

アプリケーション部品についての MVC パターンの適用

4 C - 1

鷲崎 弘宜 白銀 純子 深澤 良彰

早稲田大学理工学部

1 はじめに

GUI アプリケーションの開発時に、最も基本的な汎用コンポーネントを組み合わせ、さらに処理ロジックを付加することで特定のドメインに特化された、クラスレベルでの再利用を目的とする GUI アプリケーション部品を利用することがある。そのような GUI アプリケーション部品を WYSIWYG を実現する従来の RAD ツールにより開発すると、GUI とロジックが依存性を持って混在するコンポーネントとして作成されるため、ロジックと GUI の分離設計は部分的なものとなり、アプリケーション部品の保守性、再利用性が大きく失われる危険性を持つ。本稿では、GUI とロジック間の結合について、GUI からロジックへのイベントモデルだけでなく、ロジックから GUI への通知方法も策定することで、GUI アプリケーション部品内部で相互に依存しない完全に独立したアーキテクチャを提案する。

2 MVC パターン導入の必要性

従来の RAD ツールにより生成される GUI アプリケーション部品は、内部において各 GUI 部品からのイベントについてアダプタが用意され (Delegation イベントモデル [1])、開発者が記述するロジックと接続される (図 1)。

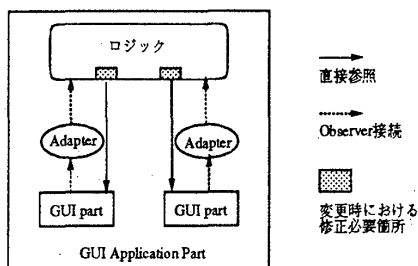


図 1: 従来の GUI アプリケーション部品

よって、各 GUI 部品からのイベント通知からロジックは独立しているが、ロジックから各 GUI 部品への参照はハードコードされ、結果として GUI 部品とロジック間に依存性が生じ、次の問題を引き起こす。

1. 保守・拡張性の低下：各 GUI 部品へのロジックからの直接参照により、GUI の一部または全ての修正について、アダプタだけでなくロジックの修正が必要になる。また、同一クラスファイル内に両部分が混在するため、修正を行いたい箇所が分かりにくく、結果として拡張についてのコストの増大 (保守性の低下) を引き起こす。
2. 分離同時開発性の欠如：GUI とロジックの分離は RAD ツール内での疑似的なものであり、実際には複数の開発者による分離同時開発を行なうことができない。

上記の問題を解決するために、GUI アプリケーション部品の開発時に、MVC パターン [2] を適用するアーキテクチャ「Beam(Beam for Mvc)」を提案する。

3 Beam アーキテクチャ

複数の汎用コンポーネントを組み合わせ、JavaBeans フレームワークに沿った GUI アプリケーション部品を開発する際に、GUI とロジックを完全に分離して開発・結合する追加的な仕組みが Beam である。Beam コンポーネントは以下の Container、Model、UI の 3 オブジェクトを中心に構成される (図 2)。

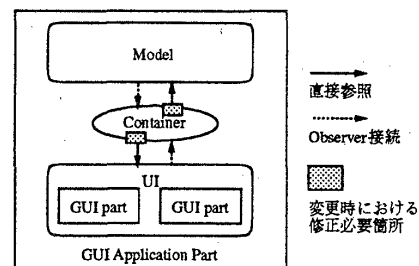


図 2: Beam による GUI アプリケーション部品

- Container：Model および UI のインスタンスを生成し、両者間でのデータ及びイベント通知の仲介を行う。BeamContainer クラスを継承。
- Model：データとデータに対する操作を持ち、Container への参照を持つ。BeamModel インターフェースを実装。
- UI：ユーザとの対話を GUI により行い、Container への参照を持つ。BeamUI インターフェースを実装。

Model と UI 間での通信は、独立性を保つために以下の仕組みにより行なわれる。

1. データ共有：Model 内のデータで UI と共有すべきデータを設計時に定め、Model 内の共有すべきデータを Observable(in dataTable)、UI を Observer とする Observer パターン [2] により、自動的な共有メカニズムを提供する。

Model 内の共有するデータは Container のデータテーブルに識別用の名前と共に登録され、値の変更時には自動的に UI へ通知される。UI は通知された名前を用いて Container より値を取り出し、出力処理を行う (図 3)。

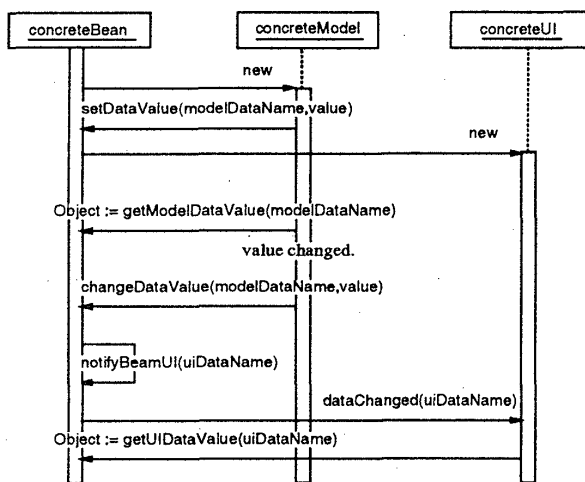


図 3: データ共有フロー

2. 内部イベント処理：UI におけるユーザ入力にともなうイベントは、全てコンポーネント内部で処理される。内部イベントは UI より Container へイベントについての識別用の名前と共に転送され、Container 内においてイベントの名前についてそれぞれ必要な処理を定義する。

その際に、UI 上の取得すべき一時的なデータは、UI と Model の独立性を最重視しているため、イベント共に送られるのではなく、Container より必要に応じて UI から公開されたメソッドを用いて取得するという方法を採用する (図 4)。

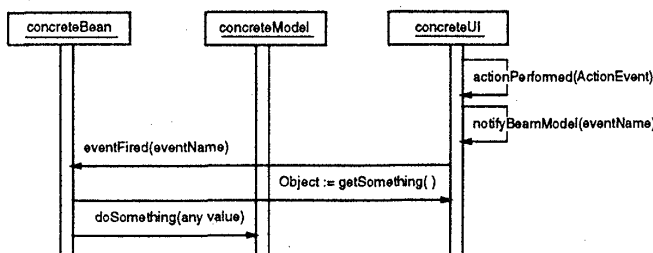


図 4: 内部イベント処理フロー

外部に対する JavaBeans コンポーネントとしての振舞いは、Model および UI のプロパティへの外部からのアクセスと、外部へのイベント通知を Container を経由させることで実現する。

Beam 適用により、以下の効果が期待される。

1. 部分単位での詳細なカスタマイズ：ロジック・GUI の一方についての変更・修正の際に、Container における結合定義の修正が必要になる可能性はあるが、結合定義は Model・UI 双方より独立しており、他方のオブジェクトに影響が出ることはない。
2. 複数 UI の登録：Model・UI はお互いを認識しないため、単一の Model に対し複数の UI を Container 経由で依存させられる。よって、GUI の動的な切替え、同一データに対する複数の表示方法の同時提供が実現できる。

4 統合開発ツール BeamBuilder

コンポーネントは本来 WYSIWYG を実現する RAD ツールによりビジュアルに使われることを目的とするため、Beam アーキテクチャに沿った Beam コンポーネントをビジュアルに開発するツールが望まれる。

BeamBuilder は Beam コンポーネントを開発するためのツールであり、オブジェクトに対する機能は各 Beam-ModelBuilder(Model 開発)、BeamIntegrator(Container 開発)、及び、ビジュアルな GUI 設計を可能とする BeamUIBuilder(UI 開発) により提供される。BeamModelBuilder 及び BeamUIBuilder は、生成された Model および UI について Container 生成に必要な情報を ModelInfo および UIInfo オブジェクトにカプセル化し、BeamIntegrator へ提供する。

5 おわりに

現在の Beam アーキテクチャでは、UI からの内部イベントについて、UI 上の一時的なデータの取得は全て Container で各イベントについて定義する必要があり、内部イベントについての処理記述が複雑化することは避けられない。これは Model と UI の独立性を最重視することに起因するが、UI 上の GUI 部品の種類が定まっているならば、各 GUI 部品の一時的なデータと厳密に一对一対応させることによって、この複雑性を回避することが考えられる。

参考文献

- [1] Graham Hamilton "JavaBeans 1.01 Specification" Sun Microsystems.(1997)
- [2] Erich Gamma,Richard Helm,Ralph Johnson,John Vlissides "Design Patterns" Addison Wesley.(1995)