ビスケットランドの実装

原田康徳
NTT CS研

いわゆるアニメーション版の Wiki ともいうべき ViscuitLand の実装について述べる.

1. はじめに

子どもや情報科学を専門としない大人に対して, プログラミングの楽しさを教えるために Viscuit(ビスケット)は開発された. それを用いた様々な活動からビスケットとネットワークとのシームレスな融合が重要視された. そこでネットワークの新しいメタファとして, 無限に広い 2 次元空間を用意し, そこに自由にお絵かきをするビスケットランドを提案した. ビスケットランドはこれまでに3回バージョンが更新され, その都度, まったく新しい要求が取り込まれ, 逆に中核の仕様を縮小し, より現実的な実装に変わってきた.

本稿はそれらの変遷を整理すると共に, 現状の問題点と, 現在実装中のシステムの詳細について述べる.

2. 本来のビスケットランドの目的

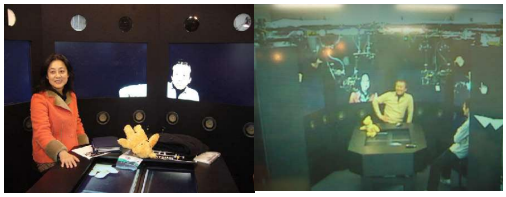
ビスケットランドの本来の目的を整理する. ビスケットでは子ども達が絵本を作ったり, あらかじめ作られた教材でプログラムを学んだり, 用意されたルールセットを使って, そのワールドを楽しんだり, 様々な使い方がある. それらに共通している問題は, ファイルのやり取りと引用である.

従来のファイルのやり取りは,「名前をつけて保存」「ファイルを開く」「メールに添付」「Web にアップロード」「ダウンロード」といったように, すべてファイル名をベースに行われてきた. しかしそのファイルに含まれている絵やルールを引用したり, さらに複数のファイルを引用した場合, ファイル名を用いると名前の衝突など, 本質的な問題が多く含まれている. また, ネットワークが必須の時代に, 新しいアーキテクチャが求められている. ひとつは, URL のようにホスト名とファイル名を組み合わせた名前を用いることである. この方法は, 簡単で分かりやすい実装が可能である反面, クライアントの PC は必ずしも固定されたホスト名がつけられているわけではないし, 負荷分散などの視点からホストを固定することの利点があまりないという問題がある.

ここで必要なことは, ネットワークワイドで一意に定めることができる名前付けの方法である. これは, ビスケットの世界観のレベルとの整合性(数学的に厳密というよりは, 直感的に理解しやすいという点)を重視したい. 必要な点は, 将来に負荷分散が容易に実現できるような, 技術的, 管理的な視点があればよいだけである.

そこでビスケットランドでは, ネットワークを無限に広い 2 次元空間として見せることにした. 名前空間が 2 次元だということである. 名前が無限であればよいというだけならば, 自然数で十分であるが, 空間というのを直感的に意識できるようにするために, 2 次元とした.

3. t-Room のミドルウェア



t-Room は同室感（離れた部屋が空間的に一致している感覚）を得るための次世代テレビ会議システムである（写真）. t-Room はモノリスと呼ばれるモジュールから構成される. 各モノリスは, ディスプレイとそのディスプレイの面を撮影するカメラ, 及びそれらを制御する PC から構成されている. 基本的な動作は, カメラで撮影された映像が遠隔地に送られ, 対応するモノリスのディスプレイに表示される. カメラとディスプレイの関係が2地点でループになるので, それをキャンセルするために偏光フィルタを用いている. モノリスに囲まれた部屋をつくり, 人がディスプレイに接した位置に立つと, あたかも遠隔地と部屋が重なっているように感じられる.

t-Room では撮影系と表示系とで同じ面を用いていることが特徴である. その結果, 撮影されたビデオを蓄積し, 後でそれを再生することで, その部屋で起きた出来事すべてを記録し再生することが可能となる（普通のビデオカメラでは記録はできるが, 再生が別のスクリーンの中になってしまう）. すなわち, 遠方の部屋と同じ空間を占めるだけでなく, 過去の記録とも同じ空間を占めることができるのである.

いま, 2地点 A, B をつなぐ t-Room を考えると, 現在の再生だけだと, A のディスプレイに B の映像を, B のディスプレイに A の映像を表示すればよいだけである. さらに過去の映像を再生する場合には, A のディスプレイには A の過去, B の過去, B の現在の映像を重ねて表示しなければならない. 過去を再生している状況も録画の対象となるため, それを見るときは, 過去1の映像, 過去1を見ている過去2の映像, 現在の映像を重ねる必要がある. すなわち映像を何重にも重ねて表示する必要がある.

また t-Room は原理的には何地点でも部屋を重ねることができる. たとえば3地点 A,B,C をつなぐ場合は, A のディスプレイには B と C の映像を重ねて表示する.

t-Room を単なるビデオ技術として実装することが考えられる. 実際に t-Room の前の2つのバージョンはビデオ合成装置を組み合わせて実装していた. 合成装置は一般に2入力1出力のものであり, 構成が複雑になるにしたがって合成装置を複数台組み合わせないと実現できない. 近年のPCの高速化によりそれらをソフトウェアで実装することを考える.

t-Room の運用を重ねた結果, カメラやディスプレイの配置の校正の容易さや, 構成の自由度の高さが求められてきた. 校正の容易さとは, ディスプレイとカメラの画角を完全に一致させるよりも, ディスプレイよりも少し大きめの範囲で少し下側をカメラで撮影した方がよい（実際の人物はディスプレイより離れて立っており, カメラは上から撮影されるため, そうしないと人物は大きく下にずれて表示されてしまう）, といった調整を行うためである. また, 構成の自由度の高さとは, たとえば, 機材の調達の制約から, 2 箇所でディスプレイのサイズを揃えられないといっ

た場合でも，重ねあわせができるようにすることである．

また，t-Room に表示されるのはビデオ映像だけではなく，コンピュータ画面を表示させ双方で共有したり，背景にパノラマ写真を用いて，部屋ごと一緒に移動したかのような錯覚を与える，といった応用もある．それらを簡単に実現したい．

以上，長々と述べてきたが，これらを整理すると次のようになる．

1．複数のビデオ，VNC，写真などをレイヤを意識して，位置，縮尺，回転の変形をほどこして重ねて表示できる仕組み．

2．それらを複数の PC で実装し，一斉に制御できる仕組み．

それを実現するために，仮想的な平面を考えて，そこに様々なソース（ビデオ，VNC（コンピュータ画面の転送），写真）を貼り付け，またディスプレイはその平面の任意の領域を表示する，というシステムで実装する．ディスプレイが表示する領域は，複数の位置を指定できて，それらをレイヤとして重ねて表示をつくる．

これをビスケットランドによって実装する．ビスケットランドにビデオ通信，VNC 通信のプロトコルをプラグインとして実装する．それらは HTTP における Web カメラと同様なアーキテクチャとなる．違いは URL の代わりに 2 次元座標が使われている点，ブラウザが複数の位置を半透明で重ねて表示する点，縮尺回転をさせてブラウザに表示させることができる点である．外部からの制御はブラウザに対してどの位置をどのように見るかという指令を送るだけである．

4. 巨大なキャンバスでのビスケットの共作

ビスケットをより楽しいイベントで使えるようにするために，複数のプロジェクタを配置して巨大なキャンバスを作る．イベントの参加者は自分の作った絵と動きを「みんなに見せる」コマンドを用いて，そのキャンバスに送ることができる．

設置の容易さから，プロジェクタにはPCが1台ずつ対応させ，サーバからの描画命令によって一斉にコントロールする．ビスケットは動作原理には乱数を用いていないので，アニメーションを開始する時刻を揃えて，同じ初期値から計算をすれば，すべてのスクリーンで同じアニメーションが得られるはずである．したがって，なんらかの同期のメカニズムを用意すれば，各PCはビスケットランドのブラウザとして実現可能となる．

5. 実装の経緯

以上の要求をふまえて，ビスケットランドはこれまでに3つのバージョンを開発してきた．その都度，新しい要求が取り込まれ，当初予想していた過剰な仕様が削られて今のバージョンに至っている．

1. バージョン1

   最初のバージョンはとにかく動作させることだけが目標であった．サーバとクライアントの区別をせずに，P2Pで情報をやり取りするといった試みがあったが，完全には動作しなかった．基本図形（枠，文字列，参照，ビットマップ）の組み合わせで，すべてのデータを表現する方針で，その基本図形のための簡単なエディタが用意されていた．昨年のプログラミングシンポジウムでのプレゼンに使われた．

   領域は固定サイズのセグメントに分割され，その単位で通信が行われた．しかし，基本図形は座標でアクセスされるために，ひとつのセグメントには複数の図形が存在していた．

2. バージョン2

   t-Roomのミドルウェアとしての開発．ビスケットの機能よりもt-Roomのデモの安定性に重点が置かれた．セグメントにまたがる大きな図形を扱えるようにするために，セグメントサイズを不定長にした．その結果，セグメントの配置にきわめて複雑な管理が必要になった．たとえば，一度配置したセグメントを連結するといった処理を，P2Pを通じて伝播させなければならなかった．これもP2Pとしては動作させるに至らず，1台のサーバと複数のクライアントという動作であった．

   セグメントには，リアルタイム属性が追加され，そこで生じた変更は，興味があるクライアントに即座に伝えられるという仕組みが実装された．これによって，ビスケットのアニメーションを複数のクライアントで共有できるようになり，プロジェクタによる画面の共有が実現できた．

   エディタは用意されず，テキストのコマンドによるオブジェクト配置と，前のビスケットのデータファイルを直接読み込む機能があった．T-Roomのデモはrubyによってオブジェクトの配置情報をテキストコマンドに変換して，それを読み込ませることで実現された．

   共有画面でのイベントでは，ユーザのPCではこれまでのビスケットが動作しており，「みせる」ボタンを押すことで，ファイルを保存して，ビスケットランドのクライアントを

起動する. クライアントは保存されたファイルを読み込み, 適切な位置にそのデータを配置する. それは自動的にサーバに送られ, アニメーションの動きを変更する.

3. バージョン 3
   これが現在開発中のものである. 上記2つのバージョンと以前のビスケットは C++ と Qt によって実装されていた. それに対しこのバージョンは Web との親和性を重視するためにクライアントを Java で実装している. サーバは現在 Web サーバと簡単な CGI によって実現される. 複数のサーバ間のやり取りは別のしかけで行う. アプレットとダウンロードの両方で動作し, ダウンロード版では負荷分散により複数のサーバを渡り歩く仕掛けも実装する.

次の節では, このバージョン 3 の詳細について述べる.

6. 現在の実装

データ構造を基本図形の組み合わせで表現することをやめ, XML とバイナリデータで表現する. セグメントも廃止する. そのかわり, すべてのデータは位置だけじゃなくて, 領域の情報も持っている. バイナリデータである画像ファイルは, その画像のサイズ分の領域を占める. XML はそれを表示するために必要な大きさの領域を占める. それぞれのデータは, 互いに重なることを禁止している. サーバ, クライアントとのプロトコルは概念的には LIST, GET, PUT の3つの操作で構成される.

- LIST (x,y,w,h):領域
  領域と重なっているデータの領域, データタイプなどの情報をリストで返す.
- GET (x,y):位置
  その位置を含んでいるデータを返す.
- PUT (x,y,w,h):領域 データ
  その領域にデータを配置する. すでにこの領域と重なるデータが存在する場合には, それらを削除してから, 配置する. 将来アカウントの情報が追加された場合には, 削除されるデータに削除権があるか調べられ, なければこの操作は失敗する.

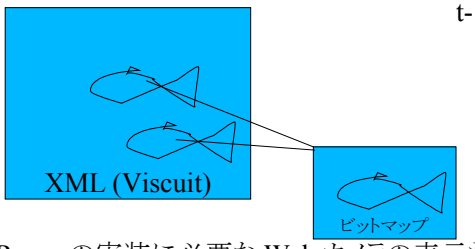
実際にはキャッシュの効果が利用できるようにタイムスタンプなどの情報も交換される. 現在はこれを HTTP+CGI によって実装している. しかし, 画面を 1 ピクセルスクロールするだけで LIST 操作が呼び出されるので効率が悪い. そこで実際には, これをキャッシュ化し, LIST, GET は HTTP の固定ファイルのアクセスとして実装され, PUT だけが CGI を呼び出すようにしている.

クライアントでは, 画面に表示する領域が決まると, LIST, GET によりその領域と重なっているデータを獲得し, その位置に表示する. XML データの場合はそれを解釈し表示する. その中には, 他のデータへの参照(x,y)が含まれる場合がある. その場合は, そのデータを GET して, それをその位置に表示する. この埋め込みは再帰的に行われる.

ビスケットの領域は XML によって表現される. ビスケット中のオブジェクト(うごかす対象としての絵)は, 他の領域の参照として実装されている. あるビットマップを参照することで, そのビットマップをこの位置に配置したことになる. オブジェクトは書き換え対象とルールの中に存在する. 書き換えは参照先を考慮せずに参照先が同じかどうかだけで行う. 参照先が別のビスケットの場合は, そこで生

成されたアニメーションをここに表示する．これは動きを部品化して作成することを可能にしている．たとえば，口をパクパクさせる動きだけをあるビスケットで表現し，それを参照することで，常にパクパクしているオブジェクトを作る．これを使って，泳がせるアニメーションを作ると，泳ぎながらパクパクするアニメーションができあがる．

t-



Room の実装に必要な Web カメラの表示も XML によって表現される．この中には接続先のカメラのアドレスなどの情報が含まれる．この XML を表示しようとすると，Web カメラに接続し，その画像情報を獲得する．

クライアントには，簡単なペイントソフトの機能が用意されている．また，あらたに XML の種類に対応するための表示編集用のプラグインによる拡張機能も用意する．

クライアントは全画面表示にして，外部から表示している位置を制御させることができるようになっている．

別のクライアントとしては，ペイントでお絵かきをする領域，その絵に動きをつける領域，動きのついた絵を配置する領域の3つで構成される．これはイベントなどでのユーザが混乱無くシステムを触れるようにするためのカスタマイズといえる．

一般のユーザのための用途・環境に応じたカスタマイズも考えられる．

これらのカスタマイズの情報（いわゆるコンフィグファイル）もやはり XML で表現され，ある領域に配置される．クライアントは起動時にどのコンフィグファイルを使うかを位置によって指定される．また実行中に使用する配置情報を動的に変更することができる．これは主に画面が小さいクライアント用で，それぞれの作業フェーズに応じた最適な画面配置を提供する．また，スキンという見た目の情報もこのコンフィグファイルに記入されている．したがって，スキンを選んだり，よいコンフィグファイルを作成しそれを流通させるという行為もすべてビスケットランド内で閉じている．

7．考察

URL をオブジェクト ID として実装されているような分散オブジェクト指向システムとして考えると，HTML の中にフレームや画像として，そのオブジェクトの URL を埋め込む場合を考える．HTML では他のサイトの HTML や画像，動画像を埋め込むことができる．ただし，回転縮小などの効果もなく，重ねて配置することもできない．ビスケットランドでは，ここに素直に座標の概念を導入し，回転縮小の効果が任意のデータに対して適用できるようにした．また，すべての処理でアルファ値が考慮されているので，半透明に重ねることも可能である．さらに進んで，アニメーションの部品としても他のオブジェクト（HTML や他の動画像）を参照しそれを動かすことができている．

現在は動きを作る方法はビスケットだけであるが，他の指定の仕方（たとえば手続き言語による動き，パラパラ漫画，アルゴリズム的な動きなど）も考えられる．それらは XML で表現され，テキストにより直接入力するだけで

なく，ビジュアル言語的なインタフェースで図的に入力するような拡張も可能でる．これらは，プラグインとして拡張する．このプラグインはビスケットランド上で配布され，文字による名前ではなくて，2 次元空間の座標によって指定される．

他のオブジェクトの引用の機構があるので，これにより視覚的な演算を定義することができる．具体的には，画像のクリッピング，色の変換といったフィルタ，時間遅れを伴うもの，2つ以上のオブジェクトの演算などである．この特殊例としてビスケットによるアニメーションがある．ワールドワイドな画像計算と呼ぶべきものになるだろう．

ビスケットランドは 2 次元の無限に広い空間である．しかし，オブジェクト ID としてみた場合は 1 次元の 1 方向の無限（自然数）だけあれば十分である．2 次元の使い方をもう少し限定することができる．

まず空間を4つの象限にわけて考える．第一象限は右と上に無限に広がっている．どちらを使ってもよいが，ここでは右に伸びるようなデータを置くことにする．右に伸びるというのは，たとえば絵本などである．左端にタイトルがあり，それが右に続いてゆく．典型的なビスケット絵本では，背景や動く部品などの絵と，それを組み合わせて作る動き，そしてそれらを配置する最終的なページの 3 層に分かれているので，新しく絵本を作るときは，絵本の紙の高さを決め，その 3 倍の高さの帯状の領域を第一象限に確保する．この場合絵本の最大ページは決められないので，右方向は無限大である．こうして横長の領域が上に積み上げられてゆく．同様に第二象限は上，第三象限は左，第四象限は下に伸びる方向を固定する．下に伸びる日記のようなコンテンツは第四象限に置かれる．

伸びる方向のほかに，伸びる速さも制約することができる．たとえば日記のような時間的なコンテンツは，日付時刻という絶対的な尺度がある．それを他のコンテンツと揃えるという使い方もある．日記は下に伸びるので，1 日に伸びるピクセル数というのを固定するのである．1 日に書く分量が多い人は，確保する横幅を多めにすればよい．

t-Room では部屋の録画再生を行う．過去の映像を再生する，ということは時刻を使って呼び出しているということである．そこで，あるカメラを 1 本の帯としてビスケットランド上に配置する．ここでは下に伸びることにしておこう．軸の進む速度はミリ秒を 1 ピクセルとする．0を 1970/1/1 としてもよいだろう．つまりあるカメラが撮影したある時刻の映像は 2 次元空間の 1 点として表現されるのである．新たに矩形の領域の参照を導入する．これにより時間軸の編集を空間軸の編集に変換することができる．上述のフィルタの拡張として時間の変形を伴うフィルタを作ることができる．たとえば，過去の映像を 2 倍速で再生するには，過去から現在を縦のスケールを半分に変形した矩形の参照によって作られる．ビデオの早送りボタンを押すという操作により，その参照が生成され，直ちに過去の動画が再生される．

過去を再生したという行為を矩形参照の生成によって実装するため，これは後の時刻からさらに参照することができる．A のビデオをみた B の様子を撮影したビデオを見た C の様子を撮影したビデオ．．．という過去の再生の入れ子構造が素直に表現できる．

ビスケットランドのもうひとつの特徴であった逆参照の管理に関しては，この実装で難しくなった．XML のドキュメントのなかに参照が埋め込まれてしまったからである．そこで参照のフォーマットを統一することにする．たとえば，

pointer=”land:250x503”

という URI 風の書式を用意することである．XML データを PUT により配置する際に，そのデータの文字列をスキャンして，land:で始まるものをすべて集める．これがこのデータから外部への参照であるから，それぞれの参照先に対してこのデータへの逆参照を挿入する．またデータを削除する場合は参照先から逆参照を削除する．この操作をサーバが保証することで，逆参照をアルゴリズムとして安心して使用することができる．

無限な 2 次元空間によるモデル化は原理的には問題ないのであるが，原点という特殊な点の導入で，使用上のいやらしさが残っている．原点に近いほうが価値の高い土地という印象が生まれてしまう．また，用途も4つの象限という大雑把な分け方しかできていないために，他の問題もある．たとえば，大人向けの日記と子供の日記が隣り合わないようにしたいといった問題である．これを根本的に解決するには，もうひとつの軸を導入するしかない．原点に名前をつけて，その原点からの相対位置として表現するのである．これの導入の最大の欠点は，ビスケットランドのもつ一枚の平面という世界観が崩れてしまうことである．名前がついた紙が複数ある，というのでは，従来のウェブサーバの分かりにくさと変わらなくなってしまう．そこで，実装上は原点名とその相対により位置を表現するが，それらを見る瞬間には，各原点間の配置を固定する．つまりある時点ではすべてのデータは 2 次元空間の絶対座標として表現できる．しかし，どこかのコンテンツの伸びが突出して，別の原点からの領域と衝突しそうになった場合に，その原点の配置を変更する．惑星間の距離が伸びてゆく，という感じである．これによって，1 枚の紙という世界観を崩すことなく，複数の種類のコンテンツを混乱なしに配置することができるようになる．

書式としては

“land://origin/XXXxYYYY”

または

“land:XXXxYYY”

である．後者は同じ原点を用いた場合の略記法である．この記法では origin にホスト名を用いることで，サーバを立ち上げて勝手に原点を追加することができるような含みを残している．

8．まとめ

ビスケットランドの実装について，要求事項，経緯とともに紹介した．具体的な使用を踏まえながら現実的な実装を行うのは困難であるが，かなり問題が整理されてきたと考える．

参考文献

[1] 原田康徳，加藤美由紀，Richard Potter: Viscuit: 柔軟な動作をするビジュアル言語，WISS 2003.

[2] Y. Harada, R.Potter: Fuzzy Rewriting – Soft Program Semantics for Children -, HCC 2003, IEEE.

[3] 原田康徳: ViscuitLand 地球より広い一枚の紙でお絵かき，第46回プログラミングシンポジウム．