Windows Host 시스템에 퀀텀 프레임워크 적용방안

김호균, 이영대, 김기환*
세명대학교 정보통신학과, *세명대학교 전자공학과

An Approach to Apply Quantum Framework to Windows Host System

Ho Kyun Kim, Young Dae Lee, Kee Hwan Kim*
Semyung University Dept. of Information and Communication,
*Semyung University Dept. of Electronic Engineering

Abstract - 스테이트 차트에서 (Unified Modeling Language: UML) 상태 기반의 이벤트를 처리하는 실행 상태를 기술하고, 구현과 임베디드 시스템에 적용하는 방안에 대하여 논의하며 퀀텀 플랫폼(quantum platform)을 이용하는데 구성요소인 퀀텀프레임워크 컴포넌트에 대한 부분을 Windows Host 환경에 적용하는 방법에 대해 연구하고 성능평가를 하였다.

1. 서 론

본 논문은 상태차트와 상태차트 기반 컴퓨팅 모델로 구현함으로써, 새로운 프로그래밍 패러다임으로 퀀텀 프로그래밍(Quantum Programming ; QP)이라고 불일 수 있다. 프로그래밍 패러다임으로써의 QP는 단순한 코드보다 훨씬 많은 것을 제공한다. QP를 C 같은 기존의 프로그래밍 언어의 추상화 수준을 향상 시키는 기법의 모음으로 간주하며 사용 역시 같이 한다. QP에서 추가된 추상화를 통해, 반응형 시스템을 C를 이용해 효과적으로 모델링하며 객체지향(object-oriented, OO) 프로그래밍 언어의 역할에 견줄 수 있다.

본 논문에서 퀀텀 플랫폼(quantum platform)을 이용하는데 구성요소인 퀀텀프레임워크 컴포넌트에 대한 부분을 Windows Host 환경에 적용하는 방법에 대해 연구하고 성능평가를 하였다.

2. 본 론

2.1 퀀텀 플랫폼

퀀텀 플랫폼은 이벤트 처리와 실시간 임베디드 애플리케이션 개발을 위한 경량의 소프트웨어 구조이다. 퀀텀 플랫폼의 특징은 현대적이며, 간단하며, 실제적인 소프트웨어이다. 또한 안정성과 이식 가능한 특징도 있다. 퀀텀 플랫폼은 설정에 대한 변경이 용이하고, 교차 환경을 지원한다. 또한 퀀텀 플랫폼은 엔지니어들이 큰 툴 없이 C 언어로 지속적 유지 가능한 계층 형 상태 메커니즘과 이벤트 구동 상식을 직접적으로 구현하는 것을 가능하게 해준다. 퀀텀 플랫폼은 각각의 퀀텀 이벤트 프로세서, 퀀텀 프레임워크, 퀀텀 커널, 퀀텀 스파이들의 컴포넌트로 구성되어 있다.

2.2 상태머신

UML 상태차트는 Mealy 오토마타와 Moore 오토마타의 특징 모두를 가지고 있는 확장상태머신이다. 상태차트에서 액션은 대개 시스템의 상태와 촉발 이벤트 모두에게 영향을 받는다. 또 상태차트는 진입 액션 (entry action)과 탈출액션 (exit action)을 제공할 수 도 있다. UML에서는 상태, 전이, 이벤트, 동작들을 그래픽으로 표현할 수 있게 제공한다. 이 표기 법을 쓰면 객체 행동을 가시화 할 수 있어 그 객체의 생활에 있는 중요한 요소들을 강조할 수 있다.

2.3 상태머신 구현방법

C와 같은 상위 수준 프로그래밍 언어로 상태머신을 구현할 때는 퀀텀 플랫폼에는 중첩된 switch 문 (Nested Switch Statement), 상태 테이블 (Action-state Tables), 상태 설계 패턴 (The "State" Design Pattern), 퀀텀 계층형 상태 머신(The Quantum Hierarchical State machines) 들로 설계되었다.

2.3.1 중첩된 switch 문

상태머신을 구현하는데 있어 가장 널리 사용되는 기법이다. 상태에서 취급되는 각 이벤트를 위해 특정의 상태들과 보조 구분들에서 객체의 행동을 반영하고 있는 구분들에 이벤트를 처리하는 기능을 분할하기 위해 2개의 중첩된 switch 문을 사용한다. 상태변수를 switch 문의 첫 번째 수준 식별자로, 이벤트 신호를 두 번째 수준 식별자로 사용한다.

중첩 switch 문 구현에는 다음과 같은 특징이 있다.

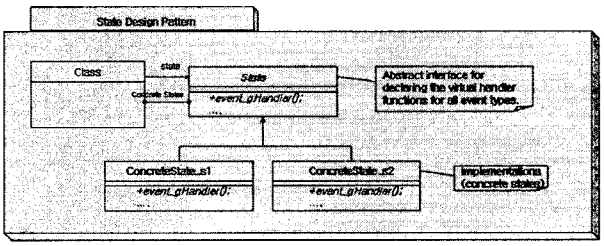
- 간단하다.
- 상태와 트리거의 열거형이 필요하다.
- 상태머신을 표현하는데 하나의 스칼라 상태변수만이 필요하므로 메모리 소모가 적다.
- 특정 문제에 맞게 상태머신의 모든 요소를 직접 코딩해야 하므로 코드 재사용이 어렵다.
- 이벤트 전달 시간이 일정치 않고 switch 문 두 번째 수준의 성능에 좌우된다.
- 상태머신 토폴로지 변화의 관점에서 볼 때 손으로 코딩한 진입/탈출 액션과 중첩된 초기전이는 오류가 생길 여지가 많고 관리하기 어렵다.

2.3.2 상태 테이블

많이 쓰이는 다른 접근법으로, 활동-상태 테이블은 특정의 상태들에서 특정한 이벤트의 도착에 관해 실행되는 기능들을 포인터에 저장한다. 이 방법은 중첩된 switch 문 보다 약간 빨리 구현 할 수 있지만 더 많은 기억을 필요로 한다.

2.3.3 상태 설계패턴

상태머신을 구현하는데 쓰이는 객체지향 접근법을 State 설계패턴이라고 한다. 그림 6은 상태 설계 패턴으로 구현한 상태머신을 나타내고 있다. 그림 1은 Class 상태에서 State 상태로 전이 되고 있으며, State 상태에는 event_gHandler(); 속성을 갖고 있다. State 상태는 ConcreteState_s1 상태와 ConcretsState_s2 상태의 부모 클래스이며 각각 s1, s2 상태에 속성을 상속시키고 있다.



<그림 1> 상태 설계 패턴으로 구현한 상태머신

이 패턴은 위임(delegation)과 다형성에 기반을 두고 있다. 구상 상태는 이벤트 처리의 공통 인터페이스를 추상 상태 클래스의 하위클래스로 표현되고 각 이벤트는 가상 메소드에 연결된다. 컨텍스트 클래스는 처리할 모든 이벤트를 현재 상태객체에 위임한다. 컨텍스트란 객체가 생명주기 동안에 다른 객체에 작용할 수 있고, 작용 받을 수 있는 동기적인 것을 포함하며 이들 메시지들은 단순한 동기 연산을 호출한다. 상태전이는 명시적이며, 포인터를 재 할당함 으로써 수행된다. 새로운 이벤트를 추가하려면 추상 상태 클래스에 새로운 메소드를 추가해야 하고, 새로운 상태를 추가하려면 이 클래스의 하위 클래스를 파생해야 한다. 단점으로는 동적인 실행부분에 대하여 어떠한 모델로 나타내어 있지 않고, 이전의 행동들을 보이는 그대로 이행해야 하기 때문에 다른 사람이 이 구현에 대하여 수정 할 수 없다.

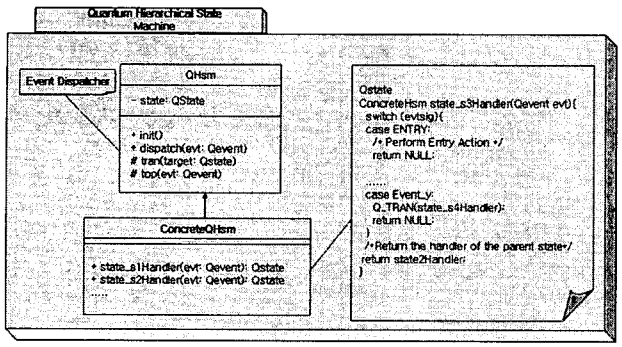
상태 설계 패턴에는 다음과 같은 특징이 있다.

- 특정 상태로 국한된 행동을 분할해 별도의 클래스에 분배한다.
- 상태전이를 효율적으로 만든다 (하나의 포인터에 재할당).
- 늦은 바인딩(late binding) 메커니즘을 통해 이벤트 전달네 매우 좋은 성능을 보인다.
- 각 이벤트 핸들러의 선언부를 입맛에 맞게 고칠 수 있다.
- 메모리를 효율적으로 사용한다.
- 상태와 이벤트를 열거할 필요가 없다.
- 모든 상태 클래스를 friend로 지정해야 하므로 컨텍스트 클래스의 캡슐화를 훼손한다.
- 컨텍스트 포인터를 통해 구상 상태 하위 클래스의 메소드에서 컨텍스트의 매개변수를 간접적으로 엑세스해야 한다.
- 상태를 추가하려면 구상 상태 하위클래스를 추가해야 한다.
- 새로운 이벤트를 처리하려면 상태 클래스 인터페이스에 이벤트 핸들러를 추가해야 한다.
- 이벤트 핸들러는 대개 상태 테이블 기법에서와 같이 아주 세밀하게 나뉘진다.

2.3.4 컨텀 계층형 상태 머신

기존 상태머신에 비해 상태차트의 가장 중요한 혁신은, 계층형으로 중첩된 상태(hierarchically nested state)의 도입이다. 이것을 적용한 것이 퀀텀 계층형 상태머신(QHsm)이라고 한다.

상태 중첩은 하나의 수준으로만 제한되지 않으며, 이 단순한 이벤트 처리 규칙이 여러 수준의 중첩에 계속해서 적용된다. 그림2에서 퀀텀 계층형 상태머신으로 구현한 상태차트를 나타내고 있다.



<그림 2> 퀀텀 계층형 상태머신으로 구현한 상태차트

다른 상태를 포함하는 상태를 복합 상태(composite state)라 한다. 반대로, 내부구조가 없는 상태를 단순 상태(simple state)라고 한다. 하위 상태가 상위 상태에 직접 포함돼 있으며 이를 직접 하위 상태(direct substate)라고 하며, 그렇지 않고 또 다른 하위 상태 안에 포함된 경우에는 추이적 중첩 하위 상태(transitively nested substate)라고 한다.

그림 2는 이벤트 배차원 기능을 위한 실현과 상태의 전이들을 실행하고 있는 기능을 제공하는 추상적인 QHsm 클래스이다. 그것에서 비롯되는 클래스들은 특정의 상태들에서 이벤트를 취급하기 위해 기능들을 실행해야 한다. 만일 이벤트가 부모 상태의 현상 또는 주소에서 성공적으로 실행되면 이 기능들은 어느 한 쪽에 NULL 값의 포인터라도 돌려준다.

퀀텀 계층형 상태 머신의 특징은 다음과 같다.

- 사용과 관리가 간단하다.
- 상태머신 토폴로지를 쉽게 바꿀 수 있다.
- 특히 전이 사슬을 직접 코딩할 필요가 없다.

런 타임 효율이 좋고 메모리 소모가 적다.

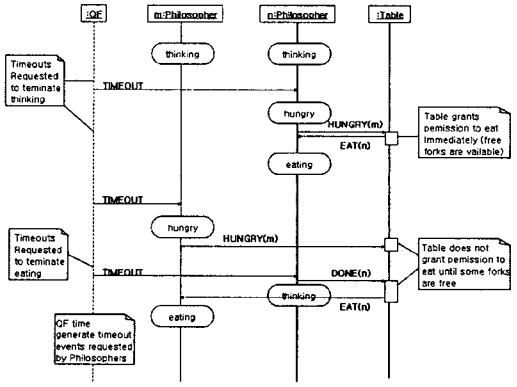
3. 구 현

본 논문에서 제안한 퀀텀 플랫폼을 적용하여 Windows Host 환경을 구축하기 위해서는 퀀텀 이벤트 프로세서, 퀀텀 프레임워크, 퀀텀 커널들이 필요하다. 퀀텀 프레임워크를 Microsoft Windows Host 환경에 적용하는 것에 재해 진행하겠다. Windows 환경에 Borland Turbo C++가 설치된 디렉터리에 QF/C 라이브러리를 GNU-make 유틸리티를 이용하여 mingw32-make 명령어로 새로 구축한다. 이것으로 Windows Host 환경에 퀀텀 플랫폼을 적용한다. 간단한 예제 프로그램인 철학자들의 만찬 프로그램에서 발생하는 문제점들에 대하여 구현하였다.

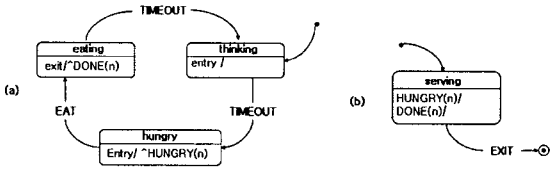
<그림 3> 철학자들의 만찬

그림 3은 철학자들의 문제를 담고 있는 그림이다. 그림에 5명의 철학자들이 가운데 있는 스파게티를 먹기 위해 앉아 있다. 철학자들 사이에 포크가 놓여있고 스파게티를 먹기 위해서 철학자들은 2개의 포크를 이용하여야 한다. 철학자들의 삶의 대부분은 생각하고 식사를 하는 것으로 이루어져 있다. 철학자들이 식사를 하고 싶을 때 포크가 필요할 것이다. 한명의 철학자가 식사를 하기 위해 포크 2개를 얻는 데 성공하고 나서 다음 동작으로는 생각하는 행동을 보일 것이다. 이와 같은 조건에서 철학자들이 식사를 하는 데 있어서 겹쳐지지 않도록 병렬적으로 프로그램해 보았다.

그림 4는 이벤트 처리방식을 이용하여 각각의 철학자들에 대한 사건들이 시간에 따라 바뀌어 지고 있는 모습을 볼 수 있다. 철학자가 식사를 하기 전에 생각을 하고 있고 생각하는 상태가 끝나면 배고픔의 상태로 전이될 때 다른 철학자들의 상태는 계속해서 생각하는 상태에 남아 있어야 한다. 그래서 식사를 다 한 철학자는 다음 상태인 생각하는 상태로 전이되면 다른 철학자들이 배고픔 상태로 전이되고 다시 식사를 하는 상태로 전이가 될 수 있다. 다음 그림에서 철학자들의 상태 챠트를 볼 수 있고, 이와 같은 행동을 하기 위해 마련된 테이블에 재한 상태 챠트를 그림으로 나타내 보았다.



<그림 4> 철학자들의 만찬에서 이벤트 처리 방식을 이용한 시퀀스 다이어그램



<그림 5> (a) 철학자들의 시간에 따른 상태차트, (b) 테이블에서 일어나는 상태 챠트

이 사건에 대하여 Windows Host 환경에서 Borland Turbo C++를 이용하여 구현하였다.

<그림 6> DOS 콘솔 시스템을 통하여 Turbo C++를 이용하여 구현된 모습

4. 결 론

본 논문에서는 UML 상태차트를 이용한 퀀텀 프레임워크를 Windows Host 환경에 적용 시키는 방법에 대하여 구현하였다. 퀀텀 플랫폼을 구성하는 퀀텀 프레임워크 컴포넌트를 Windows Host 환경에서 철학자들의 만찬에 대한 프로그램을 상태차트를 Borland Turbo C++를 이용하여 구현하였다.

[참 고 문 헌]

[1] Miro Samek, Practical Statecharts in C Quantum Programming for Embedded Systems, 2004

[2] QF/C Programmer's Manual, www.quantum-leaps.com

[3] QEP/C Programmer's Manual, www.quantum-leaps.com

[4] 송태훈, 남상엽, 알기쉬운 임베디드 시스템 기론, 2005

[5] 김호균, 임베디드 시스템에 퀀텀 프레임워크 적용방안, 2007, 석사학위 논문