논문 02-27-11C-5 한국통신학회논문지 '02-11 Vol.27 No.11C

UML과 SDL을 이용한 무선 트랜잭션 프로토콜의 구현과 성능 평가

정회원 정 호 원*, 임 경 식**

Implementation and Performance Evaluation of the Wireless Transaction Protocol Using UML/SDL

Howon Jung*, Kyungshik Lim** *Regular Members*

요 약

본 논문에서는 프로토콜 개발 도구를 이용하여 Wireless Application Protocol (WAP) 포럼에서 제안하는 무선 트랜잭션 프로토콜(Wireless Transaction Protocol: WTP)을 구현하였다. 또한, 서버모델, coroutine 모델 및 activity-thread 모델에 따라 개발도구의 지원없이 직접 개발된 WTP 구현물들과 그 성능을 비교 분석하였다. 우선 Unified Modeling Language (UML)을 사용하여 프로토콜의 요구사항을 분석함과 동시에 프로토콜 엔진의 구조를 정의하였으며, 이를 기반으로 Specification and Description Language (SDL)을 사용하여 프로토콜 엔진을 상세 설계한 후, 코드 자동 생성기를 이용하여 WTP 구현물을 생성하였다. 구현물의 성능을 분석한 결과, 3,000개 이하의 클라이언트가 동시에 접속할 경우에는 트랜잭션 처리율(throughput)과 트랜잭션 처리 지연시간(system response time) 측면에서 기존의 세 가지 모델을 이용하여 직접 개발한 프로토콜 엔진과 그 성능이 대등함을 알 수 있었다. 그러나, 5,000개 이상의 클라이언트가 동시에 접속할 경우 트랜잭션 성공률은 약 10%까지 급격히 감소하고 트랜잭션 처리 지연시간은 1,500ms까지 증가하였는데, 이는 프로토콜 개발도구를 사용한 경우에 구현물의 크기가 약 62% 증가하면서 프로토콜 처리시간 증가로 인한 것이다. 그러나, 이러한 실험 결과는 실험에 사용된 PC 서버의 사양을 고려할 때 호스트의 과부하로 인한 것이며, 부하분산 기능이 제공되는 실제 환경에서는 프로토콜 개발 도구를 사용하지 않고 직접 개발한 프로토콜 엔진과 거의 대등한 성능을 보였다.

ABSTRACT

In this paper, we design and implement the Wireless Transaction Protocol (WTP) proposed by the Wireless Application Protocol (WAP) forum using a protocol development tool, SDL Development Tool (SDT). And we conduct a comparative performance evaluation of the WTP implementation with other three implementations that are based on different implementation models respectively: the server model, the coroutine model, and the activity-thread model. To implement WTP, we first use Unified Modeling Language (UML) for analyzing the protocol requirement and defining the protocol engine architecture. Next, we use Software Development Language (SDL) to design the protocol engine in details and then generate the WTP implementation automatically with the aid of SDT. The code size of the WTP implementation generated by SDT is 62% larger than the other three implementations. However, its throughput and system response time for transaction processing is almost equal to the other three implementations when the number of concurrent clients is less than 3,000. If more than 5,000 concurrent clients tries, the transaction success rate abruptly decreases to 10% and system response time increases to 1,500ms, due to the increased protocol processing time. But, it comes from the fact that the load overwhelms the capacity of the PC resource used in this experimentation.

* 삼성전자 ** 경북대학교 컴퓨터과학과

논문번호: 020001-0104, 접수일자: 2002년 1월 4일

Ⅰ. 서 론

[1-3]무선 인터넷 프로토콜(Wireless Application Protocol : WAP)은 기존의 인터넷 표준을 기반으로 무선 통신망 및 이동단말 환경에서 최적의 기능을 수행하도록 설계되었으며, 무선 통신 환경의 문제점들을 극복하기 위하여, 유선망에서는 기존의 유선 프로토콜을 그대로 사용하고 무선망에서는 무선환경에 적합하도록 연결 분리(split connection) 기법을 채택하여 설계된 무선 트랜잭션 프로토콜(Wireless Transaction Protocol : WTP)을 전송 프로토콜로 사용하고 있다.

다양한 프로토콜 엔진 개발자들에 의한 임의의 프로토콜 분석 및 설계는 생성된 구현물의 재사용성을 낮추고 다른 플랫폼으로의 통합을 제한하였다. [4-7]이러한 문제점을 극복하기 위하여 프로토콜 공학은 Unified Modeling Language (UML)을 이용한 프로토콜의 분석과 Specification and Description Lnaguage (SDL)을 이용한 프로토콜의 설계 방법을 제시하였다. [8]또한, 본 연구에서 사용한 Telelogic-TAU SDT는 SDL로 설계된 프로토콜을 시뮬레이션 및 검정해 주며, 목적코드 자동 생성기를 이용해 유닉스, 윈도우즈, 리눅스 및 pSOS, VxWorks등의 임베디드 시스템으로의 통합을 지원해 준다.

[7,9-10]한편, 기존의 연구는 프로토콜 엔진 구현을 위한 기본 구조로 서버모델, coroutine 모델, activity-thread 모델을 제시하고 있으며, SDT는 실행상의 안정성을 위해 확장된 서버 모델을 사용하고 있다. 하지만, SDT로 구현된 프로토콜 구현물은 안정성 및 확장성을 위해 첨가된 각종 기능으로 인하여 실행화일의 크기가 크고 효율성도 떨어진다.

본 논문에서는 프로토콜 개발 도구를 이용하여 Wireless Application Protocol (WAP) 포럼에서 제안하는 무선 트랜잭션 프로토콜(Wireless Transaction Protocol: WTP)을 UML 및 SDL을 이용하여 구현하였다. 또한, 서버모델, coroutine 모델 및 activity-thread 모델에 따라 개발도구의 지원없이 직접 개발된 WTP 구현물들과 그 성능을 비교 분석하였다. 이를 위한 본 논문의 구성은 다음과 같다. 2장에서는 UML과 SDL을 이용하여 프로토콜 엔진을 설계하는 방법에 관해 설명하고, 3장에서는 구현을 위한 모델에 따른 프로토콜 엔진의 구조 및 구현에 관해 살펴본 후 각 모델별 성능을 비교하였으며, 4장에서는 결론을 맺는다.

Ⅱ. UML과 SDL을 이용한 무선 트랜잭션 프로토콜 엔진의 구현

본 연구에서는 UML과 SDL을 이용한 프로토콜 엔진의 구현 절차를 그림 1과 같이 설정하였다.

먼저 UML의 유스케이스 다이어그램을 이용하여 프로토콜의 요구사항을 분석하였으며, 그 결과를 바탕으로 클래스 다이어그램을 이용하여 엔진의 구조를 정의하고 SDL의 블럭 다이어그램으로 나타내었다. 다음으로는 순차 다이어그램을 이용하여 엔진을 이루는 각각의 엔터티(Entity)간 메시지 전송 과정을 정의하고 상태 다이어그램으로 각 엔터티의 동적 행위를 정의하여 SDL의 프로세서 다이어그램으로 나타내었다. 한편, 순차 다이어그램은 프로토콜의 검정을 위한 시뮬레이션 시나리오로 활용하였다.

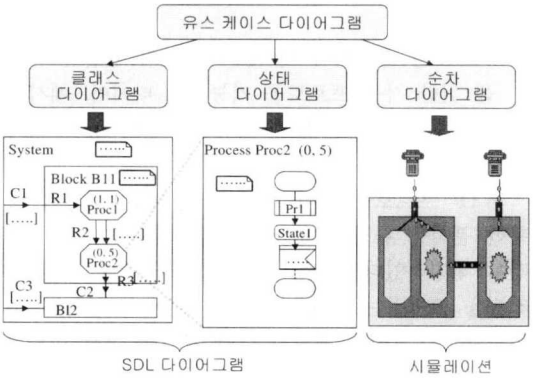


그림 1. UML과 SDL을 이용한 WTP의 개발 절차

2.1. 유스케이스 다이어그램을 이용한 WTP의 요구분석

프로토콜 엔진은 주변의 다양한 시스템 환경과 상호작용을 통하여 메시지를 전송한다. 이러한 주변 환경을 유스케이스 다이어그램의 액터로 분류하였으며, 주요 액터는 다음과 같다.

- 사용자 : 메시지를 전송하고자 하는 외부객체로 상위계층인 WSP 계층, 혹은 어플리케이션(클라이언트, 서버 등)에 해당
- 데이타그램 전송 프로토콜 : 프로토콜 개체간의 통신을 위해 생성된 패킷을 네트워크를 통해서 전송해 주는 데이터그램 전송 서비스로 WDP/UDP 계층
- 시스템 관리자 : 엔진이 운영되도록 하는 관리 모듈 및 기타 주변환경으로 통신 어플리케이션

(브라우저, 게이트웨이 등), 시스템 운영체제, 디바이스 환경 등

무선 트랜잭션 프로토콜은 데이터그램 서비스 상위에서 동작하는 트랜잭션 지향 프로토콜이다. [3] 트랜잭션 서비스는 세 종류의 클래스 즉, 비신뢰적 단방향 요구(class 0), 신뢰적 단방향 요구 (class 1) 및 신뢰적 양방향 요구/응답(class 2)으로 나누어서 차별화된 서비스를 제공한다. 이러한 기능을 수행하기 위한 프로토콜의 주요 서비스를 다음과 같은 유스 케이스로 구분하였다.

- WTP 프로토콜 서비스 : 클래스 0, 클래스 1, 클래스 2의 트랜잭션 단위 메시지 전송을 사용자에게 제공하기 위한 서비스
- 시스템 관리 서비스 : 프로토콜 엔진을 포함한 통신 시스템, 운영체제 및 상·하위 프로토콜 계층 등 주변 환경과 정보를 교환하여 적절한 서비스를 제공할 수 있도록 엔진을 관리하기 위한 서비스
- 프로토콜 유틸리티 서비스 : 엔진의 원활한 동작을 위한 각종 프로토콜 기능으로 타이머, 메시지 재전송, 트랜잭션 ID관리, PDU 분석 등의 서비스

WTP 프로토콜 서비스와 시스템 관리 기능의 정확한 분석을 위하여 유스케이스의 기능을 그림 2와 같이 확장하였다. 우선, 프로토콜 서비스는 메시지 전송을 요청하는 요구 서비스와 이에 응답하는 응답 서비스, 그리고 메시지 전달 방식을 결정하는 트랜잭션 서비스로 확장하였다. 또한, 시스템 관리 기능도 요구자 서비스 관리와 응답자 서비스 관리로 확장하였다. 한편, 확장된 서비스가 정상적으로 이루어지기 위해서는 상호작용이 필요하다. 요구자 관리 서비스는 사용자로부터의 요청에 의해 요구 서비스를 생성한다. 그리고 요구 서비스는 트랜잭션 서비스와 프로토콜 유틸리티 서비스를 이용하여 목적지의 프로토콜 객체로 PDU를 전송한다. 이러한 액터와 유스케이스의 관계를 표현하기 위해 그림 3에서와 같이 생성(0..*), 포함(include) 및 통신관계(communication)를 정의하였다. 이와같이 분석된 각 개체의 확장 및 포함관계는 시스템의 구조와 메시지의 흐름을 정의하는 기본 모델이 된다.

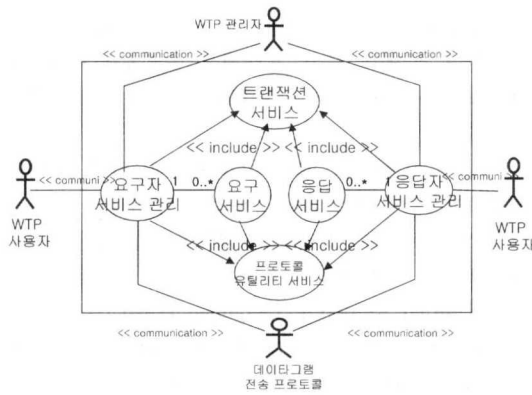


그림 3. WTP 프로토콜 엔진의 구조와 메시지 흐름 정의

2.2. WTP 프로토콜 엔진의 구조 설계

요구분석을 통해 찾아낸 유스케이스는 프로토콜 엔진 구성을 위한 클래스에 해당한다. 이들 클래스 간의 상속관계는 그림 2의 유스케이스간 확장관계를 기초로 그림 4와 같이 SDL 블럭 다이어그램을 구성한다. 한편, 블럭 다이어그램 내의 프로세서 타입은 기능에 따라서 다음 절에서 설명할 동적 행위 설계에 의해 프로세서 다이어그램으로 설계된다.

이와같이 기능에 따른 클래스의 추출과 상속관계를 바탕으로 설계된 프로세서 타입은 재사용성을

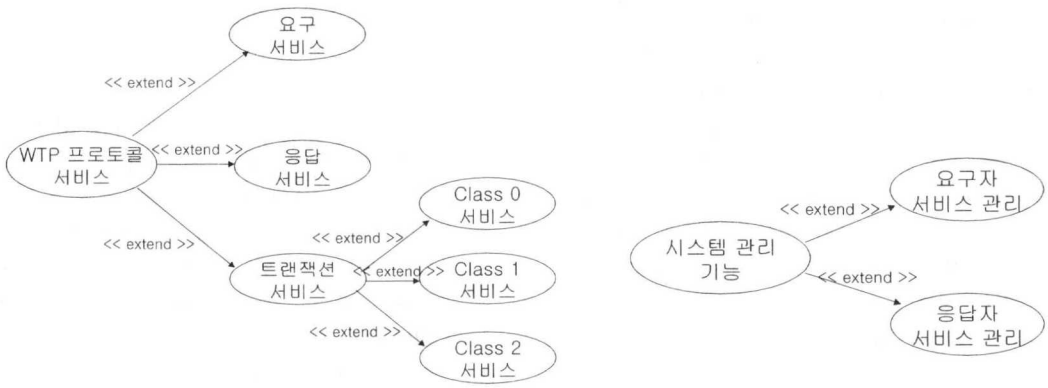


그림 2. WTP 프로토콜의 세부적인 기능 분석을 위한 유스케이스의 확장

향상시킨다. 즉, 요구자 클래스와 응답자 클래스는 WTP의 고유한 기능인 트랜잭션 단위 메시지 전송을 명세하며, 일반적인 프로토콜의 동작은 프로토콜 클래스로부터 상속받아서 사용한다. [5]만일 프로토콜의 기능을 확장해야 할 때에는 기존 다이어그램이 상속 가능하므로 이를 상속받아 기존의 기능을 유지하면서 새로운 기능을 첨가할 수 있다.

블럭 다이어그램을 구성하는 프로세서는 그림 6과 같이 메시지 전달경로를 나타내는 채널로 연결된다. 경로의 설정은 연관된 블럭 및 프로세서 사이에서 이루어지며, 이는 그림 3의 생성 및 통신관계에 해당한다. 또한, 그림 3은 외부와의 인터페이스를 위한 환경도 제시하고 있다. 이를 바탕으로 채널 설정을 위한 클래스의 연관관계 다이어그램을 그림 6과 같이 구성하였다. 여기서 외부 클래스와의 연관관계는 서비스 프리미티브와 PDU의 전달경로로 사용된다.

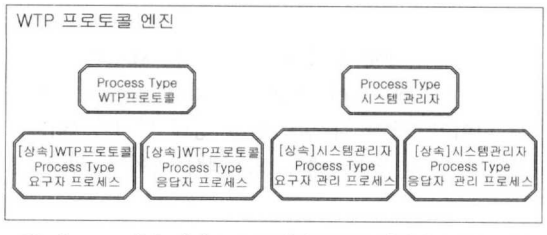


그림 4. 프로세서 타입으로 표현한 WTP 엔진의 블럭 타입과 상속관계

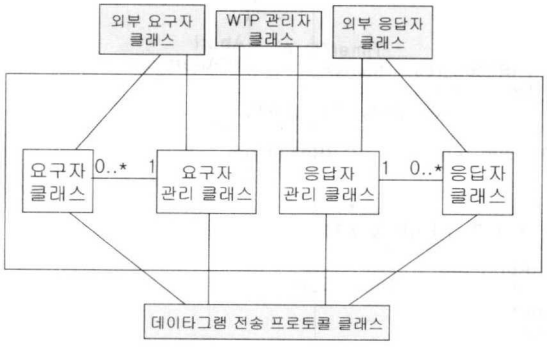


그림 5. WTP 프로토콜 엔진을 구성하는 주요 클래스들간의 생성 및 연관관계

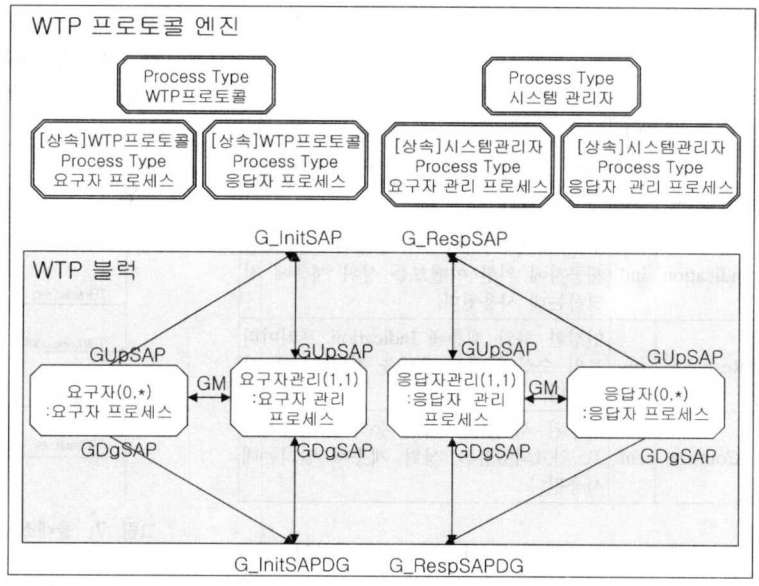


그림 6. WTP 프로토콜 엔진의 내부 채널 및 외부 인터페이스 설정

또한 내부 클래스의 연관관계는 채널 연결 이외에도 프로세서의 생성관계를 표현하기 위하여 사용하였다. 하나의 요구자 관리 클래스가 항상 존재하기 때문에 연관관계 선에는 '1'이라고 표시하였으며, 요구자 클래스는 외부의 요청에 의해 생성되므로 '0..*' 로 표시하였다. 이러한 분석 결과를 바탕으로 그림 4의 블럭 다이어그램에 채널 및 프로세서 생성관계를 첨가한 그림 6의 다이어그램을 구성하였다.

2.3. WTP 프로토콜 엔진의 동적 행위 설계

무선 트랜잭션 프로토콜 규격서는 TRInvoke, TRResult, TRAbort로 구성된 세 가지 서비스 프리미티브를 정의하고 있다. TRInvoke는 프로토콜 사용자가 새로운 트랜잭션을 시작하기 위해 사용하며, TRResult는 프로토콜 사용자가 이미 시작된 트랜잭션 결과를 원격 호스트의 프로토콜 동등 개체에 송신하기 위해 사용하고, TRAbort는 프로토콜 사용자가 수행중인 트랜잭션을 취소시키기 위해 사용한다. 각각의 서비스 프리미티브는 표 1과 같이 세분화된 서비스 종류를 가진다. 한편, 프로토콜 동등 개체들 간의 통신은 PDU의 교환으로 이루어지며, 무선 트랜잭션 프로토콜 규격서에는 Invoke, Result, Ack 등과 같이 여러 가지 목적에 따른 PDU를 정의하고 있다. [3]이는 프로토콜 제어 정보와 사용자 정보로 구성되며, 무선 트랜잭션 프로토콜 개체는 전송계층으로 IP망에서는 UDP의 서비스를, non-IP망에서는

WDP 프로토콜의 서비스를 이용하여 전송한다.

표 1. 서비스 프리미티브 종류

종 류	약어	설 명
Request	req	인접한 하위 계층에 서비스를 요구하는데 사용된다.
Indication	ind	Request 프리미티브의 발생과 프로토콜 제공자에 의한 이벤트를 상위 계층에 전달하는데 사용된다.
Response	res	인접한 하위 계층에 Indication 프리미티브의 수신에 대한 응답을 전달하는데 사용된다.
Confirm	cnf	요구된 서비스를 위한 동작이 성공적으로 완료되었음을 상위 계층에 알리는데 사용된다.

이와같은 프로토콜에서 사용되는 메시지의 전송 절차와 참여하는 엔터티를 파악하기 위하여 순차 및 상태 다이어그램을 사용하였다. 그림 7의 순차 다이어그램은 클래스 2 트랜잭션을 위해 사용하는 메시지와 이에 참여하는 엔터티로 구성된다. 또한 메시지의 교환 절차를 확인할 수 있으며, 엔진의 내부 기능을 위한 새로운 이벤트인 *Wake up* 메시지가 요구 서비스 엔터티를 생성하고 있다는 것을 알 수 있다.

그림 7에서 응답 서비스와 같이 프로토콜의 기능을 담당하는 각각의 엔터티는 다양한 입력 및 출력 메시지를 복잡한 절차에 따라 처리한다. 그러므로, 메시지 전달에 참여하는 서비스의 동작을 그림 8과 같이 상태 다이어그램을 이용하여 설계하였다. 하지만, 상태 다이어그램은 현재의 상태와 이벤트에 따른 다음 상태만을 표시하기 때문에 세부적인 행동을 나타내기에는 제약이 많으므로, 이를 보완하기 위해 표 2와 같이 상태 테이블을 사용하였다.

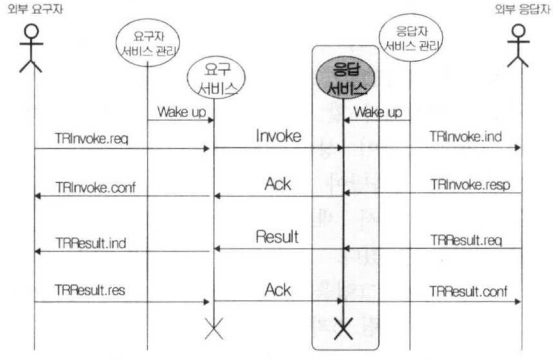


그림 7. 클래스 2 트랜잭션의 메시지 전달 절차

표 2. TIDOK_WAIT 상태에서의 입력 이벤트와 엔터티의 처리과정을 설명한 상태 테이블

WTP Responder TIDOK_WAIT			
Event	Condition	Action	Next State
TRInvoke_req	Class == 2 | 1 TIDok	Generate TRInvoke.ind Start timer, A	INVOKE_RESP_WAIT
RcvAbort		Send Abort PDU Abort Transaction	Exit
RcvInvoke	RID == 0	Ignore	TIDOK_WAIT
	RID == 1	Send Ack	TIDOK_WAIT

메시지 전달에 참여하는 클래스는 SDL의 프로세서 타입에 해당하며 그림 9와 같이 자신의 프로세서 다이어그램을 가진다. 한편 상태 다이어그램으로 분석된 동적 행위모델과 프로세서 다이어그램은 현

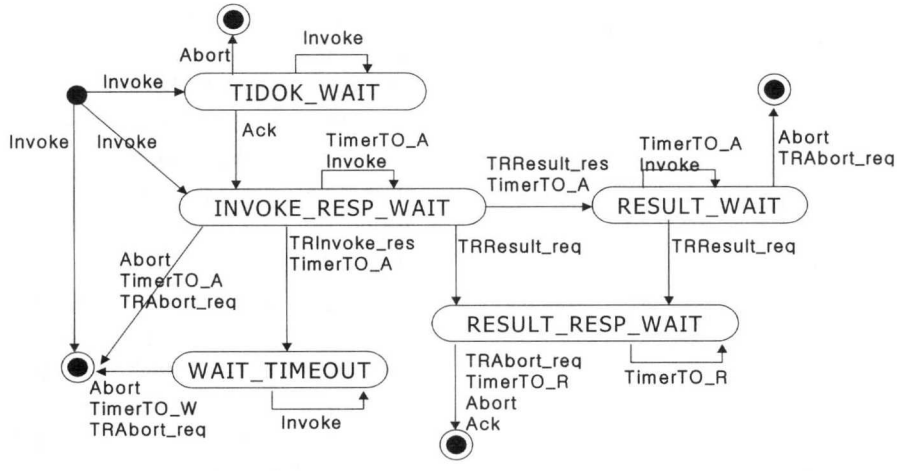


그림 8. 그림 7에서의 응답 서비스 엔터티에 해당하는 동적 행위 설계

재 상태와 입력에 의한 다음 상태로의 전환이라는 표현 형식의 공통점이 존재한다. 이를 이용하여 그림 9와 같은 기본적인 프로세서 다이어그램을 구성한 후 표 2와 같이 세부적인 동적 행위를 분석한 자료를 바탕으로 그림 10과 같이 프로세서 다이어그램을 완성하였다.

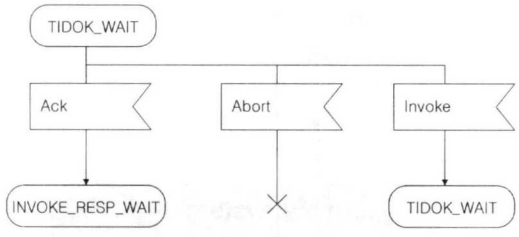


그림 9. TIDOK_WAIT 상태에서의 SDL 프로세서 다이어그램

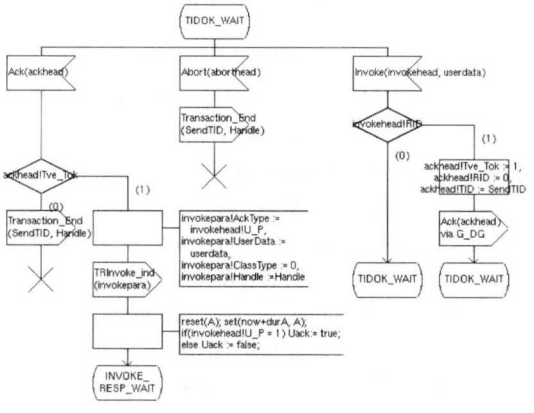


그림 10. TIDOK_WAIT 상태에서 입력 이벤트에 대한 처리과정 설계

Ⅲ. 프로토타입의 확장성 분석

3.1. 프로토콜 구현구조의 비교

성능 측정을 위한 어플리케이션은 테스트 어플리케이션, WTP 프로토콜, WDP/UDP 계층으로 이루어진다. 그리고 테스트 어플리케이션은 파일입력으로 초기값을 설정하고, 성능측정 결과를 파일로 출력하며, WDP/UDP 계층은 시스템에서 제공하는 소켓 인터페이스를 통해 네트워크에 접속한다. 구현환경은 Linux(version 2.2.14) 운영체제에서 C 코드를 이용하였다.

프로토콜을 구현하기 위한 기본적인 모델은 서버 모델, coroutine 모델, activity-thread 모델이 있으며, SDT에 의해 생성된 프로토콜은 서버 모델에 안정성과 다양한 어플리케이션으로의 확장을 위한 기능을 첨가한 확장된 서버 모델을 사용한다. 이들간의 성능을 비교하기 위하여 각 모델별로 성능 측정을 위한 모듈을 추가한 프로토콜을 구현하였다.

서버 모델에서 프로토콜 구조는 서버라 불리는 몇 개의 실행 단위로 이루어지며, 이들은 각각의 프로토콜 계층을 담당한다. 그리고, 서버들의 실행과 이들간의 통신은 비동기적으로 이루어진다. 그림 11은 서버 모델을 이용한 프로토콜 엔진(이하 WTPser)의 구성도이다. 그림과 같이 각 계층을 하나의 스래드로 구성하였으며, 메시지 전달을 위하여 상·하로 수신 버퍼를 두었다. 그리고, 이벤트 교환을 위하여 각 스래드에 파이프를 할당하였다.

그림 12는 coroutine 모델을 적용한 프로토콜 엔진의 구성도(이하 WTPcor)이다. 이 모델은 기본적인 형태가 서버 모델을 따르지만, OS의 스케줄러가 아니라 핸들러가 다음에 실행할 coroutine을 결정한다. 이를 위하여 각각의 계층을 하나의 함수 형태로 구성하였으며, 핸들러는 외부환경 혹은 프로토콜의 자체 기능에 의해 메시지가 발생하면 이를 처리하기 위한 coroutine을 호출하도록 구현하였다.

그림 13은 activity-thread 모델을 적용한 프로토콜 엔진의 구성도(이하 WTPact)이다. 이 모델에서 각 프로토콜 계층은 이벤트를 처리하는 프로시저의 집합으로 구성하였으며, 각 계층간의 메시지 교환은 해당 프로시저를 직접 호출하고 파라메타의 형태로 메시지를 전달한다. 메시지의 전달 방향에 따라 상향과 하향 스래드를 두었으며, 두 스래드간 메시지 전달이나 외부로부터의 PDU 전달을 위한 인터페이스로 각 스래드에 파이프를 할당하였다.

이상 세가지 모델의 특성을 정리하면 표 3과 같다.

표 3. 프로토콜 구현 구조별 특성

구분	Coroutine 모델	Server 모델	Activity-thread 모델
실행 단위	Coroutine (일종의 프로시저)	서버 (스래도 또는 프로세서)	스래드
실행 방법	하나의 프로세서가 coroutine 호출	각각의 서버가 비동기적으로 수행	이벤트에 의해 스래드가 동기화
이벤트 발생	프로시저 호출	IPC	IPC와 함수호출
데이타 전송	메시지 버퍼	메시지 버퍼	함수의 파라메타

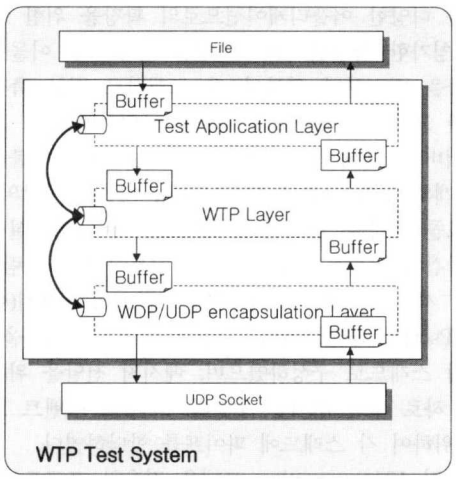


그림 11. 서버 모델을 이용한 WTP 프로토콜 엔진(WTPser)의 구조

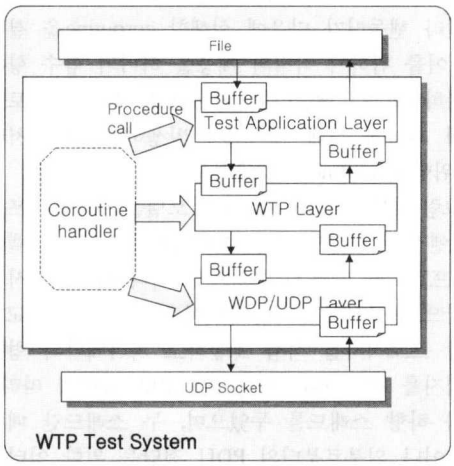


그림 12. Coroutine 모델을 이용한 WTP 프로토콜 엔진(WTPcor)의 구조

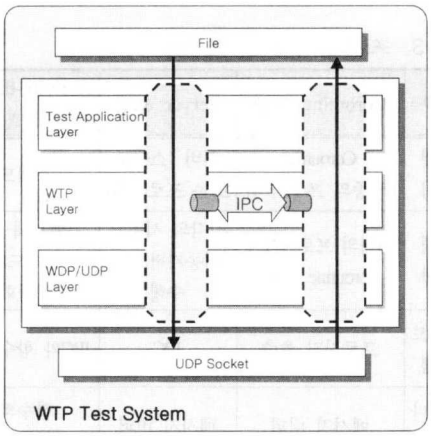


그림 13. Activity-thread 모델을 이용한 WTP 프로토콜 엔진(WTPact)의 구조

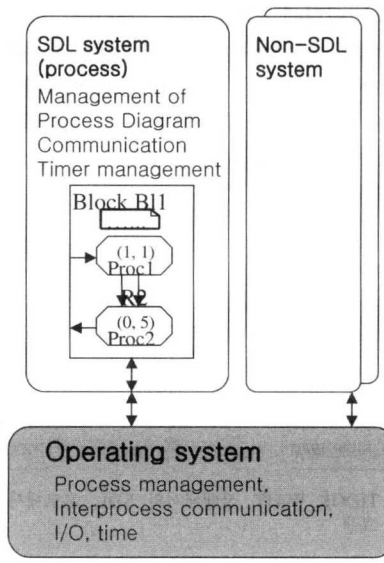


그림 14. SDT로 구현한 WTP 프로토콜 엔진(WTPsdt)의 구조

마지막으로, UML 및 SDL을 이용해 설계한 결과물을 바탕으로 코드 자동 생성기에 의해 생성된 프로토콜 엔진(이하 WTPsdt)은 그림 14와 같이 하나의 프로세서로 동작한다. 이는 시스템이 지원하는 프로세서 내에서 SDL 다이어그램의 프로세서에 해당하는 프로세서 인스턴스(process instance)가 동작하는 일종의 coroutine 모델에 해당한다. [7]하지만 코드 자동 생성기는 다양한 시스템을 구현할 수 있도록 범용성 및 안정성을 강화하고 있어서, 생성된 코드의 크기가 더 크고 시스템의 메모리와 CPU의 사용량이 앞의 세 모델에 비하여 높다.

3.2. 실험환경 구성

본 실험에서는 앞에서 설명한 네가지 프로토콜 엔진의 성능을 비교하기 위하여 클라이언트의 증가에 따른 클래스 2 트랜잭션의 처리율(throughput)과 트랜잭션 처리 지연시간(system response time)을 측정하였다. 실험을 위한 클라이언트는 동등한 성능을 제공하기 위하여 WTPcor 엔진을 사용하였다. 또한, 네가지 엔진을 사용한 각각의 테스트 서버를 구현하였다. 네트워크 환경은 100Mbps 이더넷을 사용하였으며 에러 모델은 설정하지 않았다. 실험은 1,000~7,000개의 클라이언트를 1,000개 단위로 증가하면서 각 실험당 20분씩 모델별 성능을 측정하였다. [11-15]클라이언트와 서버의 트래픽 모델링 환경은 표 4와 같다.

표 4. 트래픽 생성 모델

Process	Parameters	Distribution
Request size	min = 36bytes max = 80bytes	uniform
Result size	shape = 1.2 scale = 101bytes max = 1500bytes	truncated pareto
Transaction inter arrival time	shape = 1.5 scale = 17sec max = 46800sec	truncated pareto

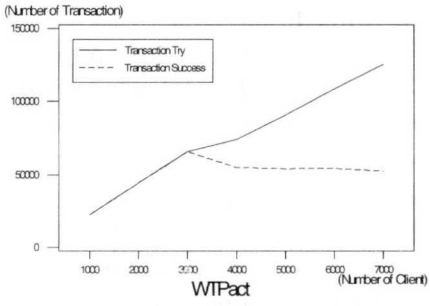
표 5. 코드 생성기로 생성된 테스트 서버와 WTP 프로토콜의 비교

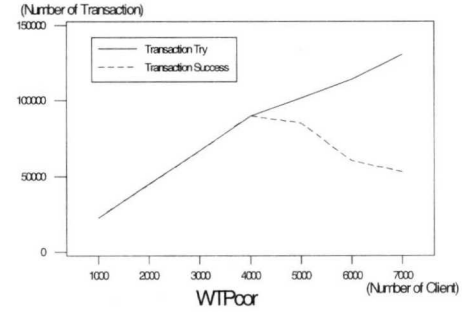
구 분	실행코드 크 기	실 행 상 태	
		메모리 점유율 (257664Kbte 기준)	동기화 스래드 개수
WTPact	60738 Byte	8.2%	3개
WTPcor	59647 Byte	8.0%	1개
WTPser	61782 Byte	8.0%	3개
WTPsdt	97204 Byte	17.9%	1개

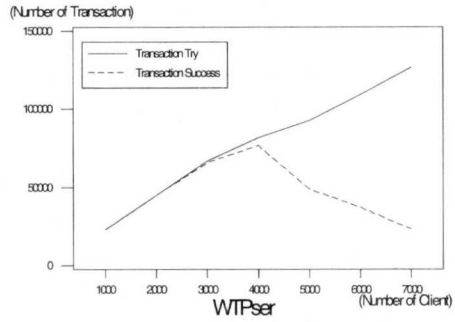
컴파일 환경: Linux 2.2.14, gcc컴파일, 옵션 -O
실행환경 : Memory 257664Kbyte Pentium II 500 Dual CPU

3.3. 성능측정 및 분석

표 5는 프로토콜 엔진의 코드 크기와 실행상태를 비교하고 있다. WTPsdt는 실행코드의 크기가 약 60Kbyte인 비교 대상 엔진들에 비하여 약 62% 증가하였으며, 실행시 메모리 점유율도 다른 모델에 비해 두 배 이상 크다. 하지만, 동기화된 스래드의 개수는 WTPact와 WTPser에 비하여 적다. 이는, WTPact가 입력 처리 스래드와 출력 처리 스래드, 그리고 타이머 스래드로 구성되며, WTPser는 프로토콜 계층 및 타이머 스래드로 구성되만, WTPcor와 WTPsdt는 하나의 프로세서 내에서 모든일을 처리하므로 한개의 동기화 스래드만으로 구성되기 때문이다.







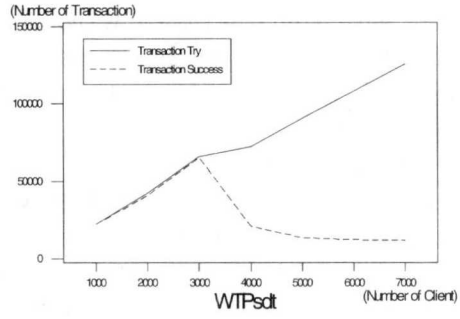


그림 15. 클라이언트 수에 따른 트랜잭션 시도 및 성공회수

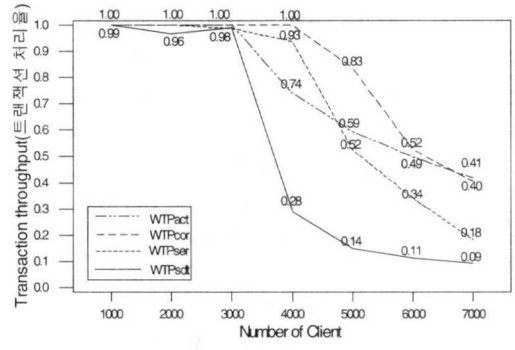


그림 16. 클라이언트 수에 따른 엔진별 트랜잭션 처리율

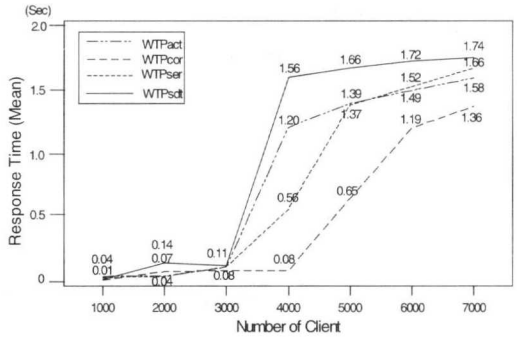


그림 17. 엔진별 트랜잭션의 처리 지연시간

다음으로는 하나의 서버에 서비스를 요청하는 클라이언트의 개수 증가에 따른 트랜잭션의 처리율과 트랜잭션의 처리 지연시간을 측정하였다.

그림 15는 각 모델별 클라이언트 수의 증가에 따른 총 트랜잭션 시도 회수와 성공 회수의 그래프이며, 그림 16은 성공회수를 시도회수로 나눈 트랜잭션 처리율의 그래프이다. WTPact는 3,000개의 클라이언트에 의해 발생되는 트랜잭션을 처리하며, 클라이언트의 개수가 증가하여도 약 50,000회의 트랜잭션을 처리하고 있다. WTPcor과 WTPser는 4,000개의 클라이언트까지 트랜잭션의 약 90% 이상을 처리해 주고 있으나, 그 이후에는 처리율이 급격히 떨어지고 있다.

WTPsdt는 2,000개의 클라이언트 접속에서부터 트랜잭션의 실패가 나타나고 있으나, 3,000개 이하의 클라이언트가 접속할 경우에는 트랜잭션의 처리율과 처리 지연시간이 다른 모델의 프로토콜 구현물과 대등함을 알 수 있다. 그러나 5,000개의 클라이언트에 이르면 트랜잭션의 처리율이 10%까지 급속히 감소하며, 트랜잭션의 성공 개수도 낮아졌다. 또한, 그림 19에서 보는바와 같이 트랜잭션 처리 지연시간은 1,500ms까지 증가하였는데, 이는 프로토콜 개발도구를 사용한 경우에 구현물의 크기가 약 62% 증가하면서 프로토콜 처리시간 증가로 인한 것이다.

한편, WTPser도 클라이언트의 수가 7,000개에 접근하면서 트랜잭션의 처리율이 18%까지 떨어졌다. 이에 비해 WTPcor과 WTPact는 비록 트랜잭션 처리율이 낮아지고 있지만 트랜잭션의 성공회수는 50,000회를 넘어서고 있다. 이러한 결과는 모델의 차이에 의한 것으로 분석된다. WTPact는 나머지 세 모델과 달리 하나의 입력에 대한 처리가 일괄적을 이루어지며, 처리 능력을 넘어서는 클라이언트의 트랜잭션 요청은 바로 취소된다. 그리고, WTPcor는 비록 메시지 큐를 사용하지만 각 coroutine별로 입력된 메시지를 일괄 처리한 후에 다음 coroutine의 처리가 진행되므로 입력된 메시지에 대한 즉각적인 처리가 가능하다. 따라서 일정 개수 이상의 트랜잭션을 지속적으로 처리해 주고 있다. 하지만 WTPser과 WTPsdt는 서버모델을 기반으로 하기 때문에 클라이언트로부터 입력된 메시지는 큐에 쌓인 후 순서대로 처리된다. 따라서 많은 클라이언트로부터 전송된 메시지가 큐에서 대기하게 되어 프로토콜이 시간 내에 처리할 수 있는 트랜잭션의 한도를 넘어설 경우에는 트랜잭션이 클라이언트로부터 요청된 시간을 초과한 후에 처리를 하게된다. 이는 클라이언트의 서비스 요청 메시지가 프로토콜 모듈 내의 메시지 큐에서 대기하고 있으나, 전송 시스템이 이 메시지를 적정시간 내에 처리하지 못하기 때문이다.

그림 17은 트랜잭션의 평균 시스템 응답시간으로 클라이언트가 하나의 트랜잭션을 수행할 때 프로토콜로부터 트랜잭션의 성공, 혹은 취소 여부를 인지하기까지의 시간에 해당한다. 무선 트랜잭션 프로토콜은 하나의 트랜잭션이 일정시간 내에 성공적으로 종료하거나 실패하기 때문에 시스템의 응답시간은 트랜잭션 처리율의 하락에 영향을 받지 않는다. 하지만 트랜잭션의 처리율이 떨어질수록 실패하는 트랜잭션이 많이 증가하며, 실패를 프로토콜이 확인하기 위해서는 일정 시간이 소모되기 때문에 그림 19와 같이 시스템의 응답시간이 클라이언트의 수에 비례하여 증가한다.

Ⅳ. 결 론

본 논문에서는 UML과 SDL을 이용하여 무선 트랜잭션 프로토콜의 엔진을 설계하고, 목적코드 자동생성도구를 이용하여 이를 구현하였다. 또한, 대표적인 세가지의 구현 모델을 적용한 프로토콜 엔진과 목적코드 자동 생성도구로 만든 엔진의 특성 및 성능을 분석하였다.

구현물의 성능을 분석한 결과, 3,000개 이하의 클라이언트가 동시에 접속할 경우에는 트랜잭션처리율과 트랜잭션 처리 지연시간 측면에서 기존의 세 가지 모델을 이용하여 직접 개발한 프로토콜 엔진과 그 성능이 대등함을 알 수 있었다. 또한, 5,000개 이상의 클라이언트가 동시에 접속할 경우 트랜잭션 성공률은 약 10%까지 급격히 감소하고 트랜잭션 처리 지연시간은 1,500ms까지 증가하였는데, 이는

프로토콜 개발도구를 사용한 경우에 구현물의 크기가 약 62% 증가하면서 프로토콜 처리시간 증가로 인한 것이다. 그러나, 이는 실험에 사용된 PC 서버의 사양을 고려할 때, 절대적인 호스트 과부하에 기인한 성능 감소로 판단된다. 따라서, 부하분산 기능이 제공되는 실제 환경에서는 직접 개발한 프로토콜 엔진과 대등한 성능을 기대할 수 있으며, 빠른 프로토타입의 생성, 프로토콜의 사전 검증 등과 같은 부가적인 이점도 제공받을 수 있다.

참 고 문 헌

[1] WAP Forum, "Wireless Application Protocol Architecture Specification," URL: http://www.wapforum.org/, November 1999.

[2] 김기조, 최윤석, 최은정, 임경식, "무선 응용 프로토콜 기술," 한국정보처리학회, 정보처리학회지, pp.44-56. May 2000.

[3] WAP Forum, "Wireless Transaction Protocol Specification," ,URL: http://www.wapforum.org/, November 1999.

[4] Object Management Group, "What Is OMG-UML and Why is it Improtant?," URL : http://cgi.omg.org/news/pr97/umlprimer.html.

[5] Jan Ellsberger, Dieter Hogrefe and Amardeo Sarma, "SDL Formal Object-Oriented Language for Communicating Systems," Prentice Hall, 1997.

[6] 한국전자통신연구원, *정보통신프로토콜공학*, 한국전자통신연구원, October 1994.

[7] Mitschele-Thiel, *System Engineering with SDL*, John Wiley & Sons Inc., 2001.

[8] Telelogic, *TAU/SDT 3.3. Telelogic*, June 1998.

[9] L. Svobodova, "Implementing OSI Systems," IEEE Journal on Selected Areas in Communications, vol. 7, pp. 1115-1130, September 1989.

[10] D. C. Schmidt and T. Suda, "Transport System Architecture Services for High-Performance Communications Systems," IEEE Journal on Selected Areas in Communications, vol. 11, pp. 489-506, May 1993.

[11] Thomas Kunz, et al., "WAP Traffic: Description and Comparison to WWW Traffic," Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems, pp. 11-19, August 2000.

[12] Hyoung-Kee Choi and J. O. Limb, "A Behavioral Model of Web Traffic," Seventh International Conference on Network Protocols, pp. 327-334, November 1999.

[13] R. Kalden, I. Meirick and M. Meyer, "Wireless Internet Access Based on GPRS," IEEE Personal Communications, vol. 7, pp. 8-18, April 2000.

[14] M. Poza and M. Iracheta, "On the Quest of a Better World Wide Web Traffic Model for UMTS," First International Conference on 3G Mobile Communication Technologies, pp. 456-460, November 2000.

[15] Yingxin Zhou and Zhanhong Lu, "Utilizing OPNET to Design and Implement WAP Service," OPNETWORK 2001, August 2001.

정 호 원(Howon Jung)

2000년 : 경북대학교 통계학과 (이학사)
2002년 : 경북대학교 컴퓨터과학과(이학석사)
2002년~현재 : 삼성전자
<주관심 분야> 이동 컴퓨팅, SDL, 무선 트랜잭션 프로토콜

임 경 식(Kyungshik Lim)

1982년 : 경북대학교 전자공학과 (공학사)
1985년 : 한국과학기술원 전산학과 (공학석사)
1994년 : University of Florida 전산학과 (공학박사)
1985년~1998년 : 한국전자통신연구원 책임연구원, 실장
1998년~현재 : 경북대학교 컴퓨터과학과 조교수
<주관심 분야> 이동 컴퓨팅, 무선인터넷, 홈 네트워킹, 컴퓨터통신