システムのバリアント

Håkan Dyrhage

Rational Software ホワイト・ペーパー

TP 155

Rational®
the software development company

目次

概要

このホワイト・ペーパーでは、システムのバリアントとそれを管理する方法について説明します。Rational Unified Process を理解することが目的の場合は、お読みになる必要はありません。 このホワイト・ペーパーは Rational Unified Process の拡張説明として利用してください。最後の項では、バリアントと可変性を導入することにより Rational Unified Process が受ける影響について簡単に説明します。

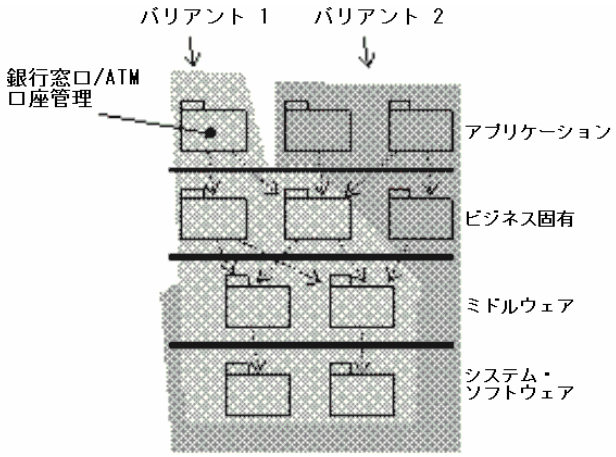
ここで説明する分野は、将来的に Rational Unified Process によって改善され、拡張されます。このホワイト・ペーパーは、この分野についての入門編です。

システムのバリアント

提供されるシステムの多くには、複数のバリアントがあります。これは、さまざまな (クラスの) 顧客に合わせてシステムの構成、パッケージ化、インストールが行われるためです。単にシステムを異なる方法でインストールして適合させることにより、多様なバリアントができる場合もあります。そのほか、さまざまな顧客にシステムの異なる部分を提供することにより、可変性が生まれることもあります。次の項では、バリアントのいくつかの例について説明します。

システムの異なる部分

完全なシステムの異なる部分がさまざまなクラスの顧客に提供されます。例えば、銀行システムは 2 つの異なる製品として提供されます。ここでは、テレフォン・バンキングに関するすべてのコンポーネントが格納されたシステムをバリアント 1、銀行窓口/ATM 口座管理に関するすべてのコンポーネントが格納されたシステムをバリアント 2 とします。実行可能プログラムは、アプリケーション・レイヤーのサブシステムで定義されます。これは、バリアント (次の図のバリアント 1) が、例えば銀行窓口/ATM 口座管理サブシステムと、コンパイルして実行する必要があるすべてのサブシステムのビルドであることを示します。 つまり、このバリアントでは、すべてのサブシステムが直接的または間接的にインポートされます。



2 つのバリアントとして開発された銀行システム

さまざまな言語

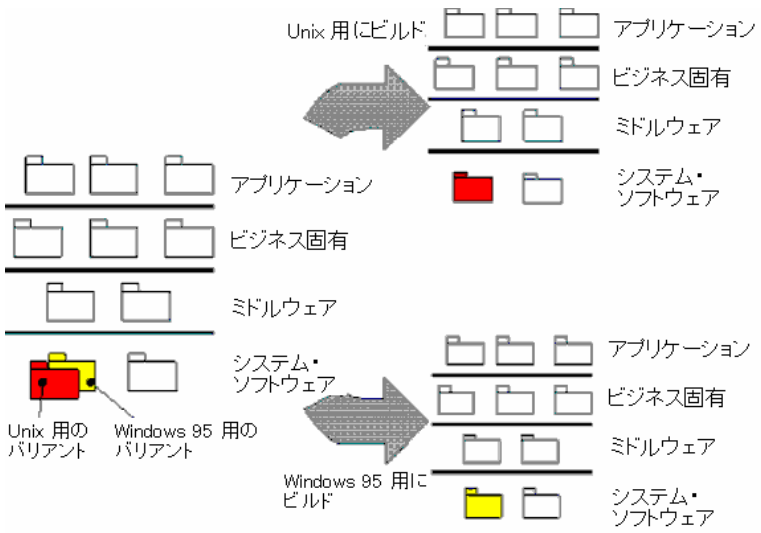
システムを、英語、フランス語、日本語などさまざまな言語で作成する場合、各国語用のシステムのバリアントを提供する必要があります。メニュー、ヘルプ・テキストなど、バリアント間で異なるすべてのテキストを固有の言語にする必要があります。

異なる言語を処理する方法の 1 つとして、すべてのテキストをファイルに集めて言語ごとにファイルを 1 つ用意することが挙げられます。特定の言語のシステムを提供する際には、その言語のテキストが格納されたファイルを含め、すべてのコンポーネントを提供します。起動時にソフトウェアでこのファイルを読み取り、関連するすべての変数を初期化します。

複数のプラットフォーム

互換性のない複数のプラットフォームをシステムでサポートする場合は、プラットフォームごとにシステムのバリアントが 1 つ必要です。例えば、システムを Windows NT と UNIX の両方で実行する場合は、システムのバリアントを 2 つ作成します。

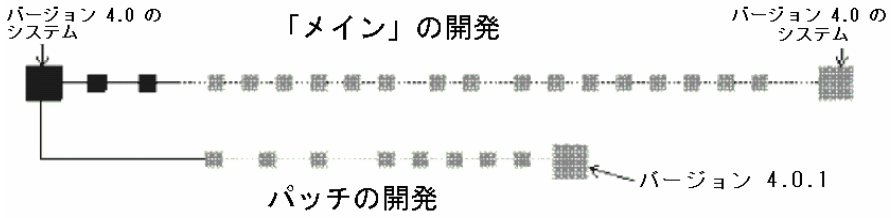
次の例では、プラットフォーム固有のコードが 1 つのサブシステムに配置されています。この例の場合、サブシステムのバリアントを 2 つ開発します。コンパイル・ファイル “makefile” は、各ソース・コード・ファイルのどのバージョンを一緒にコンパイルするかを指定します。makefile は、各サブシステムのどのバリアントをビルドの一部にするかを指定すると考えることもできます。したがって、プラットフォームごとに makefile が 1 つ必要です。



複数のプラットフォームで実行するシステムの開発

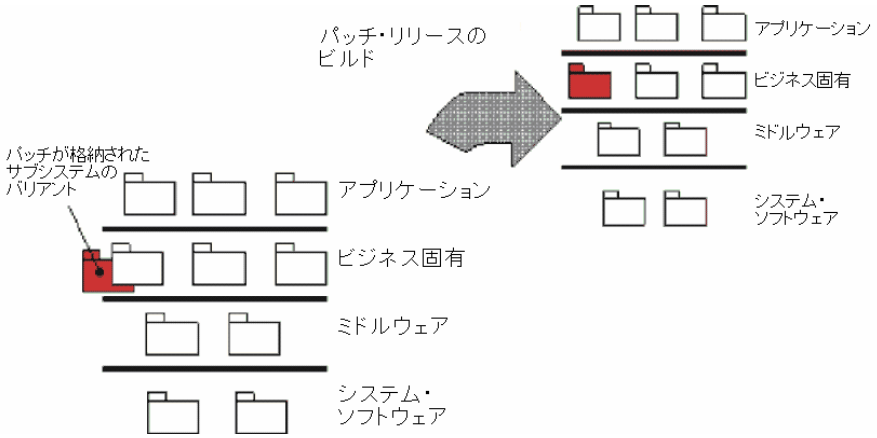
パッチ・リリース

システムのパッチ・リリースの開発が必要になることがあります。通常、これはプロジェクトの開発と並行して行われるため、パッチ・リリースはシステムの別のバリアントになり、システムの「メイン」バージョンと同時に存在することになります。



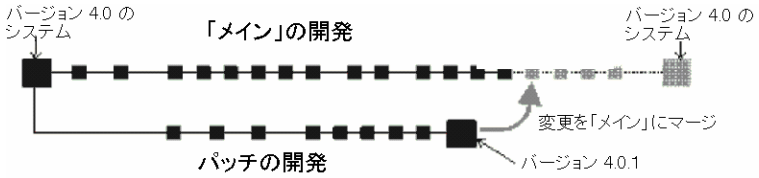
パッチ・リリースの開発は、メインの開発プロジェクトと並行して行われます。
灰色の四角は、将来のリリースとベースラインを示します。

実施する必要のある変更の内容は、1 つまたは複数の実装サブシステムに配置します。通常の開発プロジェクトが並行して進行しているため、メインの開発作業と同時にこれらのサブシステムのバリアントを開発する必要があります。ビルドを作成するには、各サブシステムのどのバリアントをビルドの一部にするかを makefile に指定します。



パッチ・リリースのビルドは、パッチが格納されたサブシステムのバリアントを使用して作成します。

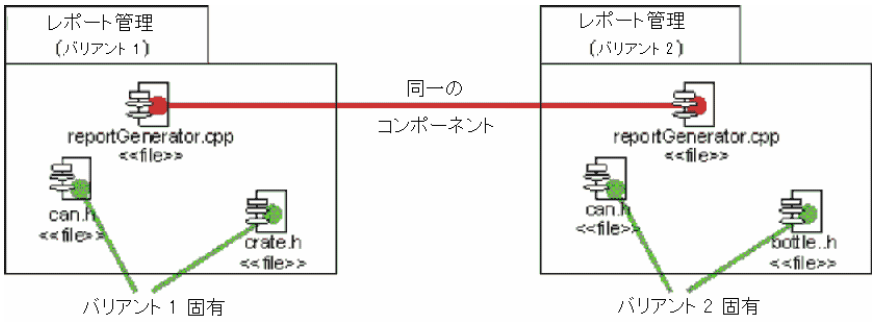
通常、この種類のバリアントの存続期間は短期です。パッチをリリースしたら、そのバリアントはそれ以上開発しないで、変更したコードのすべての内容をメインの開発作業に組み入れます。



パッチ・リリースに関連する変更内容を、都合のよい時点でメインの開発作業に組み入れます。

サブシステムのバリアント

前の例は、1 つの特定のサブシステムで複数のバリアントが必要になる場合があることを示しています。これらのバリアントにはいくつかの共通のコンポーネントがあり、そのほかは各サブシステムのバリアント独自のコンポーネントです。



Report Management と名付けられた実装サブシステムの 2 つのバリアントで、1 つのコンポーネントが同一です。その他のコンポーネントはそれぞれのバリアントに独自のものです。

サブシステムの各バリアントはそれぞれ 1 人の実装担当者により開発されることが多いため、サブシステムのバリアントは並行して開発されます。1 人の実装担当者がコンポーネントを変更してバリアントが 1 つ作成されたら、変更内容をほかのバリアントにも取り入れる必要があります。これを可能にするためには、2 つのバリアントで同一にする必要のあるコンポーネントを管理する**構成管理/バージョン管理** (以下、CMVC) ツールによるサポートが必要です。

可変性のメカニズム

システムのバリアントを作成するためのメカニズムがいくつかあります。その一部については前の項で説明しました。各メカニズムにはそれぞれ異なる特徴があります。通常、これらのメカニズムを組み合わせて使用し、システムに可変性を生み出します。

- **コンパイル (とリンク) ファイル**。コンパイル・ファイル (Unix 環境では makefile) に、各ソース・コード・ファイルのどのバリアントをコンパイルして実行可能ファイルにリンクするかを指定します。
- **動的にロードされるコンポーネント**。実行時に実行中のプログラムにリンクすることが可能な、ダイナミック・リンク・ライブラリー、アプレット、Active X コンポーネントのいずれかとしてシステムのパーツを開発します。これらのコンポーネントを CMVC ツールで管理し、顧客にコンポーネントのサブセットを提供できるようにします。
- **起動ファイル**。システムを起動する際にソフトウェアが読み取る情報が格納されたファイルを使用して、システムを初期化します。例えば、システムを起動する際に、Windows のリソース・ファイル、起動 ファイル、初期化ファイルなどをソフトウェアで読み取り、システムを異なる状態に設定します。システムがさまざまな言語にカスタマイズされている場合は、システムの起動時に読み取るテキスト・ファイルにすべてのテキストを保持するために、このファイルを使用します。
- **システムを複数の実行可能ファイルに分割**。.exe ファイルなど複数の実行可能ファイルとしてシステムを開発します。これらを組み合わせて顧客に提供することができます。実行可能ファイルをさまざまに組み合わせて CMVC ツールで管理します。

Rational Unified Process への影響

この項では、バリアントを導入することにより Rational Unified Process が受ける影響について簡単に説明します。

実装作業分野への影響

実装作業分野で次の部分を拡張する必要があります。

- 「**実装モデルの構成**」のタスクに次の手順を追加します。
- 開発する必要のあるシステムのバリアントを定義する方法
- バリアントを用意する必要のあるサブシステムを決定する方法
- 可変性を実現するために使用する可変メカニズムを決定する方法
- 「**システム統合計画**」のタスクでは、バリアントを考慮してさまざまなシステム・バリアントの統合方法を計画する必要があります。
- サブシステムのバリアント間での並行開発については、より詳細に記述する必要があります。例えば、「同一の」コンポーネントが切断された場合には何が起きるか、コンポーネントをどのように統合するか、などがあります。これは実装担当者の関連タスクの一部にも影響を与えます。

そのほかの実装タスクおよび/またはアクティビティーにも影響する場合があります。

ほかの作業分野への影響

バリアント、すなわちシステムのファミリーを開発すると、すべての作業分野が影響を受けます。前のセクションでは、実装作業分野に与えるいくつかの影響について説明しました。以下に、ほかの作業分野に与える影響を簡単にまとめます。

- **要求** — システムのバリアントを識別する方法を記述する必要があります。「各クラスの顧客が望むもの」などがあります。
- **分析と設計** — 設計モデルでバリアントをモデリングする方法、サブシステムのバリアントを設計する方法、バリアントを定義する方法を記述する必要があります。
- **テスト** — システムのファミリー (バリアント) をテストする方法を記述する必要があります。システムの各バリアントを別々のシステムとしてテストします。バリアントは統合テストにも影響を与えます。
- **管理** — サブシステムのバリアントごとに個別のチームにして、プロジェクトを編成する必要があります。大規模なプロジェクトでは、システムのバリアントごとに個別のプロジェクト管理者を置く場合もあります。
- **環境** — システムのバリアントの管理に役立つツールが必要です。
- **配置** — 最終製品を提供する顧客にいくつかのクラスがあるため、作業がより複雑になります。

Rational®
the software development company

Dual Headquarters:

Rational Software
18880 Homestead Road
Cupertino, CA 95014
Tel: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
Tel: (781) 676-2400

Toll-free: (800) 728-1212
E-mail:info@rational.com
Web:www.rational.com
International Locations:www.rational.com/worldwide