分散決済システムの形式検証

中村 剛

学生番号:s2130012

概要

この研究副テーマでは、分散システムにおけるトランザクション処理をテーマにSpin/Promelaを使って各システムの性質をモデル化し、assertによる検証とLTLによる検証を行う。

目次

第1章 はじめに

1.1 背景

ECの普及によりオンラインサイトでのクレジットカード決済は広く使われている。またサービス事業者はロイヤルカスタマーになってもらうべく、独自のポイントを消費者に付与し、決済額へ充当できる仕組みを持っている事が多い。このクレジットカード決済を行うシステムとポイント管理を行うシステムは別々に独立したシステムである事が多く、分散トランザクション処理を必要とする。

分散トランザクションとしてはXA [1]といった分散トランザクション処理のための標準規格やTwo-phase commit [2]による実現方法などが既に存在するが、性能の面で良くない点が課題であり、実用的ではない。そこで各事業者は独自にトランザクション処理をサポートするシステムを開発している。

決済に関わる重要なシステムゆえに各社テストなどは十分に行っているが、様々なケースが考えられる為、全てのケースにおいて正しく制御できることを保証できない場合がある。

1.2 目的

それぞれ独立したシステム間でトランザクション管理をする為、一時的なネットワーク不通問題、故障による障害、異常系エラーなどが想定される。こういった障害系が発生してもリトライによる二重引き落としやロールバック機能が正しく機能する必要がある。
本研究課題ではモデル検査のSpin/Promelaを用いて分散決済システムのトランザクション処理に関してシステムがとりえる「全ての状態」と「全ての実行パス」を網羅的に検証し、それぞれのシステムが満たすべき性能を正しく定義する事である。

第2章 準備

2.1 Spin/Promela

Spin/Promela [3] について概説する。Spinは並行ソフトウェアモデルの正確さを検証するためのモデル検査ツールでPromela (Process Meta Language)を用いて記述する。
Promelaでシステムの振る舞いを記述し、C言語のコード生成、コンパイル、実行する事で検証が行える。
システムの性質をLTL式（線形時相論理式）で記述してモデル検査を行う事ができる。またPromelaのコードの特定の位置で特定の条件を満たすかどうかを検証する為のassert文の検査を行う事ができる。

2.2 進行性解析

Promelaの中で非決定性構文によるモデリングが可能である。非決定性構文を使う事で「時々通信が遮断される」といったような表現が可能になる。しかし、意図とは反して特定のコードのみを無限に繰り返し実行してしまい、仮定の公平性が保たれない場合が存在する。飢餓状態を回避する為、progress [4] ラベルを導入し、その特定のコードがいつかは実行されるようにする。

2.3 検証対象のシステムシーケンス図

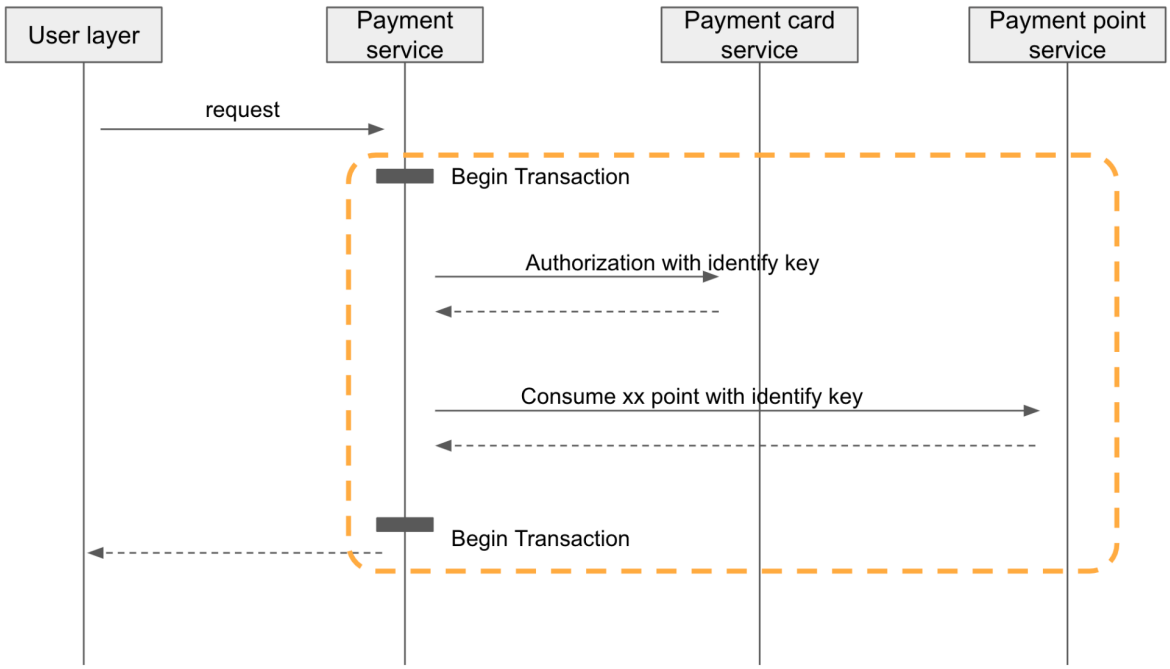

図2.3

図2.3は検証対象の分散システムのシーケンス図である。Promelaで振る舞いを記述する部分は点線で囲まれた箇所になる。それぞれ「Payment service」、「Payment card service」、「Payment point service」を一つのプロセスに見立てる。それぞれのプロセスに通信するネットワーク部分を新たなプロセスで表現し、それを通して通信する。通信はPromelaのChannel [5] を使い且つ非同期通信で行う。

2.4 トランザクション処理のフローチャート図

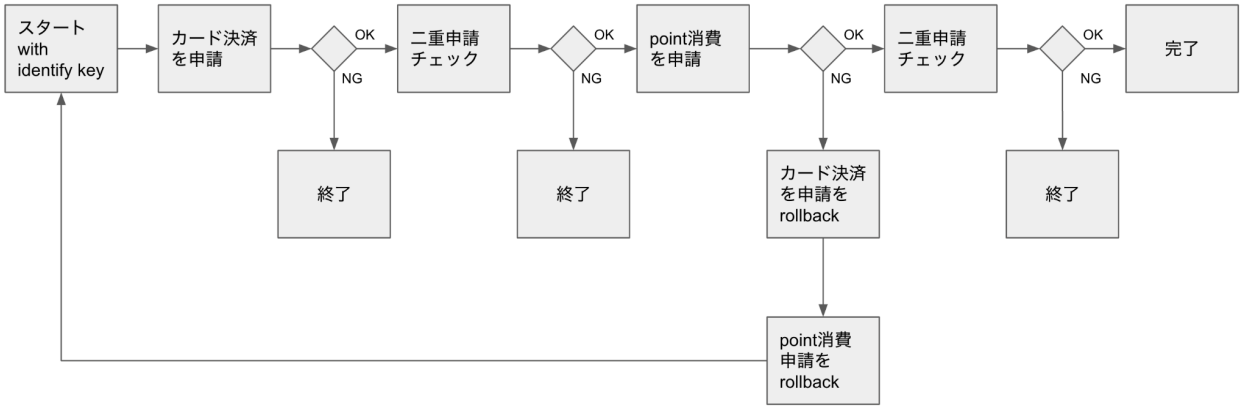

図2.4

図2.4は検査対象になっているシステムのトランザクション処理をフローチャート図にしたものである。カード決済が正常に行われるとポイント決済に進み、エラーとなった場合はロールバックの処理が行われる。カード決済がNGや既に申請済みの場合は即座に終了となる。

2.5 Assert

Assert構文 [6] はPromelaのどの部分でも使う事が可能であり、実行中のPromelaの変数の制御状態に影響を与えません。assertが実行されるたびに評価される。assertの中の式がfalse（または同等に整数値0）と評価された場合、assertion違反が報告される。

2.6 LTL

SpinではLTL式と呼ばれる線形時相論理式を扱うことができる。線形なので時間の流れが一直線でしか扱うことができず、途中枝分かれしてパラレルに表現することはできない。∧, ∨, → ¬ などの論理結合子とともに，G（いつでも），F（いつか）などの様相結合子を用いて論理式を記述する。Promelaでは下記の形式で記載する。

```
ltl [名前] {LTL 論理式 }
```

PromelaではG（いつでも）は []、F（いつか）は<>で表す。
他にもBinary Operatorsも利用できる。詳しくは [7] を参照して欲しい。

本研究ではネットワーク不通や故障時を想定した場合の到達性の検証に活用する。

第3章 検証

3.1 公平性の検査

図2.3から「Payment service」、「Payment card service」、「Payment point service」を一つのプロセスとしてPromelaに記載する。通信は「Payment service」と「Payment card service」の間に「Newwork for card」プロセス、「Payment service」と「Payment point service」の間に「Network for point」プロセスを作成し、ここで時々通信が遮断される事を表現する。このNetworkプロセスを必ず通して通信するようにする。下記の3.1の図は設計した通信を表したものになる。

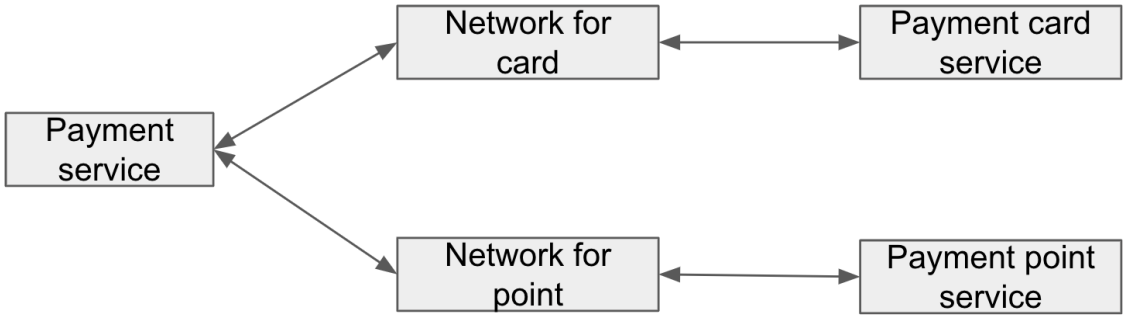

図3.1

「Network for card」「Network for point」のプロセスの中で非決定性構文により時々、通信が遮断されるようにメッセージ通信を消失させている。ここで特定のコードのみを無限に繰り返し実行しないよう第二章の2.2項目でのべたprogressラベルを付与し、公平性の検査を行った。

検査を実施した結果は以下の通りである。

```
➜  sub-theme git:(master) spin -a payment-distribution-system3.pml
ltl p1: [] ((! ((payment_service@request))) || (<> ((payment_service@response))))
ltl p2: (! ((payment_service@request))) || ([] (! ((payment_service@response))))
ltl p3: [] ((! ((((payment_service@request)) && (<> ((network_from_payment_to_card@forwa
rded1)))) && (<> ((network_from_payment_to_card@forwarded2))))) || (<> ((payment_service
@response))))
  the model contains 3 never claims: p3, p2, p1
  only one claim is used in a verification run
  choose which one with ./pan -a -N name (defaults to -N p1)
  or use e.g.: spin -search -ltl p1 payment-distribution-system3.pml
```

図3.2
図3.2はCコードを生成した時のキャプチャである。

```
➜  sub-theme git:(master) gcc -DNP pan.c -o pan
```

図3.3
図3.3はコンパイルしたときのキャプチャである

```
➜  sub-theme git:(master) ./pan -l

(Spin Version 6.5.2 -- 6 December 2019)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (:np_:)
        assertion violations    + (if within scope of claim)
        non-progress cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 148 byte, depth reached 683, errors: 0
     6389 states, stored
     1013 states, matched
     7402 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)
```

図3.4

図3.4は進行性解析を実施した時のキャプチャである。errors:0と確認できる。

また、progressラベルを外して検査した結果が下記である。

```
  <<<<<START OF CYCLE>>>>>
174:    proc  3 (payment_point_service:1) payment-distribution-system3.pml:235 (state 51)       [cable2_r_out!ack,tran,r]
176:    proc  1 (network_from_payment_to_point:1) payment-distribution-system3.pml:136 (state 4)        [cable2_r_out?m,t,r]
178:    proc  1 (network_from_payment_to_point:1) payment-distribution-system3.pml:136 (state 5)        [cable2_s_in!m,t,r]
180:    proc  5 (payment_service:1) payment-distribution-system3.pml:67 (state 34)      [cable2_s_in?ack,tran,r]
182:    proc  5 (payment_service:1) payment-distribution-system3.pml:69 (state 35)      [((r==1))]
                        point tran:2, done:1
184:    proc  5 (payment_service:1) payment-distribution-system3.pml:70 (state 36)      [printf('point tran:%d, done:%d\\n',tran,r)]
186:    proc  5 (payment_service:1) payment-distribution-system3.pml:74 (state 41)      [((tran==2))]
188:    proc  5 (payment_service:1) payment-distribution-system3.pml:74 (state 42)      [assert((ct2==pt2))]
190:    proc  5 (payment_service:1) payment-distribution-system3.pml:33 (state 47)      [else]
192:    proc  5 (payment_service:1) payment-distribution-system3.pml:33 (state 48)      [tran = 0]
194:    proc  5 (payment_service:1) payment-distribution-system3.pml:39 (state 1)       [cable1_s_out!msg,tran,1]
196:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:121 (state 1) [cable1_s_out?m,t,type]
198:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:121 (state 2) [cable1_r_in!m,t,type]
200:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:146 (state 1)[cable1_r_in?msg,t,type]
202:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:156 (state 12)[((((t==0)&&(ct0!=0))&&(type==1)))]
204:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:156 (state 13)[r = ct0]
206:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:186 (state 49)[cable1_r_out!ack,tran,r]
208:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:123 (state 4) [cable1_r_out?m,t,r]
210:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:123 (state 5) [cable1_s_in!m,t,r]
212:    proc  5 (payment_service:1) payment-distribution-system3.pml:42 (state 2)       [cable1_s_in?ack,tran,r]
214:    proc  5 (payment_service:1) payment-distribution-system3.pml:45 (state 3)       [((r==1))]
                        card tran:0, done:1
216:    proc  5 (payment_service:1) payment-distribution-system3.pml:46 (state 4)       [printf('card tran:%d, done:%d\\n',tran,r)]
                        doing next
218:    proc  5 (payment_service:1) payment-distribution-system3.pml:64 (state 32)      [printf('doing next \\n')]
220:    proc  5 (payment_service:1) payment-distribution-system3.pml:65 (state 33)      [cable2_s_out!msg,tran,1]
222:    proc  1 (network_from_payment_to_point:1) payment-distribution-system3.pml:134 (state 1)        [cable2_s_out?m,t,type]
224:    proc  1 (network_from_payment_to_point:1) payment-distribution-system3.pml:134 (state 2)        [cable2_r_in!m,t,type]
226:    proc  3 (payment_point_service:1) payment-distribution-system3.pml:196 (state 1)[cable2_r_in?msg,t,type]
228:    proc  3 (payment_point_service:1) payment-distribution-system3.pml:206 (state 14)       [(((((t==0)&&(pt0!=0))&&(type==1)))]
230:    proc  3 (payment_point_service:1) payment-distribution-system3.pml:206 (state 15)       [r = pt0]
232:    proc  3 (payment_point_service:1) payment-distribution-system3.pml:235 (state 51)       [cable2_r_out!ack,tran,r]
234:    proc  1 (network_from_payment_to_point:1) payment-distribution-system3.pml:136 (state 4)        [cable2_r_out?m,t,r]
236:    proc  1 (network_from_payment_to_point:1) payment-distribution-system3.pml:136 (state 5)        [cable2_s_in!m,t,r]
238:    proc  5 (payment_service:1) payment-distribution-system3.pml:67 (state 34)      [cable2_s_in?ack,tran,r]
240:    proc  5 (payment_service:1) payment-distribution-system3.pml:69 (state 35)      [((r==1))]
                        point tran:0, done:1
242:    proc  5 (payment_service:1) payment-distribution-system3.pml:70 (state 36)      [printf('point tran:%d, done:%d\\n',tran,r)]
244:    proc  5 (payment_service:1) payment-distribution-system3.pml:72 (state 37)      [((tran==0))]
246:    proc  5 (payment_service:1) payment-distribution-system3.pml:72 (state 38)      [assert((ct0==pt0))]
248:    proc  5 (payment_service:1) payment-distribution-system3.pml:32 (state 45)      [((tran<2))]
250:    proc  5 (payment_service:1) payment-distribution-system3.pml:32 (state 46)      [tran = (tran+1)]
252:    proc  5 (payment_service:1) payment-distribution-system3.pml:39 (state 1)       [cable1_s_out!msg,tran,1]
254:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:121 (state 1) [cable1_s_out?m,t,type]
256:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:121 (state 2) [cable1_r_in!m,t,type]
258:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:146 (state 1)[cable1_r_in?msg,t,type]
260:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:168 (state 27)[((((t==1)&&(ct1!=0))&&(type==1)))]
262:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:168 (state 28)[r = ct1]
264:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:186 (state 49)[cable1_r_out!ack,tran,r]
266:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:123 (state 4) [cable1_r_out?m,t,r]
268:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:123 (state 5) [cable1_s_in!m,t,r]
270:    proc  5 (payment_service:1) payment-distribution-system3.pml:42 (state 2)       [cable1_s_in?ack,tran,r]
272:    proc  5 (payment_service:1) payment-distribution-system3.pml:45 (state 3)       [((r==1))]
                        card tran:1, done:1
274:    proc  5 (payment_service:1) payment-distribution-system3.pml:46 (state 4)       [printf('card tran:%d, done:%d\\n',tran,r)]
                        doing next
276:    proc  5 (payment_service:1) payment-distribution-system3.pml:64 (state 32)      [printf('doing next \\n')]
278:    proc  5 (payment_service:1) payment-distribution-system3.pml:65 (state 33)      [cable2_s_out!msg,tran,1]
280:    proc  1 (network_from_payment_to_point:1) payment-distribution-system3.pml:134 (state 1)        [cable2_s_out?m,t,type]
282:    proc  1 (network_from_payment_to_point:1) payment-distribution-system3.pml:134 (state 2)        [cable2_r_in!m,t,type]
284:    proc  3 (payment_point_service:1) payment-distribution-system3.pml:196 (state 1)[cable2_r_in?msg,t,type]
286:    proc  3 (payment_point_service:1) payment-distribution-system3.pml:218 (state 29)       [(((((t==1)&&(pt1!=0))&&(type==1)))]
288:    proc  3 (payment_point_service:1) payment-distribution-system3.pml:218 (state 30)       [r = pt1]
290:    proc  3 (payment_point_service:1) payment-distribution-system3.pml:235 (state 51)       [cable2_r_out!ack,tran,r]
292:    proc  1 (network_from_payment_to_point:1) payment-distribution-system3.pml:136 (state 4)        [cable2_r_out?m,t,r]
294:    proc  1 (network_from_payment_to_point:1) payment-distribution-system3.pml:136 (state 5)        [cable2_s_in!m,t,r]
296:    proc  5 (payment_service:1) payment-distribution-system3.pml:67 (state 34)      [cable2_s_in?ack,tran,r]
298:    proc  5 (payment_service:1) payment-distribution-system3.pml:69 (state 35)      [((r==1))]
                        point tran:1, done:1
300:    proc  5 (payment_service:1) payment-distribution-system3.pml:70 (state 36)      [printf('point tran:%d, done:%d\\n',tran,r)]
302:    proc  5 (payment_service:1) payment-distribution-system3.pml:73 (state 39)      [((tran==1))]
304:    proc  5 (payment_service:1) payment-distribution-system3.pml:73 (state 40)      [assert((ct1==pt1))]
306:    proc  5 (payment_service:1) payment-distribution-system3.pml:32 (state 45)      [((tran<2))]
308:    proc  5 (payment_service:1) payment-distribution-system3.pml:32 (state 46)      [tran = (tran+1)]
310:    proc  5 (payment_service:1) payment-distribution-system3.pml:39 (state 1)       [cable1_s_out!msg,tran,1]
312:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:121 (state 1) [cable1_s_out?m,t,type]
314:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:121 (state 2) [cable1_r_in!m,t,type]
316:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:146 (state 1)[cable1_r_in?msg,t,type]
318:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:180 (state 42)[(((((t==2)&&(ct2!=0))&&(type==1)))]
320:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:180 (state 43)[r = ct2]
322:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:186 (state 49)[cable1_r_out!ack,tran,r]
324:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:123 (state 4) [cable1_r_out?m,t,r]
326:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:123 (state 5) [cable1_s_in!m,t,r]
328:    proc  5 (payment_service:1) payment-distribution-system3.pml:42 (state 2)       [cable1_s_in?ack,tran,r]
330:    proc  5 (payment_service:1) payment-distribution-system3.pml:45 (state 3)       [((r==1))]
                        card tran:2, done:1
332:    proc  5 (payment_service:1) payment-distribution-system3.pml:46 (state 4)       [printf('card tran:%d, done:%d\\n',tran,r)]
                        doing next
334:    proc  5 (payment_service:1) payment-distribution-system3.pml:64 (state 32)      [printf('doing next \\n')]
336:    proc  5 (payment_service:1) payment-distribution-system3.pml:65 (state 33)      [cable2_s_out!msg,tran,1]
338:    proc  1 (network_from_payment_to_point:1) payment-distribution-system3.pml:134 (state 1)        [cable2_s_out?m,t,type]
340:    proc  1 (network_from_payment_to_point:1) payment-distribution-system3.pml:134 (state 2)        [cable2_r_in!m,t,type]
342:    proc  3 (payment_point_service:1) payment-distribution-system3.pml:196 (state 1)[cable2_r_in?msg,t,type]
344:    proc  3 (payment_point_service:1) payment-distribution-system3.pml:230 (state 44)       [((((t==2)&&(pt2!=0))&&(type==1)))]
346:    proc  3 (payment_point_service:1) payment-distribution-system3.pml:230 (state 45)       [r = pt2]
spin: trail ends after 346 steps
#processes: 6
                ct0 = 1
                ct1 = 1
                ct2 = 1
                pt0 = 1
                pt1 = 1
                pt2 = 1
                tran = 2
                type = 1
                t0_balance = 7
                queue 1 (cable1_s_out):
                queue 4 (cable1_s_in):
                queue 2 (cable1_r_in):
                queue 3 (cable1_r_out):
                queue 5 (cable2_s_out):
                queue 8 (cable2_s_in):
                queue 6 (cable2_r_in):
                queue 7 (cable2_r_out):
                request = 0
                response = 0
                forwarded1 = 0
                forwarded2 = 0
346:    proc  5 (payment_service:1) payment-distribution-system3.pml:66 (state 61)
346:    proc  4 (:init::1) payment-distribution-system3.pml:241 (state 2) <valid end state>
346:    proc  3 (payment_point_service:1) payment-distribution-system3.pml:235 (state 51)
346:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:146 (state 1)
346:    proc  1 (network_from_payment_to_point:1) payment-distribution-system3.pml:133 (state 7)
346:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:120 (state 7)
6 processes created
```

図3.5

「START　OF　CYCLE」と記された以降のログを確認すると、ネットワークプロセスにおいてデータロストするコードが一度も実行されていない事が確認できる。よってこの公平性の検証は妥当である。

3.2 assertによる検査

図2.4からassertによる検証は下記の図の赤点線の箇所で検査を行った。

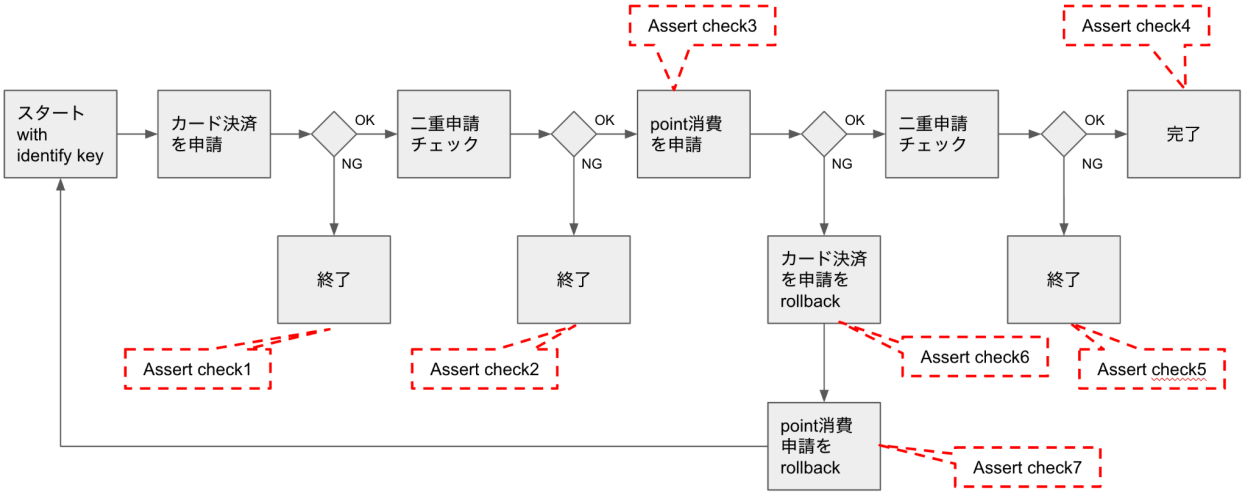

図3.2

全ての処理にトランザクションID(identify　key)を持ってアクセスが行われる。またカード決済、point決済の状態管理として下記のように番号を定義している。

決済の状態管理一覧
0：何も処理が行われていない初期状態。またはrollbackされた状態
1：決済が正常に完了した状態。
2：決済が失敗した状態。

図3.2の赤線の検査内容を下記に説明する。

1. Assert Check1
カード決済が失敗した為、カード決済の状態は2になり、point決済の状態は0である。

2. Assert Check2
既にカード決済が行われた後の重複のリクエストなのでカード決済の状態は1または2になる。
point決済の状態は全ての状態がありえる為、0または1または2になる。
1度目のpoint決済時はネットワーク不通で終わる場合、正常に行われる場合、失敗になる場合が考えられる為である。

3. Assert Check3
カード決済は成功している為、カード決済の状態は1になり、point決済はこれからになるので
point決済の状態は0になる。

4. Assert Check4
カード決済は成功している為、カード決済の状態は1になり、point決済も成功している為、
point決済の状態は1になる。

5. Assert Check5
カード決済は成功している為、カード決済の状態は1になり、point決済は重複のリクエストにな
る為、point決済の状態は1または2になる。

6. Assert Check6
カード決済はrollbackされた為、カード決済の状態は0になり、point決済の状態は失敗のまま
なので2になる。

7. Assert Check7
カード決済はrollbackされた為、カード決済の状態は0になり、point決済もrollbackされた為、
point決済の状態も0になる。

以上すべてを検査した結果が下記である。

```
→  sub-theme git:(master) ✗ ./pan
warning: never claim + accept labels requires -a flag to fully verify
pan: ltl formula p1

(Spin Version 6.5.2 -- 6 December 2019)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (p1)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     - (not selected)
        invalid end states      - (disabled by never claim)

State-vector 148 byte, depth reached 1143, errors: 0
    88023 states, stored
    19554 states, matched
   107577 transitions (= stored+matched)
        0 atomic steps
hash conflicts:        63 (resolved)

Stats on memory usage (in Megabytes):
   14.774       equivalent memory usage for states (stored*(State-vector + overhead))
   10.313       actual memory usage for states (compression: 69.80%)
                state-vector as stored = 95 byte + 28 byte overhead
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  138.788       total actual memory usage
```

図3.6
図3.6からassertion violationsが+とerrors:0と確認できる。

3.3 LTLによる検証

LTL検証ではネットワーク不通や故障時を想定した場合の到達性の検証を行う。下記の検証を行った。

1. 「Payment service」が、「Payment card service」または「Payment point service」に対してリクエストをすればレスポンスが返ってくる。LTL式にすると下記になる。

```
ltl p1 { [] ( req1 -> <> res1) }
```

この検証はネットワーク不通が起こり、レスポンスがない事があり得るのでエラーとなることが期待値になる。検証した結果が下記の図3.7になる

```
➜  sub-theme git:(master) ✗ ./pan -a -N p1
pan: ltl formula p1
pan:1: acceptance cycle (at depth 16)
pan: wrote payment-distribution-system3.pml.trail

(Spin Version 6.5.2 -- 6 December 2019)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
        never claim             + (p1)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 148 byte, depth reached 35, errors: 1
       21 states, stored (24 visited)
        1 states, matched
       25 transitions (= visited+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.004       equivalent memory usage for states (stored*(State-vector + overhead))
    0.259       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage
```

図3.7

反例を出力したものが下記の図3.8になる

```
  <<<<<START OF CYCLE>>>>>
 18:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:186 (state 49)       [cable1_r_out!ack,tran,r]
 20:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:124 (state 6) [cable1_r_out?m,t,r]
 22:    proc  5 (payment_service:1) payment-distribution-system3.pml:58 (state 26)      [(timeout)]
                        card tran:0, timeout
 24:    proc  5 (payment_service:1) payment-distribution-system3.pml:58 (state 27)      [printf('card tran:%d, timeout\\n',tran)]
 26:    proc  5 (payment_service:1) payment-distribution-system3.pml:39 (state 1)       [cable1_s_out!msg,tran,1]
 28:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:121 (state 1) [cable1_s_out?m,t,type]
 30:    proc  0 (network_from_payment_to_card:1) payment-distribution-system3.pml:121 (state 2) [cable1_r_in!m,t,type]
 32:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:146 (state 1) [cable1_r_in?msg,t,type]
 34:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:156 (state 12)       [(((t==0)&&(ct0!=0))&&(type==1)))]
 36:    proc  2 (payment_card_service:1) payment-distribution-system3.pml:156 (state 13)       [r = ct0]
spin: trail ends after 36 steps
```

図3.8

「Payment　service」のプロセスからはタイムアウトして終了しているパスが存在しているのが反例として検出された。

2.「Payment service」が、「Payment card service」または「Payment point service」に対してリクエストをすればレスポンスが返ってこないこともある。LTL式にすると下記になる。

```
ltl p2 { ( req1 -> [] !res1) }
```

この検証はネットワーク不通を加味したものになるのでエラー検出されないことが期待値である。検証した結果が下記の図3.9になる

```
➜  sub-theme git:(master) ✗ ./pan -a -N p2
pan: ltl formula p2

(Spin Version 6.5.2 -- 6 December 2019)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (p2)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 140 byte, depth reached 0, errors: 0
        1 states, stored
        0 states, matched
        1 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.000       equivalent memory usage for states (stored*(State-vector + overhead))
    0.259       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage
```

図3.9

3.「Payment service」が、「Payment card service」または「Payment point service」に対してリクエストをして、ネットワークつまり「Newwork for card」または「Newwork for point」が正しく動けば、レスポンスがある。LTL式にすると下記になる。

```
ltl p3 { [] ( req1 && (<> net1_req) && (<> net1_res) -> <> res1) }
```

この検証はネットワーク不通を加味して且つネットワークが正常に動いている時であればレスポンスが得られるという検証でエラー検出されないことが期待値である。検証した結果が下記の図3.10になる

```
→  sub-theme git:(master) ✗ ./pan -a -N p3
pan: ltl formula p3

(Spin Version 6.5.2 -- 6 December 2019)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (p3)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 148 byte, depth reached 683, errors: 0
    10375 states, stored (10565 visited)
     2050 states, matched
    12615 transitions (= visited+matched)
        0 atomic steps
hash conflicts:         1 (resolved)

Stats on memory usage (in Megabytes):
    1.741       equivalent memory usage for states (stored*(State-vector + overhead))
    1.431       actual memory usage for states (compression: 82.20%)
                state-vector as stored = 117 byte + 28 byte overhead
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  129.901       total actual memory usage
```

図3.10

第4章 おわりに

4.1 まとめ

本研究では分散システムにおけるトランザクション処理の検証を行った。第1章では本研究に取り組む背景や目的について説明した。第2章では研究を行うにあたりSpin/Promelaの解説、Spin/Promelaで行える検査内容の説明を行い、検査対象のシステムの概要を説明した。第3章では次の3つの観点から検査を行った。1つ目に公平性の検査を行い、検査が偏った内容ではないことを示した。2つ目にassert構文による検証を行い、二重決済や決済の状態管理が期待値である事を示した。3つ目にLTLによる検証を行い、ネットワーク越しにトランザクション処理を行った際の到達性の検証を行った。
結果は、1.トランザクションを行う上で一意に表すidentify keyを必ず用いる事。2. identify key毎に状態管理をして処理を行う前に必ず状態チェックをしてから後続処理を行う事。の2点が正確に出来ていればトランザクション処理が正常に管理できる事がわかった。

付録　ソースコード

本研究で利用したPromelaのコードは https://github.com/nakamura244/jaist-sub-theme に登録してある

REFERENCES

[1] X/Open XA https://en.wikipedia.org/wiki/X/Open_XA
[2] Two-phase commit protocol
https://en.wikipedia.org/wiki/Two-phase_commit_protocol
[3] Spin/Promela http://spinroot.com/spin/whatispin.html
[4] https://spinroot.com/spin/Man/progress.html
[5] https://spinroot.com/spin/Man/chan.html
[6] https://spinroot.com/spin/Man/assert.html
[7] http://spinroot.com/spin/Man/ltl.html