중소기업정보기술융합학회 논문지 제2권 제1호 pp. 33-40, 2012

UML 모델 기반 임베디드 소프트웨어 모델링 및 코드 자동 생성 기법 연구

류호동, 이우진*
경북대학교 IT대학 컴퓨터학부

A Study on UML based Modeling and Automatic Code Generation for Embedded Software

Hodong Ryu and Woo Jin Lee
School of Computer Science and Engineering, Kyungpook National University

요 약 최근에 임베디드 환경은 하드웨어의 비약적인 발전과 다양한 전자 제품에서의 소프트웨어 제어로 인하여 소프트웨어 분야에서 많은 변화를 겪고 있다. 이러한 환경적 변화로 인한 요구사항의 증가는 임베디드 소프트웨어의 복잡도를 높여왔고 이는 기존 개발방법의 한계를 보여주었다. 모델 기반의 개발방법은 이미 오래전 제기되었던 범용 소프트웨어 개발에서 발생했던 한계의 해결을 위하여 제시되고 지금까지 사용되어온 방법으로서 임베디드 환경에 닥친 한계의 극복을 위한 대안 중 하나로 꼽히고 있다. 이 논문에서는 이 모델 기반 개발 방법론을 임베디드 소프트웨어 적용하기 위한 모델 기반의 다이어그램 편집기와 이로부터 작성된 모델을 이용하여 자동적으로 코드를 생성하는 코드 자동생성기를 제안한다. 모델 정의에 사용된 다이어그램 편집기는 GMF를 이용하여 구현하며, 코드 자동생성기에는 임베디드 환경의 특징인 제한된 메모리와 동시적 병행성 문제를 해결하기 위한 코드 생성 기법을 추가한다. 아울러 생성된 코드의 검증을 위하여 기존의 코드를 대체하여 수행하는 방법을 사용한다.

키워드 : 모델기반 개발, UML, 코드 생성

Abstract Recently, embedded environment suffers a huge change, by growth of hardware and turning to be software-controlled. This has improved embedded software complexity. It also brought us the limit of the old development way to resolve the problem. Model-driven development is one solution to solve the limit common software development by previous way, and it became a one uses for embedded environment also. In this paper, we propose model based development approach for embedded software, witch consists of diagram editor and automatic code generator. The diagram editors are implemented by GMF, which include additional functions to solve memory restrictions and concurrent execution problems without OS environment to a automatic code generator. In order to verify the generated code, it will be tested in main control model of UAV by replacing existing module with generated one.

Key Words : Model based Development, UML, Code Generation

*교신저자(woojin@mail.knu.ac.kr)
접수일(2012년04월15일), 심사완료일(2012년06월07일)

1. 서론

컴퓨터 하드웨어의 비약적인 발전과 더불어 오늘날의 임베디드 환경과 그 소프트웨어는 이전의 범용소프트웨어에 버금가는 성능과 복잡도를 가지고 있다. 이러한 소프트웨어의 복잡성을 해결하기 위해 기존의 개발 방법에서 벗어나기 위한 여러 노력이 있어 왔고, 모델 기반의 개발 방법을 임베디드 소프트웨어 개발에 적용하는 시도는 그중 하나이다. 이에 이 논문에서는 모델 기반의 개발 방법에 대하여 알아보고 이를 임베디드 개발 환경에 적용하기 위한 도구를 제안한다. 이를 위해서 모델의 정의를 위한 모델링 편집도구와 코드 자동 생성을 위한 코드 생성기를 제안하고, 이를 구현하는 과정을 보인다. 아울러 이 도구의 검증을 위하여 실제 하드웨어에 적용하여 실제 동작 여부를 확인한다.

이 논문에 구성은 다음과 같다. 2장에서는 기존의 모델 기반의 방법과 모델을 이용한 코드 자동생성기에 대하여 소개하고 3장에서는 UML의 대표적인 다이어그램인 유스케이스 다이어그램과 컴포넌트 다이어그램, 상태머신 다이어그램과 Eclipse의 Graphical Modeling Framework(GMF)를 이용하여 이를 편집하는 다이어그램 편집기를 구현하는 과정을 보인다. 4장에서는 제시된 다이어그램 편집기를 통해 정의된 모델을 이용하여 코드를 생성하는 과정을 보인다. 아울러 코드 자동 생성과정에서 임베디드 환경을 위한 2가지 기법을 추가로 보인 후, 5장에서는 이 논문에서 제시된 도구들을 이용하여 무인 모형항공기를 구현하고 그 결과물을 보인다. 마지막 6장에서는 결론과 향후 연구 과제에 대하여 기술한다.

2. 관련연구

2.1 모델 기반 개발 방법

1980년대에 들어 단순한 코딩을 통한 방법에서 벗어나 상태머신이나 구조 다이어그램을 통한 새로운 접근법에 대한 연구가 시작되었고 이는 결과적으로 소프트웨어 환경에서 CASE라는 새로운 분야를 생성하였다. 하드웨어적 한계와 모델 자체의 추상적 특성으로 인하여 그 당시에는 큰 인기를 얻지는 못하였지만, 최근 들어 하드웨어의 발전 속도를 소프트웨어가 따라잡지 못하는 일들이 발생함에 따라 이에 대한 새로운 대안으로 모델 기반의 개발방법의 필요성이 점점 대두되고 있다[1]. 아울러 오늘날에는 소프트웨어 복잡성이나 기존의 구현된 모듈의 재사용성들의 관점에서 모델 기반의 개발 방법이 더욱 각광을 받고 있다. 현재 사용되는 CASE 도구들은 이미 각 분야에 특화된 많은 도구가 나와 있으며 단순히 모델을 편집하고 코드를 생성하는 기능만을 떠나 소프트웨어 개발 프로세스 전반에서 각각의 단계들을 서로 연결하고 팀원 간의 협업을 돕는 기능을 제공하는 상황에 이르러 있다.

2.2 자동 코드 생성 기법

코드 자동 생성에 대한 관심에 높아짐에 따라 여러 도구들이 사용되고 있으며 그중에서도 상용의 IBM의 Rational Rhapsody[2]와 오픈 소스 도구인 StarUML[3]은 가장 대표적인 도구들이다. Rational Rhapsody는 대표적인 상용 모델링 및 코드 자동 생성도구로서 C, C++, JAVA, ADA 등의 다양한 형식의 언어를 지원하며, UML에서 제시한 여러 다이어그램의 편집기능과 GUI와 연동된 시뮬레이션 기능을 제공한다. 아울러 최근에는 IBM사의 Jazz 플랫폼[4]에 포함되어 개발 프로젝트의 전체적인 협력 작업의 한 축으로 사용되고 있다. 실시간 팀 협업 환경이나, 실시간 프로젝트 현황 확인, 소프트웨어 팀의 통합 등과 같은 Jazz가 제공하는 라이프사이클 안에서 코드 모델링과 코드 자동생성이라는 기능으로 플랫폼내의 핵심적인 역할을 담당한다.

starUML은 오픈 소스임에도 불구하고 UML 2.0에서 제안하는 거의 대부분의 다이어그램 편집기능을 제공하고 비록 제한된 형태이지만 코드 생성의 기능도 제공하여 모델 기반개발 방법의 적용에 적합한 도구이다. 아울러 오픈 소스라는 특징에 힘입어 최근 널리 사용되고 있다.

3. 소프트웨어 모델링 및 모델링 도구

모델 기반 개발의 기본은 모델이고 이는 다양한 형태의 다이어그램으로 이루어져있다. 이중 대표적인 것이 요구사항을 정의하는 유스케이스 다이어그램과 코드 자동생성에 사용되는 구조적 요소와 행위적 요소를 각각 정의한 컴포넌트, 클래스 다이어그램과 상태머신 다이어그램이다. 본장에서는 이들 다이어그램의 특징과 이러한 다이어그램의 작성을 위한 편집기를 구현하는 과정을 보인다.

3.1 소프트웨어 요구사항의 UML 모델링

유스케이스 다이어그램은 UML의 여러 다이어그램 중 요구사항을 정의하는 것에 특화된 다이어그램이다. 일반적으로 유스케이스 다이어그램은 사용자 혹은 특정 역할을 의미하는 엑터와 각각의 기능, 그리고 엑터와 기능 혹은 기능과 기능사이의 관계를 표현하는 선으로 정의된다. 그림 1은 엑터와 기능 사이의 관계를 간단히 나타낸 유스케이스 다이어그램이다. 그림 1의 다이어그램을 통하여 2개의 엑터 중 트레이더는 메니저와 상속을 받는 관계이고 메니저는 한계를 정하는 기능, 트레이더는 위험을 분석하고, 가격을 협상하는 기능을 가지고 있으며 이 2개의 기능은 모두 가치를 협상하는 기능을 포함한다는 것을 알 수 있다.

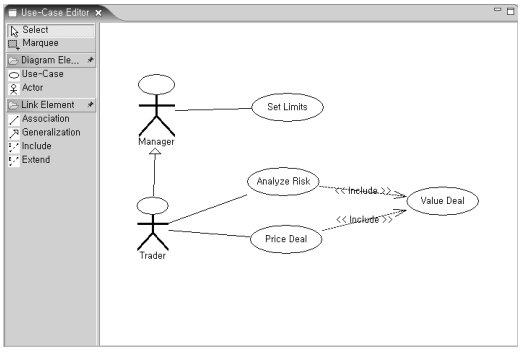

그림 1. 증권 거래 시스템의 유스케이스 다이어그램
Fig 1. Use-case diagram of Stock trading system

이와 같이 유스케이스 다이어그램은 요구사항 관점에서 기능과 그 기능 사이의 의존성을 정의하고 아울러 구현될 대상과 그 밖의 요소를 구분하여 그들 사이의 접촉점을 정의함으로써 구현 대상에 대한 요구사항을 분명하게 한다.

3.2 GMF를 이용한 모델링 도구의 구현

GMF(Graphical Modeling Framework)는 Eclipse 환경 기반에서 그래픽 편집 플러그인 제작을 위한 대표적 프레임워크이다. 그림 2는 이를 이용한 다이어그램 편집기 생성과정을 보인다. GMF를 이용한 모델링 도구의 구현은 기본적으로 모델링되는 모델의 메타 모델, 즉 각각의 모델 요소는 어떻게 구성되며 각각의 어떻게 서로 연결되는지를 EMF를 이용하여 정의하고 정의될 모델의 시각적인 모습을 정의하고 편집기에서 사용할 도구창의 모습을 정의한 후 최종적으로 이 3가지를 통합하여 다이어그램 편집기를 자동으로 생성한다. 다음에서 소개되는 편집기들은 이러한 과정을 통해 정의된다.

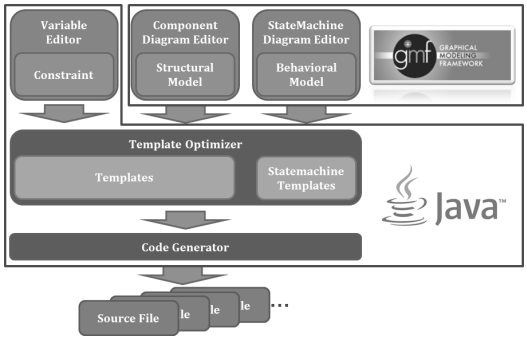

그림 2. GMF를 이용한 다이어그램 편집기 생성 과정
Fig 2. Generation process of diagram editors by GMF

3.2.1 컴포넌트 모델링 도구의 구현

컴포넌트 다이어그램은 코드 자동 생성과정에서 사용되는 대상 모델의 구조적 정보를 가진 다이어그램이다. 이는 기본적으로 컴포넌트와 컴포넌트가 제공하는 인터페이스와 이 인터페이스를 사용하는 컴포넌트 간의 관계로 정의된다. 하나의 컴포넌트는 하나 이상의 인터페이스를 제공할 수 있고, 하나 이상의 컴포넌트가 제공되는 인터페이스를 사용할 수 있다. 아울러 각각의 컴포넌트는 내부에 또 다른 컴포넌트를 포함할 수 있고 포함된 컴포넌트 사이에 인터페이스 제공과 사용관계는 그 깊이와 상관없이 모두 동일하다. 일반적으로 컴포넌트 다이어그램에서 컴포넌트는 기능적 관점의 최소 단위로 표현된다.

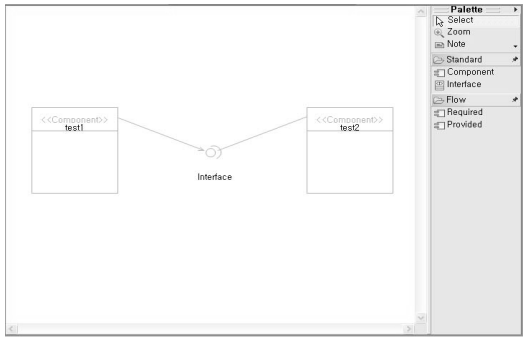

그림 3. 컴포넌트 다이어그램 편집기
Fig 3. Component diagram editor

논리적인 의미에서 특정 기능이 재사용될 때 그에 해

당하는 컴포넌트와 그로부터 생성된 코드가 기능적인 관점에서 재사용될 수 있음을 의미한다. 컴포넌트 재사용이라는 개념은 이러한 관점에서 파생된 것이다. 또한 이는 컴포넌트 모델링에서 각각의 컴포넌트를 어떠한 기준으로 나누어야 하는가에 대한 중요성 역시 의미하고 있다. 서로 밀접한 관련이 있는 기능은 하나의 컴포넌트로 묶고, 서로 관련이 없는 것은 별도의 컴포넌트로 정의하는 방법은 컴포넌트 정의에 있어 가장 기본이 되는 개념이다. 일반적으로 각각의 컴포넌트들은 코드 자동 생성기를 통해 코드화되는 과정에서 언어에 따라 클래스 혹은 하나의 소스 코드 형태로 구현된다.

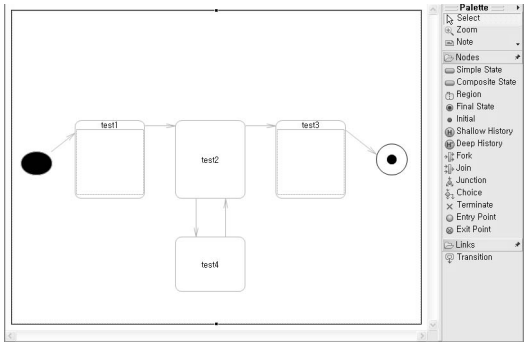

그림 4. 상태머신 다이어그램 편집기
Fig 4. State machine diagram editor

3.2.2 컴포넌트 상태 모델링 도구의 구현

컴포넌트 다이어그램이 모델의 구조적인 측면을 정의하는 다이어그램이라면 상태머신 다이어그램은 각 컴포넌트가 어떤 행위를 하는지를 상태기반으로 표현한 다이어그램이다. 기본적으로 상태의 시작과 종료를 의미하는 초기 상태와 종료 상태 사이에서의 여러 상태 간의 상태 변화와 상태 변화의 조건에 대하여 정의한 다이어그램이다.

모델 상에서 상태는 실제 코드 상에서의 어떤 행위와 매치된다. 예를 들어 어떤 위험에 대하여 알리는 상태머신 다이어그램이 있을 때 그 위험이라는 상태는 다른 어떤 조건이 만족될 때 발생하는 상태이고, 그러한 상태에는 해야 할 어떤 일, 즉 특정한 액션을 포함한다. 코드 관점에서는 이러한 관계를 다른 방향으로 해석한다. 즉 코드 자체는 해야 할 일의 목록이고 상태라는 추상적 의미가 드러나지 않기 때문에 특정 행위가 언제 일어나야 하는지에 관한 관점에서 구현되어야 한다는 것이다. 즉 상태머신 다이어그램이 상태의 변화와 각 상태에서 수행되어야할 행위를 정의한다면, 실제 코드는 각각의 행위가 수행되는 조건을 기술하는 형태로 구현된다. 다음의 그림 4는 본 논문에서 생성한 상태머신 편집기를 이용한 간단한 상태머신 편집 화면을 보이고 있다.

4. 소프트웨어 자동 코드 생성

모델기반 개발에서 코드 자동생성은 대표적인 구조정의 다이어그램인 컴포넌트 다이어그램과 각 컴포넌트 내부의 행위를 정의하는 상태머신 다이어그램을 기반으로 이루어진다. 코드 자동 생성기는 구조 정보를 이용하여, 생성될 소스파일의 틀을 만들고 컴포넌트로 정의된 각 소스파일간의 관계를 구현한다. 그림 5는 이를 도식화한 것이다.

모델을 이용한 코드 자동 생성은 먼저 다이어그램 형태의 모델을 XMI 형태의 언어로 변화시키는 과정과 변화된 모델을 분석하여 각각의 패턴을 매치시키는 과정 마지막으로 매치된 정보를 바탕으로 실제 코드로 구현하는 과정을 거친다. 컴포넌트 다이어그램과 클래스 다이어그램과 같은 구조적 관점의 다이어그램은 C 소스로 구현될 때 각각의 파일이 어떻게 구성되고, 파일간의 관계가 어떻게 이루어지는지에 대한 정보를 포함하고 있다. 예를 들면, 각각의 컴포넌트와 클래스들은 모두 독립적인 하나의 소스파일로 정의되고 컴포넌트 간의 의존성으로 #include 문 형태로 구현된다.

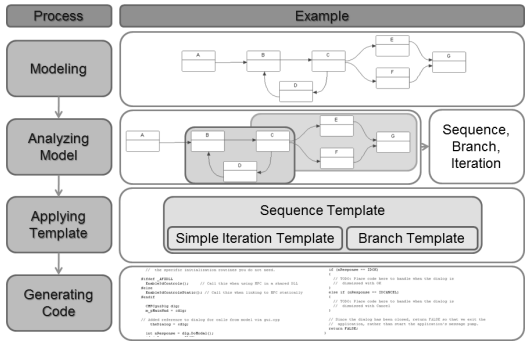

그림 5. 소스코드 자동 생성 과정
Fig 5. Automatic generation of source code

소스 파일 내부에서 정의되는 행위적 코드는 각각의

컴포넌트와 클래스가 포함하고 있는 상태머신 다이어그램에서 추출된다. 상태머신 다이어그램은 상태 머신 다이어그램을 최상위로 하여 여러 상태를 의미하는 스테이트와 이들을 연결하는 트렌지션으로 이루어진다. 각각의 스테이트는 다시 1개 이상의 영역으로 나누어져 다음에서 제시한 동시성과 같은 특성을 표현한다.

먼저 각각의 상태들은 하나하나가 함수 형식으로 정의되어 소스 파일에 구현된다. 가장 기본적인 형태의 코드 생성 패턴은 각각의 상태가 함수로 표현되고, 그 함수가 호출될 때의 상태에 따라 해야 할 일과 다음에 호출될 함수를 정의하는 형태의 코드 생성방법이다. 그림 6은 이에 대한 예를 보인다. 그림에서 보이는 상태머신 다이어그램은 모두 5개의 상태로 이루어져 있다. 각각의 상태는 초기 상태 안전상태 위험 상태 속도 증가 및 감소를 의미한다. 그리고 각각의 상태는 속도라는 변수에 따라 변화된다. 이를 코드로 변환하면 우측과 같은 형태가 된다. 즉 전체 상태머신은 하나의 함수가 되고 상태를 결정하는 distance의 값은 조건문에 조건이 된다. 상태머신 다이어그램에서 거리에 대한 확인이 필요할 때 이 상태머신 다이어그램 내부로 들어오는 것과 같이, 코드 상에서도 거리 확인을 위한 함수가 호출될 때 distance라는 변수에 값에 따라 분기되고 각 분기문 내부에서는 safe, danger 들과 같은 상태에서 정의된 코드들이 있어 이들이 수행되는 형식이다. 이와 같은 방식의 구현은 추상적인 관점의 모델과 구체적인 관점의 코드 상의 일관성을 의미한다.

최종적으로 각각의 상태에서는 소스코드에 구현될 문장 단위의 코드가 추출된다. 각각의 문장들은 하나하나가 독립적인 의미를 가지고 이러한 의미들은 분기 혹은 반복문의 형태를 거쳐 전체적인 프로세스를 정의한다. 상태 모델 속에서 다이어그램 형태로 정의된 분기문과 반복문은 코드 자동 생성기의 패턴 매칭 알고리즘을 이용하여 적합한 패턴과 매치되고 이는 for문 if-else 문과 같은 실제적인 코드 형태로 변화된다.

4.1 동시 병행 문제의 해결

상태머신 다이어그램에서는 하나의 상태를 2개 이상의 영역으로 나누어 각각의 영역이 서로 다른 영역에 상관없이 독립적으로 동시에 수행된다는 의미를 정의한다. 이는 C에서 fork()문과 같은 자식 프로세스를 생성함으로써 간단히 구현 가능하다. 하지만 대부분의 임베디드 환경에서는 기능 구현이 가능한 최소한의 환경에서 수행되기 때문에 이와 같은 기능을 제공하는 운영체제가 제공되지 않는다. 이는 추상적인 관점의 모델과 실제 수행되는 환경간의 차이가 발생하게 된다. 즉 코드 자동생성 과정에서 이에 대한 구현 방법의 제시가 필요한 것이다.

이 논문에서는 모델 상에서 구현된 동시병행 요소를 운영체제가 없는 환경에서 수행되는 코드로 생성하는 방법으로, 코드 삽입을 이용한 방법을 제안한다. 일반적으로 fork() 등을 이용한 구현도 운영체제차원에서 스케줄러에 따라 각각의 프로세스를 번갈아 수행하는 방식을 사용한다. 이 논문에서는 A라는 상태에 B와 C라는 영역이 동시 병행으로 수행됨으로 정의되었을 때 B를 C에 코드 중간에 혹은 그 반대의 형태로 정의하여 구현하는 방법이다. 이를 위해서는 어느 영역을 어느 영역에 삽입해야 할지, 삽입되는 빈도는 어떻게 정의해야 할지를 결정하여야 한다. 이 논문에서 구현하는 바로는 각 영역의 복

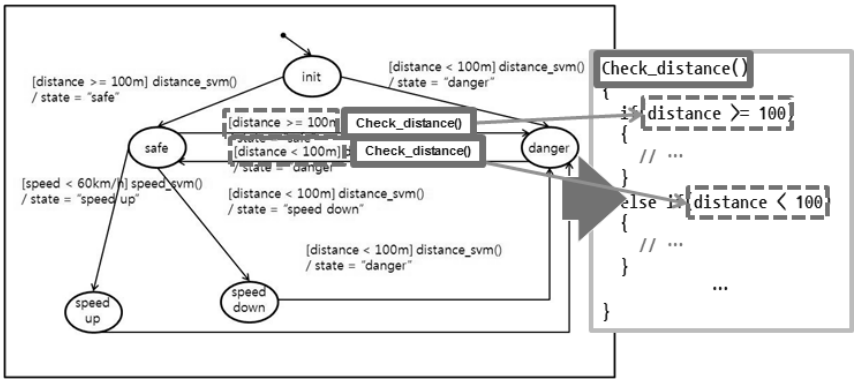

그림 6. 상태머신다이어그램에서 소스 코드 추출 과정
Fig 6. Extraction process of source code from state machine diagram

잡도를 비교하는 방법을 사용하며 이를 기반으로 각각의 코드 중간에 다른 영역의 코드를 삽입한다.

4.2 변수 재사용 기법

앞에서 언급하였듯이 임베디드 환경에서는 여러 환경적 제약이 수반된다. 그중 대표적인 것이 제한적인 메모리이다. 이 논문에서는 변수의 재사용을 통한 프로그램 수행에서는 메모리 점유율을 낮추는 방법을 제안한다. 사실 이는 일반적인 임베디드 개발환경에서도 흔히 쓰이는 방법으로 4바이트 정수 변수가 있을 때 앞 2바이트와 뒤 2바이트를 서로 다른 의미로 사용하는 방법이다. 즉 각각의 최대 및 최솟값이 2바이트 내에 표현되는 서로 다른 변수가 있을 때 이를 하나의 4바이트 변수에 적용하는 방식이다. 이는 1바이트와 3바이트 등 다양한 방법으로 수행될 수도 있다. 하지만 이는 개발자가 직접 변수의 범위를 확인하고 기존의 코드에 직접 추가해야 하는 단점이 있다. 이 논문에서는 이를 자동화하여 변수 사용코드에 적용하는 방법을 제시한다. 이를 위해 먼저 변수 재사용에 사용될 함수의 템플릿이 필요하다. 이는 최소 2바이트부터 8 혹은 16바이트의 변수를 2개의 이상으로 나누어 값을 쓰고 읽는 기능을 가진 함수의 템플릿이다. 아울러 이 함수를 적용한 변수를 찾아야 한다. 먼저 2바이트 변수 2개를 사용하기 위해 굳이 4바이트 변수를 정의할 필요가 없듯이 기준이 되는 것은 프로그램에서 가장 큰 변수의 크기이다. 아울러 이 변수가 실제 값으로 읽고 쓰는 도중에 다른 변수로 사용 되서는 안 됨으로 본래의 의미로 사용되는 범위를 알고 있어야 한다. 이는 모델 상에서 변수가 사용되는 상태를 파악함으로써 가능하다. 최종적으로 충분히 큰 변수가 유휴상태로 판단되면 코드 자동 생성기는 이를 나누는 함수를 이용하여 변수를 요구하는 다른 부분에서 사용할 변수로 제공하는 방식으로 수행된다.

5. 적용 사례 연구

그림 7은 본 논문에서 제시된 모델링 및 코드 자동 생성 도구를 이용하여 생성된 코드를 이용한 무인 모형 항공기를 보이고 있다. 기본적으로 4개의 날개를 이용하여 전후좌우 및 회전, 상승 및 하강의 8방향으로 이동이 가능한 구조이다. 4개의 날개를 이용한 미세한 이동은 기존의 컨트롤 보드를 이용하였으며 각 날개와 하단에 부착된 거리 센서를 이용하여 위치를 보정하는 기능을 본 도구를 통해 구현하였다.

다음의 그림 8과 9는 각각 무인 항공기의 호버링 모드 구현에 사용된 컴포넌트 다이어그램과 상태머신 다이어그램이다. 이 무인항공기는 메인 컨트롤러를 중심으로 앞뒤 좌우의 거리를 측정하는 센서와, 앞뒤좌우 방향으로 이동을 위한 팬 컨트롤러로 이루어진다.

그림 7. 무인 자율 비행 항공기
Fig 7. Unmanned Aerial Vehicle

메인 컨트롤러는 실시간으로 센서로부터 각 방향의 거리 즉 장애물과의 거리를 읽어오고 이를 토대로 각 방향의 거리가 특정한 값 이하로 떨어지면 그 거리를 이용하여 팬 컨트롤러에 앞뒤좌우 방향으로의 이동을 명령하는 구조이이다. 이를 통해 이 항공기는 좁은 실내에서 특정 위치를 호버링한다.

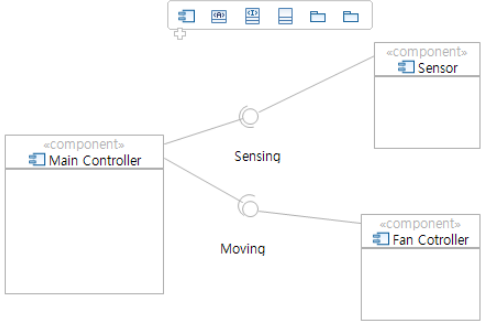

그림 8. 무인항공기의 컴포넌트 다이어그램
Fig 8. A component diagram of UAV

실제 무선 조정기를 통해 직접 조정할 경우 역동적인

비행기 가능한 기체이지만 무인 비행 과정에서 오는 위험성으로 인하여 실내에서 실험하였으며 실내에서는 GPS과 같은 절대화된 위치 정보를 수신할 수 없어 기본적으로 하단의 센서로부터 고도를 보정하고 전후좌우 센서로부터 물체가 발견되면 이를 회피하는 과정으로 비행을 수행하였다. 이를 통해 본 논문에서 제안한 다이어그램 편집기와 코드 자동생성기가 올바르게 동작함을 확인할 수 있었다.

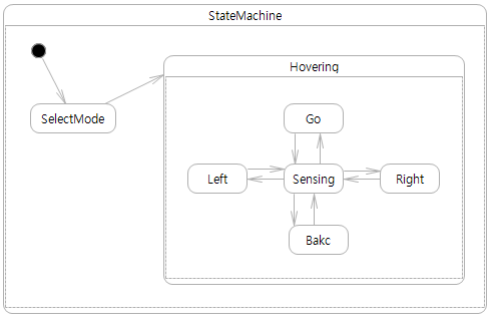

그림 9. 무인 항공기의 호버링 상태 세부 다이어그램
Fig 9. A state machine diagram of hovering state

6. 결론

예전의 단순 기능에서 벗어나 오늘의 임베디드 환경은 하드웨어의 발전에 힘입어 비약적인 발전을 이루었고 이는 기존의 개발 방법의 적용에 한계를 가져왔다. 이에 본 연구에서는 모델 기반의 개발 방법을 적용한 도구를 제안하였다. GMF를 이용하여 UML에서 제시하는 다이어그램을 작성하는 편집 도구를 구현하였고, 이로부터 작성된 모델로부터 코드를 자동 생성하는 코드 자동 생성기를 제안하였다. 아울러 코드 자동생성 과정에서 임베디드 환경의 여러 제약사항을 해결하기 위한 추가적인 코드 생성 기법을 추가하였다. 또한 본 도구를 이용하여 모델링하고 자동 생성한 코드를 모형 무인항공기의 비행 로직에 적용함으로써 본 도구에 대한 적용사례에 대해 연구하였다.

임베디드 환경이 다양한 하드웨어의 사양에 따라 각각의 환경이 매우 상이한 만큼, 모델 기반의 개발에 있어 이러한 차이점이 모델링 과정에서 정의되어야 한다. 이를 위하여 향후 연구에서는 특정한 환경 변수들을 모델상에서 표현하고 이를 코드 자동 생성과정에 반영하는 연구가 필요하다. 즉 기존의 다이어그램 편집기를 범용적으로 확장하는 방법과 코드 자동 생성기 역시 기존의 정의된 패턴에서 벗어나 실시간으로 패턴을 확장할 수 있는 기능을 가져야 한다.

참 고 문 헌

[1] Schmidt, D.C.;, "Guest Editor's Introduction: Model-Driven Engineering," Computer , vol.39, no.2, pp. 25- 31, Feb. 2006
[2] The open source UML/MDA Platform StarUML, http://http://staruml.sourceforge.net/
[3] IBM Rational Jazz, http://www.jazz.net/
[4] Graphical Editing Framework (GEF), http://www.eclipse.org/gef/
[5] Robin Milner, Communication and Concurrency, Prentice Hall, 1989
[6] P. J. Denning, et al., Machines, Languages, and Computation, Prentice Hall, 1978
[7] Robin Milner, Communication and Concurrency, Prentice Hall, 1989
[8] Frantisek Plasil, and Stanislav Visnovsky, "Behavior Protocols for Software Components," IEEE Transactions on Software Engineering, Vol. 28, No. 11, Nov. 2002.
[9] C.A.R. Hoare, Communicating Sequential Processes, Prentice Hall, 1985.
[10] Eclipse RCP: A Platform for Building Platforms, http://onjava.com/pub/a/onjava/2006/08/23/eclipse-rich-client-platform.html
[11] Lego Mindstorms NXT, http://en.wikipedia.org/wiki/Lego_Mindstorms_NXT
[12] Eclipse UML2 Tools, http://www.eclipse.org/modeling/mdt/?project=uml2tools
[13] Eclipse GMF(Graphical Modeling Framwork), http://www.eclipse.org/modeling/gmf/
[14] 이병용, 류호동, 권진욱, 석미희, 이우진, "데이터 흐름을 반영하는 임베디드 시스템의 코드 자동 생성기 설계", 한국정보처리학회 춘계학술발표대회 논문집, 제17권, 제1호, 2010
[15] Orehek M, Robl C, Farber G, "Model-based design of an ECU with data- and event-driven parts using auto code generation," PROC IEEE INT CONF ROB AUTOM. Vol. 2, pp. 1346-1351. 2001
[16] OMG, Unified Modeling Language : Superstructure, version 2.1.1, 2007.
[17] 조수란, 강성원, "소프트웨어 설계 툴의 코드자동생성능력

비교 연구", 한국정보과학회, Vol. 33, No.2, pp.325-330, 2006

[18] L.carnevali, D.D'Amico, L.Ridi, E.Vicario, "Automatic Code Generation from Real-Time Systems Specifications," International Symposium on Rapid System Protyping, 2009

[19] Eclipse Modeling Framework (EMF), http://www.eclipse.org/emf/

[20] D. Harel, "Statcharts: A visual formalism for complex systems," Sci. Comput. Prog., Vol. 8, pp 231-274, 1987

저 자 소 개

류 호 동(Hodong Ryu) [비회원]

- 2007년 2월 : 경북대학교 전자전기컴퓨터학부 학사
- 2009년 2월 : 경북대학교 전자전기컴퓨터학부 석사
- 2009년 3월 ~ 현재 : 경북대학교 전자전기컴퓨터학부 박사과정

<관심분야> : 모델 기반의 프로그래밍, 소프트웨어 테스팅

이 우 진(Woo Jin Lee) [정회원]

- 1992년 2월 : 경북대학교 전자전기 컴퓨터 학사
- 1994년 2월 : 한국과학기술원 전산학과 석사
- 1999년 8월 : 한국과학기술원 전산학과 박사
- 1999년 7월 ~ 2002년 2월 : 한국전자통신연구원 S/W 공학연구부 선임연구원
- 2002년 3월 ~ 현재 : 경북대학교 IT대학 컴퓨터학부 부교수

<관심분야> : 임베디드 소프트웨어 모델링 및 분석, 정형적 방법, 임베디드 소프트웨어 테스팅 등