4N-2

C++でのMixinにおける静的なエラー検出

今関 雄人† 高田 眞吾† 土居 範久‡†
† 慶應義塾大学理工学部情報工学科 ‡ 中央大学理工学部情報工学科

1 はじめに

ソフトウェア開発において，既存の部品を組み合わせて新しいソフトウェアを構築する方法が用いられている．本研究では，各種構築方法のうち，モジュールの合成方法として提案されたMixin [1] に着目する．

Mixinは，複数のモジュールから必要なものだけを組み合わせて合成することができるという柔軟性を持つ．このため，モジュールの組み合わせ方を変えることにより，共通の機能を持つ複数のソフトウェアを開発することが可能となる．ここで，モジュールを合成することをMixinと呼ぶ．

Mixinを実現する手法として，C++のテンプレートを用いた手法が提案されている [2]．しかし，この手法には，クラスメンバ名の衝突，タイプチェック機能の欠如，モジュールの意味的衝突という問題がある．これらの問題点はバグが発生する原因となっており，Mixinの実用を困難にしている．

そこで，本研究では，上記の問題点により生じるバグの検出と修正の支援を行うツールを提案する．

2 C++におけるMixin

2.1 概要

クラステンプレートを使用することで，C++でMixinを実現することができる．図1にMixinの例を示す．この図の右側のBaseはベースとなるクラスであり，左側のDiffは差分となるクラスである．そして，下部では，BaseにDiffをMixinした，Finalというクラスを定義している．このFinalは，Base::AとDiff::Aを合成したクラス，B, Cの3つのクラスを持つクラスとなる．

クラスDiffはテンプレート引数Superを継承している．このように，スーパークラスをパラメータ化することでMixinを実現することができる．

さらに，この図のように入れ子クラスを用い，内側のクラスのスーパークラスが外側のクラスのスーパークラスにより決定されるようにすることで，1つ以上のクラスおよびクラス断片のカプセル化を実現することができる．

```
class Base {
   class A {
      ...
   };
   class B {
      ...
   };
};
```

```
template <class Super>
class Diff : public Super {
   class A : Super::A {
      ...
   };
   class C {
      ...
   };
};
```

```
// BaseにDiffをMixinしたクラスFinalを定義
class Final : public Diff < Base > {};
```

図 1: C++でのMixin

2.2 問題点

C++でのMixinには，以下の3つの問題点がある．

1. クラスメンバ名の衝突
   C++でのMixinは，実質は継承と同じであり，同時にMixinされるモジュール間で，クラスメンバ名の衝突が発生する可能性がある．この衝突はコンパイラによって検出されず，バグが発生する原因となる．

2. タイプチェック機能の欠如
   クラステンプレートを使用しているため，どのようなモジュールでもMixinできてしまう．これは，あるモジュールをMixinするのに，特定のモジュールが既にMixinされている必要がある場合などに，そのチェックができないという問題がある．

3. モジュールの意味的衝突
   複数のモジュールを選択して組み合わせることができるため，あるモジュールを追加することにより，他のモジュールが正常に動作しなくなってしまうという問題が生じる可能性がある．

3 静的なエラー検出ツールの提案

3.1 クラスメンバ名の衝突検出

クラスメンバ名の衝突については，Mixinされている各クラスのメンバ名を比較し，衝突しているものを検出し，警告もしくは自動的な書き換えを行うことによって解決することができる．

"A Static Error Detection Tool for C++ Mixin",
Yuto Imazeki†, Shingo Takada† and Norihisa Doi‡†,
† Keio University, ‡ Chuo University

C++では，継承の際にデータメンバやメンバ関数名が衝突していても，別の変数・関数として扱われる．このため，private 同士が衝突していても問題は発生しないなど，衝突しているメンバのアクセスレベルによって異なる対処が必要となる．

また，メンバ関数はオーバーライドが行われるため，データメンバの場合と異なる対処が必要となる．そこで，本ツールでは，仮想関数はオーバーライドされ，非仮想関数はオーバーライドされないと判断する．

メンバ名衝突についてのアクセスレベルごとの対処法を，表 1 にまとめた．

表 1: クラスメンバ名衝突の対処

		差分		
		public	protected	private
ベース	public	×/○	×/×	×/×
	protected	×/△	×/○	×/×
	private	△/△	△/△	○/△

この表で，'/' の左側がデータメンバ・非仮想関数名の対処，右側が仮想関数名の対処を表している．また，○は問題なし，△は警告の表示，×はエラーの表示 (名前の書き換え支援) を行うことを表している．

この機能により，名前衝突による問題を回避することができる．さらに，名前衝突を警戒するための労力の軽減を行うことができるという利点がある．

3.2 タイプチェック機能

タイプチェックを行うために，まず前提となるモジュールを開発者が記述するための構造を与える．

ここで，C++自体の拡張は行わないため，前提はコメントとして記述する．記述する内容は，そのモジュールを Mixin する際に必要となる，具体的なモジュール名である．例えば，以下のように記述する．

```
template <class Super>
//@--$type Super : Mixin1, Base
class Mixin2 : public Super {
    ...
};
```

この記述の 2 行目で，`Mixin2` クラスのパラメータである `Super` は，その継承階層内に `Mixin1` と `Base` を含んでいなければならないことを定義している．

本ツールは，この定義を調べることにより，前提違反の検出を行う．

この機能により，コンパイラによっては検出できないエラーを検出することができる．さらに，インタフェースが異なる場合など，コンパイラによって検出されるエラーに関しても，どのモジュールの Mixin を忘れているかが分かるため，ミスを修正するのが容易であるという利点がある．

3.3 モジュールの意味的衝突検出

本ツールでは，super 呼び出し (オーバーロード前の関数の呼び出し) に着目し，あるモジュールによるオーバーライドが他のモジュールによるオーバーライドを無効にしている場合について検出を行う．

例えば，`Mixin2 < Mixin1 < Base> >` のように Mixin を行っていて，各モジュールである関数がオーバーライドされているケースを考える．ここで，`Mixin2` で super 呼び出しが行われていない場合，`Mixin1` でのオーバーライドは無効となってしまう．

そこで，本ツールでは，オーバーライドを行っている各関数について，super 呼び出しを行っているかどうかのチェックを行うことで，無効になっているものを検出する．上の例では，`Mixin2` が `Mixin1` を無効にしていると警告を行う．

また，`Mixin1 < Mixin2 < Base> >` のように Mixin すれば無効になっているものがなくなる．このため，このような候補が存在する場合には報告することでユーザへの支援を行う．

この機能により，super 呼び出しに関連する意味的衝突の検出を行うことができ，バグの早期発見・修正を行うことができる．

しかし，super 呼び出しを行っていたとしても，ベースクラスのデータメンバへのアクセスなどによりメンバ関数の事前条件などが変化してしまい，上手く動作しなくなってしまう可能性がある．この問題に関しては本ツールで検出することはできない．対処法としては，メンバ関数の事前事後条件を検査してアサートを行うモジュールの導入などが考えられる．

4 まとめ

本研究では，C++での Mixin において，メンバ名の衝突検出，タイプチェック，意味的衝突検出の 3 つを行う開発支援ツールの開発を行った．

このツールを利用することで，C++で Mixin を使用するソフトウェア開発において，開発者の負担を軽減することができる．

参考文献

[1] Gilad Bracha, William Cook: “Mixin-based Inheritance”, Proceedings of the European Conference on Object Oriented Programming on Object Oriented Programming Systems, Languages, and Applications, pp.303-311 (1990).

[2] Yannis Smaragdakis, Don Batory: “Mixin-Based Programming in C++”, Generative and Component-Based Software Engineering Symposium, Lecture Notes in Computer Science, pp.163-177 (2000).