Statechart에서의 XMI기반 소스코드생성 알고리듬 구현 연구

김진만*°, 임좌상**
*상명대학교 컴퓨터과학과, **상명대학교 디지털 미디어학부

Implementation of code generation algorithm for Statechart based on XMI

Kim Jin Man°, Lim Joa Sang
Sangmyung University
E-mail : jmkim@smu.ac.kr, jslim@smu.ac.kr

요 약

MDA는 UML과 XMI 표준을 사용해 플랫폼에 무관한 설계모델에서 플랫폼에 의존적인 실행 가능한 모델을 생산하는 것이다. 본 연구에서는 UML의 Statechart를 대상으로 XMI 정보를 추출하고 이로부터 Java 소스코드를 자동 생성하는 알고리듬을 구현하였다. 에어컨 시스템을 사례로 비즈니스 로직이 포함된 소스코드를 자동 생성하는 알고리듬을 구현하였으며 XMI의 사용으로 두 개의 다른 CASE 도구 각각에서 설계된 모델이 동일한 소스코드를 생성함을 확인 했다.

1. 서론

설계모델을 정의하고 그로부터 소스코드를 생성하면 일관되고 정확한 소스코드를 제공할 수 있다는 점에서 MDA (Model Driven Architecture)는 '설계모델'에서 '실행 가능한 구현 모델'을 자동 생성하는 기술로써 2001년 OMG에서 제안하였다. MDA는 이러한 모델을 통해 플랫폼에 무관한 실행 가능한 소스코드의 구현을 가능케 한다. MDA의 핵심 요소로 UML (Unified Modeling Language)과 XMI (XML Metadata Interchange) 표준이 사용된다. UML은 소프트웨어 시스템을 모델링하기 위한 언어이고 XMI는 메타데이터 변환 표준으로 모델간 변환을 위해 사용한다.

Together, Rose와 같은 대부분의 CASE도구에서 MDA를 지원하고 있다. 그러나 이들 도구에서의 모델 변환은 서로간에 잘 이루어지지 않고 있고, 또한 설계모델로부터 생성된 소스코드가 도구마다 상이하다. 따라서 본 연구에서는 첫째로, XMI 정보를 사용하여 생성된 소스코드가 일관성을 유지하고 있는지 확인하기로 한다.

Together, Rose와 같은 CASE 도구 대부분은 Class 다이어그램의 소스코드 자동생성 기능을 가지고 있다. Class 다이어그램의 소스코드는 Class간 관계와 Class내 속성만으로 이루어져 있

어 소스코드의 활용성이 떨어진다. UML의 동적 모델 중 하나인 Statechart는 시스템 또는 객체의 상태, 객체 간 상태 변환을 모델링 할 수 있는 다이어그램이다. Statechart의 소스코드 생성과 관련된 연구에서는 객체의 상태를 정형적인 언어로 표현하여 시스템의 로직을 포함하는 코드를 생성했다. 하지만 대부분의 연구에서 Statechart의 소스생성 기법만을 언급하고 생성된 소스코드는 그 생성기법에 따라 매뉴얼코딩을 한 것으로 이는 설계모델의 해석에 따라 소스코드결과가 다르게 나올 수 있다. 따라서 본 연구에서는 둘째로, Statechart의 소스코드생성을 자동화 하기 위한 알고리듬을 구현하여 로직이 포함된 소스코드를 생성한다.

2. Statechart 관련 연구

표 1은 Statechart와 관련된 연구 중 소스코드 생성을 다룬 연구들이다.

[표 1] Statechart와 관련된 문헌

문헌	UML	XMI	CASE	생성기법
Ali[1]	State Active	N	Elevator Simulation	State Design Pattern (SDP)
Allegrini[2]	State	Y	Turnstile	SDP
Niaz[3]	State	N	Air Conditioner	SDP
Pintér[4, 5]	State	N	N	Extended Hierarchi-cal Automata
Niaz[6]	State	N	Cassette Player	SDP
Blech[7]	State	N	N	Switch/ Case Loop
Niaz[8]	State Class	N	Air Conditioner 외 5 개	SDP
Niaz[9]	State	N	Air Conditioner	SDP
Sengupta[10]	State Seque -nce	Y	Library	Sequence Diagram

2.1 관련 연구

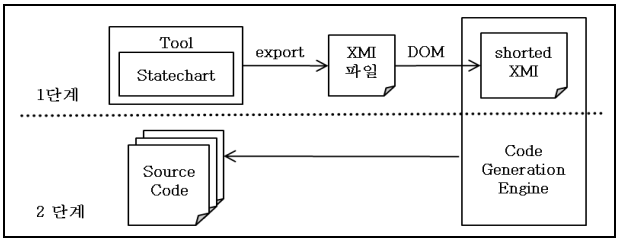
소스코드 생성면에서 Statechart는 문법 구현의 비정형성으로 인해 어려움이 있다[3, 5]. 표 1에서처럼 Statechart의 코드생성기법으로 State Design Pattern이 많은 연구에서 사용되었다[1-3, 6, 8, 9]. 특히, Niaz[3, 6, 8, 9]는 State Design Pattern을 확장하여 State간의 상속관계를 나타내었다. 각 State를 Class로, Composite State의 Substate의 모든 Transition과 Action을 Superclass 내에 포함하여 그 결과 생성된 코드는 Superclass 와 Subclass, 이들 사이의 Abstract Class를 나타낸다.

표 1에서 XMI에 대한 언급을 한 연구는 2편[2, 10]으로 이 중 Senqupta et al. [10] 연구에서는 Sequence 다이어그램을 OCL과 XMI를 사용하여 소스코드를 생성하고 생성된 소스코드를 사용하여 Statechart로 변환하여 XMI를 통한 모델간 변환을 보여주었다.

3. 소스생성 절차와 알고리듬

본 연구에서는 소스코드 생성기법으로 State Design Pattern을 사용한다. 이 기법은 State의 Substate를 Abstract Class 계로 표현하여 코드화 할 수 있다. 그러나 Hierarchy, History, Concurrency의 표현은 해결되지 않고 남아 있다[5]. 따라서 본 연구에서는 이들의 소스코드 표현을 특정 사례에 한정적으로 구현하였다.

3.1 생성 절차와 알고리듬

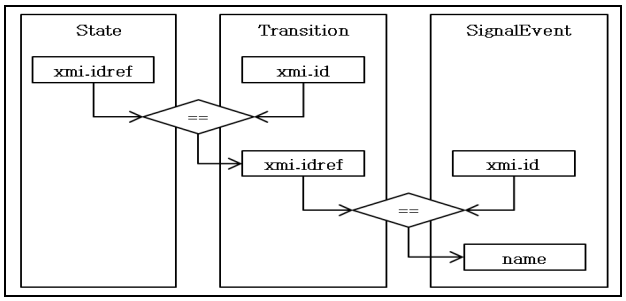


[그림 1] 소스코드생성 절차

본 연구에서는 CASE 도구에서 생성한 XMI가 스타일을 포함하고 있어 이를 줄이고 이를 바탕으로 소스코드를 생성하였다.

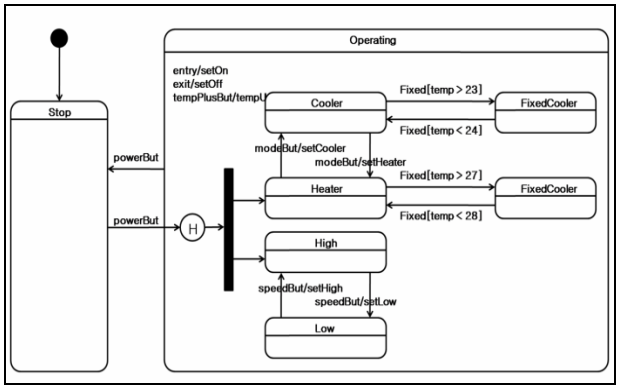
생성 절차는 그림 1과 같이 2단계로 구성되었다. 1단계에서는 CASE도구를 사용하여 Statechart로부터 XMI파일을 생성하고 DOM 파서를 사용하여 생성된 XMI 파일을 읽어 들여 소스코드 생성에 필요한 XMI 노드의 정보를 1차적으로 추출한다. 2단계에서는 1단계에서

추출된 정보를 입력으로 소스코드 생성 알고리즘을 통해 JAVA 소스코드를 생성한다. 그림 2는 1단계에서 소스코드에 필요한 다이어그램의 메타정보를 XMI파일에서 추출하는 알고리듬의 도안이다. State의 Transition은 State 노드의 xmi.idref를 통해 Transition 정보를 찾고 Transition의 xmi.idref를 통해 SignalEvent 노드의 name 정보를 찾아 State의 Transition의 Event 정보를 찾을 수 있다.



[그림 2] State의 Transition명 검색 절차

4. 적용 사례 - Air Control System



[그림 3] 에어컨 시스템의 Statechart

에어컨 시스템은 비교적 간단한 비즈니스 로직을 담고 있다. 그림 3의 에어컨 시스템은 온도와 풍속을 동시에 컨트롤할 수 있고 이를 저장할 수 있는 기능을 가지고 있다. 본 연구에서는 에어컨 시스템을 고정 온도와 고정 풍속을 제한하는 ECA (Event, Control, Action)을 포함하고 시스템이 종료될 때 마지막 수행 상태를 저장하는 History State를 포함하였다. 이의 Statechart에서 XMI를 생성하고 DOM 파서를 사용하여 그림 4와 같은 소스코드생성에 필요한 정보만을 추출하였다. 추출된 정보를 소스코드화한 결과로 그림 6과 같은 Java 소스코드를 생성했다. 이와 같은 과정을 거쳐 자동 생성된 에어컨 컨트롤 시스템의 Java 소스코드는 그림 5와 같다.

```
START {name StartState1} {xmi.id G.3} {xmi.idref G.2}
SState {name Stop} {xmi.id G.1} {xmi.idref G.0}
CompositeState {name Operating} {xmi.id G.27} {xmi.idref G.7}
InternalTransition {name entry} {xmi.id G.4} {source G.27} {target G.27
InternalAction {name entry} {name setOn}
InternalTransition {name exit} {xmi.id G.5} {source G.27} {target G.27}
---
CompositeSub {name Operating} {name Cooler} {xmi.id G.10} {xmi.idref G.
CompositeSub {name Operating} {name Heater} {xmi.id G.13} {xmi.idref G.
CompositeSub {name Operating}  {name High} {xmi.id G.15}
CompositeSub {name Operating} {name Low} {xmi.id G.17}
---
Transition {name {link: Stop -> History1}} {xmi.id G.0} {source G.1} {t
Transition {name {link: StartState1 -> Stop}} {xmi.id G.2} {source G.3}
Transition {name {link: Operating -> Stop}} {xmi.id G.7} {source G.27}
Transition {name {link: Cooler -> Heater}} {xmi.id G.8} {source G.10} {
TranAction {name setHeater} {xmi.idref G.30}
Transition {name {link: Cooler -> FixedCooler}} {xmi.id G.9} {source G.
TranGuard {body {temp < 24}} {xmi.idref G.31}
Transition {name {link: Heater -> Cooler}} {xmi.id G.11} {source G.13}
---
```

[그림 4] 소스코드 생성에 필요한 XMI 정보 추출

```
class AirCon {
        AirConState state;      // state object

        // Reference for all the state objects
        Stop stopState;         Operating operatingState;
        Cooler coolerState;     Heater heaterState;
        High highState;         Low lowState;
        FixedCooler fixedcoolerState;
        FixedHeater fixedheaterState;
        History1 history1State;

        AirCon() {      // constructor
                // create state objects only once
                stopState = new Stop(this);
                operatingState = new Operating(this);
                coolerState = new Cooler(this, operatingState);
                heaterState = new Heater(this, operatingState);
                highState = new High(this, operatingState);
                lowState = new Low(this, operatingState);
                fixedcoolerState = new FixedCooler(this, operatingState);
                fixedheaterState = new FixedHeater(this, operatingState);
                history1State = new History1(this, operatingState);

                state = stopState;      // sets the default sate
        }

        void setState(AirConState st) {  // setting new state
                state = st;
                //sets the most recent active subsate of Operating state
                if (state.equals(operatingState)) {operating.substate
                                                        = history1State.his;}
                state.entry();  // executes the entry action
        }
        void fork(AirConState st) { // setting the concurrent states
        // * fork 내용 입력하기
        }

        // Delegates incoming Events to concrete state class
        void powerBut() {
                state.powerBut();
        }
        void modeBut() {
                state.modeBut();
        }
        void speedBut() {
                state.speedBut();
        }
        void Fixed() {
                state.Fixed();
        }
        void tempPlusBut() {
                state.tempPlusBut();
        }

        // Actions becomes methods in SuperContext class
        void entry() { }
        void exit() { }
        void tempPlusBut() { }
        void setHeater() { }
        void setCooler() { }
        void setLow() { }
        void setHigh() { }
}

...
```

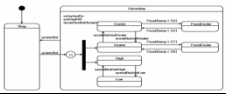
[그림 5] 자동 생성된 Java 소스코드 (일부)

5. 소스생성과 XMI 검증

본 연구의 목적을 위해 두 개의 도구 각각에 동일한 에어컨 컨트롤 시스템을 설계하고 XMI 파일로 각각 변환, 이를 본 연구에서 구현한 소스코드생성 알고리듬에 그 정보를 입력하여

JAVA 소스코드를 생성한다.

[표 2] 소스코드 생성 산출물 비교

	Borland Together Architect(2005)	IBM Rational Rose (v 7.0.0)
Model		
Export format	UML 1.4/XMI 1.1	UML 1.4/XMI 1.1
XMI	813 LOC	1577 LOC
Source Code	110 LOC	110 LOC

그 결과로 각 도구에서 생성된 XMI 파일은 약 2배 정도의 차이를 보이고 있지만 본 연구에서 구현한 엔진을 통해 생성된 코드의 LOC 는 도구간에 차이가 없음을 보이고 있다. 이는 XMI를 사용으로 동일한 설계모델에서 동일한 소스코드 생성의 가능성을 확인시켜주었다.

6. 결론

본 연구에서는 UML Statechart의 XMI 정보를 State Design Pattern을 사용하여 객체의 동적 모델을 소스코드화 하는 생성 알고리듬을 구현하였다. 이에 대한 에어컨 시스템 사례를 대상으로 110줄의 소스코드가 생성되었으며 이 코드에는 시스템의 비즈니스 로직이 표현되어 있다. 또한 XMI를 통해 설계모델을 코드화한 결과 동일한 JAVA 소스코드가 생성되어 XMI의 사용으로 소스코드의 일관성을 유지할 수 있음을 확인하였다. 그러나 본 연구에서는 UML의 최신 버전(2.0)을 고려하지 않았으며 사례에 있어 그 규모가 작아 본 연구에서 구현한 소스생성 알고리듬의 효과를 입증하기엔 부족한 점이 있다. 따라서 향후 연구로 UML 2.0의 메타모델 정보를 사용하고 도메인을 웹으로 국한하여 실제 사례에 적용 가능하게 만들고자 한다.

[참고문헌]

1. Ali, J. and J. Tanaka, *Implementing the dynamic behavior represented as multiple state diagrams and activity diagrams.* ACIS Int. J Comp. Inf. Sci., 2001. VOL. 2(1): p. 24-36.
2. Allegrini, T., *Code generation starting from statecharts specified in UML*. 2002, Universita Degli Studi di Pisa.
3. Niaz, I.A. and J. Tanaka. *CODE GENERATION FROM UML STATECHARTS*. in *The 7th IASTED International Conference on Intelligent Systems and Control (SEA 2003)*. 2003.
4. Pintér, G. and I. Majzik. *AUTOMATIC CODE GENERATION BASED ON FORMALLY ANALYZED UML STATECHART MODELS*. in *Formal Methods for Railway Operation and Control Systems*. 2003.
5. Pintér, G. and I. Majzik, *PROGRAM CODE GENERATION BASED ON UML STATECHART MODELS.* Periodica Polytechnica Electrical Engineering, 2003. VOL. 47, NO. 3–4: p. 187–204.
6. Niaz, I.A. and J. Tanaka. *MAPPING UML STATECHARTS TO JAVA CODE* in *IASTED International Conf. on Software Engineering (SE2004)*. 2004.
7. Blech, J.O., S. Glesner, and J. Leitner. *Formal Verification of Java Code Generation from UML Models*. in *the 3rd International Fujaba Days 2005*. 2005.
8. Niaz, I.A., *Automatic Code Generation From UML Class and Statechart Diagrams*, in *Graduate School of Systems and Information Engineering* 2005, Tsukuba.
9. Niaz, I.A. and J. Tanaka, *An Object-Oriented Approach To Generate Java Code From UML Statecharts.* Computer & Information Science, 2005. VOL. 6, NO. 2.
10. Sengupta, S., A. Kanjilal, and S. Bhattacharya, *Automated Translation of behavioral models using OCL and XML* TENCON 2005 IEEE Region 10, 2005: p. 1-6.