KIPS Tr. Software and Data Eng.
Vol.4, No.11 pp.479~486 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2015.4.11.479

A Case Study on Model Checking Online-Game Server Party System Using SPIN

Goanghun Kim† · Yunja Choi††

ABSTRACT

Model checking method is able to check all possible cases automatically and is applicable to specifications or design before actual implementation so that some critical systems have adopted this method actively. However, the current practice of software verification is largely dependant on basic methods such as manual testing because of lack of understanding about this rigorous method and high verification cost. In this paper we conducted an experimental research for the automated verification using the SPIN model checker on an online-game server to study the applicability of the technique in this domain. The results show that we could verify major features of the online-game server party system with 5~7 GB memory and within 10 minutes execution time, and also found a hidden system error that passed existing testing process. This result shows the possibility of rigorous and effective verification with reasonable cost in comparison to manual testing.

Keywords : SPIN Model Checker, Online Game Server Verification

온라인 게임 서버의 파티 시스템 검증을 위한 스핀 모델 체커 적용에 관한 연구

김 광 훈† · 최 윤 자††

요 약

모델 체킹 방법은 가능한 모든 경우를 자동으로 확인할 수 있으며, 코드가 구현되기 이전의 명세서나 디자인의 검증에도 적용할 수 있어 고위험 시스템의 검증에 활발히 적용되어왔다. 그러나 이러한 엄밀한 검증기법에 대한 일반적인 이해 부족과 테스팅에 비해 높은 검증 비용으로 인하여, 일반적인 소프트웨어들은 여전히 인력에 의한 테스팅과 같은 기초적인 방법에 의존하여 검증이 수행되고 있다. 본 논문에서는 그 대표적인 예인 온라인 게임 서버를 대상으로, SPIN 모델 체커(SPIN model checker)를 이용한 자동화 검증 방법을 적용하는 실험적인 연구를 수행하여 검증 비용 대비 효과에 근거한 적용성을 판단하였다. 연구 결과, 5~7GB 이내의 메모리와 10분 이내의 시간 내에서 온라인 게임 서버 파티 시스템의 주요 특성들을 검증할 수 있음을 보였고, 이 과정에서 기존에 파악하지 못한 오류도 검출하였다. 이로부터 인력에 의한 테스팅에 비해 납득할만한 수준의 검증 비용으로 엄밀하고 효과적인 검증이 가능하다는 결론을 도출할 수 있었다.

키워드 : 스핀 모델 체커, 온라인 게임 서버 검증

1. 서 론

오늘날 우리는 수많은 정보통신기술(Information and Communication Technology, ICT) 시스템에 둘러싸여 있다. 가깝게는 우리 손에 들려 있는 스마트폰부터 시작해서 의료기기, 인터넷 뱅킹, 항공권 발매 시스템 등 ICT 시스템이 없는 생활은 상상하기 어렵다. 그에 맞춰 신뢰할 수 있는 ICT시스템에 대한 요구도 커지고 있다. 신뢰할 수 없는 ICT시스템의 경우 스마트폰 오작동과 같은 가벼운 오류를 유발할 수도 있지만 인텔 펜티엄 부동소수점 연산 오류로 인한 교환 사태 및 Therac-25 방사선 치료기기로 인한 사망사건 같은 치명적인 결과를 초래할 수도 있다[1]. 다행히 고위험 시스템(Critical system)[2]의 경우 대부분의 사람들이 시스템 신뢰도의 중요성을 잘 알고 있고 다양한 검증(Verification)[3] 방법을 사용하고 있다.

반면 오류 대응 비용이 상대적으로 적은 시스템의 경우 소프트웨어 검증 절차가 간략한 경우가 많다. 대표적으로

※ 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 ICT융합고급인력 과정지원사업의 연구결과로 수행되었음(IITP-2015-H8601-15-1002).
† 준 회 원 : 서울대학교 전기정보공학부 박사과정
†† 정 회 원 : 경북대학교 컴퓨터학부 부교수
Manuscript Received : April 6, 2015
First Revision : July 23, 2015
Second Revision : September 4, 2015
Accepted : September 6, 2015
* Corresponding Author : Yunja Choi(yuchoi76@knu.ac.kr)

온라인 게임 서버를 들 수 있는데, 이는 상시 서비스를 제공하는 시스템임에도 불구하고 비용의 문제에 의해 주로 인력에 의한 테스팅과 단위 테스팅(Unit testing)에 의존하고 있다. 하지만 온라인 게임 서버는 대량의 사용자가 동시에 서비스를 제공받기 때문에, 잠재적인 오류는 다수의 사용자에게 동시에 불편을 유발할 수 있는 큰 장애를 가져올 수 있다.

따라서 본 연구에서는 온라인 게임 서버에서 특정 모듈을 사례로 엄밀한 검증 방법인 모델 체킹(Model checking)을 수행하고, 이에 대한 검증 비용이 실제로 타당한지를 파악하는 실험적인 연구를 수행한다. 이를 위해 온라인 게임 서버를 모델링한 예시 및 검증하고자 하는 속성을 설명하고, 실제 모델 체킹 결과 및 검증에 소요된 비용에 대해 논의한다.

2. 모델 체킹

모델 체킹[4]이란 시스템을 추상화한 모델에 대해, 해당 모델이 시스템에서 반드시 만족해야 할 특성들을 만족하고 있는지를 확인하는 방법이다. 모델 체킹을 위해서는 우선 모델 명세 언어를 활용해 모델을 작성하고, 시제 논리를 활용해 시스템이 만족해야 하는 검증 특성을 표현해야 한다. 모델 체킹 도구는 모델과 검증 특성을 입력받아, 수학적 논리 증명과정을 통해 해당 모델이 검증 특성을 만족하는지를 입증하고, 만약 검증이 실패한다면 이에 대한 반례를 사용자에게 제공한다. 사용자는 이 반례를 활용해 소프트웨어에서 발생할 수 있는 위험성을 파악하고, 모델 혹은 검증 대상 시스템을 수정할 수 있다. Fig. 1은 이에 대한 과정을 도식화한 것이다.

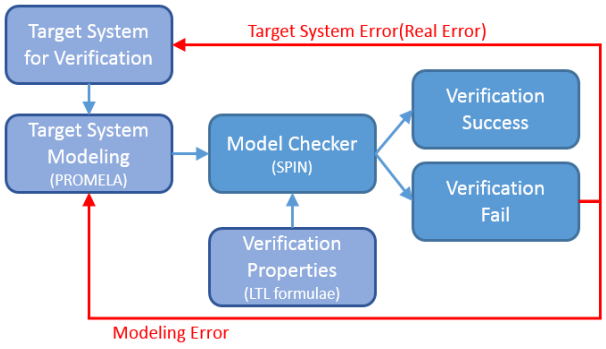

Fig. 1. Model Checking Process

모델 검증 도구 SPIN[5, 6]은 오픈 소스 소프트웨어로서 동시성(concurrency)을 갖는 소프트웨어 시스템의 검증에 주로 사용한다. SPIN에서 모델은 정형 명세 PROMELA로 표현되고, 검증 특성은 LTL(Linear Temporal Logic)의 형태로 표현된다. PROMELA는 비결정적 오토마타 및 비동기적 분산 프로세스의 설계를 지원한다. 또한 메시지 채널을 이용해 메시지의 동기적/비동기적 통신 모델링이 가능하다. 본 논문의 검증 대상인 온라인 게임 서버의 경우, 서버-사용자 형태의 분산 시스템으로 구성되며 메시지를 통해 통신하기 때문에 해당 기능들을 제공하는 PROMELA로 표현하기 적합한 시스템이다.

SPIN은 우주 임무를 위한 다수의 알고리즘[7, 8]이나, 차량 전장용 운영체제[9] 등의 안전중요성 시스템 검증에 성공적으로 활용되었다.

PROMELA는 C와 유사한 문법을 사용하며 Table 1에서 기본적인 예제 코드를 볼 수 있다. 이 예제는 두 고객과 그들의 주문을 처리하는 점원을 모델링하고 있으며 재고가 부족한 상황이 발생할 수 있는지를 검증하고 있다. 시작은 init 프로세스에서 하게 되며 init 내에서 하나의 점원 프로세스(ProcClerk)와 두 개의 고객 프로세스(ProcCustomer)를 실행한다. 이 네 개의 프로세스들은 임의의 순서대로 실행될 수 있으며 SPIN 모델 체커에서 가능한 모든 상황들을 확인하게 된다. 고객 프로세스는 주문 채널(g_orderChannel)을 통해 점원과 통신하게 되며 if 구문과 3개의 skip 구문은 무작위 선택을 위한 것이다. 결과적으로 고객 프로세스는 주문 채널에 한 개 상품 주문, 두 개 상품 주문, 세 개 상품 주문 중 하나를 전달한 후 종료한다. 점원 프로세스는 do 무한루프 내에서 주문 채널로 들어온 주문량을 확인해 재고 수량(g_stock)에서 빼는 처리를 반복 수행한다. 여기서 검증하고자 하는 것은 초기 재고 수량이 5일 때 두 명의 고객을 처리하는 과정에서 재고가 부족한 경우가 발생할 수 있는지를 확인하려는 것으로 LTL식 p0로 나타냈다. LTL식 p0는 재고 수량이 항상("[]"연산자) 0 이상이어야 한다는 것으로 이 조건을 만족하지 못하는 경우가 있으면 검증은 실패한다. 두 고객 모두 3개씩 주문할 때 재고 수량이 -1이 되고 이 경우를 확인한 SPIN 모델 체커는 실패를 반환한다.

Table 1. PROMELA Sample with a LTL Formula

```
chan g_orderChannel = [1] of { int }; // 주문 채널
int g_stock = 5; // 재고 수량

proctype ProcClerk() { // 점원 프로세스
  int currentOrder = 0;
  do
  :: g_orderChannel ? currentOrder ->
    g_stock = g_stock - currentOrder;
  od
}
proctype ProcCustomer() { // 고객 프로세스
  if
  :: skip -> g_orderChannel ! 1;
  :: skip -> g_orderChannel ! 2;
  :: skip -> g_orderChannel ! 3;
  fi
}
init {
  run ProcClerk(); // 점원 프로세스 실행
  run ProcCustomer(); // 고객1 프로세스 실행
  run ProcCustomer(); // 고객2 프로세스 실행
}
// 재고 수량은 음수가 될 수 없다
ltl p0 { [](g_stock >= 0) }
```

Table 1이 나타내는 시스템의 상태들을 그래프로 나타내면 Fig. 2와 같다. 스핀 모델 체커는 해당 그래프를 깊이 우선 탐색으로 순회하면서 LTL식 p0가 만족하는지 확인하며 점선으로 표시된 실패 경로에 이를 때 실패 경로와 함께 오류를 반환하게 된다.

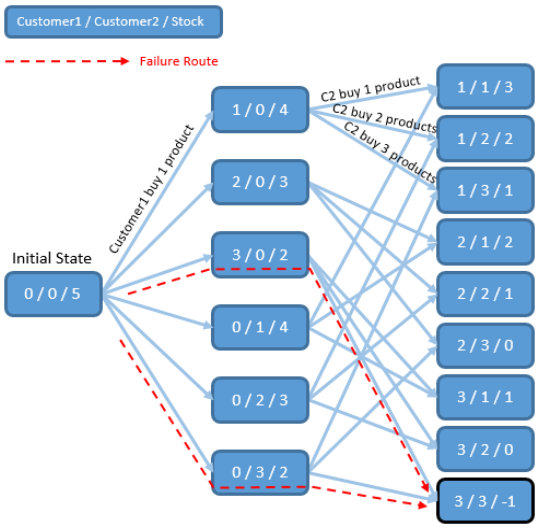

Fig. 2. State Graph for the Table 1 System

Fig. 2에서도 쉽게 볼 수 있듯이 전체 상태의 개수는 지수적으로 증가하게 되는데 이에 따라 메모리 사용량과 처리 시간도 같이 늘어나게 된다. 이 문제를 해결하기 위해 다양한 최적화 기법에 대한 연구가 진행되었으며[10], 이러한 기법들과 가용 메모리 증가로 인한 결과를 Fig. 3에서 볼 수 있다. Fig. 3은 벨 연구소에서 실험용 전화기 스위치를 대상으로 20년에 걸쳐 진행한 것으로 모델 체킹의 현실적인 적용 가능성을 보여준다.

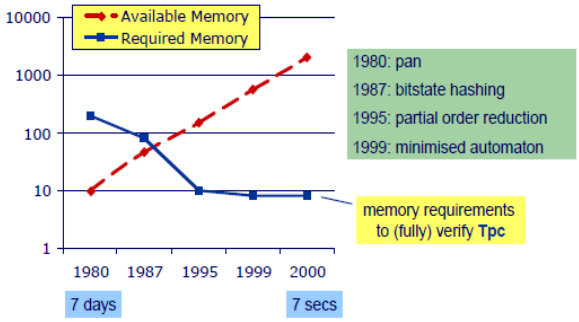

Fig. 3. Moore's Law & Advanced Algorithms Verification of Tpc(Experimental Telephone Switch)[10, 11]

3. 온라인 게임 서버의 파티 시스템

본 논문에서 사용할 검증 대상은 현재 개발 중인 R사의 온라인 게임 T의 파티 시스템으로, 현 시스템은 아직 개발자 내부 테스팅만 거친 상태이다.

게임에서의 파티 시스템은 혼자서 처리하기 어려운 적이나 임무를 여러 게임 사용자들과 협력하여 해결하게 해주는 그룹 구성 시스템이다. 파티는 여기서의 개개 그룹을 나타내며 다수의(보통 2~5인) 게임 사용자가 서로의 장점을 살려가며 게임을 하게 된다. 파티 내의 각 사용자를 파티원(員)이라 하고 초대, 추방 등의 권한을 가지고 있는 파티원을 파티장(長)이라 한다. 파티의 수명주기(life cycle)를 간략하게 나타내면 Fig. 4와 같다.

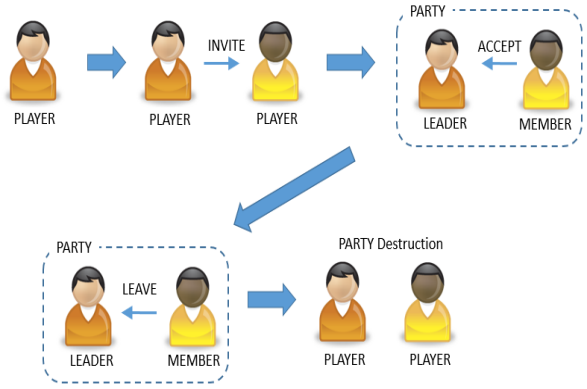

Fig. 4. Life-cycle of a Party

파티 시스템은 많은 온라인게임에서 지원하는 주요 요소로 시스템의 완전성(Integrity)을 유지하지 않으면 파티원 전체에게 정상적인 서비스를 제공할 수 없을 뿐 아니라 심각한 경우 이로 인해 서버 전체 장애를 유발할 수도 있다.

기존에도 Fig. 5에 나타난 것처럼 다양한 테스트 방법을 사용하고 있었지만 가능한 모든 경우를 검증하는 것은 쉽지 않으며 악의적인 사용자의 공격(프로토콜을 따르지 않은 무작위 패킷 발송 같은)까지 고려하게 되면 모든 경우를 검증한다는 것은 사실상 불가능하다.

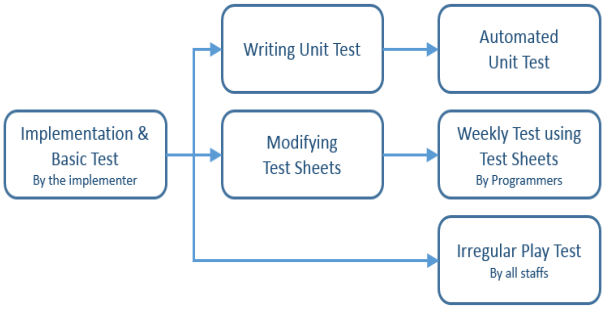

Fig. 5. Current Test Process

이로 인해 본 논문에서는 자동화가 가능하면서 동시에 모든 경우를 검증할 수 있는 모델 체킹을 이용한다. 모델 체킹은 오토마타 기반이므로 실수형 변수 사용같이 상태의 수에 거의 제한이 없게 되는 경우에는 사용하기 어려운데, 파티 시스템에서의 파티 생성/파괴, 파티원 초대/추방 등의 처리는 모두 이산적이어서 이를 이용한 검증에 적합하다.

4. SPIN을 이용한 검증

4.1 시스템 모델링

온라인 게임은 게임 서버 하나에 다수의 사용자가 접속하는 형태로 프로세스를 모델링할 수 있다. 또한 다수의 사용자는 서버로 동시에 메시지를 보내지만 서버에서는 하나의 메시지 큐(message queue)를 사용하므로 사용자와 서버 간 통신은 하나의 채널로 표현했다. Fig. 6은 본 시스템에 대한 개요를 나타낸다.

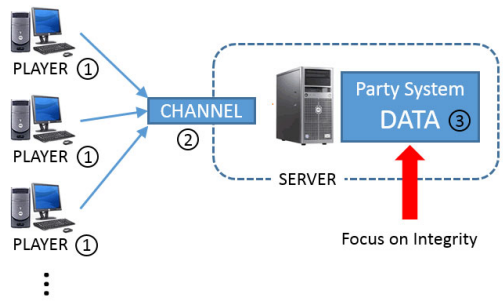

Fig. 6. Online-Game Modeling

각 사용자가 서버로 보내는 메시지(mtype)는 Table 2와 같으며 각 사용자들은 이 메시지를 공통의 채널을 통해 서버로 보내게 된다.

Table 2. Message Types Sent to a Server

Message Name	Meaning
INVITE	파티원 초대
ACCEPT	초대 승낙
REFUSE	초대 거절
CANCEL	초대 취소
PROMOTE	파티장 위임
KICK	파티원 추방
LEAVE	파티 이탈

메시지, 채널, 사용자 및 파티를 모델링한 PROMELA 코드는 다음 Table 3과 같으며 Fig. 6과 대응하는 요소들은 원문자(①, ②, ③)로 표시되어있다.

Table 3. PROMELA Snippet - Message, Channel, Player & Party

```
mtype = {  INVITE, ACCEPT, REFUSE, CANCEL,
           PROMOTE, KICK, LEAVE };
chan g_toServer = [1] of { mtype, byte, byte }; // ②

typedef Player {
  // 내가 현재 속한 파티, g_parties의 인덱스이며
  // NUM_PARTIES이면 파티에 속해있지 않음을
  // 나타낸다.
  byte partyID;
  // 현재 내가 보낸 초대.
  // 참이면 해당 사용자에게 초대를 보낸 상태이다.
  bool invitations[NUM_PLAYERS];

  // 내가 현재 보낸 전체 초대 개수
  // 동시에 일정 개수 이상의 초대는 보낼 수 없다.
  byte invitationSent;

  // 내가 받은 초대의 합
  // 동시에 일정 개수 이상의 초대는 받을 수 없다.
  byte invitationReceived;
};
Player g_players[NUM_PLAYERS]; // ①

typedef Party {
  // 파티원의 수, 0이면 생성되지 않은 파티이다.
  byte numMembers;
  // 이 파티의 파티장, g_players의 인덱스이다.
  byte leaderID;
};
Party g_parties[NUM_PARTIES]; // ③
```

4.2 게임 사용자

게임 사용자는 악의적 사용자를 가정한다. 악의적 사용이란 정해진 프로토콜과 다르게 메시지를 보내는 것으로 해킹 시도에 해당한다. 파티 초대를 받지 않았는데도 초대 승낙 메시지를 보내는 경우가 한 예이다.

게임 사용자를 나타내는 모델은 Table 4와 같으며 해당 프로세스는 모든 메시지를 임의로 보내도록 모델링되었다.

Table 4. PROMELA Snippet - Player Process

```
proctype ProcPlayer(byte playerID) {
  do
  :: g_toServer ! INVITE, playerID, 0;
  :: g_toServer ! INVITE, playerID, 1;
  :: g_toServer ! INVITE, playerID, 2;
  :: g_toServer ! INVITE, playerID, 3;
  :: g_toServer ! ACCEPT, playerID, 0;
  :: g_toServer ! ACCEPT, playerID, 1;
  ...
  :: g_toServer ! KICK, playerID, 2;
  :: g_toServer ! KICK, playerID, 3;
  :: g_toServer ! LEAVE, playerID, 0;
  od
}
```

4.3 게임 서버

서버는 메시지 큐에서 메시지를 하나씩 꺼내 해당 메시지 종류에 맞는 처리를 수행한다. 메시지는 메시지 종류에 해당하는 핸들러에서 처리된다.

Table 5. PROMELA snippet - Server Process

```
proctype ProcServer() {
  …
  do
  :: g_toServer ? INVITE, playerID, targetID ->
        d_step { OnInvite(playerID, targetID); }
  :: g_toServer ? ACCEPT, playerID, targetID ->
        d_step { OnAccept(playerID, targetID); }
  :: g_toServer ? REFUSE, playerID, targetID ->
        d_step { OnRefuse(playerID, targetID); }
  :: g_toServer ? CANCEL, playerID, targetID ->
        d_step { OnCancel(playerID, targetID); }
  :: g_toServer ? PROMOTE, playerID, targetID ->
        d_step { OnPromote(playerID, targetID); }
  :: g_toServer ? KICK, playerID, targetID ->
        d_step { OnKick(playerID, targetID); }
  :: g_toServer ? LEAVE, playerID, targetID ->
        d_step { OnLeave(playerID); }
  od
}
```

아래 Table 6은 메시지 핸들러 중 하나인 OnInvite()를 나타낸다.

Table 6. PROMELA Snippet - Server Message Handler OnInvite()

```
inline OnInvite(playerID, targetID) {
  if
  :: ((g_parties[0].numMembers > 0) &&
      (g_parties[1].numMembers > 0))
     -> skip; // 추가 생성 가능한 파티가 없다.
  :: (playerID == targetID)
     -> printf("invalid call\n"); // 나 자신을 초대하고 있다.
  :: (g_players[targetID].partyID < NUM_PARTIES)
     -> skip; // 대상이 다른 파티의 파티원이다.
  ::(g_players[targetID].invitationReceived
     >=MAX_INVITATION_RECEIVE)
     -> skip; // 대상이 이미 최대 개수의 초대를 받았다.
  :: (g_players[playerID].invitationSent
     >= MAX_INVITATION_SEND)
     -> skip; // 내가 이미 최대 개수의 초대를 보냈다.
  :: (g_players[playerID].invitations[targetID])
     -> printf("invalid call\n"); // 중복초대
  :: ((g_players[playerID].partyID < NUM_PARTIES) &&
     (g_parties[g_players[playerID].partyID].numMembers
     >= MAX_PARTY_MEMBERS))
     -> skip; // 이미 최대 파티원
  :: ((g_players[playerID].partyID < NUM_PARTIES) &&
     (g_parties[g_players[playerID].partyID].leaderID
     != playerID))
     -> printf("invalid call\n"); // not a leader
  :: ((g_players[playerID].partyID < NUM_PARTIES) &&
     (g_players[playerID].invitationSent +
     g_parties[g_players[playerID].partyID].numMembers
     >= MAX_PARTY_MEMBERS))
     -> skip; // 이미 최대 개수의 초대를 보냈다.
  :: else
     -> // 초대 작업
     g_players[playerID].invitations[targetID] = true;
     g_players[playerID].invitationSent++;
     g_players[targetID].invitationReceived++;
  fi
}
```

Table 6의 "**:: else**" 앞에 나타나는 부분들은 해당 메시지의 요청이 유효한 요청인지 검증하는 부분으로, 이는 실제 게임서버의 검증 절차와 동일하다.

사용자 및 서버 프로세스를 생성하는 초기화 모델은 아래 Table 7과 같으며, 해당 모델은 다수의 사용자 프로세스와 하나의 서버 프로세스를 생성한다.

Table 7. PROMELA Snippet - Initial Process

```
init {
  byte playerIndex;
  for (playerIndex : 0 .. (NUM_PLAYERS-1)) {
    // partyID가 NUM_PARTIES이면 현재 파티에
    // 속해있지 않음을 나타낸다.
    g_players[playerIndex].partyID = NUM_PARTIES;
  }
  for (playerIndex : 0 .. (NUM_PLAYERS-1)) {
    run ProcPlayer(playerIndex);
  }
  run ProcServer();
}
```

4.4 검증 특성

본 논문에서는 서버 파티 시스템 정보의 완전성(Integrity) 검증에 초점을 맞춘다. 검증할 특성을 표현한 LTL식들은 아래 Table 8과 같으며 지면관계상 유사한 식들은 제외하였다.

Table 8. PROMELA Snippet - LTL Formulae

```
// 모든 파티는 최대 파티원 수를 초과할 수 없다.
ltl p0 { [](g_pa rties[0].numMembers
          <= MAX_PARTY_MEMBERS) &&
          (g_parties[1].numMembers
          <= MAX_PARTY_MEMBERS)) }

// 1인 파티는 존재하지 않는다.
ltl p1 { [](g_parties[0].numMembers != 1) &&
          (g_parties[1].numMembers != 1)) }
// 파티가 처음 생성되면 항상 파티원이 2명이다.
// (초대한 사람 + 승낙한 사람)
ltl p2 { [](g_parties[0].numMembers == 0)
        -> ((g_parties[0].numMembers == 0) W
            (g_parties[0].numMembers == 2))) }

// 파티장은 항상 해당 파티의 파티원이어야 한다.
ltl p3 { [](g_parties[0].numMembers > 0)
       -> ((g_players[g_parties[0].leaderID].partyID) == 0)) }

// Party.numMembers와 실제 파티원 수는 반드시 일치
ltl p4 { [](g_parties[1].numMembers == 0)
       -> ((g_players[0].partyID != 1) &&
           (g_players[1].partyID != 1) &&
           (g_players[2].partyID != 1) &&
           (g_players[3].partyID != 1))) }

// 파티장이 아닌 파티원은 초대를 보낼 수 없다.
ltl p5 { [](((g_players[0].partyID < NUM_PARTIES) &&
         (g_parties[g_players[0].partyID].leaderID != 0))
      -> ((g_players[0].invitationSent == 0) &&
          (!g_players[0].invitations[0]) &&
          (!g_players[0].invitations[1]) &&
          (!g_players[0].invitations[2]) &&
          (!g_players[0].invitations[3])))}

// 파티원은 초대를 받을 수 없다.
ltl p6 { [](((g_players[0].partyID < NUM_PARTIES)
      -> ((g_players[0].invitationReceived == 0) &&
          (!g_players[0].invitations[0]) &&
          (!g_players[1].invitations[0]) &&
          (!g_players[2].invitations[0]) &&
          (!g_players[3].invitations[0])))}
```

5. 결 과

검증을 수행한 환경은 다음과 같다.
OS : MS Windows7 64-bit
CPU : Intel i7-3770 @ 3.40GHz,
Memory : 12GB
Spin Version : 6.3.2

또한 메모리 사용량을 줄이기 위해 해시 압축(hash compaction) 옵션을 사용하였다. 아래 Table 9는 검증에 사용된 설정과 각각의 특성을 검증하기 위해 소요된 자원 및 그 결과를 나타낸다.

주요 검증 설정값인 전체 사용자 수, 전체 파티 수 및 최대 파티원 수는 메모리제약 때문에 파티 시스템 제어 중 발생 가능한 경우를 모두 포함하는 범위 내에서 최소화하였다.

Table 9. Verification Settings & Results for LTL Formulae

```
# of players : 4
# of parties : 2
# of max party members : 3
# of max concurrent invitations to send : 2 (per player)
# of max concurrent invitations to receive : 2 (per player)
```

LTL formula	Memory (MB)	Time (sec)	Error
p0	4750.291	307	No
p1	4750.291	302	No
p2	7040.037	682	No
p3	4750.291	291	No
p4	4750.291	295	No
p5	4750.291	296	No
p6	399.424	0.373	**Yes**

검증 결과 LTL 식 p6의 검증이 실패했는데, 이는 파티 생성 시 파티원들의 초대 상태 초기화의 문제로 인한 것이었다. 예를 들면, 초대한 사용자 U0와 승낙한 사용자 U1으로 새로운 파티를 만드는 순간, 다른 사용자 U2가 U0와 U1에게 동시에 초대를 보낸 상태였다고 하면, 파티 생성 과정에서 U2가 U1에게 보낸 초대는 정상적으로 초기화(제거)하는 데 비해 U2가 U0에게 보낸 초대는 초기화하지 않고 있었다. 반례의 분석 결과 현재 파티 시스템에 치명적인 문제를 야기하는 오류는 아니었다. 그러나 초기화 오류는 시스템의 안전성과 신뢰도를 저해하는 주요 원인 중의 하나로 알려져 있다.

또한 검증 결과 최대 12분의 시간과 7GB의 메모리가 소요되었으며, 이는 최근 일반적인 장비로도 충분히 소화할 수 있는 비용이다. 이와 같이 납득할만한 비용 내에서 모델 체킹을 이용한 검증을 수행할 수 있다는 것을 확인할 수 있었다.

p2의 경우에 다른 LTL식들에 비해 많은 자원이 필요한 것을 알 수 있는데 이는 LTL식에 포함된 시제연산자 수의

차이로 인한 것이다. LTL식 검증의 시간 복잡도는 |φ|가 LTL식의 길이(시제연산자 수), |S|가 상태의 개수일 때 $O(|S|\cdot 2^{|\phi|})$이며[12], 다른 LTL식들이 [] 하나의 시제연산자만을 포함하고 있는 것에 비하여 p2는 [] 와 W, 두 개의 시제연산자를 포함하고 있다.

Table 10은 다른 설정은 동일하게 하고 전체 사용자 수만 조정했을 때의 메모리 사용량과 처리 시간을 보여주는데 전체 상태 증가에 따른 소요자원의 지수적 증가를 쉽게 확인할 수 있다.

Table 10. Results for LTL Formula p0 (by Varying # of Players)

# of Players	Memory (MB) Total	Memory (MB) for States	Time (sec)
1	395.224	0.281	0.005
2	395.420	0.467	0.014
3	416.513	21.263	0.977
4	4750.291	3921.220	307
5	Out of Memory	-	-

6. 결론 및 향후 연구

본 논문에서는 게임서버의 파티시스템에 SPIN 모델 체커를 이용한 자동화 검증 시스템을 적용하였다. 검증 대상이 된 파티 시스템이 어느 정도 정형화된 상태이고 내부 테스팅이 진행된 후라 심각한 오류는 발견되지 않았다. 하지만 테스팅으로 발견하지 못한 오류를 발견할 수 있었고 검증 비용도 현재의 데스크톱 컴퓨터로 무리가 없는 수준이었다.

다만 모델링 과정에서 세심한 주의가 필요하고 Table 10에서 볼 수 있듯 허용하는 자원에 맞는 시스템 모델링의 규모를 정할 때도 많은 시행착오를 거치게 된다. 아직까지 그다지 사용자 친화적이지 않은 SPIN 작업환경도 많은 개선의 여지가 있다. 그럼에도 불구하고 일반적인 소프트웨어 프로젝트에서 테스팅이 차지하는 시간과 비용이 대략 50%라는[13] 사실을 고려해본다면 모든 상태를 검사할 수 있고 자동화되어있다는 점에서 중요도가 크면서 복잡도가 높지 않은 시스템의 추가적인 검증 방법으로 고려할만하다.

그 예로 게임 아이템 거래를 들 수 있는데 아이템 판매는 온라인 게임 매출의 절반 이상(2013년 기준 54.7%)[14]을 차지할 정도로 중요하지만, 관련된 오류는 끊임없이 나타나고 있으며[15, 16] 이로 인해 정상 사용자를 제재하는 경우까지 발생하였다[17]. 이러한 오류들은 대부분 초기 작업 및 업데이트 과정에서 발생한 일반적이지 않은 오류 발생 경로를 기존 검증 과정에서 검출하지 못할 때 발생하므로 모델 체킹 방법을 사용하면 오류 발생 빈도를 상당히 낮출 수 있다. 또한 개인정보 보호와도 밀접한 관계가 있는 사용자 인증 처리도 시스템의 복잡도에 비해 그 중요성이 훨씬 크므로 충분히 적용할 가치가 있을 것으로 판단한다.

References

[1] Christel Baier, et al., “Principles of Model Checking,” pp.1-2, The MIT Press, 2008.

[2] Nancy Leveson, “Safeware: System Safety and Computers,” Addison-Wesley, 1995.

[3] Christel Baier, et al., “Principles of Model Checking,” pp.3-6, The MIT Press, 2008.

[4] Edmund M. Clarke, et al., “Model checking,” The MIT press, 1999.

[5] Gerard Holzmann, “The model checker SPIN,” *IEEE Transactions on Software Engineering*, Vol.23, No.5, 1997.

[6] Model checker SPIN [Internet], http://spinroot.com.

[7] Francis Schneider, et al., “Validating Requirements for Fault Tolerant Systems using Model Checking,” *Third IEEE Conference on Requirements Engineering*, 1998.

[8] Klaus Havelund, et al., “Formal Analysis of the Remote Agent Before and After Flight,” *Proceedings of the 5th NASA Langley Formal Methods Workshop*, 2000.

[9] Yunja Choi, “Model checking Trampoline OS: a case study on safety analysis for automotive software,” *Softw. Test., Verif. Reliab.*, pp.38-60, 2014.

[10] Theo Ruys, et al., “Advanced SPIN Tutorial,” *Proceedings of the 11th SPIN Workshop*, 2004.

[11] Gerard Holzmann, “Software Model Checking,” *NATO Summer School*, Vol.180, pp.309-355, 2000.

[12] Orna Lichtenstein, et al., “Checking That Finite State Concurrent Programs Satisfy Their Linear Specification,” *Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pp.97-107, 1985.

[13] Glenford J. Myers, “*The Art of Software Testing, 2nd*,” Wiley, 2004.

[14] KOCCA(Korea Creative Content Agency), “2013 White Paper on Korean Games,” 2013.

[15] ThisIsGame, “메이플스토리, 아이템 복사 버그 발생…복제 아이템 전량 회수 조치,” [Internet], http://www.inven.co.kr/webzine/news/?news=132336, 2015.

[16] ThisIsGame, “아이템 복사하면 영업방해죄 적용? 최강의 군단, 버그 악용자에 강경대응,” [Internet], http://www.thisisgame.com/webzine/news/nboard/4/?n=56389, 2014.

[17] ThisIsGame, “‘죄 없는 유저 영구정지’ 검은사막, 복사 버그 제재 사고,” [Internet], http://www.thisisgame.com/webzine/news/nboard/4/?n=57804, 2015.

김 광 훈

e-mail : hun172@outlook.com
2015년 경북대학교 컴퓨터학부(석사)
현 재 서울대학교 전기정보공학부 박사과정
관심분야 : 실시간 렌더링, 게임 프로그래밍

최 윤 자

e-mail : yuchoi76@knu.ac.kr
2003년 미네소타대학 전산과(박사)
2003년~2006년 프라운호퍼연구소 연구원
현 재 경북대학교 컴퓨터학부 부교수
관심분야 : 소프트웨어 안전성 분석, 정형검증, 모델기반 개발방법론