組込みシステムにおける振舞い検証に関する研究

2007MI069 稲垣 尋紀　　2007MI221 杉浦 友紀　　2007MI224 鈴木愛美香
指導教員　沢田 篤史　蜂巣 吉成

1　はじめに

組込みシステムの開発では，並行性，実時間性，耐故障性など本質的に困難な問題を含んでいるので，実装時の欠陥を減らすことが重要である．システムの振舞いを検証する手法として，モデル検査がある．モデル検査を用いてアーキテクチャの振舞いを検証することで，アーキテクチャレベルにおいてシステムの欠陥を検出することができる．モデル検査を用いた振舞い検証を実用化する際の問題として，状態爆発の発生と厳密な仕様記述が困難なことがある．モデル検査はシステムの振舞いを網羅的に探索するので状態爆発が起こる場合がある．また，並行に動作するシステムの振舞いは非決定的なので，厳密に仕様を記述することが困難である．状態爆発を回避するには検証を分割することが有効であるが，整合性を保証する必要がある．

本研究の目的は，モデル検査を用いた振舞い検証における，検証を分割した際の整合性の保証および仕様記述の支援である．検証の分割方法として，コンポーネントごとの検証を最小単位とした分割方法を提案する．検証の分割により状態数を削減する．整合性の検証の支援として，検証条件をインターフェースの振舞いを用いて記述することを提案する．整合性を検証する際の仕様記述の支援として，インターフェースの振舞いの型付けをおこなう．

本研究で提案するコンポーネント合成時の整合性の検証条件を事例に適用した．事例として，本研究室が提案している E-AoSAS++ [4] に基づいたシステムのアーキテクチャを検証した．インターフェースの振舞いの型の抽出には，GoF デザインパターン [2] を利用した．本研究で提示する手法を適用することで，整合性の検証をする際の状態数を減らすことができ，仕様記述の省力化をおこなうことができた．

2　本研究のアイデア

本研究では，検証の分割方法の提案，検証を分割した際の整合性の検証条件の提案と仕様記述の支援をおこなう (図 1)．インターフェースの振舞いはコンポーネント間の振舞いを表しており，インターフェースの振舞いを検証することでコンポーネント間の整合性が検証できる．整合性を検証する際の仕様記述の支援として，インターフェースの振舞いの型付けをおこなう．型付けられたインターフェースの振舞いを再利用することで，型に応じた仕様を記述でき，効率良く記述できるようになる．また，提案方法の適用範囲を組込みシステムのアーキテクチャに対する検証とし，状態遷移図において，受信できるイベントがきめられている場合とする．

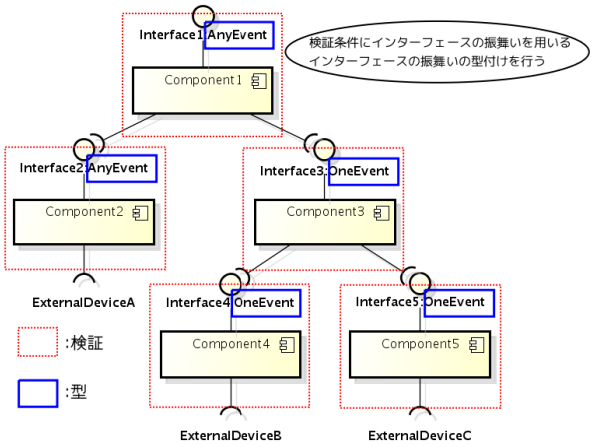

図 1　本研究のアイデア

3　振舞い検証の枠組み

CSP を用いたアーキテクチャの振舞い検証について説明する．

3.1　E-AoSAS++ の並行実行モデル

E-AoSAS++ に基づいたシステムは，並行に動作する状態遷移機械の集合からなる．各状態遷移機械がキューを介して互いにイベントを非同期に通信することで並行動作を実現している (図 2)．

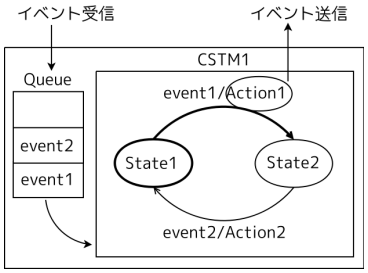

図 2　キューを介したイベントの送受信

3.2　CSP を用いた振舞い検証

アーキテクチャの振舞い，検証条件をプロセス代数である CSP を用いて表現し，FDR で詳細化関係を検証することにより，振舞い検証をおこなう．

3.2.1　CSP によるプロセス記述

図 2 の状態遷移機械，イベントキューは CSP によって以下のように記述する．イベントキューのようなアーキテクチャの振舞いは，CSP ライブラリとして汎用的に表現する．

CSP による状態遷移機械と非同期通信の記述

```
状態遷移機械
CSTM1 = State1
State1 = RECV(CSTM1, event1) ; Action1 ; State2
State2 = RECV(CSTM1, event2) ; Action2 ; State1
Action1 = SEND(CSTM2, event3)
Action2 = SEND(CSTM2, event4)

非同期通信 (FIFO)
CH_FIFO(ch, seq, size)=
  #seq < size & send.ch?event->
   CH_FIFO(ch, seq^<event>, size)
  []
  #seq > 0 & recv.ch!head(seq)->
   CH_FIFO(ch, tail(seq), size)
```

3.2.2 環境と仕様

仕様は，環境に対する振舞い (振舞い仕様) であり，着目するイベントに対する振舞いとして記述する．環境と振舞い仕様は，表 1 のプロセス記述を用いて表現する．

表 1　プロセス記述 (一部抜粋)

	アーキテクチャの振舞い	CSP 記述
実行プリミティブ	イベント送信	SEND(channel, event)
	イベント受信	RECV(channel, event)
演算子	逐次実行	P1 ; P2
	選択実行 (外部選択)	P1 [ ] P2
	選択実行 (内部選択)	P1 |~| P2
	並行実行 (インターリーブ)	P1 ||| P2

```
SEND(channel, event) = send.channel.event -> SKIP
RECV(channel, event) = recv.channel.event -> SKIP
```

3.2.3 CSP を用いた検証

FDR では, デッドロックフリー, ライブロックフリー, 詳細化関係 (Refinement) を検査できる．FDR が詳細化関係を用いて検証できる性質を表 2 に示す．

表 2　詳細化関係で検証できる性質

詳細化関係	検証できる性質
Trace refinement	安全性
Failure refinement	活性，デッドロックフリー
Failure divergence refinement	ライブロックフリー

振舞い検証の概略を図 3 に示す．環境，振舞い仕様，検査対象をそれぞれ CSP で記述し，FDR で詳細化関係を検証することにより，安全性，活性の検証ができる．

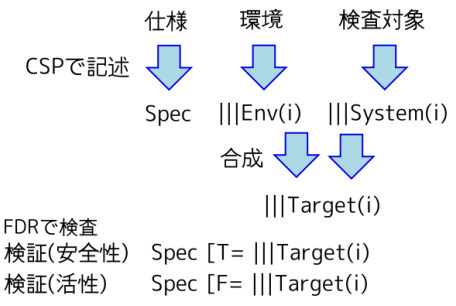

図 3　検証の概略

4 コンポーネントとインターフェースの振舞いによる整合性の検証

整合性の検証条件にインターフェースの振舞いを用いることを提案し，コンポーネント合成時の整合性の支援をする．インターフェースは二つの間の振舞いを表したものであり，コンポーネントとインターフェースに着目する事で整合性を検証する際の仕様および環境をインターフェースの振舞いで表す事ができる．

4.1 コンポーネントとインターフェースの振舞い

コンポーネントは状態遷移機械，インターフェースの振舞いはコンポーネント間のイベントの送受信で表現する．インターフェースの振舞いは，提供側から見たインターフェースの振舞い (提供インターフェースの振舞い) であり，要求側から見たインターフェースの振舞い (要求インターフェースの振舞い) の送受信を入れ換えたものとなる (図 4)．

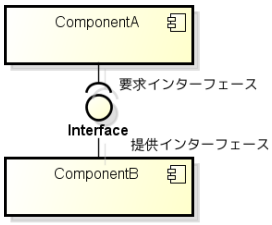

図 4　コンポーネントとインターフェース

```
Provider  = RECV(event1) ; SEND(event2)
Requester = SEND(event1) ; RECV(event2)
```

4.2 整合性の検証

整合性の検証の意義，検証条件を示す．検証をコンポーネントごとに分割しておこなう場合，コンポーネント間の整合性の検証をおこなう必要がある．コンポーネント間のインターフェースの振舞いを仕様として検証することにより，コンポーネント間の振舞いを検証できるので整合性を確認できる．検証条件はコンポーネント，インターフェースの振舞い，要求インターフェースの振舞いで構成される (図 5)．

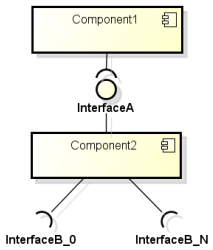

図 5　検証の単位

検証条件として，仕様は提供インターフェースの振舞い，環境は複数の要求インターフェースの振舞いを合成したものを記述する．振舞い検証として，環境とコンポーネントを合成したものと仕様間の詳細化関係を検証する．

```
assert InterfaceA [= Environment||System(i)
Environment = R(InterfaceA)||(||i:0..N@R(InterfaceB_i))
```

環境を記述する際に，複数の要求インターフェースの振舞い間で同期させることにより，一つのコンポーネントに対するイベントの同時発生を防ぐ．

4.3 事例検証

複合ボタンを事例に，コンポーネントの検証条件を示す．CompButton は，Button1 または Button2 が押されたことを，Controller に通知する．Button1，Button2，CompButton について，それぞれ検証をおこなう (図 6)．

CompButton の検証条件を以下に示す．

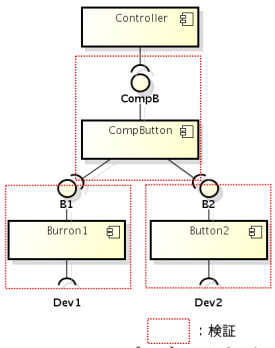

図 6　CompButton のコンポーネントとインターフェース

CompButton の整合性の検証

```
SYNC_ENV =
{send.CompButton.b1Pushed, send.CompButton.b2Pushed}

ENV = (B1 ||| B2) [|SYNC_ENV|] R(CompB)
SPEC = CompB

CompB =
 RECV(CompButton,enable);
 ((RECV(CompButton,b1Pushed)
   []
   RECV(CompButton,b2Pushed));
  SEND(Controller,pushed)
  []
  RECV(CompButton,disable))
B1 = RECV(B1,enable);
     (RECV(B1,push); SEND(B1,b1Pushed)
      []
      RECV(B1,disable))
B2 は B1 と同様
  ※ R(CompB)：CompB の要求インターフェース
```

複合ボタンの振舞い検証の結果は真となった．

5　インターフェースの振舞い型付け

型を用いてインターフェースの振舞いを再利用することにより，仕様記述の省力化をはかる．型はよくある振舞いの記述を一般化したものであり，型に必要な情報をパラメータ化する．

5.1 インターフェースの型による仕様記述

インターフェースの振舞いを型付けする際の視点として，よくある設計のパターンをカタログ化したデザインパターンに着目した．デザインパターンの動的振舞いを参考にして，インターフェースの型付けをおこなう．

■ Composite パターンによる型付け

Composite パターンは再帰的な構造において，複合オブジェクトとプリミティブオブジェクトを同一視する (図 7)．同じインターフェースの振舞いが複数存在するものをインターフェースの振舞いの型とする．

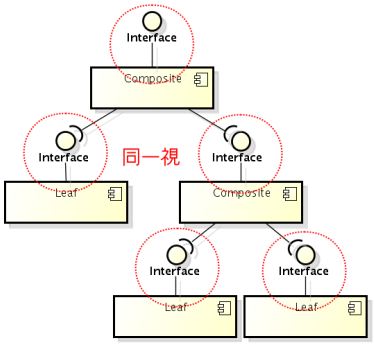

図 7　Composite パターンにおけるインターフェースの振舞い

5.2 インターフェースの振舞いのパラメータ化

インターフェースの振舞いは，提供側と要求側のコンポーネントによって決まる．提供側と要求側のコンポーネントおよび通信するイベントをパラメータ化することで，仕様記述の際にインターフェースの振舞いを再利用できる．4.3 章における複合ボタンのインターフェースの振舞いをパラメータ化したものを以下に示す．

インターフェースの振舞いのパラメータ化

```
OneEvent(pro, req, event) =
    RECV(pro, enable) ;
    (SEND(req, event) [] RECV(pro, disable))

CompB : OneEvent(CompButton, Controller, pushed)
B1    : OneEvent(Button1, CompButton, b1Pushed)
B2    : OneEvent(Button2, CompButton, b2Pushed)
```

5.3 型を利用した事例検証

4.3 章で示した複合ボタンは Composite パターンに基づいているので，インターフェースの振舞いの型が適用できる．CompButton の検証条件を以下に示す．

型を利用した CompButton の検証

```
SYNC_ENV =
{send.CompButton.b1Pushed, send.CompButton.b2Pushed}

ENV = (B1 ||| B2) [|SYNC_ENV|] R(CompB)
SPEC = CompB

CompB = OneEvent(CompButton, Controller, pushed)
B1 = OneEvent(Button1, CompButton, b1Pushed)
B2 = OneEvent(Button2, CompButton, b2Pushed)
```

インターフェースの型を用いることにより，仕様の記述が決まり検証条件が導出できる．

6　考察

本研究で提案する手法の有用性を評価するために，提案する手法を E-AoSAS++ に基づいたシステムに適用し，状態数の削減および仕様記述の省力化について考察する．また，GoF デザインパターンを参考にしたインターフェースの振舞いの類型化について考察する．

6.1　状態数の削減に関する考察

本研究で提案する検証方法を複合ボタンシステムに適用した．検証方法を適用した場合と適用しない場合を比較することで，状態数が削減できたかを評価する．

表 3　状態数の比較

	適用前	適用後
コンポーネント数	3	3
状態数	34	28
実行プリミティブ数	49	32

比較した結果，検証方法を適用したことにより，状態数および遷移数が削減できたことがわかる．また，大規模なシステムを検証すれば，さらに状態数を削減できると考えている．

6.2　仕様記述の省力化に関する考察

本研究で提案するインターフェースの型を自動販売機システムに適用した．型の適用結果を図 8 に示す．

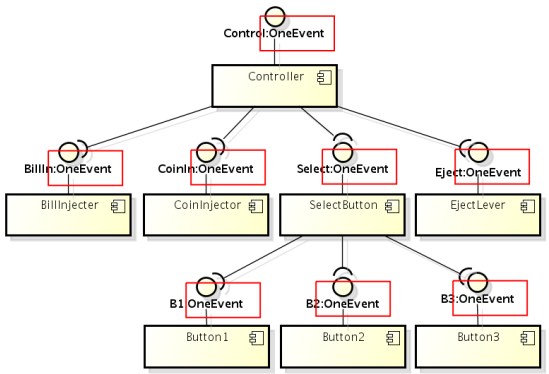

図 8　Controller 部分のコンポーネントとインターフェースの振舞い

自動販売機システムの Controller 部分に型を適用できた．型を適用した場合のインターフェースの振舞いの記述を以下に示す．

適用後の仕様記述

```
Bill   :OneEvent(BillInjector, Controller, drop_in)
Coin   :OneEvent(CoinInjector, Controller, drop_in)
Select:OneEvent(SelectButton, Controller, select)
B1     :OneEvent(Button1, SelectButton, select1)
B2     :OneEvent(Button2, SelectButton, select2)
B3     :OneEvent(Button3, SelectButton, select3)
Eject :OneEvent(EjectLever, Controller, return_money)
```

インターフェースの振舞いの型を用いることでインターフェースの振舞いの再利用が可能となり，仕様記述を省力化できると考える．

6.3　インターフェースの振舞いによるデザインパターンの整理

インターフェースの振舞いの型付けをする際に，GoF デザインパターンを参考にした．GoF デザインパターンは再利用を目的としたパターンをカタログ化したものである．デザインパターンの動的振舞いに着目して分類したところ，以下の 2 種類が型となると考えた．

- 1 型：複数の同じインターフェースの振舞いが存在する
- 2 型：インターフェース間に依存関係がある
- その他：インターフェース間に依存関係がない

各視点から，デザインパターンを分類した結果を表 4 に示す．

表 4　動的振舞いによる分類

1 型	2 型	その他
Abstract Factory	Adapter	Facade
Adapter	Bridge	Singleton
Bridge	Builder	Template Method
Chain of Responsibility	Chain of Responsibility	
Command	Command	
Composite	Composite	
Decorator	Decorator	
Factory Method	Factory Method	
Interpreter	Flyweight	
Mediator	Interpreter	
Memento	Iterator	
Observer	Memento	
Prototype	Proxy	
Proxy	Visitor	
State		
Strategy		
Visitor		

1 型はコンポーネント間の整合性の検証をおこなう際に利用できた．また，2 型は複数のコンポーネント間の検証をする際に利用できると考えている．

7　おわりに

本研究では，モデル検査を用いた振舞い検証において，インターフェースの振舞いの型付けをおこなうことにより，振舞い検証の省力化をおこなった．

今後の課題として，インターフェースの振舞いの型ごとのイベントの検証方法を提案する必要がある．

参考文献

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

[2] E.Gamma, R.Helm, R.Jhonson, and J.Vlissiders, *Design Patterns: Elements of Reusable Object Oriented Software,* Addison-Wealey, 1995.

[3] Formal Systems(Europe), *“FDR2 User Manual,”* http://fsel.com/documentation/fdr2/html/fdr2manual.html, 2005.

[4] 加藤大地，蜂巣吉成，沢田篤史，野呂昌満，“ アスペクト指向に基づくソフトウェアアーキテクチャの文書化方式，” 知能ソフトウェア工学研究会 (KBSE), vol.108，no.449，pp.55-60，2009.