データベースシステム 129－12
（２００３．１．２４）

デジタルデータ放送コンテンツの HTML への変換

上原 康貴，疋田 輝雄

概要

デジタルデータ放送のコンテンツおよびその記述言語である BML (Broadcast Markup Language)は、デジタルテレビ、チューナーの普及が伸び悩んでいるため、広く認知されているとは言い難い現状にある。コンテンツの流通を広める一手段として、コンテンツを一般の Web ブラウザで表示することを試みる。Web ブラウザで表示するためには、BML で記述されたコンテンツを HTML で記述することが必要となる。そこで、BML から HTML に変換するツールを作成した。変換ツールでは、BML と HTML の記述の差異を緩和するとともに、入力デバイスのイベントの違いを解決するために、リモコンと同じ動作をするボタン機能を追加した。

本稿では、BML と HTML との違い、変換を実現するための手法、問題点、変換することによって得られる有効性について述べる。

Converting digital-broadcasting contents to HTML

Yasutaka Uehara, Teruo Hikita

Abstract

The contents of digital data broadcast expressed in BML (Broadcast Markup Language) are not yet fully recognized, since digital televisions and tuners are not spreading to wider audience than expected. As a way to promote more circulations of digital broadcast contents, we try to show them on the general web browsers. In order to do this, the contents in BML must be converted to HTML documents. In this paper we show our newly developed conversion tools from BML to HTML. There exist differences between BML and HTML, although BML was originally developed using HTML as its model language. One of the notable differences is in the user input device: the remote control device is used in the case of BML, while the user usually uses function buttons on the browser in the case of HTML. This paper explains in detail the differences between BML and HTML, the techniques for converting BML to HTML, and finally the validity obtained by the conversion is shown by examples.

1．はじめに

2000 年 12 月 1 日から BS デジタル放送の本放送が始まり、デジタルデータ放送サービスが各局から提供されている。「見るテレビから使うテレビへ」のキャッチフレーズのとおり、データ放送では、ニュースや天気予報、オンラインショッピング、アンケート投票などさまざまなサービスが展開されている。それらデータ放送のコンテンツは、社団法人電波産業会（ARIB）で標準化された、BML (Broadcast Markup Language)で記述されている[1]。このような高度なサービスを提供することができるデータ放送ではあるが、デジタルテレビ・チューナーの普及が伸び悩んでいる現状もあり、広く認知されているとは言い難い。

そこで本論文では、データ放送のコンテンツを、広く普及している一般の Web ブラウザで視聴、操作できるように BML ソースコードを HTML ソースコードへ自動変換することを試みる。BML と HTML の共通な構造を利用した変換手法や、スタイルシート、スクリプト、入力デバイスなど BML と HTML の記述の違いとその違いを解決する方法を説明する。また、データ放送コンテンツを Web コンテンツとして利用することの利点について述べる。

明治大学理工学研究科
School of Science and Technology,
Meiji University

2．BML の構造および HTML との差異

BML は、ヘッダー部とボディー部で構成されている（図 2.1）。ヘッダー部では、 スタイルシート、スクリプト、イベント定義の３つのスクリプト定義がある。スタイルシートは、W3C が策定している CSS（Cascading Style Sheet）が採用されており、スクリプトは、ヨーロッパの機関である ECMA(European Computer Manufacturers Association)によって規格化された ECMAScript をベースとして、機能を拡張している。ボディー部分は、XHTML1.0 をベースに構成されている[2]。

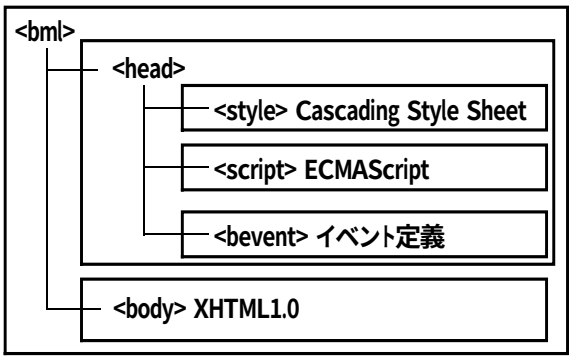

図 2.1：BML のソース構造

BML と HTML の違いとしては、主に以下のようなものがある。

A. スタイルシート

(1)位置指定の方法

top、left で表示位置を指定するプロパティーがある。BML では、親ボックスを基準とした絶対位置で指定されている。一方、HTML では絶対位置指定の他に、相対位置を指定するプロパティー値がある。

(2)色指定の方法

背景色を始め、要素の文字色やボーダーの色など、色に関する指定を行うプロパティーがある。BML では、色指定のプロパティー値として RGB 指定、インデックスカラー指定、キーワード指定、YCBCR 指定がある。一方、HTML では数値指定や色名指定、システムカラーで指定する。

B. スクリプト

(1)オブジェクトの種類

スクリプトでは、部品や情報をオブジェクトとして取り扱うことができる。このオブジェクトの値を変更したり、値を調べて各々の処理を設定することで動的な変化を与える。BML で使用する ECMAScript では、放送特有の機能を実現するために Browser 擬似オブジェクトが追加されている。HTML では、ブラウザ全体を window オブジェクトとして扱う。

(2)メソッドの違い

オブジェクトにはメソッドが定義されており、メソッドを呼び出すことでオブジェクトの動作を指定する。BML では放送用に拡張されているので、HTML では対応していないメソッドや、同じ処理を行うが名前が一致しないメソッド、引数の数が違うメソッドなどがある。

C. ボディー

(1) 位置指定の方法

要素の属性として、要素の表示位置を指定することができる。A の(1)で説明した位置指定の方法と同様の違いが、ボディー部分の各要素の属性でも起こりうる。

(2)色指定の方法

色に関する指定を要素の属性として定義することができる。色指定の方法に関しても、A の(2)で説明した色指定の方法と同じことが言える。

(3)イベントハンドラの指定

要素では、特定の動作が起こったタイミング（イベント）に応じた関数の呼び出しを属性として指定することができる。たとえば、BML では、何かボタンを押された時の処理指定を行う場合は、onkeydown というイベントハンドラを要素の属性に指定するが、HTML では onkeydown イベントハンドラを指定しても無効となり、関数の呼出しが実行されない。

D. 入力デバイス

(1) リモコン操作によって生じるイベント

テレビを媒体として使用する放送コンテンツにおいては、入力デバイスとしてリモコンが使用される。一方、パソコンを主要な媒体とする Web ブラウザでは、一般的にマウスが使用されている。このデバイスの違いにより、発生するイベントハンドラにおいても違いが生じる。たとえば、デジタルデータ放送では、ある場面でリモコンキーを押下した時、どのキーが押されたかによって処理を選択することが可能である。しかし、Web 上では、「クリックした」というイベントしか判断することができない。HTML に変換する際には、リモコンと同じような動作をするロジックが必要となる。

3. BML の記述例と変換の流れ

この節では、BML のソースコードの実例と表示例を示す。また、BML を HTML に変換するための手順を説明する。

3.1 BML の記述および表示

図 3.1 に BML の簡単な記述例を示す。

```
1  <bml>
2  <head>
3  <title>sample</title>
4  <style>
     <![CDATA[
5    body
     {background-color-index:5;}
     ]]>
6  </style>
7  <script>
8  </script>
9  </head>
10 <body style="resolution:960x540;
     display-aspect-ratio:16v9;">
```

```
11 <div
     style="left:50px;top:30px;width:860px;
     height:480px;">
12 <p style="top:48px;left:47px;width:366px;
     height:43px;border-style:solid;border-
     width:2px;">この画面は、サンプルです。
     </p>
13 <object
     style="top:134px;left:189px;width:64p;
     height:48px;" data="3061.png"
     type="image/X-arib-png"/>
14 </div>
15 </body>
16 </bml>
```

図 3.1: BML の記述例

図 3.1 の BML では、1 行目で bml の宣言をし、2 行目から 9 行目は head 要素であることを示している。4 行目から 6 行目ではスタイルシート部分であり、5 行目で、body 要素の背景色を指定するインデックスカラーを 5 と指定している。7 行目から 8 行目は、スクリプトを定義する部分である。10 行目から 15 行目は body 要素であり、実際画面に表示される部分である。11 行目では、領域の指定を行っている。12 行目は、一つの段落を表し、画面に「この画面はサンプルです。」と表示する要素で、属性として表示位置や領域の幅や高さ、ボーダーの種類、幅が指定されている。13 行目は、object として 3061.png というファイル名の画像を表示することを意味している。

図 3.1 で記述されたソースコードを BML ブラウザでそのまま表示した結果を図 3.2 に表示する。BML ブラウザは ACCESS 社の NetFront を使用した。また、図 3.1 のソースコードのうち主要な要素は変換せずに、1 行目および 16 行目の bml という記述を html に変更しただけのソースコードを Web ブラウザ（Mozilla1.1）で表示した結果を図 3.3 に示す。

図 3.2 と図 3.3 を比較すると、背景色や p 要素の表示位置に違いがあることがわかる。これは、p 要素の属性での背景色指定、表示位置指定の方法が、BML と HTML で違うからである。また object 要素で指定した画像が Web ブラウザでは表示されていないが、これは、属性の type の記述に違いがあるためである。

図 3.2：BML ブラウザの表示例

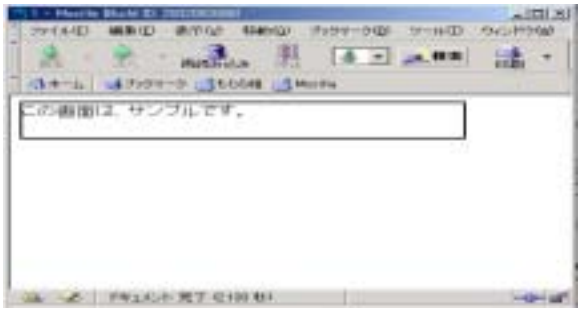

図 3.3：Web ブラウザの表示例

3.2 変換システムの構成

BML ソースコードおよび HTML ソースコードは、その構造上木構造で表現することが可能である。そこで、共通である木構造を生かして BML 木の各 Node を解析した情報を持つ“中間木”を作成する。BML 木から HTML 木に変換する間に中間木を作成することで、BML 木の構造を崩すことなく効率よく変換作業を行うことができるとともに、構造の違う HTML 木を作成する場合も想定している。図 3.4 はこの変換を実行するシステムを示す。

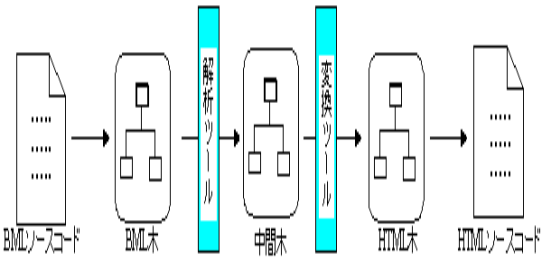

図 3.4：変換システムの構成

今システムはすべて Java で実装した。木構造をしたソースコードの扱いに優れていることと、木構造に沿ったソースコードを出力することが容易であることから、BML 木と HTML 木は Java と親和性の強いオープンソースの API である JDOM[7]を用いる。BML 木から中間木を作成する解析ツールは、BML 木の各 Node の情報の解析を担当するモジュール群である。スタイルシートを解析する StyleConverter、スクリプトを解析を担当する ScriptConverter、Body 要素以下の解析を行う BodyAnalyzer、中間木を作成する RelayTreeBuilder の 4 つのモジュールを作成した。

中間木から HTML 木の作成を担当するのは変換ツールである。変換ツールは、headElement 以下の要素の橋渡しを行う headElementManager、Body 要素の変更を行う BodyConverter、リモコン機能を作成する RemoteControlBuilder と、それらのモジュールからの情報を元に HTML 木を作成する HTMLTreeBuilder の 4 つで構成されている。BML 木から HTML 木を作るまでの一連の処理をまとめて、変換モジュールと呼ぶ。

図 3.5 は BML 木から HTML 木への構造が変わらない場合の変換例である。この場合はさらに、BML 木と

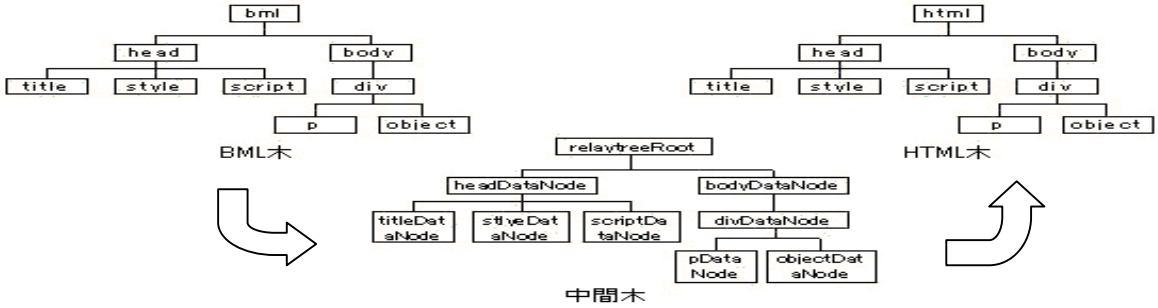

図 3.5: BML 木から HTML 木への変換例

HTML 木の各頂点の名称には変換が必要ないが、各頂点の内容は各々のブラウザに対応するような変換が加えられている。

図 3.6 に変換作業を行う変換モジュールの構成を示す。

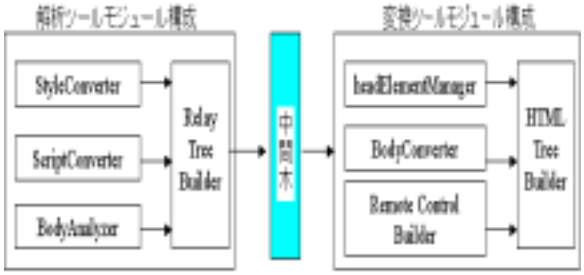

図 3.6：変換モジュールの構成

4.中間木の作成方法とコードの解析・変換

この章では、中間木の構成およびその作成方法について説明する。また、中間木作成に必要な BML ソースコードの解析・変換についても述べる。

4.1 中間木の構成

中間木は、BML 木の各 Node を深さ優先探索でアクセスし、解析したデータを元に新たな Node を中間木の対応する場所に追加していくことで BML 木と同じ構造となるようにする。

実際には、BML を DOM 木として扱い、スタイルシート部分、スクリプト部分、ボディー部分のそれぞれの部分に分けてアクセスする。スタイルシート部分、スクリプト部分にアクセスがあると、解析ツールの対応するモジュールでコードの解析と並行して変換の対象となる個所に Web ブラウザに対応するような処理を加える。ボディー部分は BodyAnalyzer モジュールで処理され、ボディー部分以下の子要素を一つの Node として扱い、アクセスされた各要素の情報を解析する。各モジュールの解析の結果を元に、RelayTreeBuilder モジュールで、解析結果を Content として保持した Node を作成し、中間木の対応する場所へ登録する。

表 4.1 に、中間木の Node 構造を示す。

保持している情報	説明
Type	要素（タグ）の種類
Content	要素が持っている属性などを格納する Bean（解析データ）
ChildrenList	要素が持っている子要素のリスト

表 4.1: 中間木の Node の構造

4.2 スタイルシート部分の解析と変換

スタイルシートとは、コンテンツのデザインを定義するものである。データ放送は、テレビで表示することを前提としているので、ボックス（タグ）の配置においては、親要素の box 左上端からの絶対位置で指定される。一方 HTML では、位置指定の方法がいくつか用意されている。スタイルシート内に位置を指定する文言があった場合には、絶対位置を指定する文言を追加する必要がある。

変換方法は、セレクタごとに分け、セレクタ内に位置指定プロパティーが宣言されていたら、position:absolute という一文を追加する。図 4.1 に変換前の BML ソースコードのスタイルシート、図 4.2 に変換後のスタイルシートを示す。

```
# positionExample
{
  top:0px;left:750px;
}
```

図 4.1:BML の位置指定スタイル

```
# positionExample
{
  position:absolute;
  top:0px;left:750px;
}
```

図 4.2：変換後の位置指定スタイル

スタイルシートでは、文字やボーダーなど色を指定することが可能である。BML ではその指定をインデックス番号を用いて記述する場合があり、16 進数で表現された RGB の組み合わせやカラーネームで指定する

が、HTML ではインデックス番号は無効となり適用されない。そこで、色指定を行っている部分に関して、インデックス番号を取得して、そのインデックス番号に対応する RGB16 進数に変更する。その際、色指定文言に対しても同時に変更する。

実際には、セレクタ内に color-index という文字列を含んでいるプロパティーを発見した場合、そのプロパティー値（カラーインデックス）を取得して、プロパティー値を引数とした変換メソッドを用いて対応する RGB16 進数に変更する。その際、プロパティー名も BML と HTML では違うので変更する必要がある。図 4.3 は背景色とボーダーの上辺の色をカラーインデックスで指定しているスタイルシートの例である。図 4.4 で図 4.3 の色指定スタイルを変換した結果を示す。下線部分が変換されている。

```
# colorexample
{
  background-color-index:3;
  border-top-color-index:43;
}
```

図 4.3：BML の色指定スタイル

```
# colorexample
{
  background-color: #ffff00;
  border-top-color: #aa5500;
}
```

図 4.4: 変換後の色指定スタイル

解析・変換処理を終了すると、スタイルシート部分を一つの文字列として扱い、Content として保持する styleNode を作成し、headNode の子要素として中間木に登録する。

4.3 スクリプト部分の解析と変換

スクリプト部分には、メソッドの名前、メソッドの引数の数、対応していないオブジェクトなど、文言の変更や追加だけでは対応しきれない処理が数多くある。そこで、まずスクリプト部分の構文解析を行い、その結果を元に再びコードを形成していく際に変換の対象となる個所に対して JavaScript に対応するコードへと変更する。構文解析を行うので、BML のスクリプト部分が正規表現で記述されていることが変換を正しく行うための前提条件となる。構文解析を行うパーサーは、フリーの ECMAScript インタプリタである FESI を利用する[5]。

変換方法としては、まず、変換の対象となるオブジェクトやメソッドを登録する変換対象リストを用意する。そして、スクリプト解析時に、スクリプトに記述されている各ファンクション内で変換対象リストに登録されているオブジェクトやメソッドを発見するとチェックを入れておく。そのファンクション内にチェックされた部分がある場合は、チェックの対象となったものに応じた変換をファンクションのコードを再構築する時に行う。

図 4.5、4.6 でスクリプト部分の変換前後のコードを示す。図 4.5 は、browser オブジェクトに定義されている setInterval メソッドを呼び出しているファンクションの例である。変換後のコードである図 4.6 では、browser オブジェクトを表す部分が無くなっていることと、引数の数が変わっていることがわかる。（setInterval メソッドは JavaScript では window オブジェクトに属しており、window オブジェクトは省略して宣言することができる。）

```
function example()
{
browser.setInterval("Handler();",500,0 );
}
```

図 4.5: ECMAScript のソースコード例

```
function example()
{
setInterval("Handler();",500);
}
```

図 4.6: 変換後の JavaScript のソースコード

放送用に拡張拡張された、ハードウェアに依存する EPG(Electronic Program Guide:電子番組ガイド)や字幕などの処理は、今回は変更の対象外としている。また、映像や音声のストリーミングに関する処理も現在のところ対応していない。

解析・変換処理を終了すると、スクリプト部分を一つの文字列として扱い、Content として保持する scriptNode を作成し、headNode の子要素として中間木に登録する。

4.4 ボディー部分の解析と変換

ボディー部分は、実際に画面に表示されるものが記述されており、数種類の要素（タグ）によって構成されている。BML で使用されている要素は、HTML でも使用可能であるが、要素の属性については HTML でサポートされていないものがある。解析の手順としては、bodyNode 以下の部分木に対して、深さ優先探索を用い各 Node にアクセスし、その Node の種類、属性、入れ子となっている要素などの解析を行う。解析によって得られたデータは、RelayTreeBuilder に渡されて ElementDataNode が新たに作成される。BML 木の Node の解析と同時に新たな Node を作成していくことで、BML 木と同じ構造をもつ中間木を作ることができる。

5．HTML 木の作成

HTML 木の作成は、BML ソースコードを解析する際に作成した中間木を元に変換ツールで行う。中間木はほとんどの場合、木として BML 木や HTML 木と同じ構造となっているため、深さ優先探索を用いて HTML 木を作成していく。ヘッダー部分のスタイルシート、スクリプトの各部分に関しては、中間木を構築中に

HTML ソースコードに対応する形へと変換が完了しているため、headElementManager を通して HTMLTreeBuilder にそのままデータを渡す。ボディー部分では、アクセスされた ElementDataNode の種類ごとに、HTML 要素として使用できるよう選別・変更を加えたプロパティーとプロパティー値を付加していく。

図 5.1 に BML ソースコードの p 要素の記述例、図 5.2 に変換後の p 要素のソースコードを示す。下線部分が変更されたところであり、各要素のポジションの指定、色指定、イベントハンドラの指定が変換されている。

```
<p
class="example" id="debug"
style="top:270px; color-index:11;"
onkeydown="keyHandler();">
Hello BML World!!
</p>
```

図 5.1：BML の p 要素ソースコード例

```
<p
class="example"  id="debug"
style="position:absolute;Top:270px;"
color :#aaaa00;
onmousedown = "keyHandler();">
Hello BML World!!
</p>
```

図 5.2：変換後の p 要素ソースコード例

6．リモコンのボタン操作によるイベントとその対応

データ放送コンテンツはテレビでの運用を想定して記述されているため、入力デバイスとしてリモコンを使用する。一方、Web ブラウザでは、主にマウスを使用してコンテンツを操作する。Web ブラウザでは、リモコンと同じイベントを発生させるデバイスが用意されていないため、スタイルシートやスクリプト、ボディー部分の変換だけでは想定していた動作を行えない場合がある。そこで、フォームの構成部品である汎用ボタン（button）を代用することでリモコンと同じ動作を実現する。

データ放送コンテンツでは、リモコンキーの押下に伴い、フォーカス状態の要素の対象となる割り込み事象が発生し、要素に指定されたファンクションを実行する。リモコンキーの判別は、ハードウェアの性質上知ることができ、KeyCode として取得できる。一方、マウスで汎用ボタンを操作する場合、イベント自体は判断することができるが、KeyCode は知ることができない。そこで、汎用ボタンのイベント定義部分に、KeyCode を引数に持つ間接的なファンクション（ファンクション選別 Function）を定義する。また、汎用ボタンを操作する際にフォーカスがボタンにあたってしまうので、どのファンクションを実行するのか判断できない。そのため、要素を指定する SelectBox を作成する。ファンクション選別 Function では、KeyCode を保存し、SelectBox で選択された要素に定義されていたファンクションを呼び出す。ファンクション選別 Function を経由して呼び出されたファンクションは、保存された KeyCode を取得するための変更が必要となってくる。そこで、スクリプト部分に関する解析と変換を行う部分（4.3 節）で、押下されたリモコンキーの KeyCode を取得するコード（document.currentEvent.keyCode）がファンクション内に出現したら、そのコード部分をファンクション選別 Function で保存された KeyCode を取得するためのメソッドに変換することで対応する。

データ放送で使用するリモコンキーは、BML ソースコードの body 要素の used-key-list 特性で指定された<key-group>で判断することができる。そこで指定されたリモコンキーに対応する汎用ボタンを、作成した HTML ソースコードの body 要素の下部に作成する。これらのリモコンキーに関する処理は、変換ツールの RemoteControlBuilder で行う。

図 6.1 は、リモコンキーとフォーカス選択を表示した画面の例である（BML ソースコード内の body 要素の属性で used-key-list:basic が指定された場合）。

図 6.1：リモコン機能ボタンイメージ

7．実行例

ここでは、これまで説明してきた変換方法を実現したツールの検証を行う。文字や、画像などを表示する静的なコンテンツだけでなく、動的なコンテンツを使用して検証する。検証用データとして、データ放送コンテンツを制作しているデジタル･ラボラトリー[4]から提供していただいた「ブロック崩し」のサンプルコードを用いる。ブロック崩しとは、バーを移動することでボールの動きを制御し、ブロックにボールを当てるゲームである。図 7.1 は、ブロック崩しの BML ソースコードを表示した画面イメージである。図 7.2 は、BML ソースコードを HTML ソースコードに変換し、それを Web ブラウザで表示した結果である。

BML ブラウザでは、リモコンで操作されるバーを、Web ブラウザでは、画面下部に表示された汎用ボタンで操作することができる。

3.1 節で発生していた表示位置の違い、色指定、画像が表示されないといった事象が、図 7.1、7.2 を比較すると解決されていることがわかる。

図 7.1： BML ブラウザで表示したコンテンツ

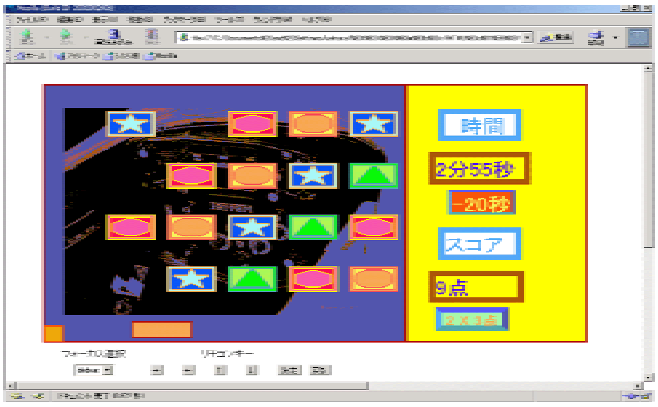

図 7.2： Web ブラウザで表示した変換後のコンテンツ

8．おわりに

8.1 HTML 変換における有効性

(1) **情報の有効活用**

デジタルデータ放送はテレビ放送の一サービスとして運用されているものだが、テレビから得られる情報の利点として、リアルタイム性と信頼性が上げられる。テレビで放送された情報を Web コンテンツの情報としていち早く公開することができるならば、Web コンテンツにおいてもリアルタイム性・信頼性が得られる。

(2) **コンテンツの流通**

現在のデータ放送会社の Web サイトにおけるデータ放送コンテンツの紹介は、コンテンツの静的な画像と番組の紹介文にとどまっていることが多く見受けられる。放送を視聴することのできないユーザーに対して、実際の放送と同じような動作を Web 上で体験してもらうことでデータ放送のコンテンツを広めることができる。

(3) **新規性のある放送コンテンツの展開**

データ放送と Web を連動させることにより、新たなコンセプトに基いた番組を制作することが可能となる。

8.2 今後の対応

デジタルデータ放送コンテンツを HTML 形式に変換するツールを作成した。このツールでは、BML と HTML における内部的な違い、テレビとパソコンにおけるハードウェア的な違いを緩和することを実現した。それにより、BML を知らない Web 管理者でも BML で記述された情報を扱うことが可能となった。

今後は、機能の追加・改善を進め、デジタルデータ放送コンテンツと Web コンテンツの差異をさらに無くしていくとともに、テレビと同じようなポインティングデバイスを持つ携帯電話に対応できる変換を行っていく。

参考文献

[1] デジタル放送におけるデータ放送符号化方式と伝送方式（ARIB STD-B24 3.0 版），社団法人　電波産業会，2001 年 5 月

[2] デジタル放送ガイドブック 2002 BML のすべて，日経 BP 社，2001 年

[3] 半場 万人: 詳解 JavaScript 辞典，株式会社秀和システム，2002 年 7 月

[4] 株式会社　デジタル・ラボラトリー http://www.digilab.co.jp/

[5] The FESI(pronounced like fuzzy) home page http://home.worldcom.ch/~jmlugrin/fesi/

[6] HTML クイックリファレンス http://www.htmq.com/index.htm

[7] JDOM http://www.jdom.org/index.html