インタラクティブシステムの自動生成に関する研究

2014SE012 古川敦啓　2014SE054 近藤諒
2014SE068 水谷康佑
指導教員：野呂昌満

1　はじめに

近年，スマートデバイスの急速な普及に伴い，その上で稼働するアプリケーションソフトウェアの開発の省力化が必要となってきた．開発の省力化を目指して自動生成に関する研究が盛んに行われている [3].

スマートデバイスアプリケーションの多くはインタラクティブシステム なので，MVC アーキテクチャを用いて開発されているのが通常である．View や Controller の自動生成は容易であり，それらに関する研究もある [5]．他方で Model はビジネスロジックを記述したコンポーネントであり，アプリケーションソフトウェアの論理はアプリケーションによって異なる．よって Model の自動生成は本質的に難しい．

本研究の目的は，本研究室で提案されている CSA/I-Sys における Model の自動生成の可能性を考察することである [1]．現在 UML を入力とする自動生成に関する研究は行われているが [4]，本研究では Model の中でもアプリケーションの論理の自動生成の可能性を考察する．アプリケーションの論理を自動生成するにあたり，特定の言語に依存しない生成系の実現を目指す．アプリケーションの論理の自動生成を目指すことで，アプリケーションソフトウェアの開発の省力化を実現する．

ドメイン内に共通する振舞いを分類し，抽象化を行い，それをメタ状態遷移機械で記述する．メタ状態遷移機械を詳細化するために，Composite パターンを適用する．これにより，複合状態はベース状態遷移機械を複数持っているので，BusinessLogic の状態遷移機械は階層化される．ベース状態遷移機械をコンポーネントとして標準化し，各コンポーネントにラベルを付け，それにパラメータを与えることでアプリケーションの論理の自動生成を実現する．

State パターンと Composite パターンを用いたアプリケーションフレームワークを設計することにより，HotSpot と FrozenSpot を明確にした．特定のドメインに対して，状態遷移機械を抽象化することで，アプリケーションの論理となる HotSpot を自動生成できた．

2　背景技術

2.1　デザインパターン [2]

デザインパターンとは，一般的な設計構造のキーとなる側面に名前を付け，抽象化，識別化し，再利用可能なオブジェクト指向設計を生み出すのに有用となるようにしたものである．本研究では，State パターンと Composite パターンの 2 つのデザインパターンを適用する．

2.1.1　State パターン

State パターンは，オブジェクトの内部状態が変化したときに，オブジェクトが振舞いを変えるようにするデザインパターンである．クラス内では，振舞いの変化を記述せず，状態を表すオブジェクトを導入することでこれを実現する．次に示すいずれかの場合に，State パターンを利用する．

- オブジェクトの振舞いが状態に依存し，実行時にはオブジェクトがその状態により振舞いを変えなければならない場合
- オペレーションが，オブジェクトの状態に依存した多岐にわたる条件文を持っている場合

2.1.2　Composite パターン

Composite パターンは，部分–全体階層を表現するために，オブジェクトを木構造に組み立てるデザインパターンである．次に示すいずれかの場合に，Composite パターンを利用する．

- オブジェクトの部分–全体階層を表現したい場合
- クライアントが，オブジェクトを合成したものと個々のオブジェクトの違いを無視できるようにしたい場合

2.2　CSA/I-Sys[1]

CSA/I-Sys は，アーキテクチャとアプリケーションの設計及びコードの理解， 変更を容易にし，ライブラリやミドルウェアを大きな粒度で変更する枠組みを提供する．CSA/I-Sys の具象アーキテクチャを図 1 に示し，以下に Model 部について示す．

– Model

Business Logic : Model や View 更新のロジックを状態遷移機械として抽出化したもの
ApplicationDataModel : データベース

3　アプリケーションフレームワークの設計

アプリケーションフレームワークを設計するにあたり，以下の点を考慮し設計を行う．

- 言語独立なアプリケーションフレームワークの設計

インタラクティブシステムは様々な言語で開発されているので，言語独立なアプリケーションフレームワークの設計を行うことを考えた．

本研究では状態遷移機械で記述できるエンタープライズなアプリケーションを対象とする．よって BusinessLogic

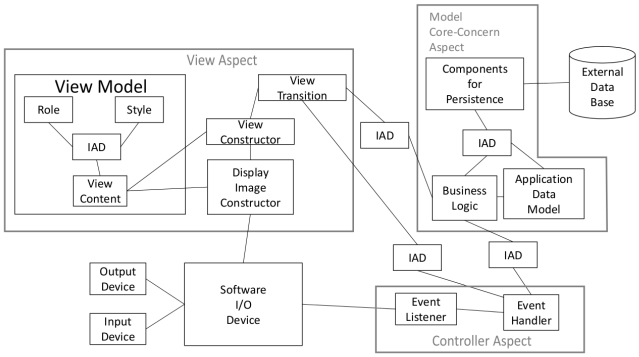

図 1　CSA/I-Sys の具象アーキテクチャ

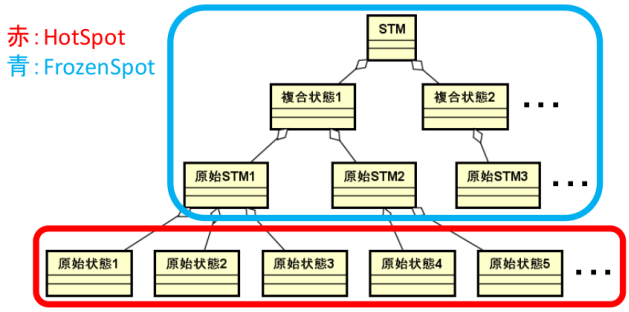

図 2　BusinessLogic の HotSpot と FrozenSpot

は状態遷移機械で記述する．状態遷移機械には State パターンと Composite パターンの 2 つのデザインパターンを適用する．これによりパターンを用いた素直な設計を実現できる．

State パターンは状態によって振舞いを変えるデザインパターンであるのでイベントに対して状態ごとにアクションを変えることができる．状態遷移機械に State パターンを適用させることにより，BusinessLogic の状態をオブジェクトとして定義できる．

Composite パターンはオブジェクトを木構造に組み立てるデザインパターンであるので，状態遷移機械は複合状態と原始状態に階層化できる．複合状態は複数の状態遷移機械を持っており，原始状態は状態のみを表したものである．よって，State パターンと Composite パターンを組み合わせることで，状態遷移機械を詳細化することができる．

アプリケーションフレームワークを設計する際に，HotSpot と FrozenSpot を特定する必要がある．FrozenSpot は様々なアプリケーションにおいて同じような記述となるので，状態遷移機械を生成系の入力とすれば，FrozenSpot は自動生成可能である．一方，HotSpot はアプリケーションの機能によって異なる部分なので，その自動生成は本質的に難しい．本研究ではベース状態遷移機械をコンポーネント化しラベル化することで，HotSpot の自動生成を実現する．

State パターンと Composite パターンを用いたアプリケーションフレームワークにおいて BusinessLogic の HotSpot と FrozenSpot は図 2 のようになっている．

4　ビジネスロジックの生成

4.1　生成の概要

アプリケーションの論理を決定するために，BusinessLogic を抽象化し，メタ状態遷移機械として定義する．しかし，アプリケーションの論理はドメインにより大きく異なるので，状態遷移機械を定義することは難しい．そこでドメインを特定し，そのドメイン内に共通する振舞いを分類し抽象化を行う．メタ状態遷移機械を生成系の入力とすれば，アプリケーションフレームワークにおける FrozenSpot は自動生成可能である．

個々のアプリケーションに対応させるためには，状態遷移機械を詳細化する必要がある．定義したメタ状態遷移機械の各状態は複合状態であり，複数のベース状態遷移機械を持っているので，階層化が可能となる．よって状態遷移機械は詳細化できる．

最下層に位置するベース状態遷移機械はアプリケーションフレームワークにおける原始状態遷移機械であり，各状態は原始状態である．原始状態遷移機械は HotSpot であるので，コンポーネントとして標準化する必要がある．ベース状態遷移機械を標準化したコンポーネントとして定義し，HotSpot カスタマイズに使用する．そして標準化したコンポーネントにラベル付けを行い，これを自動生成の鍵とする．

状態遷移機械を仕様として与え，標準化したコンポーネントをインスタンス化することで HotSpot の自動生成は可能となる．標準化したコンポーネントをインスタンス化するためには，ベース状態遷移機械をパラメータ化する必要がある．本研究ではパラメータを決定するために，ステレオタイプを用いたクラス図を与える (以下，パラメータ図と呼ぶ)．ベース状態遷移機械にパラメータを与えればインスタンス化が可能となるので，HotSpot は生成できる．

4.2　生成系の設計

我々はドメイン内に共通する振舞いを分類し，メタ状態遷移機械として抽象化する．ベース状態遷移機械を標準化されたコンポーネントとして定義し，各コンポーネントにラベルを付け意味を持たせ，これを自動生成の鍵とする．HotSpot はベース状態遷移機械をインスタンス化することで生成可能となる．

5　事例検証

本研究ではショッピングサイトを事例とし，BusinessLogic の自動生成について考察する．ショッピングサイトというドメイン内に共通する商品購入という振舞いを分類し抽象化した．これを最上位のメタ状態遷移機械として図 3 に示す．

この状態遷移機械を入力とすれば，アプリケーションフ

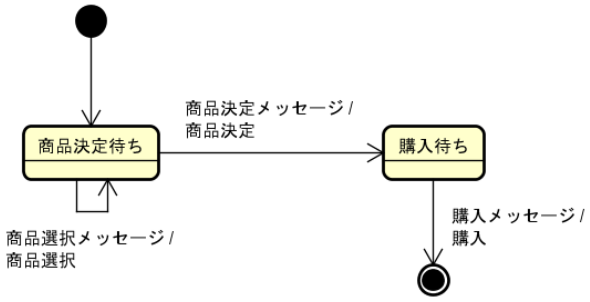

図 3　最上位のメタ状態遷移機械

レームワークの FrozenSpot である複合状態遷移機械の自動生成が可能である．図 3 の商品決定待ちと購入待ちはそれぞれ複合状態であるので複数のベース状態遷移機械を持っている．これにより，状態遷移機械は階層化される．

それぞれのベース状態遷移機械を定義することで，BusinessLogic の状態遷移機械は詳細化される．例として，商品カスタマイズのベース状態遷移機械を図 4 に示す．

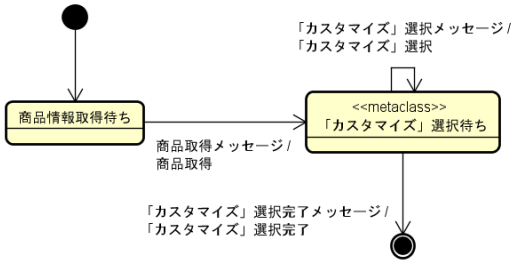

図 4　商品カスタマイズの状態遷移機械

ベース状態遷移機械を標準化したコンポーネントとし，パラメータ化する．商品カスタマイズの仕様として，「カスタマイズ」は商品カスタマイズのパラメータである．パラメータとして考えられるものには色やサイズが考えられ，これを与えることでカスタマイズ選択待ちは色選択待ちやサイズ選択待ちとなる．

パラメータが複数与えられた場合，与えられたすべてのパラメータを並列に処理する．すべてのパラメータを並列に処理するために，商品カスタマイズは並行状態として記述され状態遷移機械は図 5 のような構造となる．

コンポーネントとして定義するだけでは，自動生成の鍵とすることは難しい．よってコンポーネント自体にラベルを付けることで意味を定義する必要がある．ここでは商品カスタマイズの状態遷移機械に“商品決定「カスタマイズ」”というラベルを付ける．ここから，商品決定状態において商品カスタマイズを選択しているということが分かる．「カスタマイズ」の部分はパラメータとして与える部分である．

パラメータを決定するにあたり，図 6 のようなパラメータ図を入力とする．ベース状態遷移機械は Java でいうジェネリックスのような総称型で定義されている．パラメータ図は，そのベース状態遷移機械に対するパラメータ

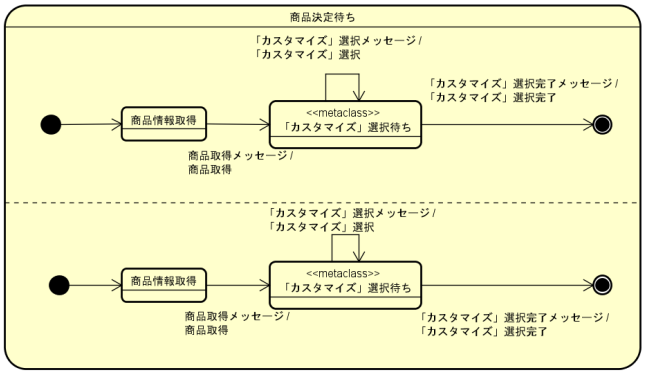

図 5　並行状態で記述された商品カスタマイズの状態遷移機械

を記述する．<< metaclass >> はインスタンスがクラスであるクラスを表すステレオタイプである．よって，カスタマイズ選択待ちは総称型となり，色やサイズがパラメータとなる．

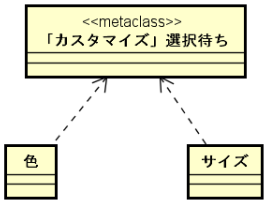

図 6　パラメータ図

パラメータ図とベース状態遷移機械を対応させるために，標準化したコンポーネントに付けたラベルをパラメータ図にも付ける．パラメータ図を生成系の入力とすることで，ベース状態遷移機械のパラメータが決定する．

実際に商品カスタマイズにパラメータ図を与えたベース状態遷移機械を図 7 に示す．

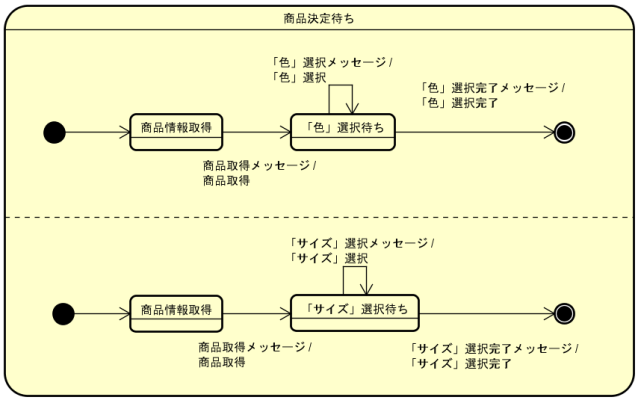

図 7　パラメータ図を与えたベース状態遷移機械

これにより，最下層の部分が生成できたので HotSpot が生成できたといえる．よって特定のドメインに対してアプリケーションの論理の自動生成が可能となる．

6　考察

本研究と先行研究を比較し，考察を行う．先行研究として「UML を入力とするソースコード自動生成ツールの開発」[4] と「ステートマシン図を用いたプログラムの自動生成支援」[6] が挙げられる．

[4] の概要を図 8 に示す．

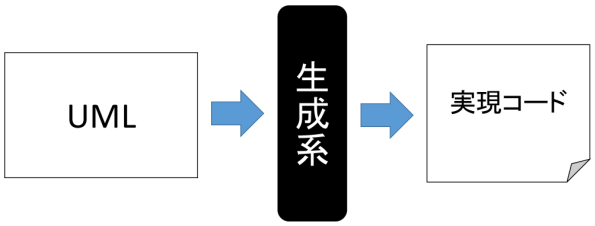

図 8　先行研究 [4] の概要

[4] では図 8 に示すように，UML とコードの関係はほぼ等価なものである．これは図式からコードへの変換を行っているだけであり，FrozenSpot のみを自動生成している．一方，我々は [4] に加え，コンポーネントをラベル化し，状態遷移機械とパラメータ図を入力とすることで HotSpot であるアプリケーションの論理を自動生成することを考えた．これについて，第 6 章に示したような簡単な例であればアプリケーションの論理を自動生成できることを確認した．

[6] では，ステートマシン図の状態と遷移にネイティブな実現コードを埋め込むことで，ステートマシン図からプログラムの実現コードの生成を行っている．[6] の生成系への入力であるステートマシン図を図 9 に示す．

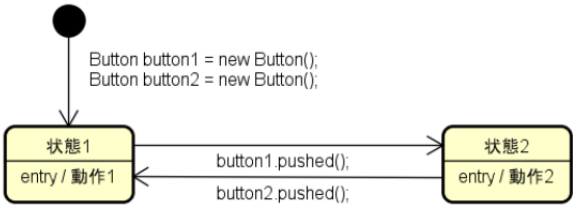

図 9　先行研究 [6] におけるステートマシン図

この手法では HotSpot の生成も行っているが，ステートマシン図の状態と遷移に記述する実現コードは手書きする必要がある．一方，我々は標準化したコンポーネントを予め用意し，状態遷移機械とパラメータ図からプログラムの実現コードの生成を行っている．よって [6] と比較すると，手書きする実現コードの記述量は少なくなると考える．

本研究はドメイン内に共通する振舞いを分類することで自動生成の実現を目指しているので，特定のドメインに限定された話ではない．ショッピングサイト以外のドメインにおいても，ドメイン内に共通する振舞いを分類し階層化することで自動生成できると考える．しかし，どのドメインにも共通しない独自な機能を実装したい場合は手書きで記述する必要がある．本研究ではショッピングサイトを事例として取り上げた．ショッピングサイトの特徴としては商品の選択や購入処理が考えられるので，これらの振舞いを分類することで自動生成の可能性を考察した．

7　おわりに

スマートデバイスの急速な普及に伴い，その上で稼働するアプリケーションソフトウェアの開発の省力化が求められている．インタラクティブシステムの多くは，MVC アーキテクチャを用いて開発されている．View や Controller の自動生成に関する研究は行われているが，Model の自動生成は本質的に難しい．

本研究では CSA/I-Sys におけるアプリケーションの論理の自動生成を実現するために，状態遷移機械の詳細化を行った．状態遷移機械を詳細化するにあたり，ドメイン内に共通する振舞いを分類し抽象化する．これをメタ状態遷移機械として記述した．複合状態は複数のベース状態遷移機械を持つことから状態遷移機械を階層化した．

アプリケーションフレームワークの設計と標準化されたコンポーネントが定義されている前提のもと，状態遷移機械とパラメータ図を生成系への入力とする．これにより，プログラミングをすることなく BusinessLogic の生成を行うことができるので，アプリケーション開発の省力化に繋がると考える．

今後の課題としては実際に生成系を設計しアプリケーションの論理が生成できることを確認することが挙げられる．

参考文献

[1] A. ESAKA, M. NORO, and A. SAWADA, “Design of Common Software Architecture as Base for Application Generator and Meta-Generator for Interactive Systems,” *IEEE*, vol. 2, pp. 323-328, 2017.

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

[3] S. Lazetic et al, “A Generator of MVC-based Web Applications.” *World of Comp. Sci. & Info. Tech. J.* , vol. 2, no. 4, 2012, pp. 147-156.

[4] 河村美嗣，浅見可津志，“UML を入力とするソースコード自動生成ツールの開発，”情報処理学会全国大会講演論文集，vol.72，pp.337-338，Mar. 2010.

[5] 京谷和明，伊藤恵，“コード生成を用いたフレームワーク向け Web アプリケーション開発支援ツールの作成，”日本ソフトウェア科学会大会論文集，vol.31，pp.155-161，Nov. 2014.

[6] 平野雄一，伊藤恵，“ステートマシン図を用いたプログラムの自動生成支援，”社団法人情報処理学会研究報告，vol.119，pp.93-100，Nov. 2005.