PS-Block 구조를 사용한 PS-Block Timing Model의 설계 및 구현

김 윤 관[†] · 신 원[††] · 김 태 완[†††] · 장 천 현[††††]

요 약

실시간 시스템은 소형 임베디드 시스템부터 대형 분산 시스템까지 다양한 시스템에서 사용되고 있다. 이러한 실시간 시스템은 시간적 정확성을 갖기 때문에 프로그램을 개발할 때 시간적 특성을 지키기 위한 노력이 필요하다. 실시간 시스템에 대한 연구 결과로서 제안된 TMO 모델은 실시간 개념에 따른 다양한 시간 처리 기능을 지원하고, 개발자가 정의한 응답시간을 보장한다. 따라서 개발자는 응답시간을 정의하고, 그 정확성을 확인하기 위한 기준점이 필요하다. 이를 위해 TMO 정적 분석 도구의 기반 구조로서 개선된 PS-Block을 설계하였다. 기존의 PS-Block은 블록을 중복 생성하는 구성 정책으로 부하가 생기는 문제점이 발생한다. 이에 본 논문에서는 블록의 중복으로 인한 부하문제를 개선하고, 탐색을 위한 베이스 클래스를 정의하여 PS-Block Timing Model을 구현하였다. 개선된 PS-Block 구조를 사용한 PS-Block Timing Model은 프로그램을 PS-Block 구성 정책에 따라 실행시간을 분석할 수 있는 기반을 제공하고 시간 정보 결정의 기준을 제공한다. 이를 통해 실시간 메소드의 적시성을 쉽게 확인하여 신뢰성을 향상시키고, 개발 기간을 단축할 수 있다.

키워드 : 실시간 프로그램, TMO 모델, 실행 시간 분석, 정적 분석 도구

Design and Implementation of PS-Block Timing Model Using PS-Block Structue

Yun Kwan Kim[†] · Won Shin[††] · Tae Wan Kim[†††] · Chun Hyon Chang[††††]

ABSTRACT

A real-time system is used for various systems from small embedded systems to distributed enterprise systems. Because it has a characteristic that provides a service on time, developers should make efforts to keep this property about time when developing real-time applications. As the result of research about real-time system indicates, TMO model supports various functions for time processing according to the real-time concept. And it guarantees response time which developers defined. So developers need a point of reference to define deadline and check the correctness of time. This paper proposes an improved PS-Block as an infrastructure of analysis tools for TMO to present a point of reference. There is a problem that the existing PS-Block has overhead caused by a policy making duplicated blocks. As such, this paper implements a PS-Block Timing Model to reduce the overhead due to block duplication, and defines a base class for searching in PS-Block. The PS-Block Timing Model, using an improved PS-Block structure, offers a point of reference of deadline and an infrastructure of execution time analysis according to the PS-Block configuration policy. Therefore, TMO developers can easily verify deadline of real-time methods, and improve reliability, and reduce development terms.

Key Words : Real Time Program, TMO Model, Execution Time Analysis, Static Analysis Tool

1. 서 론

실시간 시스템은 주어진 실행 시간을 보장하는 시간적 정확성을 가지고 동작한다. 때문에 소형 임베디드 시스템부터 대형 분산 시스템까지 시간적 정확성을 필요로 하는 다양한 시스템에서 사용될 수 있고 앞으로도 사용 분야가 더욱 넓어질 수 있는 컴퓨터 분야이다. 이러한 실시간 시스템의 확대에 따라 실시간 시스템을 기반으로 하는 실시간 프로그램에 대한 사용도 늘어나고 있다. 실시간 프로그램은 주어진 시간에 정확하게 일을 수행해야 하는 특성을 가지고 주어진 시간을 지키지 못하는 경우 경제적 피해를 주거나 인명사고로 이어질 수 있다. 제한 시간을 벗어나는 문제는 개발자의 실시간 개념 부족이나 시스템의 시간 지연 요소에 의해 발생할 수 있다. 따라서 실시간 프로그램 개발자들은 프로그

※ 본 연구는 정보통신부 및 정보통신 진흥원의 대학 IT연구센터 특성·지원사업의 연구결과로 수행되었음.
† 준 회 원 : 건국대학교 컴퓨터공학과 석사과정
†† 준 회 원 : 건국대학교 컴퓨터공학과 박사과정
††† 정 회 원 : 건국대학교 컴퓨터공학과 강의교수(교신저자)
†††† 종신회원 : 건국대학교 컴퓨터공학과 정교수
논문접수 : 2006년 2월 17일, 심사완료 : 2006년 4월 4일

램을 개발할 때에 개발에 전념하지 못하고 프로그램 실행시간을 정의하고, 정의한 실행시간의 정확성 여부를 검사하기 위하여 프로그램 분석에 많은 시간을 보낸다[1].

이러한 문제들 때문에 실시간 시스템에 대한 많은 연구가 이루어졌고, 그 결과중 하나로 TMO(Time-triggered Message-triggered Object) 모델이 나오게 되었다[3]. TMO 모델은 실시간 개념에 따른 시간 처리에 다양한 기능을 지원하고 응답시간을 보장하여 개발자가 프로그램 개발에 집중할 수 있도록 도와준다[4]. 하지만 설계 단계에서 실행시간의 정의는 개발자에 의해 이루어지기 때문에 이를 정의하고 정의된 실행시간의 정확성 여부를 확인하는 작업에는 여전히 어려움이 있다. 이러한 문제로 인하여 실행시간 정의의 기준점을 제시할 수 있는 도구에 대한 연구가 필요하다. 이러한 정적 분석 도구를 위한 연구로서 TMO 기반의 프로그램을 블록 단위로 나누어 분석을 수행할 수 있는 기반 구조인 PS-Block을 설계하였다. PS-Block은 소스코드를 하나의 주기성을 띄고 독립적으로 실행될 수 있는 루틴으로 나눈 Program Segment를 구문 분석해 생성된 PS-AST를 종류에 따른 PS-Block 구성 정책에 의해 그룹화하여 생성된다. 하지만 초기의 PS-Block 구성 과정은 중복되는 블록을 생성시키는 구성 정책으로 분석 과정에서 오버헤드를 발생시키는 문제점이 발생한다.

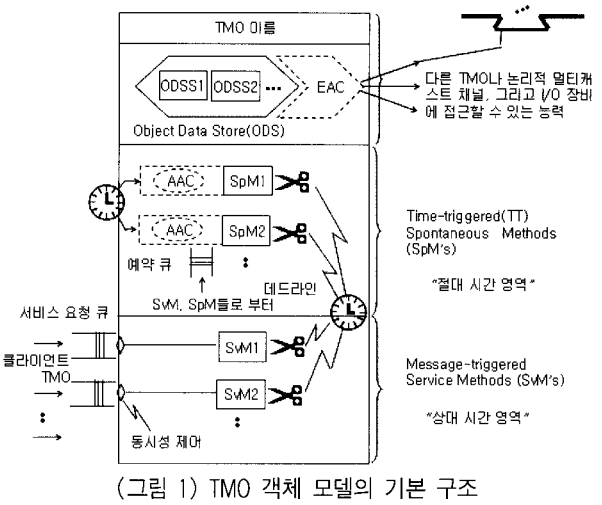
이에 본 논문에서는 중복되는 블록으로 인한 오버헤드를 줄이기 위해 구성 정책을 개선하고, PS-Block의 탐색을 위해 베이스 클래스를 정의하여 이를 기반으로 PS-Block Timing Model을 구현하였다. PS-Block Timing Model은 프로그램을 PS-Block 구성 정책에 따라 작은 블록 단위로 나누어 실행시간을 분석할 수 있는 기반을 제공하고, 복잡한 프로그램을 단순한 블록 구조로 분석할 수 있으며, 특정한 블록을 추출해 분석이 가능하도록 하여, 시간작업의 시간 정보 결정의 기준을 마련할 수 있도록 한다. 이를 통해 실시간 메소드의 적시성 확인을 쉽게 함으로써 실시간성/신뢰성 향상과 개발 기간을 단축할 수 있다.

본 논문은 2장에서 관련 연구로 실시간 시스템과 소스코드 분석 및 분해 과정 및 PS-Block의 설계에 대해 소개하고 3장에서 TMO 기반의 정적 분석 도구를 위한 PS-Block Timing Model에 대해 기술한다. 4장에서는 PS-Block Timing Model의 특징에 대하여 설명하고 마지막으로 5장에서는 결론과 향후 연구 방향을 제시한다.

2. 관련 연구

2.1 TMO

TMO 모델은 적시성 서비스 기능을 디자인 단계에서부터 보장하고, 실시간 시스템이 가지는 시간적인 행동, 메시지에 의한 기능적인 행동에 대한 추상화 등을 지원한다. TMO 모델은 실시간 시스템의 시간적 행동을 추상화하기 위한 SpM(Spontaneous Method), 메시지에 의한 행동을 추상화하기 위한 SvM(Service Method), 그리고 이들이 공유하는



(그림 1) TMO 객체 모델의 기본 구조

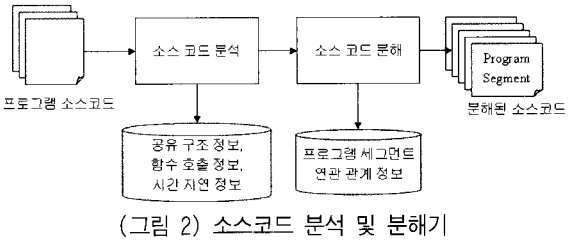
데이터를 세그먼트 단위로 저장하기 위한 ODSS (Object Data Store Segments)등으로 구성되어 있고, 이들을 TMO라는 하나의 객체로 모델링 할 수 있게 해준다. TMO의 시간적 행동은 AAC에서 정의되고, 정의된 주기와 제한시간을 가지고 반복 실행되는 SpM에 의해 SvM이 동작한다. (그림 1)은 TMO객체 모델의 기본 구조를 나타낸다.

TMO는 정시보장 컴퓨팅(Timeliness Guaranteed Computing)을 목표로 제안된 실시간 객체 모델로 경성/연성 실시간 응용뿐만 아니라 일반적인 응용 프로그램에도 사용할 수 있으며, 설계 시 시간 보장 개념을 제공한다[5, 6].

2.2 소스코드 분석 및 분해기

실시간성/신뢰성 정적 분석 과정에서 실행시간 분석을 위하여 소스의 재구성을 위한 함수의 호출 관계, 명시된 시간 지연 정보, 그리고 중복 계산을 피하기 위한 공유 함수에 대한 정보가 필요하다. 또한, TMO의 SpM과 같이 실시간 프로그램의 특성을 따르는 쓰레드들은 전체 흐름과 상관없이 시간에 의하여 동작하기 때문에 독립적으로 분석이 가능하도록 재구성하고, 재구성된 소스들 사이의 관계 정보를 도출하는 과정이 필요하다. 이러한 과정은 소스코드 분석 및 분해기에서 이루어지며 (그림 2)는 이러한 과정을 나타낸다.

소스코드 분석 과정 중 공유 함수와 전역 변수 등의 정적 구조는 공유 구조 정보를 구성하여 저장하게 되고, 코드에 명시되어 있는 sleep, delay, TMO의 AAC와 같은 시간 정보는 시간 지연 정보에 저장된다. 함수 호출 정보는 소스코드 내부에서 이루어지는 호출 관계와 참조 관계를 가지고,



(그림 2) 소스코드 분석 및 분해기

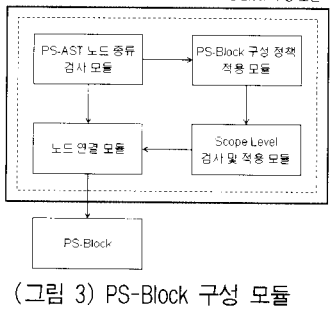
실행되지 않는 코드를 제거하며, 호출이나 자료 교환이 이루어 지지 않는 코드를 분리하는 근거가 된다. 이 함수 호출 정보를 근거로 소스코드 분해 과정에서 소스코드를 Program Segment단위로 분해한다. 여기서 Program Segment란 소스코드를 분석하고 재구축하여 하나의 주기성을 띄고 독립적으로 실행될 수 있는 루틴으로 나눈 것이다. 이와 함께 도출되는 프로그램 세그먼트 연관 관계 정보는 Program Segment들 사이의 연관관계를 저장한다.

2.3 PS-Block 구조

실시간 프로그램의 응답 시간을 보장하고, 시간 처리의 자유로움을 보장하는 TMO는 설계 단계에서부터 적정한 AAC를 정의해야 한다. 따라서 개발자는 소스코드의 분석을 통해 실행시간을 예측하여 메소드의 시간 처리에 대한 기준점을 제시하고, 실시간성 위반여부를 확인하며, 수정할 수 있도록 실행시간의 분석이 필요하다.

TMO 기반의 정적 분석을 위하여 TMO 프로그램을 분석하면 시간에 의한 행동인 SpM과 메시지에 의한 행동인 SvM에 의해 동작하기 때문에 시간에 따라 동작하는 분리된 흐름들로 구성되는 것을 알 수 있다. 따라서 이 흐름들을 소스코드 분석 및 분해기를 통하여 중복 실행되는 코드를 Static Structure로 분리함으로써 시간 측정의 중복을 피할 수 있는 Program Segment로 나누고 분석을 위한 기초 정보들을 도출한다. 분리된 각각의 흐름인 Program Segment를 분석하여 시간 계산에 필요한 최소 단위로 구성된 PS-AST (Program Segment Abstract Syntax Tree)로 구성하고 각각의 PS-AST 노드를 종류별로 묶어 블록 단위의 실행시간 분석을 위한 기반으로 PS-Block (Program Segment Block)구조로 재구성한다. PS-Block이란 Program Segment를 구분 분석한 PS-AST의 각 노드들을 실행시간의 분석을 위하여 그 종류에 따라 PS-Block 구성 정책을 적용하여 블록 단위로 나누어 재구성한 구조로서 프로그램을 작업 단위로 분리해 분석할 수 있다. 또한, 프로그램의 실행 경로를 명확하게 하고, 반복문의 범위와 구조를 간단하게 표현할 수 있다. (그림 3)은 PS-Block를 구성하는 과정을 나타낸다.

PS-Block은 PS-AST 노드 종류 검사 모듈에서 적용할 정책을 결정하고, PS-Block 구성 정책 적용 모듈에서 정책에 맞춰 블록 생성 및 연결과 통과 여부를 적용한다. Scope Level 검사 및 적용 모듈에서는 적용된 정책에 따라 생성된 PS-Block을 Level을 참조하거나, 연결할 PS-AST노드를 PS-AST의 영역을 가리키는 Scope를 참조해 노드 연결 모듈을 통해 적절한 상위의 PS-Block에 추가하여 PS-Block을 구성한다[2].

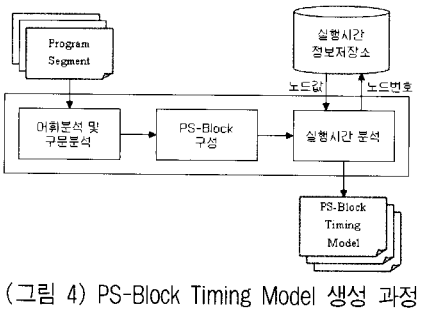


(그림 3) PS-Block 구성 모듈

3. PS-Block Timing Model

프로그램의 수행 전에 실행 시간을 분석하는 정적 분석은 실행 시간 분석을 위한 기반 구조인 PS-Block을 사용하여 소스코드를 PS-Block 단위로 분석할 수 있다. PS-Block 단위로 분석된 실행 시간들은 TMO 프로그램의 주기적인 흐름을 분석하기 위해 소스코드의 구조에 맞는 구조를 이루어야 한다. PS-Block Timing Model은 분석된 실행시간 값을 가진 PS-Block들이 모여 각 블록 단위의 실행시간을 분석하여 전체 소스의 실행시간을 분석할 수 있도록 트리 형태를 이룬 구조이다. PS-Block Timing Model은 TMO 프로그램 소스가 갖는 각각의 일정한 흐름을 분리한 Program Segment를 기반으로 한다. PS-Block Timing Model의 생성 과정은 컴파일러 전단부와 일부분 유사한 순서로 생성된다. PS-Block Timing Model은 (그림 4)와 같이 어휘 분석 및 구문 분석 단계와 PS-Block 구성 단계와 실행 시간 분석 단계를 거쳐 생성된다.

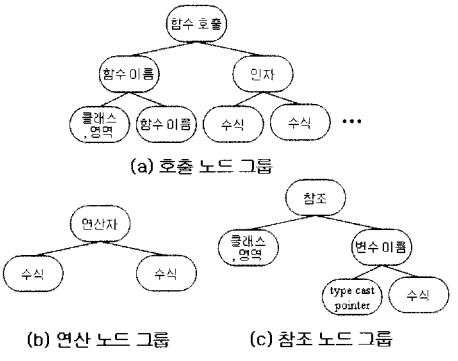
PS-Block Timing Model은 다음과 같은 과정을 거쳐 생성된다. 먼저 Program Segment는 어휘 분석 및 구문 분석을 통해 시간 계산의 최소 단위로 구문 분석한 문법 트리인 PS-AST를 생성한다. 생성된 PS-AST에 블록 구성 정책을 적용하여 PS-Block들을 생성하는 PS-Block 구성 과정을 거친다. 실행시간 분석 과정에서는 구성된 PS-Block에서 PS-AST노드 단위의 예상 실행시간을 실행시간 정보 저장소를 통해 분석하여 PS-Block 단위의 실행시간 값을 가지는 PS-Block Timing Model을 생성한다. 실행시간 정보 저장소는 실제 실행시간과의 차이를 보정하기 위한 구조로서 특정 시스템에 의존하지 않도록 분석의 독립성을 갖는 PS-AST노드의 실행시간 정보를 가진다.



(그림 4) PS-Block Timing Model 생성 과정

3.1 어휘 분석 및 구문 분석 모듈

어휘 분석 및 구문 분석 모듈은 문법 트리를 생성하기 위해 Program Segment를 토큰 단위로 분리하고 구문 분석을



(그림 5) PS-AST 노드 그룹

통해 PS-AST를 생성한다. 문법 트리의 구조는 C++ 언어의 구문 구조 및 의미 구조를 기반으로 분류하여 설계되었다.

PS-AST는 Program Segment를 구문 분석하여 생성된 문법 트리로서 실행시간의 분석을 위하여 연산자와 실행문, 변수의 이름을 중심으로 재구성을 거친 구문 트리이다. 재구성은 AST의 노드 중 실행시간 연산에 필요 없는 전역변수를 정의하는 노드와 일반 선언문을 나타내는 노드, 그리고 문법 구조에 따라서 생성되어 실행시간 분석과 연관이 없는 비 단말 노드들을 트리에서 제거하여 더욱 빠르고 효과적으로 실행시간의 연산이 가능하도록 한다.

PS-AST의 노드들은 크게 함수나 변수의 선언과 각종 연산을 수행하는 수식과 문법에 따르는 명령문들을 구성하는 노드의 그룹으로 나눌 수 있다. 재구성은 연산을 수행하는 수식들로 이루어진 노드를 중심으로 이루어지고, PS-AST의 수식들을 구성하는 노드들은 재구성을 통해 (그림 5)와 같이 각각 연산, 참조, 호출을 나타내는 노드의 그룹들로 구성된다.

호출 노드 그룹은 사용자 함수나 시스템 콜을 호출하는 문장을 표현하는 수식이고, 연산 노드 그룹은 연산자를 중심으로 두 수식 사이의 관계를 나타내는 수식이며, 참조 노드 그룹은 변수나 상수를 표현하는 수식이다. 각 노드 그룹들은 서로간의 재귀적 구성을 통해 PS-AST를 구성한다.

3.2 PS-Block 구성 모듈

PS-Block 구성 단계에서는 PS-AST를 노드의 종류에 따라 PS-Block 구성 정책을 적용하여 블록 단위로 나누어 재구성 한다. PS-Block 구성 정책은 반복문과 선택문에 대한 실행시간 분석을 분리하여 수행하기 위해 PS-AST를 반복문 및 선택문과 중괄호로 묶인 영역으로 나눌 수 있도록 특정한 PS-AST노드의 종류에 따라 PS-Block를 구성하도록 소스코드의 실행횟수와 Scope에 따라서 PS-Block의 종류를 정의한다. 이로써 PS-Block은 프로그램을 작업 단위로 나누어 분석할 수 있도록 하고, 프로그램의 실행 경로를 명확하게 하며, 또한 반복문의 범위와 구조를 간단하게 표현한다.

3.2.1 블록의 중복 생성에 의한 문제

기존의 PS-Block 구성 정책에서는 중괄호로 묶인 코드를 분리하기 위한 Compound Block을 정의하였다. 하지만 실제 소스코드에서는 함수나 반복문, 또는 선택문을 사용할 경우 중괄호를 사용하여 그 몸체를 구분한다. 따라서 반복문을 처리하기 위한 Iteration Block이나 선택문에 의한 분기를 나타내는 Selection Block 및 함수를 구분하는 Function Block과 함께 Compound Block이 생성될 수 있어 분석 과정에서 블록의 중첩이 발생하였다. PS-Block 구성 단계를 구현하며 중복되는 블록으로 인한 오버헤드를 줄이기 위하여 Compound Block을 구성 정책에서 제거하고 중괄호를 구분하는 역할을 다른 블록들이 대체할 수 있도록 한다.

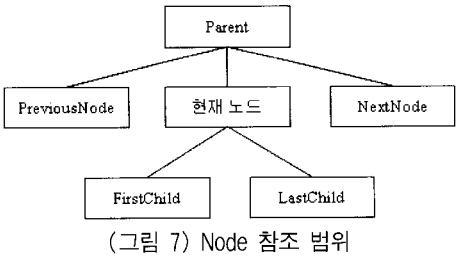
3.2.2 주위 블록에 대한 참조 문제

PS-Block은 블록마다 실행시간을 분석한 값을 저장하기 위한 필드를 가지고 있다. 이 실행시간 필드는 해당 블록을 분석한 실행시간과 자식 블록들의 실행시간의 합을 저장하는 필드로 나뉜다. 따라서 상위 블록은 하위 블록의 전체 실행시간을 가지고 블록 단위의 개별 실행시간을 분석할 수 있도록 한다.

블록 정보		실행시간		PS-AST		PS-Block	
블록 번호	블록 종류	실행시간	자식의 실행시간	PS-AST 자식 수	PS-AST 포인터	PS-Block 자식 수	PS-Block 포인터

(그림 6) PS-Block의 데이터 구조

이러한 실행시간의 분석을 위하여 실행시간 분석은 상위 블록보다도 하위 블록에서부터 이루어져야 한다. 이러한 특성으로 PS-Block은 블록간의 이동이 자유롭게 이루어져야 한다. 하지만, 기존 PS-Block의 구조는 자식블록의 주소만을 가지고 있어 상위 블록이나 이전 블록을 참조하기 어려운 구조를 가지고 있다. 이를 해결하기 위해 베이스 클래스 노드를 정의하였다.



(그림 7) Node 참조 범위

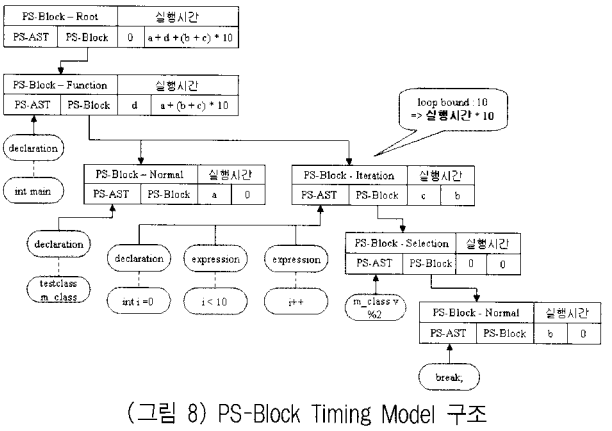
베이스 클래스 Node는 트리 구조에서 부모 노드와 부모 노드의 다른 자식 노드를 참조하는 방법을 제공한다. PS-Block은 Node를 상속받음으로서 실행시간 분석할 때 쉽게 다른 블록으로 이동할 수 있다.

3.3 Timing Analysis 모듈

어휘 분석 및 구문 분석 모듈에서 생성된 PS-AST는 PS-Block 구성 모듈을 통해 PS-Block으로 구성된다. Timing Analysis 모듈은 PS-Block 단위의 분석을 통해 각각의 PS-AST노드 그룹이 가지는 실행시간의 합으로서 시간 계산을 수행한다. PS-Block 구조의 부모 자식 블록 관계

는 프로그램의 실행 경로와 연관된다. 이는 하위블록이 나타내는 프로그램 코드는 반드시 상위 블록의 코드를 지나야 함을 의미하고, 따라서 하위 블록의 실행시간을 상위 블록이 포함하기 때문에 가장 하위의 블록에서부터 실행시간을 분석한다. 실행시간을 분석한 PS-Block Timing Model의 구조는 (그림 8)과 같다.

PS-Block의 실행시간 분석은 최하위에 위치한 블록부터 시작된다. 하위 블록들의 실행시간 분석이 완료되면 상위 블록에서 전체 하위 블록의 실행시간을 계산한다. 하위블록에 Iteration Block이 위치할 경우 Timing Analysis 모듈은 Iteration Block의 반복횟수를 계산하여 분석된 실행시간에 곱한 값으로 반영한다. 실행시간의 분석이 완료되면 루트 PS-Block에는 분석한 작업의 전체 실행시간에 대한 예측값이 저장된다.



(그림 8) PS-Block Timing Model 구조

4. PS-Block Timing Model의 특징

기존의 PS-Block에서는 중괄호로 묶인 코드를 분리하기 위해 정의한 Compound Block에 관한 정책으로 분석과정에 오버헤드가 발생할 수 있고, 상위 블록이나 주위 블록에 대해 참조가 어려운 문제가 있다. 이에 PS-Block Timing Model의 구현 과정에서 중복되는 블록으로 인한 오버헤드를 줄이기 위해 구성 정책에서 Compound Block에 관한 정책을 제거하였고, 상위 블록과 같은 Scope의 다른 블록의 참조 방법을 제공하기 위하여 베이스 클래스 Node를 정의하였다. 이러한 PS-Block 구조의 개선을 통해 PS-Block으로 구성된 PS-Block Timing Model은 블록 수의 감소로서 실행 시간 분석 과정에서의 오버헤드를 줄이고 블록 사이의 참조가 자유롭게 이루어지도록 하였다. 기존의 PS-Block 구성 정책으로 생성된 PS-Block과 개선된 구성 정책을 적용하여 생성된 PS-Block의 분석 시간을 비교한 결과는 다음 <표 1>과 같다. 이 실험을 통해 PS-Block Timing Model의 개선을 통한 PS-Block의 감소와 이에 따른 실행 시간 분석 과정의 작업 시간의 감소를 확인할 수 있었다.

PS-Block Timing Model은 독립된 PS-AST 노드를 가지고 있는 PS-Block들로 이루어져 PS-Block 단위의 실행시간의 분석이 가능하다. 또한, 블록 구성 과정의 오버헤드

〈표 1〉 기존 구조와 개선된 구조의 분석 시간 비교

항 목		개선 전	개선 후
fork.c	PS-Block 수	382 개	303 개
	분석 시간	12.0163 μs	10.7153 μs
sched.c	PS-Block 수	1835 개	1453 개
	분석 시간	43.3426 μs	38.5677 μs

를 개선한 PS-Block 단위로 분리하여 분석함으로써 복잡한 프로그램을 단순한 블록 구조로 분석할 수 있으며, 특정한 블록의 추출이 가능하다. 따라서 PS-Block Timing Model을 통한 정적 분석은 프로그램의 실행 전에 추상화된 각 메소드들에 대하여 분리된 각 소스코드의 실행시간을 분석할 수 있도록 하고, 프로그램의 실행시간이 가장 긴 수행경로를 분석하는 최장 경로 분석 방법의 적용이 쉽다. 이처럼 분석 대상을 나눔으로써 각각 실행시간의 분석이 가능하도록 하고, 이를 기준점으로 제시하여 개발자의 AAC의 정의와 대상 메소드의 실행시간 적시성 확인을 도울 수 있어 프로그램의 실시간성 및 신뢰성을 향상시키고 개발 기간을 단축할 수 있도록 한다.

5. 결 론

실시간 프로그램은 시간적 정확성을 가져야 하기 때문에 개발자는 이를 고려하여 실행시간을 정의하고 그 정확성 검사를 수행해야 한다. 응답시간을 보장하는 TMO 모델에서도 개발자의 시간 정보 정의와 실시간성/신뢰성의 검증이 필요하고, 이를 위한 정적 분석 도구가 필요하다. 이러한 정적 분석 도구를 위한 연구로서 TMO 기반의 프로그램을 블록 단위로 나누어 분석을 수행할 수 있는 기반 구조인 PS-Block을 설계하였다. 하지만 초기의 PS-Block 구성 과정은 중복되는 블록을 생성시키는 구성 정책으로 분석과정에 오버헤드를 발생시키는 문제점이 발생하였다.

본 논문에서는 TMO 기반의 정적 분석 도구를 위한 PS-Block을 구현하는 과정에서 오버헤드를 발생하는 PS-Block 구성 정책을 개선하고 주변 PS-Block의 참조를 위한 베이스 클래스를 정의하며 이를 기반으로 PS-Block Timing Model을 구현하였다. 개선된 PS-Block 구조를 바탕으로 하는 PS-Block Timing Model은 프로그램을 PS-Block 구성 정책에 따라 작은 블록 단위로 나누어 실행시간을 분석할 수 있는 기반을 제공한다. 또한, 작은 단위로 분리하여 분석함으로써 복잡한 프로그램을 단순한 블록 구조로 분석할 수 있고, 특정한 블록의 추출이 가능하도록 하여 시간작업의 시간 정보 결정의 기준을 마련할 수 있도록 한다. 이를 통해 실시간 메소드의 적시성 확인을 쉽게 함으로써 실시간성/신뢰성 향상과 개발 기간을 단축할 수 있다.

향후 연구 과제로는 구현된 PS-Block Timing Model을 사용하여 정적 분석기를 구현하고, 실행시간 분석 결과의 정확성 및 신뢰성 향상을 위하여 실제 실행시간과의 차이를 보정하기 위한 방법과 반복문과 선택문에 의한 실행시간의 변화에 대한 연구를 진행할 예정이다.

참 고 문 헌

[1] 김태완, 장천현, 김문회, "TMO 네트워크로 구성된 분산 실시간 시스템을 위한 실시간성 분석기 설계", ITRC forum 2004.

[2] 김윤관, 신원, 김태완, 장천현, "TMO기반 정적 분석 도구를 위한 PS-Block 구조의 설계", 정보처리학회 추계학술발표논문집 제12권 제2호 pp.263-266, 2005.

[3] K. H. (Kane) Kim, "A TMO Based Approach to Structuring Real-Time Agents," 14th IEEE ICTAI, pp.165-172, 2002.

[4] Kim, K.H. "Commanding and reactive control of peripherals in the TMO programming scheme," 5th IEEE ISORC, pp.448-456, 2002.

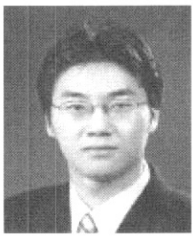
[5] K.H. Kim, Juqiang Liu, Moon-Hae Kim, "Deadline Handling in Real-Time Distributed Objects," 3rd IEEE ISORC, pp.7-15, 2000.

[6] K.H. Kim, "Object-Oriented Real-Time Distributed Programming and Support Middleware," 7th IEEE ICPADS, pp.10-20, 2000.

[7] C. Healy, M. Sjödin, V. Rustagi, D. Whalley, "Bounding Loop Iterations for Timing Analysis," 4th IEEE RTTAS, pp.12-21, 1998.

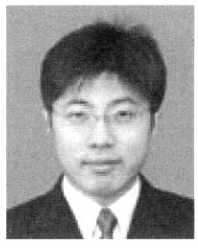
[8] Jakob Engblom, Andreas Ermedahl, Friedhelm Stappert, "Comparing Different Worst-Case Execution Time Analysis Methods," 21st IEEE RTSSWIP 2000, 2000.

[9] Yau-Tsun Steven Li, Sharad Malik, "Performance analysis of embedded software using implicit path enumeration," ACM SIGPLAN, Vol.30, Issue 11, pp.88-98, 1995.



김 윤 관

e-mail : apostlez@konkuk.ac.kr
2005년 건국대학교 컴퓨터공학과(학사)
2005년~현재 건국대학교 컴퓨터공학과 석사과정
관심분야 : 컴파일러, 실시간 프로그래밍, 프로그래밍 언어, 임베디드 시스템



신 원

e-mail : wonjjang@konkuk.ac.kr
2005년 건국대학교 컴퓨터공학과 (공학석사)
2005년~현재 건국대학교 컴퓨터공학과 박사과정
관심분야 : 임베디드 시스템, 컴파일러, 프로그래밍 언어, 모니터링 시스템

김 태 완

e-mail : twkim@konkuk.ac.kr
1994년 건국대학교 전자계산학과(공학사)
1996년 건국대학교 전자계산학과(공학석사)
1996년~2001년 현대중공업 기전연구소 연구원
2003년~2004년 경민대학 인터넷비즈니스과 겸임교수
1996년~현재 건국대학교 컴퓨터공학과 박사과정
2004년~현재 건국대학교 컴퓨터공학부 강의교수
관심분야 : 프로그래밍 언어, 실시간 프로그래밍, 자동화 소프트웨어, 산업기기 가시 진단 제어 시스템

장 천 현

e-mail : chchang@konkuk.ac.kr
1977년 서울대학교 계산통계학(학사)
1979년 KAIST 전산학(석사)
1985년 KAIST 전산학(박사)
현 재 건국대학교 컴퓨터공학과 정교수
관심분야 : 프로그래밍 언어, 컴파일러, 실시간 시스템