IBMプロフェッショナル論文

IBM Professionals' Papers

ドキュメント技術に基づくシチュエーショナル・アプリケーション作成手法

神山 淑朗　嶋 幸太郎　三浦 圭司

A Method for Building Situational Applications Based on Document Technologies

Yoshiroh Kamiyama, Kotaro Shima and Keiji Miura

Web2.0 時代のアプリケーションは高度化しつつあるが，その一方で作り手は IT 技術のプロフェッショナルから利用者自身であるビジネス・ユーザーへという流れが加速しつつある．本論文では，こうした要求に応えるための Web アプリケーション作成手法を提案する．プログラミングではなくドキュメント技術を活用してアプリケーションを記述する手法により，欲しいときに欲しい機能を利用者自身の手で作る「シチュエーショナル・アプリケーション」の作成環境が実現可能であることを示す．また，その作成環境のインフラとなるブラウザー・ツール・プラットフォームの実装を通じて提案手法の有効性を確認する．

Applications in the Web 2.0 era are becoming increasingly complicated and sophisticated, while applications are increasingly developed or customized by business users themselves, not IT professionals. In this paper, we propose a way to create web applications that meet these requirements. We show how we can achieve a development environment for "situational applications," which are usually built to solve an immediate, specific business problem, by utilizing document technologies, not programming. We also show the effectiveness of this idea by implementing a browser-based tool platform, which can act as the infrastructure of such an environment.

Key Words & Phrases：Web 2.0，エンタープライズ 2.0，SaaS，シチュエーショナル・アプリケーション，
Web 2.0, Enterprise2.0, SaaS, Situational Application

1．はじめに

Web2.0 というキーワードで語られる Web 技術の新しい利用形態が活発化しつつある［1］［2］．現在の Web2.0 のサービスは無料で利用でき，個人を対象としたものが多いが，近年，その技術やコンセプトが企業内の情報システムの在り方にも大きな影響を与え始め，「エンタープライズ 2.0」と総称されている［3］［4］．業務アプリケーションの Web2.0 化の要求に伴い，解決すべき幾つかの課題が生じてきた．

従来はブラウザー上で高度な表現力・操作性を持つアプリケーションを実現するのは困難とされてきた．外部プログラムによりそれを克服する試みが数多くなされてきたが，特定のプラットフォームや外部プログラムに依存したくないという要求は根強く，多くは広く受け入れられるまでには至らなかった．そのような中，古くから存在する JavaScript，ダイナミック HTML，XML HttpRequest といった一般的なブラウザーに標準で備わる技術が「再発見」され，新しい使い方，従来の想定をも超える使い方が考案され，ゼロ・フットプリント（外部プログラムをインストールしない）でありながら高度なアプリケーションが実現されるようになってきた［5］．それに伴い，ブラウザー上のアプリケーションはますます複雑化・高度化しつつあり，開発環境やクロス・ブラウザーの問題もあるため，開発効率が重要な課題となってきた．

また，業務用ソフトウェアの提供形態はパッケージ・ソフトからインターネット上のサービスとしてソフトを提供する SaaS（Software as a Service）［6］の形態へシフトしつつある．ここで，業務内容に合わせたアプリケーションのカスタマイズ性が重要なポイントとなっている．ソフトウェア配布の点では，一般にパッケージ・ソフトに比べれば Web アプリケーションの方がカスタマイズ要求に対応しやすいと考えられるが，その都度サーバー・サイドの開発が発生するとしたら決して容易なことではない．

さらに，カスタマイズをサービス提供者側ではなくユーザー側で行いたいという要求がある．アプリケーションの

提出日：2007年9月3日　再提出日：2008年6月30日

利用者自身が自分のニーズを満たすのに十分な機能を持つ簡単なアプリケーションをその場で作って使うという形態はシチュエーショナル・アプリケーションと呼ばれ，企業のITの在り方に変化をもたらそうとしている[7]．シチュエーショナル・アプリケーションにおいては，ITの専門家でないビジネス・ユーザーが既存のサービスやコンポーネントを自由に組み合わせて目的のアプリケーションを作ったりカスタマイズしたりできる世界を目指している．それを実現するためには，目的のアプリケーションを簡単かつ効率的に，しかもプログラミングすることなしに作ることを可能にするインフラの提供が課題である．

本論文では，前述の要求に応えるための技術的手法を提案し，それを実現するブラウザー・ベース・ツールのプラットフォームの実装により有効性を確認する．

以下，2章で提案技術の基本コンセプトについて述べた上で，3章，4章でそのアプローチにより実現されることを整理する．5章ではシチュエーショナル・アプリケーションの要件およびそれを満たす上での提案技術の優位性について述べ，6章で具体的な実装例を示す．

2. ドキュメントによるアプリケーション記述

本章では，高度でプラットフォームに中立なWebアプリケーションを「効率良く」作成でき，「カスタマイズ性」に優れ，「シチュエーショナル・アプリケーション」の世界を実現可能にするための技術の基本コンセプトについて説明する．

2.1 サーバー・サイドWeb技術の問題点

JSP（JavaServer Pages）/JSF（JavaServer Faces）[8]に代表される動的なWebページ生成技術では，ビジネス・ロジックやカスタム・タグのレンダラーのプログラムをJavaで開発する．ページはカスタム・タグやスクリプト・レットを使って記述し，サーバー・サイドのプログラムによりレンダリング（カスタム・タグをHTMLやJavaScriptに変換）される．ブラウザーは生成されたHTML・JavaScriptを解釈し，表示・実行する．この方式は従来型のWebアプリケーションの構築に広く使われているが，前述の要件を満たす観点では次のような問題点がある．

（1）作成したページやその作成方法はサーバー・サイドの技術（J2EE，PHPなど）に強く依存してしまう．
（2）HTMLやJavaScriptコードを動的に生成する手法は複雑になりがち．
（3）サーバーを動かさないと表示や動作の確認ができない．
（4）ページ作成ツールはサーバー・サイドでのレンダリング結果をエミュレートしなければならない．
（5）新規に作成，またはカスタマイズしたページは，ビルド，デプロイといった工程を経なければ実行できない．

2.2 アプリケーションとドキュメントの融合

かつての典型的なアプリケーションは，C/C++といった言語で記述し，コンパイルすることによって得られるオペレーティング・システム（OS）に密接に結びついた実行形式であった．やがて，OSに依存しないバイトコードを生成するJavaなどの言語が広く使われるようになり，さらに，Ruby/PHPなどのコンパイルを必要としないインタープリター型の言語でアプリケーションを記述するケースが増えてきた．さらに，アプリケーションのUI画面をXMLで宣言的に記述する例も見られるようになった．コンパイル不要・宣言的といった特徴は，プログラムというよりはドキュメントに近く，「アプリケーションのドキュメント化」が進行しつつあると言える．

一方，かつての典型的なドキュメントは，ワード・プロセッサーの文書やHTMLに代表される純粋に文字情報を伝える媒体であった．しかし，ワード・プロセッサーにはマクロ言語が搭載され，HTML文書にはJavaScriptやダイナミックHTMLが埋め込まれるようになり，文書に動作・対話性が与えられることが一般的になった．この傾向は，「ドキュメントのアプリケーション化」ととらえることもできる．

こうしたアプリケーションとドキュメントの歩み寄りは，「アプリケーションの対話性は欲しいが，ドキュメントの手軽さで書きたい」という要求の現れでもある．多様性を増すヘテロな環境においてプラットフォームに中立なアプリケーションを構築したいという要求や，アプリケーションのライフサイクルの短縮に伴う開発・変更の容易性への要求が増す中で自然な発想と言える（図1）．

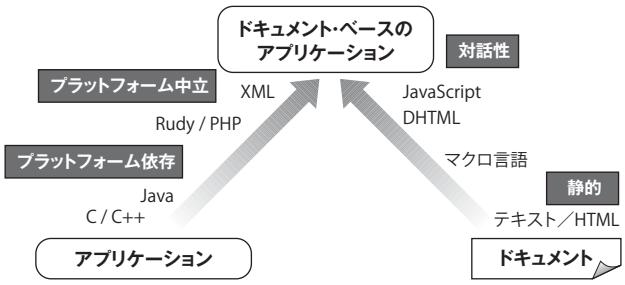

図1．アプリケーションとドキュメントの融合

2.3 クライアント・サイド・レンダリング

そこで，今回提案する手法では，プログラムではなく，ドキュメントの技術を使ってアプリケーションを記述する．

そのドキュメント・モデルでは，カスタム・タグを使ってページを記述するが，そのページはクライアント・サイドでレンダリングされる．つまり，ブラウザーはカスタム・タグの入ったページを受け取り，JavaScript によってタグを対応する UI コンポーネントの実体に置き変えることでページの表示を行う．さらに，UI だけでなくアプリケーションの動作もタグで記述する．これにより，2.1 節で挙げた問題は次のように解決される．

(1) 作成したページはサーバー・サイドでは解釈しないので，サーバー・サイドの技術には依存しない．
(2) HTML や JavaScript を動的に生成しない．
(3) サーバーがなくてもローカル・ファイルだけで表示・実行が可能．（デバッグが容易）
(4) ページ作成ツールは実行時と同じコンポーネントを編集時にも利用可能．（後述）
(5) 新規に作成，またはカスタマイズしたページは，即座に実行可能．（後述）

3. ブラウザー上でのオーサリング

ビジネス・ユーザーがコンテンツ編集の要領で簡単にアプリケーションを作成するためには，ドラッグ・アンド・ドロップなどの簡単な操作で見た目を確認しながら編集できるオーサリング・ツールの提供が不可欠である．本章では，2 章で述べたドキュメントによるアプリケーション記述のアプローチが，そのようなツールの実現に有効であることを示す．

3.1 真の WYSIWYG の実現

IT の専門家でないビジネス・ユーザーがアプリケーションを作成することを可能にするには，WYSIWYG（what you see is what you get）型のオーサリング・ツールの提供が必要となる．提案するドキュメント・モデルでは，プログラムを使わずにタグだけでアプリケーションを表現するため，ツールで扱いやすいという利点がある．

さらに，本来の意味の通りの「真の WYSIWYG（True WYSIWYG）」を実現可能である．一般的な WYSIWYG 型の Web オーサリング・ツールは次のような問題がある．

(1) サーバー・サイドでレンダリングされるタグは表示結果を擬似的に（画像を使うなどして）エミュレーションしなければならない．
(2) コンポーネントの動作までエミュレートするのはコンポーネントそのものを作ることにも匹敵し，現実的ではない．
(3) エミュレーションは実際の表示と正確に同じではない．細かい調整をするには，プレビュー画面に切り替えて表示を確認しながら作業する必要がある．
(4) JavaScript や DHTML を使って作られた複雑なコンポーネントをサポートするのが困難．

これに対し，クライアント・サイド・レンダリング方式ではタグを実際の表示に展開する処理がクライアント側で行われるため，エディター上でも動的に実際の表示ができる．このとき，アプリケーション実行時に使う実際のコンポーネントをエディター上での編集用にそのまま使うことができるため，オーサリング・ツールはエミュレーションを提供する必要がない．しかも，エディターの編集領域の中で実際とまったく同じ表示・動作を再現することができる．

3.2 即時フィードバックの実現

アプリケーションを構成する UI コンポーネントの中には，グラフ（**図２右**）やデータ・グリッド（**図２左**）のように，データ・モデルと結び付けられ，そのデータをビジュアルに表示するものがある．通常，データはサーバー上に存在するため，アプリケーションをサーバー環境で実行して初めて実際のデータと共に動作を確認できる．従って，オーサリング・ツールによってはダミーのデータを提供して実行時の雰囲気に少しでも近づける工夫をしているものもあるが，UI 画面編集はデータなしの状態で行うことも多い．

一方，クライアント・サイド・レンダリング方式の場合，UI コンポーネントにとって，実際の実行環境で実行中なのか，エディターの編集領域内で実行中なのか，違いはない．従って，エディター上でサーバーのデータと正しくバインドされれば，編集中でありながらも即座に実際のデータが表示される（即時フィードバック，Immediate Feedback）．この特徴により，アプリケーションの実行時

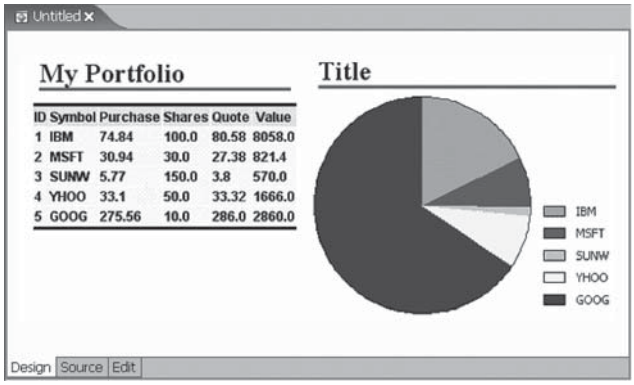

図 2. 編集領域内のコンポーネント

の様子を確認しながら編集作業を行うことが可能となる（図 2）.

3.3 ライブ編集の手法

真の WYSIWYG および即時フィードバックは，実行時と同じ本物の UI コンポーネントを編集時にも使用することで実現されるが，単に編集領域に実行時と同じようにコンポーネントを置くだけで済むわけではない．例えば,ドラッグ・アンド・ドロップ操作でツリー・ビューからデータ・モデルをコンポーネントにバインドしようと思ったら，コンポーネント側がそのようなドロップを受け付ける（ドロップ・ターゲットになる）ように作られていなければならない．しかし，コンポーネントに編集用の機能を実装するのは望ましくない．

そこで，筆者らは編集時メソッド（Design-time Methods）およびヘルパー（Helper）と呼ぶ概念を考案した（図 3）．コンポーネントは JavaScript のオブジェクトとして実装されるため，メソッドやプロパティーを外部から追加あるいは上書きすることが可能である．この性質を利用し，コンポーネントが編集ツール上でインスタンス化されたときにメソッドを追加（mix-in）する．ドロップ･ターゲットの設定，そのハンドラー，マウスによる移動処理，フォーカス枠の表示など，幾つかの機能はどのコンポーネントにも共通して必要なので，ツール側が編集時メソッドとして提供する．

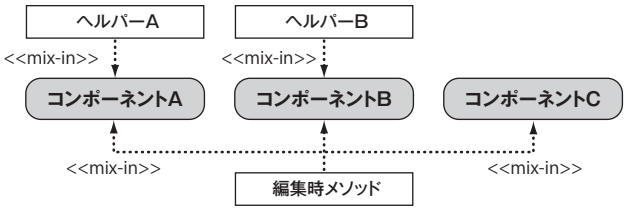

図 3．編集時メソッドとヘルパー

一方，コンポーネントに特有な編集時専用の機能を持たせたい場合もある．例えば，ラベル・コンポーネントは実行時には文字列を画面に表示するだけだが，編集時にはダブル・クリックして入力フィールドを表示させ，文字列を編集する機能を持たせたいかもしれない．そのような場合はコンポーネントごとにヘルパーとして実装しておくと，編集ツール上で自動的にコンポーネントにその機能が追加される．

4．マークアップによる動作記述

3 章までに，ドキュメントによるアプリケーション記述のアプローチにより，コンポーネントの配置およびデータとのバインド操作を,実行時と同じ表示を確認しながら行えるツールを実現可能であることを示した．本章では，さらにそのアプローチでアプリケーションの動作を記述する手法について述べる．

4.1 コンポーネント間の連携

アプリケーションは通常複数のコンポーネントから構成されるため，必然的にそれらの連携が必要となってくる．コンポーネント間の連携にはさまざまなレベルがある．例えば，互いを知り密接な関わりがあるコンポーネント同士であれば，直接的なメソッド呼び出しやプロパティーの参照が適当であるかもしれない．

提案する手法では，コンポーネント同士はできる限り疎な関係で，かつ自由な組み合わせが可能であることを目指すため，基本的には Pub/Sub（Publish/Subscribe）デザイン・パターンによるメッセージ通信で連携を行う．Pub/Sub を使えばメッセージの配信元があるトピックに対してメッセージを送信（Pub）すると,そのトピックを受信（Sub）しているすべての受信者がそのメッセージを受け取ることができる．ここで，送信側・受信側がメッセージを受け渡すのに必要なのはトピック名だけである．そこで，例えば送信側コンポーネントは自分の id をプレフィックスとして SelectionChanged などのメッセージを送信するように実装しておく．そのメッセージを受信したい場合は，次の例のようにしてタグの属性で送信者の id を指定すればよい．

```
<editArea id="editArea1"/>
<propertiesView target="editArea1"/>
```

以上のようにして，マークアップだけでコンポーネント間の連携を容易に指定可能であることがわかる．

4.2 コンポーネント間のデータ交換

コンポーネント間でデータを受け渡したいことがある．例えば，地名を入力として緯度･経度情報を返すジオコーディング・コンポーネント（Geocoding）と，緯度・経度情報を入力としてその地点の地図を表示する地図コンポーネント（Map）を連携することを考える．一般的なプログラミング・モデルでは，Geocoding コンポーネントのボタンにイベント・ハンドラーを設定し，Map コンポーネントのメソッドを直接呼び出すであろう（図 4）．

しかし，ここでの目的は，そのようなプログラミングをす

```
Button.onclick = function(e){
  ....
  map.update(lat, lng);
}
```

図 4．ボタンのイベント・ハンドラー

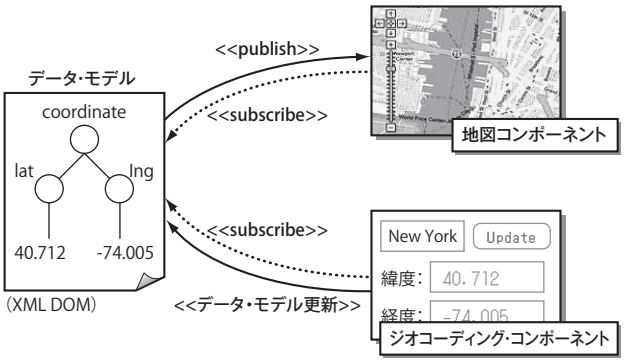

図 5. データ・モデルの共有によるデータ交換

ることなしにドラッグ・アンド・ドロップなどの簡単な操作で実現することである.そこで,提案する手法では,両コンポーネント間でデータ・モデルを共有し，同期を図ることでデータ交換をする（図 5）. この手法では，両コンポーネントは同じデータ・モデルの変更を監視（subscribe）する. それにより，一方のコンポーネントがデータ・モデルを更新すると，他方は即座にその変更後のデータを共有することができる. 例えば，Geocoding コンポーネントは，ボタンが押され，緯度・経度が求まると，その情報でデータ・モデルを更新する. データ･モデルは変更を通知（publish）する. それを監視している Map コンポーネントは通知を受け取り，地図の表示を更新することができる.

この仕組みによると，連携に必要な編集操作は，1 つのデータ･モデルを 2 つのコンポーネントに結び付ける（バインド）だけでよい. すなわち，あらかじめ提供されている緯度・経度データ・モデルをドラッグし，Geocoding と Map の両方にドロップするといった簡単な編集操作で連携を可能にすることができる.

ここで生成すべきマークアップは，図 6 に示す例のように簡単なものになる. データ・モデルを表す <Model> タグの id を両コンポーネントが参照している. ボタンのイベントの処理やアクションの記述といった複雑さは登場しない.

```
<Model id="model1" src="coordinate.xml"/>
<Map modelId="model1"/>
<Geocoding modelId="model1"/>
```

図 6. 生成するマークアップの例

5.シチュエーショナル・アプリケーション作成環境の実現

IT の専門知識も，プログラミング知識もないビジネス・ユーザーが自らの手でアプリケーションを作り，カスタマイズし，ビジネスの現場で活用するといったことを可能にするには，次のような要件が必要となる.

（1）プログラミングではなくコンテンツ編集の要領で作ることが可能
（2）作ったアプリケーションの共有が容易
（3）作ると同時に実行可能

前章までに（1）が可能であることを述べた. また,（2）はゼロ・フットプリント Web アプリケーションの最大の利点でもあり，問題はない. ここでは，（3）が実現可能であり，さらにそれに対して複雑さを低減する設計が可能であることを示す.

5.1 インスタント・デプロイメント

2.1 節で述べたサーバー・サイド Web 技術では，アプリケーションを新規作成またはカスタマイズするには，コーディング→ビルド→デプロイといった工程を経なければならない. サーバー･サイド･プログラミングの知識を必要とし，開発エンジニアがやらなければならない作業である.

一方，今回提案するドキュメント・モデルでは，レンダリングにサーバー・サイドの処理を必要としないので，サーバー・サイドの技術に固有のビルドやデプロイといったステップを踏む必要はない. 従って，オーサリング・ツールで作成したページをサーバーにアップロードするだけでよく，アップロード完了と同時に実行が可能である（インスタント・デプロイメント）.

また，このときサーバー・サイドに要求される機能は，基本的には「要求された静的なページをそのまま送信する」ことだけであるため，サーバー・サイドのシステムはどのようなプラットフォームや技術を使っても実現可能なほどシンプルであり，柔軟な設計が可能である.

5.2 RESTful な設計

HTTP でリクエストするとシンプルな XML を返す簡易 Web API を REST と総称することが多いが，特に REST の本来の原則に従うシステムは RESTful と呼ばれる［9］. RESTful システムは次の特徴を持ち，複雑さを低減する効果がある.

（1）サーバー・サイドはステートレス
（2）すべてのリソースは固有の URI を持つ
（3）GET/POST/PUT/DELETE でリソースを操作

例えば，JSF の場合はサーバー・サイドに存在するライ

フサイクルや UI コンポーネント・ツリーでステートを管理しながら処理が進む．そのため，想定されたシーケンス通りにページ遷移しなければ正常に動作しないことになり，そのままでは RESTful な設計には向かない．

一方，クライアント・サイド・レンダリング方式ではサーバー・サイドには一切ステートを持たないので，RESTful な設計が可能である．シチュエーショナル・アプリケーション作成環境においては，ページをリソースととらえ，リソースの操作は RESTful の原則に従うことで，クライアント，サーバー双方で見通しの良い設計を行うことができる．

6.ブラウザー・ツール・プラットフォームの実装

これまで，ドキュメントによるアプリケーション記述の手法により，マークアップだけでコンポーネントを組み合わせたアプリケーションの表現が可能であり，それゆえにそのオーサリング・ツールが実現可能で，さらに作ると同時に実行させることが可能であることを述べてきた．その種のニーズにはさまざまな要求のバリエーションが存在し，今後ますます広がりを見せていくと考えられる．そこで，筆者らはさまざまなニーズに合わせたブラウザー・ベースのオーサリング・ツールを効率よく開発可能なブラウザー・ツール・プラットフォームを実装した．

6.1 プラットフォームのアーキテクチャー

ブラウザー・ツール・プラットフォームは Eclipse プラットフォーム［10］に影響を受けたアーキテクチャーとなっている（図 7）．Eclipse 同様にビューやエディターという概念があり，それらを組み合わせる形でツールが構成される．ツールで共通に必要となる，アクション，ドラッグ・アンド・ドロップ，セレクション，コマンド，イベントなどの機構はプラットフォームが提供する．

6.2 ツールの構成

プラットフォームのユーザーであるツール開発者は，既存のビューやエディターを拡張したり，独自に実装して追加することができる．ここで，ツールで作成するアプリケーションと同様に，ツール自身もドキュメントで記述されている（図 8）．従って，ビューの追加やレイアウト変更はドキュメントのタグを書き換えるだけで簡単に実現できる． また，ビュー間あるいはビュー

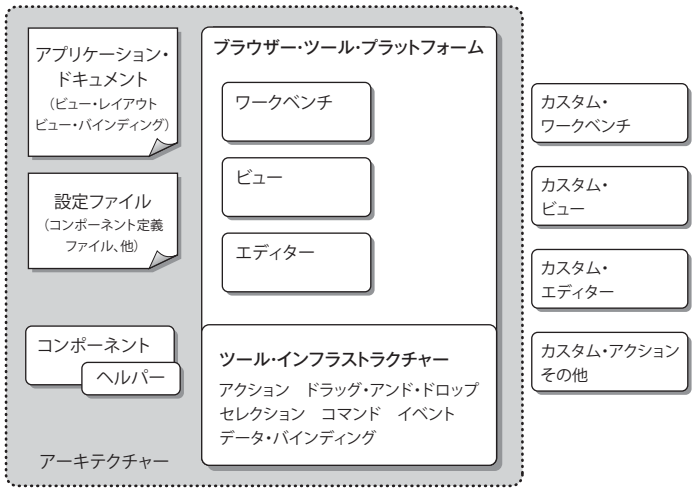

図 7．プラットフォームのアーキテクチャー

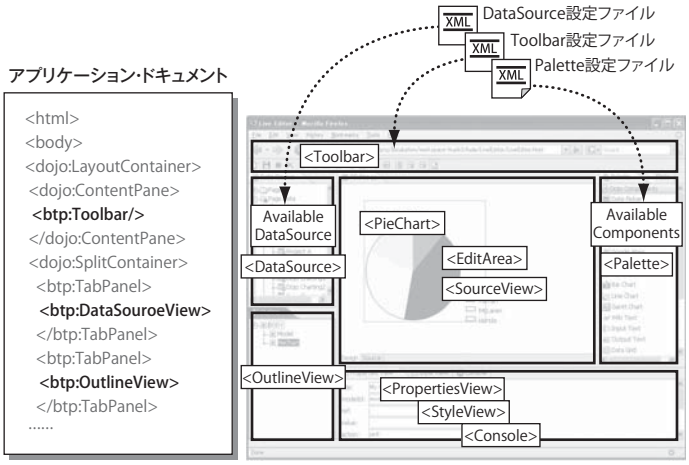

図 8．ツールの構成

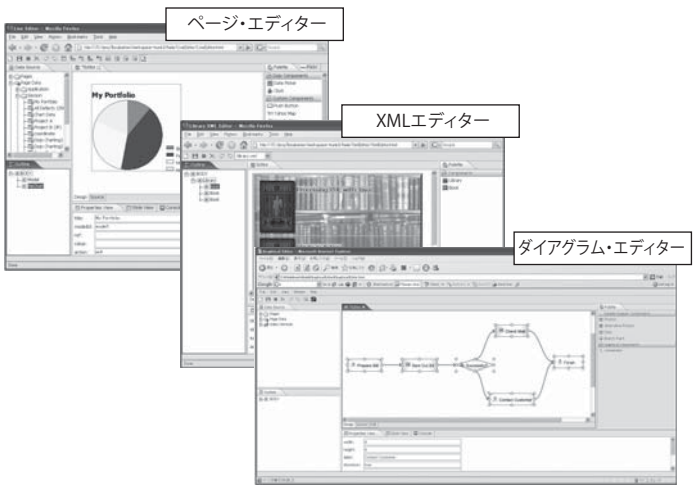

図 9．オーサリング・ツールの例

とエディター間の連携に関しても，4.1 節で述べた手法でタグの属性を用いて実現できる．疎結合なコンポーネント指向の柔軟なアセンブリー・モデルが提供されるので，ツール開発者にとってもドキュメント技術の活用は開発効率を向上させる効果がある．図 9 にプラットフォーム上に作られたツールの例を示す．

7. おわりに

本論文では，Web2.0 時代の高度でプラットフォームに中立な Web アプリケーションを効率良く作成するには，ドキュメントによるアプリケーション記述が有効であることを示した．プログラミングではなく，ドキュメントの技術・手法が使えることにより，IT の専門家ではないビジネス・ユーザーにもコンテンツ編集の要領でアプリケーションの作成ができるオーサリング・ツールの実現も可能となる．さらに，Wiki の感覚でアプリケーションを手軽に作成またはカスタマイズし，その場で直ちに実行可能な「シチュエーショナル・アプリケーション」の世界も実現可能であることを示した．

また，多様なニーズに効率よく対応するために，ブラウザー・ツール・プラットフォームを実装し，その上に複数種類のオーサリング・ツールを実装することで，さまざまなタイプの要求に対して提案手法が有効であることを確認した．

ブラウザー・ベースのアプリケーションの高度化，ライフ・サイクルの短縮化，シチュエーショナル・アプリケーション環境へのニーズの高まり，といった背景の中で，提案手法は 1 つの解になると考える．

参考文献

［1］ T. O'Reilly, What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software, O'Reilly Media, Inc., http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html
［2］ 若尾, 神山: “Web 2.0 – Webの最新動向 –,” 電子情報通信学会誌, Vol. 89, No. 12, pp. 1085-1090（2006）.
［3］ McAfee, Andrew P.: “Enterprise 2.0: The Dawn of Emergent Collaboration,” MIT Sloan. Management Review, Vol. 47, No. 3, pp. 21-28（2006）.
［4］ 吉田健一: エンタープライズ2.0 ~次世代ウェヴがもたらす企業変革~, インプレスR&D, ISBN978-4844324492（2007）.
［5］ J. Ponzo, L. D. Hasson, J. George, G. Thomas, O. Gruber, R. Konuru, A. Purakayastha, R. D. Johnson, J. Colson, and R. A. Pollak: “On Demand Web-Client Technologies,” IBM Systems Journal, Vol. 43, No. 2, pp. 297-315（2004）.
［6］ W. Sun, K. Zhang, S.-K. Chen, X. Zhang, and H. Liang: “Software as a service: an integration perspective,” in Proc. Int. Conf. Service Oriented Computing（ICSOC）, pp. 558-569（2007）.
［7］ L. Cherbakov, A. Bravery, B. D. Goodman, A. Pandya, and J. Baggett: “Changing the corporate IT development model: Tapping the power of grassroots computing,” IBM Systems Journal, Vol. 46, No. 4, pp. 743-762（2007）.
［8］ 若尾, 三浦, 神山, 田添: はじめてのJSF―Web開発を変えるJavaフレームワークのすべて, 日経BP社, ISBN4822221253（2004）.
［9］ Roy T. Fielding: Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine（2000）.
［10］ Eclipse.org, Eclipse Foundation, http://www.eclipse.org/.

日本アイ・ビー・エム株式会社
ソフトウェア開発研究所
WPLC 開発 & サービス
アドバイザリー・ソフトウェア開発エンジニア

神山 淑朗 Yoshiroh Kamiyama

［プロフィール］
1992 年，日本 IBM 入社．Web オーサリング・ツール，機械翻訳システムの開発を担当．2003 年以降，リッチ・クライアント，シチュエーショナル・アプリケーション開発ツール，Web メール・クライアントなどの研究開発に従事している．

日本アイ・ビー・エム株式会社
ソフトウェア開発研究所
AIM/Rational® 開発 & サービス
スタッフ・ソフトウェア開発エンジニア

嶋 幸太郎 Kotaro Shima

［プロフィール］
2003 年，日本 IBM 入社．ホームページ・ビルダー® の開発などを経て，エンドユーザー向けのアプリケーション開発ツールの研究開発に従事．現在は Rational ClearQuest® の Web フロント・エンドの開発に携わっている．

日本アイ・ビー・エム株式会社
ソフトウェア開発研究所
IM 開発 & サービス
スタッフ・ソフトウェア開発エンジニア

三浦 圭司 Keiji Miura

［プロフィール］
2001 年，日本 IBM 入社．ホームページ・ビルダー，Rational Application Developer の開発を担当．現在はエンタープライズ向け製品の Web アプリケーションのユーザー・インターフェース開発に従事している．