, 1-8.

Eclipse Foundation (2011), Eclipse IDE Tutorial, http://eclipse.org/, SOFTWARE

Farooq, U., Wainer, G., and Balya, B. (2006), DEVS modeling of mobile wireless ad hoc networks, *Simulation Modelling Practice and Theory*, **15**(3), 285-314.

Fu, W., Chang, Y. S., Myo, M. A., Makatsoris, C., and Oh, C. H. (2008), *WSN based intelligent cold chain management*, In Proceedings of the 6th International Conference on Manufacturing Research (ICMR), Brunel University, UK, 353-360.

Kang,Y., Jin, H., Ryou, O., and Lee, Y. (2012), A Simulation Approach for Optimal Design of RFID Sensor-Tag-Based Cold Chain Systems, *Journal of Food Engineering*, **113**, 1-10.

Kelton, W. D., Sadowski, R. P., and Sturrock, D. T. (2007), *Simulation with Arena*, McRraw-Hill Korea.

Kim, S., Sarjoughian, H., and Elamvazhuthi, V. (2009), *DEVS-Suite : A Simulator Supporting Visual Experimentation Design and Behavior Monitoring*, in Proc. of the Spring Simulation Conf., San Diego, CA, 29-36.

North, M. J. and Macal, C. M. (2007), *Managing Business Complexity : Discovering Strategic Solutions with Agent-Based Modeling and Simulation*, Oxford University Press, 59-75.

Pujo, P., Pedetti, M., and Giambiasi, N. (2006), Formal DEVS Modelling and Simulation of a Flow-Shop Relocation Method Without Interrupting the Production, *Simulation Modelling Practice and Theory*, **14**(7), 817-842.

Pereira, D. P., Dias, W. R. A., Braga, M. D. L., Barreto, R. D. S., Figueiredo, C. M. S., and Brilhante1, V. (2008), *Model to integration of RFID into Wireless Sensor Network for tracking and monitoring animals*, In Proceedings of the 11th IEEE International Conference on Computational Science and Engineering(CSE), SãoPaulo, Brazil, 125-131.

Sarjoughian, H. (2010), *DEVS-Suite WebStart*, http://acims1.eas.asu.edu/WebStarts/.

Sarjoughian, H. and Zeigler, B. P. (1998), *DEVSJAVA : Basis for a DEVS-based Collaborative M&S Environment*, Proc. of SCS Western Multi-Conference, 5, San Diego, CA, 29-36.

Traub, K., Allgair, G., Barthel, H., Burstein, L., Garrett, J., Hogan, B., Rodrigues, B., Sarma, S., Schmidt, J., Schramek, C., Stewart, R., and Suen, K. (2005), *The EPC global Architecture Framework*, EPC global Final Version.

Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000), *Theory of Modeling and Simulation*, New York : Academic Press.