情報処理学会論文誌　Vol. 49　No. 7　2328–2337 (July 2008)

ソフトウェアの動的モデルに着目したラウンドトリップエンジニアリングの支援

今 関 雄 人[†1]　高 田 眞 吾[†2]　土 居 範 久[†3]

ラウンドトリップエンジニアリングは，モデリング段階とコーディング段階を往復しながらソフトウェア開発を行う手法である．ラウンドトリップエンジニアリングを支援するために，クラス図の変更をソースコードに自動的に反映させ，またソースコードの変更をクラス図に自動的に反映させるツールがある．しかし，従来のツールでは，クラス図などの静的側面のモデルを扱うのみで，動的側面のモデル，すなわちシーケンス図やステートチャートとソースコード間のラウンドトリップエンジニアリングを扱うことはできない．そこで，本研究では動的モデルとソースコード間のラウンドトリップエンジニアリングを支援するツールを提案する．本ツールは MVC パターンに基づいたアプリケーションを対象とし，シーケンス図，ステートチャートとソースコード間のラウンドトリップエンジニアリングを支援する．本ツールを使用することにより，動的モデルとソースコード間の効率的なラウンドトリップエンジニアリングが可能となる．

Supporting Round-trip Engineering Focusing on Dynamic Models

Yuto Imazeki,[†1] Shingo Takada[†2] and Norihisa Doi[†3]

Round-trip engineering is a software development method that iterates between the modeling phase and coding phase. Conventional tools support class diagrams and code for round-trip engineering, i.e., they automatically reflect changes in class diagrams to code and vice versa. Thus, although they support static models such as class diagram, they do not support dynamic models such as sequence diagrams and statecharts. In this paper, we propose a tool that supports round-trip engineering between dynamic models and source code. This tool targets MVC pattern-based applications, and supports sequence diagrams, statecharts, and source code. We use information from the controller of the MVC pattern to conduct the transformations between sequence diagrams, statecharts, and source code. This tool makes possible efficient round-trip engineering between dynamic models and source code.

1. はじめに

ラウンドトリップエンジニアリングは，フォワードエンジニアリングとリバースエンジニアリングを反復しながら開発を行う手法である[4),9),13)]．フォワードエンジニアリングでは，クラス図やシーケンス図などのモデルを基にしてソースコードを開発し，リバースエンジニアリングでは，ソースコードからモデルを生成する．すなわち，ラウンドトリップエンジニアリングとは，モデリング段階とコーディング段階を反復しながら開発を行う手法であるといえる．ラウンドトリップエンジニアリングでは，モデリング段階とコーディング段階の反復を通して，モデルとソースコードを徐々に詳細化していく．このため，ラウンドトリップエンジニアリングは機能の追加や要求の変更に強いという特徴があり，プロトタイピングなどの反復的なソフトウェア開発プロセスに対して効率的な手法であるといえる．

ラウンドトリップエンジニアリングを行う際には，モデルとソースコードの整合性を保つ必要がある．したがって，モデルの変更をソースコードに反映させる必要や，ソースコードの変更をモデルに反映させる必要がある．これらの反映を自動化し，ラウンドトリップエンジニアリングを支援するツールがある[1),5),12)]．しかしながら，これらのツールでは，静的側面のモデルであるクラス図とソースコード間のみをサポートしており，動的側面のモデル，すなわちシーケンス図やステートチャートとソースコード間を扱うことができないという問題がある．そこで，本研究では動的モデルとソースコード間のラウンドトリップエンジニアリングを支援するツールを提案する．本ツールは MVC パターンに基づいたアプリケーションを対象とし，シーケンス図，ステートチャートとソースコード間のラウンドトリップエンジニアリングを支援する．本ツールを使用することにより，動的モデルとソースコード間の効率的なラウンドトリップエンジニアリングが可能となる．

本論文の構成は次のとおりである．まず 2 章でラウンドトリップエンジニアリングの関連研究と，その問題点について述べる．3 章では提案するラウンドトリップエンジニアリング支援ツールについて述べ，4 章で提案ツールの適用例を示し，考察を行う．最後に，5 章

†1 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University
†2 慶應義塾大学理工学部
Faculty of Science and Technology, Keio University
†3 中央大学理工学部
Faculty of Science and Technology, Chuo University

© 2008 Information Processing Society of Japan

では結論を述べる．

2. 関 連 研 究

本章ではラウンドトリップエンジニアリングを支援する関連研究とその問題点について述べる．

2.1 ラウンドトリップエンジニアリング支援ツール

ラウンドトリップエンジニアリングを支援するツールとして，Together [1]，Rational Software Modeler [5]，EclipseUML [12] などがある．これらのツールは，クラス図の変更をソースコードに自動的に反映させる機能や，ソースコードの変更をクラス図に自動的に反映させる機能を持つ．この機能により，クラス図とソースコードの整合性を保持しながらモデリング段階とコーディング段階を往復することができ，クラス図とソースコードの効率的なラウンドトリップエンジニアリングが可能となる．しかしながら，これらのツールは静的側面のモデルであるクラス図を扱うのみで，動的側面のモデルについては扱うことができないという問題がある*1．

ソフトウェアの動的側面は主にシーケンス図とステートチャートを用いてモデリングを行う．そのため，動的モデルのラウンドトリップエンジニアリングを支援するには，モデルとソースコード間だけでなく，動的モデル間，すなわちシーケンス図とステートチャート間についても考える必要がある．次に，それぞれについての関連研究をあげる．

2.2 シーケンス図とステートチャートの相互変換

Hasegawa らは，MSC（Message Sequence Charts）[6] と UML ステートチャートの相互変換を行う手法を提案している [3]．MSC は bHSC と呼ばれるシーケンス図と hMSC と呼ばれるシーケンス図のつながりを表すモデルからなり，hMSC の情報を利用して変換を行う．

Hasegawa らの手法では，シーケンス図におけるメッセージの送信と受信をステートチャートにおける状態遷移に対応させる．この対応関係を使用し，まず各シーケンス図から各オブジェクトのステートチャートを生成する．その後 hMSC の情報を利用して，生成したステートチャートどうしを結合することにより，最終的なステートチャートを作成する．

また，ステートチャートの変更をシーケンス図に反映させることが可能である．ステートチャートに行った変更に対応する差分 bMSC を記述し，その差分を既存の MSC に統合することにより，反映を行う．しかし，差分 bMSC は開発者が記述する必要があり，また，ステートチャートの変更は遷移の追加のみしか反映させることができないという問題がある．さらに，シーケンス図がない状態では，ステートチャートからシーケンス図を生成することができない．

2.3 モデルからソースコードへの変換

動的側面のモデルからソースコードへ変換する関連研究として，Executable UML [10),14)] と，Harel らの手法 [2] があげられる．

Executable UML はステートチャートを拡張し，完全に実行可能なソースコード（テンプレートレベルのソースコードではない）を生成可能にしたものである．Executable UML では，アクション言語を導入しており，ステートチャートの状態内にメッセージの送信などの様々な処理を記述することができる．Executable UML の状態はメソッドに対応し，アクション言語による記述がメソッド内のソースコードに対応する．この対応関係を利用することにより，実行可能なコードが生成可能である．しかしながら，完全に実行可能にするためには詳細なコードまでアクション言語で記述しなければならず，モデルが肥大化し，読みにくくなるという欠点がある．

Harel らの手法は，シーケンス図を用いた手法であり，LSC（Live Sequence Chart）というシーケンス図を拡張したモデルを用いる．LSC は，メッセージに加え，分岐構造やループ構造，一時変数などの構造を持っており，これらの要素をソースコードと対応させ，シーケンス図を 1 つの関数のように表現する．また，シーケンス図のメッセージとメソッドが対応しており，メッセージに対応する実際のコードは LSC とは別に用意する必要がある．そのため，LSC のみでは完全に実行可能なコードを生成することはできず，LSC はクラスを組み合わせるためのドライバのような役割を果たす．なお，Harel らの手法では，LSC を直接実行するため，コードの生成については考慮していないが，LSC を用いることでコードの自動生成も可能である．

2.4 ソースコードからモデルへの変換

Tonella らは，ソースコードを静的に解析し，シーケンス図（およびコラボレーション図）を生成する手法を提案している [15]．Tonella らの手法では，まずソースコードを main 関数から解析し，OFG（Object Flow Graph）と呼ばれる，各オブジェクトの生成および代入の流れを表すグラフを作成する．次に，ユーザが指定したメソッドからソースコードを再び解析し，シーケンス図を作成する．この際，OFG の情報を利用することにより，オブジェクトを識別する．Tonella らの手法をラウンドトリップエンジニアリングに用いる場合，既存のモデルと対応をとるためのメソッドを探す必要がある．すなわち，既存のすべてのモ

*1 Together はシーケンス図に関しても一部サポートしている．

© 2008 Information Processing Society of Japan

デルについて，対応するモデルを生成するためのメソッドを指定しなければならない．また，各シーケンス図を別々に生成するため，生成した複数のモデル間の関連情報は存在せず，hMSC に対応するような情報も生成できない．そのため，ステートチャートのようなモデルを生成するのは困難である．そのような情報は解析の際にあらかじめ与えるか，実際に実行することにより解析する必要がある．

Malloy らはプログラムを実際に実行した結果を解析することによりクラス図やコールグラフ，シーケンス図などの様々なモデルを生成する手法を提案している[8]．しかしながら，実際に実行する手法では，特定の入力に対する実行結果のみに依存したモデルしか得ることができない．そのため，ある入力では発生しなかったメッセージが，別の入力では発生する場合などが考えられ，モデルに変換する際に情報が欠落してしまう可能性がある．変換の際に情報が欠落してしまうと，再変換した際に元に戻すことができなくなってしまう．このため，プログラムを実際に実行する手法は，ラウンドトリップエンジニアリングには不適切であるといえる．

3. 動的モデルに着目したラウンドトリップエンジニアリング支援ツール

本論文では，動的モデルに着目したラウンドトリップエンジニアリング支援ツールを提案する．本ツールは MVC パターンに基づいたアプリケーションを対象とし，シーケンス図，ステートチャート，ソースコード間のラウンドトリップエンジニアリングを支援する．

3.1 概　　要

MVC（Model，View，Controller）パターン[7] は，ソフトウェアの設計パターンであり，ユーザへの入出力部分（ビュー）と，ソフトウェアの内部的なロジック（モデル）を分離し，ビューとモデルをコントローラにより制御するというパターンである．近年，Web アプリケーションを中心として，広く用いられている．

MVC パターンアプリケーションでは，モデル部がどのように実行されるかという情報をコントローラ部が保持しているという特徴がある．すなわち，モデル部の動的側面に関する情報の一部を，コントローラ部が保持しているといえる．また，モデル部に関しては従来のアプリケーションと同様であり，従来のオブジェクト指向開発を行うことができる．そこで，本研究では，コントローラ部の情報を利用することにより，モデル部の動的側面のラウンドトリップエンジニアリングの支援を行う．また，コントローラ部の開発と，ラウンドトリップエンジニアリングへの情報の利用を容易にするために，コントローラモデルを提案する．これはコントローラ部をモデリングするためのモデルであり，コントローラのソースコードは，コントローラモデルから自動生成可能である．なお，ビュー部はデータを表示するだけの単純な構造を想定しているため，本研究では扱わない．

3.2 機　　能

提案ツールは，次の機能を持つ．

(1)　シーケンス図からステートチャートの生成
(2)　ステートチャートからシーケンス図の生成
(3)　ステートチャートからソースコードの生成
(4)　ソースコードからシーケンス図の生成
(5)　コントローラモデルからコントローラコードの生成

本ツールでは，シーケンス図として UML シーケンス図，ステートチャートとして UML ステートチャートを扱う．また，ソースコードは Java を対象とする．なお，本ツールはクラス図に関してもサポートしているが，本論文では省略する．

本ツールでは，シーケンス図とステートチャートの間は完全に相互変換が可能であるため，動的モデルをひとまとめに扱うことが可能である．すなわち，シーケンス図とソースコード間，ステートチャートとソースコード間を別々に扱う必要はなく，動的モデルとソースコード間と，ひとまとめにして扱うことが可能である．そのため，シーケンス図から直接ソースコードを生成する機能はないが，この機能は 1 度ステートチャートに変換してからソースコードの生成を行うことにより実現できる．同様に，ソースコードからステートチャートの生成は，1 度シーケンス図を生成し，そのシーケンス図をステートチャートに変換することにより実現できる．

これらの機能により，MVC パターンのモデル部に関して，シーケンス図，ステートチャート，ソースコード間のラウンドトリップエンジニアリングを支援する．また，コントローラ部に関しては，完全に実行可能なソースコードがコントローラモデルから自動生成可能である．

次に，まず提案するコントローラモデルについて述べ，その後，各機能についての詳細を述べる．

3.3 コントローラモデル

コントローラモデルはビューとロジックの流れを表したモデルであり，ページ，アクション，イベントの 3 つの要素からなる．ページとは入出力が行われるビュー部であり，JSP などの Web アプリケーションでは Web ページに対応する．アクションはコントローラが呼び出すロジックであり，各アクションとシーケンス図が 1 対 1 で対応する．また，イベン

© 2008 Information Processing Society of Japan

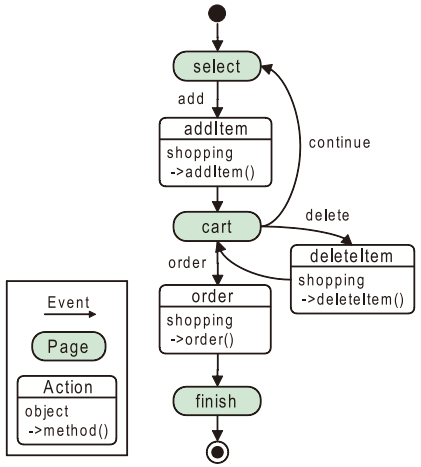

図 1　コントローラモデル
Fig. 1　Controller model.

トはビューから送られるリクエストに対応し，遷移を引き起こす．

コントローラモデルの例を**図 1** に示す．図 1 はオンラインショッピングシステムの例であり，たとえば，select ページで add というイベントが発生すると，addItem というアクションを実行し，cart ページに遷移することを表現している．アクション内には設計情報として呼び出すロジックを記述してあり，たとえば addItem には shopping オブジェクトの addItem メソッドを呼び出すことが記述されている．なお，コントローラモデルに対しては shopping オブジェクトのインスタンス生成やセッション管理などの様々な指定を行うことができるが，詳細は省略する．

3.4　シーケンス図→ステートチャート

シーケンス図からステートチャートの生成は，Hasegawa らの手法[3] と同様，各シーケンス図 1 つ 1 つに対応するステートチャートを作成し，シーケンス図のつながりの情報を利用してステートチャートを組み立てる．つながりの情報として，本手法では，コントローラモデルを hMSC の代わりに用いる．

また，Hasegawa らの手法では，メッセージの受信と送信がステートチャートの状態遷移に対応する．しかし，本手法では，ソースコードとの対応関係を考慮し，Harel らの手法[2] と同様，メッセージの受信をメソッドの呼び出しと対応付け，Executable UML [10),14)] と同様，状態をメソッドに対応付ける．すなわち，本手法では状態の遷移はメッセージの受信のみと対応し，送信とは対応しない．また，本手法では Executable UML のアクション言語を導入する．本手法ではメソッド呼び出し，分岐，ループの 3 つの構造のみを扱う独自の記述法を導入し，シーケンス図のメッセージの送信，分岐，ループ構造をそれぞれ対応付ける．なお，自分自身へのメッセージの送信は，アクションの記述にのみ対応させ，状態の変化には対応させない．

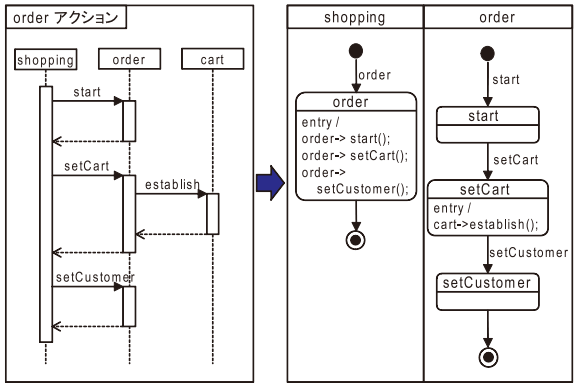

図 2　シーケンス図→ステートチャート
Fig. 2　Sequence diagram to statechart.

ステートチャートの生成手順を以下に示す．

(1)　1 つのシーケンス図に対応するステートチャートの作成
メッセージの受信を状態の遷移とし，遷移および状態名はメッセージと同じ名前にする．また，メッセージの送信元にはアクション記述を挿入する．分岐やループがある場合にも，同様にアクション記述を挿入する．これをすべてのシーケンス図について行う．

図 2 は 1 つのシーケンス図に対応するステートチャートの作成例である．このシーケンス図は図 1 の order アクションに対応するシーケンス図であり，このシーケンス図から shopping，order，cart の 3 つのステートチャートが生成される（図 2 では cart は省略している）．たとえば，shopping から order への start メッセージの送信はステートチャート shopping の order 状態内のアクション記述 order->start() に対応し，受信はステートチャート order の状態遷移 start に対応する．また，ス

© 2008 Information Processing Society of Japan

テートチャート shopping の order 状態への遷移は，図 1 の order アクション内の shopping->order() に対応している．

(2)　コントローラモデルから hMSC の作成
コントローラモデルに登場するすべてのページを取り除き，1 つのアクションを 1 つの状態としたステートチャートを作成する（以後，これを便宜上 hMSC と呼ぶ）．ページを取り除く際に，ページに対する入力の遷移と出力の遷移をつなげた遷移を作成する．入力，出力が複数ある場合には，それらのすべての組合せの遷移を作成する．なお，同じ遷移が複数できる場合には，それらをまとめて 1 つにする．

(3)　ステートチャートの組み立て
hMSC の各状態はシーケンス図に対応しているため，各状態を手順 (1) で作成したステートチャートで置き換え，ステートチャートを組み立てる．なお，対応するステートチャートが存在しない場合（オブジェクトがアクションに登場しない場合）には，単に状態を削除し，入力と出力の遷移をつなげる．
ステートチャートの組み立て例を**図 3** に示す．図 3 (a) は手順 (2) で作成した hMSC であり，図 3 (b) が手順 (1) で作成したステートチャートである．これらを組み立て，図 3 (c) のステートチャートを作成する．

(4)　手順 (2)，(3) をシーケンス図に登場するすべてのオブジェクトに対し繰り返し，ステートチャートを生成する．

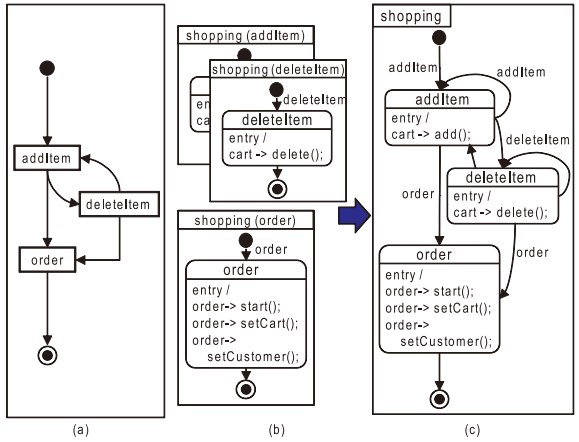

図 3　ステートチャートの組み立て
Fig. 3　Completion of statechart.

3.5　ステートチャート→シーケンス図

ステートチャートからシーケンス図への変換は，状態内のアクション記述をトレースし，メッセージの流れを解析することにより行う．ここで，コントローラの情報を利用することにより，トレースの開始位置を決定することができる．具体的な手順は以下のとおりである．

(1)　解析開始状態の決定
コントローラモデルを使用し，解析の開始位置を決定する．シーケンス図はコントローラの各アクションに対応するため，各アクションが呼び出すメソッドが解析開始位置となる．

(2)　アクション記述のトレース
決定した解析開始位置から状態内のアクション記述を再帰的に解析しながらシーケンス図を作成する．メッセージの送信を発見した場合，まず，本ツールはシーケンス図にメッセージを挿入する．次に，そのメッセージに対応する状態をステートチャートから検索し，アクション記述の解析位置をその状態に移す．また，分岐を発見した場合には，分岐構造をシーケンス図に挿入し，分岐が閉じられるまでは，メッセージなどの挿入先をその分岐構造内にする．ループの場合も同様に行う．

(3)　手順 (1)，(2) をすべてのアクションに対し繰り返し，シーケンス図を生成する．

ステートチャートからシーケンス図を生成する例を**図 4** に示す．これは図 1 の order アクションに対応するシーケンス図の作成例であるため，order アクション内の記述より，解析開始位置はステートチャート order 状態となる．shopping の order 状態内では，まず order->start() という記述があり，これがシーケンス図の start メッセージに対応する．次に，解析位置をステートチャート order の start 状態に移すが，アクション記述が何もないため，解析位置を shopping の order 状態に戻す．これを解析開始状態の最後に達するまで繰り返すことにより，図右のシーケンス図を作成する．

3.6　ステートチャート→ソースコード

本ツールでは，Executable UML [10),14)] と同様，各ステートチャートがクラスに対応し，状態がメソッドに対応する．そして，アクション記述がメソッド内のコードに対応する．しかし，アクション記述を非常に詳細に記述しない限り，完全なソースコードをステートチャートから生成するのは不可能である．

そのため，本ツールでは既存のソースコードが存在しない場合（すなわち，ステートチャー

© 2008 Information Processing Society of Japan

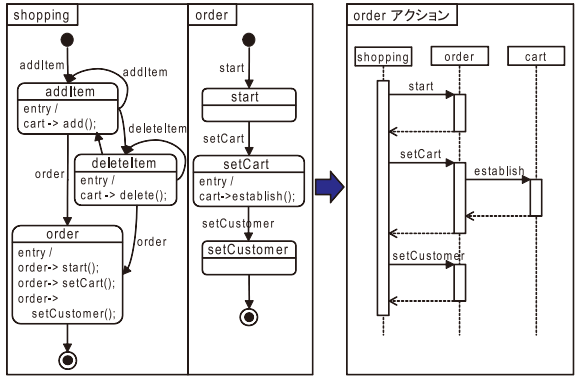

図 4　ステートチャート→シーケンス図
Fig. 4　Statechart to sequence diagram.

トからソースコードを最初に生成する場合）には，ソースコードのテンプレートを生成する．テンプレートコードの生成では，前述の対応関係を使用し，クラスおよびメソッド定義，そしてアクション記述に対応するメソッド内のコードを生成する．

ソースコードがすでに存在する場合には，本ツールはステートチャートと既存のソースコードの比較を行い，整合性を検証する．比較を行う際，本ツールはソースコードから 1 度シーケンス図を生成し，そのシーケンス図からステートチャートを生成することで，現在のソースコードに対応するステートチャートの生成を行う[*1]．次に，本ツールは生成したステートチャートと現在のステートチャートの比較を行う．比較では，各ステートチャートについて対応する状態が存在しているかどうかを確認し，存在している場合には，アクション記述が一致しているかどうかの比較を行う．本ツールでは記述可能なアクション記述をメッセージ送信，分岐，ループに限定しているため，アクション記述の比較は単純であり，1 行ずつ比較を行う．比較の結果，相違が発見された場合には，本ツールはユーザに報告を行う．

3.7　ソースコード→シーケンス図

ソースコードからシーケンス図の生成は，ステートチャートからシーケンス図の生成と類似している．まず，本ツールは解析開始位置をコントローラモデルから決定し，その後，アクション記述の代わりにソースコードを静的に解析する．

アクション記述ではメッセージの送信先オブジェクトがシーケンス図のオブジェクト名に対応していたが，ソースコードの場合はエイリアスがあるため，必ずしも対応しない．そこで，本ツールでは Tonella らの手法[15] に基づき，ソースコードから，各オブジェクトの生成，代入の流れを表すグラフ（OFG）を作成する．本ツールでは，解析開始位置はコントローラモデルから決定し，各アクションごとに OFG を作成する．しかし，各アクションごとにオブジェクトの流れを解析するため，解析中のアクションのロジックの外でインスタンス生成されるオブジェクトに関しては，インスタンス生成の場所に関する情報がない．このため，各シーケンス図でオブジェクトの対応関係をとることができない．すなわち，あるシーケンス図のオブジェクトと別のシーケンス図のオブジェクトが同じインスタンスかどうかを確認することができない．

そこで，本ツールでは各アクションごとに作成した OFG を統合することにより，すべてのオブジェクトのインスタンス生成位置を明らかにし，オブジェクトの対応関係を作成する．しかしながら，実行パスによって変数の参照するオブジェクトが変化する場合には，OFG の統合は困難である．そこで，本研究では，クラスのフィールドが参照するオブジェクトは変化しないという制約を設け，その制約のもとに OFG の統合を行う．

シーケンス図の生成アルゴリズムは以下のとおりである．

(1)　解析開始位置の決定
　ステートチャートからシーケンス図の生成と同様の手順で解析開始位置を決定する．

(2)　ソースコードの解析
　各解析開始位置からソースコードを再帰的に解析し，OFG および最初のシーケンス図（まだシーケンス図間のオブジェクトの対応関係がとれていない）を作成する．

(3)　冗長な分岐とループの除去
　ソースコードの解析では，内部にメッセージ送信を含まない，冗長な分岐やループが現れてしまう．そのため，それを取り除く．

(4)　OFG の統合
　すべてのインスタンス生成した場所の情報が含まれない OFG について，その OFG のルートノード（オブジェクトへの最初のアクセス）と一致するノードを他の OFG から検索し，OFG を結合する．OFG のノードはオブジェクトの参照を保持している変数を表しており，エッジはオブジェクトの代入を表している．すなわち，ルートノードの変数に対するオブジェクトの代入を，他の OFG から検索し，OFG を結合

*1 ソースコードからシーケンス図の生成については 3.7 節を参照．

© 2008 Information Processing Society of Japan

するといい換えることができる．この際，同じノードが複数見つかってしまうと，どちらに統合するかの判断が困難であるため，前述の制約を設けた．

(5)　オブジェクトの決定

シーケンス図の各オブジェクトについて，OFG 内の対応するノードを検索し，OFG のルートノード名で，シーケンス図のオブジェクト名を置き換える．

図 5 はソースコードからシーケンス図を生成した例である．これは図 1 の addItem アクションに対応するシーケンス図の作成例であるため，解析開始位置は Shopping クラスの addItem() となる．ソースコードの各メソッド呼び出しはシーケンス図のメッセージに対応し，メソッドの引数もメッセージの引数に対応する．なお，Cart.add() 内の if は，内部にメッセージ呼び出しがないため，シーケンス図には登場しない（Stock.modify() については分岐に関係なく実行されるため，残っている）．また，シーケンス図のオブジェクト名はインスタンス生成された時点での変数名と型により置き換えられる．たとえば，cart:Cart (shopping.cart) は shopping.cart() 内でインスタンス生成された Cart クラスのインスタンス変数 cart を表している．また，Stock (static) は Stock が静的クラスであることを表している．

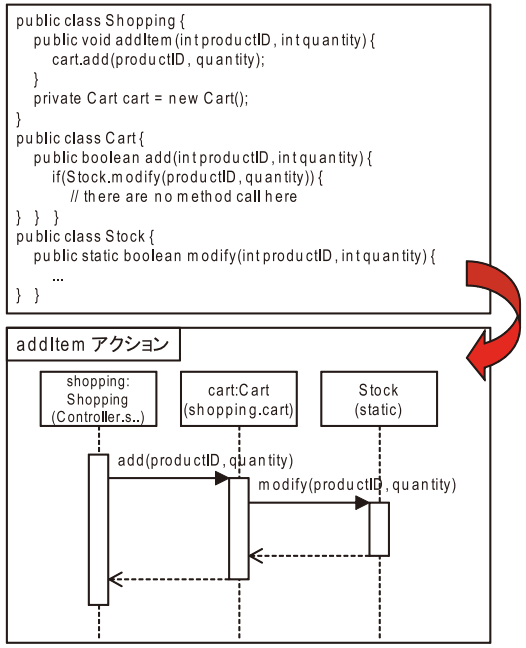

図 5　ソースコード→シーケンス図
Fig. 5　Sourcecode to sequence diagram.

3.8　コントローラモデル→コントローラコード

コントローラコードは，受信したイベントと現在のビューをもとに，ページやアクションへの遷移を行うためのコードからなる．コントローラコードはビューの実装に依存するため，ビューの実装ごとに異なるコード生成器が必要となる．本ツールは現在，JSP Web アプリケーション用のコントローラコードの生成をサポートしている．

図 6 に生成したコードの例を示す．図 6 のコントローラでは，各アクションで shopping オブジェクトの，アクション名と同名のメソッドを呼び出すように指定している．コントローラは HttpServlet クラスのサブクラスとして実装しており，doGet と doPost メソッドをオーバライドし，リクエストを共通の processRequest メソッドで処理している．processRequest では，イベント名と現在のビューにより呼び出すアクションを分岐している．本ツールは各アクションについて action_addItem のようなメソッドを作成し，分岐からはそのメソッドを呼び出す．また，3.3 節で述べたとおり，各アクションには呼び出すべきメソッド以外にも様々な指定が可能である．図 6 では省略しているが，実際には shopping インスタンスを

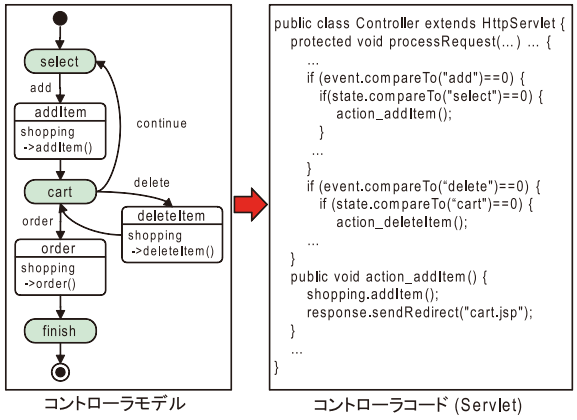

図 6　コントローラモデル→コントローラコード
Fig. 6　Controller model to controller code.

© 2008 Information Processing Society of Japan

生成する処理[*1]や，リクエストからパラメータを取得してアクションに渡す処理などの様々なコードを生成する．

なお，本ツールはコントローラモデルからコントローラコードを生成するのみで，コードからモデルの生成については考慮していない．しかしながら，コントローラモデルから完全なコードを生成できるため，開発者はコントローラコードを編集する必要はないと考えられる．

4. 適用事例，考察

本章では，本研究で提案するツールの適用例を示し，考察を行う．本ツールの適用例として，オンラインショッピングシステムの開発を行った．反復型の開発プロセスを想定し，まずプロトタイプを作成し，プロトタイプに対して様々な変更を行った．最終的に，アクション数 23，ソースコード約 4,000 行のシステムを作成し，本ツールの有用性および変換の正確性を確認した．また，各種変換にかかる時間は，最大でも 1 秒程度であり，現実的な時間で収まった．変換時間の測定環境は以下のとおりである．

- CPU：Intel(R) Pentium(R) 4 CPU 3.40 GHz
- RAM：1,024 MB RAM

以下で，プロトタイプ開発，プロトタイプ拡張のそれぞれについて述べる．

4.1 プロトタイプ開発

プロトタイプの開発では，まずコントローラモデルを使用してシステム全体の流れを設計した．これにより，システムの機能が明確となり，シーケンス図の作成単位が明確になるという利点がある．次に，コントローラモデルの各アクションをシーケンス図を用いてモデリングし，本ツールを使用してシーケンス図からステートチャートを作成した．その後，本ツールを使用してシーケンス図とステートチャートを相互に変換しながら，モデルを詳細化していった．具体的には，状態の追加や分割をステートチャートレベルで行い，クラスの追加や引数の追加をシーケンス図レベルで行った．この際，片方のモデルに変更を加えた後，すぐに本ツールを使用してもう片方のモデルに反映を行った．これにより，つねにシーケンス図とステートチャートが同期を保った状態とすることができ，また，片方のモデルの変更がもう片方のモデルに与える影響も容易に知ることができ，効率的な開発が可能であった．

モデルを十分に詳細化した後，本ツールを使用してステートチャートからソースコードのテンプレートを作成した．その後，テンプレートを詳細化し，プロトタイプを完成させた．最後に，本ツールを使用してソースコードからシーケンス図を生成することにより，設計どおりに実装できていることを確認することができた．

4.2 プロトタイプの拡張

プロトタイプに対し，様々な拡張や変更を行った．たとえば，カートにある商品の個数を変更する機能の追加は，直接ソースコードを修正することにより行った．また，カートを空にする機能では，新たにシーケンス図を記述し，その後ソースコードを記述した．ほかにも，注文の最終確認ページの追加では，新たなシーケンス図の記述と既存のシーケンス図の修正を行い，変更をソースコードに反映させた．ほかにも，支払い方法の選択やメール送信機能など，様々な拡張を行った．

機能の追加や変更の際には，コントローラモデルを変更する必要が生じた．コントローラ部の変更はモデル部の動的側面に影響を与えるため，コントローラモデルの変更をシーケンス図やステートチャートに反映させる必要が生じる．本ツールでは，ステートチャートかソースコードからシーケンス図，ステートチャートを再生成することにより，コントローラモデルの変更を自動的に反映させることができる．なお，モデル部の変更はコントローラ部には影響を与えないため，モデル部からコントローラ部への反映は不要である．

本ツールを使用することで，機能の追加や要求の変更に対し，モデリング段階に戻って設計をモデルレベルで変更し，それをソースコードに反映させることができた．さらに，ソースコードを直接編集し，それをモデルに反映させることもできた．このように，本ツールを使用することにより，シーケンス図，ステートチャート，ソースコードから適切なものを選択して修正を行うことができ，効率的に開発を行うことができた．

4.3 考　　察

本節では，変換の妥当性とモデリング時の制約，および本ツールの適用範囲についての考察を行う．

シーケンス図からステートチャートへの変換では，メッセージに加え，分岐やループを扱うことができ，これらのすべての要素がステートチャートの要素に対応している．このため，変換により情報が失われることはない．ただし，メッセージをメソッドに対応させたため，同期メッセージのみしか扱うことができない．このため，非同期制御のシーケンス図を扱うことはできないという問題がある．

一方，ステートチャートからシーケンス図の変換では，本ツールは状態や遷移ではなく，

*1 shopping インスタンスはコントローラの初期化時に生成されるように指定している．このようなクラスはセッション内で保持され，マルチユーザアプリケーションでユーザを区別するのに使用可能である．

© 2008 Information Processing Society of Japan

アクション記述のみを解析して変換を行う．このため，状態の追加や削除を行った際，対応するアクション記述も追加/削除しないと，シーケンス図から再生成した際に元に戻らなくなってしまうという問題がある．この問題を解決するために，正しく変換できるステートチャートかどうかを検査する機能や，対応するアクション言語を自動的に修正する機能が必要である．同様に，遷移も変換には使用しないため，遷移時のアクティビティやガード条件は変換の際に欠落してしまう．このため，これらは状態内のアクションとして記述する必要がある．

また，本ツールではソースコードを解析する際に，クラスのフィールドが参照するオブジェクトは変化しないという制限を設けた．制限が守られている限り，OFG の統合によりオブジェクトを特定できるため，必要な情報が欠落することはないが，違反する場合には正しい結果を得ることができない．本適用事例においてはこの制限が問題となることはなかったが，制限に違反するオブジェクトがあった場合には，そのオブジェクトを解析対象から外す必要がある．違反するオブジェクトが別のオブジェクトにメッセージを送信している場合には，そのメッセージ送信がモデルから欠落してしまうという問題が発生する．しかし，メッセージ送信がなければ，他のオブジェクトに影響を与えないため，他のオブジェクトに関しては解析が可能である．

次に，本ツールの適用範囲について考察を行う．本ツールでは，各種変換において，コントローラモデルが必須である．このため，本ツールを使用する場合には，基本的には最初から本ツールを使用して開発を行う必要がある．しかし，コントローラモデルを用意することにより，既存のアプリケーションに対しても使用可能である．この際，コントローラとモデル（ロジック）が明確に分離している必要はなく，各ロジックが任意のメソッドが呼び出されることにより起動する構造になっていれば，本ツールに対応するコントローラモデルを（手動で）作成することが可能である．このため，既存のアプリケーションなどに対しても本ツールによる各種変換が可能であるが，この場合，コントローラモデルが正しいかどうかを検証することができず，コントローラコードの自動生成も行うことができない．

5. 結　　論

動的モデルとソースコード間のラウンドトリップエンジニアリングを支援するツールを提案した．本ツールは MVC パターンに基づいたアプリケーションを対象とし，シーケンス図，ステートチャートとソースコード間のラウンドトリップエンジニアリングを支援する．本ツールを使用することにより，動的モデルとソースコード間の効率的なラウンドトリップエンジニアリングが可能となる．

今後の課題としては，4.3 節で述べたステートチャートの検証およびアクション記述の自動修正や，ソースコードの静的解析で設けた制限の解決，既存の MVC フレームワークとの連携などがあげられる．

参　考　文　献

1) Borland: Borland Together Technologies (2006). http://www.borland.com/us/products/together/
2) Harel, D. and Marelly, R.: Specifying and Executing Behavioral Requirements: The Play-In/Play-Out Approach, *Software and System Modeling*, pp.82–107 (2003).
3) Hasegawa, H., Takada, S. and Doi, N.: Supporting the Iterative Development of Sequence Diagrams and Statecharts, *Proc. 8th IASTED Int'l Conf. on Software Engineering and Applications*, pp.736–742 (2004).
4) Henriksson, A. and Larsson, H.: A Definition of Round-trip Engineering, Technical Report (2003). http://www.ida.liu.se/~andhe/re.pdf
5) IBM: Rational Software Modeler (2006). http://www-306.ibm.com/software/rational/
6) ITU-T. Recommendation Z.120 (1999).
7) Krasner, G.E. and Pope, S.T.: A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 system, *Journal of Object Oriented Programming*, Vol.1, No.3, pp.26–49 (1988).
8) Malloy, B.A. and Power, J.F.: Exploiting UML Dynamic Object Modeling for the Visualization of C++ Programs, *Proc. ACM Symposium on Software Visualization*, pp.105–114 (2005).
9) Medvidovic, N., Egyed, A. and Rosenblum, D.S.: Round-Trip Software Engineering Using UML: From Architecture to Design and Back, *Proc. 2nd Workshop on Object-Oriented Reengineering*, pp.1–8 (1999).
10) Mellor, S.J. and Balcer, M.J.: *Executable UML – A Foundation for Model-Driven Architecture*, Addison Wesley (2002).
11) OMG: Unified Modeling Language (UML), version 2.1.1 (2007). http://www.omg.org/
12) Omondo: EclipseUML (2006). http://www.eclipseuml.com/
13) Sendall, S. and Kuster, J.: Taming Model Round-Trip Engineering, *Proc. Workshop on Best Practices for Model-Driven Software Development* (2004).
14) Starr, L.: *Executable UML – How To Build Class Models*, Prentice-Hall, Inc. (2002).
15) Tonella, P. and Potrich, A.: Reverse Engineering of the Interaction Diagrams from

© 2008 Information Processing Society of Japan

C++ Code, *Proc. 19th Int'l Conf. on Software Maintenance*, pp.159–168 (2003).

(平成 19 年 10 月 17 日受付)
(平成 20 年 1 月 8 日採録)

今関　雄人

2005 年慶應義塾大学理工学部卒業．2007 年同大学大学院理工学研究科修士課程修了．同年同博士課程入学，現在に至る．ソフトウェア工学に関する研究に従事．

高田　眞吾（正会員）

1990 年慶應義塾大学理工学部卒業．1992 年同大学大学院理工学研究科修士課程修了．1995 年同博士課程修了．博士（工学）．同年奈良先端科学技術大学院大学情報科学研究科助手．1999 年慶應義塾大学理工学部情報工学科専任講師．2006 年より同大学助教授（現在は准教授）．ソフトウェア工学，情報検索等の研究に従事．電子情報通信学会，日本ソフトウェア科学会，ACM，IEEE CS 各会員．

土居　範久（正会員）

1969 年慶應義塾大学大学院博士課程単位取得退学．慶應義塾大学理工学部教授を経て，2003 年より中央大学理工学部教授，慶應義塾大学名誉教授．工学博士．現在，日本学術会議副会長，文部科学省科学技術・学術審議会委員，総務省情報通信審議会委員・情報通信技術分科会会長，科学技術振興機構（JST）社会技術研究開発センター「情報と社会」領域総括，特定非営利活動法人日本セキュリティ監査協会会長，国際計算機学会（ACM）日本支部長，等．専門はソフトウェアを中心とした計算機科学．情報処理学会名誉会員．

© 2008 Information Processing Society of Japan