アナリシスパターン勉強会
11章　トレーディング・パッケージ

2004.04.16

担当：宮下

資料作成支援：白川

目次

概要

- 大規模システムを扱うときに行う「分割」に必要なパッケージと可視性の概念を学ぶ。
- ここでは層別アーキテクチャーで「問題領域モデル」に当たる領域の分割を対象とする。
- 例として「トレーディングの概念」を取り上げ、3つのパターンを考察し理解をする。
  - 11.1 パッケージへの多重アクセスレベル
  - 11.2 相互可視性
  - 11.3 パッケージのサブタイプ化

11.1 パッケージへの多重アクセスレベル

―問題提起―

- ポートフォリオ、シナリオ、市場指標のパッケージ化を取り上げる。このときに、シナリオパッケージが2つの型を必要とし**多重可視性の問題**が発生する。直接的な公開／非公開アプローチでは必要とされる可視性が異なるためうまくいかない。

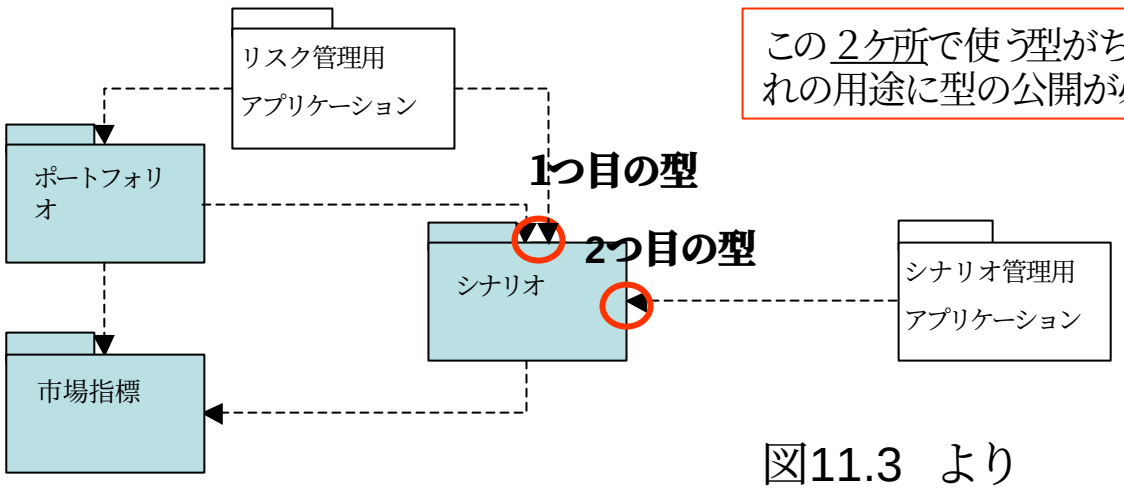


図11.3 より

11.1 パッケージへの多重アクセスレベル

―問題提起 :補足1―

- シナリオパッケージ(図11.1)について考えると。。。

まずポートフォリオに対して公開する型がある。→1つ目の型

このとき、図11.2のようなインターフェイスを用意するのが良い方法であるが。。

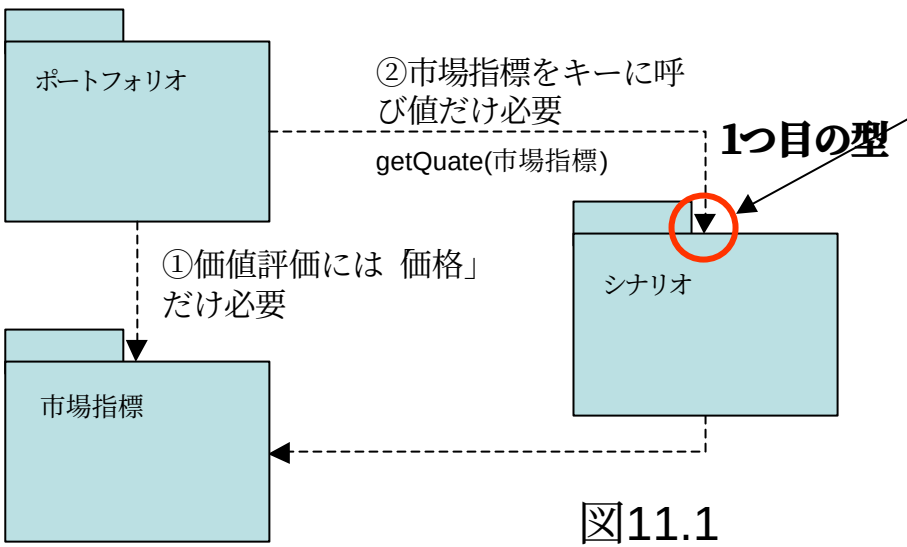


図11.1

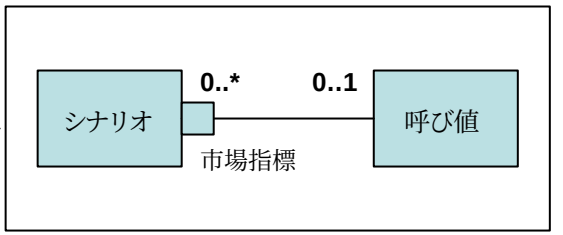


図11.2

ポートフォリオへのインターフェイス。

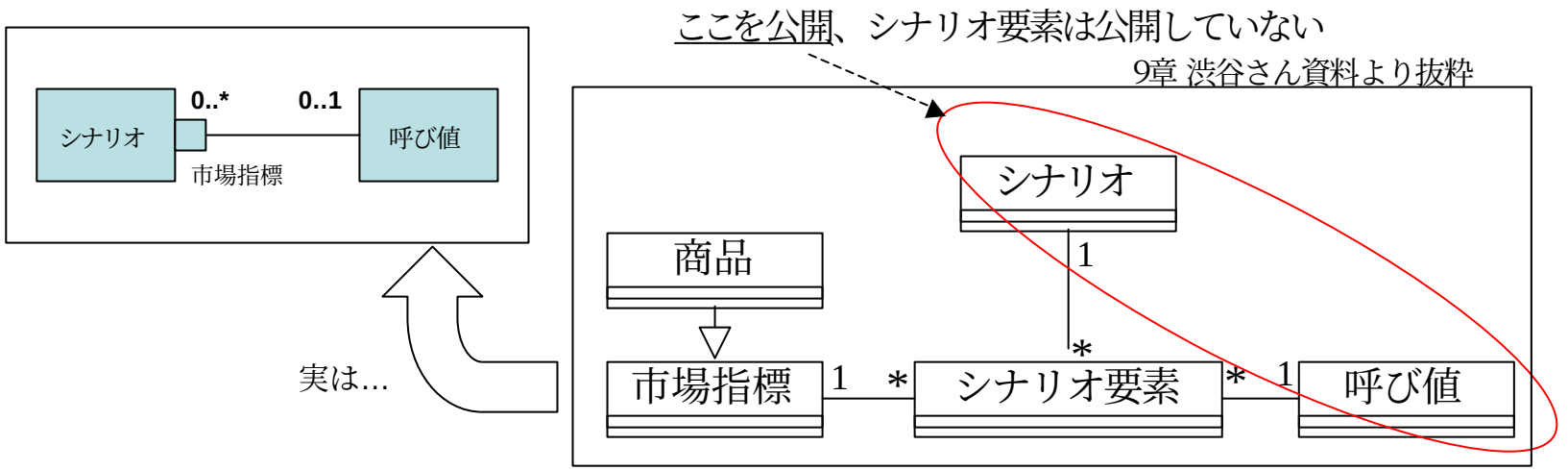
ポートフォリオからはシナリオ要素を知らなくても呼び値を獲得できる。

実際はシナリオ要素を介して呼び値を獲得している。

11.1 パッケージへの多重アクセスレベル

―問題提起 :補足2 ―

- シナリオ要素を非公開としたので。。。
  - パッケージの内部の型を公開型、非公開型に分割する方法があり、シナリオではポートフォリオに対しては、シナリオ要素を非公開（下図参照）としているが・・・



11.1 パッケージへの多重アクセスレベル

―問題提起 :補足3 ―

- シナリオの設定にはシナリオ管理アプリケーションが必須であるため、シナリオは新たに公開型(ここではシナリオ要素？)の追加が必要となる。
  - ということは、保守の難しさの問題が残ってしまう。

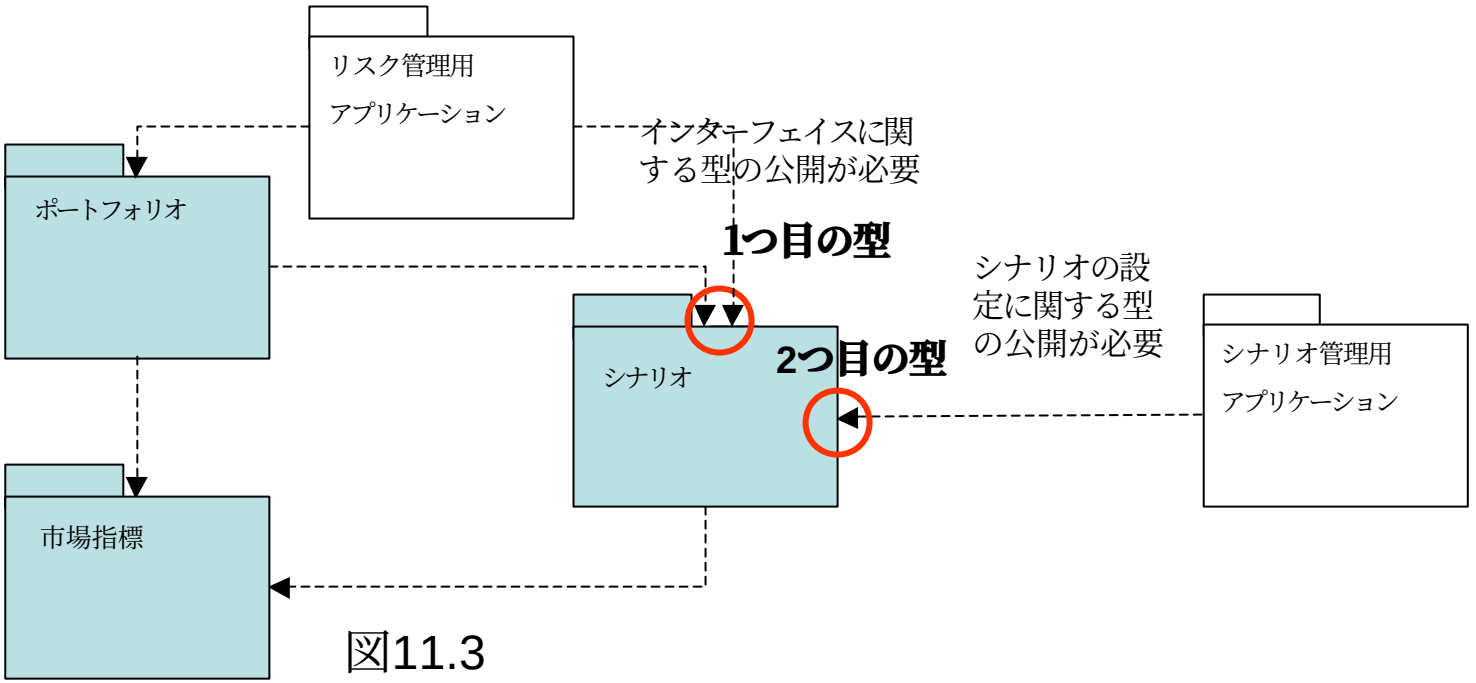
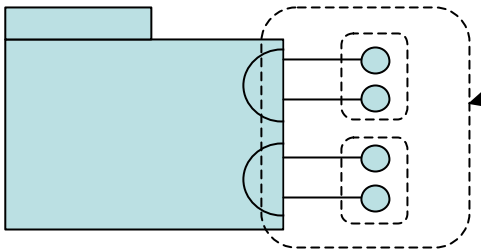


図11.3

11.1 パッケージへの多重アクセスレベル
―解決方法 その1―

- 分離プロトコルを用いる。
  - Wirfs-Brockによる提案 :1つのパッケージに複数のプロトコルを許す



「プロトコル」とは、こんなイメージ?

- Wirfs-Brockは、 契約」という語を使用
- 操作ではなく、型の集合としてもよい
- 半円形であらわす。

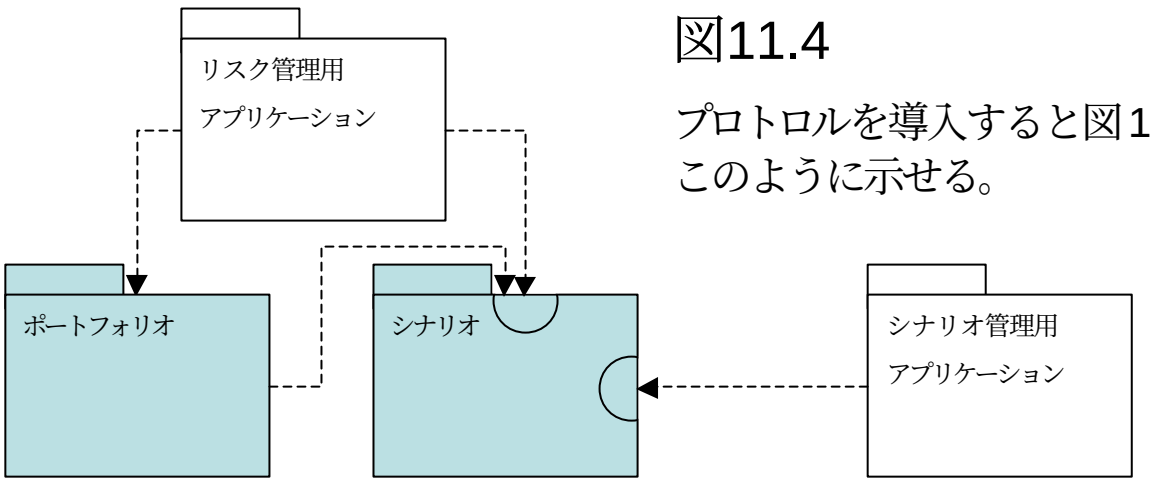


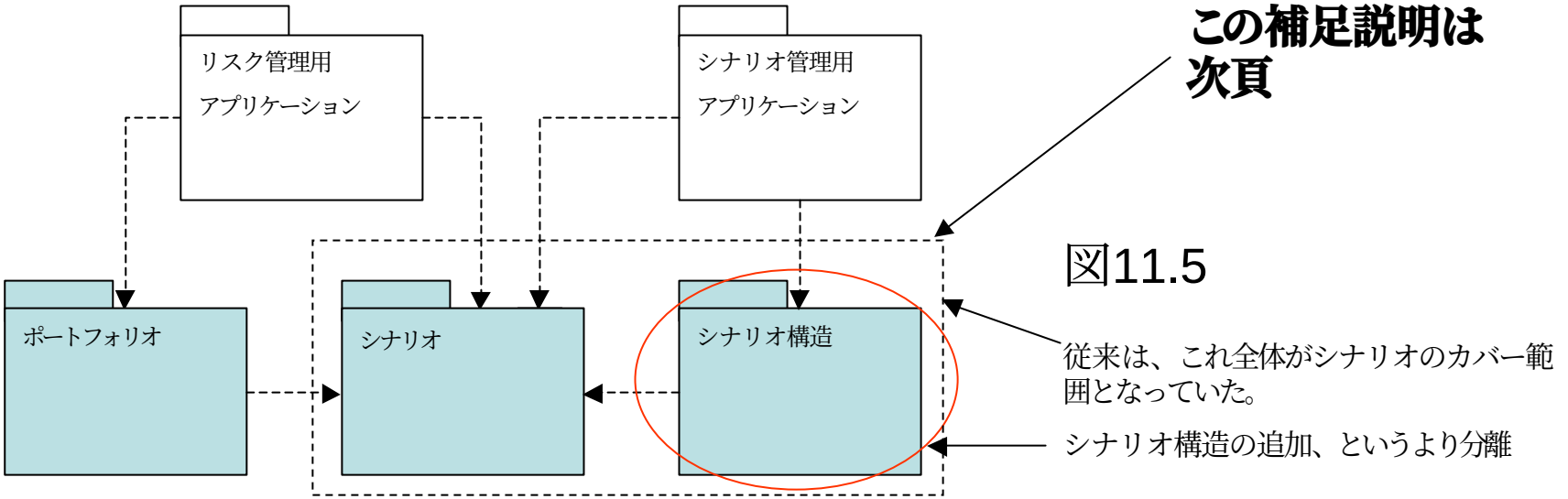
図11.4

プロトロルを導入すると図11.3は、
このように示せる。

11.1 パッケージへの多重アクセスレベル

―解決方法 その2―

- 追加パッケージを導入する。
  - シナリオ構造パッケージの追加（図11.5）
    - シナリオパッケージ：シナリオ型とその単純な関連のみ。（いわば抽象クラスを置く）
    - シナリオ構造パッケージ：シナリオ要素とそのサブタイプ
  - シナリオには、可視性を追加する。

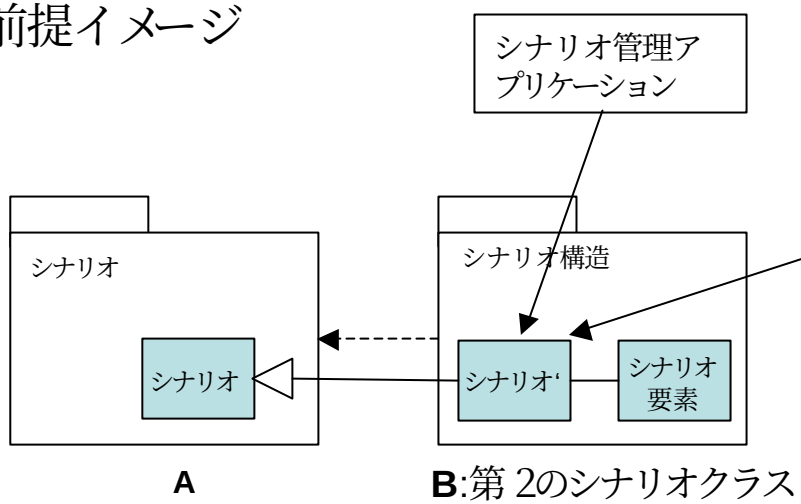


11.1 パッケージへの多重アクセスレベル

―シナリオ構造からのシナリオへの可視性の補足―

- 継承・ポリモーフィズムによる可視性である。
- 外部からはシナリオパッケージのみ可視性をもつとしてもシナリオ構造パッケージのインスタンスを扱うことができる。
- 可視性は、コンパイルやローディングの依存性を反映していない。

前提イメージ



シナリオ管理アプリケーション用には新たなシナリオのサブタイプが必要である。

しかし、とくにそのサブタイプのみに存在する特別な特性をつかわなければこれを持たなくても良い。

11.2 相互可視性

―問題提起―

- 取引契約とパーティのパッケージ化を取り上げる。
  このパッケージ間に相互可視性の問題が発生する。

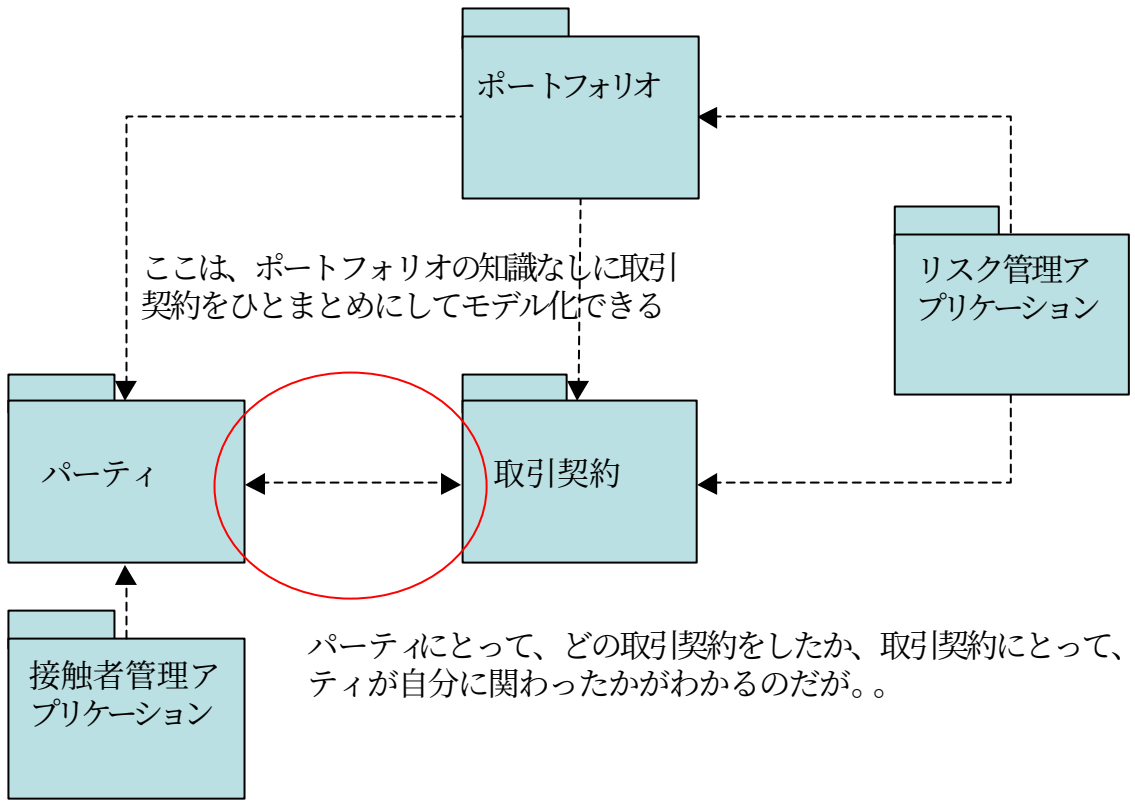


図11.6

取引契約とパーティを加えてみると、取引契約とパーティ間に相互可視性が発生する。

11.2 相互可視性

― 解決方法―

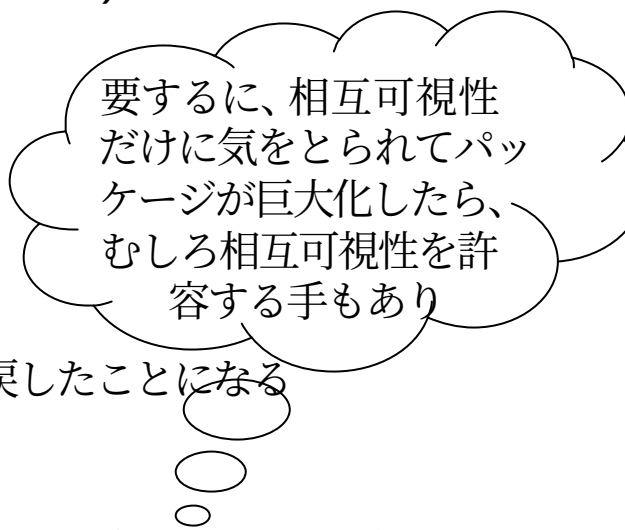
- つぎの2つの方法があり、トレードオフで決める。
  1. 相互可視性を一方向に制限する。
     - 結合度（カップリング）を抑えパッケージ開発の作業量を減らすが、両方の型を必要とするアプリケーション開発が複雑になる。
  2. ひとつのパッケージとする。
     - 両方の型を必要とするアプリケーション開発をシンプルにするが、パッケージ開発の作業量は増える。

- モデリングの原則
  - どちらの方法を採択するかは、型の開発者の作業量の低減と型の利用者の利便性の向上のトレードオフで決める。

11.2 相互可視性

― あらたな問題と解決方法―

- あらたな問題
  - パッケージ結合をした場合、接触者管理パッケージは必要以上の可視性を持つことになる。つまり、利用者(接触者管理の開発者)の利便性の低下を引き起こす。
- 解決方法
  - 相互可視性や他の循環は、最小限許容する。
    - さもないと、一方向か双方向関連かのトレードオフか、すべてが可視となる巨大パッケージの選択しかない。
  - 結局、解決方法というより相互可視性がある元の形に戻したことになる
- モデリングの原則
  - もしパッケージが他のパッケージの一部に対する可視性だけを必要とするならば参照されるパッケージを相互可視性をもつ2つのパッケージに分割することを検討せよ。



11.2 相互可視性

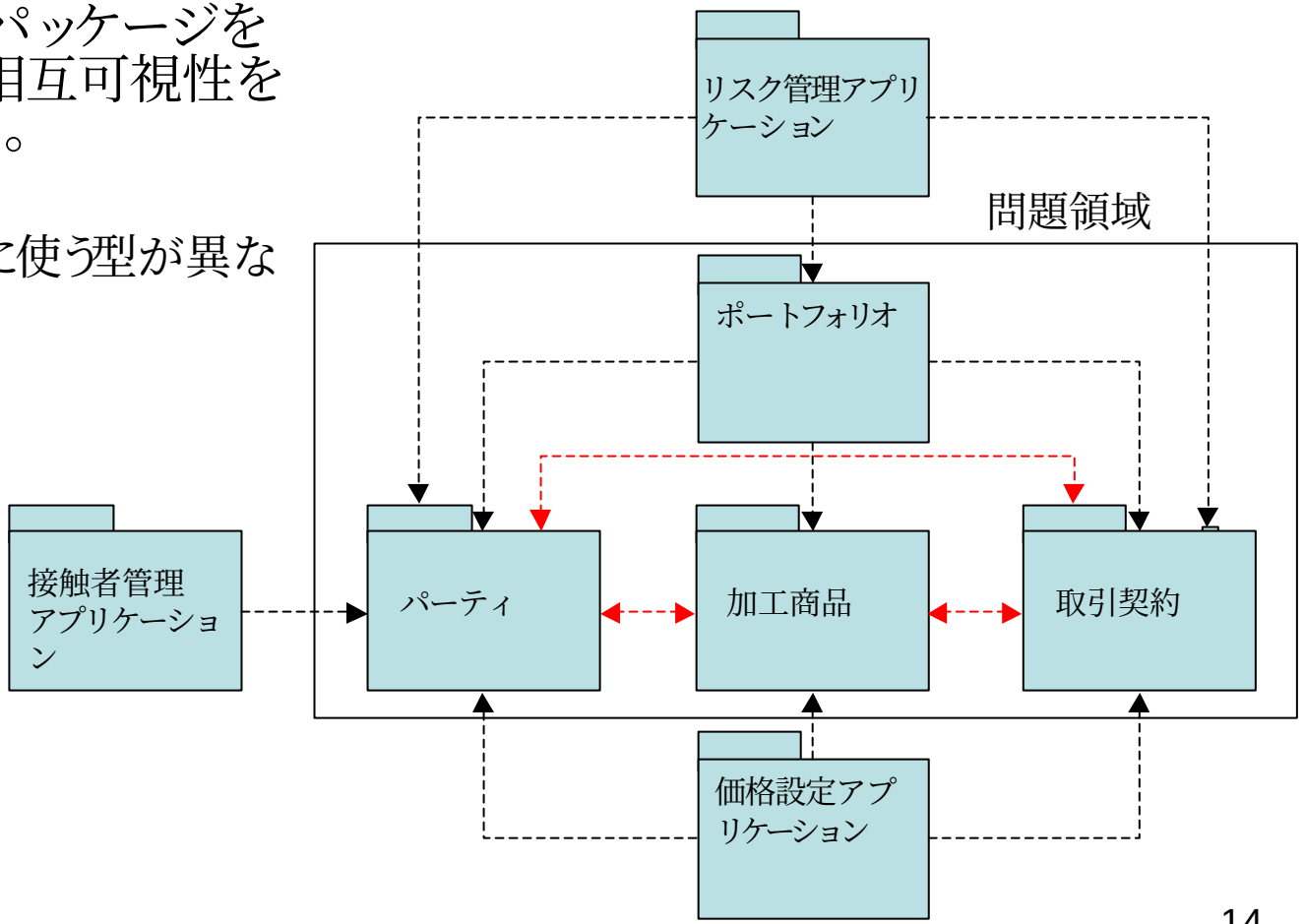
―解決方法:相互可視性キープの例―

- 解決方法

むりに、3つのパッケージをひとつにせず相互可視性をそのままとする。

パッケージごとに使う型が異なる場合に有効。

図11.8

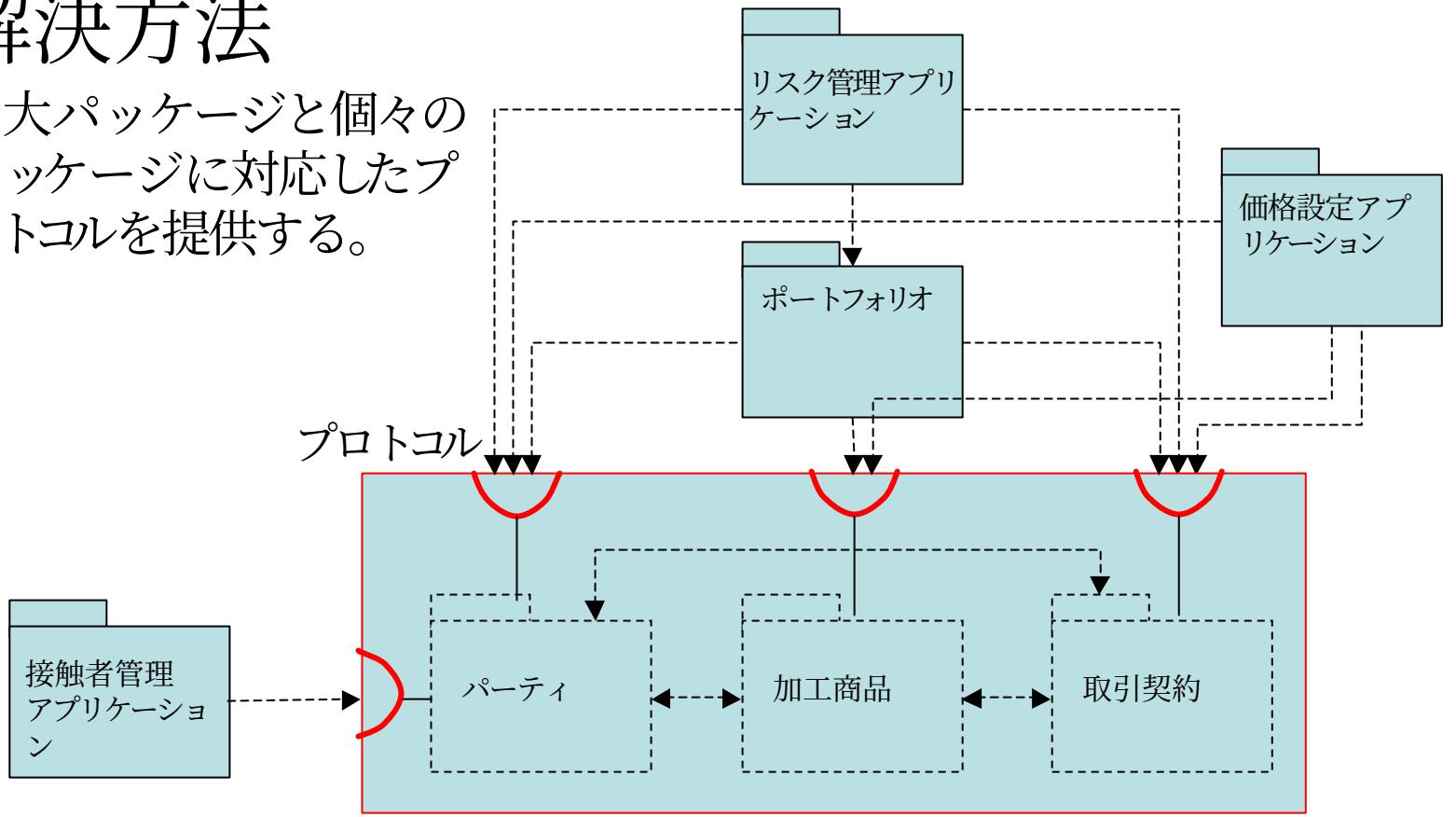


11.2 相互可視性

ー解決方法:巨大パッケージと分離プロトコルの例ー

- 解決方法

  巨大パッケージと個々のパッケージに対応したプロトコルを提供する。



図(想像図)　巨大パッケージ

11.2 相互可視性
―解決方法:まとめ―

- 型同士が密結合しているときの解決方法
  1. 関連を一方向とする。
  2. 巨大パッケージとする。
  3. ふたつの相互可視的パッケージとする。
  4. 分離したプロトコルを持つ巨大パッケージとする。

11.3 パッケージのサブタイプ化

―問題と解決方法―

- 問題
  - サブタイプが混在するモデルの可視性をどのように扱うか。

- 解決方法
  - サブタイプ追加の場合には、スーパータイプとの相互可視性を避けること
    - スーパータイプに関わりなく拡張できる
    - 後の拡張で報われる。
  - 自分のサブタイプを知っているような型は使わない。
    - 拡張性に問題がおこりやすい。
  - 2，3のサブタイプを統合するまではスーパータイプを固定しないこと

11.3 パッケージのサブタイプ化

―解決方法:例―

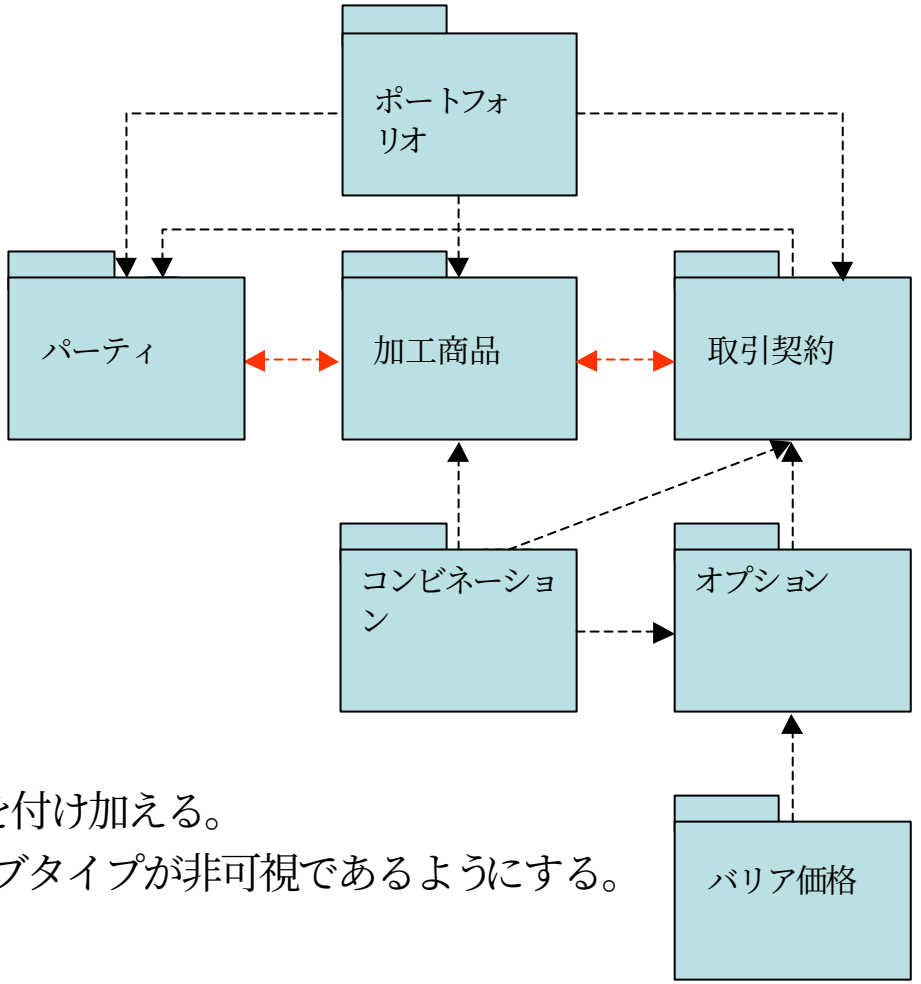


図11.9

いろいろなオプションを付け加える。

スーパータイプからサブタイプが非可視であるようにする。

11.4 まとめ

- 可視性の問題は常にトレードオフを内包する。
  - 可視性を制限（一方向の可視性）するとナビゲーションを難しくしてしまう。
  - 双方向の可視性はナビゲーションを容易にし、作成や修正のコード量を低減できるが、詳細部分が相互に参照しあっているとモデル変更の影響を統制するのが難しくなる。
- トレードオフの考え方はモデラーにより異なるが可視性を一方向に制限するのは難しい。
  - モデラーによっては、型についての一方向関連、可視グラフにより可視性を高度に制限するという考え方もあるが、
  - 筆者は、この制限をゆるめ、可視性は型の段階ではなく、パッケージの段階で考慮
    - 要するに、階層間の可視性は一方向になるが、 問題領域のパッケージ間の可視性を一方向に制限するのは難しい」と言っていると思われる。
- パッケージアーキテクチャーの開発は重要
  - 大企業の情報システム統合化のためには、企業全体規模のモデルをつくる時間がかかりすぎてしまうので、より日和見的なアプローチをとるべきである。
  - この作業には、少なくともパッケージと可視性は必要なツールである。