http://dx.doi.org/10.3745/KIPSTA.2012.19A.2.093

LLRP(Low Level Reader Protocol) 서버를 위한 멀티쓰레드 구조의 설계

이 태 영[†] · 김 윤 호[††] · 성 영 락[†††] · 오 하 령[††††]

요 약

LLRP(Low-Level Reader Protocol)는 RFID 리더(LLRP 서버)와 RFID 어플리케이션들(LLRP 클라이언트)간의 인터페이스를 기술하고 있다. LLRP 서버는 여러 기능들을 동시에 수행해야 한다. 본 논문에서는 LLRP 서버를 멀티 쓰레드 구조로 설계한다. 이를 위하여 (i) LLRP 서버와 클라이언트들 간의 동작 절차를 분석하고, (ii) LLRP 서버가 만족해야 할 기능적인 요구조건들을 제시하고, (iii) 그 요구조건들을 만족할 수 있도록 LLRP 서버의 역할을 여러 쓰레드로 나누고, (iv) 쓰레드 수준에서 LLRP 동작 절차를 세분하였다. 설계된 구조를 검증하기 위하여 이산사건 시스템을 계층적이고 모듈화된 방식으로 기술하는 언어인 DEVS 형식론을 이용하여 설계된 내용을 모델링하고 시뮬레이션 하였다. 시뮬레이션 결과, 제안된 구조는 LLRP 표준과 주어진 모든 기능적 요구 조건들을 만족함을 알 수 있었다.

키워드 : LLRP, RFID, 멀티스레드, DEVS 형식론, 시뮬레이션

Design of a Multi-Thread Architecture for an LLRP Server

Tae-Young Lee[†] · Yun-Ho Kim[††] · Yeong Rak Seong[†††] · Ha-Ryoung Oh[††††]

ABSTRACT

LLRP (Low-Level Reader Protocol) specifies an interface between RFID readers and RFID applications, also called LLRP servers and clients respectively. An LLRP server should concurrently execute various functions. This paper designs an LLRP server of a multi-threaded architecture. For that, (i) the operational procedure between LLRP servers and clients is investigated, (ii) the functional requirements of LLRP servers are presented, (iii) the operation of an LLRP server is decomposed into several threads to satisfy those functional requirements, and (iv) the operational procedure is further examined in thread-level. To validate the designed architecture, it is modeled and simulated by using the DEVS formalism which specifies discrete event systems in a hierarchical, modular manner. From the simulation result, we can conclude that the proposed architecture conforms the LLRP standard and satisfies all the given functional requirements.

Keywords : LLRP, RFID, Multi-Thread, DEVS Formalism, Simulation

1. 서 론

최근 들어 유비쿼터스 사회의 핵심 기술 중 하나로써 RFID(Radio Frequency IDentification) 기술과 관련 산업에 대한 관심이 높아지고 있다. 일반적으로 상위 애플리케이션, RFID 리더, RFID 태그로 구성되는 RFID 시스템은 그 자체만으로도 하나의 산업군을 형성하고 있지만, 다른 산업 분야에 융합된 촉매적인 기능으로써 더 다양한 가능성을 내포하고 있다[1]. 하지만 이러한 가능성에 비하여 그동안 RFID 기술이 실질적으로 사용되는 산업 분야의 폭은 제한적이었다. 그 이유 중 하나는 리더와 태그 간 통신에는 거의 모든 제조사가 표준화 된 무선인터페이스를 따르는 반면, 상위 애플리케이션과 리더 간 통신은 리더 제조사마다 제공하는 프로토콜이 달랐기 때문이다.

이러한 문제점을 해결하기 위해 EPCglobal에서는 상위 애플리케이션과 RFID 리더 간 프로토콜 표준으로 LLRP (Low-Level Reader Protocol)[2]을 제정하였다. LLRP는 RFID 리더와 태그 간의 무선인터페이스에 대한 정확한 이해를 바탕으로 상위 애플리케이션과 RFID 리더간의 상호작용을 정의한 프로토콜이다. LLRP는 TCP/IP를 기반으로 클라이언트-서버(client-server) 구조를 따른다. 일반적으로 클

† 준 회 원 : 국민대학교 전자공학과 석사
†† 준 회 원 : 국민대학교 전자공학과 박사과정
††† 종신회원 : 국민대학교 전자공학과 교수
†††† 정 회 원 : 국민대학교 전자공학과 교수
논문접수 : 2011년 5월 2일
수 정 일 : 1차 2011년 10월 18일, 2차 2011년 12월 19일
심사완료 : 2011년 12월 19일

라이언트-서버 구조는 클라이언트의 서비스 요청에 서버가 서비스 응답을 하는 방식으로 동작되는데, LLRP 시스템에서는 상위 어플리케이션이나 RFID 미들웨어가 클라이언트의 역할을 하고, RFID 리더가 서버의 역할을 하게 된다.

LLRP를 사용하는 시스템에서의 RFID 리더(이하 LLRP 서버 혹은 리더)는 클라이언트와의 통신 및 메시지 처리, 각종 이벤트 관리와 태그와의 통신, 결과 값 보고, 에러처리의 역할을 수행한다[2]. LLRP 서버를 구현하기 위해서는 TCP/IP 뿐만 아니라 앞서 언급한 역할들을 동시에 수행해야 하므로 운영체제를 탑재하지 않은 펌웨어(firmware) 레벨에서는 구현의 한계가 있다. 또한 순차적인 방식의 싱글 쓰레드로는 앞서 언급한 역할 수행이 어렵기 때문에 시분할(time sharing) 방식의 태스크 스케줄링(task scheduling)을 지원하는 운영체제 환경 하에 멀티쓰레드 구조로 구현하는 것이 바람직하다.

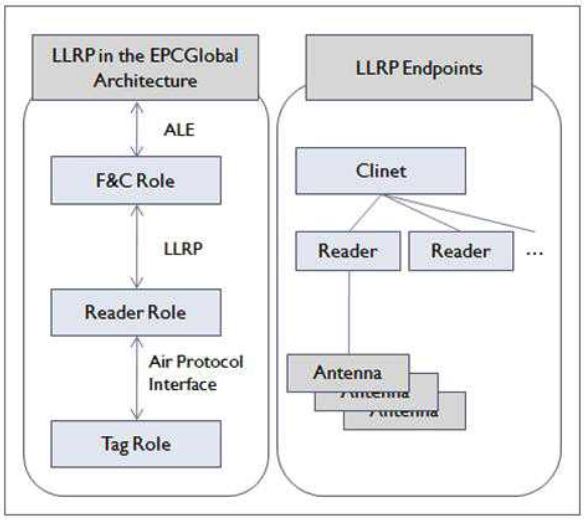
본 논문에서는 멀티쓰레드 구조의 LLRP 서버를 설계한다. LLRP 시스템의 동작 절차로 부터 LLRP 서버 설계를 위한 요구사항을 도출하고 이를 만족하기 위해 분리되어야 할 쓰레드들을 정의하였다. 또한 일반적인 동작절차를 분리된 쓰레드 수준에서 다시 표현하고 이를 바탕으로 각 쓰레드의 역할 및 관계를 명확히 하였다. 설계된 내용의 검증에는 이산사건 시스템을 기술하는 언어인 DEVS 형식론[3][4][5]이 이용되었다. 본 논문에서 DEVS 형식론을 사용한 이유는 (i) 멀티 쓰레드 시스템은 손쉽게 이산사건 시스템으로 모델링할 수 있으며, (ii) 일반적인 프로그래밍 환경과 달리 DEVS 형식론에서는 구성요소(이 논문에서는 쓰레드)들의 동작을 가상 시간(virtual time)을 기반으로 한 동기화 메커니즘으로 엄격하게 제어할 수 있고, (iii) DEVS 형식론의 추상화된 시뮬레이터(abstract simulator) 알고리즘을 이용하여 DEVS로 기술된 내용을 쉽게 시뮬레이션 할 수 있기 때문이다.

본 논문의 구성은 다음과 같다. 2장에서는 LLRP 시스템을 간략하게 소개하고, 3장에서는 이를 바탕으로 LLRP 서버시스템을 위한 멀티쓰레드 구조를 제안한다. 4장에서는 3장에서 제안된 멀티쓰레드 구조를 DEVS 형식론으로 모델링하고 시뮬레이션 한다. 마지막으로 5장은 본 논문의 결론이다.

2. LLRP 시스템

LLRP는 EPCglobal에서 제정한 RFID 리더-클라이언트 간의 프로토콜로써, (그림 1)에서 나타난 바와 같이 RFID리더와 클라이언트 사이의 상호작용을 정의하는 인터페이스이다. LLRP는 TCP/IP를 기반으로, 상위 어플리케이션으로 태그에 저장한 데이터를 전달하는 데이터 경로, 리더의 상태 및 장치 설정의 관리를 위한 경로, 리더의 제어 경로를 제공한다[2].

LLRP의 통신은 기본적으로 메시지를 통하여 수행되며, 메시지 내부에 미리 규정된 Spec과 Trigger에 따라 동작 방



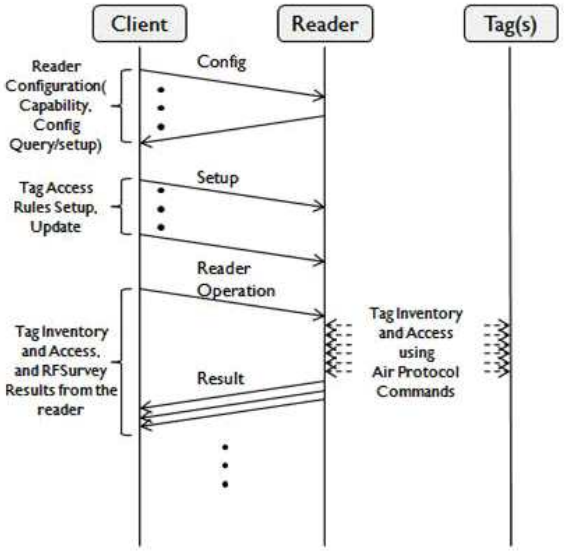
(그림 1) LLRP 시스템 구조[2]

식을 결정하고 수행한다. 메시지는 버전, 메시지 타입, 길이, ID, 파라미터를 포함하고 있다. LLRP의 주요 Spec 및 Trigger는 <표 1>과 같으며, 보다 자세한 내용은 [2]를 참조하기 바란다.

〈표 1〉 LLRP의 주요 Spec 및 Trigger

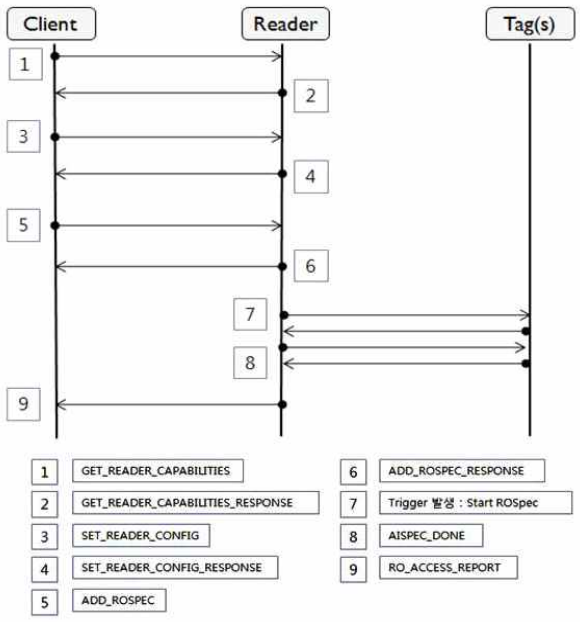
Spec	내 용
ROSpec	한 번의 리더 동작을 정의하며, 하나 이상의 AISpec 혹은 RFSurveySpec을 포함
AISpec	한 번의 인벤토리 동작에 필요한 무선인터페이스 프로토콜과 RF 파라미터 포함
Access-Spec	액세스 작업을 수행할 TagSpec과 수행할 액세스 작업의 정의인 OpSpec을 포함하며, 두 가지 Spec 모두 특정 무선 프로토콜에 맞추어 정의
RFSur-veySpec	안테나의 주파수 집합에 대해 전력 레벨을 측정하고 조사하는 동작에 대한 파라미터를 포함
Report-Spec	ROReportSpec과 AccessReportSpec을 포함하며, ROReportSpec은 ROSpec의 실행 중에 발생한 결과를 보고하는데 사용되고, AccessReportSpec은 AccessSpec의 실행 중에 발생한 결과를 보고할 때 사용됨
Trigger	LLRP의 시작과 종료 시점을 결정하기 위해 사용

(그림 2)는 LLRP 기반 RFID 시스템의 일반적인 동작 절차를 보여준다. 클라이언트에서 리더의 Capability, Config등을 확인할 수 있으며, 필요에 따라 Config값을 수정할 수 있다. 그 후에 ROSpec, AISpec, RFSurvey-Spec, AccessSpec 등을 추가하여 동작시키고자 하는 내용을 정의할 수 있다. 클라이언트에서 정의한 내용에 따라 리더는 해당 역할을 수행하게 되며, 인벤토리 작업을 진행하게 된다. 인벤토리 작업 후 그에 따른 결과를 보고함으로써 일반적인 동작이 완료된다.



(그림 2) LLRP 시스템의 일반적인 동작절차[2]

(그림 3)은 위에서 설명한 일반적인 동작절차를 메시지를 통하여 좀 더 구체적으로 나타낸 것이다. GET_READER_CAPABILITIES 메시지를 통하여 리더의 Capability를 읽어오며, SET_READER_CONFIG 메시지를 통하여 리더의 Config값을 설정해준다. ADD_RO- SPEC 메시지를 통하여 클라이언트에서 수행하고자 하는 ROSpec을 리더에게 전달하고, START_ROSPEC 메시지로 인벤토리 작업의 시작을 명령하면, 리더는 그에 따른 응답메시지를 보내고, 태그와 통신을 시작하게 된다. 클라이언트로부터 수신된 작업이 끝나게 되면, RO_ACCESS_REPORT 메시지를 통하여 결과값을 클라이언트에게 전송하고 동작을 완료한다.



(그림 3) 메시지로 표현한 LLRP 시스템의 일반적인 동작절차

이러한 간단한 동작 중에서도 예외사항은 언제든지 발생할 수 있다. 기본적으로 수신된 메시지가 잘못되었을 경우를 생각할 수 있다. 이런 경우에 리더는 클라이언트에게 에러 값을 포함한 RESPONSE 메시지를 전달한다. 또한 ADD_ROSPEC 메시지의 경우에도 Op-Spec이 비어있거나, 추가하려는 ROSpec의 상태가 비활성화 상태일 경우 응답메시지를 통하여 문제가 있음을 보고한다. START_ROSPEC 메시지의 경우에는 메시지 파라미터에 들어있는 ROSPEC ID와 리더가 가지고 있는 ROSPEC의 ID가 일치하지 않는 경우 에러 값을 포함한 응답메시지를 클라이언트에게 전달하게 된다. 이러한 에러 처리 동작은 매우 광범위하며, 보다 자세한 내용은 [2]를 참조하기 바란다.

3. 멀티쓰레드 구조 설계

본 장에서는 LLRP 서버를 설계한다. LLRP 프로토콜 문서는 LLRP에 대한 기본적인 내용들만을 담고 있으므로, 이를 이용하면 매우 다양한 형태의 LLRP 서버가 설계될 수 있다. 본 논문에서는 다음의 요구사항을 고려하여 LLRP 서버를 설계하였다.

(R1) LLRP 서버는 비동기적으로 발생하는 클라이언트의 요청 메시지나 돌발적으로 발생하는 예외상황을 처리하기 위해 시스템 내부의 상태와 상관없이 메시지를 송수신할 수 있어야 한다.
(R2) LLRP 서버는 클라이언트로부터 전달된 각종 메시지의 파라미터를 저장하고 관리할 수 있어야 한다.
(R3) LLRP 서버는 정해진 시간에 스케줄 된 이벤트나 동작을 처리할 수 있어야 한다.
(R4) LLRP 서버는 RFID 무선 인터페이스의 저수준(low level) 접근이 가능해야 한다.
(R5) LLRP 서버는 다양한 제조사의 리더를 지원할 수 있어야 하며, 이 경우 하드웨어에 의존적인 부분을 최소화할 수 있어야 한다.
(R6) LLRP 서버는 태그와의 통신 결과 및 리더 상태 정보를 축적하고 관리하며 클라이언트의 요청에 따라 보고할 수 있어야 한다.
(R7) LLRP 서버는 LLRP의 통신 프로토콜을 따르기 위해 TCP/IP를 지원해야 한다.

(R1)~(R7)의 요구사항들을 동시에 수용하기 위해서는 일반적인 펌웨어(firmware) 환경보다는 시분할(time sharing) 방식의 태스크 스케줄링(task scheduling)을 지원하는 운영체제 환경에서 다수의 작업들을 동시에 병행해서 처리할 수 있는 멀티쓰레드 구조로 LLRP 서버를 설계하는 것이 적합하다.

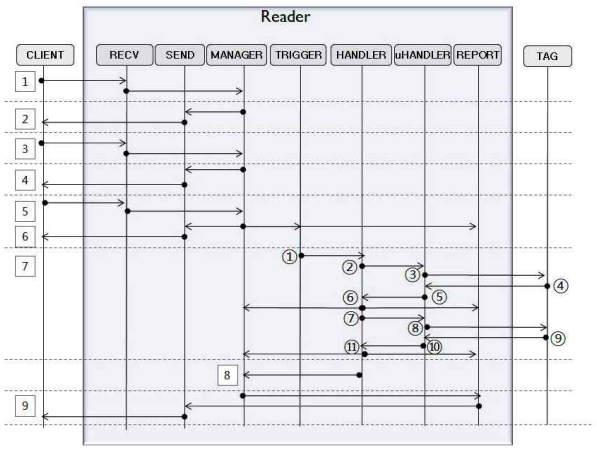
먼저 (R1)로부터 메시지 송수신 부분 분리가 필요함을 알 수 있다. 송수신 부분을 분리함으로써 통신의 응답성을 향상시키며 실제 산업현장의 필요에 의해 통신인터페이스 변

경이 불가피할 경우 수정 부분을 최소화할 수 있는 이점이 있다. 본 논문에서는 이러한 이유로 수신을 담당하는 *RECV* 쓰레드와 송신을 담당하는 *SEND* 쓰레드를 다른 부분들과 분리하였다. 또한 (R3)을 만족시키기 위해서 단독으로 시간에 관련된 처리를 담당하며 이를 이벤트화 해주는 기능을 *TRIGGER* 쓰레드로 분리하였다.

이렇게 메시지 송수신 부분과 시간과 관련된 이벤트를 다루는 쓰레드를 분리함으로써 메시지 응답성을 높이고 시간적 오차를 줄일 수 있지만 여전히 수신된 메시지를 처리하고 관련 파라미터를 저장하는 부분과 태그와의 통신을 통해 결과 값을 모으거나 보고하는 역할 등이 혼재하고 있다. 그리고 이러한 상황은 임계영역(critical region)이나 데이터의 일관성(consistency) 등의 다양한 문제들을 야기할 수 있는 가능성을 가지고 있다. 따라서 태그와의 통신을 전반적으로 관리하는 기능과 태그관련 작업의 결과값 및 리더상태에 관한 정보를 저장하고 관리하는 기능이 분리될 필요가 있다. 이러한 기능을 분리시키면 리더는 태그와의 무선인터페이스에 접근하는 동안에도 클라이언트와 통신이 가능하며, 다양한 예외상황에 대처가 가능할 뿐만 아니라 태그 통신과 관련된 파라미터들을 관리하기가 수월하다. 또한 결과 값 및 리더상태에 관한 정보를 따로 저장하고 관리함으로써 클라이언트에게 효과적으로 정확한 정보를 보고할 수 있으며 (R4) 및 (R6)도 만족시킬 수 있다. 본 논문에서는 태그와의 통신을 담당하는 *HANDLER* 쓰레드와 작업 결과 값을 관리하는 *REPORT* 쓰레드로 분리하였다. 또한 다양한 제조사의 리더를 지원(R5)하기 위해 *HANDLER* 쓰레드의 기능 중에서 하드웨어에 의존적인 부분은 *uHANDLER* 쓰레드로 분리하였다.

마지막으로 (R2)를 포함하여 쓰레드들의 동작을 전반적으로 관리하기 위하여 *MANAGER* 쓰레드를 만들었다. *MANAGER* 쓰레드는 리더의 전반적인 동작을 관리하며, 클라이언트의 요구에 따라 파라미터 값을 저장하고 리더 동작에 반영할 수 있다. 이와 같은 과정들을 통해 LLRP 서버의 역할을 모두 만족할 수 있는 7개의 쓰레드를 분리하였다.

(그림 4)는 분리된 쓰레드들의 역할을 바탕으로 (그림 3)의 동작 절차를 쓰레드 수준으로 표시한 것이다. (그림 4)의 1번은 리더의 능력을 조회하고자 하는 메시지로 (그림 3)의 1번 메시지에 대응된다. 제안한 멀티쓰레드 구조는 메시지의 수신 기능을 *RECV*가 담당하므로 *RECV*가 메시지를 수신한 다음 *MANAGER*로 메시지를 전달한다. *MANAGER*는 전달된 메시지를 분석하여 다른 쓰레드에게 필요한 파라미터를 넘기거나 내부적으로 처리 결과를 저장한다. (그림 4)의 2번 메시지는 *MANAGER*가 메시지를 처리하여 응답하는 과정을 보여준다. *MANAGER*가 가지고 있는 리더의 Capability를 응답메시지의 파라미터에 추가한 후에 클라이언트에게 보낸다. 메시지를 송신하는 기능은 *SEND*가 담당하므로 *MANAGER*는 *SEND*에게 메시지를 전달하고, *SEND*가 클라이언트에게 메시지를 송신한다.



(그림 4) (그림 3)의 동작절차에 대한 쓰레드 수준에서의 표현

리더의 능력 조회 후 클라이언트는 LLRP 프로토콜과 관련된 다양한 파라미터들을 설정하기 위한 메시지를 리더에게 전송한다. 리더는 전송된 파라미터를 내부적으로 저장하고 응답 메시지를 생성한다. (그림 4)의 3, 4번 메시지는 이와 같은 과정을 나타내며 이때 쓰레드 간 동작 절차는 1, 2번 메시지와 동일하다.

리더의 능력 값 조회 및 설정 후, 클라이언트는 실질 적인 태그와의 무선인터페이스 작업을 위한 메시지를 리더에게 전송한다. (그림 3)의 5, 6번은 이러한 작업이 정의되어 있는 ROSpec을 리더에게 전달하고 메시지의 성공 여부를 응답받는 과정을 나타낸다. 이를 쓰레드 수준에서 분석해 보면 5번 메시지가 리더 동작의 전반적인 관리를 하는 *MANAGER*에게 전달되어야 한다. 그러면 *MANAGER*는 ROSpec에 포함 된 시간 이벤트와 결과 보고 파라미터 처리를 위해 *TRIGGER*, *REPORT*에게 메시지를 전달한다. *TRIGGER*는 START_ROSPEC을 발생시킬 시간을 설정하며, *REPORT*는 파라미터의 설정에 따라 결과를 바로 보고하거나 축적하게 된다. 마지막으로 *SEND*가 메시지 처리의 성공여부를 추가한 응답메시지를 클라이언트에게 전송함으로써 6번 메시지까지 동작이 완료된다.

앞선 과정을 통해 ROSpec을 전달받은 리더는 클라이언트로부터의 START_ROSPEC 메시지 수신이나 ROSpec에 저장된 태그 인벤토리 작업의 시작 조건이 만족하였는지를 모니터링 하게 된다. 이들 중 하나라도 만족하면 즉각적으로 태그와의 무선인터페이스 작업을 시작한다. (그림 3)에서 7번 메시지는 5번 메시지를 통해 전달된 ROSpec의 시작 조건 중 시간에 관한 부분이 만족되어 태그와의 통신이 시작되는 과정을 나타낸 것이다. 시간에 관련된 파라미터를 저장하고 모니터하던 *TRIGGER*는 시간이 만족됨과 동시에 (그림 4)의 ①을 통하여 태그와의 무선인터페이스를 관리하는 *HANDLER*에게 작업을 시작할 것을 알리며, *HANDLER*는 실질적인 태그와의 통신을 하는 *uHANDLER*에게 ②를 통하여 태그와의 통신 내용이 담겨있는 OpSpec을 전달한다. ③, ④과정을 통하여 *uHANDLER*는

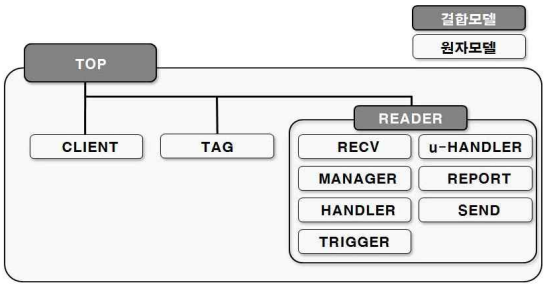
TAG와 통신을 하며 그 결과 값을 ⑤를 통하여 *HANDLER*에게 전달한다. *HANDLER*는 ⑥을 통하여 *MANAGER*와 *REPORT*에게 결과를 보고한다. *MANAGER*는 전체적인 동작을 관리하기 위하여, 어떤 작업이 할당되어있고 얼마만큼 진행되었는지 파악하며, *REPORT*는 결과를 축적한다. *HANDLER*는 더 수행되어야 할 OpSpec이 있는지 확인 후에 다음 동작을 결정한다. (그림 4)에서는 수행되어야 할 OpSpec이 2개라고 가정한 것이다. 따라서 *HANDLER*는 ②~⑥과정을 1번 더 반복한다. ⑪의 과정까지 끝나면 *HANDLER*는 더 이상 수행할 OpSpec이 없으므로 8번 AISPEC_DONE 메시지로 할당된 작업이 끝났음을 보고한다. 8번 메시지로부터 더 이상의 태그 관련 작업이 없음을 알게 된 *MANAGER*는 *REPORT*에게 작업이 끝났고, 축적된 결과 값을 보고하라는 메시지를 전달하게 된다. *REPORT*는 9번 메시지를 통하여 축적된 결과 값을 응답 메시지의 파라미터에 추가하여 *SEND*에게 보내고, *SEND*는 클라이언트에게 메시지를 전달한다. 이렇게 해서 (그림 4)의 동작이 완료된다.

4. 모델링 및 시뮬레이션

본 절에서는 앞서 설계한 내용들을 검증한다. 설계된 소프트웨어를 검증하기 위해서는 설계된 내용을 우선 구현한 다음에 검증하는 것이 일반적이다. 설계된 내용에는 특정한 상태에서 특정한 입력이나 사건들이 발생하는 경우에 처리하는 방법에 대한 것들이 포함된다. 이러한 내용을 정확하게 검증하기 위해서는 프로그램이 그런 특정한 상태를 가지도록 한 후에 특정한 입력이나 사건들을 발생시켜 상태의 변화를 추적해야 한다. 이를 위해서 일반적인 프로그래밍 환경에서는 대개의 경우 디버거(debugger) 프로그램을 이용하고 있다. 하지만 멀티 쓰레드 환경에서는 디버거의 사용이 매우 어렵다. 또한 멀티 쓰레드 환경에서는 여러 쓰레드들의 실행 흐름이 마구 뒤섞이는 경쟁조건(race condition)이 발생하는 문제도 생긴다.

본 논문에서는 이러한 여러 문제들을 이산사건 시스템을 기술하는 언어인 DEVS 형식론을 이용하여 해결한다. 앞서 설계한 내용을 살펴보면 여러 쓰레드들이 계속해서 상태를 바꾸어가며 발생된 입력이나 사건들을 처리하고 있으므로 손쉽게 이산사건 시스템으로 모델링할 수 있다. 일반적인 프로그래밍 환경과 달리 DEVS 형식론에서는 구성요소(쓰레드)들의 동작의 흐름을 가상 시간을 기반으로 한 동기화 메커니즘으로 엄격하게 제어할 수 있다. 또한 DEVSim++[4]와 같은 DEVS 추상화 시뮬레이터 환경을 이용하면 쉽게 모델링된 결과를 시뮬레이션 할 수 있어서 실제 코드를 구현하지 않고서도 설계된 내용을 검증할 수 있다.

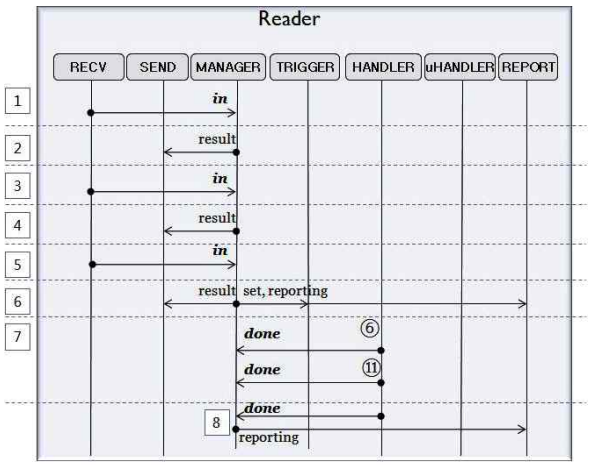
DEVS 형식론으로의 모델링은 3장의 최종 설계 결과인 (그림 4)에서부터 시작된다. (그림 4)의 쓰레드들은 각각 DEVS 형식론의 원소모델(atomic model)로 모델링된다. 또한 쓰레드들의 계층적인 구성과 쓰레드들 사이의 메시지의



(그림 5) 전체 모델의 계층적 구조

전달 관계들은 결합모델(coupled model)들로 모델링된다. DEVS 형식론의 원소모델과 결합모델의 정의에 대한 구체적인 내용은 [3]를 참조하여라.

(그림 5)는 전체 시스템을 나타내는 TOP 모델을 포함하여 분리 된 전체 모델의 계층구조를 나타낸다. 이 중 TOP 모델과 READER 모델은 결합모델이며 나머지 모델들은 원자모델이다. 분리된 각 모델들은 입출력 포트를 이용하여 서로 통신하므로 분리된 컴포넌트들을 DEVS 형식론으로 나타내기 위해서는 각각의 입출력 관계를 바탕으로 포트 이름을 결정해야 한다.



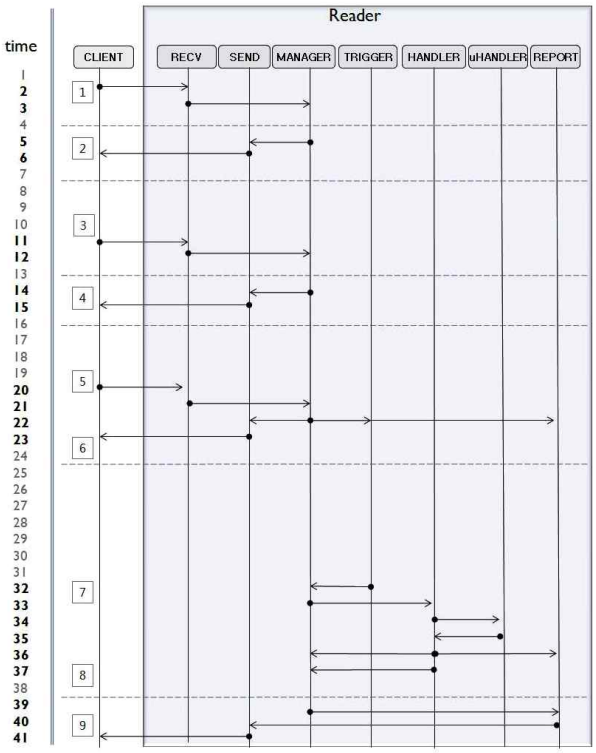
(그림 6) 원자 모델의 일반적인 동작절차

(그림 6)은 원자 모델 중 한 예로 MANAGER의 입출력 흐름을 표시하고 포트 이름을 결정한 것이다. 1~8은 (그림 3)의 메시지와 동일하다. MANAGER는 in 포트를 통해 입력을 받으며 (그림 6)에는 없지만 handle 포트를 통하여 HANDLER에게 메시지를 전송할 수 있도록 연결되어 있다. 또한, result 포트를 통하여 SEND에게 결과 값을 알려주거나 reporting 포트를 통하여 REPORT에게 결과 값을 전달할 수 있으며, set 포트를 통하여 TRIGGER에게 시간을 설정할 수 있다. 각 포트의 이름은 MANAGER의 동작과 상대 모델을 참고하여 결정하였으며 이와 같은 과정을 걸쳐서 모든 모델의 포트 이름을 결정할 수 있다. 결정된 모델의 포트이름과 입출력 관계를 도식화할 수 있다.

제안된 모델을 시뮬레이션 하기 위해 앞서 언급한 LLRP 서버의 모델들을 DEVSim++로 구현하였다. 또한 시뮬레이션 할 수 있는 여러 가지 경우를 조사하였다. 본 논문에서는 그 중 가장 기본이 되는 (그림 3)의 절차에 대한 시뮬레이션을 예로써 설명하겠다. 시뮬레이션을 하기 전 다음과 같은 설정 값을 임의로 가정하였다.

1. 리더의 Capabilities는 정의되어 있다.
2. 리더의 Config값은 기본 값이 있다.
3. ADD_ROSPEC 메시지를 통하여 다음의 파라미터들을 설정하였다.
   가. ROReportSpec 파라미터에서 Report시기는 AI-Spec이 완료되었을 때이다.
   나. ROSpecStartTrigger 파라미터에서 10초 후에 트리거가 발생하도록 설정되어있다.
   다. OpSpec은 2개만 있다.
4. 동작의 검증을 위한 클라이언트의 메시지 전송은 (그림 3)을 따른다.
5. uHANDLER와 TAG모델간의 동작은 생략한다.
6. CLIENT는 결과값을 받은 후 5초 후에 다음 메시지를 발생시킨다.

이러한 설정 값을 적용하고 시뮬레이션을 수행한 결과값으로 시간에 따른 모델간의 메시지 전송 결과를 얻을 수 있었다. 결과 메시지들을 보기 쉽도록 도식화 한 것을 (그림 7)에 나타내었다.



(그림 7) 시뮬레이션 결과

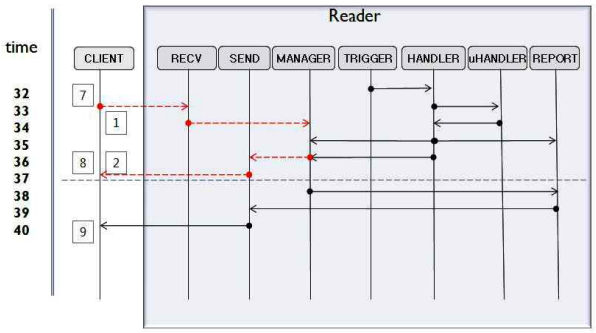
(그림 7)의 ①~⑨는 (그림 3)의 메시지와 동일하다. 본장의 도입부에서 설명했듯이, 이 상황을 동작하고 있는 코드 상에서 재현하기는 굉장히 어렵다. 하지만 시뮬레이션을 함으로써, 정확하게 원하는 시점에서 이벤트가 일어날 수 있도록 타이밍을 조절할 수 있고, 이벤트들이 언제 발생했는지 알기도 쉽다. (그림 7)은 결과 값이 시간 순서로 정렬되어 있으며, (그림 3)의 일반적인 동작절차를 제대로 수행하고 있음을 확인 할 수 있다. 특히 (그림 7)은 (그림 4)와 매우 비슷한 것을 볼 수 있다. 단지 다른 것은 시뮬레이션을 통해 메시지의 송수신이 된 시간을 알 수 있다는 것과, 앞서 가정했듯이 TAG와 통신하는 부분이 생략되었다는 것이다. (그림 7)의 결과는 앞서 임의로 설정해준 값을 제대로 반영하고 있는 것을 확인할 수 있다. ADD_ROSPEC 메시지를 통하여 ROSpec을 생성할 때 시뮬레이션을 위해 임의로 START_TRIGGER가 10초 후에 발생하도록 설정을 했는데, msg_TRIGGER_CONFIG메시지로 설정 된 후 10초 후에 START_ROSPEC메시지가 발생한 것을 확인할 수 있다. 또한 CLIENT는 메시지를 수신하고, 5초 후에 다음 메시지를 송신하는 것을 확인하였다.

기본적인 동작 절차 외에 여러 가지 예외상황을 고려하여 시뮬레이션 하였으나, 본 논문에서는 그 중 대표적인 예를 보인다.

ex1) ROSPEC이 동작하는 동안 클라이언트로부터 요청 메시지가 있을 경우
ex2) ROSPEC이 동작하는 동안 추가적인 ROSPEC이 들어왔을 경우
   ex2.1) 추가적인 ROSPEC의 우선순위가 동작 중인 ROSPEC의 우선순위보다 낮을 경우
   ex2.2) 추가적인 ROSPEC의 우선순위가 동작 중인 ROSPEC의 우선 순위보다 높을 경우

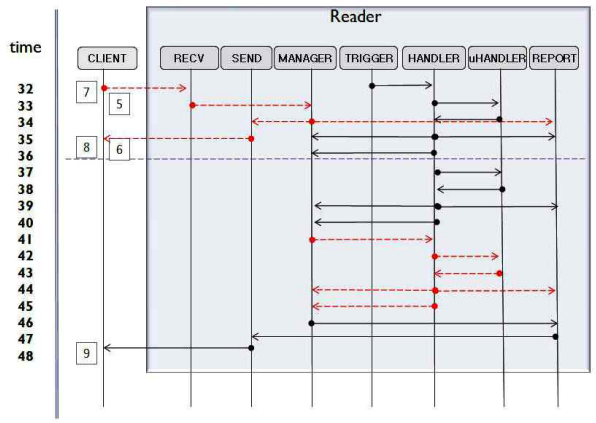
특히 위와 같은 경우에 대한 시뮬레이션은 CLIENT가 다수가 있을 경우나 여러 가지 명령을 제대로 수행하는지와 READER가 동작하는 도중 새로운 명령에 대하여 대처하는 방법에 대하여 알 수 있다. 가장 먼저 ex1의 경우 리더가 태그와 통신하고 있는 중, 클라이언트로부터 어떠한 요청메시지가 들어올 경우 리더가 제대로 대처할 수 있는지 확인하기 위한 시뮬레이션이었다. 전체적인 동작은 기본적인 동작절차와 같으나, 태그와 통신을 시도하는 시점에 클라이언트가 GET_READER_CONFIG 메시지를 리더에게 전달하도록 상태를 조절하여 보았다.

(그림 8)에서 일반 화살표는 기존의 기본적인 동작절차에서 주고 받은 메시지를 나타내며, 점선 화살표는 임의로 생성한 메시지이다. (그림 8)에서 확인할 수 있듯이, 기본적인 동작절차는 그대로 진행이 되며, 임의로 생성한 메시지인 GET_READER_CONFIG 메시지에 대해서도 GET_ READER_CONFIG_RESPONSE 라는 응답 메시지를 제대로 생성하여 CLIENT에게 전달하는 것을 확인 할 수 있었다.



(그림 8) 리더 동작 중 클라이언트로부터 요청 메시지가 올 경우

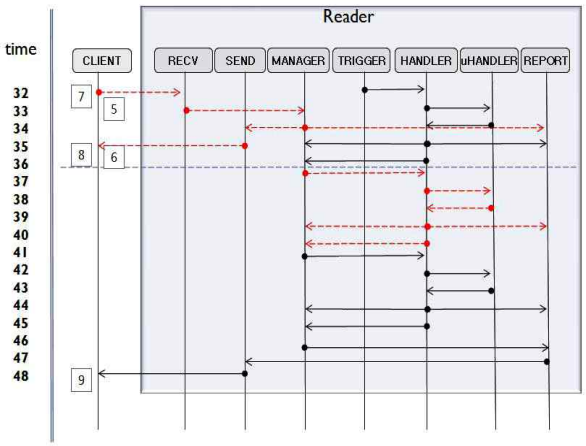
다음으로는 리더가 전달받은 ROSpec을 통해 태그와 통신을 하고 있는 도중에, 새로운 ROSpec 메시지를 전달 받았을 경우, 우선순위에 따라 제대로 동작하는지 확인하여보았다. 프로토콜 문서에서는 ROSPec의 동작은 우선순위에 따라 선점 될 수 있다고 명시되어 있으나, 동작 중인 ROSpec이 어느 시점에 선점 되어야 한다는 내용은 명시되어 있지 않다. 본 논문에서는 태그와의 통신의 안정성을 위해 하나의 OpSpec이 종료 되는 시점에 선점 될 수 있도록 하였다. 다음은 우선순위에 따라 선점 되거나 비 선점되었을 경우를 나타낸다.



(그림 9) ROSpec의 우선순위에 따라 비선점 되었을 경우

우선 비선점 되었을 경우 리더의 동작을 도식화하여보았다. ROSpec의 우선 순위는 MANAGER 모델이 관리하게 된다. (그림 9)에서 일반 화살표를 통해 이미 ROSpec의 동작이 시작된 것을 확인 할 수 있다. 그 시점에 임의로 점선 화살표인 ADD_ROSPEC 메시지를 생성하여 보았다. MANAGER는 ADD_ROSPEC 메시지를 받은 후에 REPORT 모델에게 보고 시기를 설정하고 SEND 모델을 통해 클라이언트에게 응답메시지를 보내게 된다. 하지만 HANDLER에게는 아무런 메시지를 보내지 않을 것을 확인할 수 있다. 동작 중인 ROSpec이 DONE_AIRFSPEC 메시지를 통하여 동작이 완료되었음을 MANAGER에게 알리면 MANAGER는 그 때 ADD_ROSPEC 메시지를 통하여 저장하고 있던 ROSpec을 HANDLER에게 전달하게 된다.

이와는 반대로 ADD_ROSPEC 메시지를 통하여 전달 된 ROSpec의 우선 순위가 동작중인 ROSpec보다 높을 경우는 다음과 같이 동작하게 된다.



(그림 10) ROSpec의 우선순위에 따라 선점 되었을 경우

(그림 10)은 ROSPec이 선점되었을 경우를 나타낸다. MANAGER는 ADD_ROSPEC 메시지를 받은 후에 REPORT 모델에게 보고 시기를 설정하고 SEND 모델을 통해 클라이언트에게 응답메시지를 보내게 된다. 동작 중인 ROSpec이 DONE_AIRFSPEC 메시지를 통하여 하나의 OpSpec 동작이 완료되었음을 MANAGER에게 알리면 MANAGER는 그 때 선점해야 하는 ROSpec을 HANDLER에게 전달하게 된다. 선점 된 ROSpec이 동작 완료하게 되면, MANAGER는 기존의 ROSpec의 나머지 동작을 완료하기 위해 다시 HANDLER에게 기존 ROSpec 정보를 전달하게 된다.

(그림 8), (그림 9), (그림 10)의 동작을 싱글쓰레드에서 처리하게 되면, 임의로 생성한 점선의 화살표는 READER가 동작 중이므로, 곧바로 결과를 얻을 수가 없다. 싱글쓰레드에서는 동작 중 메시지를 받게 되면, 진행 하던 동작을 완료 후에야 새로운 메시지를 처리할 수 있다. 이러한 결과는 클라이언트의 요청이 많을수록 응답성에 차이가 커지는 것을 알 수 있다. 또한, ROSpec의 선점 같은 경우는 싱글쓰레드에서 처리할 수가 없다. 따라서 멀티쓰레드로 설계를 하여야 기본 기능에도 만족시키며, 응답성도 좋아질 수 있다는 것을 알 수 있다.

대표적으로 설명한 기본적인 동작절차 외에도 다양한 경우에 따라 시뮬레이션을 수행하였으며, 그 결과 메시지 송수신이 READER 내부 상태와 상관없이 비동기적으로 수행가능하며, 설정된 이벤트들이 정해진 시간에 정확하게 발생하는 것을 확인할 수 있었다. 또한 각 요청 메시지 및 수시로 발생하는 예외상황에 대해 READER 내부 원자모델들의 상태변화 및 결과 값이 올바르게 축적되고 최종적으로 올바른 응답메시지를 발생시키는 것을 확인하였다.

5. 결 론

본 논문에서는 멀티쓰레드 구조를 이용하여 LLRP 서버를 설계하고 구현하였다. LLRP 서버의 메시지별 동작 절차를 바탕으로 동시성을 요구하는 LLRP 서버의 다양한 역할들을 효과적으로 수행하도록 쓰레드를 분리하였고, 시스템의 응답성과 확장성, 응용환경의 유연성을 높이기 위한 쓰레드들을 추가하였다. 완성된 멀티 쓰레드 구조는 송수신 쓰레드의 분리로 통신의 응답성을 향상시키며, 시간관리 쓰레드의 생성으로 스케줄 된 이벤트의 정확한 실행을 제어한다. 또한 결과값을 저장하는 쓰레드와 태그와 통신하는 쓰레드의 분리를 통해 태그와 무선인터페이스에 접근하는 동안에도 클라이언트와의 통신이 가능하다는 장점이 있다.

제안된 멀티쓰레드 구조는 DEVS 형식론을 이용하여 모델링 및 시뮬레이션 하였으며, 시뮬레이션 결과 설계한 멀티쓰레드 구조가 빠른 응답성을 가지고 일반적인 상황 및 여러 예외상황에서 올바른 응답 메시지를 생성하는 것을 검증 할 수 있었다. 또한, 모델링 된 결과를 토대로 실제 개발 보드에서 구현하였으며, 구현 결과 LLRP 서버의 요구사항을 만족시키는 것을 확인할 수 있었다.

본 논문에서 구현한 멀티쓰레드 구조는 LLRP 서버 기능을 RFID 리더에 적용시키기가 용이하고 다양한 응용환경의 수용이 가능할 것이다. 마지막으로 클라이언트와 리더 간의 네트워크 운용방식 및 태그와 통신 시 발생할 수 있는 다양한 예외 상황, 상용 리더에 의존적인 부분은 향후 실제 RFID 시스템에 적용한 연구가 필요할 것이다.

참 고 문 헌

[1] C. Floerkemeier and S. Sarma, “An Overview of RFID System Interfaces and Reader Protocols,” IEEE Press, 2008.

[2] EPCGlobal, ‘Low Level Reader Protocol(LLRP),’ Version 1.1, 2010.

[3] Bernard P. Zeigler, ‘Object-Oriented Simulation with Hierachical, Modular Models,’ Academic Press, 1990.

[4] Bernard P. Zeigler. Multifacetted Modeling and Discrete Event Simulation. Academic Press, 1984.

[5] Zeigler, B.P., Praehofer, H. and Kim, T.G., “Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems” Academic Press, 2000.

[6] Tag Gon Kim, ‘DEVSim++ User’s Manual : C++ Based Simulation with Hierarchical Modular DEVS Models,’ Systems Modeling Simulation Lab., KAIST, 1994.

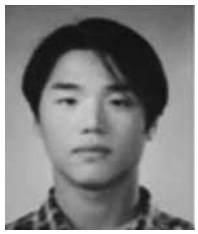
이 태 영

e-mail : endingg@kookmin.ac.kr

2009년 국민대학교 전자공학과(학사)

2011년 국민대학교 전자공학과(석사)

관심분야 : RFID, Embedded system



김 윤 호

e-mail : unonet@nate.com

2005년 국민대학교 전자공학과(학사)

2008년 국민대학교 전자공학과(석사)

2008년～현 재 국민대학교 전자공학과 박사과정

관심분야 : RFID, Sensor network, 모델링&시뮬레이션

성 영 락

e-mail : yeong@kookmin.ac.kr

1989년 한양대학교 전자공학과(학사)

1991년 KAIST 전기 및 전자공학과 (공학석사)

1985년 KAIST 전기 및 전자공학과 (공학박사)

1998년～현 재 국민대학교 전자공학부 교수

관심분야 : RFID, 실시간 처리, 이산사건 모델링&시뮬레이션, 스케줄링

오 하 령

e-mail : hroh@kookmin.ac.kr

1983년 서울대학교 전기공학과(학사)

1988년 KAIST 전기 및 전자공학과 (공학석사)

1992년 KAIST 전기 및 전자공학과 (공학박사)

1996년～현 재 국민대학교 전자공학부 교수

관심분야 : RFID, Embedded system, 실시간 처리, ASIC