UML과 LVOOP를 기반으로 한 내장형 시스템의 설계 구현 방안

조혁규* · 정민포* · 정덕길**

*영산대학교, **동의대학교

The Design and Implementation of the Embedded System based UML and LVOOP

Hyuk-Gyu Cho* · Min-Po Jung* · Deok-Gil Jung**

*YoungSan University, **Dongeui University

E-mail : minpo@ysu.ac.kr

요 약

UML은 객체지향 개념을 매우 잘 설명하고 있으며 요구 분석 단계에서 시스템 기능을 분석하기 위한 유스케이스 다이어그램과 도메인의 객체들을 개념적으로 표현한 클래스 다이어그램을 지원하고 있으며, 설계 단계에서 클래스 내부적인 연결을 보기 위한 시퀀스 다이어그램을 지원한다. 객체지향을 지원하는 대표적인 비주얼 기반 언어인 LabVIEW OOP는 직관적인 설계 도구로서 설계와 동시에 실행할 수 있는 개발 도구이다.

그러나, LabVIEW OOP 역시 시스템 개발자가 시스템을 설계하기 위해 설계 단계에서 객체지향 개념을 잘 표현하고 이해할 수 있는 방법이 필요하다. 논문에서는 객체지향 개념을 표현하는 UML을 이용하여 설계된 모델링을 LabVIEW OOP로 설계 시에 적용 가능한 기법을 제안한다.

ABSTRACT

UML is a very good description of object-oriented concepts and supports the use case diagram for analyzing the system functionality, the class diagram for representing the domain object in the analysis step, the sequence diagram for connecting the action in the class. The visual-based, object-oriented LabVIEW OOP is an intuitive design tool that can be executed at the same time as a development tool.

However, even the system developer using LabVIEW OOP is needed known about the object-oriented concept in the design stage. In this paper, we suggest the method to applying the UML modeling diagram to LabVIEW OOP design.

키워드

UML, LabVIEW, LabVIEW OOP, Embedded System

Ⅰ. 서 론

UML[1][2][3][4]은 객체지향 개념을 매우 잘 설명하고 있으며 요구 분석 단계에서 시스템의 기능을 분석하기 위한 유스케이스 다이어그램과 도메인의 객체들을 개념적으로 표현한 클래스 다이어그램을 지원한다. 설계 단계에서 크래스의 내부적인 연결을 보기 위한 시퀀스 다이어그램을 지원한다.

비주얼 기반의 LabVIEW OOP[6,7] 역시 객체지향 개념을 잘 표현하고 직관적인 설계도구로서 바로 실행할 수 있는 개발 도구이다. LabVIEW OOP는 클래스(VI)와 클래스 사이를 연결하는 메시지(Wire), 객체의 상태를 저장하는 프라이빗 데이터를 저장하는 프라이빗 데이터 컨트롤로 구성된다.

그러나 초급 개발자들이 LabVIEW OOP를 바로 사용하기 위해서는 객체지향 개념이 요구되어 다소 프로그래밍에 어려움을 호소한다. 이 논문에서는 최종 결과물인 비주얼 기반의 프로그래밍 도구인 LabVIEW OOP의 구성 요소를 객체지향 개념에 따라 설계하는 방법을 제시하기 위해, 객체지향 설계 도구인 UML의 다이어그램을 이용하여 LabVIEW OOP 프로그램을 좀 더 효율적이고 개념적으로 설계할 수 있는 방법을 제시한다.

Ⅱ. 본 론

내장형(Embedded) 프로그래밍을 주로 구현하는 LabVIEW OOP 프로그래밍을 구현하기 위해, 개발자는 객체지향 개념으로 개발을 위한 설계도를 만들 필요가 있다. II-1절에서 LabVIEW OOP를 위한 UML의 구성 요소를 소개와 변환 규칙을 설명한다. II-2절에는 변환 규칙을 적용을 위한 UML 클래스를 설계한다. II-3절에서는 설계된 UML 클래스를 바탕으로 LabVIEW OOP 클래스를 구현한다.

II-1. LabVIEW OOP를 구현하기 위한 UML 구성 요소

UML은 객체지향 개념을 설계하기 위해 아래와 같이 많은 다이어그램을 지원한다. 요구 분석 단계에서는 시스템의 기능을 알기 위한 유즈케이스 다이어그램(Use case Diagram), 도메인의 객체들을 개념적으로 표현한 클래스 다이어그램(Class Diagram), 사용자와 시스템 간의 상호활동을 표현한 활동 다이어그램(Activity Diagram), 특별한 생명주기에 유용한 상태 다이어그램(State Diagram)을 사용한다. 설계 단계에서는 클래스의 내부적인 연결을 보기 위한 클래스 다이어그램(Class Diagram), 클래스 간의 메시지 전달을 표기한 시퀀스 다이어그램(Sequence Diagram), 소프트웨어의 구조를 큰 단위로 나누는 패키지 다이어그램(Package Diagram), 클래스의 복잡한 생명 기록을 알아내기 위한 상태 다이어그램(State Diagram), 소프트웨어의 물리적인 레이아웃을 그리기 위해 배치도를 사용한다.

그 외에 정적 모델링을 하는 컴포넌트 다이어그램(Component Diagram), 복합 구조 다이어그램(Composite Structure Diagram), 기능적 모델링을 하는 활동 다이어그램(Activity Diagram), 동적 모델링을 하는 의사소통 다이어그램(Communication Diagram), 상호연동 개요 다이어그램(Interaction Overview Diagram), 타이밍 다이어그램(Timing Diagram)이 있다[8][9].

많은 UML 다이어그램 중에, 클래스 다이어그램과 시퀀스 다이어그램을 LabVIEW OOP로 변환하기 위한 클래스로 사용한다.

그림 1은 이 논문에서 제안하는 UML의 다이어그램에서 LabVIEW OOP로의 변환 규칙을 제시한다.

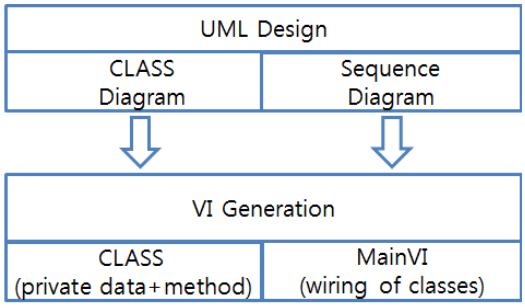

그림 1. UML 다이어그램에서 LabVIEW OOP로의 변환 규칙

UML에서 설계된 클래스 다이어그램의 클래스는 LabVIEW OOP 클래스와 1:1로 매핑된다. 예를 들어, UML의 클래스 다이어그램이 10개면, LabVIEW OOP의 클래스 역시 같은 이름의 가지는 클래스(VI) 10개를 가지게 된다. UML에서 설계된 시퀀스 다이어그램은 LabVIEW OOP에서는 객체들 사이의 와이어링(Wiring)을 가지는 VI 한 개로 매핑된다.

II-2. 변환규칙을 적용을 위한 UML 클래스 설계

변환 규칙을 적용하기 위해 먼저 클래스 다이어그램을 생성한다. 그림 2에서 예제에 사용될 RFID 불량판단 시스템에 대한 클래스 다이어그램을 보여준다.

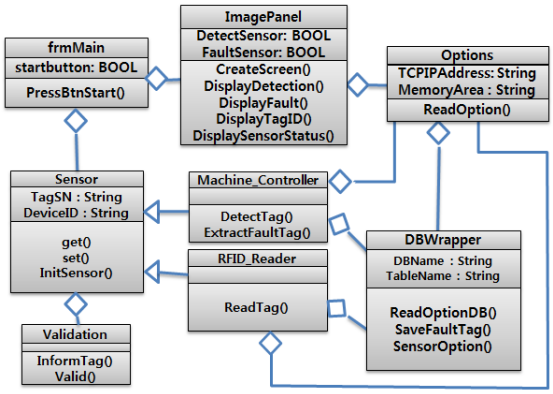

그림 2. RFID 불량 검출 시스템에 대한 클래스 다이어그램

RFID 불량 검출 시스템의 클래스 다이어그램은 frmMain, ImagePanel, Options, DBWrapper, Machine_Controller, RFID_Reader, Sensor, Validation 클래스 설계된다. Machine_Controller와 RFID_Reader는 Sensor 클래스의 자식 클래스로 설계된다.

다음 단계로는 시퀀스 다이어그램을 생성한다. 그림 3은 RFID 불량 검출 시스템에 대한 시퀀스

다이어그램을 제시한다.

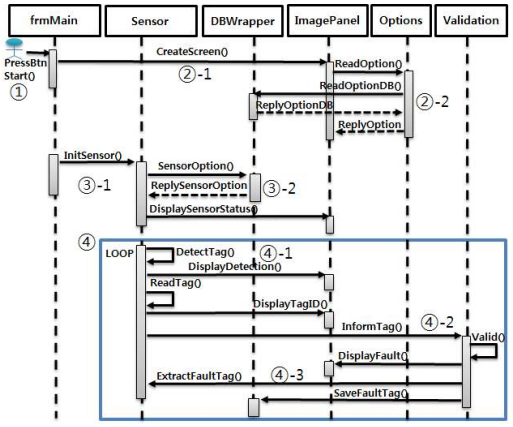

그림 3. RFID 불량 검출 시스템에 대한 시퀀스 다이어그램

시퀀스 다이어그램의 시퀀스는 만들어진 클래스 다이어그램을 사용하여 프로그램의 요구 사항에 따라 설계된다. frmMain 객체의 PressBtn() 호출을 시작으로 ImagePanel 객체의 CreateScreen()을 호출하여 불량 태그 감지를 위한 화면을 생성한다. ImagePanel 객체는 Options 객체의 ReadOption() 객체를 통하여 상화별로 설정된 옵션을 읽는다. Options 객체는 옵션을 읽기 위해, DBWrapper 객체의 ReadOptionDB() 함수를 사용한다.

다음 단계로 frmMain 객체는 센서를 초기화하기 위해, Sensor() 객체의 InitSensor() 객체를 호출하고 DBWrapper 객체의 SensorOption() 함수를 사용하여 센서 설정 옵션을 읽어들인다. 센서 초기화 후, Sensor 객체는 프로그램이 종료할 때까지 DetectTag() 함수를 사용하여 태그를 인식하고 그 결과를 ImagePanel 객체의 DisplayDetection() 함수를 통하여 결과를 화면에 나타내고 Sensor() 객체의 ReadTag() 함수를 통하여 태그를 읽어들인다. 읽어 들인 태그의 정보는 ImagePanel 객체의 DisplayTagID() 함수를 통해 화면에 출력한다.

그 이후, Validation 객체의 Valid() 함수를 사용하여 태그의 불량 여부를 판단하고 판단 결과를 ImagePanel 객체의 DisplayFault() 함수를 통하여 출력한다. 불량이라고 판단되면, Sensor 객체의 ExtractFaultTag() 함수를 통하여 태그를 추출한다.

II-3. LabVIEW OOP 클래스의 설계

변환 규칙을 적용하여 UML의 클래스 다이어그램을 LabVIEW OOP 클래스로 적용한 결과를 표 1에서 제시하였다.

표 1. UML 클래스 다이어그램에서 객체트리로의 1:1 변환

UML 클래스 다이어그램	전환	객체 트리
frmMain	→	frmMain
ImagePanel	→	ImagePanel
Sensor	→	Sensor
Machine_Controller	→	Machine_Controller
RFID_Reader	→	RFID_Reader
Validation	→	Validation
Options	→	Options
DBWrapper	→	DBWrapper

또한, 각각의 UML 클래스의 메소드는 각각의 VI로 변환된다. 표 2에서 UML 클래스의 각 메소드가 객체 트리내의 클래스의 각 VI로 변환되는 것을 정리하였다.

표 2. UML 클래스의 메소드에 대한 LabVIEW OOP의 1:1 매칭 관계

UML 클래스	메소드	전환	VI	객체 트리
frmMain	PressBtnStart()	→	PressBtnStart.vi	frmMain
Sensor	get()	→	get.vi	Sensor
	set()		set.vi	
	InitSensor()		InitSensor.vi	
RFID_Reader	ReadTag()	→	ReadTag.vi	RFID_Reader
Machine_Controller	DetectTag()	→	DetectTag.vi	Machine_Controller
	ExtractFaultTag()		ExtractFaultTag.vi	
ImagePanel	CreateScreen()	→	CreateScreen.vi	ImagePanel
	DisplayDetection()		DisplayDetection.vi	
	DisplayFault()		DisplayFault.vi	
	DisplayTagID()		DisplayTagID.vi	
	DisplaySensorStatus()		DisplaySensorStatus.vi	
Validation	InformTag()	→	InformTag.vi	Validation
	Valid()		Valid.vi	
Options	ReadOpti	→	ReadOpt	Options

	on()		ion.vi	
DBWrapper	ReadOptionDB()	→	ReadOptionDB.vi	DBWrapper
	SaveFaultTag()		SaveFaultTag.vi	
	SensorOption()		SensorOption.vi	

UML 클래스에서 설계된 클래스의 데이터(속성)에 대한 정의는 LabVIEW OOP에서는 클래스 프라이빗 데이터(Class Private data)로 정의된다. 표 3에서 UML 클래스의 클래스 데이터 정의가 LabVIEW OOP의 클래스 프라이빗 데이터로 정의되는 것을 정리하였다

표 3. UML 클래스의 데이터 타입에 대한 LabVIEW OOP의 1:1 매칭 관계

UML 클래스	클래스 데이터	전환	클래스 프라이빗 데이터	객체 트리
frmMain	startbutton:BOOL	→	frmMain.ctl	frmMain
Sensor	TagSN:String	→	Sensor.ctl	Sensor
	DeviceID:String			
RFID_Reader		→	RFID_Reader.ctl	RFID_Reader
Machine_Controller		→	Machine_Controller.ctl	Machine_Controller
ImagePanel	DetectSensor:BOOL	→	ImagePanel.ctl	ImagePanel
	FaultSensor:BOOL			
Validation		→	Validation.ctl	Validation
Options	TCPIPAddress:String	→	Options.ctl	Options
DBWrapper	DBName:String	→	DBWrapper.ctl	DBWrapper
	TableName:String			

표 3에서 제시한 바와 같이 UML 클래스에서 정의된 데이터 타입은 LabVIEW OOP에서 LabVIEW OOP의 데이터 타입 정의를 가지는 클래스 프라이빗 데이터(Class Private Data)로 정의된다.

III. 결 론

LabVIEW OOP는 객체지향과 그래픽 UI를 가지는 내장형 시스템을 제어하기 위한 제어도구이다. 이러한 도구를 이용하기 위해, 개발자는 객체지향적인 개념을 충분히 가지고 설계를 하는 것이 바람직하다. 이에 본 논문에서는 객체지향을 설계하기 위한 UML을 접목하여 좀더 효율적이고 직관적인 내장형 시스템을 설계 및 구현하기 위한 방법을 제시하였다.

적용결과, 논문에서 제시하는 변환 방법을 사용하여 LabVIEW OOP를 이용한 내장형 시스템을 객체지향 개념을 가지면서 내장형 시스템을 설계할 수 있었다.

참고문헌

[1] 이노우에 타츠키, “다이어그램으로 쉽게 배우는 UML”, 한빛미디어, 2008.

[2] 강현구, 천두완, 김수동, “UML 2.0 기반 객체지향 모델링 프로세스 및 지침”, 한국정보과학회 가을 학술발표논문집 Vol.31 No.2, pp.307-309, 2004.

[3] “UML (Unified Modeling Language)”, http://www.uml.org/

[4] OMG, “UML 2.0 Infrastructure-Final Adopted Specification”, OMG document ptc / 03-09-15, 2003.

[5] OMG, “UML 2.0 Diagram Interchange - Draft Adopted Specification”, OMG document ptc/03-07-03, Available at: http://www.omg.org/cgi-bin/doc?ptc/2003-07-03, 2003.

[6] Tutorium, “LabVIEW Object-Oriented Programming: The Decisions Behind the Design”, http://zone.ni.com/devzone/cda/tut/p/id/3574, 2009.

[7] D. Beck, “LabVIEW-DIM Interface”, Proc., 2005.

[8] Fowler M, Scott K, UML Distilled Second Edition, Addison Wesley, 2000.

[9] Fowler M, Scott K, UML Distilled Third Edition, Addison Wesley, 2004.