社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－SE－155 (15)
2007／3／23

UML図面要素に対応するJava協調クラス群の抽出ツール

菅井拓海† 小谷正行† 落水浩一郎†

本稿では，UML図面上のモデル要素に対応するJava協調クラス群を抽出するツールを紹介する．抽出にはUML図面のモデル要素とJavaクラス群それぞれが担う責務と協調の概念を利用する．抽出アルゴリズムはUML図面モデル要素と名前が一致するJavaクラスを探し，そのJavaクラスを中心としてJavaクラス間の関係を追跡する．また図面間のモデル要素の依存関係を付加し，変更波及解析に利用するツールを実装し，結果を評価する．

An extraction tool for Java collaboration classes coresponding to UML model elements

Takumi Sugai,† Masayuki Kotani† and Koichiro Ochimizu†

In this paper, we propose a tool for extracting a set of Java collaboration classes corresponding to a UML model element. We use Responsibility and Collaboration concepts to define a correspondence between UML model elements and Java classes. An outline of extracting algorithm is as follows; at first it searches a Java core class that has the same name model element has. Next it traces Java classes connected to the Java core class based on a criteria describe in this paper. We also show the evaluation of the tool.

1. はじめに

ソフトウェア共同開発では，開発プロセスを通じて多量の成果物が作成される．それらの間には様々な依存関係が存在する．このため，ある成果物の変更は，他の成果物の変更を要求することがある．ここで，依存関係とは，成果物Aが成果物Bに依存するとき，成果物Bが変更されたとき，成果物Aも変更される可能性があることを示す関係である．しかし，それらを多量の成果物の中から探し出すことには多大な労力を要する．

我々は，開発プロセスを通じて作成された膨大な文書やプログラムの変更を安全かつ効率的に行うことを支援するための変更作業支援ツールを開発している．このツールは，UML図面のモデル要素を変更する時に，変更される可能性があるモデル要素とJavaソースコードのクラス（以下Javaクラス）群を表示する機能をもつ．

すでに，モデル要素間の波及を特定するための依存関係付加・変更波及解析手法は提案し実装した[1)2)]．この手法に，モデル要素に対応するJavaクラス群を特定する手法を付加することで，変更される可能性があるクラス群を表示する．

本稿では，2節で関連研究と本研究の位置づけについて述べる．3節で，モデル要素とJavaクラス群の対応づけについて述べる．4節で，対応づけの手法である協調クラス群抽出アルゴリズムについて述べる．5節で，アルゴリズムの適用例を述べる．6節で，Eclipseプラグインとして実装したツールについて述べる．7節で，実験とその結果を報告する．8節で，本稿のまとめと今後の課題を述べる．

† 北陸先端科学技術大学院大学 情報科学研究科

2. 関連研究

関連研究に，UML図面からJavaソースコードを生成する研究事例として文献3)やプログラム中のパターンを抽出するという研究事例として文献4)がある．

文献3)では，図とソースコードの変換規則を作成し，UMLのクラス図・アクティビティ図からJavaソースコードを生成する試みを行っている．これは図面のモデル要素とソースコード上のクラスが一対一に対応することが前提となる．しかし，図面のモデル要素とJavaクラスは一般的に一対一に対応しないことが多い．例えば状態によって動作が変化するモデル要素を，デザインパターンのStateパターンを利用して実装する場合である．このとき状態の数だけJavaクラスを追加するので，モデル要素に対応するJavaクラスは複数となる．本研究では，モデル要素とJavaクラスが一対一に対応しない場合も考慮する．

文献4)では，ソースコードの構造を解析する静的解析と，さらに実行中のメソッド呼び出しやオブジェクトに関する情報を取得する動的解析を組み合わせて，Javaソースコードからデザインパターンを抽出する試みを行っている．

3. モデル要素とJavaクラス群の対応づけ

本節では，モデル要素とJavaクラス群の対応を定義する．責務と協調という概念を利用する．

3.1 モデル要素やJavaクラスの責務・協調の定義

本稿では，モデル要素とJavaクラス群の対応の定義に，責務と協調という概念を利用する．責務と協調の概念を**図1**に示す．

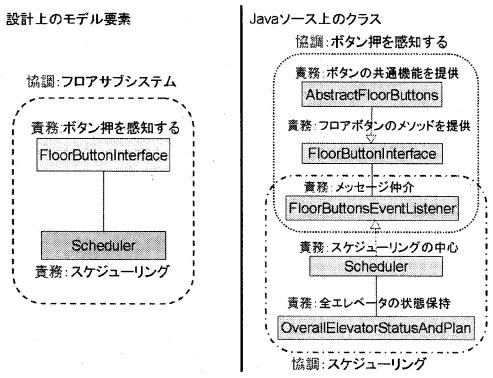

図 1 責務と協調

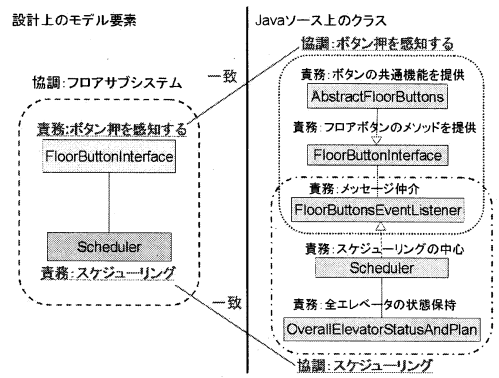

図 2 モデル要素の責務とクラス群の協調の対応付け

一般的に責務は，モデル要素やクラスが果たす機能[5]として定義される．しかしモデル要素やクラスの機能を図面から認識することは困難である．文献 5) の CRC カードのように，図面に責務が記述されていればよいが，実際はそうでない場合が多い．

本稿では，責務を「名前で抽象された，モデル要素やクラスが単体で担う機能」と定義する．モデル要素やクラスを作成するとき，作成者は割り当てる機能を踏まえて名前をつけることが多い．そのためモデル要素の名前は，モデル要素の機能を抽象して表現される．例えば図 1 のモデル要素 FloorButtonInterface は，ボタン押を感知するという責務を持っており，それが FloorButtonInterface という名前に抽象化されている．ソースコード上のそれぞれの Java クラスも，同様に責務を持っている．

協調を，「複数のクラスやモデルが通信してある機能を果たすこと，またはその機能」と定義する．例えば図 1 のモデル FloorButtonInterface と Scheduler は，通信してフロアサブシステムとしての機能を果たす．またソースコード上においても，Java クラス同士が通信してある機能を果たす．

また協調は，責務を中心に実現される．例えば Java クラスにおけるボタン押を感知する機能は，AbstractFloorButtons，FloorButtonInterface，FloorButtonsEventListener の 3 つのクラスの責務の連携によって実現される．モデル要素上の協調も同様である．

3.2 対応付けの基本方針

本稿では，モデル要素の責務と Java クラス群の協調を対応づける手法を提案する（**図 2**）．

プログラム作成者は，モデル要素の責務を実現するために，クラス群を作成する．モデル要素をソースコードに写像するとき，まずモデル要素に対応するクラスが作成される．次に実際の処理を行うクラスが複数作られる．このとき作られた Java クラスの協調がモデル要素の責務と一致する可能性が高い．

モデル要素の責務を担うために協調するクラス群を，協調クラス群と定義する．協調クラス群に含まれる，モデル要素をそのままソースコードに写像したクラスは，モデル要素の名前と同じ名前であることが多い．このクラスを協調クラス群のコアと定義する．コアは，Facade パターンにおける Facade クラスのような役割を果たすことが多い．すなわち，メッセージを他の協調クラス群から受け取り，自身の協調クラス群の要素にメッセージを振り分けるという役割である．Facade の役割を果たす Java クラスが作られる理由は，設計図上のメッセージの流れと，Java クラス群の協調を実現するメッセージを分離するためである．設計図上のメッセージの流れが，Facade クラス間のメッセージとなり，Java クラス群の協調を実現するメッセージが，Facade クラスからメッセージを受け取る他の Java クラスとなる．

4. 協調クラス群の抽出

本節では，協調範囲を決定するための Java クラス間の関係を定義し，協調クラス群を抽出するためのアルゴリズムを述べた後，その適用例を示す．

4.1 Java クラス間の関係の定義

協調範囲を決定するための，クラス間の関係を分析する．本稿では 3 種類に Java クラス間の関係を分類する（**図 3**）．

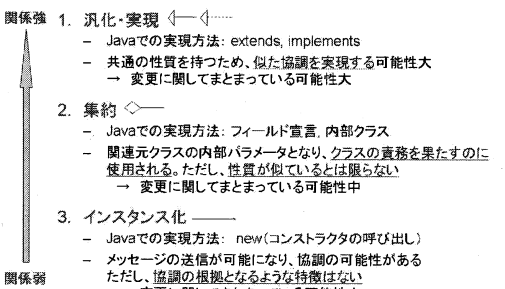

図 3 協調範囲算出に利用する Java クラス間の関係

4.1.1 Java クラス間の関係

汎化・実現は，extends, implements によって接続された Java クラス間の関係である．汎化・実現で接続された Java

クラスは，共通のメソッドやフィールド，またはシグネチャを持つ．

集約は，フィールド宣言や内部クラス宣言により接続されたJavaクラス間の関係である．関連先クラスは関連元クラスの内部パラメータとして利用される．

インスタンス化は，new演算子によって接続されたJavaクラス間の関係である．関連元クラスから関連先クラスへ，メッセージの送信が可能になる．

4.1.2 関係の強さ

関係には，変更に関してまとまっているかという観点から強さを定義する．

最も強い関係は汎化・実現である．汎化・実現の関係になるクラスは，共通のメソッドやフィールド，シグネチャを持つという特徴から，同じ協調を担うことが多い．そのため，モデル要素の責務を変更する場合は共に変更される可能性がある．よって汎化・実現を最も強い関係とする．

次に強い関係は，集約である．集約は関連先クラスが関連元クラスの内部パラメータとなり，関連元クラスの責務を担うために利用される．よって関連元クラスの責務が変更される場合，関連先クラスの責務も変更される可能性がある．ただし関連元クラスと関連先クラスには，共通する性質があるとは限らないため，同じ協調の範囲に含まれている可能性は汎化・実現より低くなる．

最も弱い関係はインスタンス化である．インスタンス化によって関連元クラスから関連先クラスへメッセージの送信が可能となるため，協調の可能性が出てくる．しかし，協調の根拠となるような特徴がインスタンス化には見られない．よってインスタンス化は変更のまとまりに関して最も弱い関係になる．

4.2 協調クラス群抽出アルゴリズム

協調クラス群を抽出する，クラス間の関係の強さとモデル要素の名前を利用したアルゴリズムを述べる．アルゴリズムの概要を**図4**に示す．

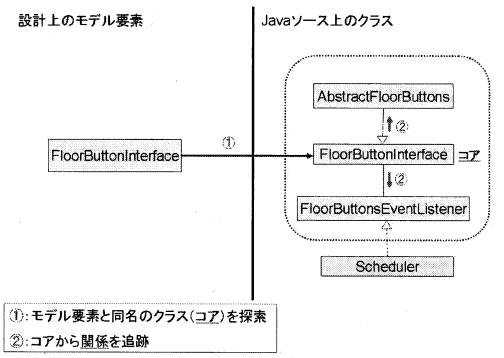

図4 協調クラス群抽出アルゴリズムの概要

協調クラス群抽出アルゴリズムは以下の4ステップからなる．

(1) **モデル要素の名前と一致する協調クラス群のコアを見つける**

(2) **コアから，汎化・実現の関係を追跡し，継承グループを作成する**

(3) **継承グループの各要素から追跡ルールに従い関係を追跡する**

(4) **追跡できる関係が無くなるまで，3を繰り返す**

ただし，関係は必ず追跡されるのではなく，協調の可能性があるか判断する追跡ルールに基づいて追跡が行われる．追跡ルールを以下に示す．

(1) **汎化・実現は関連元・関連先どちらからでも追跡できる**

(2) **追跡先がコアだった場合，追跡しない**

(3) **インスタンス化を追跡するとき，インスタンス化するものが他の継承グループに含まれていたら，追跡しない**

ルール1は，汎化・実現の関係にあるクラスは情報を共有しているため，方向を考慮する必要が無い．

ルール2は，追跡先がコアということは，異なるモデル要素に対応するクラスとなる．

ルール3については，インスタンス化には協調するという根拠がなく，他の継承グループに処理を委譲している可能性があると推測できる．

5. 適 用 例

文献7)のエレベータ制御システムを実装したプログラムの一部に，アルゴリズムを適用した例を述べる（**図5**）．適用した部分にはElevatorManager, ElevatorController, ArrivalSensorInterfaceの3個のモデル要素があり，責務はそれぞれ配送するエレベータを選択する，エレベータを制御する，エレベータが到着したことを感知する，となっている．これらの責務を担うJavaクラスは8個存在する．

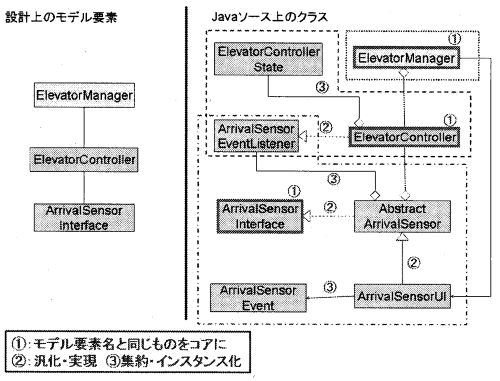

図5 アルゴリズムによる抽出の例

ステップ1では，モデル要素名と同じ名前のクラスをソースコードから探索し，そのクラスを協調クラス群のコアとする．図の例では，ElevatorManager, ElevatorController, ArrivalSensorInterfaceの3個のモデル要素と同名のクラスがコアとなる．

ステップ 2 では，コアから汎化・実現の関係で追跡できるクラス群を継承グループとする．図の ElevatorController から ArrivalSensorEventListener へは実現の関係で追跡可能なので，この 2 つは継承グループとなる．

ステップ 3 では，継承グループ内の各要素から集約・インスタンス化の関係で追跡できるものを，協調クラス群として含める．ArrivalSensorUI から ArrivalSensorEvent へインスタンス化の関係があるので，ArrivalSensorEvent を ArrivalSensorInterface をコアとする協調クラス群の要素として追加する．

この事例で，追跡ルールが適用された部分を**図 6** に示す．

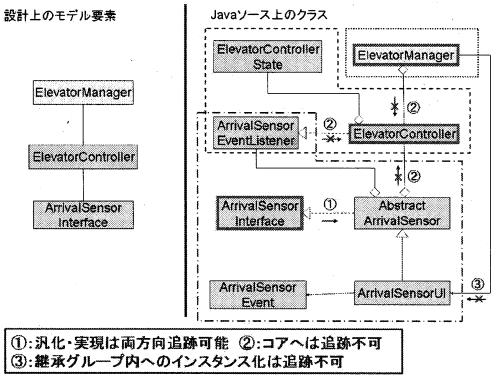

図 6 追跡のルール適用の例

ルール 1 は，ArrivalSensorInterface から AbstractArrivalSensor への追跡に適用されている．ルール 2 は，各要素から ElevatorController への追跡に適用されている．ルール 3 は，ElevatorManager から ArrivalSensorUI への追跡に適用されている．

6. ツールの実装

協調クラス群抽出アルゴリズムを使用し，UML モデル要素に対応する Java 協調クラス群の抽出ツールとして Eclipse プラグインを実装した．ツールの画面を**図 7** に示す．

この画面は，開発プロセスのフェーズとフェーズ間のトレース，フェーズで作成された成果物を示している．例えばオブジェクト指向ソフトウェア開発方法論 COMET では，要求モデリング，分析モデリングなどのフェーズがあり，要求モデリングフェーズには，システムの要求を示すユースケース図が登録される．

右のサイドバーから add SourcePhase を選択すると，実装フェーズを追加することができる．実装フェーズは，実装した Java ソースコードを登録する．登録後，設計図が作成されたフェーズへトレースを張ると，そのフェーズ内の図面の各モデル要素に対応する，Java 協調クラス群を算出し表示する．

さらにダブルクリックして変更されるモデル要素を指定すると，依存関係付加・変更波及解析の機能を使用して，変更される可能性があるモデル要素を色を変えて表示する．これにより，あるモデル要素を変更した時に，どの Java クラスを変更すればよいかユーザに示すことが可能となる．

7. 評　価

協調クラス群抽出アルゴリズムの性能を評価するため，以下の 4 種類のプログラムを利用して実験を行った．1，2 のプログラムは文献からの引用であり，3，4 のプログラムは現在稼働中のプログラムである．

(1) **デザインパターン抽出実験**
(2) **エレベータ制御システムでの実験**
(3) **協調クラス群抽出プログラム自身での実験**
(4) **図書管理システムのプログラムでの実験**

デザインパターン抽出実験により，GoF の 23 個のデザインパターンにどれだけ対応しているか評価する．デザインパターンはモデルをクラスに写像するときに使用されるパターンであり，協調クラス群になる可能性が高い．そのためデザインパターンに対する高い抽出精度が望まれる．プログラムは文献 6) の事例を使用した．

エレベータ制御システムのプログラムは，文献 7) に記載されている事例である．汎化・実現の関係が多いことが特徴である．

協調クラス群抽出プログラム自身での実験は，小規模なプログラムの事例として行う．

図書管理システムは，文献の貸し出し，返却を管理するプログラムである．筆者の所属研究室で稼働中であり，1000 冊程度の蔵書を管理している．特徴としては汎化・集約・インスタンス化など多用な関係が存在することが挙げられる．

実験の評価には適合率と再現率という指標を用いる．アルゴリズムによって抽出したクラスの集合を A，正解のクラスの集合を B とすると，適合率 *Precision* と再現率 *Recall* の計算式は以下のようになる．

$$Precision = A \cap B \div A * 100$$

$$Recall = A \cap B \div B * 100$$

適合率は抽出したクラスのうちの正解クラスの割合，再現率は正解クラスのうち抽出できたクラスの割合を示す．両者を対象実験の各モデルについて求め，対象実験における適合率，再現率の平均を算出した．

実験の結果を**表 1** に示す．デザインパターン抽出実験は，1 つのプログラムではなく各パターン別のプログラムを使用したため，Java クラス数はパターン別プログラムの平均を表示している．

汎化・実現の多いエレベータ制御システムの例では，適合率，再現率ともに 100%であった．デザインパターンは適合率が 100%であり，再現率は 89.6%であった．モデル要素数，Java クラス数共に一番多く，実験の中で最も厳しい条件であると思われる図書管理システムの例では，適合率が 86.4%となり，再現率は 95.4%であった．

失敗事例は以下の 3 種類であった．

(1) Singleton パターンへの未対応
(2) コレクションクラスへの未対応
(3) 汎化関係の誤追跡

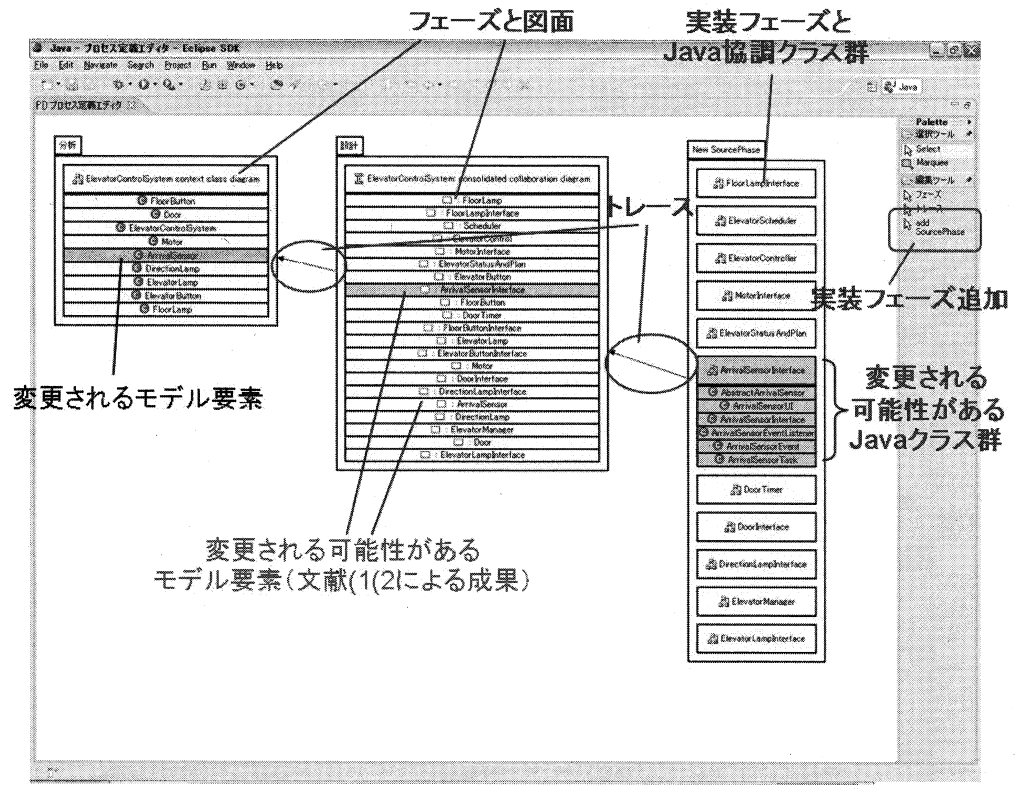

図 7 実装ツール画面

実験	モデル要素数	Java クラス数	適合率平均	再現率平均	特徴
デザインパターン抽出	23	5(各パターン平均)	100.0%	89.6%	デザインパターン
エレベータ制御システム	29	21	100.0%	100.0%	汎化・実現多
協調クラス群抽出アルゴリズム	10	19	92.5%	100.0%	小規模プログラム
図書管理システム	22	83	86.4%	95.4%	多用な関係

表 1 アルゴリズムの性能評価実験の結果

1 について，アルゴリズムは自分自身をインスタンス化するクラスを協調として含まないためである．自分自身をインスタンス化するクラスの参照を獲得した場合は，獲得したクラスから獲得されたクラスへのインスタンス化とみなすことで解決できる．

2 について，ArrayList などのコレクションクラスをフィールドとする場合，リストに追加されるクラスを集約として保持できないからである．Java1.5 より導入された Generics を使用して，リストに追加されるクラスを集約とすることで解決できる．

3 について，具象クラスがコアの場合，自分以外の具象クラスまで協調クラス群に含めたからである．これは，具象クラスから抽象クラスへ追跡した後，また別の具象クラスへ追跡してしまうことが原因である．これには，具象クラスから抽象クラスへ追跡した後，他の具象クラスへ追跡しないというルールを追加することで対応できる．

8. おわりに

本稿では，UML モデル要素と Java クラス群を対応付ける手法として，協調クラス群抽出アルゴリズムを考案し，Eclipse プラグインによるツールとして実装した．実験の結果，86%以上の適合率，90%以上の再現率を確認し，本手法が有効であることを確認した．

今後は，失敗事例への対応による精度の向上，オープンソースなど，より一般的な事例での検証，Java 言語以外へのオブジェクト指向言語への対応を進めていく予定である．

参 考 文 献

1) 小谷正行, 落水浩一郎, UML 図面群の変更波及解析に利用可能な依存関係の自動生成法, JAIST Research Report, IS-RR-2006-002, 2006.
2) 小谷正行, 菅井拓海, 落水浩一郎, UML 図の変更波及解析ツール, 電子情報通信学会, ソフトウェアサイエン

ス研究会, 2007.4.(発表予定)
3) 下村希世人, 片山徹郎, UML から Java ソースコードへの変換規則の抽出と変換ツールの試作, 電子情報通信学会, 信学技報, SS2005-57, pp13-18, 2005.
4) 堅田純也, 小林隆志, 佐伯元司, 静的解析と動的解析を用いたデザインパターン検出手法, 電子情報通信学会, 信学技報, SS2005-04, pp19-24, 2005.
5) David Bellin, Susan Suchman Simone, The CRC Card Book (Addison-Wesley Series in Object-Oriented Software Engineering), Addison-Wesley Pub (Sd), 1997. (邦訳：今野睦, 飯塚富雄, 桜井麻里, 実践 CRC カード ローププレイとブレーンストーミングによる大規模システム開発手法, ピアソンエデュケーション, 2002.
6) 結城 浩, Java 言語で学ぶデザインパターン入門, ソフトバンク パブリッシング株式会社, 2003.
7) Hassan Gomaa, Designing Concurrent, Distributed, and Real-time Application with UML, Addison-Wesley, 2000.