논문 2011-1-6

UML Activity 모델의 Colored Petri Net 모델로의 변환

정효택*

Transformation of an UML Activity Model into a Colored Petrin Net Model

Hyo Taeg Jung*

요 약

UML(Unified Modeling Language)이 객체지향 소프트웨어 개발의 산업 표준으로 자리 잡음에 따라 많은 시스템 모델들이 UML로 표현되고 있다. UML은 시스템을 여러 관점에서 모델링 할 수 있도록 많은 모델 들을 제공하고 있다. 예를 들면 시스템을 기능적인 관점 및 정적인 관점에서 모델링을 지원할 수 있도록 use case 모델 및 클래스 모델을 각각 제공하고 있으며, 다이내믹한 관점에서 모델링을 지원할 수 있도록 액티비티 모델(activity model)과 시퀀스 모델(sequence model)을 제공한다. 특히 액티비티 모델은 상세한 비즈니스 로직을 모델링하는 데는 가장 적합하다고 알려져 있으며, 활용성이 매우 높은 모델 중 하나이다. 본 논문에서는 UML의 액티비티 모델을 Colored Petri Nets (CPN) 모델로 변환하는 매핑법칙과 변환 알고리즘을 제안한다.

Abstract

As the Unified Modeling Language (UML) becomes an industrial standard for object-oriented software development, many system models have been specified in UML notation. For example, a system can be described in terms of the functional view through the use case model, the static view through the class model, and the dynamic view through activity or sequence model. In particular, activity model has more to do with the subject of the modeling and the experience of the modeler; for business modeling, for modeling the logic captured by a single use case or for modeling the detailed logic of a business rule. In this paper we propose mapping rules and a transformation algorithm to translate a UML activity diagram into a Colored Petri Nets (CPNs).

한글키워드 : 액티비티 모델, 매핑법칙, 변환 알고리즘

1. 서 론

UML 모델을 Petri Net 모델로 변환하는 매핑 법칙과 변환 알고리즘에 대한 연구가 다수 행해

* 한국전자통신연구원 콘텐츠연구본부
(email: htjung@etri.re.kr)
접수일자: 2011.3.15 수정완료: 2011.5.11

져 왔다. 일반적으로 상태도(statechart)나 액티비티 다이어그램과 같은 행위 다이어그램(behavioral diagram)과 인터랙션 다이어그램은 변환 알고리즘의 입력으로 종종 사용되었다.

한편 Constraints-based Modular Petri Net(CMPN)[2], Colored Petri Nets (CPN)[2], Extended Colored Petri Nets(ECPN)[3], High-Level Timed Petri Nets(HLTPN)[4] 등의 Petri Net은 출력으로 종종 사용되었다[5,6]. 특히 Petri Net을 확장한 High-Level Petri Net(HLPN) 은 추상 데이터 타입 이론과 네트 이론(net theory)을 통합하였으며, CPN과 Predicate/ Transition Net[7]이 그 일종이다. Extended CPN (ECPN)과 더불어 CPN은 UML 모델 변환의 출력으로 가장 널리 이용되었다. CPN을 지원하는 도구로는 CPNTools[8]가 있으며, 학계와 산업계 등에 4000본 이상이 배포되어 있다.

본 논문에서는 UML의 액티비티 모델을 CPN 모델로 변환하는 매핑법칙과 변환 알고리즘을 소개하고, CPNTools를 이용하여 제안된 법칙과 알고리즘의 타당성을 확인하였다.

2. 관련연구

W. J. Lee는 use case 모델의 비정형적인 특성을 정형화하는 기법을 제공하는 Constraints-based Modular Petri Nets(CMPN)을 제안하였다[1]. 이 접근법은 use case 모델이 정형적인 신택스(syntax)와 시맨틱스(semantics)를 제공하지 못할 뿐만 아니라, use case 간의 인터랙션과 전체 시스템의 행위(behavior)를 효과적으로 표현하지 못한다는 점에 착안하여, use case를 CMPN으로 변환하는 프로시져를 지원하고 CMPN 모델의 일관성(consistency)과 완전성(completeness)을 위한 가이드라인을 제공하고 있다.

J. M. Fernandes는 use case 다이어그램과 시퀀스 다이어그램에 기반을 둔 UML 모델을 CPN 모델로 변환하는 변환기법을 제안하였다[9]. 이 기법은 UML 2.0 시퀀스 다이어그램의 새로운 fragment type 중에서 option, alternative, parallel, loop, reference에 대한 변환방법을 제공하고 있다.

H. Störrle은 UML 2.0 액티비티 다이어그램을 CPN 모델의 시멘틱스와 신택스로 간결하게 표현하였으며, CPNTools로의 매핑 기법을 제공하였다[10,11]. 이 기법은 액티비티 다이어그램의 요소들을 CPN의 요소로 매핑하는데 직관적인 방법을 사용함으로써 타당성 확인이 필요하다.

M. E. Shin은 대규모 시스템의 다이내믹한 행위를 분석하기 위한 모델 변환법을 제시하였다[12,13]. use case 다이어그램, 클래스 다이어그램, 협동 다이어그램(collaboration diagram) 에 기반을 둔 시스템 모델이 CPN 모델로 변환되어 소프트웨어 개발 초기 단계에서 deadlock을 체크하거나 use case의 시나리오를 분석하는데 활용된다.

Z. Hu는 상태도와 협동 다이어그램을 CPN 모델과 매핑함으로써 다이내믹 모델의 분석이 가능한 UML-CPN 변환 프레임워크를 제안하였다[14]. 먼저 상태도가 CPN 모델로 변환되고 다음에 협동 다이어그램이 이러한 객체 모델간의 관계를 안내하기 위해 사용된다.

Y. Shinkawa는 use case 다이어그램, 액티비티 다이어그램, 상태도 등에 기반을 둔 UML 모델 간의 일관성을 유지하는 모델링 프로세스를 제안하였다[15]. 이 기법은 CPN 모델을 UML 액티비티 다이어그램, 시퀀스 다이어그램, 상태도로 변환하는 방법을 제공한다.

S. Yao는 ECPN 모델을 복합상태(composite

state)를 가지고 있는 상태도로 변환하는 접근법을 제안하였다[3]. ECPN에서 플레이스(place)는 객체의 메시지나 이벤트를 가지고 있으며 inhibitor는 synchronous 통신과 트랜지션의 우선순위를 표현하는데 사용된다.

ECPN 모델은 개별 객체의 상태 전이와 객체들간의 인터랙션을 정형 명세하는데 사용되며, Petri Net 이론에 근거하여 모델들의 일관성을 확인하는데 사용된다.

3. 매핑법칙과 변환 알고리즘

본 장에서는 UML의 액티비티 다이어그램을 CPN으로 변환하는 매핑법칙과 변환 알고리즘을 제안한다. UML 액티비티 다이어그램은 시스템의 행위나 활동들을 모델링하는데 주로 사용되며, CPN은 토큰이 여러 종류의 데이터 타입을 지원함으로써 플레이스(place)와 트랜지션(transitions)의 갯수를 줄일 수 있음으로써, 복잡한 시스템의 행위를 간단하게 모델링하는데 주로 사용된다. 또한 CPN은 신택스와 시멘틱스가 잘 정의되어 있기 때문에 정형적이고 수학적인 표현이 가능한 장점이 있다.

3.1 UML 액티비티 다이어그램의 메타모델

그림 1은 activity node, activity edge, activity group으로 구성되어 있는 UML 액티비티 다이어그램의 메타모델을 보여주고 있다.

activity node는 activity edge와 연결되어 액티비티들의 흐름을 활성화하는 추상 노드로서, object node, executable node, control node로 구성되어 있다. activity edge는 토큰을 가지고 activity node 간을 연결하는 추상 클래스로서, control flow와 object flow로 구성되어 있다. activity group은 activity에 있는 노드와 에지를 식별하는 추상 클래스이다.

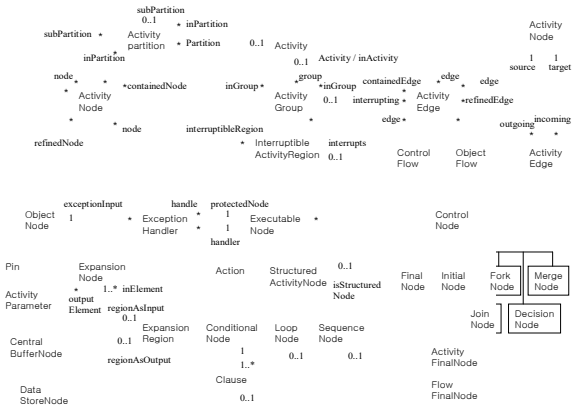
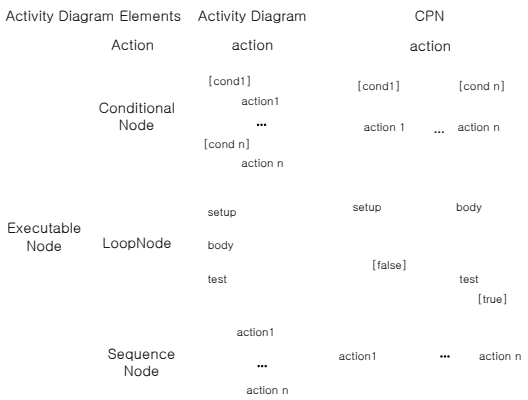


그림 1. UML activity diagram의 메타모델

3.2 매핑법칙

본 절에서 제안하는 매핑법칙에서는, 한 플레이스는 (또는 트랜지션) 다른 플레이스(또는 트랜지션)와 직접 연결될 수 없다는 Petri Net의 기본적인 제한사항을 만족한다.



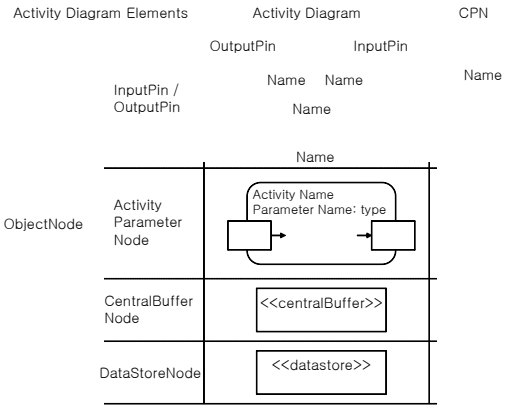


그림 3. executable node 매핑법칙

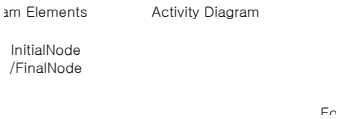


그림 4. control node 매핑법칙

그림 5. activity edge 매핑법칙

1) object node (그림 2)
- pin(inputpin과 outputpin)은 명칭(name)을 가진 플레이스로 매핑된다.
- activity parameter node는 타입명칭(type name)을 가진 플레이스로 매핑된다.
- central buffer node와 data store node는 플레이스로 매핑된다.

2) executable node (그림 3)
- action은 트랜지션으로 매핑된다.
- conditional node에 있는 action은 트랜지션으로 매핑된다. 이때 트랜지션은 입력 플레이스(input place)와 출력 플레이스(output place)에 연결된다. conditional node에 있는 condition은 아크(arc)의 감시 조건(guard condition)으로 매핑된다.
- loop node에 있는 setup, body, test는 각각 트랜지션으로 매핑되고, test에 있는 condition은 아크의 감시 조건으로 매핑된다.
- sequence node에 있는 action은 아크를 통해 플레이스와 연결되는 트랜지션으로 매핑된다.

3) control node (그림 4)
- initial node, final node는 플레이스로 매핑된다.
- fork node와 join node는 트랜지션으로 매핑된다.
- merge node와 decision node는 플레이스로 매핑된다.

4) activity edge (그림 5)
- control flow와 object flow는 아크로 매핑된다.

액티비티 다이어그램의 activity edge에 관련된 법칙은 모든 에지에 다 적용될 수 없으며, 예외법칙이 존재한다. 예를 들면 그림 6처럼 action이 서로 연결되어 있거나 merge 혹은 decision node와 연결되어 있는 경우 위의 법칙을 적용할 경우, CPN 모델에서 트랜지션과 트랜지션이 혹은 플레이스와 플레이스가 직접 연결됨으로써 Petri Net의 기본적인 제한사항을 위배하는 경우가 발생하게 되며, 이런 경우에는 예외법칙을 적용하여야 한다.

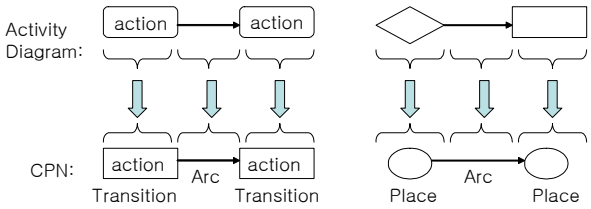


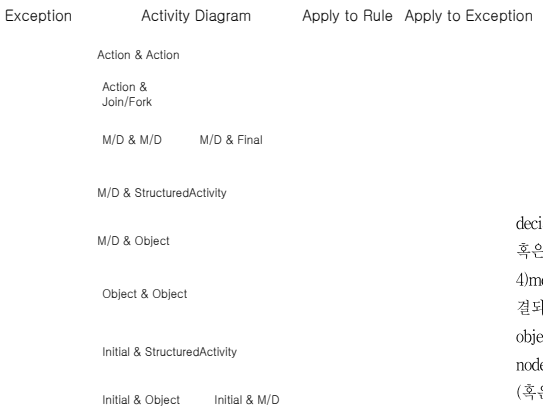
그림 6. 예외법칙이 필요한 경우

5) 트랜지션-트랜지션 간의 예외법칙 (그림 7)

- 두 트랜지션 간의 아크는 다음의 경우에는 incoming 아크와 outgoing 아크를 가진 플레이스로 매핑된다: 1) action이 action으로 연결되어 있는 경우; 2)action이 join(혹은 fork)로 연결되어 있거나, 그 반대일 경우

6) 플레이스-플레이슨 간의 예외법칙 (그림 7)

- 두 플레이스 간의 아크는 다음과 같은 경우 incoming 아크와 outgoing 아크를 가진 트랜지션으로 매핑된다: 1)merge(혹은 decision) node가 다른 merge(혹은 decision) node로 연결되어 있는 경우; 2)merge(혹은 decision) node가 final node에 연결되어 있는 경우; 3)merge(혹은 decision) node가 structured activity node나 혹은 그 반대로 연결되어 있는 경우; 4)merge(decision) node가 object node로 연결되거나 그 반대일 경우; 5) object node가 object node로 연결되어 있을 경우; 6) initial node가 structured node, object node, merge (혹은 decision) node와 연결되어 있는 경우



3.3 변환 알고리즘

본 절에서는 3.2절에서 제안한 매핑법칙을 기본으로 하여 UML 액티비티 다이어그램을 CPN으로 변환하는 알고리즘을 제안한다. 표 1에서는 알고리즘에 사용된 function 들을 보여주고 있다.

표 1. 알고리즘에 사용된 function들

하여 수행한다.

(5) 변환된 노드와 에지들을 식별한다. 만약 노드가 n번 변환되었다면 그 노드와 연결된 요소들을 n-1번 제거하고, 에지를 CPN 요소에 재배치한다.

(6) findNextNode를 인보킹함으로써 탐색 알고리즘을 이용하여 변환되어야 할 다음 노드를 찾는다.

(7) 탐색할 노드가 없을 때 까지 (1) ~ (7)을 반복 수행한다.

표 2와 표3은 변환 알고리즘 및 변환 알고리즘의 검증의 일부분을 보여준다.

표 2. 변환 알고리즘

```
/* Transformation Algorithm to translate UML activity diagram into CPN model
Input: A UML activity diagram AD = (ADNode, ADEdge), where
(1) ADNode is a set of activity nodes, and consists of a set of object nodes,
    executable nodes, and control nodes,
    ADNode = (ObjNode, ExecNode, CntlNode).
    The number of ADNode is described as |ADNode| = N.
(2) ObjNode is a set of object nodes and, consists of a set of pins, activity
    parameters, central buffer nodes, data store nodes, and expansion nodes.
(3) ExecNode is a set of executable nodes, and consists of a set of actions and
    structured activity nodes,
    ExecNode = (Action, StructActNode).
(4) StructActNode is a set of structured activity nodes, and consists of a set of
    conditional nodes, loop nodes, and sequence nodes,
    StructActNode = (CondNode, LoopNode, SeqNode).
(5) CondNode and LoopNode have a set of actConds and testConds as guard
    conditions, respectively.
(6) CntlNode is a set of control nodes, and consists of a set of initial nodes,
    final nodes, fork nodes, join nodes, merge nodes, and decision nodes,
    CntlNode = (InitNode, FinalNode, JoinNode, ForkNode, MergeNode,
                DecisionNode).
(7) ADEdge is a set of activity edges, and connects ADNode.5
    The number of ADEdge is described as |ADEdge| = E.

Output: A Colored Petri Nets CPN = (Place, Transition, Arc), where
(1) Place is a finite set of places.
(2) Transition is a finite set of transitions.
(3) Arc is a finite set of arcs such that Place ∩ Transition = Place ∩ Arc =
    Transition ∩ Arc = Φ.  */

Main Algorithm:
/* SourceNode and SourceEdge are sets of source nodes and edges in AD,
   respectively. TargetNode is a set of look-ahead nodes in AD. CheckedNode
   and CheckedEdge are sets of the transformed nodes and edges. Inscriptions
   of places, transitions, and arcs in CPN are not described. */
{
  |ADNode| = N;  // the total number of nodes in AD
  |SourceNode| = |CheckedNode| = |CheckedEdge| = 0;
  SourceNode = CheckedNode = CheckedEdge = Φ;
  CPN.Place = CPN.Transition = CPN.Arc = Φ;

  while (|ADNode| ≠ 0) {
      SourceEdge = Φ; TargetNode = Φ; //initialvalueforedgeandlook-aheadnode
      SourceNode = read(ADNode);  //read one node
      if (SourceNode != CheckedNode) { //check if the node is transformed
          SourceEdge = readAll(ADEdge); //read all edges connecting to the node
          // check whether the edges connecting to the node exist or not
          while (|SourceEdge| ≠ 0) {
              TargetNode = read(ADNode); // read the look-ahead node
              TransActivityCPN (SourceNode, TargetNode, CPN.Place,
                                CPN.Transition, CPN.Arc);
              |SourceEdge| = |SourceEdge| -1;
              // identify the transformed edge
              CheckedEdge = CheckedEdge ∪ SourceEdge;
              |CheckedEdge| = |CheckedEdge| + 1;
          } // while statement
          // identify the transformed node
          CheckedNode = CheckedNode ∪ SourceNode;
          |CheckedNode| = |CheckedNode| + 1 ;
          ADNode = ADNode – SourceNode;
          |ADNode| = |ADNode| – 1;
      } // if statement
      findNextNode(AD);  // find next node not transformed
  } //while statement
  return CPN(Place, Transition, Arc);
}
.....
```

표 3. 변환 알고리즘의 검증

```
Part1 (Verify of Inner While Loop)

/* Precondition: SourceEdge is a set of edges in AD which are not transformed
   into CPN ∧ |SourceEdge| = S, and CheckedEdge is a set of edges in AD
   which are transformed into CPN ∧ |CheckedEdge| = C. */

  while (|SourceEdge| ≠ 0) {
      TargetNode = read(ADNode);
      TransActivityCPN (SourceNode, TargetNode, CPN.Place,
       CPN.Transition, CPN.Arc);
      |SourceEdge| = |SourceEdge| -1;
      CheckedEdge = CheckedEdge ∪ SourceEdge;
      |CheckedEdge| = |CheckedEdge| + 1;
  }
/* Postcondition: S – |SourceEdge| = |CheckedEdge| – C.
Proof:
(1) On the initial entry to loop:
    From Precondition, |SourceEdge| = S and |CheckedEdge| = C, and also
    S – |SourceEdge| = S – S = 0, and |CheckedEdge| – C = C – C = 0.
    Therefore, Postcondition: S – |SourceEdge| = |CheckedEdge| – C is true.

(2) Suppose that Postcondition is true before a loop iteration.
    Assume |SourceEdge| = X, |CheckedEdge| = Y, and S – X = Y – C
    Then after the iteration:  |SourceEdge| = |SourceEdge| – 1, and
                               |CheckedEdge| = |CheckedEdge| + 1,
    thus, S – (X – 1) = (Y + 1) – C, i.e., S – X = Y – C.
    Therefore, after the iteration, Postcondition: S – |SourceEdge| =
    |CheckedEdge| – C is true.

(3) So long as the loop has not terminated,
    (Postcondition ∧ (SourceEdge ≠ 0))
    = S - |SourceEdge| = |CheckedEdge| - C   (SourceEdge ≠ 0)
    ⇒ |SourceEdge| > 0.
    After each iteration, |SourceEdge| = |SourceEdge| -1.

(4) On exit from the loop:
    (Postcondition ~(SourceEdge ≠ 0)) = (Postcondition  (SourceEdge = 0))
    ⇒ S – 0 = |CheckedEdge| – C, i.e., S = |CheckedEdge| – C
    Therefore, Postcondition: S – |SourceEdge| = |CheckedEdge| – C is true.
.....
```

4. 시뮬레이션

본 장에서는 간단한 시뮬레이션을 통하여 3장에서 제안한 매핑법칙과 변환 알고리즘의 타당성을 확인한다. 그림 8에서는 예금인출 기능을 액티비티 다이어그램으로 모델링하였다. 예금인출을 위해 핀넘버(pin number)와 인출금액을 입력하여야 하며, 만약 핀넘버가 부정확하거나 인출희망 금액이 예금잔고를 넘어서면 정상 인출이 되지 않는다. 액티비티 다이어그램으로부터 변환

된 CPN 모델을 CPNTools (덴마크의 University of Aarhus의 CPN 그룹에서 개발한 CPN 시뮬레이션 도구)를 사용하여 정상 인출과 비정상적 인출을 각각 시뮬레이션 함으로써 각 기능의 정확성을 확인한다.

4.1 정상 인출

그림 8의 모델은 매핑법칙과 변환 알고리즘에 따라 변환함으로써 그림 9와 같은 CPN 모델로 변환된다. 시뮬레이션을 위하여 CPNTools의 Simulation 기능 및 CPN ML 언어로 color sets, functions, constant values 등의 정보를 정의하고 이를 declaration에 기술해줌으로써 CPN 모델에서의 여러 조건들을 생성한다. 그림 9에서 보는 것처럼 핀넘버는 "11", 인출 희망금액은 "150", 잔고는 "800"으로 가정하였다. 그림 10에서는 정확한 금액이 인출되고 잔고도 정확하게 계산되었음을 보여주고 있다.

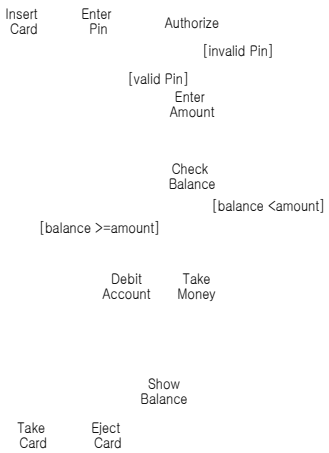


그림 8. activity diagram을 이용한 예금인출 모델링

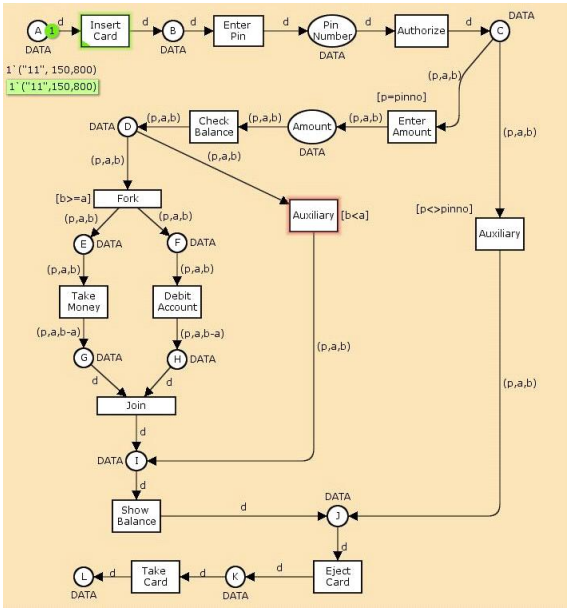


그림 9. 정상 인출의 초기 화면

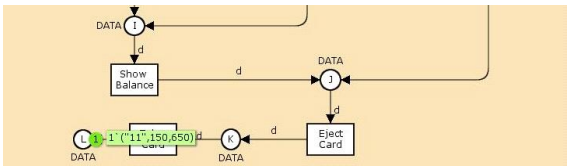


그림 10. 정상 인출된 최종 화면

4.2 부적합한 핀넘버

그림 8에서 예금자가 핀넘버를 입력하면 Authorize가 핀넘버가 정확한지를 체크한다. 그림 11에서 처럼 핀넘버로 "999"가 입력될 경우 Auxiliary 트랜지션과 연결된 플레이스 C가 enable되며, 그림 12에서 알 수 있듯이 정상적인 인출이 이루어지지 않았다.

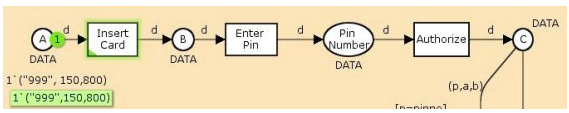


그림 11. 부적합한 핀넘버가 입력된 초기 화면

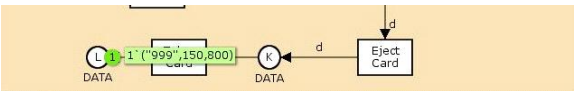


그림 12. 부적합한 핀넘버가 입력된 최종 화면

4.3 부적합한 인출금액

잔고보다 많은 금액을 인출하고자 할 경우 그림 8에서 CheckBalance가 인출희망 금액과 잔고를 비교, 확인하여 만약 그림 13에서 처럼 인출희망금액이 잔고보다 많은 경우 Auxiliary 트랜지션과 연결된 플레이스 D가 enable 되며, 그림 14에서 처럼 정상적인 인출이 이루어지지 않는다.

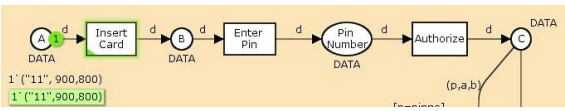


그림 13. 부적합한 인출금액이 입력된 초기화면

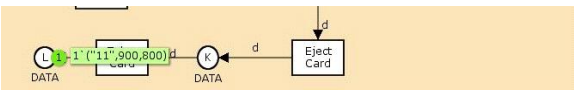


그림 14. 부적합한 인출금액이 입력된 최종 화면

5. 결론

본 논문에서는 UML activity 모델을 CPN 모델로 변환하는데 사용되는 매핑법칙과 변환 알고리즘을 제안하였다. 또한 간단한 예금인출 시뮬레이션을 통하여 변환된 기능들이 의도한대로 정확하게 실행됨을 보임으로써, 제안한 매핑법칙과 변환 알고리즘의 타당성을 간접으로 확인하였다.

다음과 같은 연구가 향후 후속연구로 진행될 수 있다. 첫째, 제안된 매핑법칙과 변환 알고리즘은 확장될 수 있다. 본 연구에서 다루지 않았던 stereotyped activity, note, OCL constraints, exception handler, expansion node, expansion region 등의 요소를 고려함으로써 UML 2.0의 액티비티 모델을 지원할 수 있게 될 것이다. 둘째, Petri Net 이외의 정형명세를 대상으로 본 연구를 확장할 수 있다. 예를 들면 LOTOS를 UML 모델로 변환하는 매핑법칙에 대한 연구가 시도된 적이 있지만 완전히 구현되지 않았으며, LOTOS도 좋은 대상이 될 수 있다. 셋째, 본 매핑법칙과 변환 알고리즘을 쌍방향으로 지원하는 도구의 개발이다. 반자동 혹은 자동화된 도구가 개발되어 이러한 쌍방향 변환을 지원한다면 UML 액티비티 모델이 가지고 있는 비정형성에 대한 보완이 정형 모델을 통하여 보완될 것이다.

참 고 문 헌

[1] W. J. Lee, S. D. Cha, and Y. R. Kwon, “Integration and Analysis of Use Cases Using Modular Petri Nets in Requirements Engineering,” IEEE Transactions on Software Engineering, vol. 24, no. 12, pp. 1115-1130, December 1998.

[2] K. Jensen, “Colored Petri Nets: Basic Concepts, Analysis Methods and Practical Use Volume I, II, and III,” Springer-Verlag, 1992.

[3] S. Yao, F. Tang, and Y. Liu, “An Object-Oriented Model for Parallel Softwares,” Proceedings of the 27th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS ‘98), pp. 245-250, Beijing, China, September 1998.

[4] L. Baresi, “Improving UML with Petri Nets,” Electronics Notes in Theoretical Computer Science, vol. 44, no. 4, pp. 107-119, July 2001.

[5] H. Jung and S. Joo, “Transformation of an

Activity Model into a Colored Petri Net Model," in Proceedings of the 2nd International Conference on Trends in Information Sciences & Computing 2010 (TISC 2010), Sathyabama University, Chennai, India, December 17-19, 2010.

[6] H. Jung and G. Lee, "A Systematic Software Development Process for Non-Functional Requirements," in Proceedings of the International Conference on ICT Convergence 2010 (ICTC 2010), Ramada Plaza Hotel, Jeju Island, Korea, November 17-19, 2010.

[7] H. J. Genrich, "Predicate/Transition Nets," LNCS 254, Springer-Verlag, pp. 207-247, 1987.

[8] L. Wells, "Performance Analysis Using CPN Tools," Proceedings of the 1st International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS), Pisa, Italy, October 2006.

[9] J. M. Fernandes, S. Tjell, J. B. JØrgensen, and Ó. Ribeiro, "Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Colored Petri Net," Proceedings of the 6th International Workshop on Scenarios and State Machines (SCESM '07), Minneapolis, Minnesota, May 2007.

[10] H. Störrle, "Semantics of Control-Flow in UML 2.0 Activities," Proceedings of the IEEE Symposium on Visual Language and Human Centric Computing (VLHCC '04), pp. 235-242, Rome, Italy, September 2004.

[11] H. Störrle, "Towards a Petri-net Semantics of Data Flow in UML 2.0 Activities," Technical Report 0504 (04-2005), Ludwig-Maximilians-Universität München, 2005.

[12] M. E. Shin, A. H. Levis, and L. W. Wagenhals, "Transformation on UML-based System Model to Design/CPN Model for Validating System Behavior," Proceedings of the 6th International Conference on the UML / the Workshop on Compositional Verification of UML Models, San Francisco, California, October 2003.

[13] M. E. Shin, A. H. Levis, L. W. Wagenhals, and D. S. Kim, "Analyzing Dynamic Behavior of Large-Scale Systems through Model Transformation," International Journal of Software Engineering and Knowledge Engineering, vol. 15, no. 1, pp. 35-60, 2005.

[14] Z. Hu and S. M. Shatz, "Mapping UML Diagrams to a Petri Net Notation for System Simulation," Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering (SEKE '04), pp. 213-219, Banff, Canada, June 2004.

[15] Y. Shinkawa, "Inter-Model Consistency in UML Based on CPN Formalism," Proceedings of the 13th Asia Pacific Software Engineering Conference (APSEC '06), pp. 411-418, Banglore, India, December 2006.

저 자 소 개

정 효 택

1986년 경북대학교 전자공학과 졸업 (학사)
1997년 연세대학교 컴퓨터과학과 (공학석사)
2008년 The University of Texas at Dallas Computer Science (S/W공학박사)
1988년~현재 한국전자통신연구원

<주관심분야> Software Engineering, Requirement Engineering, Software Architecture, Software Quality