원격 실시간 개발환경에서 디버그에이전트의 설계 및 구현

공기석*, 손승우*,김흥남*
*한국전자통신연구원 컴퓨터 · 소프트웨어기술연구소 S/W 공학연구부
e-mail : { kskong,swson, hnkim}@etri.re.kr

A Design and Implementation of Debug Agent for Real-time Remote Development Environment

Kisok Kong*, Seungwoo Son*,Heungnam Kim*
*S/W Engineering Dept., ETRI-Computer & Software Technology Lab.

요 약

인터넷 정보가전을 위한 내장형 실시간 응용프로그램을 개발하기 위해서는 개발도구의 지원이 필요하다. 이러한 도구들은 주로 원격개발환경에서 실행되는데, 디버그에이전트는 호스트 컴퓨터에서 수행되는 도구들의 요구를 타겟 시스템에서 실행하기 위한 타겟 상주형 태스크이다. 디버그에이전트는 도구들의 요구를 받아 이를 해석하고, 실행하며 그 결과를 호스트 컴퓨터로 전송한다. 호스트로부터의 요구들은 디버그 프로토콜로 정의된다. 이 논문에서는 실시간 응용프로그램 개발 환경을 위한 디버그에이전트의 구조와 기능을 제안한다. 타겟 독립성을 부여하며 최소한의 타겟 자원만을 요구하도록 설계된 디버그프로토콜에 대해서도 소개한다.

1. 서론

인터넷의 급속한 발전으로 인하여 웹 TV, PDA, 웹 폰 등의 많은 기기들이 인터넷에 직접 접속되기 시작하였다. 이러한 기기들은 복잡한 응용 프로그램을 수행하기 위하여 실시간 운영체제 (RTOS) 를 필요로 한다. 실시간 응용 프로그램 (또는 내장형 프로그램)의 개발은 디버거 등을 포함하는 적절한 개발 도구가 없기 때문에 어렵다 [1][2][3]. 현재 시장에서 판매되고 있는 많은 소프트웨어 개발도구 들은 실시간 특성을 지원하지 않기 때문에 실시간 시스템을 구축하는 데 효율적이지 않다. 실시간성을 지원하는 소프트웨어 개발도구 들도 메모리, CPU 성능, I/O 등의 면에서 자원이 매우 부족한 타겟 시스템에 과도한 부하를 거는 것으로 분석되고 있다 [4][5][6]. 따라서 현재의 인터넷 가전에서 실행되는 매우 복잡한 응용 프로그램을 개발하기 위해서는 타겟 시스템의 자원의 극히 일부만을 사용하면서 플랫폼 독립성을 제공하는 실시간 시스템 개발도구의 지원이 매우 중요하다 [7][8].

실시간 응용 프로그램은 이제 까지 주로 어셈블리 언어를 사용하여 개발되어 왔다. 디바이스 드라이버로부터 GUI 에 이르기까지 거의 대부분의 소프트웨어들이 RTOS 의 지원 없이 개발되었다. 그러나 오늘날의 복잡한 프로그램들은 OS 와 고급 언어의 지원을 요구한다. 원격개발환경은 자원이 부족한 타겟 시스템을 대신하여 호스트 시스템에서 고급언어를 사용하여 프로그램을 작성하고 디버거, 대화형 쉘, 자원 모니터 등의 도구를 사용하여 개발을 수행할 수 있는 환경으로서 내장형 실시간 시스템에 개발에 적절하다.

이러한 원격개발환경에서는 타겟 시스템에서 호스트의 도구들과 통신하면서 이들의 요구사항을 실행하는 프로그램이 존재하여야 하는데 본 논문에서는 이를 *디버그에이전트* (*debug agent*)라고 정의한다. 디버그에이전트는 디버거 및 다른 도구들의 요청을 받아 이를 해석하고 실행하며 그 결과를 호스트로 전송하는 역할을 수행한다.

본 논문에서는 원격 호스트에서 실시간 프로그램을 디버깅하기 위한 실시간 프로그램 개발환경을 제안한다. 디버그에이전트를 위한 타겟 상주형 모듈들의 설계와 구현에 관련된 내용을 기술한다. 디버그에이전트는 ETRI 에서 StrongARM CPU 보드를 타겟으로 제작하고 있는 “Qplus” RTOS 위에서 개발되고 있다

[9][10].

2 장에서는 원격개발환경에 대하여 설명하고, 3 장에서는 디버그 프로토콜을 제시한다. 4 장에서는 디버그 에이전트와 이를 구성하는 각 모듈의 기능에 대하여 설명한 후, 마지막으로 결론과 앞으로의 작업에 대하에 기술한다.

2. 원격개발환경 (Remote Development Environment)

그림 1 에서는 클라이언트/서버 구조를 가진 원격 실시간 소프트웨어 개발환경을 보여주고 있다 [4].

호스트 시스템에서의 *타겟서버*는 도구들 (디버거, 대화형 쉘, 자원 모니터 등) 과 타겟 시스템 간의 일종의 브로커 역할을 한다 [10]. 도구들은 요구사항을 미리 정의된 API 를 통하여 타겟 서버에게 보낸다. 타겟서버는 이 요구들을 디버그에이전트에게 보내어 타겟 상에서 특정한 함수를 실행하게 된다.

교차 컴파일러 (cross compiler) 는 고급 언어로 작성된 응용 프로그램을 타겟의 바이너리 코드로 생성한다. 소스레벨 디버거는 타겟 상에서 실행되는 코드를 호스트 상에서 소스 프로그램 레벨로 디버깅하도록 하여, 브레이크포인트를 설정하거나 디버깅 되고 있는 프로그램의 상태를 조회할 수 있도록 한다.

대화형 쉘은 명령어 쉘로서 호스트에서 타겟의 모든 런타임 (runtime) 기능들을 접근할 수 있도록 해준다. 쉘을 통하여 목적 코드를 타겟에 다운로드 할 수 있고, 특정한 변수들을 조회할 수 있으며, 지정한 루틴을 호출할 수 있고 태스크나 시스템의 정보를 살펴볼 수 있다.

자원 모니터는 타겟 시스템사의 자원들 (태스크, 메모리, 스택, 동기화 프리미티브 (synchnorization primitive) 등) 에 대한 감시기능을 제공한다.

디버그에이전트는 호스트에 있는 도구들로부터의 요구사항을 수용하기 위한 핵심적인 서비스들을 구현한 것이다. 타겟서버를 통하여 전송되는 이러한 요구들에는 메모리 트랜잭션, 브레이크포인트나 기타 이벤트를 위한 통보(notification) 서비스, 태스크 제어, 가상 I/O 에 관한 것들이 포함되어 있다.

디버그에이전트는 Remote Procedure Call (RPC) 기반의 메시지 프로토콜인 Qplus 디버그 인터페이스 (QDI) 를 지원하는 경량 UDP/IP 스택을 포함하고 있다 [11]. *QDI 프로토콜*은 ETRI 에서 수행되고 있는 조립형 실시간 OS 개발과제의 일환으로 개발되고 있다. 이 프로토콜은 타겟 독립성 (target independence)을 제공하며 최소한도의 타겟 자원만을 사용하도록 설계되었다. 여기에는 디버거를 포함한 호스트 도구들의 요구를 실행하기 위한 최소한도의 서비스들이 규정되어 있다.QDI 프로토콜은 이기종간의 데이터 전송을 위하여 Sun 의 XDR [12] 을 사용하고 있다.

3. 디버그 프로토콜

다음의 표 1 에서는 QDI 프로토콜과 그 내용을 설명하고 있다.

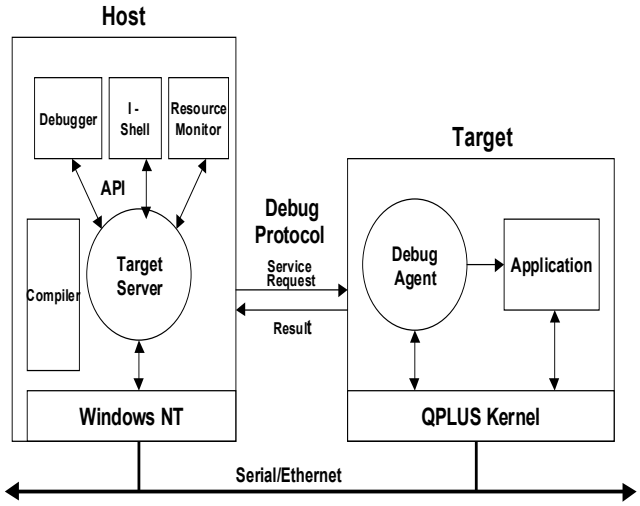

그림 1 원격 실시간 개발 환경

표 1 QDI 프로토콜

Protocol	Description
CONTEXT_CONT	Continue a context
CONTEXT_CREATE	Create a context on the target
CONTEXT_KILL	Kill a context on the target
CONTEXT_RESUME	Resume a context on the target
CONTEXT_STEP	Single step a context
CONTEXT_SUSPEND	Suspend a context on the target
DIRECT_CALL	Call a function in the target in the debug agent's context
EVENT_GET	Upload an event from the target
EVENTPOINT_ADD	Add an async. eventpoint
EVENTPOINT_DELETE	Delete an eventpoint
FUNC_CALL	Call a function in the target
MEM_CHECKSUM	Checksum a block of target memory
MEM_FILL	Fill target memory with a pattern
MEM_MOVE	Move memory on the target
MEM_CACHE_TEXT_UPDATE	Synch. cache after writing to the target
MEM_READ	Read memory from the target
MEM_SCAN	Scan a block of target memory for a pattern
MEM_WRITE	Write the contents of host memory to target memory
REGS_GET	Get the registers of a context on the target
REGS_SET	Set the registers of a context on the target
TARGET_CONNECT	Connect the target server to the debug agent
TARGET_DISCONNECT	Disconnect the target server from the debug agent
VIO_WRITE	Write data to a virtual I/O channel

표 1 에서 볼 수 있듯이 많은 프로토콜이 디버거와 관련된 것들이다. 여기에 대화형 쉘의 오브젝트 모듈 다운로드를 위한 메모리 제어 프로토콜이 추가되어 있다. 모듈을 다운로드하고 실행하기 위해서는 태스크 생성, 삭제, 정지 등의 태스크 제어를 위한 프로토콜도 필요하다.

호스트-타겟 통신을 위한 최하위 계층에는 Sun 의 RPC 가 사용된다 [11]. RPC 의 사용으로 호스트로부터 타겟의 서비스 요청은 동일한 기계 내에서의 함수 호출과 유사하게 된다. 함수의 입출력 인자들은 상이한 시스템간의 데이터 호환성 지원을 위하여 Sun 의 XDR 을 사용하여 인코딩된다. RPC 를 사용한 서비스의 호출은 다음과 같은 형식을 따른다:

```
RpcClntCall(Protocol_Name,Function_Name,
Input_Arguments, Output_Arguments)
```

4. 디버그에이전트의 구조와 기능

4.1 디버그에이전트의 구조

Qplus 의 디버그에이전트의 내부구조를 그림 2 에 나타내었다. 디버그에이전트는 12 개의 모듈로 구성되어 있는데, 각각의 모듈은 해당되는 QDI 프로토콜의 처리를 담당한다.

다음과 같은 메시지들이 타겟서버의 후단(backend)으로부터 전송된다.

- 태스크 컨텍스트의 생성, 삭제, 재개 등을 위한 요구
- 타겟 시스템상의 이벤트
- 타겟 메모리상에 적재된 모듈의 실행과 태스크의 종료에 대한 통보
- 타겟 메모리에 적재된 오브젝트 코드내의 특정한 함수의 호출
- 타겟상의 특정한 레지스터와 메모리의 상태정보
- 타겟 시스템상에 콘솔이 없을 때 가상 I/O 를 사용하기 위한 요구

그림 2 의 각 모듈들은 각각의 타입에 따라 이러한 요구사항 들을 처리한다. 각 모듈들의 기능은 다음과 같다.

- **메인 모듈**: 디버그에이전트와 설치된 서비스 모듈들을 초기화하여 서비스를 시작할 준비를 한다.
- **컨텍스트 관리 모듈**: 타겟상의 태스크의 컨텍스트를 관리한다.
- **브레이크포인트 관리 모듈**: 이중 연결 리스트의 형태로 디버거가 설정한 브레이크포인트들을 관리한다.
- **함수 호출 관리 모듈**: 타겟서버에 의하여 요구된 함수를 새로운 태스크를 생성하여 실행한다.
- **예외 통보 모듈**: 예외상황 (exception) 을 호스트에 통보한다.
- **직접 호출 모듈**: 타겟 서버에 의하여 요구된 함수를 디버그에이전트의 컨텍스트 내에서 호출한다.

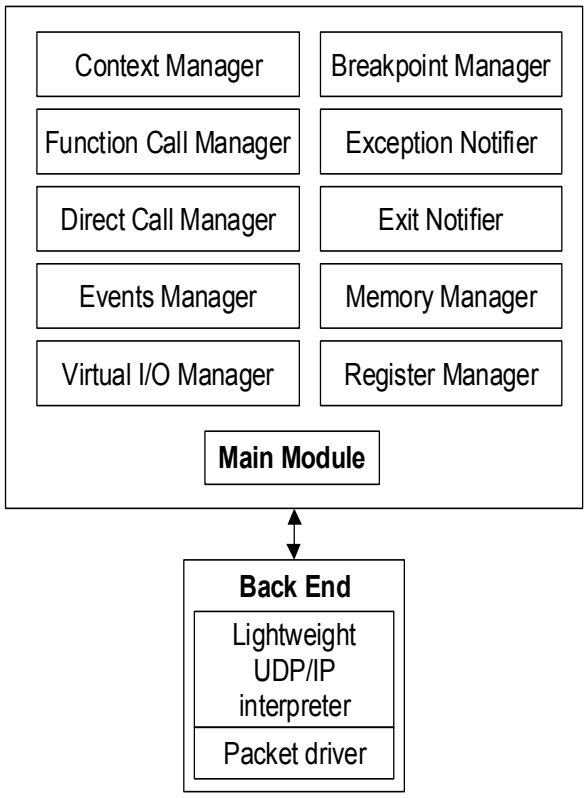

그림 2 디버그에이전트의 구조

- **종료 통보 모듈**: 컨텍스트의 종료 (exit) 를 호스트에 알린다.
- **이벤트 관리 모듈**: 브레이크포인트, 비동기 함수 호출, 예외 등의 비동기 이벤트를 추가하거나 삭제한다.
- **메모리 관리 모듈**: 타겟 메모리에 대한 체섬 (checksum), 읽기, 쓰기, 채우기 등을 실행한다.
- **가상 I/O 관리 모듈**: 타겟상의 가상 입출력 채널에 데이터를 입출력하는 서비스를 제공한다. 데이타는 호스트 도구의 가상 터미날로 전송된다.
- **레지스터 관리 모듈**: 타겟상의 레지스터의 값을 읽거나 쓰는 동작을 수행한다.
- **백엔드 (backend) 모듈**: 물리적 네트웍을 통하여 UDP/IP 패킷을 전송한다. 타겟의 부족한 자원을 고려하여 최소한도의 메모리를 요구하는 UDP/IP 해석기를 사용한다.

4.2 디버그에이전트의 동작

그림 3 에서는 디버그에이전트의 메인 모듈의 동작을 나타내는 흐름도 (flowchart) 를 보여주고 있다. 메인 모듈은 호스트로부터의 서비스 요청을 기다리고 있다가 요청이 도착하면 그 요청을 받았다는 것을 호스트에 알리고 요청의 타입을 결정한다. 그 다음, 그 요청을 해당 모듈로 보낸다. 만약 타겟 시스템에서 예외상황이 발생했다면 예외 통보 모듈을 통하여 이를

호스트에 알리도록 한다. 또 태스크가 종료되었다면 종료 통보 모듈을 통하여 이를 호스트에 알리고 시작 상태로 되돌아 간다.

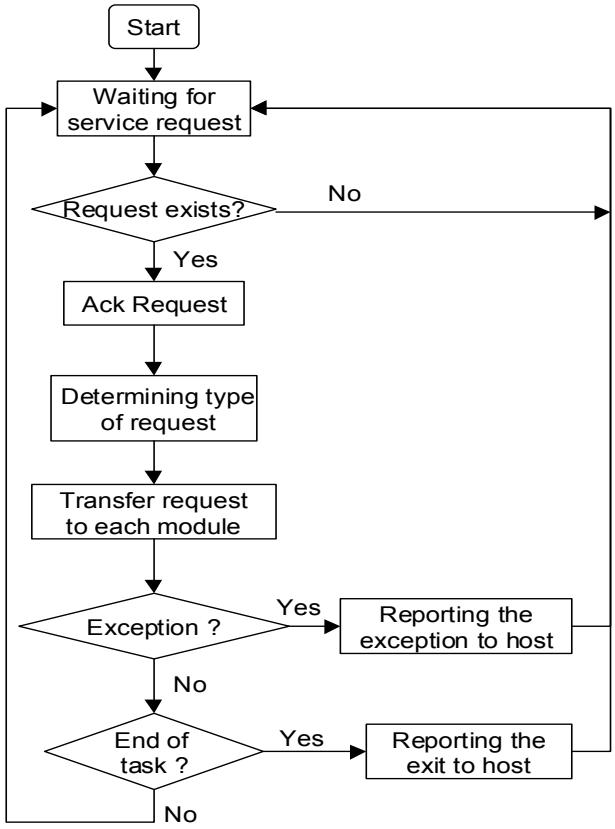

그림 3 메인 모듈의 동작

메인 모듈이 서비스 요청을 적절한 해당 모듈로 보냈을 때 이 서비스 모듈에서 이루어지는 처리 과정이 그림 4 에 나타나 있다. 서비스 요청 메시지가 수신되면 해당 서비스 모듈은 먼저 필요한 전처리 (pre-processing) 과정을 수행한 후, 타겟에 적재된 RTOS 에 커널 함수가 있는가를 찾는다. 만약 커널 함수가 존재한다면 이를 호출하고, 요구 받은 서비스를 실행한다. 예외 상황이 발생했으면 예외 관리 모듈을 통하여 이를 호스트에 알리도록 한 후, 태스크가 종료되었는가를 판단한다. 태스크가 종료되었으면 종료 통보 모듈을 통하여 호스트에 태스크의 상태를 알리도록 한다. 종료되지 않았으면 태스크의 실행 결과를 메인 모듈에 전달한다. 커널 함수가 존재하지 않은 경우는 태스크 종료 여부를 판단하는 단계로 바로 넘어간다.

디버거에서 브레이크포인트를 설정하였을 때 타겟 서버와 디버그에이전트, RTOS 커널에서 이루어 지는 일련의 동작 과정을 그림 5 에 보이고 있다.

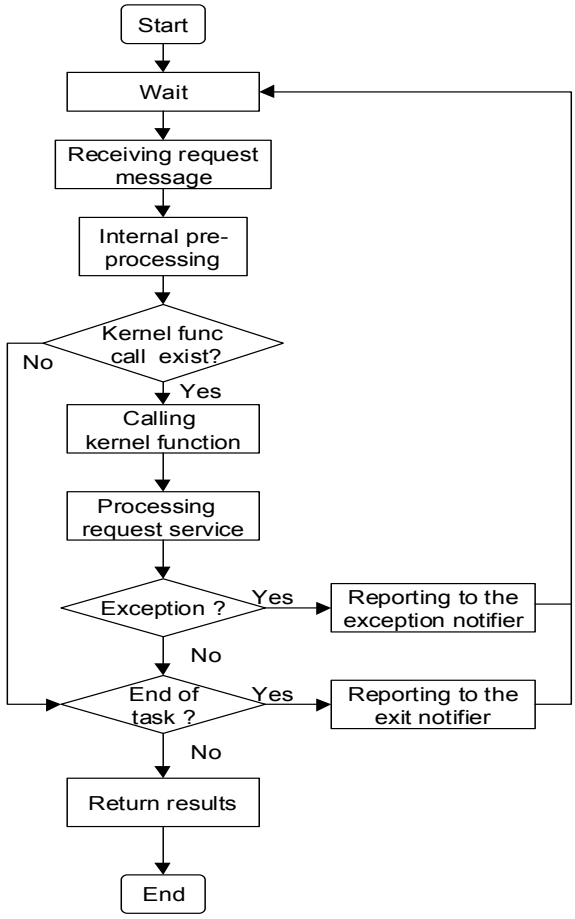

그림 4 서비스 모듈의 동작

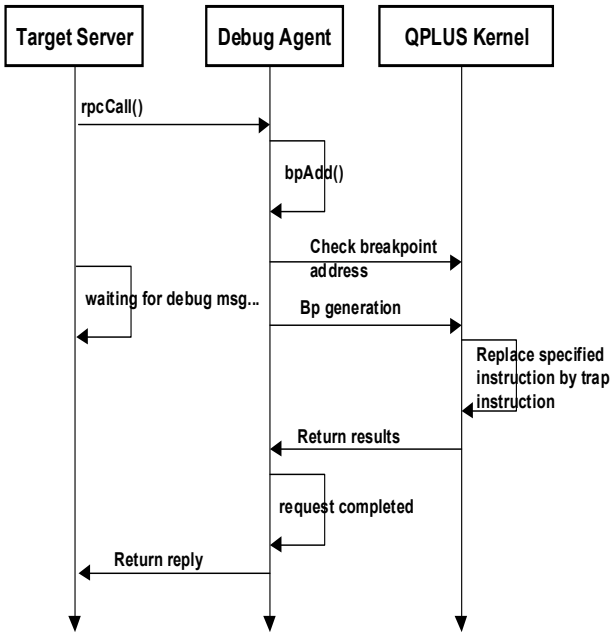

그림 5 브레이크포인트의 설정 과정

5. 결론 및 앞으로의 과제

본 논문에서 우리는 인터넷 가전기기 등의 내장형 실시간 시스템을 위한 소프트웨어의 개발환경에 대하여 기술하였다. 효과적인 원격개발을 위하여 ETRI 에서 현재 구현중인 디버그에이전트의 구조와 기능을 제시하였고 서비스 메시지를 구성하는 디버그 프로토콜인 QDI 프로토콜을 제안하였다.

타겟 독립적이고 최소한의 타겟 자원만을 사용하는 QDI 프로토콜의 사용을 통하여 본 디버그에이전트는 높은 이식성을 가지며 다양한 실시간 응용 프로그램 개발에 있어 생산성을 보다 높일 수 있다.

QDI 프로토콜의 효율성을 높이는 작업이 앞으로의 과제이다. 프로토콜의 오버헤드를 줄이는 것은 매우 중요한 연구과제로서 우선 프로토콜의 갯수를 줄이는 일과 호스트-타겟 간의 메시지 양을 줄이는 작업이 필요하다. 또 개발도구의 성능을 향상시키기 위한 디버그에이전트 구조개선 작업도 병행할 예정이다.

현재 디버그에이전트는 StrongARM 보드 [9] 상에서만 구현이 되어있다. 우리는 디버그에이전트의 이식성을 높이고 ETRI 에서 현재 개발중인 실시간 소프트웨어 개발도구 (“Esto” [10])의 범용성을 높이기 위하여 다양한 CPU 보드들에 이를 이식할 예정이다.

하드웨어 독립성 (hardware independence) 과 RTOS 독립성 (RTOS independence) 을 갖는 내장형 실시간 소프트웨어 개발 환경은 Post-PC 시대를 맞아 그 중요성을 더해갈 것이다.

참고문헌

[1] Jack G. Ganssle, “Debuggers for Modern Embedded Systems,” *Embedded Systems Programming*, Nov. 1998.

[2] Jonathan B. Rosenberg, *How Debuggers Work*, John Wiley & Sons, 1996.

[3] Hideyuki Tokuda and Makoto Kotera, “A Real-Time Tool Set for the ARTS Kernel,” *Proceedings of Real-Time Systems Symposium*, 1988.

[4] WindRiver, *Tornado User’s Guide,* 1995.

[5] WindRiver, *Tornado API Guide 1.0.1*, 1997.

[6] Microtec, *Spectra Boot and VRTX Real-Time OS*, 1996.

[7] Eun-Hyang Lee, et al., “A Cross Debugging Architecture for Switching Software,” *Proceedings of International Conference on Communication Technology* (ICCT), 1996.

[8] YoungJoon Byun, et al., “High-Level CHILL Debugging System in Cross-Development Environments,” *Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Processing*, 1998.

[9] Intel, *StrongARM EBSA-285 Evaluation Board*, Oct. 1998.

[10] C. Lim, et al., “A Tool Broker in Remote Development Environments for Embedded *Applications,” Proceedings of AI2000*, Feb. 2000.

[11] W. Richard Stevens, *Unix Network Programming*, Prentice Hall, 1994.

[12] Norishi Iga, et al., “Real-Time Software Development System RTIplus,” *Proceeding of the 12th TRON Project International Symposium*, 1995.