継承異常を軽減するための並行性の抽象化

4 E − 1

Andrew E. Santosa　河田 恭郎　前川 守

電気通信大学大学院情報システム学研究科

1 はじめに

並行オブジェクト指向プログラミング言語における継承異常問題 [3] の解が既に多く提案されている。最近の提案の中には、親クラスのメソッド本体または、ガードや相互排除制御式を含む、メソッドの起動条件への参照などの単純で低レベルな再利用機構を用いるものが多い [1, 4]。それらの機構はコードの持つ意味を反映しにくいため、プログラミングが困難になる。そこで、より抽象度の高い機構が求められる。

継承はあるクラスで記述される振舞いにさらに新しい振舞いを追加するために行う場合が多い。ここでいう振舞いとは、オブジェクト内の状態の変化とその変化を起こすための、外界とのインターフェースを表すものであり、状態遷移図として表現することができる。我々は三つの振舞いについて話しを進める。第一の振舞いは親クラスの振舞いで、第二の振舞いは親クラスの振舞いに新しく追加する振舞いで、第三の振舞いは第一と第二の振舞いを合わるような、子クラスの振舞いである。それぞれの振舞いは状態遷移図として表現することができる。本稿では、第一の振舞いを表す状態遷移図と第二の振舞いを表す状態遷移図をある手順を用いることによって合成し、第三の振舞いを表す状態遷移図を生成する。その手順を行なう際には、第一の振舞いを表す状態遷移図の部分のどんな再定義も必要としないため、継承異常は起こらない。第一と第二の振舞いを表す状態遷移図をそれぞれクラス定義に置き換えても、状態遷移図を合成する際と同様の手順を利用すれば、いかなる再定義も含まないような子クラスの定義ができる。

2 状態遷移図の合成による継承異常の解決

有限バッファを表現するクラス b_buf を最初に定義しよう。b_buf は put と get メッセージに対応するインターフェースを持つ。put はバッファにアイテムを一つ入れるように要求する。一方、get はバッファからアイテムを一つ取り出すように要求する。ただし、put はバッファが一杯のときに受理できない。同様に、get はバッファが空のときに受理できない。b_buf のインスタンス内における状態の変化を表す状態遷移図を図 1に示す。

get と同じ要求を持つが、put が処理された直後には受理できないようなメッセージへのインターフェースを有限バッファに追加したいとする。そのために b_buf から子クラス gb_buf* を派生する。gb_buf では gget メッセージへのインターフェースが新しく定義される。gget の内容は get のそれと同じだが、gget は put の直後に受理できない。いくつかの言語では、各メソッド本体に、put が最後に実行されたかどうかを表すフラグを設定しなければならないため、再定義

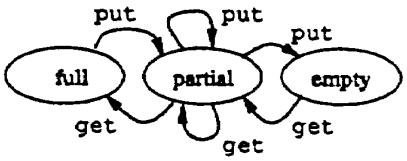

図 1: b_buf オブジェクトの状態遷移図

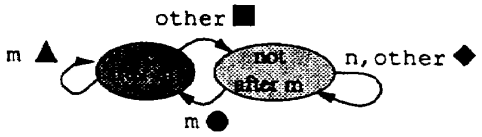

図 2: gb_buf に追加したい振舞い

が必要になり、継承異常が起こってしまう。

gb_buf に追加したい振舞いの仕様は次のようである：

1. オブジェクトに到達するメッセージを m と n と「その他のメッセージ」に分類する。
2. n として分類されるメッセージは m として分類されるメッセージが処理された直後に受理できない。

この振舞いを状態遷移図で表すと図 2†のようになる。このように、新しく追加したい振舞いを状態遷移図として定義できることを示した。

次に、図 1と図 2とを一つの状態遷移図に統合させることによって、gb_buf の振舞いを表現する状態遷移図を構成する。統合は次のように行なう：

1. put は m のような振舞いを持つことになるので、図 1の put を図 2の m に対応させる。
2. gget を図 1の上に定義する。gget は get と同じ処理を要求するため、状態遷移図上では、図 1の get と同じ弧に対応する。
3. gget を、m の直後に受理できない図 2の n に対応させる。
4. 図 1の get を、m と n 以外の遷移を表す図 2の other に対応させる。

プログラマが上の操作を行った後、システムがステートチャート [2] の OR 分割に似た仕組みで構造化された図 3を自動的に生成することができる。ここで説明した状態遷移図の統合の際の手順を利用すれば、どのような再定義も必要ないため、継承異常を解決できる。

図 2のメッセージ識別子が図 3で表されなくなるので、図 2と図 3の間の各アークの対応関係はそれぞ

A Solution to Inheritance Anomaly Based on Composition of State Transition Diagrams

Andrew E. Santosa, Yasuro Kawata and Mamoru Maekawa
Graduate School of Information Systems,
University of Electro-Communications
1-5-1 Chofugaoka, Chofu-shi, Tokyo 182-8585, Japan

*[3] では、他にもいくつかの継承異常の例が示されたが、紙面が限られているので、ここでは gb_buf の例のみを取り上げる。

†図 2では、■、▲などのような図形があり、これらの図形を後で説明する。

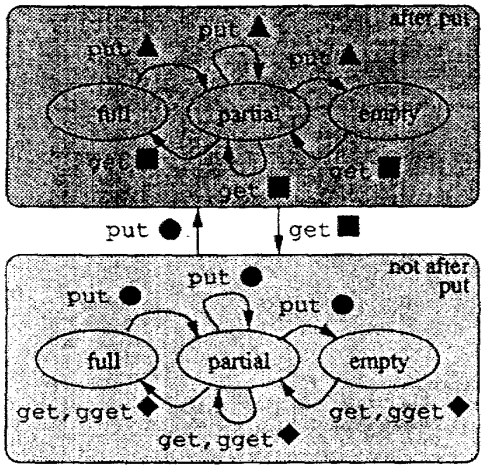

図 3: gb_buf オブジェクトの状態遷移図

```
class b_buf: Concurrency {
   sequential_bounded_buffer buf;
public:
   void put(int x) when (!buf.full()) {
      buf.put(x); }
   int get() when (!buf.empty()) {
      return buf.get(); }
};
```

図 4: b_buf クラス

れの図のメッセージ識別子の右に描かれている図形(■、▲など)で表される。

3　プログラミング言語への展開

本論文ではガード付きメソッドを支援する疑似並行オブジェクト指向プログラミング言語を前提とする。この言語では一つのオブジェクト内に複数のスレッドが存在することはできるが、常に一つ以下のスレッドしか実行中の状態にない。言語の構文は C++ に似た構文を使用する。

図 1の状態遷移図で表される振舞いを図 4の b_buf[‡] クラス定義に変換することができる。また、図 2で表現される振舞いを図 5の after_m クラスとしても表現することができる。after_m のメソッドは transitions 節で定義される。それらのメソッドはサービスを外界に提供するのではなく、振舞いを表現するためだけにあるので、引数も戻り値も持たない。

b_buf を継承し、かつ、図 3で示す振舞いを持つ子クラス gb_buf を次に定義する。図 1と図 2から図 3を生成する際の、メッセージとメッセージを対応させるステップを gb_buf クラスにコーディングする。結果は図 6に示す、behaves_like 式で構成される gb_buf の定義である。behaves_like 式はメッセージの対応関係を表す[§]。図 6では、b_buf の定義のどの部分の再定義も必要としないため、ここでも継承異常は起こら

```
class after_m {
   boolean flag;
transitions:
   m { flag = True; }
   n when (!flag) { flag = False; }
   default other { flag = False; }
};
```

図 5: after_m クラス

```
class gb_buf: b_buf, after_m {
public:
   gget is get behaves_like n;
   put behaves_like m;
};
```

図 6: gb_buf クラス

ない。

behaves_like は内部的にメソッドのコードの統合 [5] を行う。再利用には子クラスのメソッドから親クラスのメソッドの部分 (ガードまたは本体) への単純な参照を用いるのではなく、より抽象度の高いメソッド統合機構を用いるため、他の言語と比べれば、継承異常をより簡単に解決することができる。

4　結論

本稿では、クラスで記述される振舞いを表す状態遷移図と新しく追加したい振舞いの状態遷移図を特殊な手順によって合成することで継承異常を解決できることを示した。同様の原理をプログラミング言語に適用することによって抽象度の高い、継承異常の解決法を得る。

参考文献

[1] Gianpaolo Cugola and Carlo Ghezzi. CJava: Introducing concurrent objects in Java. In *Proceedings of the 4th International Conference on Object-Oriented Information Systems (OOIS '97)*, 1997. Brisbane (Australia), 10–12 November 1997.

[2] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, Vol. 8, pp. 231–274, 1987.

[3] Satoshi Matsuoka and Akinori Yonezawa. Analysis of inheritance anomaly in object-oriented languages. In Gul Agha, Peter Wegner, and Akinori Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*, chapter 4, pp. 107–150. MIT Press, 1993.

[4] S. E. Mitchell and A. J. Wellings. Synchronisation, concurrent object-oriented programming and the inheritance anomaly. *Computer Languages*, Vol. 22, No. 1, pp. 15–26, April 1996.

[5] Andrew E.Santosa, 河田恭郎, 前川守. 継承異常を解決するためのメソッド統合機構. Submitted to Joint Symposium on Parallel Processing (JSPP'98).

[‡]詳細を省くために、b_buf は単なる、逐次的に実装された有限バッファのクラスの sequential_bounded_buffer への並行インターフェースとして実装した。

[§]図 5では other が default として定義されたため、図 6では get と other の対応を明示的に表現する必要がない。