状態機械を用いるフレームワーク記述言語FwMLの設計とMVCアプリケーションへの適用

中鉢欣秀[†,††] 岡田健[†††] 大岩元[†]

ソフトウェアの実行時状態を XML で表現し，MVC 構造におけるコントローラを状態機械で記述することにより，再利用性のあるコントローラ部品を作成するための言語 Framework Markup Language（FwML）および実行環境 FwML Runtime Engine（FwML-RE）を開発した．本技術を用いて作成したコントローラ部品はコンポーネント・ベース開発におけるアプリケーション・フレームワークとしての性質を備えるため，本提案全体を FwML Meta-Framework（FwML-MF）と呼ぶ．本稿では，FwML-MF の設計と実装について述べる．また，2 つのサンプル・アプリケーションから共用されるコントローラ部品が FwML-MF を用いて作成可能であり，アプリケーション・フレームワークとなることを示す．これをふまえ提案技術の評価を行い，コントローラ部品の再利用性，CASE ツールとの連携，開発支援機能等において開発者に有用な技術であることを述べる．

Design of Framework Markup Language with State Machines and Its Implementation for MVC Applications

Yoshihide Chubachi,[†,††] Ken Okada[†††] and Hajime Ohiwa[†]

We propose a new language to develop reusable controller components named the Framework Markup Language (FwML) and its runtime environment named the FwML Runtime Engine (FwML-RE). The runtime status of the MVC structured software is represented as XML data structure and the controller components of the software are described as state machines. Since those reusable controller components have a feature of the framework in the context of the component based development, the whole of our proposal can be called the FwML Meta-Framework (FwML-MF). In this paper, we describe the design and implementation of the FwML-MF. Also we explain that two sample applications can reuse one set of controller components written in the FwML. This example shows that these components work as the application framework with the FwML-RE. We will evaluate this result and mention the benefits of our proposal from the viewpoints of the reusability of the controller components, the combinations of CASE tools, the supporting functions for the developers.

1. はじめに

1.1 MVC アプリケーション開発の課題

MVC 構造は GUI を備えたソフトウェアの構築に利用される一般的なデザインパターンである．MVC 構造のコントローラは，ビューとモデルとを仲介し，ソフトウェア全体の振舞いを規定する役割を担う[1)]．複雑な GUI アプリケーションを開発する開発者は，ソフトウェア全体の状態変化を適切に制御し，求めるユーザ操作が実現されるようコントローラを実装する必要がある．

† 慶應義塾大学環境情報学部
Faculty of Environmental Information, Keio University
†† 合資会社ニューメリック
Numeric & Co., Ltd.
††† 慶應義塾大学大学院政策・メディア研究科
Graduate School of Media and Governance, Keio University

コントローラはソフトウェアの動的側面の中核を担うものであるから，モデルにおけるデータ構造，ビューにおけるビュー・コンポーネントの配置といった静的側面の実装よりも難しい作業である．すなわち，開発者はユーザが起こしうる様々な操作に応じて，設計したとおりにアプリケーション全体が動作するように注意を払わなくてはならない．

このように，コントローラの開発は MVC 構造を持つソフトウェアの開発者にとって負担の大きい作業であるため，コントローラ部分の開発における作業の効率化が求められる．

1.2 オブジェクト指向開発における課題

前節で述べた問題に対し，一度作成したコントローラの再利用が容易になれば，開発者の負担が削減できることになる．しかしながら，従来のオブジェクト指向技術における基本的な MVC パターンでは MVC 間の結合度が高くなるという問題点が指摘されている[2)]．

したがって，オブジェクト指向設計技術を用いたとしても，ビューからコントローラへのインタフェイス，およびコントローラからモデルへのインタフェイスが設計ごとに異なれば，コントローラの再利用は非常に困難になる．

また，一般的にUMLを用いてコントローラの動的側面を記述する場合，シーケンス図またはコラボレーション図でMVC間のメッセージ・フローを記述する．しかしながら，これらの図では条件分岐を含んだ総称的な記述をすると図が煩雑になるという欠点がある．UMLでは状態図（ステート図）を用いることでソフトウェア全体の動的振舞いを協調動作する複数の状態機械として構造化した記述ができる．したがって，コントローラは状態図を用いて記述できるが，従来のオブジェクト指向言語には協調動作する状態機械を記述するための言語機能が備わっておらず，設計を実装に直截的に反映できないという問題がある．

1.3 FwML メタ・フレームワーク

本研究では，前節の課題点をふまえ，①アプリケーションの実行時状態の表現にXMLを用いることによる，再利用可能なコントローラの実現，②コントローラが担うアプリケーションの実行時状態の変化を状態機械で制御するための専用言語の開発，の2つの観点から解決法を提案する．

ここで，MVC構造におけるコントローラが再利用可能であるということは，コントローラそのものがコンポーネント・ベースの開発手法[3),4)]でいうところのアプリケーション・フレームワークとしての性質を備えることになる．すなわち，開発者はアプリケーションの目的に応じてビューおよびモデル・コンポーネントを作成し，すでに作成されてるコントローラに接続（プラグ・イン）することでアプリケーション・コンポーネントを完成させることができるようになる．

本研究では，MVC構造のコントローラ部分を状態機械で記述できる言語Framework Markup Language（以下，FwML）とその実行環境であるFwML Runtime Engine（以下，FwML-RE），および，実行時状態をXMLで表現するための仕様であるFwML State Model（以下，FwML-SM）を提案する．これらによって実現されるコントローラはフレームワークとなることから，本提案全体はメタ・フレームワークというのが適切であり，これをFwML Meta-Framework（以下，FwML-MF）と呼ぶ．

本稿では，FwMLおよびFwML-REの設計と実装，ならびにFwML-SMの仕様について述べ，FwML-MFの全体構成を説明する．これをふまえ，FwML-MFを用いてサンプル・アプリケーションのためのコントローラ部品が作成可能であり，かつ，再利用可能なフレームワークであることを示す．サンプルはGUIを備えたMVC構造を持つアプリケーション・コンポーネントである．以下，2章ではFwML-MFの設計について述べ，3章でFwML-MFの使用例を示す．4章でFwMLで再利用可能なコンポーネント部品，すなわち，アプリケーション・フレームワークを実現することの有用性を評価し，5章で考察とまとめを行う．

2. FwML-MF の設計

2.1 実装に用いる技術

初めに，FwML-MFで利用する技術について述べる．状態機械はHarel[5)]が提案し，UML仕様[6)]で拡張された状態図の仕様を用いている．この状態図には，パラメータをともなうイベント，遷移時の活動やガード条件，シグナル送信による他の状態機械との協調を記述できるため，本稿で示すようにアプリケーション・フレームワークの振舞い定義に利用できる．

FwML-MF全体は，XML技術とJava技術により実現されている．状態機械の活動およびガード条件の記述のために，XPath[7)]技術を利用する．XPath式はXMLを表現する標準データ構造であるDOM[8)]のノードを参照するための仕様であり，より汎用的なXMLデータの検索機能を実現するXQueryもXPath規格を土台として実現されている．

FwML-MFにおけるXMLデータの文書型定義はRELAX NG[9)]を用い，Java言語との連携にはRelaxer[10)]を用いている．FwMLの実装にあたっては，言語の文書型をRELAX NGで定め，Relaxerを用いてJava言語のクラス構造に変換した．FwMLで記述された状態機械の実行には，XMLファイルであるFwMLをパースして生成する構文木をトラバースしながらFwML-REが評価し，実行時状態をDOMとXPath式を用いて操作する．

2.2 全体構成

FwML-MFの主な特徴は，以下の3点である．

(1) 協調動作する状態機械からなるコントローラ部品のFwMLによる記述
(2) FwML-REによるGUIアプリケーション・フレームワークの実現
(3) FwML-SM仕様に基づくソフトウェア実行時状態のXML形式による表現

FwML-MFには概念的に2種類の利用者，すなわち，アプリケーション・フレームワークの開発者である「フレームワーク開発者」および，アプリケーショ

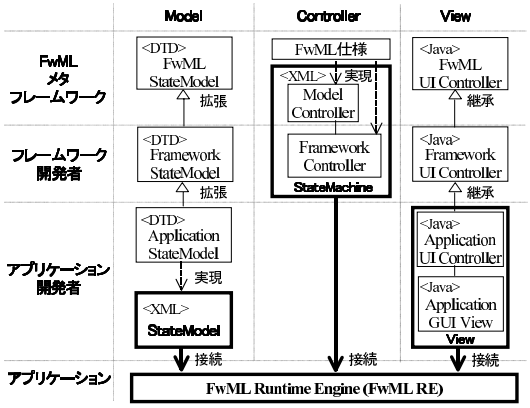

図 1 FwML メタ・フレームワークの構成
Fig. 1 The overview of FwML meta-framework.

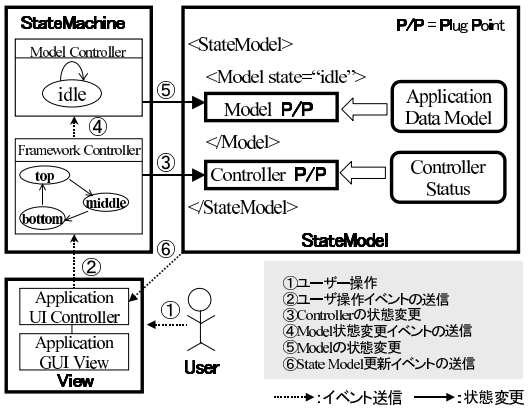

図 2 状態機械間のイベント伝達による MVC 構造の実現
Fig. 2 The implementation of MVC structure with events passing.

ン・コンポーネントの開発者である「アプリケーション開発者」が存在する．

FwML-MF の全体構成は**図 1** のようになる．縦軸は，メタ・フレームワークが提供する部分，フレームワーク開発者が開発する部分，アプリケーション開発者が開発する部分，および，実際のアプリケーションの実行環境部分を示す．横軸は，MVC 構造におけるモデル，コントローラ，ビューの各構成要素との対応を示す．図中，黒太線で囲んだ部分が最終的にアプリケーションを構成するコンポーネントである．以下，これら 4 つのコンポーネントについて説明する．

「StateModel」は，FwML-SM 仕様に基づいた XML で表現されたデータであり，実行時状態を格納する．StateModel は，FwML-MF で定められている基本データ形式「FwML StateModel」を基にして，フレームワーク開発者がアプリケーション・フレームワークで使用する大枠のデータ構造「Framework StateModel」に拡張する．これをさらにアプリケーション開発者が拡張して定める「Application StateModel」のインスタンスが StateModel となる．

「StateMachine」は，FwML 仕様に基づいた XML で記述される状態機械の集合であり，コントローラ部品である．あらかじめ提供されている「Model Controller」状態機械があり，これにフレームワーク開発者が任意個の「Framework Controller」状態機械を開発して追加することで構成される．これらは協調動作し，StateModel の状態を変更する．

「View」は，Java 言語で記述されたソフトウェア部品であり，アプリケーションの GUI 部に含まれる．あらかじめ提供されている「FwML UI Controller」を継承して，フレームワーク開発者が「Framework UI Controller」を作成する．これをさらに継承して，アプリケーション開発者が「Application UI Controller」を開発する．これは，アプリケーションのユーザ操作に従い，StateMachine に対しイベントを送信し，かつ，StateModel の更新通知を受け取る機能を実装したものである．Appplication UI Controller と実際のユーザインタフェースのための，ボタンやテキスト・フィールドといった部品から構成される「Application GUI View」とで View 全体が実現される．

「FwML Runtime Engine（FwML-RE）」は Java 言語で実装された FwML インタプリタである．図に示すとおり，3 つのプラグ・ポイントを備え，アプリケーション・フレームワークを実現する中核となっている．すなわち，XML 形式の StateMachine ファイルと，アプリケーションの初期状態が格納されている StateModel ファイルを読み込み，Java 言語で作成された View クラスを接続することでアプリケーション全体を構成する．

2.3 FwML-MF による MVC 構造の実現

FwML-MF を用いて実装したソフトウェア全体が，MVC 構造に基づくイベント伝達を実現し，実際のアプリケーションとして動作する仕組みを**図 2** に示す．図は，図 1 で黒太線で囲んだ構成要素のうち，StateModel，StateMachine，および，View の関係図であり，FwML-RE によって全体の機構が実現される．

以下，FwML-RE が MVC 構造を実現する仕組みを図中の①〜⑥の順に述べる．ユーザからの操作を View が受信する（①）．View は，Application UI Controller を経由して StateMachine にイベントを送信する（②）．イベントは StateMachine の Framework Controller 状態機械が受信し，Controller P/P☆に配置されたコントローラの状態を更新する（③）．また，

Framework Controller 状態機械は，FwML-MF が用意した Model Controller 状態機械にイベントを送信することで協調動作する（④）．これを受けた Model Controller 状態機械は，イベント内容に従い，Model P/P に配置されたモデルの状態を変更する（⑤）．最後に，FwML-RE は StateModel を監視する Application UI Controller に対し，更新イベントを送信する（⑥）．

ここで，FwML-MF におけるイベントの仕様は，UML の状態図に関する仕様と等しく，「イベント名」と，引数に相当する任意個の「イベント・パラメータ」からなる．StateMachine 内における各状態機械はこのイベントを受信することで状態遷移する．このとき，状態の遷移時に行う活動が記述されていれば，StateModel の状態が操作される．

StateModel 内における「Controller P/P」には，フレームワーク開発者がデータ構造を定義した「Controller Status」が置かれる．これは，コントローラ部品の状態を示した XML データである．「Model P/P」にはアプリケーション開発者がデータ構造を定義した「Application Data Model」が置かれる．これは，アプリケーションで使用する XML データである．

最後に，Controller Status と Application Data Model の目的の違いを述べる．前者がソフトウェアの実行時に内部的に使用する状態変数，モード，フラグといった情報を表現するのに対し，後者は，アプリケーションでユーザが操作する実際のデータであり，ワープロであれば編集中の文書に相当する．したがって，Application Data Model をシリアライズすることにより，ユーザがソフトウェアを使用して作成したデータを XML 形式でファイルに保存できる．

2.4 Framework Controller

図 3 にフレームワーク開発者が開発する Framework Controller の振舞いを示す．前掲の図 2 には Framework Controller を 1 つだけ示したが，実際には本図のように複数の Framework Controller が存在し，それらが協調動作する．

Framework Controller は図の黒矢印で示したように StateModel の Controller P/P 部にある XML 要素に接続する．状態機械が設定されているタグを UML の用語にならい「コンテキスト要素」と呼ぶ．コンテキスト要素の位置を指定するためには XPath 形式を用いる．XPath 形式の位置指定は，

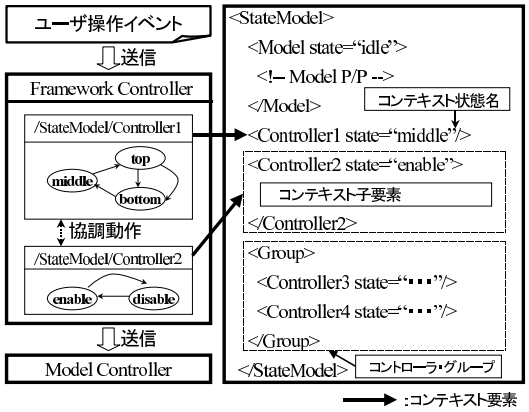

図 3 Framework Controller 状態機械の振舞い（図 2 の②〜④）

Fig. 3 The behavior of Framework Controller state machine (See Fig. 2 ②〜④).

XML データの要素を Unix のディレクトリ指定で用いるパスの形式と同じように指定する．図の 2 つの状態機械は，「/StateModel/Controller1」および「/StateModel/Controller2」の XPath で示されるコンテキスト要素に対して接続されている．

コンテキスト要素には，state 属性があり，これに状態機械における現在の状態名が格納されている．これを「コンテキスト状態名」と呼ぶ．状態機械の状態遷移にともないコンテキスト状態名が変化する．また，コンテキスト要素には，フレームワークで定義する任意の子要素を持たせることができる．これを「コンテキスト子要素」と呼ぶ．Framework Controller は，遷移時の活動の際に自分のコンテキスト要素のコンテキスト子要素を操作する．また，StateModel に Framework Controller 状態機械を配置する際，任意のタグでグルーピングできる．これを「コントローラ・グループ」と呼び，協調して動作する複数の Framework Controller をアプリケーション開発者に分かりやすくするためのものである．図中，/StateModel/Group の下にあるタグがグループ化されているコンテキスト要素である．

2.5 Model Controller

FwML-MF には，「Model Controller」と呼ばれる Controller が標準で用意されている．Model Controller は，Framework Controller から送信されたイベントを受信し，Application P/P における Application Data Model を操作するための状態機械である．XML ツリーの任意のノードを XPath 式で指定し，データの追加，挿入，削除，置換の各 DOM 操作を要求するイベントを受理する．

☆ P/P=Plug Point

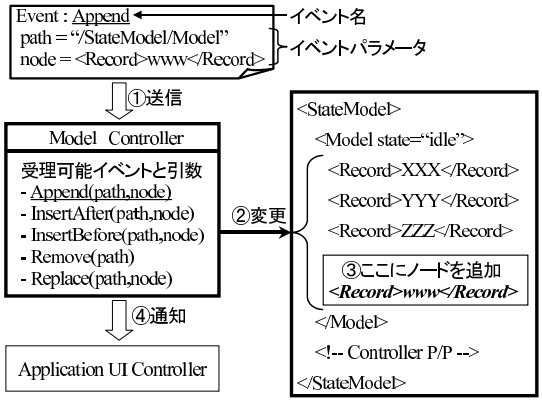

図 4 Model Controller 状態機械の振舞い（図 2 の④〜⑥）
Fig. 4 The behavior of Model Controller state machine (See Fig. 2 ④〜⑥).

以下，**図 4** において，データの追加イベント，すなわち，イベント名が Append であり，イベント・パラメータにデータを追加する XML 要素への XPath 式，および，追加するデータを含んだイベントが Model Controller に送信されたときの振舞いを示す．

図中の①〜④の順に，Append イベントを Model Controller が受理する（①）．イベントに従い，Model Controller が Application Data Model を操作する（②）．その結果として `<Record>` タグが追加される（③）．State Model が変更されたので Application UI Controller に変更を通知する（④）．

Model Controller が受理するイベントと，受理した際に行われる活動を以下にまとめる．今，イベントパラメータ内の *path* が XPath 式で記した任意のタグへの参照，*node* は DOM で定義されている XML ノード，すなわち Element（タグ），または，Text（テキスト）であるとき，次のイベントを受理する．

Append(*path*,*node*) *path* で示されるすべての要素の子に *node* を追加する．

InsertAfter(*path*,*node*) *path* で示されるすべての要素の下に *node* を挿入する．

InsertBefore(*path*,*node*) *path* で示されるすべての要素の上に *node* を挿入する．

Remove(*path*) *path* で示されるすべての要素を削除する．

Replace(*path*,*node*) *path* で示されるすべての要素を *node* に置換する．

Model Controller のコンテキスト要素は "/StateModel/Model" となり，コンテキスト状態名はつねに "idle" である．Model Controller にはただ 1 つの状態「idle」があり，上記イベントを受理した際であっても状態遷移は行わない．しかし，他のコントローラ部品と同様，Model Controller を状態機械として実装することによって，FwML-MF が提供するイベント・メカニズムを通じて他のコントローラ部品と協調して動作できるように設計されている．

2.6 Application UI Controller および Application GUI View

FwML-MF は，Java 言語の GUI コンポーネントと連携する．このためには，FwML-RE のイベント・メカニズムと Java コンポーネントを連結しなくてはならない．FwML には，このための Java クラスとして FwML UI Controller を提供する．FwML UI Controller は，FwML ではなく Java 言語で実装されるが，FwML-MF 内では StateMachine の Framework Controller および Model Controller と同様の状態機械である．したがって，FwML-RE のイベント・メカニズムにより以下の 2 つの機能が実現できる．

(1) GUI 操作を FwML-RE を経由して StateMachine にイベント送信する．

(2) StateModel の特定部分の変更を監視し，変更があった場合 FwML-RE からイベント通知を受ける．

このことにより，MVC 構造の最も基本的なメカニズムであるオブザーバ機構が得られる．このため，FwML UI Controller には，StateMachine へのイベント送信，および，更新の監視対象とする StateModel の範囲を XPath で設定する機能がある．

フレームワーク開発者は，FwML UI Controller を継承して Framework UI Controller を作成する．これは，アプリケーション・フレームワークであるコントローラ群と，GUI とのインタフェースの共通項をまとめた抽象クラスになる．アプリケーション開発者は，Framework UI Controller を継承して Application UI Controller を実装することで，アプリケーションの実際の GUI になる Application GUI View と連結させることができる．

たとえば，前節の図 4 の例では，/StateModel/Record を監視対象として設定されている Application UI Controller に最終的に変更通知が送られる．したがって，Application UI Controller は，Application GUI View を操作し，ユーザにデータの更新を伝えることができる．なお，Application GUI View の実装には，Java 言語が提供する Swing 等の GUI フレームワークが利用可能である．

2.7 FwML

FwML は，XML 形式のマークアップ言語であり，UML で定義される状態図の基本的な機構に対し，拡

```
<!ELEMENT StateMachine (State+)>
<!ATTLIST StateMachine context CDATA #REQUIRED>
<!ELEMENT State (Transition*)>
<!ATTLIST State name CDATA #REQUIRED>
<!ELEMENT Transition (Guard?, Action?)>
<!ATTLIST Transition event CDATA #REQUIRED>
<!ATTLIST Transition target CDATA #REQUIRED>
<!ELEMENT Guard (#PCDATA)>
<!ELEMENT Action (Expression*)>
```

図 5 FwML の DTD 定義（状態遷移部）
Fig. 5 The document type definition of FwML (State machine part).

張を加えたものである．FwML で利用できる状態図の機構は，標準仕様から内部遷移（entry，exit 等）に関するものを省略し，外部遷移を引き起こすイベントは非同期のシグナル・イベントのみとする，等の簡略化を行ったものである．状態機械の記述のための FwML の DTD 定義を**図 5** に示す．FwML は FwML-RE によって評価され，実行される．評価結果は，FwML State Model におけるコンテキスト状態名の変化，または，コンテキスト子要素の変化となる．

コンテキスト状態名の変化は，XPath 式で記述されるガード条件を有する状態遷移の結果を反映する．ガード条件は，真偽値を返す XPath 式を記述し，評価の結果が真となる遷移が記述されていた場合，target 属性に記述する状態名に遷移する．状態遷移の結果は状態名の変化として FwML-RE が自動的に StateModel 内のコンテキスト状態名を書き換える．

コンテキスト子要素の変化は，状態遷移時に FwML-RE が実行する活動の記述に従う．記述には，DOM が提供する XML データの操作機能を用いたコンテキスト子要素の操作，および，XPath が提供する XML データに対する演算が含まれる．

FwML は XML であるから，UML の XML 表現である XMI [11] を変換することで FwML の雛形を得ることができる．したがって，ユーザは，FwML を直接記述するのではなく，CASE ツールを利用してコントローラ部品の設計を行い，XMI 形式で出力したものを利用することで開発の手助けとすることができる．

なお，FwML には XML 形式の他に短縮表現形式がある．FwML をコンパクトに記述するためのものであり，これは UML の状態図に活動の内容を記述するような場合にも用いることができる．短縮表現による FwML の言語仕様を付録 A.1 に記載してある．

3. FwML-MF による実装例

本章では，FwML の評価にあたり実際にフレームワークを実装し，これを用いて異なる 2 つのアプリケーションが実装できることを具体例を用いて示す．

3.1 アプリケーション仕様

作成するアプリケーションの仕様には，竹田ら[12] で用いられている「住所録」と「金銭出納帳」の仕様を一部改変して使用する．竹田らはソフトウェア技術者育成の立場から，両者の共通部品の再利用過程を C 言語を用いた漸進的開発プロセスの例として学習者に提示している．

本評価においては，この延長として FwML-MF を用い，初めに共通部品をアプリケーション・フレームワークとして構築する．次に，個別の仕様を実現するアプリケーション・コンポーネントを実装する．実装されたアプリケーションは GUI を備え MVC 構造を持つ XML アプリケーションとなる．

仕様 1：住所録

住所録は，名前（JName），フリガナ（KName），郵便番号（Zip），住所（Address），電話番号（TelNo）からなるレコードのリストを操作するアプリケーションである．システムはレコードをインデックス番号で特定する．ユーザ操作の対象となるレコードを注目レコードと呼び，これを示す番号をカーソル（Cursor）という．

ユーザはリストに対して新規レコードの追加（append），編集（edit），削除（delete），カーソルを 1 つ前に移動する（previous），および 1 つ後に移動する（next）操作を行うことができる．なお新規レコードはリストの末尾に追加され，同時にカーソルも末尾に移動する．

カーソルが先頭にあるときの previous 操作等，実行できない操作をユーザに許してはならない．

仕様 2：金銭出納帳

金銭出納帳は，日付（Date），品目（Item），分類（Category），収入金額（Income），支出金額（Outgoing），残高（Amount）からなるレコードのリストを操作するアプリケーションである．ユーザはリストに対して住所録と同様の操作を行うことができる．操作の制限についても同様である．

新規レコードの追加の際に，残高はシステムが自動的に計算するものとする．

3.2 フレームワークの実装

仕様に基づき，両者の共通部分をアプリケーション・フレームワークとして実装する．このために，2.2 節の図 1 における Framework StateModel，Framework Controller，Framework UI Controller を作成する．

3.2.1 Framework StateModel の実装

2 つの仕様に共通するデータ構造を抽出し，FwML

```
<!ELEMENT StateModel (Model,Cursor)>
<!ELEMENT Model (Record)*>
<!ATTRIBUTE Model state (#PCDATA) #REQUIRED>
<!ELEMENT Record ANY>
<!ELEMENT Cursor (#PCDATA)>
<!ATTRIBUTE Cursor state (#PCDATA) #REQUIRED>
```

図 6 Framework StateModel の定義
Fig. 6 Define of Framework StateModel.

表 1 Cursor 状態機械がとりうる状態
Table 1 State of Cursor State Machine.

empty	レコード総数が 0 のとき
single	レコード総数が 1 のとき
top	カーソルが先頭にあるとき
middle	カーソルが中間にあるとき
bottom	カーソルが末尾にあるとき

表 2 Cursor 状態機械の状態遷移表
Table 2 State transition table of Cursor State Machine.

	empty	single	top	middle	bottom
append	single	bottom	bottom	bottom	bottom
edit	-	single	top	middle	bottom
delete	-	empty	[count($M)=2] single [else] top	[cursor=count($M)-1] bottom [else] middle	[count($M)=2] single [else] bottom
previous	-	-	-	[cursor=2] top [else] middle	[count($M)=2] top [else] middle
next	-	-	[count($M)=2] bottom [else] middle	[cursor=count($M)-1] bottom [else] middle	-

StateModel を拡張して Framework StateModel を定義する．DTD で定義した Framework StateModel を**図 6** に示す．図に示されるとおり，2 つの仕様に共通するデータである，任意個の Record 要素と 1 つの Cursor 要素が定義されている．

ここでは，Record 要素をフレームワークとアプリケーションのプラグ・ポイントとするため，Record 要素のデータ型を可変（ANY）型として定義する．また，カーソルは Cursor 要素として定義し，注目レコードのインデックス番号を子要素に文字列（#PCATA）型で保持する☆．加えて，Cursor 要素を次項で述べるように状態機械とするため，state 属性が設定する．

3.2.2 Framework Controller の実装

Framework Controller の状態機械を設計し，状態遷移時の活動を定義する．ここでは Cursor 要素を状態機械とし，ユーザ操作は Cursor 状態機械に対するイベントとして表現する．ユーザの操作は 2 つのアプリケーションに共通しており，3.1 節の仕様で示した 5 つのユーザ操作に対するイベント，すなわち append(r)，edit(r)，delete(r)，previous()，next() を受け取る状態機械を構成すればよい．イベント・パラメータ r にはアプリケーション・コンポーネントで生成したレコードが渡されるものとする．

ユーザ操作の制限は，Cursor 状態機械の状態遷移において各操作が受理可能であるか判定することで行う．このために考慮すべき状態を，**表 1** の 5 つの状態として抽出した．

Cursor 状態機械の状態遷移表を**表 2** に示す．また，状態図を付録 A.2.1 の**図 10** に示す．これに基づき，ユーザ操作の制限も規定される．たとえば Cursor 要素の値が 3 でレコード総数が 3 であれば bottom 状態であるから，ユーザの next 操作を許してはならない．

Framework Controller の状態遷移時に行う活動も定義する．2.5 節で述べたように，Cursor 状態機械は Model Controller にイベント送信することで，Application Data Model を操作することができる．付録 A.2.2 に，Framework Controller の FwML スクリプトによる定義の一部を示す．また，付録 A.2.3 には，Model Controller の定義を示してあるので，あわせて参照されたい．

3.2.3 Framework UI Controller の実装

GUI と状態機械を接続する機能の共通部品が，Framework UI Controller である．2 つの仕様で共通するのは，表 2 の横軸に示した 5 種類のイベントを Cursor 状態機械へ送信する処理である．フレームワーク開発者は FwML UI Controller を継承してイベント送信手続きを加えた Framework UI Controller を作成する．

3.3 アプリケーション・コンポーネントの実装

住所録と金銭出納帳のアプリケーション・コンポーネントを実装する．このため，2.2 節の図 1 における Application StateModel，Application UI Controller をそれぞれのアプリケーション用に作成する．

☆ 整数型でないのは，本稿で説明に用いた DTD では文字列型以外の型定義ができないためである．RELAX NG 仕様を用いれば厳密な型定義が可能である．この際，FwML-SM のプラグ・ポイントには同仕様の include 機能を利用することができる．

```
<!ELEMENT Record  (JName,KName,Address,Tel,Zip)>
<!ELEMENT JName   (#PCDATA)>
<!ELEMENT KName   (#PCDATA)>
<!ELEMENT Address (#PCDATA)>
<!ELEMENT Tel     (#PCDATA)>
<!ELEMENT Zip     (#PCDATA)>
```

図 7 住所録で扱う Record を定義する DTD
Fig. 7 Document type definition of Record element for the address book.

```
<!ELEMENT Record   (Date,Item,Category,
                    Income,Outgoing,Amount)>
<!ELEMENT Date     (#PCDATA)>
<!ELEMENT Item     (#PCDATA)>
<!ELEMENT Category (#PCDATA)>
<!ELEMENT Income   (#PCDATA)>
<!ELEMENT Outgoing (#PCDATA)>
<!ELEMENT Amount   (#PCDATA)>
```

図 8 金銭出納帳で扱う Record を定義する DTD
Fig. 8 Document type definition of Record element for the cashebook.

3.3.1 Application StateModel の実装

アプリケーション開発者はアプリケーションで扱うデータ形式を Application StateModel として定義する．住所録で扱うデータの定義を**図 7** に，金銭出納帳で扱うデータの定義を**図 8** に示す．

これらを 3.2.1 項で用意したプラグポイントに接続することで，StateModel の全体が定義される．**図 9** に住所録で使用する StateModel の例を示す．

3.3.2 Application UI Controller および Application GUI View の実装

アプリケーション開発者は，Application GUI View と Application UI Controller を実装し，View を作成する．本稿では，Application GUI View は Java 言語の Swing を用いて実装する．GUI からの操作とフレームワークとの接続は Application UI Controller を介して行われる．

たとえば，画面にユーザがカーソルを移動させるための「次へ」ボタンが配置されているとき，ユーザがこれを押下すると，Framework UI Controller を経由して Framework Controller の状態機械へ next イベントが送信される．状態機械は FwML の記述に従い状態遷移する．この遷移には，StateModel の変更操作が記述されているから，オブザーバとして登録してある Application UI Controller に通知される．その結果として，GUI が更新されるよう，Application UI View を実装する．

金銭出納帳の GUI も同様である．住所録と同様のインタフェースに加えて，仕様の非共通部分である残高の表示欄がある．アプリケーション・コンポーネントには，この残高の計算ロジックを組み込む必要があるが，これ以外の共通部分については，住所録で使用しているフレームワークを再利用できる．

```
<StateModel>
  <Model state="idle">
    <Record>
      <JName>岡田健</JName>
      <KName>オカダケン</KName>
      <Address>神奈川県藤沢市 XXXX</Address>
      <Tel>0466-XX-XXXX</Tel>
      <Zip>252-XXXX</Zip>
    </Record>
    <Record>
      <JName>青山希</JName>
      <KName>アオヤマノゾム</KName>
      <Address>神奈川県横浜市 XXXX</Address>
      <Tel>045-XX-XXXX</Tel>
      <Zip>245-XXXX</Zip>
    </Record>
  </Model>
  <Cursor state="top">1</Cursor>
</StateModel>
```

図 9 住所録の StateModel 例
Fig. 9 An example of StateModel of the address book.

3.3.3 開発支援機能

最後に，FwML-RE に備わっている開発支援機能について述べる．FwML-RE は状態遷移のログと実行時状態のダンプを出力する機能を提供する．今回のサンプルを実装した際に得られた出力例として，図 9 の状態から，ユーザがレコード削除操作を行った場合のログを付録 A.2.4 に，その結果変更された StateModel を A.2.5 に示す．

4. FwML-MF の評価

4.1 実装例から示された有用性

本稿では,「住所録」と「金銭出納帳」という異なるアプリケーションを FwML-MF によって実装できることを具体例で示している．住所録と金銭出納帳の共通部分は FwML で記述したコントローラ部品によってフレームワークとなり，それぞれのアプリケーション・コンポーネントから再利用されている．

フレームワーク化には，両者に共通するデータ構造の定義と，共通する振舞いの記述が必要である．実装例により，2.2 節で述べた協調動作する状態機械からなる StateMachine と，実行時状態を格納する StateModel によってアプリケーション・フレームワークが構築できることが示された．

両アプリケーション・コンポーネントはともに MVC 構造を備えている．2.3 節で述べたように，これを実現するためのメカニズムは FwML-MF が提供する．実装例から，このメカニズムを利用して GUI アプリケーションが実現できることが確認された．

2.4 節で述べた Framework Controller と，2.5 節で述べた Model Controller を用いることで，アプリケーション・フレームワークの実行時状態と，個別のアプリケーション・データの両方が StateModel に格納できることも示されている．2.6 節で述べた Application UI Controller および Application GUI View によって，アプリケーションの View が作成できることについても述べた．これらは，2.7 節で述べた言語仕様にしたがって記述した FwML および Java 言語で実現されている．

4.2 コントローラ部品の再利用性

本提案技術では，データ構造を XML 形式に統一しているから，コントローラ部品はモデルとビューが XML データで結合していることにのみ依存し，アプリケーション・データ・モデルの構造を関知する必要がない．これにより MVC 間の結合度の問題を回避している．

これは，本技術を用いて開発したコントローラ部品の再利用性が高いことを意味する．このことを，住所録と金銭出納帳という異なるユーザインタフェースとアプリケーション・データ・モデルを持ったアプリケーションが，共通のコントローラ部品，すなわち，アプリケーション・フレームワークによって実現されることで示した．

4.3 UML 対応 CASE ツールとの連携

本提案技術では，MVC 構造のコントローラの振舞いを状態機械で記述する．フレームワーク開発者が状態機械を設計するときには UML の状態図を用いることができる．このため，設計作業に UML 対応の CASE ツールが利用できる．また，UML 対応 CASE ツールの多くは，UML のデータ標準交換形式である XMI で UML モデルを出力できるため，これを変換して FwML を得ることができる．

これは，アプリケーション開発者が FwML を参照する場合にも同様である．XML 形式の記述では一覧性に欠けるが，XMI 形式に変換し CASE ツールを用いることで状態図を得ることができる．このことは，アプリケーション開発者がフレームワークの振舞いを正確に理解するのに役に立つ．

筆者らの経験によると，未知のアプリケーション・フレームワークを利用したアプリケーション開発で一番難しいのは動的モデルの理解である．ライブラリとして提供されるフレームワークの振舞いを正確に理解するためにソースコードを読むのは苦痛である．状態図によって可視化される FwML を用いた開発ではこのようなアプリケーション開発者の負担を軽減する．

4.4 開発支援機能

FwML-RE には開発支援機能が備えられていて，フレームワークおよびアプリケーション開発者が利用できる．状態の遷移，および，遷移時の活動のログは自動的に記録される．このため，フレームワーク開発者は，トレースのためのログ出力機能を自分で作成する必要がない．アプリケーション開発者はログに出力されるアプリケーション・フレームワークの動作を参考に，コンポーネントの開発を行うことができる．

FwML-RE は，任意の時点で実行時状態をダンプできる．これはフレームワーク開発者が作成した状態機械のバグを発見する際に有用である．特に，状態機械がイベントを受理できなくなった場合には，エラーになっている実行時状態を自動的にダンプする．フレームワーク開発者は，前述のログとあわせて，バグの発生原因を特定するのに利用できる．

また，本提案手法に基づくフレームワーク開発では，ダミーの GUI コンポーネントをスタブとして用意しなくても GUI アプリケーション・フレームワークが作成できる．フレームワーク開発者は，前述のログ出力および実行時状態のダンプ機能を用いてアプリケーション・フレームワークが実装できる．このことから，フレームワーク開発者の開発工数が削減できる．

4.5 従来のアプリケーション・フレームワークとの比較

一般に普及している MVC 構造を備えたアプリケーション・フレームワークは依存メカニズムのほかに，データ・モデルの永続化や UI 部品を提供する．このうち，モデルの永続化について，XML に対応していない従来のアプリケーション・フレームワークは，通常ベンダが提供する内部データ構造に基づくバイナリ形式で永続化されるため，他のアプリケーションとのデータの互換性に問題があった．

最新の Java 言語は XML 形式の永続化が可能となっているが，ベンダが提供するデータ構造を変更することは現時点では難しい．本提案手法では，XML 形式で自由にデータ構造を定義できるから，これを永続化させることで開発者が望むデータ構造を持つ XML データを得ることができる．

なお，本提案技術は従来のアプリケーション・フレームワークとの整合性も考慮されており，サンプル・ア

プリケーションでは FwML-MF が提供する FwML UI Controller を用いて，Java 言語の Swing フレームワークが提供する UI 部品との連携ができることを示した．

4.6 デザイン・パターンとの比較

文献 1)，13) に紹介されているデザイン・パターンのうち，MVC 構造に必要な Observer[Gof95] パターン以外にも，開発者は次のようデザイン・パターンを実装するためのプログラミングをする必要はない．

State[GoF95] は，オブジェクトの内部状態が変化したときに，オブジェクトの振舞いを変えるようにすることが目的のパターンである．本提案技術を用いずに，Java 言語でこのパターンを使用する場合，状態をすべて Java のクラスとし，状態管理の機構も実装しなくてはならない．本提案ではこの機構は提供されており，プログラミングの手間が削減される．

Mediator[GoF95] パターンは，オブジェクト群の相互作用をカプセル化するオブジェクトを定義することが目的のパターンである．この目的のためには，2.4 節で述べた Controller Group と，Mediator に相当する Framework Controller を作成することで対応できる．Snapshop[Grand98] は，FwML-RE が提供する実行時状態のダンプを用いて同じことが実現できる．

4.7 パフォーマンス

実行速度に関する定量的な評価は行っていないが，パフォーマンスについては改善の余地があると考えている．改善を検討すべき主なポイントとしては，1）XPath 式を用いた FwML-SM の操作，2）FwML-SM から View コンポーネントへの変更通知，の 2 点である．

1）に関しては，XPath を実現するために用いているライブラリである Xalan [14] の性能に依存している．Xalan は現在も改良が続けられており，パフォーマンスの改善もなされている．FwML-MF の次期バージョンでは新しいライブラリを導入することで対応したい．

2）に関しては，状態遷移時の活動，すなわち，FwML スクリプトの実行の結果による StateModel の変更点の抽出のため，事前の状態を一時的に記録し，事後の状態と比較しているが，この部分のアルゴリズムは改良の余地がある．FwML-SM の操作対象となったノードに対してマーキングを行う等し，変更箇所を効率良く抽出できるよう改良を行っていく予定である．

5. 考察とまとめ

MVC 構造を持つアプリケーション・ソフトウェアの実現のため，提案技術である FwML および FwML-RE からなるメタ・フレームワークを用いることで，再利用性のあるコントローラ部品を作成でき，これをコンポーネント・ベース開発におけるアプリケーション・フレームワークとすることができる．このことを，本稿では「住所録」と「金銭出納帳」という異なる仕様を持つアプリケーション・コンポーネントが実装できることで実証した．

提案技術では，MVC 構造におけるコントローラの振舞いを状態機械で記述することでアプリケーション・フレームワークを構築する．XML で実行時状態を表現するため，コントローラが依存するデータ構造と，モデルとビューが依存するアプリケーション固有のデータ構造とを分離できる．このため，再利用性の高いアプリケーション・フレームワークが構築できる．

状態機械を記述するための FwML も XML であるから，アプリケーション・フレームワークそのものが XML 形式となる．そのため，フレームワーク開発時には，XML 対応の CASE ツールを用いることができる．また，アプリケーション開発時には，FwML を参照することでアプリケーション・フレームワークの振舞いを正確に理解できる．

FwML-RE には実行時状態の出力および，状態遷移のログ出力機能が標準で備わっているから，開発およびデバッグ時に有用である．また，フレームワーク開発時に，GUI コンポーネントをスタブとして用意する必要もないから，開発工数を減らすことができる．本技術には State，Mediator，Snapshop，Observer の各デザイン・パターンが実装済みであるため，これらを利用するために新たにコードを追加する必要がない．

以上述べたように，本提案技術は MVC アプリケーションのためのアプリケーション・フレームワーク開発に有用である．今後，パフォーマンスの改良を行うとともに，サーバ・クライアント型のネットワーク・アプリケーションへの展開等を行う．また，最新バージョンは Web ページ [15] よりダウンロード可能である．

謝辞 本研究にあたり，青山希君，澤田千代子さんをはじめ，慶應義塾大学湘南藤沢キャンパスの CreW Project [16] の皆さんに感謝いたします．

付 録

A.1 FwML 言語仕様

FwML の文法のうち，遷移時の活動を記述する部分の詳細な言語仕様を述べる．以下，説明する FwML は，本文中図 5 の Expression 要素になる．実際の FwML は XML 形式であるが，紙面の都合上，構文を短縮表現で示す．これは，以下の形式である．

Exp(p1:string,p2:number):node-set

また，使用例については，次の形式で示す．

Exp(StringExp(“a”),NumberExp(1))

これを，XML 形式に変換すると次のようになる．

```
<Exp>
  <StringExp>a</StringExp>
  <NumberExp>1</NumberExp>
</Exp>
```

A.1.1 基本データ型

FwML で定義されている基本データ型を**表 3** に示す．ノード集合型，論理値型，数値型，文字列型は，XPath 式の評価結果を格納できるよう，XPath 仕様の基本データ型と互換性がある．ノード型は DOM 仕様で定義されているものと等しい．

ただし，DOM の属性ノード（Attr）はコンテクスト状態名の格納に用いるため，FwML からは DOM ノードの Element 型および Text 型のみが操作できる．コンテキスト状態名の変更は，FwML で記述される状態遷移に従い，FwML-RE が自動的に行う．

また，配列型は，イベント・パラメータを格納するために用いられる．配列要素には，前述のすべての型の値が格納可能である．

A.1.2 文字列の扱い

FwML における文字列定数は，DOM 仕様における Text 型のインスタンスとして表現される．すなわち，XML 表現である FwML のタグで囲まれた文字列は文字列定数である．また，文字列を返す式 Str() が定義されているとき，その評価結果を文字列定数に結合して使用したい場合がある．この際，

```
<Exp>te<Str/>xt</Exp>
```

では，式 Exp() の第 1 引数に文字列定数 “te”，第 2 引数に式 Str() の評価結果，第 3 引数に文字列 “xt” が格納されるため，目的を達しない．そのため，これらをすべて結合した文字列を式 Exp() の第 1 引数に渡すには，任意個の文字列を結合する式 Cat() を使用して，

```
<Exp><Cat>te<Str/>xt</Cat></Exp>
```

とする必要がある．

なお，短縮表現においては，文字列定数はダブルクオートで囲むことで表す．また，Cat() 式に相当する中値演算子 “+” が使用できる．したがって，上の例を短縮表現で表すと，

Exp(“te”+str()+“xt”)

となる．FwML における文字列に関する短縮表現の定義を示す．

“text” 文字列定数「text」を示す．

“string1”+“string2”+… 文字列を連結する中置表現．

Cat(…):string 引数で与えられたすべての文字列を連結し，新しい文字列を生成する．

A.1.3 XPath 特殊変数と定数

FwML において，XPath 式を記述する際に使用できる特殊変数と定数が定義されている．特殊変数の値は実行時に FwML-RE が自動的に決定し，値を代入する．

$$ コントローラが接続されているコンテキスト要素を示す XPath 式（特殊変数）

$M FwML State Model の Model 要素を示す XPath 式（定数）

$C FwML State Model の StateModel 要素を示す XPath 式（定数）

たとえば，状態機械のコンテキスト要素の直下にあるテキストノードを取得するための XPath 式は “`$$/text()`” になる．

A.1.4 XPath 評価式

FwML から XPath 式を評価する式を定義する．文字列型の x が有効な XPath 式であるとき，以下の式が評価可能である．

Select(x:string):node-set x で示される XPath 式を評価し，ノード集合を返す．

Eval(x:string):string x で示される XPath 式を評価し，結果を文字列型に変換する．

XPath 式の評価の結果は，node-set または boolean，number，string である．このうち，***Select()*** 式は，node-set 型を得るのに使用する．また，***Eval()*** 式の結果は XPath の仕様にしたがってすべて文字列に変換される．

A.1.5 DOM ノード生成式

FwML 内から，新規に DOM ノードを生成するための式を次のとおり定義する．s が文字列であるとき，

Element(s:string):node s を値として持つ Element ノードを生成する．

Text(s:string):node s を値として持つ Text ノードを生成する．

表 3 FwML 基本データ型
Table 3 Data types of FwML script.

型	説明
node	ノード型
node-set	ノード集合型
boolean	論理値型
number	数値型
string	文字列型
array	配列型

A.1.6 DOM ノード操作式

引数 ns が操作対象となるノード集合，引数 n が追加または置換されるノードであるとき，以下の式が評価可能である．引数 n は子ノードも含む全体がディープ・コピーされ，各操作に引き渡される．

Append(ns:node-set,n:node):void ns で示されるノード集合の全要素の子として n で示されるノードを追加する．

InsertAfter(ns:node-set,n:node):void ns で示されるノード集合の全要素の下に n で示されるノードを挿入する．

InsertBefore(ns:node-set,n:node):void ns で示されるノード集合の全要素の上に n で示されるノードを挿入する．

Replace(ns:node-set,n:node):void ns で示されるノード集合の全要素を n で示されるノードに置換する．

Remove(ns:node-set):void ns で示されるノード集合の全要素を削除する．

A.1.7 シグナル・イベント式

FwML から，他の状態機械に対するシグナル・イベントを発行できる．r を送信先（reception）の状態機械を示す XPath 式，t を状態機械に送るイベント名を示す文字列，p をパラメータの配列としたとき，以下の式を評価すると，イベントが FwML-RE のイベント・キューにポストされる．

Event(r:string,t:string,p:array):void キューにシグナル・イベントをポストする．

A.1.8 パラメータ生成式

シグナル・イベントのパラメータ配列を生成する式を次のように定義する．

Param(...):array すべての引数を要素とするパラメータ配列を生成する．

A.1.9 パラメータ参照式

状態機械に与えられたイベントに含まれているパラメータを取得する式を次のとおり定義する．i がパラメータ番号であるとき，以下の式が評価可能である．なお，型変換に失敗した場合は実行時エラーとなる．

PNode(i:number):node i 番目のイベント・パラメータをノード型として取得する．

PSelect(i:number):node-set i 番目のイベント・パラメータで示される XPath 式を評価し，ノード集合を返す．

PString(i:number):string i 番目のイベントパラメータを文字列型として取得する．

A.1.10 オブザーバへの通知

コンテスト要素の状態名またはコンテキスト子要素が変わったことを，オブザーバに通知する．オブザーバイベントが発行される．

notfiy(p:array):Void オブザーバに更新イベントを送信する．

A.2 サンプル・フレームワーク

A.2.1 Cursor 状態機械の状態図

本文中で説明したサンプルにおける，Cursor 状態機械の状態遷移を UML の状態図で記述したものを，図 10 に示す．

A.2.2 Framework Controller

図 10 の top 状態における Cursor 状態機械の状態遷移と活動を記述する FwML を短縮表現で示す．

リスト中，StateMachine の context 属性はコンテキスト要素への XPath 式，State の name 属性は状態名，Transition の trigger 属性はイベント名，target 属性は遷移先の状態名である．Effect 部分が活動の記述であり，A.1 で述べた構文を用いている．

```
StateMachine: context="$C/Cursor"
  State: name="top"
    Transition: trigger="append" target="bottom"
      Effect:
        replace(select("$$/text()"),
          text(eval("count($M/Record)+1")))
        event("$M","append",
          params("$M/",pnode("1")))
        notify(params())
    Transition: trigger="edit" target="top"
      Effect:
        event("$M","replace",params(
          "$M/Record[number($C/Cursor/text())]",
          pnode("1")))
        notify(params())
    Transition: trigger="delete" target="single"
      Guard:test(eval("count($M/Record) = 2"))
      Effect:
        event("$M","remove",params(
         "$M/Record[number($C/Cursor/text())]"))
        notify(params())
    Transition: trigger="delete" target="top"
      Effect:
        event("$M","remove",params(
         "$M/Record[number($C/Cursor/text())]"))
        notify(params())
    Transition: trigger="next" target="bottom"
      Guard:test(eval("count($M/Record) = 2"))
      Effect:
        replace(select("$$/text()"),
          text(eval("number($$/text()) + 1")))
        notify(params())
    Transition: trigger="next" target="middle"
      Effect:
        replace(select("$$/text()"),
          text(eval("number($$/text()) + 1")))
        notify(params())
```

A.2.3 Model Controller

Model Controller の FwML による記述を示す．

```
StateMachine: context="$M"
  State: name="idle"
```

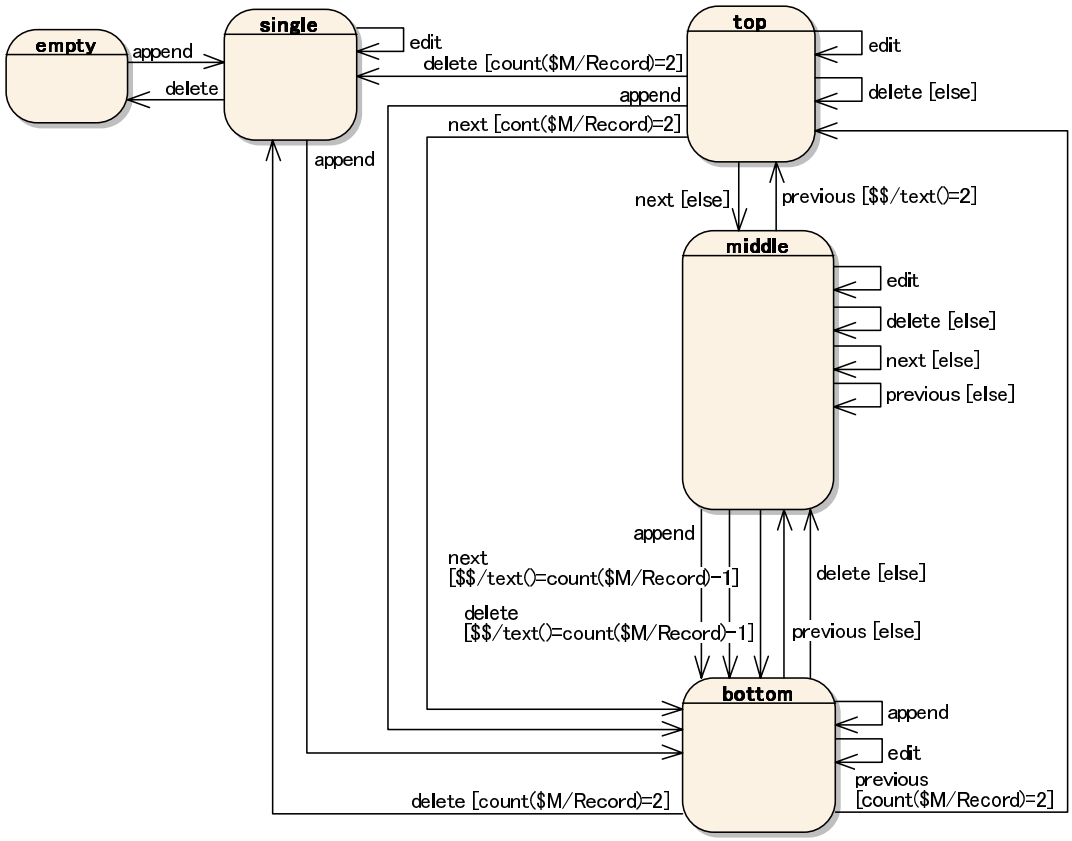

図 10 /StateModel/Cursor の状態図
Fig. 10 The state chart of /StateModel/Cursor.

```
Transition: trigger="append" target="idle"
  Effect:
    append(pselect("1"),pnode("2"))
    notify(params(pnode("2")))
Transition: trigger="insertAfter"
            target="idle"
  Effect:
    insertAfter(pselect("1"),pnode("2"))
    notify(params())
Transition: trigger="insertBefore"
            target="idle"
  Effect:
    insertBefore(pselect("1"),pnode("2"))
    notify(params())
Transition: trigger="remove" target="idle"
  Effect:
    remove(pselect("1"))
    notify(params(pnode("2")))
Transition: trigger="replace" target="idle"
  Effect:
    replace(pselect("1"),pnode("2"))
    notify(params(pnode("2")))
```

A.2.4 削除（delete）操作のログ出力

```
01:EVENT START
02:POST:/StateModel/Cursor.delete()
03:-->EVENT QUEUE:1
04:DISPATCH:/StateModel/Cursor.delete()
05:RECEIVE:/StateModel/Cursor
06:STATE:top
07:GUARD:
  <Guard><Test><Eval>
    count(/StateModel/Model/Record) = 2
  </Eval></Test></Guard>
08:TRANSITION:single
09:POST:/StateModel/Model.remove
          (/StateModel/Model/Record[1])
10:POST:ObservableEvent.update()
11:POST:ObservableEvent.update()
12:EFFECT:
  <Effect>
    <Event>
      <Target>/StateModel/Model</Target>
      <Trigger>remove</Trigger>
      <Params>
        <Cat>
          /StateModel/Model/Record[
          <Eval>
            number(/StateModel/Cursor/text())
          </Eval>
          ]
        </Cat>
      </Params>
    </Event>
    <Notify><Params/></Notify>
  </Effect>
13:<--EVENT QUEUE
14:-->EVENT QUEUE:3
15:DISPATCH:/StateModel/Model.remove
              (/StateModel/Model/Record[1])
16:RECEIVE:/StateModel/Model
17:STATE:idle
18:TRANSITION:idle
19:POST:ObservableEvent.update()
```

```
20:POST:ObservableEvent.update()
21:POST:ObservableEvent.update()
22:EFFECT:
  <Effect>
    <Remove>
      <PSelect>1</PSelect>
    </Remove>
    <Notify><Params/></Notify>
  </Effect>
23:CALLBACK:sample.addressbook.AddressTable
24:CALLBACK:sample.addressbook.Browser
25:<--EVENT QUEUE
26:-->EVENT QUEUE:3
27:CALLBACK:sample.addressbook.AddressBookMain
28:CALLBACK:sample.addressbook.AddressTable
29:CALLBACK:sample.addressbook.Browser
30:<--EVENT QUEUE
31:EVENT END
```

A.2.5 StateModel の変更結果

```
<StateModel>
 <Model state="idle">
  <Record>
   <JName>青山希</JName>
   <KName>アオヤマノゾム</KName>
   <Address>神奈川県横浜市 XXXX</Address>
   <TelNo>045-XX-XXXX</TelNo>
   <Zip>241-XXXX</Zip>
  </Record>
 </Model>
 <Cursor state="single">1</Cursor>
</StateModel>
```

参 考 文 献

1) Gamma, E., et al.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley (1995).
2) Johnson, R.E., et al.：パターンとフレームワーク，共立出版 (1999).
3) D'Souza, D.F. and Wills, A.C.: *Objects, Components and Frameworks with UML:Catalysis Approach*, Addison-Wesley (1999).
4) Stevens, P. and Pooley, R.: *Using UML: Software Engineering with Objects and Components*, Updated Edition, Pearson Education Limited (1999, 2000).
5) Harel, D.: On Visual Formalisms, *Comm. ACM*, Vol.31, No.5, pp.512–530 (1988).
6) OMG: UML 仕様書，アスキー出版 (2001).
7) W3C: XML Path Language (XPath) Version 1.0. http://www.w3c.org/TR/xpath
8) W3C: Document Object Model (DOM). http://www.w3.org/DOM/
9) Clark, J.: RELAX NG home page. http://relaxng.org/
10) Asami, T.: Relaxer. http://www.relaxer.org/
11) OMG: XML Metadata Interchange (XMI), version 1.2. http://www.omg.org/technology/documents/formal/xmi.htm
12) 竹田尚彦，大岩　元：プログラム開発体験に基づくソフトウェア技術者育成カリキュラム，情報処理学会論文誌，Vol.33, No.7, pp.944–954 (1992).
13) Grand, M.: *Patterns in Java: A catalog of reusable design patterns, illustrated with UML*, Volume 1, John Wiley & Sons (1998).
14) Apache XML Project: Xalan-Java. http://xml.apache.org/xalan-j/index.html
15) 中鉢欣秀：Framework Markup Language (FwML). http://www.crew.sfc.keio.ac.jp/~yc/fwml/
16) CreW Project: Web Site. http://www.crew.sfc.keio.ac.jp/

(平成 15 年 5 月 20 日受付)
(平成 15 年 7 月 8 日採録)

中鉢　欣秀（正会員）

1995 年慶應義塾大学環境情報学部卒業．2001 年同大学大学院政策メディア研究科博士課程単位取得退学．同年同大学非常勤講師．1997 年合資会社ニューメリック設立．オブジェクト指向関連技術の研究およびコンサルテーションに従事している．

岡田　　健（学生会員）

2001 年慶應義塾大学環境情報学部卒業．現在，同大学大学院政策メディア研究科博士課程在学中．日本語プログラミング言語，オブジェクト指向関連技術の研究に従事している．

大岩　　元（正会員）

1965 年東京大学理学部物理学科卒業．1971 年同大学大学院博士課程修了．理学博士．1992 年慶應義塾大学環境情報学部教授．情報教育学，ソフトウェア工学，認知工学の研究に従事している．