UML に基づく Web アプリケーション

Jim Conallen

Rational Software ホワイト・ペーパー

TP 157, 6/99
この資料の原案は、「Communications of the ACM」の 1999 年 10 月号 (ボリューム 42、ナンバー 10) に掲載されています。

Rational®
the software development company

目次

要約

Web アプリケーションは複雑さを増し、その重要性はますます高まっています。この複雑さを管理するにはモデリングが必要です。統一モデリング言語 (UML) は、ソフトウェア集約的なシステムをモデリングするために使用する標準的な言語です。UML を使用して Web アプリケーションのモデリングを試みると、一部のコンポーネントが標準の UML モデリング要素にうまく適合しないことが明らかになります。システム全体 (Web コンポーネントと従来の中間層コンポーネント) で 1 つのモデリング表記法に従うには、UML を拡張する必要があります。このホワイト・ペーパーでは、形式的な拡張メカニズムを使用した、UML の拡張について説明します。拡張は、Web 固有のコンポーネントがシステムのほかのモデルに統合されるように設計され、Web アプリケーションの設計者、実装担当者、アーキテクトに対して適切な抽象度と詳細さのレベルが公開されます。

概説

この数年の間に、IT 業界では「Web アプリケーション」という新しい用語が使用されるようになりました。ビジネス・ソフトウェア・システムの開発元では、多くのビジネス関連以外のソフトウェア開発と共に、Web アプリケーションの構築も計画されているようです。このアーキテクチャーを早くから採用した多くの開発者に対して、「Web アプリケーション」という用語は、システム自体と同じように、小規模な Web サイト・アドオンから強固な n 層アプリケーションに発展してきました。Web アプリケーションが、世界中の数万人のユーザーに対して同時にサービスを提供することもまれではありません。Web アプリケーションのアーキテクチャーを構築することは、重要なビジネスです。

実際に確認してみると、Web アプリケーションという用語は人によって意味が違っています。Web アプリケーションは Java を使用するものであるという意見もあれば、Web サーバーを使用するものが Web アプリケーションであるという考え方もあります。一般的な合意は、その中間のどこかにあります。このホワイト・ペーパーの目的として、ここでは Web アプリケーションという用語を、ユーザーの入力 (ナビゲーションとデータ入力) がビジネスの状態に影響を与える Web システム (Web サーバー、ネットワーク、HTTP、ブラウザー) と定義します。この定義により、Web アプリケーションはビジネスの状態を伴うソフトウェア・システムであり、その「フロントエンド」は大部分が Web システムを通して得られると確立されます。

Web アプリケーションの一般的なアーキテクチャーは、クライアント・サーバー・システムのアーキテクチャーにいくつかの相違点を加えたものになります。Web アプリケーションの最も重要な利点の 1 つに、その配置があります。Web アプリケーションの配置とは、通常、ネットワークにサーバー側のコンポーネントを設定するということです。クライアント側には、特別なソフトウェアや構成は必要ありません。もう一つ別の重要な相違は、クライアントとサーバーの通信の特性です。Web アプリケーションの主要な通信プロトコルは HTTP です。これは、通信のスループットを最大化する代わりに、頑強性と障害への耐性を目的として設計されたコネクションレスのプロトコルです。Web アプリケーションにおけるクライアントとサーバー間の通信は、サーバー側とクライアント側のオブジェクト間の直接通信ではなく、通常は Web ページのナビゲーションに向けられています。抽象化の特定のレベルでは、Web アプリケーションにおけるすべてのメッセージ処理を、Web ページ・エンティティーの要求や受信として表現できます。一般的に、Web アプリケーションのアーキテクチャーは、動的な Web サイトのアーキテクチャーとそれほど相違がありません。

Web アプリケーションと Web サイト (動的なサイトを含む) の相違は、その使用にも影響します。Web アプリケーションはビジネス論理を実装し、その使用により (システムによって獲得される) ビジネスの状態が変化します。この点は、モデリング作業の焦点を定義しているため重要です。Web アプリケーションはビジネス論理を実行します。したがって、システムの最も重要なモデルは、プレゼンテーションの詳細ではなく、ビジネス論理とビジネスの状態に焦点を置きます。表現は重要です (そうでなければ、システムは何もできません) が、ビジネスの問題とプレゼンテーションの問題は明確に分けます。プレゼンテーションの問題が重要な場合、または複雑になった場合は、これらもモデリングします。ただし、必ずしもビジネス論理モデルの一部としてモデリングする必要はありません。また、プレゼンテーションで使用されるリソースは美的感覚に関する部分であり、ビジネス・ルールの実装にはあまり関係ありません。

Web システムの開発に関連する方法/表記法の 1 つとして、RMM (Relationship Management Methodology) があります。RMM は、イントラネットとインターネット Web システムの設計、作成、メンテナンスに関する方法論です。RMM の主要な目的は、データベースで駆動される動的な Web サイトの保守費用を削減することです。RMM では、設計を議論しやすくするために、システムを視覚的に表現することが提案されています。反復的なプロセスでは、Web ページの視覚的な

要素と、それらのデータベース・エンティティーとの関連が分解されます。RMM は、動的な Web サイトの作成と保守のための「総合的な」方法です。

Web アプリケーションを構築する場合は、RMM では十分ではありません。ビジネス論理中心の Web アプリケーションには、ビジネス論理を実装するための多くの技術的なメカニズムが含まれます。 RMM 表記法ではこれらのメカニズムを適切に扱うことができません。クライアント側のスクリプト、アプレット、ActiveX コントロールなどの技術は、システムのビジネス・ルールを実行するために非常に役立ちます。また、Web アプリケーションは、分散オブジェクト・システムの導入メカニズムとしても使用できます。アプレットや ActiveX コントロールには、Web サーバーと独立して、RMI や DCOM を通じてサーバー側のコンポーネントと非同期に対話するコンポーネントを含めることができます。高機能のアプリケーションでも、クライアント側で複数のブラウザーのインスタンスやフレームを利用します。これらは、独自の通信メカニズムを確立し保守します。

これらのすべてのメカニズムはシステムのビジネス論理に役立つため、モデリングする必要があります。また、これらが表しているのはビジネス論理の一部だけであり、システムの残りのモデルと統合される必要があります。多くの場合、ビジネス論理の大部分は、サーバー側の階層のいずれかにある Web サーバーで実行されます。どのモデリング言語と表記法を選択するかは、通常、サーバー側アプリケーションのニーズによって決まります。公式のオブジェクト・モデリング言語として OMG の UML を選択すると、システムは UML 表記法を利用してより表現しやすくなります。多くの場合、UML はソフトウェア集約型のシステムをモデリングするために選択される言語です。Web アプリケーションをモデリングする際の主な問題は、Web 固有のコンポーネントで実行されるビジネス論理を、アプリケーションのほかの部分と一緒にどのように表現するかということです。その答えは、システムのビジネス論理の実行を、Web 固有の要素や UML を使用する技術で表現する能力にあります。

このホワイト・ペーパーは、Web アプリケーションのモデリングに関する問題や解決策についての概要を説明しています。ここでは、アーキテクチャー上、特に Web アプリケーションに関して重要なコンポーネントや、UML を使用してこれらをモデリングする方法について説明します。また、読者が UML、オブジェクト指向の基本、Web アプリケーションの開発に精通していることを前提としています。ここで説明している作業は、次のような一般的な仮定に基づいています。

- Web アプリケーションは、より複雑化し、より重大なロールが割り当てられるソフトウェア集約的なシステムである。
- ソフトウェア・システムの複雑さを管理する 1 つの方法は、これらを抽象化し、モデリングすることである。
- 通常、ソフトウェア・システムには複数のモデルがあり、それぞれが異なる視点、抽象度と詳細さのレベルを表す。
- 抽象度と詳細さの適切なレベルは、開発プロセスにおける成果物とアクティビティーに依存する。
- ソフトウェア集約的なシステムをモデリングするための標準的な言語は、統一モデリング言語 (UML) である。

ここで説明されている概念やアイデアは、まもなく発行される書籍、「Building Web Applications with UML」でより詳しく説明されています。この書籍は、Addison Wesley Longman の Object Technology Series として、年内に発行される予定です。

モデリング

モデルは、詳細さを簡略化することにより、システムを理解する助けになります。何をモデリングするかについての選択は、問題の理解や解決策の形態に大きく影響します。Web アプリケーションは、ほかのソフトウェア集約的なシステムと同様、通常はモデルのセット (ユースケース・モデル、実装モデル、配置モデル、セキュリティー・モデルなど) で表されます。
Web システムだけで使用されるモデルに、サイト・マップがあります。 これは、システム全体の Web ページとナビゲーション経路を抽象化したものです。

現在、実践されているほとんどのモデリング技術は、Web アプリケーションのさまざまなモデル開発に適しています。これらについては、これ以上の議論は必要ありません。ただし、重要なモデルである分析/設計モデル (ADM) では、Web ページやそれに関連付けられる実行可能コードを、ほかの要素と一緒にモデルに含めることが困難です。

モデリングの方法を決定する場合、抽象度と詳細さについて適切なレベルを決定することは、モデルのユーザーに利点を与えるために重要です。一般的には、システムの成果物 (最終的な製品を生成するために作成され、操作される、これらの実際のエンティティー) をモデリングすることが最適です。Web サーバーの内部構造または Web ブラウザーの詳細を

モデリングしても、Web アプリケーションの設計者やアーキテクトの役に立ちません。ページ、ページ間のリンク、ページを構成するすべての動的なコンテンツ、クライアントで表示される動的なコンテンツをモデリングすることが重要です。設計者が設計し、実装担当者が実装するのは、これらの成果物です。クライアントとサーバー上のページ、ハイパーリンク、動的なコンテンツは、モデリングする必要があります。

以下は、これらの成果物をモデリング要素としてマッピングする手順です。例えば、ハイパーリンクはモデルの関連要素に対応します。ハイパーリンクは、ページ間のナビゲーション・パスを表します。この考えを拡張すると、ページはモデルの論理ビューのクラスに対応します。Web ページがモデルのクラスであるとすると、ページのスクリプトはクラスの操作に対応します。スクリプトがページ内で使用する変数は、クラスの属性に対応します。Web ページに (ページの動的コンテンツを用意している) サーバー上で実行されるスクリプトや、クライアントのみで実行される完全に別のスクリプト (つまり、JavaScript) が含まれると考えると、問題が発生します。この場合、モデルの Web ページ・クラスに注目すると、ユーザーがクライアントでページと対話しているときに、何の操作、属性、関係が (ページが準備されるときに) サーバー上でアクティブになるか、どれがアクティブになるかについて混乱が生じます。また、Web アプリケーションの形で配信される Web ページは、システムのコンポーネントとして最適にモデリングされます。Web ページと UML クラスを単純にマッピングするだけでは、システムをより深く理解する役には立ちません。

UML をそのまま利用するだけでは、特定の分野や特定のアーキテクチャーに関する意味を十分に把握できない状況が必ずあることを、UML の作成者は理解していました。このような状況を処理するために、UML のセマンティクスを拡張できる形式的な拡張メカニズムが定義されています。このメカニズムを利用して「ステレオタイプ」、「タグ付き値」、「制約」を定義し、これらをモデル要素に適用できます。

ステレオタイプは、モデリング要素に付与する新しい意味を定義する修飾子です。タグ付き値は、モデリング要素に関連付けることができるキー値のペアで、任意の値をモデリング要素に「貼り付ける」ことができます。制約は、モデルの整形式性を定義するルールです。これらは、フリー・フォームのテキスト、またはより形式的なオブジェクト制約言語 (OCL) で表現できます。

ここで説明する作業により、UML に Web アプリケーション用の拡張が導入されます。この拡張を完全に説明することはこのホワイト・ペーパーで扱う範囲を超えていますが、概念や説明のほとんどについては議論します。

モデリングに関する最終的なポイントは、ビジネス論理とプレゼンテーション論理の間に、明確な区別が必要であるということです。一般的なビジネス・アプリケーションの場合は、ビジネス論理だけを ADM の一部とします。 ボタンのアニメーションや吹き出しヘルプ、その他の UI の強化機能などのプレゼンテーションの詳細は、通常は ADM に属しません。 アプリケーションに別の UI モデルが作成される場合は、このモデルに配置します。ADM では、ビジネスの問題やソリューション・スペースの表現に引き続き重点を置く必要があります。今日では、Web ページの外観や操作性は専門家 (技術的なグラフィック・アーティスト) によって最適に設計され、実装されます。 従来の開発者が、これらの作業を行うことはありません。

Web アプリケーションのアーキテクチャー

Web アプリケーションの基本的なアーキテクチャーには、ブラウザー、ネットワーク、Web サーバーが含まれます。ブラウザーは、サーバーに「Web ページ」を要求します。各ページは、HTML で表現される、コンテンツとフォーマット命令が混在したものです。ページには、ブラウザーで解釈される、クライアント側のスクリプトが含まれる場合があります。これらのスクリプトは、表示ページの動的な振る舞いを定義します。 また、ブラウザー、ページのコンテンツ、ページに含まれる追加コントロール (アプレット、ActiveX コントロール、プラグイン) とやり取りを行う場合もあります。ユーザーは、ページのコンテンツを見たり、ページと対話します。ユーザーがページのフィールド要素に情報を入力し、これをサーバーに送信して処理する場合もあります。ユーザーは、ハイパーリンクを通してシステムの別のページに移動することもできます。いずれの場合も、ユーザーはシステムに対して、システムの「ビジネスの状態」を変化させる入力を与えています。

クライアントの視点から見ると、Web ページは常に HTML でフォーマットされた文書です。一方、サーバーの視点から見ると、「Web ページ」は自身をさまざまな方法で表します。最初に説明した Web アプリケーションでは、動的な Web ページは CGI (Common Gateway Interface) を使用して作成されていました。CGI は、ページ要求と共に渡された情報へアクセスするために使用される、スクリプトとコンパイルされたモジュールのインターフェースを定義します。一般的に CGI に基づくシステムでは、ページ要求に応答してスクリプトを実行できるように、Web サーバーに特別なディレクトリーが構成されます。CGI スクリプトが要求されると、サーバーはファイルの内容を (HTML フォーマットのファイルとして) 返す代わり

に、ファイルを適切なインタープリター (通常は PERL シェル) で処理し、実行し、出力を要求元のクライアントにストリームとして返します。このプロセスの最終的な結果は、要求元クライアントに返される HTML フォーマットのストリームです。ビジネス論理は、ファイルの処理中にシステムで実行されます。この間に、データベースや中間層コンポーネントなどのサーバー側のリソースとやり取りを行うことができます。

最近の Web サーバーは、この基本設計が改善されています。セキュリティーがより意識され、サーバーでのクライアントの状態の管理、トランザクション処理の統合、リモート管理、リソース・プーリングなどの機能が追加されています。 最新の Web サーバーは、これらの問題をひとまとめにして処理しています。これらの問題は、責任重大で、拡張しやすく、堅牢なアプリケーションのアーキテクトにとって重要です。

CGI スクリプトのロールに注目した場合、今日の Web サーバーは、スクリプト・ページ、コンパイル・ページ、これら 2 つの混成、という 3 つの主なカテゴリーに分類できます。最初のカテゴリーでは、クライアントのブラウザーが要求できる各 Web ページは、Web サーバーのファイル・システム上にスクリプト・ファイルとして表現されます。通常、このファイルは HTML とほかのスクリプト言語の組み合わせです。ページが要求されると、Web サーバーはこのページの処理を、ページを認識するエンジンに渡します。 最終的な結果は、要求元のクライアントに返される HTML でフォーマットされたストリームです。この Web サーバーの例は、Microsoft の Active Server Pages、Java Server Pages、Cold Fusion などです。

2 つ目のカテゴリー (コンパイル・ページ) では、Web サーバーはバイナリーのコンポーネントをロードして実行します。このコンポーネントは、スクリプト・ページと同様に、ページの要求と一緒に送信されたすべての情報 (フォーム・フィールドの値とパラメーター) にアクセスします。通常、コンパイルされたコードは要求の詳細を使用し、サーバー側のリソースにアクセスして、クライアントに返される HTML ストリームを生成します。ルールではありませんが、コンパイル・ページはスクリプト・ページよりも多くの機能を持つ傾向があります。コンパイル・ページの要求にパラメーターを渡すことにより、異なる機能が得られます。実際に、コンパイルされた 1 つのコンポーネントが、ディレクトリー中のすべてのスクリプト・ページの、すべての機能を含む場合もあります。このタイプのアーキテクチャーを表現する技術には、Microsoft の ISAPI と Netscape の NSAPI があります。

3 つ目のカテゴリーは、最初に要求されたときにコンパイルされるスクリプト・ページを表します。 その後のすべての要求では、このコンパイルされたバージョンが使用されます。元のページのコンテンツが変更された場合にだけ、ページがコンパイルされます。このカテゴリーは、スクリプト・ページの柔軟性とコンパイル・ページの効率性が歩み寄った方法です。

Web ページのモデリング

スクリプト・ページとコンパイル・ページのいずれの場合でも、Web ページは UML のコンポーネントと 1 対 1 で対応します。コンポーネントは、システムの「物理的」で置き換え可能な部分です。モデルの実装ビュー (コンポーネント・ビュー) は、システムのコンポーネントとその関係を表します。Web アプリケーションの場合、このビューはシステムのすべての Web ページと、ページ間の関係 (つまり、ハイパーリンク) を表します。Web システムのコンポーネント図は、あるレベルで、サイト・マップのようになります。

コンポーネントはインターフェースの物理的なパッケージを表すだけなので、ページ内のコラボレーションをモデリングするには適していません。この抽象化のレベルは、設計者と実装担当者にとって非常に重要であり、モデルの一部とする必要があります。まず、各 Web ページはモデルの設計ビュー (論理ビュー) の UML クラスであり、ほかのページへの関係 (関連) はハイパーリンクを表すと表現できます。ただし、Web ページがサーバーにしか存在しない機能やコラボレーションのセットを表すことができると考えられる場合や、完全に異なるセットがクライアントだけに存在すると考えられる場合、この抽象化は機能しません。このようなページの例として、Dynamic HTML (クライアント側のスクリプト) を出力の一部として使用するサーバーのスクリプト Web ページがあります。この問題から予想される対応は、クラスの各属性または操作をステレオタイプ化して、サーバーまたはクライアント側で有効であるかどうかを示すことです。この時点で、本来、単純化を行うためのモデルは、非常に複雑になります。

この問題に対するより適切な方法は、「問題の分離」の基本を考えることです。論理的に、サーバー上での Web ページの動作は、クライアントにおける動作とは完全に異なります。サーバーで実行されるときに、Web ページはサーバー側のリソース (中間層コンポーネント、データベース、ファイル・システムなど) にアクセスします (つまり、リソースとの関係があります)。クライアントでは、同じページ (HTML ストリーム出力によるページ) に対する振る舞いと関係のセットが完全に異なります。クライアントでは、スクリプト・ページは DOM (文書オブジェクト・モデル) を通じてブラウザー自身と関係を持ち、ページで指定される Java アプレット、ActiveX コントロール、プラグインとも関係を持ちます。用心深い設計者であれば、ク

ライアントに別の「アクティブな」ページとの関係を追加して、ほかの HTML フレームやブラウザーのインスタンスに表示することもできます。

問題を分離することで、Web ページのサーバー側の面とクライアント側の面を個別にモデリングできます。UML の拡張メカニズムを使用して、それぞれ (サーバー・ページとクライアント・ページ) のステレオタイプとアイコンを定義し («server page» と «client page»)、これら 2 つを区別します。UML のステレオタイプにより、モデリング要素の新しいセマンティクスを定義できます。UML のダイアグラムでは、ステレオタイプ化されたクラスは独自のアイコンか、ステレオタイプ名を «» で囲んで表記します。アイコンは概要図で役立ちます。 クラスの属性と操作が公開されているときには、簡単なタグを使用するのが最適です。

Web ページの場合、ステレオタイプは、クラスがクライアントまたはサーバーの Web ページの論理的な振る舞いの抽象化であることを表します。2 つの抽象化は、方向を持った両者間の関係で関連付けられます。この関連は、«build» としてステレオタイプ化されます。 これは、サーバー・ページがクライアント・ページを作成すると考えることができるからです (図 1)。動的な Web ページ (つまり、コンテンツが実行時に決定されるページ) はいずれも、サーバー・ページで作成されます。各クライアント・ページは 1 つのサーバー・ページで構築されますが、1 つのサーバー・ページから複数のクライアント・ページを構築することも可能です。

Web ページ間の一般的な関係は、ハイパーリンクです。Web アプリケーションのハイパーリンクは、システム内のナビゲーション・パスを表します。この関係は、«link» としてステレオタイプ化された関連を持つモデルで表現されます。この関連は、常にクライアント・ページを起点とし、クライアント・ページまたはサーバー・ページをポイントします。

ハイパーリンクは、Web ページの要求としてシステムに実装されます。 Web ページは、実装ビューのコンポーネントとしてモデリングされます。クライアント・ページへのリンク関連のほとんどは、クライアント・ページを構築するサーバー・ページへのリンク関連と等価です。これは、リンクが実際にはページの要求であり、クラスの抽象化ではないためです。Web ページのコンポーネントは両方のページの抽象化を実現するので、ページのコンポーネントによって実現されたクラスへのリンクは等価です。

タグ付き値は、リンク要求と一緒に渡されるパラメーターを定義するために使用されます。«link» 関連のタグ付き値 "Parameters" は、要求を処理するサーバー・ページで予測され、使用される、パラメーター名 (ならびに、オプション値) のリストです。図 2 では、SearchResults ページに含まれる GetProduct サーバー・ページへのハイパーリンクの数は一定していません (0..*)。各リンクの productId パラメーターの値は異なります。GetProduct ページは、productId パラメーターで指定される製品の ProductDetail ページを作成します。

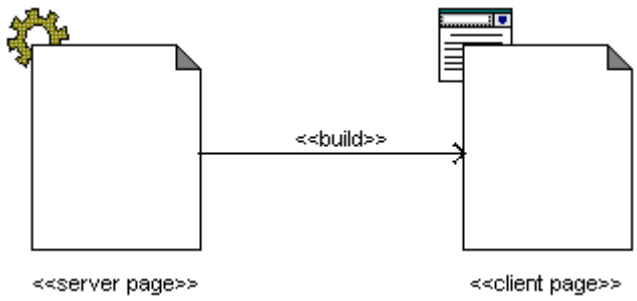

図 1 : サーバー・ページによるクライアント・ページの作成

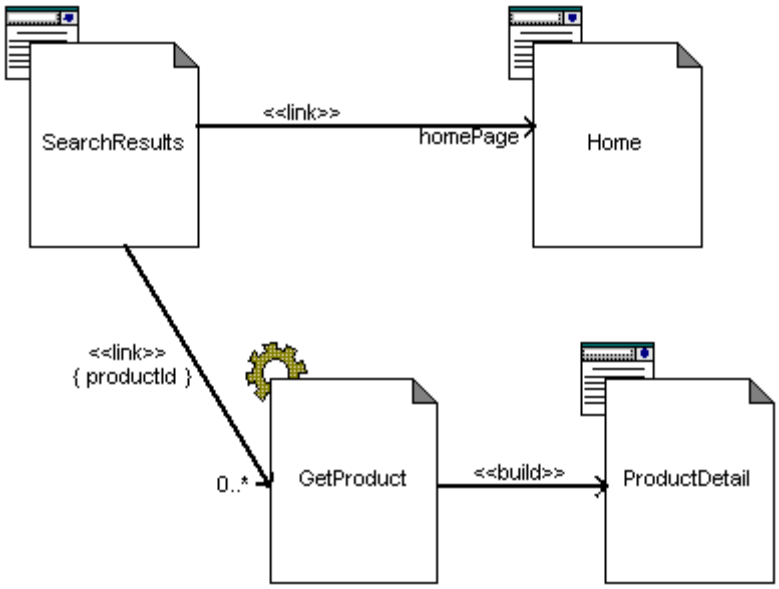

図 2 : ハイパーリンク・パラメーターの使用

これらのステレオタイプを使用すると、ページのスクリプトと関係を簡単にモデリングできます。«server page» クラスの操作は、そのページのサーバー側スクリプトの関数になります。 その属性は、ページ内のスコープを持つ (ページの関数によってグローバルにアクセス可能な) 変数になります。«client page» クラスの操作と属性は、同様にクライアントで参照可能な関数と変数になります。ページのサーバー側の面とクライアント側の面を別のクラスに分離することの主な利点は、ページとその他のクラス間の関係にあります。クライアント・ページは、クライアント側のリソース (DOM、Java アプレット、ActiveX コントロール、プラグイン) への関係を使用してモデリングされます (図 3)。サーバー・ページは、サーバー側のリソース (中間層コンポーネント、データベース・アクセス・コンポーネント、サーバーのオペレーティング・システムなど) への関係を使用してモデリングされます (図 4)。

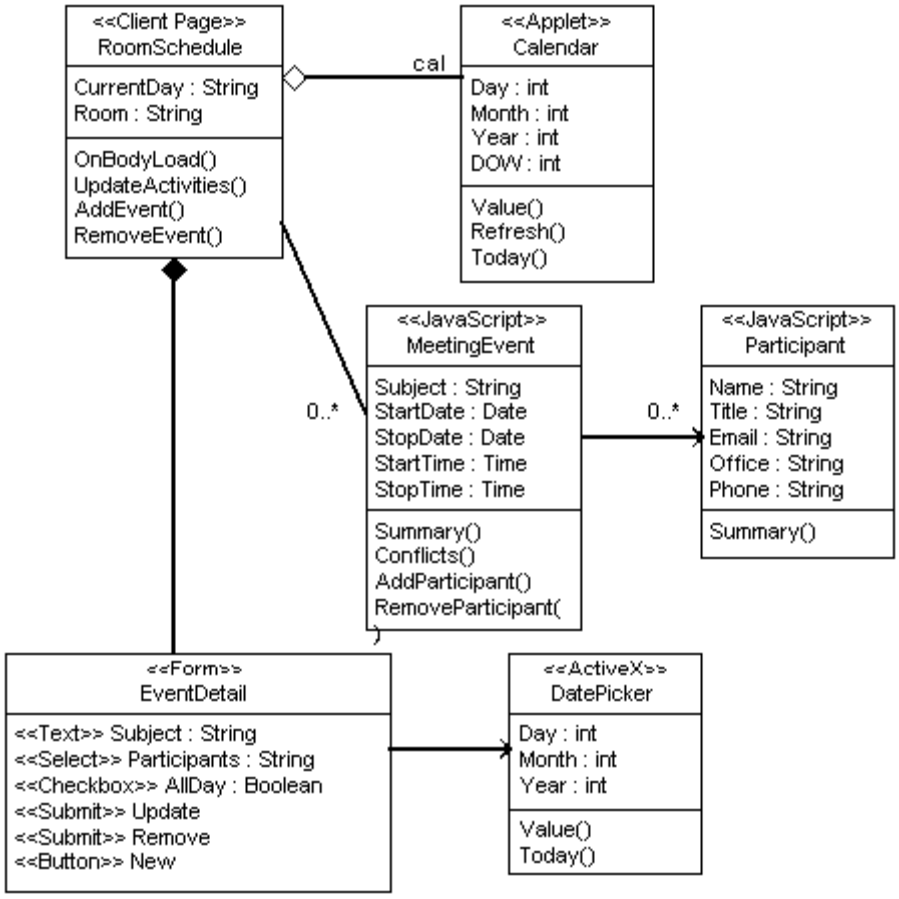

図 3 : クライアントのコラボレーション

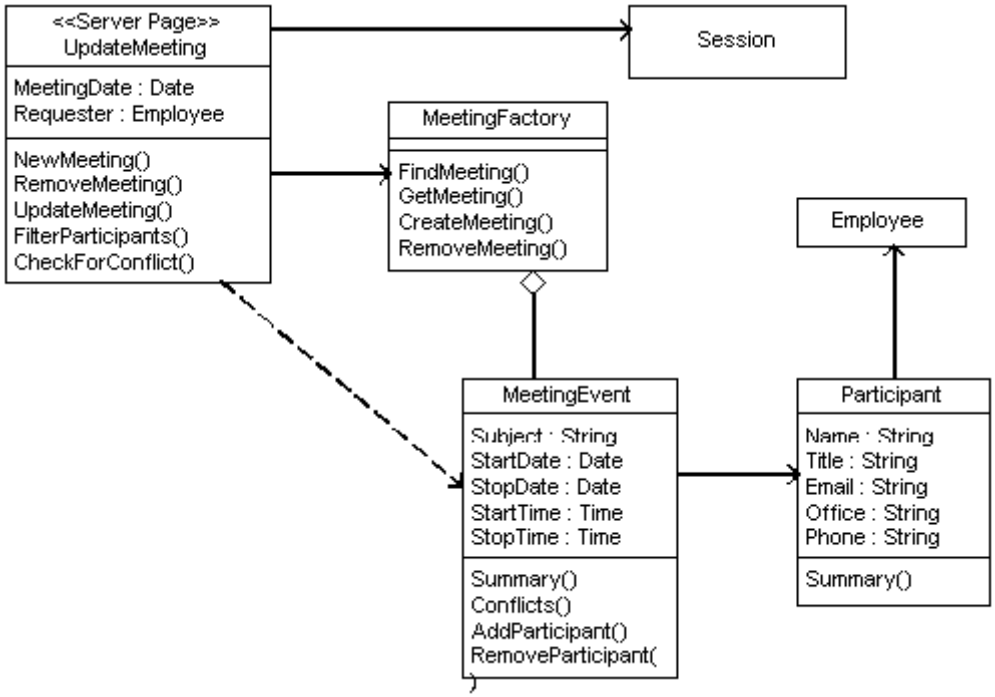

図 4 : サーバーのコラボレーション

クラスのステレオタイプを使用して Web ページの論理的な振る舞いをモデリングすることの最大の利点の 1 つは、サーバー側のコンポーネントによるコラボレーションを、ほかのサーバー側のコラボレーションと同様に表現できることです。«server page» は、システムのビジネス論理に参加するほかのクラスに過ぎません。より概念的なレベルでは、一般的にサーバー・ページはコントローラーのロールを引き受け、必要なビジネス・オブジェクトのアクティビティーを編成して、ブラウザーのページ要求によって開始されたビジネス目標を達成します。

クライアント側では、コラボレーションは少し複雑になる場合があります。これは、使用される技術の多様性によるものです。最も単純な場合、クライアント・ページは、コンテンツとプレゼンテーション情報の両方を含む HTML 文書です。ブラウザーは、ページ内のフォーマット命令を使用して HTML ページを表示します。場合によっては、別個のスタイル・シートが使用されることもあります。論理モデルでは、この関係はクライアント・ページから «Style Sheet» ステレオタイプ・クラスへの依存関係で表現されます。ただし、スタイル・シートは主にプレゼンテーションの問題であり、ADM では扱われません。

フォーム

Web ページの主要なデータ入力メカニズムはフォームです。フォームは、HTML 文書の <form> タグで定義されます。各フォームは、自身を送信するページを指定します。フォームには、HTML タグで表現される、多くの入力要素があります。最も一般的なタグは、<input>、<select>、<textarea> タグです。input タグはオーバーロードされ、テキスト・フィールド、チェック・ボックス、ラジオ・ボタン、プッシュ・ボタン、イメージ、隠しフィールド、その他のあまり一般的でないタイプを表します。フォームのモデリングには、クラスのステレオタイプ «Form» を使用します。«Form» には操作がありません。<form> タグで定義される操作は、実際にはクライアント・ページで所有されるからです。フォームの入力要素はすべて、«Form» クラスのステレオタイプ化された属性です。«Form» は、入力コントロールとして機能するアプレットや ActiveX コントロールと関係を持ちます。各フォームは、サーバー・ページ (フォームの送信内容を処理するページ) とも関係を持ちます。この関係は、ステレオタイプの «submit» です。フォームは完全に HTML 文書の一部であるため、UML のダイアグラムに強い集約の形式で表現されます。図 5 は、簡単なショッピング・カート・ページを示しています。 この図は、フォームを定義し、処理を行うサーバー・ページへの submit 関係を示しています。

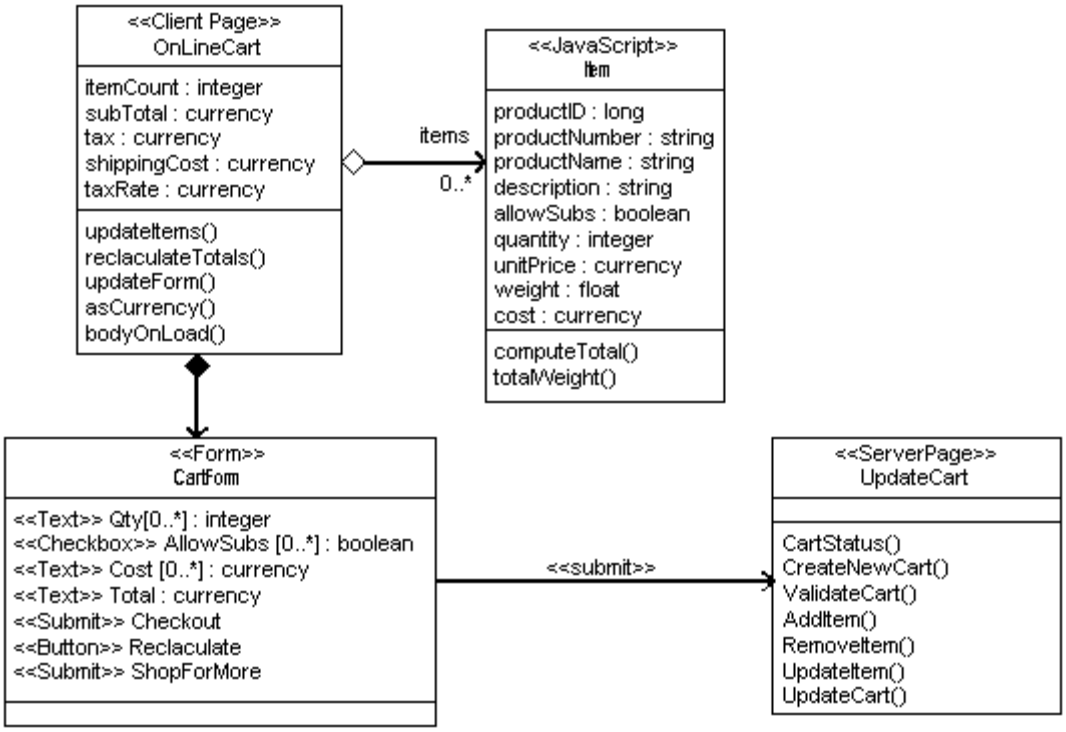

図 5 : サーバー・ページに送信されるフォーム

図 5 では、«JavaScript» ステレオタイプ・クラスが、ショッピング・カート内の項目を表すオブジェクトです。インスタンスの数が一定していないフィールドでフォームのプロパティーを表すために、配列の構文が使用されます。このショッピング・カートの場合、カートには 0 以上の項目を設定可能で、それぞれに Qty、AllowSubs、Cost、Total の <input> 要素があります。

クライアント・ページのすべての処理は JavaScript によって実行されます。JavaScript は型のない言語であるため、これらの属性に指定されているデータ型は、実装担当者の説明用に使用されています。JavaScript または HTML の入力タグを実装するときには、型は無視されます。これは、関数パラメーターにも適用されます。このダイアグラムでは十分に示されていませんが、関数パラメーターもモデルの一部です。

フレーム

Web サイトまたは Web アプリケーションでの HTML フレームの使用は、その導入の時点から議論の話題となっています。フレームを利用すると、ユーザーに対していつでも複数のページをアクティブにして表示できます。最も一般的なブラウザーの最新機能では、ユーザーのマシンで複数のブラウザー・インスタンスをアクティブにできます。Dynamic HTML のスクリプトとコンポーネントを使用して、これらのページは互いに対話できます。クライアントでの複雑な対話の可能性は重要であり、これをモデリングする必要性も高くなります。

アプリケーションでフレームまたは複数のブラウザーのインスタンスを使用するかどうかは、ソフトウェア・アーキテクトによって決定されます。その場合、前の説明と同じ理由で、このクライアント側の振る舞いのモデルは ADM で表現する必要があります。フレームの使用をモデリングするために、さらに 2 つのクラスのステレオタイプ «frameset » と «target» と、関連のステレオタイプ «targeted link » を定義します。frameset クラスはコンテナー・オブジェクトを表し、HTML の <frameset> タグに直接対応します。このクラスには、client page と target が含まれます。target クラスは、ほかのクライアント・ページから参照される、名前付きのフレームまたはブラウザーのインスタンスです。targeted link 関連は、特定の target で表示される、ほかのページへのハイパーリンクです。図 6 に示す例では、共通の概要ビューが 2 つのフレームを使用するブラウザーに表示されています。一方のフレームは target で名前が付けられ (Content)、もう一方のフレームは単純に 1 つのクライアント・ページを含みます。このクライアント・ページのフレームは、書籍の目次 (TOC) です。このペー

ジのハイパーリンクは、参照先が Content フレームに表示されるように設定されます。結果として、左側には目次が固定で表示され、右側のページには書籍の内容が章ごとに表示されます。

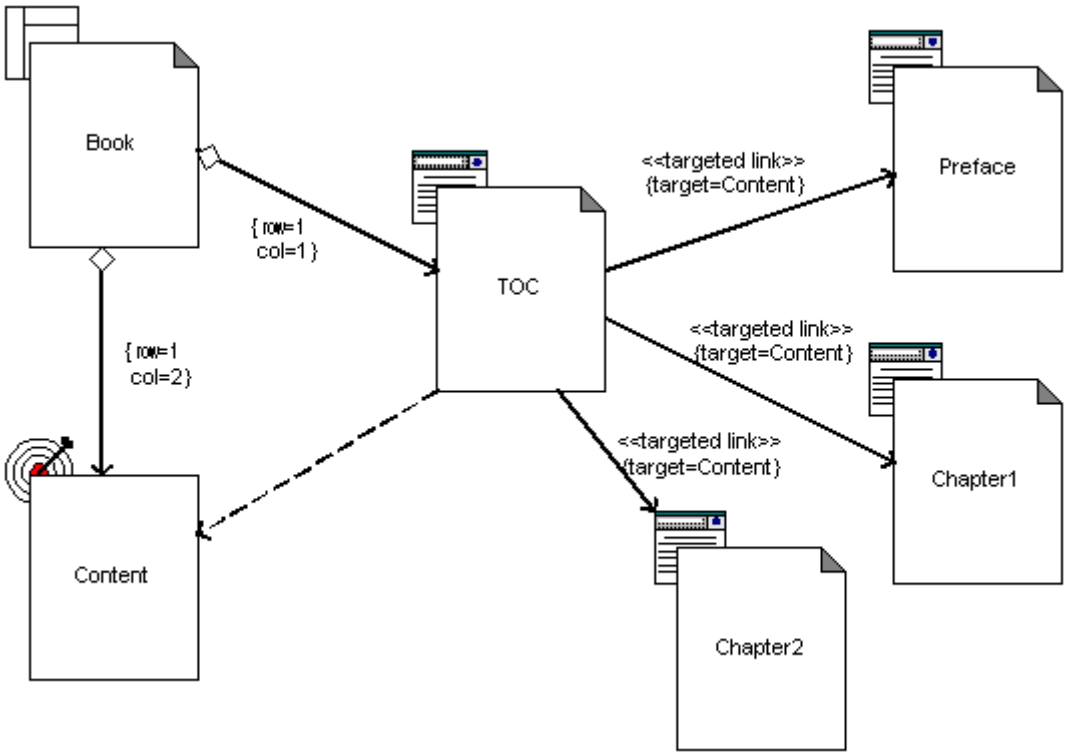

図 6 : フレームの例

実際のプレゼンテーション仕様の多くは、frameset と関連のタグ付き値によって獲得されます。frameset と target 間の集約関係またはクライアント・ページの 2 つのタグ付き値は、target またはページが属する frameset の行と列を指定します。targeted link 関連のタグ付き値 "Target" は、ページが表示される «target» を識別します。

target が frameset で集約されない場合は、ページが別のブラウザー・インスタンスを使用して表示されることを意味します。この表記法はクライアント・マシンの 1 つのインスタンスを表していることに注意してください。複数の独立した target はすべて同じマシンで実行されると仮定され、ダイアグラムは 1 つのクライアント・インスタンスのクライアント側の振る舞いを表現しています。より理解を深めるために、ほかの配置構成をモデルに文書化する必要がある場合もあります。

結論

ここで説明したアイデアと概念は、Web アプリケーションに固有の要素を UML を使用してモデリングする場合の問題点と解決策の概要です。ここでの目標は、詳細さと抽象度のレベルが Web アプリケーションの設計者、実装担当者、アーキテクトにとって適切となるように、Web 固有の要素をアプリケーションの残りの部分と統合する矛盾のない完全な方法を示すことです。Web アプリケーションに対する UML の正式な拡張の最初のバージョンは、ほぼ完成しています。この拡張により、アーキテクトと設計者に対して、Web アプリケーションの設計全体を UML を使って表現する共通の方法が提供されます。

この拡張に関する最新の情報は、Rational の Rational Rose と UML の項目に掲載されています。

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