CSP を用いた振る舞い仕様記述に関する事例研究
―自動販売機システムを題材にして―

2012SE159 水野堅登　2012SE193 小倉卓也　2012SE273 渡會真朗
指導教員：張 漢明

1　はじめに

高信頼性のソフトウェア開発において, 形式手法は有効であるといわれている. その適用事例として FelicaIC チップが有名である. 本研究では一般的な開発においても形式手法は有用であると考えている. また実際の開発現場で使われている開発文書に形式手法を適用することで信頼性が向上すると考えている.

本研究の目的は事例を用いて振る舞い仕様記述における CSP の有用性の確認をすることである. 実用レベルの開発文書において形式手法が有効であると考えている. しかし, 現在事例として実用レベルの記述は公開されていない. 振る舞いに対してプロセス代数 CSP が有効であることを示していく. 本研究の基本的なアイデアは実用レベルの開発文書に CSP を適用することである. 自然言語で書かれた開発文書の振る舞いの分析手順の提示を行う. また UML での記述法を提示し,UML の記述から CSP への対応付けをし変換を行う. 本研究ではテスト設計コンテスト ’15 の仕様書を対象としてユーザから見た販売の振る舞いをアクティビティ図で記述した. また, そのアクティビティ図から CSP の検証を行った. 本研究の評価として振る舞いにおける CSP の有効性の確認をした. 振る舞いとして書くべきことが明示になり,UML の記述法を提示することにより直感的な理解が可能になった.

2　背景技術

本章では, 形式手法の概要,CSP の概要,UML の概要, 振る舞い検証について述べる.

2.1　形式手法の概要

形式手法 (Formal Method)[4] はソフトウェア開発工程に仕様記述言語による数学的な証明を持ち込んだ手法である. 形式化の概念として, 機能, オブジェクトなどの形式化が代表的である. また形式化に用いられる数学的な道具としては, 形式論理, 普遍代数, 集合論などがある.

2.2　CSP の概要

本研究ではモデルを記述する形式仕様言語として CSP(Communicationg Sequential Process)[1] を用いる.CSP は並行システムを形式的に厳密に記述し, 解析するための理論であるプロセス代数の一つであり, イベントの集合としてシステムをモデル化する.CSP では複数のイベントの実行順序を並行合成することで並行システムを表すことができる. 組み込みシステム, セキュリティなどの幅広い分野の検証に適用されている.

2.3　UML の概要

UML(Unified Modeling Language)[6] とはグラフィカルな記法の一種で, ソフトウェアシステム, 特にオブジェクト指向スタイルを使って構築するシステムの記述や設計に利用されている.UML では様々なダイアグラムがあり, 構造に関するものとして, クラス図, オブジェクト図がある. また振る舞いに関するものとして, ユースケース図, アクティビティ図, ステートマシン図などがある.

2.4　振る舞い検証

振る舞いとはイベントの実行順序を定義したものであり. 振る舞い検証では, 設計が要求仕様を満たしていることを検証する.CSP における詳細化関係 [2] は 3 つあり, 以下に述べる.

- トレース詳細化 (P [T= Q)
  詳細化の過程で危険な実行例が入り込まない安全性を検証するために使うことが可能.
- 失敗発散詳細化 (P [F= Q)
  詳細化の過程でこれ以上何もイベントを実行することができない状態が入り込まない状態で検証するために使うことが可能.
- 失敗発散詳細化 (P [FD= Q)
  詳細化の過程で外部からは観測されない無限ループが内部で発生している状態で検証するために使うことが可能

モデル検査とはシステムの全ての有限状態を網羅的に調べて, 仕様を自動的に検証するものである. プロセス代数 CSP に対応した代表的なモデル検査器として FDR[2] がある.FDR では CSP で記述された設計と仕様の詳細関係を調べることにより検証対象が仕様を満たしているかを網羅的に走査し調べることが可能である.

3　形式手法の適用手順

本章では, 文書の分析指針,UML を用いた記述法,CSP を用いた記述法, 検証の枠組みについて述べる.

3.1　振る舞いの定義

振る舞いとはイベントの実行順序を表したものである. 本研究では開発文書の動詞に着目し, イベントの候補と定義する.

3.2　振る舞いの分析手順

イベントの候補の外部の動作と内部の動作に着目し, これをイベントと定義する. また外部イベントを要求仕様と

し, 内部イベントを設計とする. 振る舞いの分析の手順を以下に示す.

1. イベントの候補に印をつける.
2. 外部イベントと内部イベントに区別し, 印をつける. またイベント名を定義する.
3. イベントの実行順序に印をつける.

3.3 UML を用いた記述法

振る舞いの記述として UML を用いる. 外部イベントの実行順序を表す際にアクティビティ図を用いる. また内部イベントの実行順序を表したものをステートマシン図を用いる.

3.4 CSP を用いた記述法

UML から,CSP に変換して設計と仕様を記述していく上で,CSP ではライブラリを使用し記述した. 以下に使用したライブラリを CSP で記述する.

仕様

仕様はアクティビティ図をもとに外部イベントの実行順序を記述することで定義することができる.

設計

設計はステートマシン図をもとに各オブジェクトごとに記述することができる.

- 状態の有限集合

  `datatype State = 状態名`

- 入力文字列の有限集合

  `datetype Event = イベント名`

- 出力文字列の有限集合

  プロセスの集合を記述する.

- 遷移関数

  `StateTransition(オブジェクト名, 状態名, イベント名) = 遷移する状態名`

- 出力関数

  `ACTION(オブジェクト名, 状態名, イベント名)= 出力する名前`

- 開始状態

  `InitialState(オブジェクト名) = 状態名`

3.5 検証の枠組み

振る舞い検証では、仕様と設計を記述する. 仕様は外部からの入力に対する期待される結果を記述する. 設計ではプロセスの集合で表し, 各プロセスの状態遷移の振る舞いを記述する. 振る舞い検証では, 設計に入力を加えた時の振る舞いと, ユーザーの振る舞いに対して期待する結果の振る舞いの双方で詳細化関係が成り立つことで等価であると確かめる. 図 1 で検証の概要を示す.

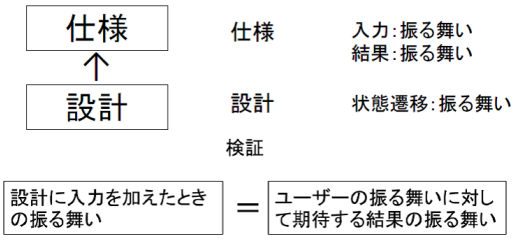

図 1 検証の概要

外部とシステムの境界の事象の仕様をインターフェースで記述する. 振る舞い仕様には, インターフェースで定義されたイベントを用い, 外部とシステムとの入出力を記述する. 設計には, システムの各コンポーネント間の振る舞いを記述する. インターフェースと振る舞い仕様の間のイベントのやり取りが変わらなければ, 設計を変えるだけで記述することが可能になるので, 振る舞い仕様記述を再利用できる.

区間

前提として並行システムでは複数の事象が同時に発生することがある. よって同時に事象が発生することをを考慮する必要があると考えられる. そこで区間 [3, 5] の概念を用いて事象の同時生を記述することを明示する. 区間では, 複数の事象が同時に発生することを表すために, 開始と終了を明示する. 本研究では同時に発生するということを, ある区間の中で複数の事象が発生することとし, 開始終了付き事象という概念を用いて定義する. 開始終了付き事象は, 開始, 終了, 事象の 3 項組として定義され, 事象は開始と終了の間で発生する, または発生しないとして振る舞いが行われる. 区間は開始終了付き事象の集合として定義され, インターリーブとして扱われる. 区間の開始は, 開始終了付き事象のいずれかの開始で, 終了は開始終了付き事象の終了が全て同期した時である. 本研究ではこの区間の概念を取り入れ, 検証を行った.

設計

設計はステートマシン図から各コンポーネントの振る舞いを抽出し,3.4 項の CSP のライブラリを使用して記述している. 区間振る舞いのモデルから, 複数の区間を合成して, 同時に発生する事象を考慮し状態遷移を記述する. 設計を記述する上で, 区間の概念を用いて, 以下の式で記述した.

```
SECTION_INPUT_SELECT(ss, S) =
[] s:ss @ SECTION_INPUT(s, S)

SECTION_INPUT(occurred, S) =
(||| (start, end, event):occurred @
start -> event -> end -> SKIP)
|||
(||| (start, end, event):diff(S, occurred) @
```

start -> end -> SKIP)
この式を用いることで, 選択事象と同時事象同時に設計で記述することが可能となった.

仕様

仕様は外部とコンポーネントの入出力関係を記述する. 本研究では入出力関係を, 外部からの入力に対応して, コンポーネントは入力に応じた振る舞いをすることとする. 入力時の出力の振る舞いを定義し, 出力は外部事象との入出力関係で定義される.

検証

仕様が設計を満たす検証を行うには, 以下の式が成り立てば良い

仕様 = コンポーネントの集合\ {**要求仕様以外のイベント**}

システムの振る舞いが要求仕様を満たしているかを調べる検証において要求仕様と検証対象となるシステムが双方で詳細化関係が成り立てば良い, その検証方法として下記がある.

1. 仕様 $\sqsubseteq$ コンポーネントの集合
   検査対象のコンポーネントが求められる要求仕様を満たしているか調べる検証であり, 要求仕様の振る舞いの詳細化が検証対象のコンポーネントであれば検証対象のシステムは求められた要求仕様を満たしている.
2. 仕様 $\sqsupseteq$ コンポーネントの集合 \ {**要求仕様以外のイベント**}
   検査対象のコンポーネントの記述が正しいか調べる検証であり, 検査対象のコンポーネントを要求仕様記述されているイベント以外を隠蔽する. 隠蔽したシステムの振る舞いが, 要求仕様との詳細化関係を満たせばコンポーネントの記述は正しい.

4 事例:自動販売機システムへの適用

本章では, 自動販売機システムの概要, 開発文書の分析結果,UML の記述結果, 検証結果について述べる.

4.1 自動販売機システムの概要

本研究ではテスト設計コンテスト'15 の ASTER 自動販売機ユースケース仕様書. また ASTER 自動販売機ハードウェア構成および販売者用機能仕様書を対象とする.

4.2 開発文書の分析結果

開発文書の分析結果を以下に示す.

1. 開発文章の動詞に赤線を引く.
2. 赤線を引いた動詞を外部イベント, 内部イベントにそれぞれ青, 緑の線を引き, イベント名を定義する.
3. イベントの実行順序を表した部分に黄線を引く.

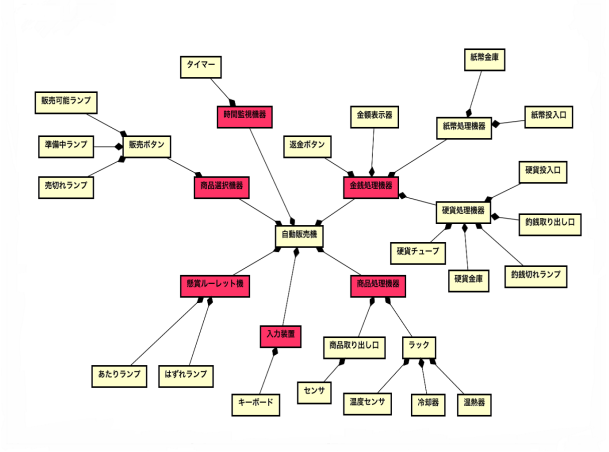

図 2 自動販売機のオブジェクト図

開発文書の動詞を抽出

- 硬貨を投入する.
- 販売ボタンを押下する.
- 正当不当判定を行う.

イベント候補を外部イベント, 内部イベントに分け, イベント名を定義する. まず, 外部イベントを定義する.

- 硬貨を投入する→コイン投入
- 販売ボタンを押下する→ボタン押下

次にイベントの実行順序を抽出する.

- 販売ボタンを押下し商品を受け取る.

4.3 UML の記述結果

仕様書を分析し, アクティビティ図にした結果を以下に示す.

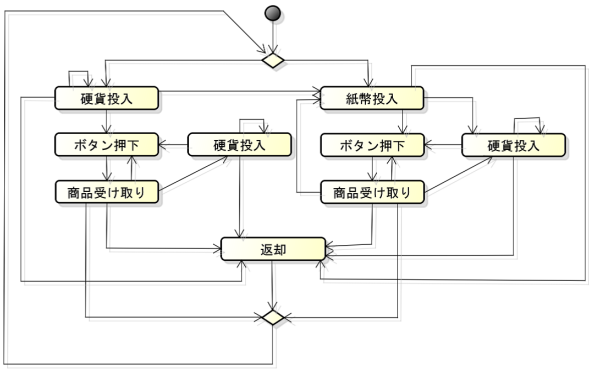

図 3 分析したアクティビティ図

図 3 は, コイン, 紙幣を入れてから返却, 商品を排出するまでの一連の動作を記述している. 返却, 商品排出後にはじめに戻るのは連続購入を表している. 対象としている自動販売機システムの開発文書に紙幣は一枚しか投入できない記述があり, 商品購入後にはもう一枚紙幣の投入が可能と

なる. またコインからのボタン, 商品, コインの構造と紙幣からのボタン, 商品, コインの構造は同じである.

4.4 検証結果

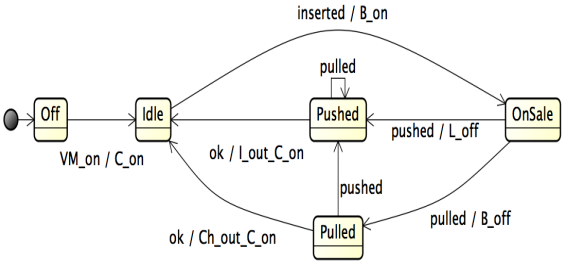

図 4 VM のステートマシン図

本研究ではボタン 1 商品 1 のモデルの自動販売機を検証した. そこでの仕様は以下のようになる.

- コインを入れてボタンを押したら商品を排出.
- レバーを引いたらお釣りを排出.
- ボタンとレバーが同時に事象が発生した場合商品を排出.

これらの仕様の検証を行った. 以下に検証式を示す.

- 設計を CSP での式を以下に示す

```
PL = SECTION_INPUT_SELECT({{COIN}}, {COIN});
SECTION_INPUT_SELECT
({{BUTTON},{LEVER},{BUTTON,LEVER}},
{BUTTON,LEVER});
PL
```

ここではそれぞれの区間ごとに集合を定義した. ボタン, レバーの区間では, ボタンかレバー, またはボタンとレバーの同時の事象で発生した場合を記述した.

- 仕様を CSP での式を以下に示す

```
PL_SPEC = EXT_INSERTED;
((EXT_PUSHED;EXT_ITEM)|~|
(EXT_PULLED; EXT_CHANGE)|~|
((EXT_PUSHED ||| EXT_PULLED); EXT_ITEM ));
PL_SPEC
```

ここではボタンを押したら商品を排出, レバーを引いたらお釣りを排出, 同時に事象が発生したら商品を排出するという三つの場合を内部選択で記述した.

- 設計と仕様の検証式を示す

```
assert PL_SPEC [FD=
TARGET(PL_MODEL, ExternalEvents)
assert TARGET(PL_MODEL, ExternalEvents) [FD=
PL_SPEC
```

この二つの詳細化関係が成り立つことで等価であると言え, この自動販売機が正常に動作することが検証されたと言える.

5 考察

本章では, 開発文書への効果,UML 記述の効果, 検証の効果についての考察を述べる.

5.1 開発文書への効果

開発文書は機能と振る舞いに関して構造化されていない. 文章内の記述の問題点は, 同じ動詞が何度も使われている. また, ユーザという言葉をアクターと記述していたり, 言葉の統一性がないことと, 硬貨投入やボタン押下などには, ユーザから見た能動的な文章と自動販売機から見た受け身の文章があり, 二つとも同じイベントになるが見極めるのが困難であることがあげられる.

5.2 UML 記述の効果

分析の結果を UML で記述することで一目で見て何が書かれているのかわかりやすい. また,UML の知識がある人の中では共通の認識を持つことが期待出来る.

5.3 検証の効果

CSP のライブラリを使用することで, 仕様の記述が容易となり, 仕様の意味も容易に理解することができる. ライブラリを使用することで, 入出力の記述を全て列挙する必要がなくなり, 効率の向上にも繋がり, 入力部分を複数に増やした場合も容易に行えると考えた.

6 おわりに

本研究では振る舞い記述を CSP で記述することで, ソフトウェア開発においての実用レベルの文書において CSP の有効性を確認することができた. 今後の課題として, より複雑なシステムの仕様の記述とその検証を行い, 複雑なシステムでも有効であることを確認することがあげられる.

参考文献

[1] C.A.R.Hoare, Communicating Sequential Processes, Prentice-Hall,1985.

[2] 磯部祥尚, 並行システムの検証と実装 形式手法 CSP に基づく高信頼並行システム開発入門 トップエスイー実践講座, 近代科学社,2012.

[3] 井上啓佑, 寺西祐斗, 友松良輔,”同時性を考慮したアーキテクチャの振舞い検証手法に関する研究,”南山大学 2014 年度卒業論文,2014.

[4] 玉井哲雄, ソフトウェア工学の基礎, 岩波書店,2004.

[5] 張漢明, 野呂昌満, 沢田篤史,”同時性を考慮した並行システムの振舞い検証に関する考察,”情報処理学会研究報告, vol.2015-EMB-36, no.13, 2015.

[6] マーチン・ファウラー,UML モデリングのエッセンス, 翔泳社,2005.