J2EE を使って開発するなら、まずは読まなきゃだめでしょう。

http://archives.java.sun.com/j2eepatterns-interest.html
http://www.corej2eepatterns.com/

パターンカタログ

[J2EE パターン]

プレゼンテーション層のパターン

■ Intercepting Filter

文脈

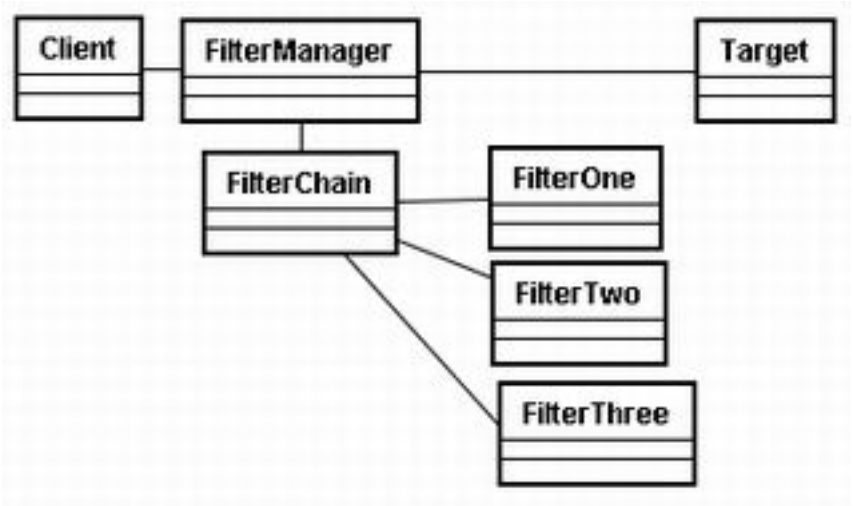
プレゼンテーション層では、リクエストをさまざまな方法により処理しなければならない

問題

クライアントからの Web リクエストやレスポンスに対して、前処理や後処理が必要

解決策

プラグイン可能なフィルタを作成し、リクエスト処理コードのコア部分への変更が不要な標準的な方法で共通サービスを処理。
フィルタは入力リクエストと、出力レスポンスを横取りし、前処理と後処理が行えるようにする。



■ Front Controller

文脈

プレゼンテーション層のリクエスト処理では、複数のリクエストにまたがった各ユーザの処理を制御、統合する必要があり、集中または分散管理する。

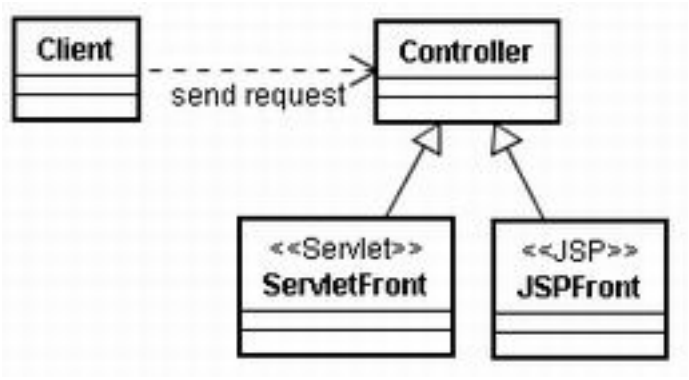
問題

ユーザが集中したメカニズムを通らずにビューにアクセスすると、

・各ビューで独自にサービスを提供する必要が発生し、コードが重複
・ビューのナビゲーションがビュー自身に任され、コンテンツとナビゲーションが混在してしまう
・制御が分散すると保守が困難

解決策

リクエストを処理するための最初の入り口として、コントローラを使用し、リクエストの処理を管理する。



■ View Helper

文脈

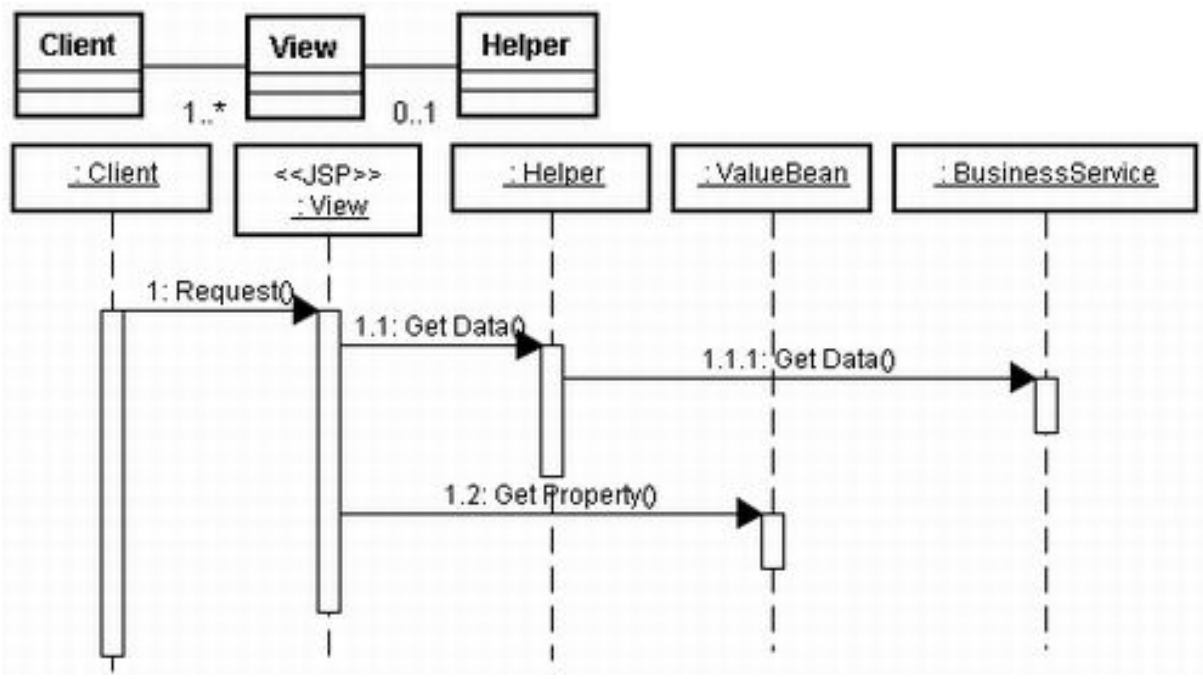
表示用コンテンツ作成のために、ビジネスデータを動的に処理する必要がある

問題

プレゼンテーション層の変更は頻繁に発生するが、ビジネスデータのアクセスロジックと表示用のロジックが絡み合っていると、開発、保守が困難

解決策

ビューは整形用コードのみを含み、処理に関する責務を JavaBean または、カスタムタグとして実装されたヘルパークラスに委譲する。また、ヘルパーにはビューのための中間データモデルが格納され、ヘルパーはビジネスデータに対するアダプタの働きをする



■ Composite View

文脈

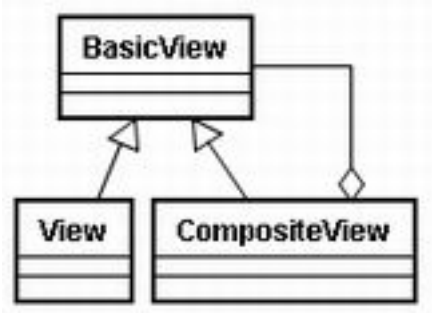
複雑な Web ページでは、1 つの表示用ページを、複数のサブビューを使って、多数のデータソースから集めたコンテンツを表示することで構成する

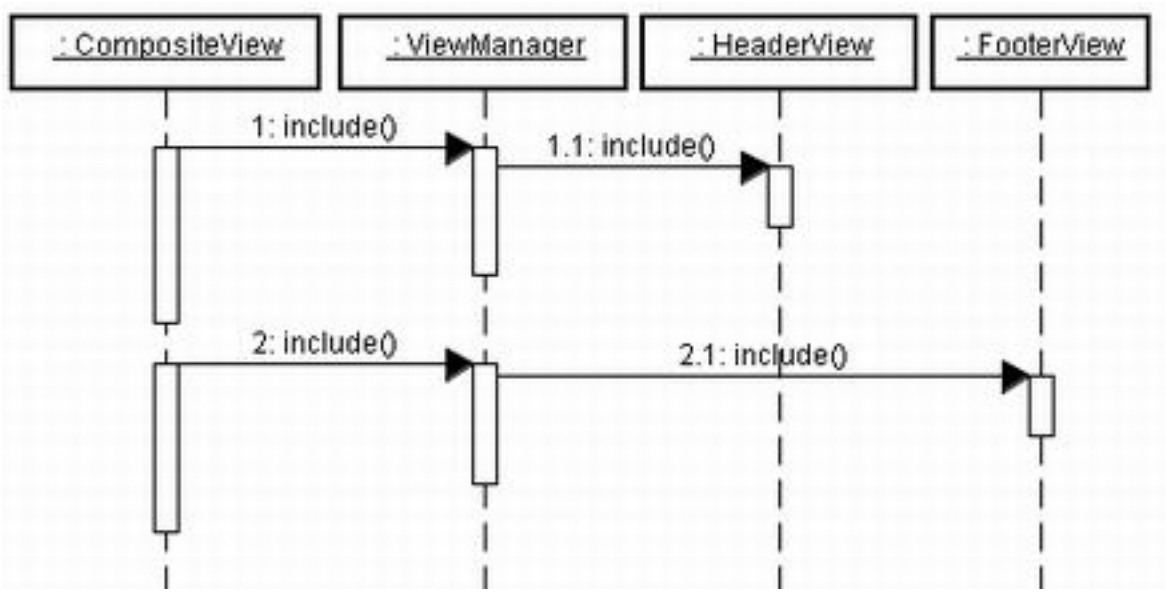
問題

原子的なビュー部品を組み合わせるのではなく、それぞれのビューに整形コードを直接埋め込んでページが構築されている

解決策

複数の原子的なサブビューから構成された複合ビューを使用する。テンプレートの各コンポーネントは全体ビューの中に動的に組み込まれ、ページレイアウトはコンテンツとは別に管理できる





■ Service to Worker

文脈

システムは実行のフローとビジネスデータに対するアクセスを制御し、ビジネスデータから表示用のコンテンツを作成する

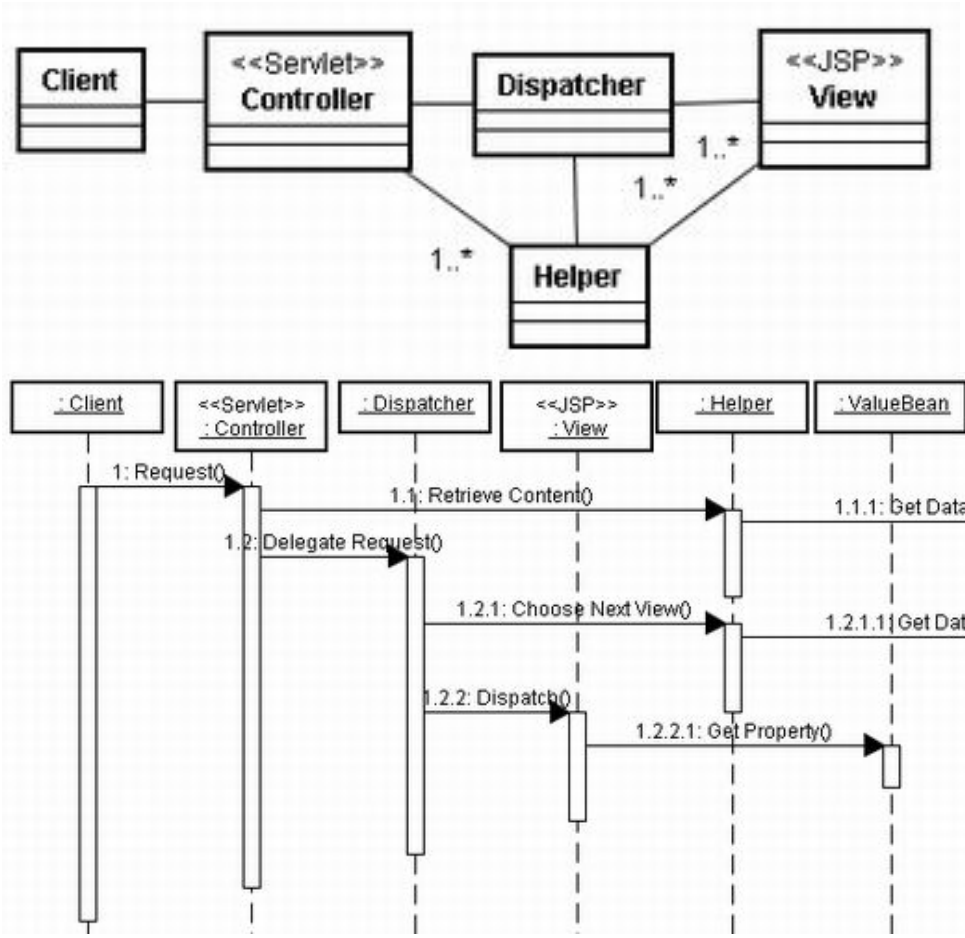
問題点

アクセス制御やコンテンツの検索の管理、ビューの管理を集中して行うコンポーネントが存在せず、制御コードがビューに複製されている。ビジネスロジックと、整形用ロジックが、ビューに入り混じっている

解決策

コントローラとディスパッチャをビューやヘルパーと組み合わせて、クライアントからのリクエストを処理する。
Service to Worker では、ディスパッチャコンポーネントを利用した、Front Controller パターンと View Helper パターンの組み合わせを記述する

・コントローラはコンテンツの検索をヘルパーに委譲
・ヘルパーはビュー用の中間的なモデル生成を管理
・ディスパッチャは、ビューの管理とナビゲーションの責務をもつ



■ Dispatcher View

文脈

システムは実行のフローと表示処理に対するアクセスを制御し、表示処理では動的コンテンツの生成に関する責務を持つ

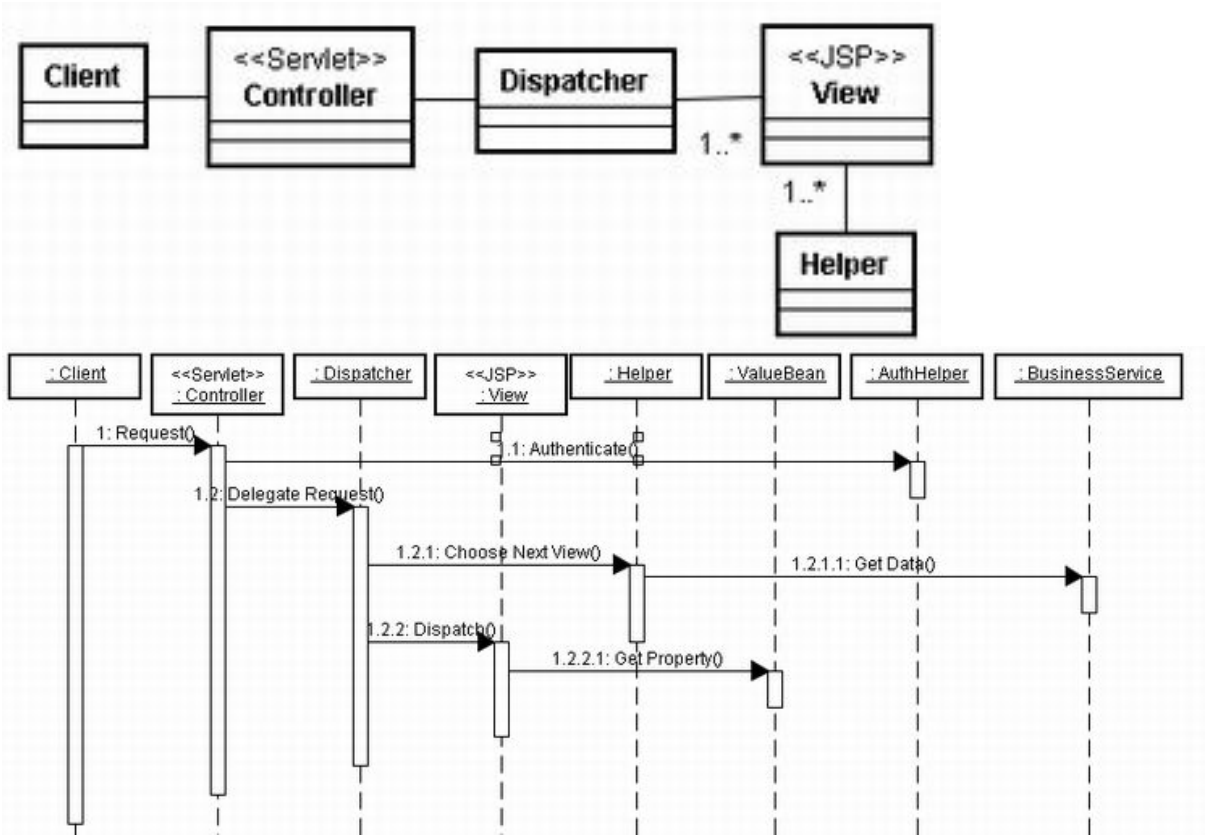
問題点

アクセス制御やコンテンツの検索の管理、ビューの管理を集中して行うコンポーネントが存在せず、制御コードがビューに複製されている。ビジネスロジックと、整形用ロジックが、ビューに入り混じっている

解決策

コントローラとディスパッチャをビューやヘルパーと組み合わせて、クライアントからのリクエストを処理する。

- コンテンツの検索はビュー処理時まで遅延されるので、コントローラは処理をヘルパーに委譲しない
- ディスパッチャは、ビューの管理とナビゲーションの責務を持つ



ビジネス層のパターン

■ Business Delegate

文脈

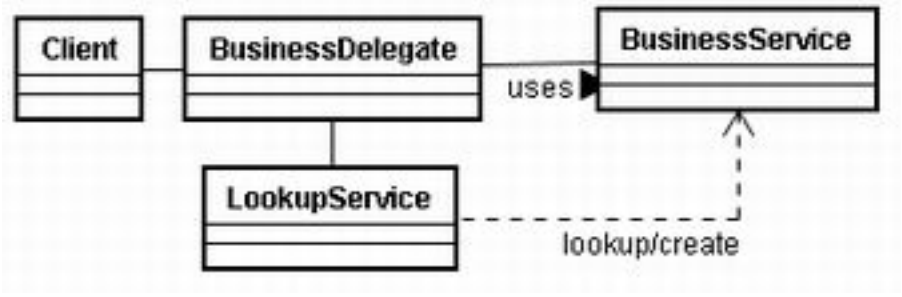
多層分散型システムでは、層をまたがったデータの送受信には、リモートメソッド呼び出しを行う必要があり、クライアントは複雑な処理が必要

問題

プレゼンテーション層から直接ビジネスサービスへ作用すると、ビジネスサービスの API 実装の詳細が、公開されてしまう。結果、ビジネスサービスの変更にプレゼンテーション層が影響を受けやすくなる。
さらにネットワークパフォーマンスに悪影響

解決策

Business Delegate を利用して、プレゼンテーション層のクライアントとビジネスサービスの結合度を低くする。Business Delegate は、EJB アーキテクチャの詳細や、ビジネスサービスの実装の詳細を隠蔽する



■ Value Object

文脈

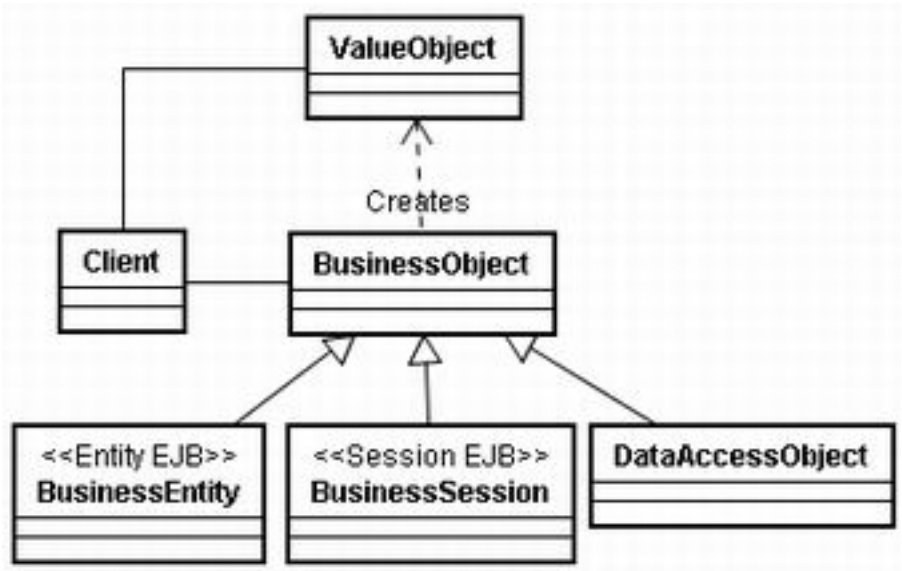
クライアントはエンタープライズ Bean とデータをやり取りする必要がある

問題

- サーバ側のビジネスコンポーネントをセッション Bean やエンティティ Bean として実装している。
- ビジネスコンポーネントのメソッドの中には、クライアントにデータを返すものがある
- 多くの場合、クライアントが必要な属性地をすべて取得するには、get メソッドを複数回呼び出す必要がある
- このメソッド呼び出しは、リモートで行われる可能性があり、ネットワークのオーバーヘッドが伴う。
- このような呼び出しが頻繁におきると、パフォーマンスが大きく低下する

解決策

Value Object を使ってビジネスデータをカプセル化し、1 回のメソッド呼び出しで、バリューオブジェクトを送受信する。



■ Session Facade

文脈

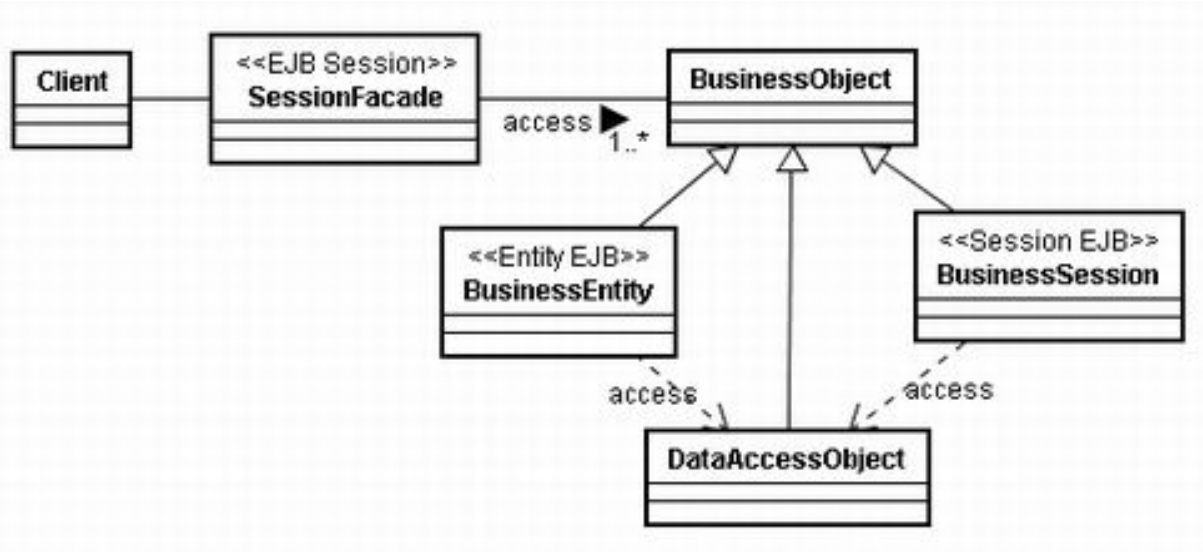
エンタープライズ Bean は、ビジネスロジックとビジネスデータをカプセル化し、インターフェースを公開する。分散サービスの複雑な部分が見えてしまう

問題

・結合度が高くなり、クライアントとビジネスオブジェクトとの間に直接の依存関係が発生
・クライアント、サーバ間のメソッド呼び出しが多くなり、ネットワークパフォーマンスの問題が発生
・クライアントからの一貫した呼び出し方法がなく、間違った使われ方をする

解決策

セッション Bean をファサードとして使うことで、ワークフローに参加するビジネスオブジェクトどうしの複雑な相互作用をカプセル化する。Session Facade は、ビジネスオブジェクトを管理し、クライアントに対して一貫した粗粒度のサービスアクセスレイヤを提供する



■ Composite Entity

文脈

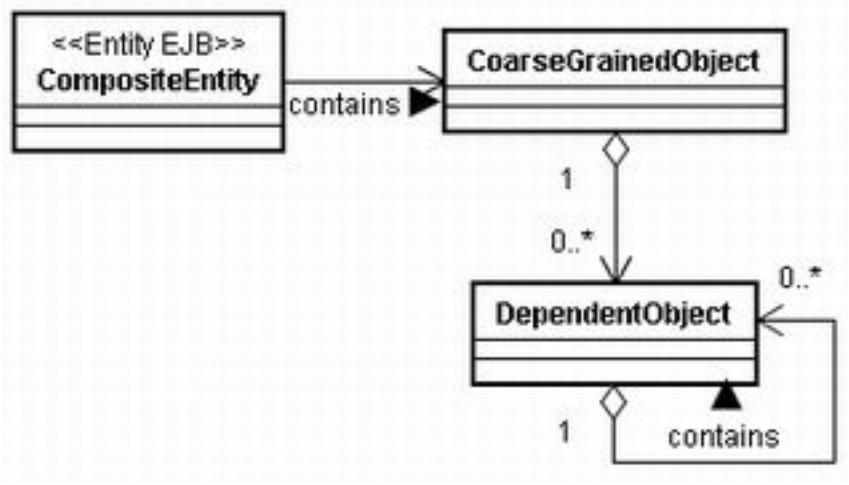
エンティティ Bean はオブジェクトモデル内の個々の永続オブジェクトを表現するためのものではなく、より粗粒度の永続ビジネスオブジェクトを表現するのに適している

問題

J2EE アプリケーションでは、クライアントはリモートインターフェースを通してエンティティ Bean にアクセスする。エンティティ Bean のオブジェクトの粒度が細かい場合、エンティティ Bean のメソッド呼び出しが多くなり、ネットワークのオーバーヘッドが増大

解決策

互いに関係する一連の永続オブジェクトを、個々の細粒度のエンティティ Bean として渡すのではなく、Composite Entity を使ってモデル化して管理する。
永続オブジェクトとは、何らかの種類のデータストアに格納されるオブジェクトで、通常複数のクライアントにより共有される。



■ Value Object Assembler

文脈

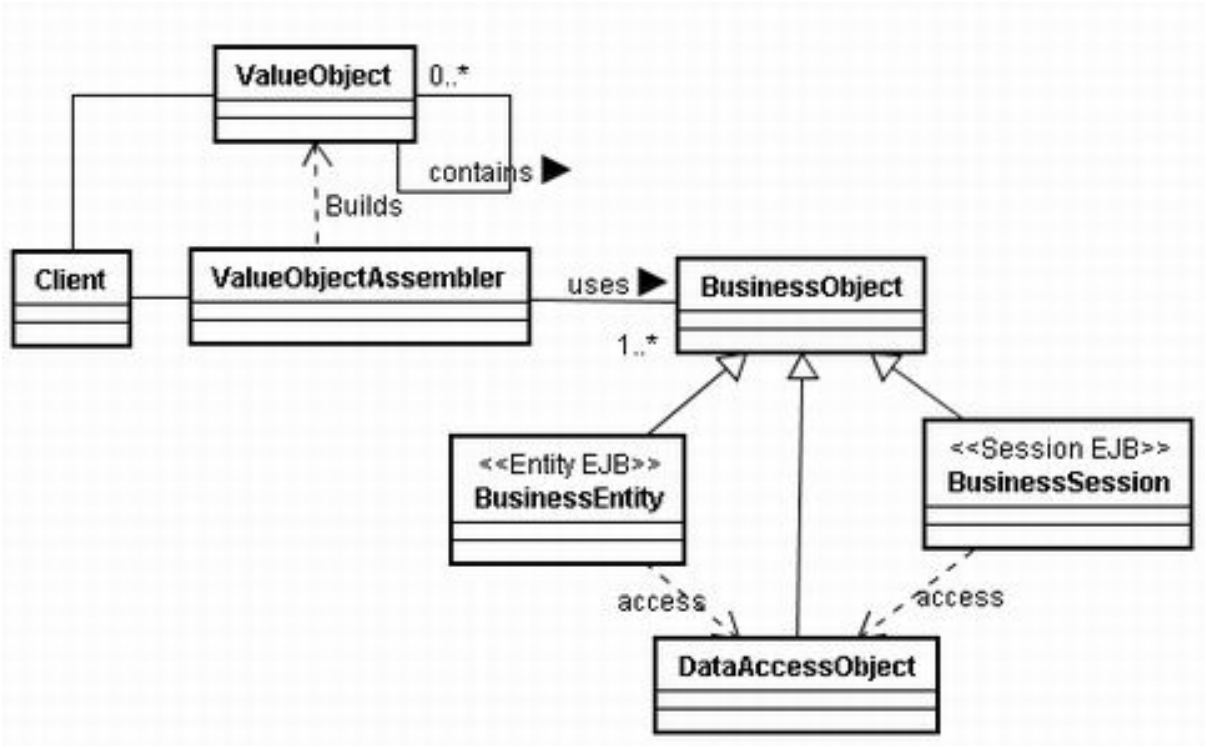
アプリケーションクライアントはしばしば、セッション Bean やエンティティ Bean、DAO 等、複数のオブジェクトから構成されたデータにアクセスしなければならない

問題

アプリケーションクライアントは大抵モデルのデータを使用する必要があり、モデルはツリーやグラフとしてまとめられたオブジェクトの集合として表されることがある。J2EE アプリケーションでは、セッション Bean やエンティティ Bean、DAO 等から構成されるが、クライアントがモデルデータを取得するためには、モデルを定義する個々の分散オブジェクトにアクセスする必要がある。

解決策

Value Object Assembler を使って、必要となるモデルやサブモデルを構築する。Value Object Assembler は、モデルを定義する各種オブジェクトから、バリューオブジェクトを使ってデータを取り出す



■ Value List Handler

文脈

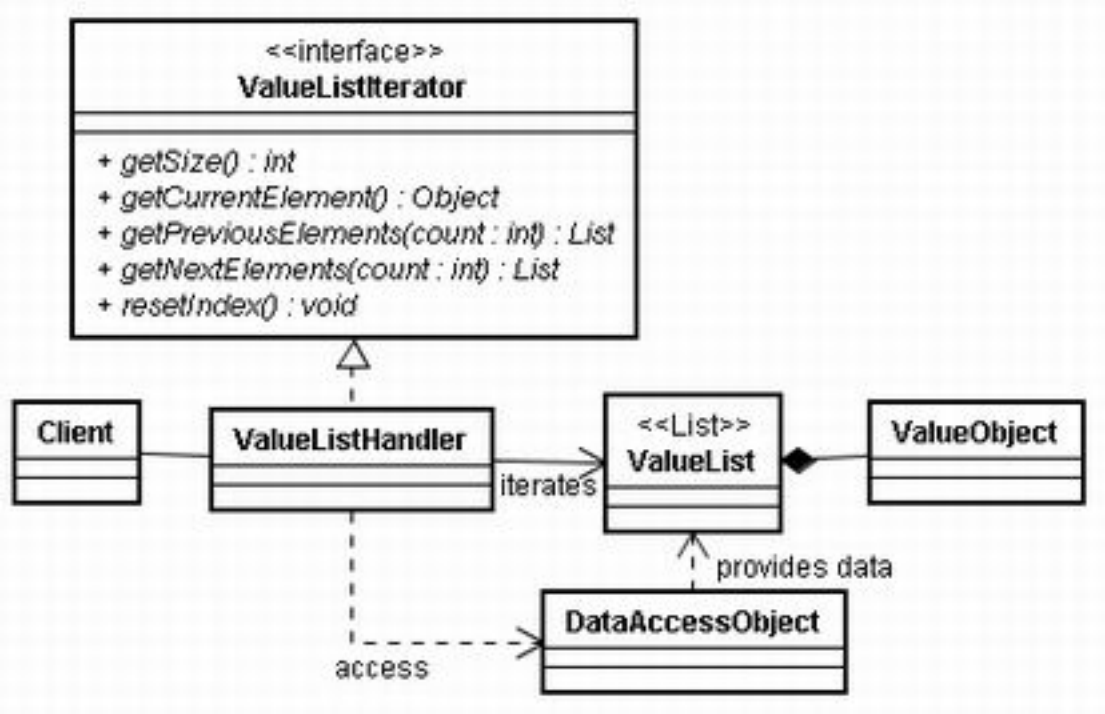
表示用のサービスに対してクライアントが項目のリストを要求するが、項目数は不明でああり、かなり大きくなる可能性がある

問題

検索結果がかなり大きくなる場合がある。クライアントは通常結果リストを表示するために結果を使用するが、参照するのは、最初の 2、3 レコードだけであったり、すぐに結果に対してトランザクションを実行しないこともよくある。
リモートオブジェクトの集合を返す、ejbFind() メソッドを呼び出し、各エンティティ Bean に対して値のリストを取得するという方法は、ネットワーク負荷が高く悪い方法と考えられている

解決策

Value List Handler を使うことにより、検索を制御し、結果をキャッシュする。クライアントにその用件を満たすサイズや走査方法を持つ結果セットを提供する。
Value List Handler は、必要な問い合わせを実行できる DAO に直接アクセスする。



■ Service Locator

文脈

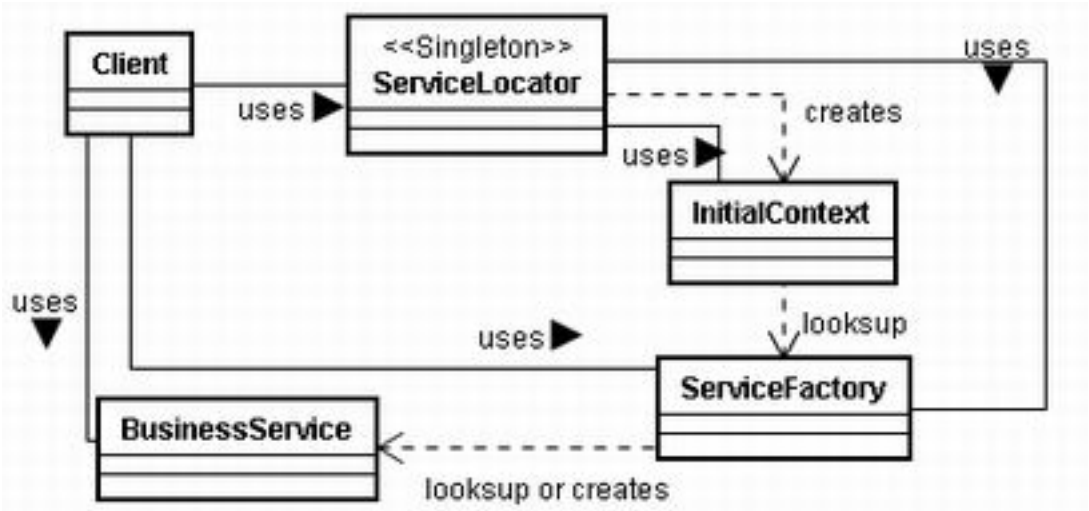
サービスを検索し生成するために、複雑なインターフェースやネットワーク操作が必要

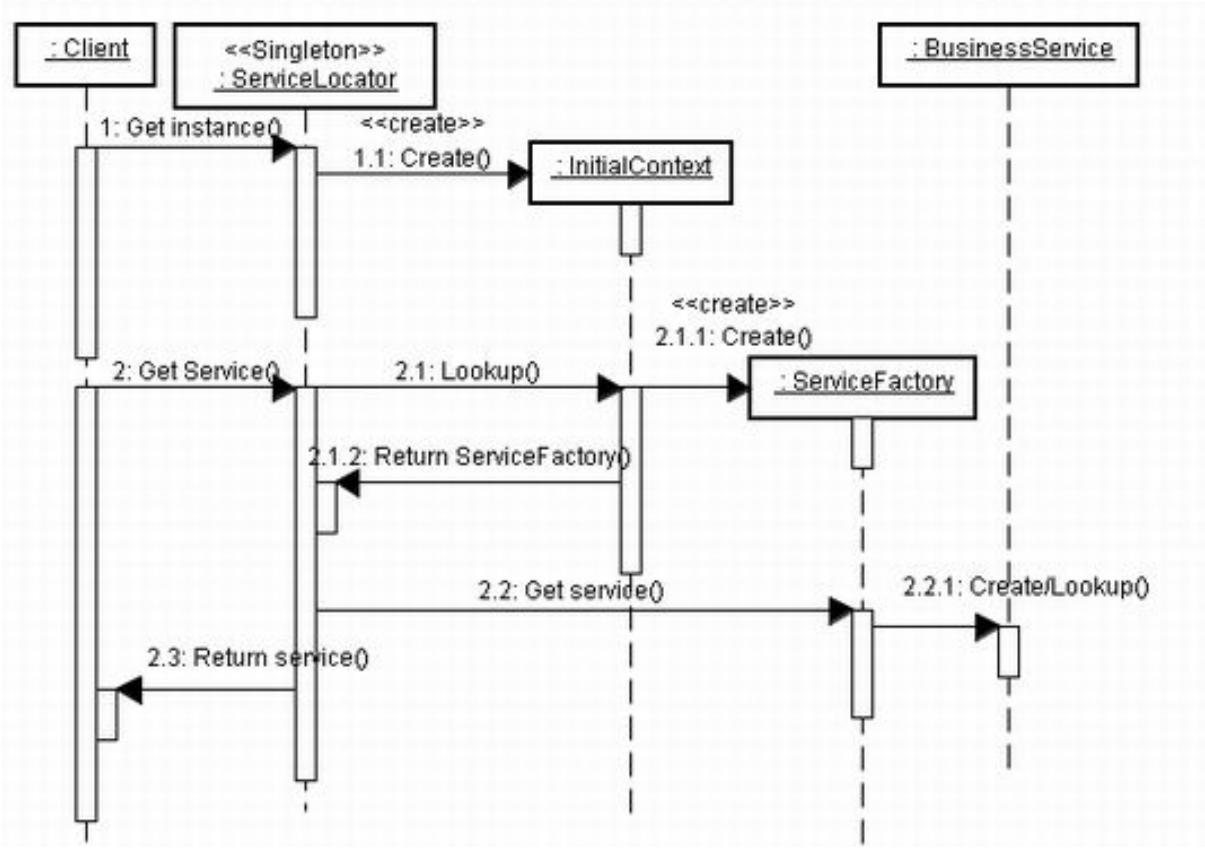
問題

J2EE クライアントは、EJB や JMS といったサービスコンポーネントと相互作用するために、検索操作を行うか、新しいコンポーネントを生成しなければならない。
J2EE クライアントはすべて JNDI の共通機能を使ってコンポーネントの検索や生成を行い、その検索処理はクライアントに共通であるため、多種多様なクライアントにそのコードが散在してしまい、不必要に重複してしまう

解決策

Service Locator オブジェクトを使って、JNDI を使用する部分をすべて抽象化し、複雑な処理を隠蔽する。複数のクライアントが、オブジェクトを再利用できるため、コードの複雑さを緩和し、唯一の制御点を提供し、パフォーマンスを改善できる





■ Data Access Object

文脈

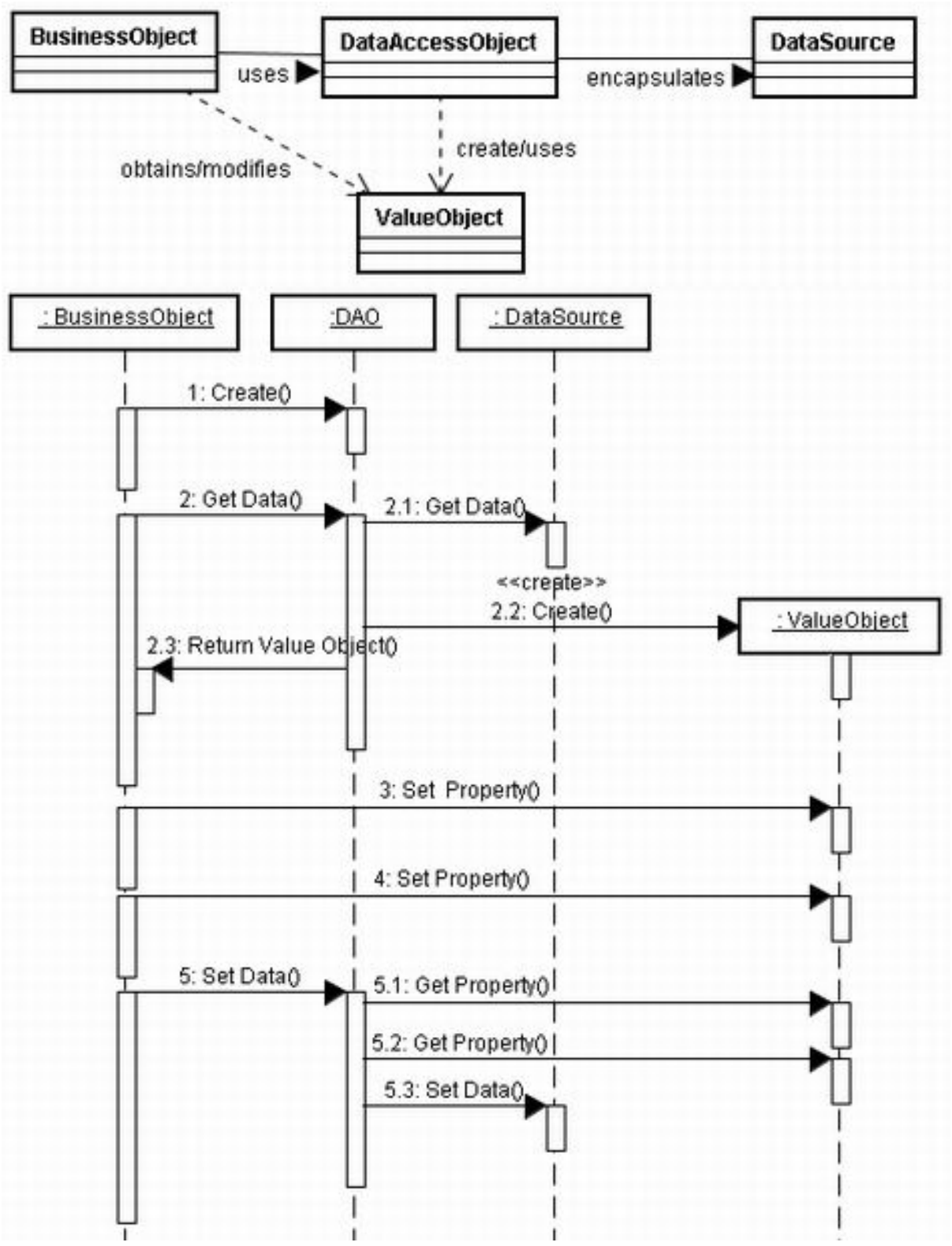
データに対するアクセス方法はデータソースによって異なる。

問題

J2EE アプリケーションの多くはどこかで永続データを使用する必要があるが、永続ストレージは異なるメカニズムで実装されているため、API に顕著な差がある。

解決策

Data Access Object を使ってデータソースに対するすべてのアクセスを抽象化し、カプセル化する。DAO はデータを取得し更新するために必要なデータソースとの接続を管理する



Service Activator

文脈

エンタープライズ Bean と他のビジネスサービスを、非同期に活性化する方法が必要

問題

エンタープライズ Bean へのアクセス時､検索やリモートメソッド呼び出しなどのメソッド呼び出しは、すべて同期的に行われる。すなわちクライアントはメソッドから結果が返ってくるのを待たねばならない

解決策

Service Activator を使ってクライアントからの非同期のリクエストやメッセージを受け取る。Service Activator はメッセージを受け取ると、ビジネスサービスやコンポーネントを検索、メソッドを呼び出し、非同期でリクエストを処理