형식 언어 기반 임베디드 소프트웨어 개발 기법의 효율성 연구

설진호[O] 남영호[+] 박재흥[++]
[O++]경상대학교 컴퓨터과학과, [+]경상대학교 컴퓨터교육과
jinoboy@dreamwiz.com[O], yhnam@nongae.gsnu.ac.kr[+], pjh@gsnu.ac.kr[++]

A Study on Efficiency of Embedded Software Development Method Based on Formal Language

Jinho Seol[O] Yuongho Nam[+] Jaeheung Park[++]
[O++]Gyeongsang National University of Computer Science
[+]Gyeongsang National University of Computer Education

요 약

하드웨어의 성능향상으로 임베디드 통신 소프트웨어의 복잡도가 크게 증가되고, 이에 따르는 추가적인 개발 비용이 발생하고 있다. 개발자들은 임베디드 통신 소프트웨어의 복잡도를 해결하는데 필요한 소프트웨어 도구와 기술을 필요로 하고 있다. 본 논문에서는 임베디드 통신 소프트웨어 개발에 효과적인 SDL(Specification Description Language) 기반의 통합 개발도구인 SDT(SDL Design Tool)를 사용하여 소프트웨어를 개발하는 절차에 대하여 기술한다. 그리고 통신 소프트웨어인 ITU-T V.76 프로토콜에 개발 절차를 적용하여, SDT를 사용한 개발 절차와 일반적인 임베디드 통신 소프트웨어 개발 절차를 비교, 분석한다. 그 결과, SDT를 사용한 개발 절차가 개발 효율성과 유지보수 효율성에서 더 좋은 성능을 보였다.

1. 서 론

프로세스, 하드웨어 디바이스의 성능 향상으로 임베디드 통신 시스템은 소형화, 경량화, 고성능화, 다기능화 되어가고 있다. 이로 인해 통신 소프트웨어의 중요성은 점차 높아지고 복잡도가 크게 증가하여, 소프트웨어의 개발 기간이 늘어남으로써 추가적인 비용이 발생하고 있다. 이에 개발자들은 임베디드 통신 소프트웨어의 복잡도와 추가적인 비용을 해결하는데 필요한 소프트웨어 도구와 기술을 원하고 있다.

임베디드 통신 소프트웨어는 실시간성의 요구, 표준 규격의 잦은 갱신 및 변경, 새로운 기능이 빈번히 추가되는 특성을 가지고 있다. 이러한 특성 때문에 개발자들은 통신 소프트웨어의 설계, 구현 및 검증을 위한 설계 자동화 도구 및 테스트 자동화 도구를 필요로 하며, 이렇게 개발된 소프트웨어를 타겟에 빠르게 다운로드 할 수 있는 도구를 필요로 하고 있다.

본 논문에서는 형식 언어 SDL(Specification Description Language)[1]기반의 임베디드 통신 소프트웨어 개발 도구인 SDT(SDL Design Tool)[2]를 사용한 개발 절차와 일반적인 개발 절차[3]에 대하여 설명한다. 그리고 통신 소프트웨어인 ITU-T의 V.76[4] 프로토콜을 개발하기 위하여 두 개발 절차를 적용하고, 각 절차 별로 개발된 통신 소프트웨어를 ISO/IEC 9126[5]와 IEEE1061[6]에 근간한 몇몇의 측정법으로 각각의 효율성을 비교, 분석하였다.

본 논문의 구성은 다음과 같다. 2장에서는 V.76 프로토콜의 개발에 SDT를 사용한 개발 절차와 일반적인 개발 절차를 적용하고, 3장에서는 V.76 프로토콜 개발에 대한 두 개발 절차의 결과를 비교, 분석하고, 마지막으로 4장에서는 결론 및 향후 과제를 제시한다.

2. 임베디드 소프트웨어 개발 절차

본 논문에서는 ITU-T의 V.42 LAPM 기반의 멀티플렉서인 V.76 프로토콜을 SDT를 사용한 개발 절차와 일반적인 개발 절차로 각각 개발 하였다. V.76 프로토콜의 개발을 위한 호스트 환경과 타겟 환경의 플랫폼은 두 개발 절차 모두 win32 운영체제를 사용하였다.

2.1 SDT를 사용한 개발 절차

V.76 프로토콜을 개발하기 위한 SDT를 사용한 개발 절차는 다음과 같다.

1단계 : 요구사항 분석 (Requirement analysis)
2단계 : 시스템 명세 (System specification)
3단계 : 검증 (Verification)
4단계 : 코드 생성 (Code generation)
5단계 : 타겟 통합 (Target integration)
6단계 : 테스팅 (Testing)

먼저 V.76 프로토콜 설계 규격서의 요구사항을 분석하여 SDL로 시스템을 명세한다. 시스템 명세 단계는 SDL의 개발도구인 SDT에서 제시하는 개발 방법론에 따라 시스템, 블록, 프로세스 레벨 순으로 계층화하여 명세한다. 그림 1은 SDL로 명세된 V.76 프로토콜의 전체 시스템 개관을 나타낸다. 시스템 레벨은 그림 1과 같이 하나의 V76_DLC 블록으로 구성되고 환경과 통신하는 DLC_SU_A 채널과 데이터 링크 계층과 통신하는 DLC_DL 채널로 연결된다. 블록 레벨에서 V76_DLC 블록은 정적 프로세스인 dispatch와 동적 프로세스인 DLC로 구성된다. 최초 V.76 시스템이 실행되면 하나의 dispatch 프로세스가 생성되고 DLC 프로세스는 dispatch 프로세스에 의해 동적으로 생성된다.

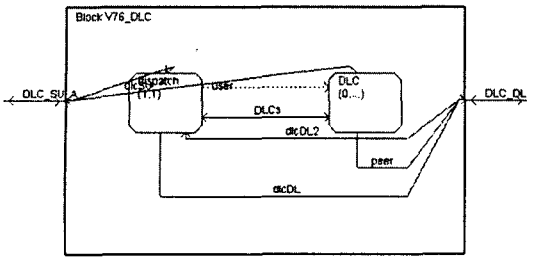

그림 1 V.76 시스템 개관

프로세스 레벨에서 프로세스는 시스템에서 실제 동작을 명세하는 부분이다. dispatch 프로세스는 서비스 사용자로부터 L_DataReq, L_ReleaseReq, L_EstabReq, L_Setparm Req 등과 같은 시그널을 수신한다. 각 시그널에 따라 정보의 전송, 데이터 링크 연결의 해체와 같은 동작을 수행한다. DLC 프로세스는 그림 2와 같이 dispatch 프로세스에서 전달되는 V76frame, L_DataReq, L_ReleaseReq 등과 같은 시그널을 수신하여 명세된 동작을 실행한다.

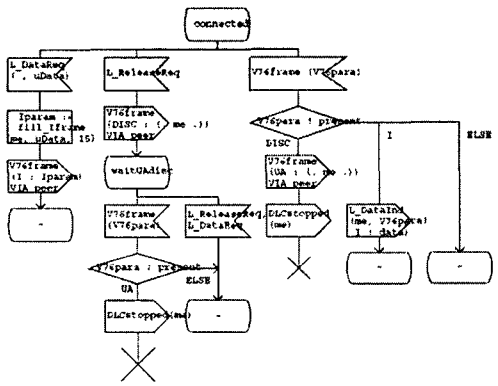

그림 2 SDL로 명세된 DLC 프로세스의 일부

검증은 명세된 V.76 시스템을 의미, 구문 분석기(analyzer)를 통하여 SDL 구문 상에서 발생할 수 있는 구문적, 의미적 오류를 검출하여 수정한다. 그리고 SDL로 명세된 시스템을 Simulation과 Validation을 수행하는 단계를 거친다. 시뮬레이션은 SDT에서 제공되는 Simulator를 통해 수행하여 명세된 시스템이 요구사항 명세서에 제시된 기능대로 동작하는지를 검증하고, 실행 시간 동안에 발생할 수 있는 여러 가지 오류를 검출하여 수정한다. 이 단계에서는 MSC(Message Sequence Chart)[7]를 통해 시스템의 행동을 도식적으로 검증 할 수 있다. 다음 단계는 Validator를 통해 검증을 시행하여 교착상태, 배열의 범위 초과 오류, 프로세스 생성 오류, 신호 출력 오류와 같은 SDL 시스템 개발의 치명적인 설계 오류나 비일관성 문제 등을 찾아 수정하고 시스템의 견고성을 증가시킨다[8].

SDL로 명세된 시스템의 검증 과정이 모두 끝나게 되면 SDT에서 제공되는 코드 생성기를 통해 코드를 생성 시킨다.

생성된 코드는 SDT에서 제공되는 Light 통합과 Tight 통합과 같은 몇몇의 통합 프레임워크를 통해 타겟에 통합될 수 있다[2]. 본 논문에서는 운영체제의 여러 프리미티브를 효과적으로 사용할 수 있는 Tight 통합을 적용하여 win32 운영체제에 통합하였다.

다음 단계는 통합된 소프트웨어가 요구사항을 잘 반영하는지, 기능들은 정상적으로 동작하는지 테스트한다.

SDT를 사용한 개발 절차는 SDL로 명세된 시스템을 다양한 방법으로 검증하여 개발 초기 단계에서 발생할 수 있는 오류를 제거할 수 있다. 그리고 2에서 7단계까지 모든 단계가 SDT에 의해 자동화되므로 전체 개발 절차를 SDT에서 일괄적으로 관리 할 수 있고 개발 중 이전 단계에서 발생된 오류를 수정하는데 시간 및 비용을 단축시킬 수 있는 장점이 있다.

2.2 일반적인 개발 절차

V.76 프로토콜을 개발하기 위한 일반적인 개발 절차는 다음과 같다.

1단계 : 요구사항 분석 (Requirement analysis)
2단계 : 명세 및 설계 (Specification and Design)
3단계 : 구현 (Implementation)
4단계 : 타겟 통합 (Target integration)
5단계 : 테스팅 (Testing)

요구사항 분석 단계는 V.76 프로토콜 설계 규격서의 요구사항에 대한 이해와 개발 절차 및 필요한 정보를 정의한다. 분석된 요구사항은 명세 단계를 통하여 요구사항 분석서, 프로세스 계층도, 프로세스 흐름도 등으로 명세된다[9]. 설계 단계는 기본 설계와 상세 설계로 나누어진다. 기본 설계는 요구사항을 통신 시스템의 구현 관점에서 '어떻게' 해결할 것인가를 결정하고 상세설계 단계는 기본 설계안을 토대로 프로그램을 위한 구조와 모듈을 설계한다. 설계 단계에서는 응용시스템 구조도, 응용시스템 흐름도, 시스템 구성도, 프로그램 간 흐름도, 프로그램 구조도와 같은 설계서를 작성한다. 구현 단계에서는 설계서를 바탕으로 프로그램 원시코드를 구현한다. 본 논문에서는 구현과 디버깅을 위하여 Microsoft의 Visual C/C++을 사용하였다. 구현 단계에서 작성된 원시코드는 단위테스트를 통하여 검증된다. 원시코드의 검증이 완료되면 작성된 원시코드는 실제 타겟 시스템에 통합되어 시스템에 대한 모든 요구사항분석 및 명세, 설계서의 내용들이 시스템에 정확히 반영되고 기능들이 정상적으로 수행되는지를 테스트한다.

일반적으로 임베디드 소프트웨어의 개발 환경은 소프트웨어를 개발하는 호스트 환경과 호스트환경에서 개발된 소프트웨어가 탑재되는 타겟 환경으로 나눠진다. 호스트 환경과 타겟 환경의 플랫폼이 틀린 경우 크로스 컴파일을 사용함으로 최종 시스템이 완성될 때까지 소프트웨어를 테스트 해볼 수 가 없다. 최악의 경우 요구 분석, 명세 단계에서 문제가 발생 하였을 때는 처음부터 다시 개발을 착수해야하는 위험이 생길 수도 있다. 이러한 문제를 해결하기 위해 SDL과 같은 실행 가능한 명세를 사용하여 요구 분석, 명세 상에서 발생 될 수 있는 문제를 개발 초기 단계에 검증하는 것은 중요하고 많은 비용을 절감할 수 있다.

3. 실험 및 비교 분석

3.1 실험

표 1은 SDT를 사용한 개발 절차에서 발생된 V.76 프로토콜 결과물을 열거한 것이다. 각 결과물의 "()" 안의 수는 실험 결과 발생한 문서 수 혹은 파일의 수를 나타낸 것이며, 비교 분석을 위해 사용된다.

단 계	결과물(수)
요구분석	통신프로토콜 표준안(1)
시스템 명세	통신 프로토콜 정형명세서(5) 기타 시스템 설계서(1)
검 증	Simulation 파일(3) Validation 파일(3) 기타 Log file(1)
코드생성	V.76 소스 및 기타 구현 파일(14) V.76 오브젝트 및 기타 오브젝트 파일(5) 최종 실행 파일(1)
총 합	(34)

표 1 SDT를 사용한 개발 절차의 결과물

일반적인 개발 절차에서 개발한 V.76 프로토콜의 결과물은 표 2와 같다.

단 계	결과물(수)
요구분석	통신 소프트웨어 표준안(1)
명 세	요구사항 분석서(1), 프로세스 계층도(1), 프로세스 흐름도(1)
설 계	응용시스템 구조도(1), 응용시스템 흐름도(1), 시스템 구성도(1), 프로그램 간 흐름도(1), 프로그램 정의서 (1), 프로그램 구조도(1)
구 현	V.76 소스 및 기타 구현 파일(4) V.76 오브젝트 및 기타 오브젝트 파일(2) 최종 파일(1)
총 합	(17)

표 2 일반적인 개발 절차의 결과물

3.2 비교 분석

두 개발 절차의 성능을 평가하기위해 ISO/IEC 9126, IEEE 1061에 근간하여 개발 효율성과 유지보수 효율성을 측정하고, 최종 실행 파일 크기를 비교, 분석한다.

개발 효율성은 소프트웨어 개발 절차에서 발생된 전체 결과물 중 비용이 발생되지 않는 결과물의 비율을 계산하여 소프트웨어 개발의 효율성을 측정하는 것이다. 그 계산식은 아래와 같다.

$$C = \frac{A-B}{A} \times 100$$

- A : 전체 결과물의 수
- B : 비용(노력)이 발생된 결과물의 수
- C : 비용(노력) 발생 절감 비율

유지보수 효율성은 소프트웨어 유지 보수 차원에서 변경되는 결과물 중에 비용이 발생되지 않는 결과물의 비율을 계산하여 유지보수의 효율성을 측정하는 것이다. 그 계산식은 아래와 같다.

$$C = \frac{A-B}{A} \times 100$$

- A : 변경되는 Output 수
- B : 변경 시 비용(노력)이 발생하는 결과물 수
- C : 비용(노력) 발생 절감 비율

개발 효율성에 대한 두 개발 절차의 측정결과 값은 표 3에서와 같이 SDT를 사용한 개발 절차가 약 30% 정도 더 많은 비용을 절감한다. 각 개발 단계별로 측정된 유지보수 효율성은 표 4에서처럼 요구분석, 명세 혹은 설계 단계에서 SDT를 사용한 개발 절차가 두 배 이상의 비용을 절감하였음을 볼 수 있다. 표 3의 실행 파일 크기는 타겟에 다운로드 되는 최종 실행 파일의 크기를 나타낸다. 현재 기술의 발달로 프로세스의 성능 면이나 메모리의 용량의 증가 면에서 실행 파일의 크기 차이는 근소하다고 볼 수 있다.

	일반적인 개발 절차	SDT를 사용한 개발 절차
개발 효율성	35.2941 %	64.7059 %
실행 파일 크기	40,960 byte	86,016 byte

표 3 개발 효율성과 최종 실행 파일 크기 측정 결과

유지보수 효율성	일반적인 개발 절차	SDT를 사용한 개발 절차
요구분석	20.0000 %	54.1667 %
명세 I 설계	36.3636 %	73.6842 %
구 현	80.0000 %	91.6667 %

표 4 각 단계별 유지보수 효율성 측정 결과

5. 결론 및 향후 과제

본 논문에서는 SDL을 기반으로 하는 통합 개발도구인 SDT를 사용한 임베디드 소프트웨어 개발 절차를 제시하고 이를 일반적인 개발 절차와 비교, 분석하였다. SDT를 사용함으로써 요구 분석상의 오류를 미리 제거하고, SDL로 명세된 소프트웨어의 정확성을 증진시켰다. 그리고 전체 개발 단계를 하나의 개발 도구에서 관리함으로써 소프트웨어 개발 일정을 단축시키고, 유지 보수로 인한 비용 및 노력을 절감시키는 효과적인 개발을 하였다.

향후 연구 과제는 SDT로 개발된 임베디드 통신 소프트웨어의 타겟에 통합되는 최종 실행 파일의 크기를 줄여 메모리 사용률에서도 최적화된 소프트웨어를 개발하는 것이다.

참고 문헌

[1] Z.100, "Specification and Description Language (SDL)", ITU, 1996.
[2] Telelogic AB, "Telelogic Tau 4.6 User's Manual", 2004.
[3] Ye Huang, Michael Hughes, "Using SDL in Embedded Systems Design : A Tool for Generating Real-Time OS pSOS based Embedded Systems Applications Software", 11th IEEE Workshop on Real-Time Operating Systems and Software, IEEE Computer Society Press, Los Alamitos, P39-43, 1994.
[4] V.76, "Generic multiplexer using V.42 LAPM-based procedures", ITU-T, 1996.
[5] ISO/IEC 9126, "Information Technology - Software Quality Characteristics and metrics - Part 1,2,3,4", 1999
[6] IEEE 1061, "Standard for a Software Quality Metrics Methodology", IEEE, 1998.
[7] Z.120, Message Sequence Chart(MSC), ITU, 1996.
[8] 강경훈, 남영호, 정광렬 "IMT-2000/비동기방식(3GPP) RRC 프로토콜의 SDL 설계 및 검증" 한국정보과학회 학술발표논문집, 28권, 1호, P442-444쪽, 2001.
[9] TTA.KO-11.0002, "분석단계 소프트웨어 문서 작성 지침", 한국정보통신기술협회, 1998.