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

2005－SE－150 (13)
2005／11／29

メタパターンを用いたJavaソースコードにおける協調クラス群の抽出

金 旭東† 早坂 良† 小谷 正行† 落水 浩一郎†
†北陸先端科学技術大学院大学 情報科学研究科
〒923-1292 石川県能美市旭台1－1
E-mail:{xudong, ryoh, m-kotani, ochimizu}@jaist.ac.jp

概要 本論文では、オブジェクト間の参照関係をメタパターンを用いて解析することにより協調クラス群を抽出する方法を提案する。参照には協調に関与するものとそうでないものがある。協調関係の判定には、メタパターンの構成要素であるテンプレートメソッドとフックメソッド間の関係を用いる。テンプレートメソッドとフックメソッドの協調の型を定義し、それを利用して協調クラス群を抽出するアルゴリズムを開発する。簡単な例を用いて、抽出された協調クラス群の有効性を確認する。最後に、今後の課題について議論する。
キーワード：メタパターン、テンプレートメソッド、フックメソッド、協調クラス群

Extraction of Collaborating Classes Using Meta Patterns

Xudong Jin, Ryo Hayasaka, Masayuki Kotani, Koichiro Ochimizu
School of Information Science, Japan Advanced Institute of Science and Technology
Asahidai1-1, Nomi-city, Ishikawa, 923-1292 Japan
E-mail:{xudong, ryoh, m-kotani, ochimizu}@jaist.ac.jp

Abstract In this paper we propose the method that extracts collaborating classes by analyzing the references between classes using meta patterns. The references related to some collaboration is decided by the structural relationships between template methods and hook methods which are important components of a meta pattern. We show the effectiveness of our algorithm based on several experimental results.

1. はじめに

ソフトウェア共同開発においては、すでに作成した成果物を参照しつつ、新しい成果物を生成するため、成果物間にさまざまな依存関係が存在する。このことは開発時の修正作業やバグ発生に伴って生じる変更作業を困難にする。さらに、通常、成果物間の依存関係は図面やプログラムを管理するため、設計者やプログラマ自身により定義されるが、試行錯誤を伴う設計活動ではその管理は容易ではない。筆者等は、このような依存関係を自動生成することで、変更に要するコストの軽減を図る研究を進めている。具体的には、UMLおよびJava 言語を対象に、 図1に示すように、依存関係を自動抽出し、それを利用して変更作業支援のワークフローを自動生成する研究を行っている。すでにUMLモデリング要素間に依存関係を自動生成する研究については、図1の1、2、6、7について一定の成果を得ている [2]。

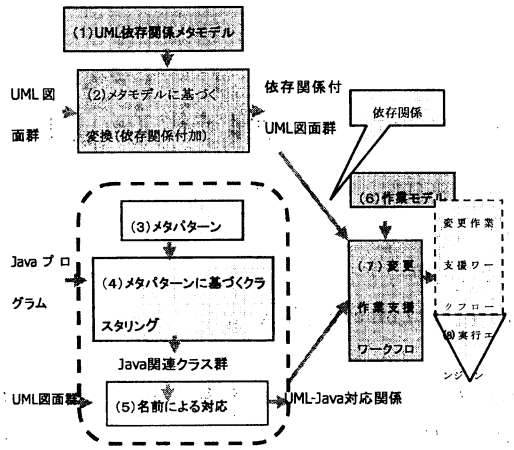


図1変更作業支援ワークフロー

本論文で報告する内容は、図1の3,4,5の部分、協調関係にあるJavaクラス群を抽出し、UMLのモデル要素と対応させる部分である。一般にUML図面中のあるモデリング要素を実装した場合、必ずしも単一のクラスとはならない。多くの場合、図2に示すように一個以上の協調するクラス群として実現される。

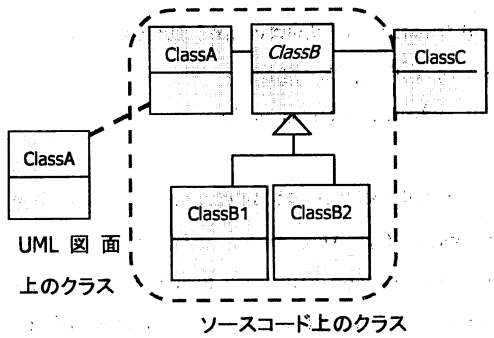


図2　設計時構造と実装時構造の差異

図2において、UML図面上のClassAを実現しているソースコード中の対応部分がClassA, *ClassB*, ClassB1, ClassB2であるとする。このときClassAの*ClassB*への参照を協調に関与すると判定し、*ClassB*のClassCへの参照は協調に関与しないと判定することが必要となる。「ClassAから生成されるオブジェクトがClassB1およびClassB2から生成されるオブジェクトの参照を持ちメッセージを送れる」ということと、「ClassB1、ClassB2がClassCへの参照を持ちメッセージを送れる」ことの間には違いはないが、「ClassAにテンプレートメソッドがあり*ClassB*にはフックメソッドがあるが、*ClassB*とClassCの間にはそのような関係はない」ということで協調に関与するか否かを判定する。

本稿では、上記の協調を解析の対象とする。このような協調構造はほとんどのデザインパターンが利用しているものである。

2．アプローチ

Preee[1]によれば、Gammaのデザインパターンは7つのメタパターンに分類される。我々は、さらに7つのメタパターンを、テンプレートメソッドとフックメソッドが協調しあう構造的な特徴により三つの代表的な型に分類した。この三つの代表的な型に対応するJavaの言語的特徴を整理し、Javaプログラム内の協調するクラス群を抽出する方針を取る。

本節では、まず、Preeによる、テンプレートメソッドとフックメソッドを用いたメタパターンの分類について説明する。

2.1　テンプレートメソッドとフックメソッドの定義

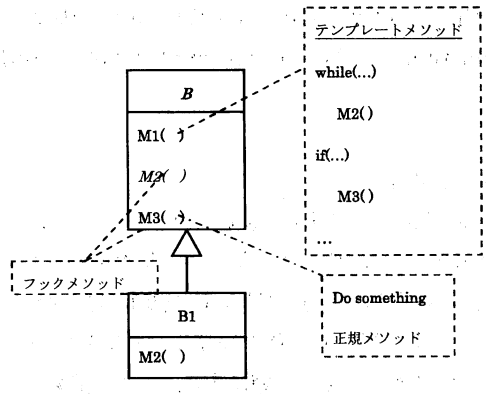


図3テンプレートメソッドとフックメソッド

図３の例を用いて、テンプレートメソッドとフックメソッドを説明する。

テンプレートメソッドとは：　具象メソッド、抽象メソッド、フックメソッドの任意の組み合わせを呼び出すメソッドで、その詳細を実装することなくアルゴリズム輪郭だけを決定するメソッドである。図3におけるM1は抽象メソッドと具象メソッドの組み合わせを呼び出す例である。

フックメソッドとは：サブクラスでオーバーライトされることを想定したメソッドである。抽象メソッド、具象メソッド、テンプレートメソッドのどれでもフックメソッドになりうる。図3において、クラスBのM2はクラスB1でオーバーライトされている、クラスBのM3はオーバーライトされてないがその可能性はある。

フックメソッドを含むクラスを、対応するテンプレートメソッドを含んでいるクラスのフッククラスと呼ぶ。テンプレートメソッドを含むクラスをテンプレートクラスと呼ぶ。以後、テンプレートクラスを　Tと記し、フッククラスを　Hと記す。

2.2　メタパターンの分類(Pree [1])

Pree によるメタパターンの分類を❶から❼に示す。分類において結合、再帰という用語が使用されているが以下の意味を持つ。

TとHを上の３つの参照方法により結び付けることを結合といい、以下の３つに分類できる。

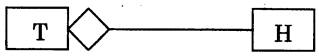
❶　T型のクラス中で、H型のインスタンスを生成する場合。

❷　メソッドの引数としてオブジェクトの参照を渡す場合。

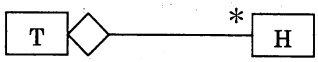
❸　グローバル変数によってクラス H のオブジェクトを参照する場合。

クラスからそのクラス自身への参照を行うことを再帰という。

❶　Tのオブジェクトが Hのちょうど一つのオブジェクトを参照する場合１：１結合メタパターンという。

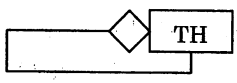


❷　Tのオブジェクトが Hのいくつかのオブジェクトを参照する場合は１：N結合メタパターンという。

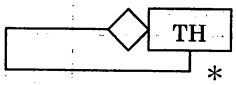


❸　TとHが一つのクラスからなっているがテンプレートメソッドとフックメソッドが同じ名前ではない場合は統合メタパターンという。

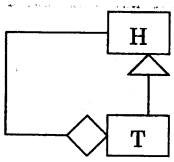
❹　TとHが統合され、対応するテンプレートメソッドとフックメソッドも同じ名前のメソッドに統合されているようなただ一つメソッドだけが存在する、Tのオブジェクトが Hのちょうど一つのオブジェクトを参照する場合１：１再帰的統合メタパターンという。



❺　Tのオブジェクトが Hのいくつかのオブジェクトを参照する場合は１：N再帰的統合メタパターンという。

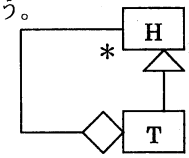


❻　Tのオブジェクトが Hの一つのオブジェクトを参照する場合１：１再帰結合メタパターンという。



❼　Tのオブジェクトが Hのいくつかのオブ

ジェクトを参照する場合１：N 再帰結合メタパターンという。



2.3 メタパターンによる Gof デザインパターンの分類

上記のメタパターンを利用した Gof のデザインパターンの分類結果を表１に示す。表１において、3 つのデザインパターン（Façade, Singleton, Memento）における協調は本稿の検討対象外となり今後の課題である。

また、「１：N 再帰的統合メタパターンに属するデザインパターン」は存在しないので実際には６つのメタパターンが利用することになる。

3. メタパターンによる協調クラス群の抽出

テンプレートクラスのオブジェクトがフッククラスのオブジェクトをいくつ参照するかについては、協調に関与する参照を判断する立場からは無関係なので注目しない。

このような視点からすると、構造的特徴を３つにまとめることができる（表２）。

表２メタパターンから３つ代表的な構造へ分類

メタパターン	三つの協調構造
統合メタパターン １：１再帰的統合メタパターン	統合的協調構造
１：１結合メタパターン １：N 結合メタパターン	結合的協調構造
１：１再帰的結合メタパターン １：N 再帰的結合メタパターン	再帰的結合協調構造

なお、表 2 において統合メタパターンは、統合的協調構造の特殊な場合とする。

表１　メタパターンによる Gof デザインパターンの分類表

メタパターン	GoF デザインパターン (23)
統合メタパターン	TemplateMethod, Factory Method
１：１結合メタパターン	Builder, Bridge, State, Strategy, Mediator, Visitor
１：N 結合メタパターン	AbstractFactory, Command, Proxy, Prototype, Adapter, Flyweight, Observer, Iterator
１：１再帰的結合メタパターン	Decorator
１：N 再帰的結合メタパターン	Composite, Interpreter
１：１再帰的統合メタパターン	Chain of Responsibility
１：N 再帰的統合メタパターン	なし
メタパターンを含まない	Façade, Singleton, Memento

3.1 3 つの協調構造から見たクラス間の協調関係

（１）**統合的協調構造**：図３に統合的協調構造の一例を示す。統合的協調構造においては、テンプレートメソッドとフックメソッドは、一つのクラスに存在する。

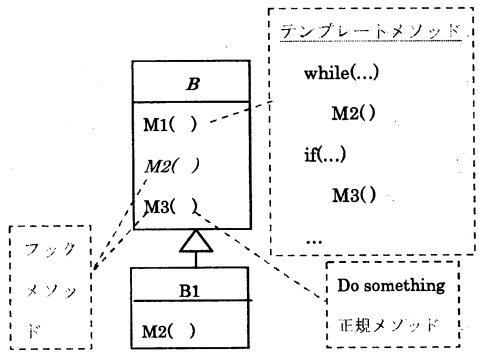


図３　統合的協調構造

（2）結合的協調構造：図4に結合的協調構造の一例を示す。テンプレートメソッドとフックメソッドが異なるクラスに存在する。

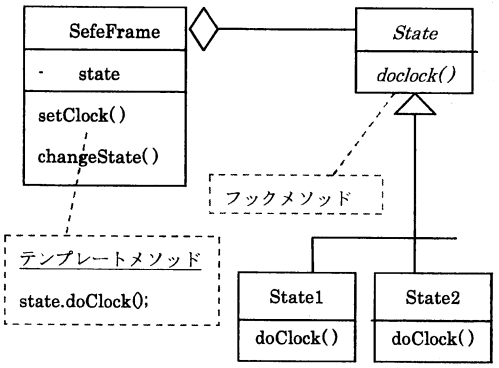


図4　結合的協調構造

（3）再帰的結合協調構造：図5に再帰的結合協調構造の一例を示す。テンプレートメソッドとフックメソッドが異なるクラスに存在し、テンプレートクラスとフッククラスの間に再帰的関係がある。

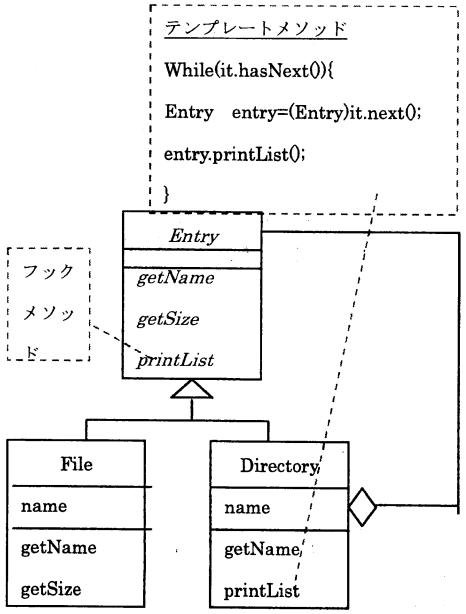


図5　再帰的結合協調構造

4.　協調クラス群を抽出するアルゴリズム

本節では、協調関係の判定法とJavaソースコード中のすべてのクラスに対し協調関係があるクラス群を抽出するアルゴリズムを図6に示す。

4.1　協調関係の判定法

図6の下半分は、テンプレートメソッドとそれに対応するフックメソッドが存在するか否かにより協調関係の確認を行う部分である。

❶ テンプレートメソッドの探索

（1）　Javaソースコード中の任意のあるクラスについて、まず、継承関係があるか否かをチェックする。継承関係がない場合はそのクラスのメソッドが同じクラスのメソッドを利用しているか、または、ほかのクラスのメソッドを利用しているかをチェックする。

ある場合は：　ステップ❷でフックメソッドの探索を行う。ない場合は：　協調関係がないと判定する。

（2）　継承によりグループになったクラス群に対して、グループのすべてのクラスのメソッドに対して（1）と同じようなチェックを行う。

❷ フックメソッドの探索

（1）　同じクラス内のメソッドを利用した場合、利用されたメソッドがそのクラスの子孫クラスにオーバーライトされていないかをチェックする。オーバーライトされていたら：協調関係があると判定する。オーバーライトされていなかったら（❶－（2）の場合のみ）：❶－（2）に戻り探索を続ける。

（2）　ほかのクラスのメソッドを参照する場合：参照されたメソッドがオーバーライトされてないかをチェックするそうであれば：協調関係があると判定する。

そうでなければ（❶－（2）の場合のみ）：❶－（2）に戻り探索を続ける。

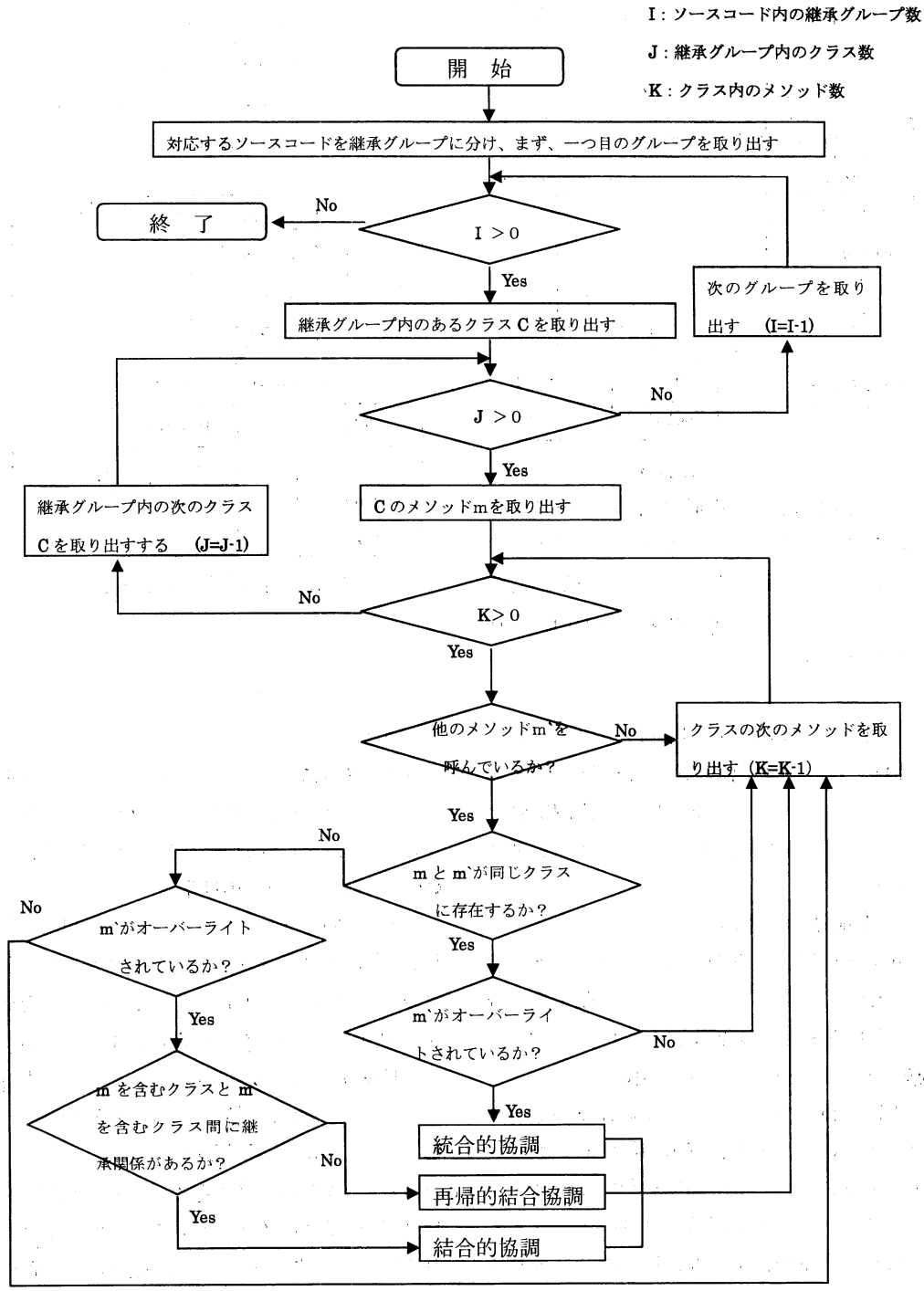


図6　協調クラス群を抽出するアルゴリズム

5. 抽出実験

5.1 GoF のデザインパターンの抽出

図 6 に示したアルゴリズムの妥当性を確認するため、まず、デザインパターンに適用してみた。結果を表 3 に示す。Factory Method、 Flyweight, Abstract Factory については抽出できなかった。Factory Method と Abstract Factory については、生成されるオブジェクト群を併せて解析する必要があり、また、Flyweight については今後の課題である。

表 3 Gof デザインパターンの抽出結果

メタパターン	GoF のデザインパターン	抽出可・否
統合メタパターン	Template Method, Factory Method	抽出可能 未対応
1:1 結合メタパターン	Builder, Bridge, State, Strategy, Mediator, Visitor	抽出可能
1:N 結合メタパターン	Prototype, Adapter, Observer, Iterator, Proxy, Command, Flyweight, Abstract Factory	抽出可能 未対応
1:1 再帰的結合メタパターン	Decorator	抽出可能
1:N 再帰的結合メタパターン	Composite, Interpreter	抽出可能
1:1 再帰的統合メタパターン	Chain of Responsibility	抽出可能

5.2 一般的な協調クラス群の抽出

デザインパターンを含む一般的な協調関係の抽出能力を確認するため、文献4で与えられたエレベータ制御システムの仕様に基づくソースコードを開発し本アルゴリズムで解析した。

その結果、図8において、ElevatorController クラスと DoorClosingToMoveUp クラスに関してそれぞれ協調クラス群を抽出できた。

ElevatorController クラスと協調するクラス群は、図 8 において黒く塗りつぶしたクラス群であり、State パターンに対応する。

DoorClosingToMoveUp クラスと協調するするクラス群として図 8 全体が抽出される。これは、デザインパターン以外の協調が抽出された例である。

この結果から、本アルゴリズムが一般的な協調クラス群の抽出にも適用できることがわかる。

5.3 異なる実装による異なる協調クラス群

すべての協調クラス群を抽出するためには、異なる実装による異なる協調クラス群の抽出も可能でなければならない。図8とは実装の構造が異なるソースコードを作成し解析してみた。図 8 のプログラムをシステムの機能に影響を与えない前提

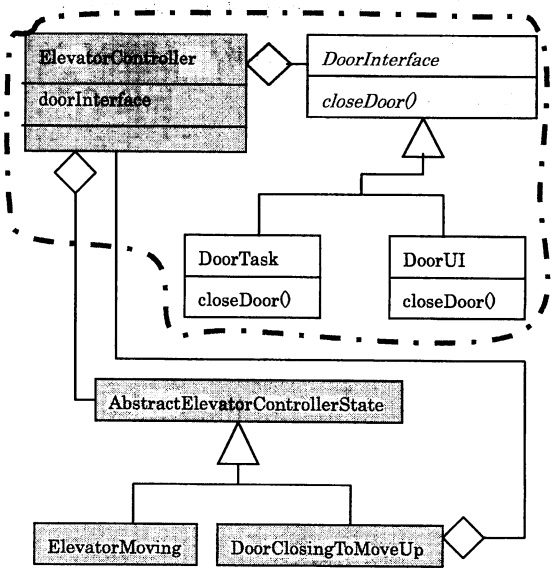


図 8 抽出された協調クラス群

で、以下のように書き換える(図8の一点鎖線で囲んだ部分)。元のソースコードでは、

DoorClosingToMoveUp クラス ElevatorController クラス中の *DoorInterface* のインスタンス変数を通じて *DoorInterface* のサブクラスを参照することで *DoorInterface* にメッセージを送るようになっていた。これを DoorClosingToMoveUp が ElevatorController へメッセージを送るように委託し、そして ElevatoerController で DoorInterface メッセージを送るように構造を変えることも可能である。この場合も、適切な協調クラス群が抽出された。

6．まとめと今後の課題

本論文では、メタパターンを用いて協調に関連する参照のみを取り出すアルゴリズムを提案した。テンプレートメソッドとフックメソッド間の構造的関係を利用して有効な協調構造を抽出できることを確認した。

今後は、異なる協調構造も検討する予定である。

謝辞

本研究は、文部科学省科研費特定領域研究（２）課題番号 16016239 の補助の元に実施された。記して謝意を表する。

文　献

[1] Wolfgang Pree, “Design Patterns for Object-Oriented Software Development”by the ACM Press, a Division of the Association for Computing Machinery, Inc.(ACM). 1995.

[2] 小谷 正行, 落水 浩一郎, ″依存関係を用いた UML 文書間の波及解析法″, 電子情報通信学会ソフトウェアサイエンス研究会, SS2004-62, 2005.03.

[3]堅田 淳也,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, 2005-4.

[4] Hassan Gomma, “Designing concurrent, distributed, and real-time application with UML”, Addision Wesley, Inc. 2000.