データベースシステム 110－9
（1996. 10. 24）

データベース属性とGUIオブジェクトとの結合手法

竹川 弘志，飯沢 篤志，中山 秀明，白田 由香利
{takegawa, izw, nakayama, shirota} @ src.ricoh.co.jp
株式会社リコー　システムソリューション事業本部　ソフトウェア研究所
〒112 東京都文京区小石川 1-1-17

マルチメディア・データベースのアプリケーションの作成において、プログラミングコストのかかる部分として、データベース属性とGUI上のオブジェクトとの結合がある。今日普及しているGUIプログラミングスタイルでは、GUIに関連するプログラム動作が多数のGUIオブジェクトのコールバック手続きに細分されてしまい、かつ、それらのコールバックが複雑に関係しあっているため、プログラム全体の構造の見通しを悪くしてしまうという問題がある。マルチメディア・データベースアプリケーションの場合、データベース属性とGUIオブジェクトとの間の複雑なデータ変換が必要となるので、この問題はさらに顕著になる。この問題を解決するため、我々はデータベース属性とGUIオブジェクトとの関連部分に着目し、その結合部分を記述するスタイルを現在のイベントドリブン方式から、非手続き的に宣言する手法 “transvalue” を考案した。全てのGUIから発生するイベントの殆んどはこの結合部分に関連するものなので、この部分を簡潔に書くことができれば全体の開発効率も向上する。

Binding Methodology of Database Attributes to GUI Objects

Hiroshi Takegawa, Atsushi Iizawa, Hideaki Nakayama, and Yukari Shirota
{takegawa, izw, nakayama, shirota} @ src.ricoh.co.jp
Software Research Center, System Solution Business Group, RICOH Company, Ltd.
1-1-17 Koishikawa, Bunkyo-ku, Tokyo, 112 Japan

When we develop a multimedia database application, programming to bind database attributes to GUI objects is a major bottleneck which involves time-consuming and error-prone work. In the GUI programming style of today, a large spaghetti program is hard to avoid, because program actions related to GUI objects are fragmented into call-back procedures of many different GUI objects and these call-back procedures are related with each other in a complicated manner. The call-backs get tangled in a worse way in multimedia database applications, because much more data conversions are required between the database attributes and the GUI objects. To solve the problem, we propose in this paper a binding method of database attributes to GUI objects in a declarative style, not in an event-driven style. We call the method “transvalue.” Since almost all events occurring on the GUI are related to the binding part, by simplifying the binding part using “transvalue” method, we can reduce the total programming cost.

1 まえがき

マルチメディア・データベースのアプリケーションの作成において、プログラミングコストのかかる部分として、データベース属性とGUI上のオブジェクトとの結合がある。我々のグループはKasuga Scriptというマルチメディアデータベースアプリケーション開発用4GL[Shi96]を開発したが、その言語仕様設計の際、「データベース属性とGUIオブジェクトの結合を簡単に記述できること」及び「シームレスにデータベース属性とGUIウイジェット間のデータの流れを表現できること」を目標とした。この手間のかかる部分を簡単に記述できれば全体のプログラム効率が向上する、と考えたのである。

我々はKasuga Scriptにおいて、データベース属性とGUIオブジェクトとの間の結合を現在のプログラミングスタイルのようにイベントドリブン方式で記述するのではなく、静的に宣言する手法 "transvalue" を提案した。transvalueは両者の結合を定義するだけでなく、両者間のデータの変換手続きも宣言的に定義できることを特長とする。本論文では、transvalueの仕様及び内部アーキテクチャについて論ずる。以下、2章では、データベースアプリケーション作成における属性とGUIオブジェクトの結合のもつ問題点を論じる。第3章ではtransvalueのデータ変換モデルを、第4章では対応付けの方法を述べる。第5章はtransvalueの例を示す。第6章では、実装のための内部機構を説明する。

2 データベース応用作成における問題点

今日普及しているGUIプログラミングスタイルでは、ボタン、メニュー、テキストフィールド、ラジオボタンなどのウイジェット[1]と呼ばれるGUIオブジェクトにコールバックと呼ばれる手続きをアタッチすることによりGUIの動作を予め定義する。実行時には、操作者がこれらのウイジェットに、ボタンを押す、などの操作を行うことによりイベントが発生する。それによりアプリケーションは操作者の操作を知り、GUIシステムが予めそのイベントに設定されていたコールバックを起動する。この方式の問題は、プログラム全体の構造の見通しを悪くしてしまうことである。理由は、実務におけるデータベースアプリケーションのGUIには、数百から数千のウイジェットがあり、GUIに関連するプログラム動作がそれらのウイジェットのコールバック手続きに細分されてしまい、かつ、それらのコールバックが複雑に関係しあっているためである。

この問題を解決するため、我々はデータベース属性とGUIオブジェクトとの間の結合を現在のGUIプログラミングスタイルのようにイベントドリブン方式で記述するのではなく、静的に宣言する手法 "transvalue" を提案した。

GUI付きマルチメディア・データベースアプリケーションの開発におけるもうひとつの問題として、「データベースアプリケーションにおけるギャップ問題」がある。我々がSQLでプログラムを書くとき、しばしば埋め込みSQLを使うが、そこではSQLコードとCやFortranのような親言語部の間に形式的なギャップがある。これをSQLをインピーダンスミスマッチ問題という[牧之内95]。加えて応用プログラムのコントロールフロー部とGUIライブラリの間には別のギャップがある。応用プログラムとGUIではデータ構造が違うため、その間でデータ変換が必要となってしまうのである。このデータ変換を多数のコールバックから何度も呼び出すようにプログラムを書くことも、プログラマにとっては大きな負担となる[西尾96]。

これら2つのギャップを解消すること、つまり、データベース属性とGUIウイジェット間をシームレスに結合することを目標として、我々はKasuga Scriptにtransvalueという機構を備え、シームレスな言語仕様を目指した。

3 transvalueのデータ変換モデル

Kasuga Scriptでは、データベース属性とウイジェットを結合する変数として、tag変数というものを宣言して使う。このtagはプログラム中で通常使われる変数としての機能ももつ。データベース属性とtagの間でのデータ変換は、transvalueという機構によって予め定義される。同様にウイジェットとtagの間のデータ変換もtransvalueによって予め定義される。図1にtagを取り巻くデータ変換の様子を示した。transvalueは値の変換ルールを記述するものであり、データベースの属性値を変換してtagに対応づける場合と、tagの値を画面に表示する場合のデータ[2]の生成に使われる。transvalueの変換ルールはtagに対して設定されるものであり、transvalueの設定は、tagというオブジェクトに対するメソッドの一種である。transvalueとは、tagまわりのデータ変換機構を指すと同時に、transvalueの設定メソッド名としても使う。transvalueルールは、データベース属性とtagの間、あるいは、ウイジェットと

[1] X ツールキット [McC88] では、画面上に配置するボタン、テキストフィールド、メニュー、スクロールバー付きウインドウなどのGUIオブジェクトのことをウイジェット (widget) と呼ぶ。

[2] 最も多いデータ型は文字列であるが、マルチメディア・データベースの場合、イメージ、動画なども対象となる。

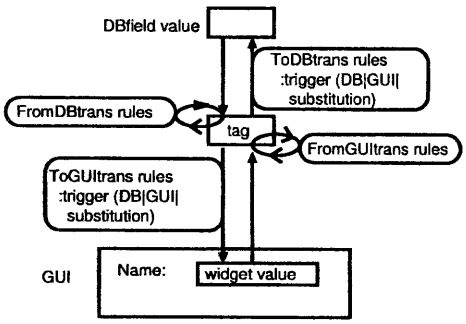

図 1: データベース属性、tag、ウイジェットの間の transvalue 変換ルール。

tag の間に設定される。多くの場合、一つのデータベース属性に対して、一つの tag というようにその関係は1対1であるが、多対多の関係も許す[3]。

我々は transvalue の設計にあたり、tag を取り巻くデータ変換ルールを、以下のように分類し、データ変換のモデルを作った。

- 変換の起こる場所とその方向性:
  FromDB 変換ルール, FromGUI 変換ルール, ToDB 変換ルール, ToGUI 変換ルール.
- ToDB 及び ToGUI の場合、トリガの発生元:
  DB 側、GUI 側、tag への値代入

FromDB 変換とは、データベース側の何らかのトリガにより tag 値が変更された場合[4]、引き続いて起こる tag 値の変換手続きである。ユーザの視点からは、FromDB 変換前の状態の tag 値は見えない。FromGUI 変換も、同様に定義できる。

From (DB | GUI) 変換に引き続いて、To (DB | GUI) 変換を起動させたい場合がある。例えば、FromDB 変換の後、その tag 値を GUI 側と DB 側へ反映させたい場合である。このように変更後の tag 値を To (DB | GUI) 変換する場合、元々のトリガの発生場所に依りその変換の意味が異なってくる。我々は、トリガの発生元として、データベース側、GUI 側、及び tag への直接の値代入の 3 種類を定義した。

図 1にあるように、FromDB 及び FromGUI 変換が tag 値に変更を施すのに対し、ToDB 及び ToGUI 変換は tag 値には変更を及ぼさず、データベース属性値あるいはウイジェットとの間の値の変換のみ行っている。ユーザの視点から見た場合、トリガ発生から tag 値の変更、From 変換、To 変換までは一連の状態遷移として起こり、途中の状態を見ることはできない。上記分類により、同じ tag〜ウイジェット間に対しても、From 変換が 1 個、To 変換が 3 個、合計で 4 個の異なる変換ルールが設定できる。tag〜データベース属性間に対しても同様である。transvalue の設計の際、このモデルになるまで多くの試行錯誤があった。多数のケースを分析することにより、このモデルであれば各種の応用における tag まわりのデータ変換が記述可能であると判断した。メソッド transvalue の文法は付録に記す。

transvalue の変換ルール中には、データ更新操作やウイジェット値更新なども記述できるので、それらの更新により発生するイベント "valueChange" がさらに他の transvalue 変換ルールのトリガとなり、transvalue のデータ変換が伝搬していくこともある。一般に、伝搬関係中に同じ transvalue 変換ルールに到るループが存在するとプログラムが停止しなくなるので、ループが無いことのチェック機能が必要となる。ユーザインタフェース内のこのような伝搬関係の解析は [Shi91] に詳しい。伝搬ループの存在をチェックするには、[Shi91] にあるような伝搬関係チェックモジュールを、transvalue を設定する度に実行する必要がある。

transvalue のメリットをユーザサイドからまとめると以下のようになる。

■ tag が格納しているデータ値は、どのウイジェットに表示すべきであるのかを予め定義可能である。

■データベースに対して更新が行われると、対応する tag 値も変更される。

■ウイジェットのデータ変更により tag 値も変更される[5]。

■ tag 値の更新によりウイジェットの保持するデータ値も変化され、それはウイジェット上に即表示される。

一度 transvalue を定義した後は、変換は自動的に行われるため、プログラム及びその保守にかかる手間の削減がはかれる。一般にデータ変換を transvalue のようにルールとして宣言することは次のようなメリットがある [Sto90]。

＊ データ変換が一箇所にまとめられるため、プログラムの見通しがよく、保守が容易になる。

＊ 現在どのようなデータ変換が設定されているかわかる。

Stonebraker が [Sto90] は、ルール (トリガ、制約) は特定の関数やコレクションと関連づけられるべきではなく、DBMS によって施されるべきである、と [Sto90] で主張している。このルールと transvalue の類時点は、非手続き的あるいは宣言的にデータ更新の内容を定義

[3] 例えば、2 個のデータベース属性の平均値をとる、ような場合である。

[4] 例えば、データベース属性値が代入されたなど。

[5] 例えば、操作者がテキストフィールドに対してデータ入力を行ったような場合である。

するところである。Stonebrakerがデータベースのみを対象とする一方、transvalueでは、「データベース内で格納されている属性値であれば、それはGUI上でも表現されるべきものである」と考え、GUIとの関連を中心においている。マルチメディア・データベースの応用開発効率向上を考えた場合、GUIは重要かつ必須のものであるので、transvalueはマルチメディア・データベース応用開発用に、ルールの考え方をGUIにまで発展させたモデルであると言える。

4 対応付け

本章では、Kasuga Scriptにおけるtagとデータベース属性の対応付け、及びtagとウイジェットの対応付けの方法を説明する。transvalueに関する定義は以下の3種類がある。

(1)tagとデータベース属性(あるいはウイジェット)との対応付け

(2) 変換ルールの手続き本体の定義

(3)tagへのtransvalueメソッドの設定

上記(1)及び(2)は(3)に先だって行われなくてはいけない。これを満たす限り、(3)の場所はプログラム中どこで行ってもよい。それに対して(1)の対応付けの場所は決められている。

4.1 tagとデータベース属性の対応付け

Kasuga Scriptでは対象データベーススキーマの定義はpath_schemaクラスによって行なう。以下に定義例を示す。

```
path_schema order {
     database: /db/order;
     path: 発注書../発注先/..会社
           as 主パス発注;
     var:  ID = 発注書.ID;
           日付 = 発注書.日付;
           会社名 = 会社.会社名; (途中略す)
} order1, order2;
```

ここではorderという名前でpath_schemaクラスを定義し、そのクラスのオブジェクトとしてorder1とorder2を宣言している。Kasuga Scriptのデータ操作機能は、G-BASE[6][リコー95]の機能を呼び出すことにより実現しているので、そのデータ対象記述は、G-BASEのパス表現を使って書く。パス表現は、通常の関係型データベースが表のみを対象とするのに対し、レコード型及びリンク型の並びとなる。path: 以降がパス表現記述部である。この例では、「発注書」と「会社」が両端のレコード型であり、「発注先」がその間を関連付けるリンク型である。asの後ろについているのはそのパス表現名である。

tagとデータベース属性の対応付けは、var: 以下に記述する。この例の変数id, date, companyなどがtag変数である。この対応付けにより、パス表現中の特定属性の値が、データ検索の結果として、tagに代入される。Kasuga Scriptプログラム中では、order2.companyというようにtagを通してその結果のデータを参照できる。

4.2 tagとウイジェットの対応付け

Kasuga Scriptではひとつの応用プログラムは、formオブジェクトを単位として構成される。一般に応用プログラムは複数の画面から構成されるが、ひとつの画面に対してひとつのformを定義することが多い。tagとウイジェットの対応づけは以下のようにformクラス中のassign部で行なう。

```
form 発注フォーム {
     screen:
        scr0; scr1; scr2;
        /* GUI画面は他で定義してある */
     path_schema: order1;
     assign:
        scr2!text1 = order1.ID;
        scr2!text2 = order1.日付;
        scr2!text3 = order1.会社名;
              :
     init:  /* 初期設定 */
        scr0.popup(); /* 初期画面表示 */
        open(); /* データベースopen */
              :
};
```

この例にあるようにassign: 以降に記述されている式はウイジェットとtagの対応付けを行っている。左辺がウイジェットのインスタンス名[7]で、右辺がtag名である。tagは、どのスキーマクラスであるかを"."の前の部分で示している。

[6](株)リコーの開発したDBMSで、リンク指向を特長とする。

[7]ウイジェット名を表す場合、Kasuga Scriptでは、ウイジェット・インスタンスから構成されるウイジェットツリーのルートから順番にウイジェット・インスタンス名を"."で接続して書くが、その簡潔な記述として曖昧性が無い場合に限り、"ルート名!ウイジェット名"という書き方ができる。

```
transvalue get_population(in)(out){
  table:{0,'Italy'},
        {1,'France'},
          ....
        {100, 'Japan'};
 /* この時点での変換結果は,outに入っている */
  database: /db/nation;
  path:     nation[name == $out];
  value:    out = population;
};
transvalue put_population(in)(out){
  out = round(in, 1000000);
};
```

図 2: transvalue データ変換本体の記述例。表引き及びデータ検索の簡略記法を使っている。

5 transvalue の例

以下に transvalue の変換ルールの定義例と transvalue メソッド適応の例を示す。変換ルールは Kasuga Script によって記述することを基本とするが、データベース応用において最もよく使われる次の 2 タイプの変換は簡略表現を用意した。

1. 表引き: 入出力の値の対応が表の形で与え、表を引いた結果を返す。
2. データ検索: データベースに対して問合せを行い、入力値に関連する属性値を返す。

5.1 表引き及びデータ検索による変換

GUI 上のテキストウイジェット "A" に国名を数字コード (1〜100 までの整数) で入力すると、以下のようなデータ変換が起こるとする。

1. FromGUI 変換 ("get_population"): 数字コードから実際の国名文字列が表引きされ、その値を使ってデータベース検索を行い、その国の人口を求め、人口の値を tag 値に代入する。
2. ToGUI 変換 ("put_population"): tag 値の人口を 100 万人単位で四捨五入してテキストウイジェット "A" に表示。

上記 2 つの transvalue 変換ルールは図 2 のように書ける。"table:" 以下は表引きの簡略表現である。

その後ろは、データ検索の簡略表現である。"database:" にはデータ検索の対象となるデータベース名を指定する。"path:" には対象パス表現を書く。例では、"nation" はレコード型名、"name" はその属性名である。"value:" 以下には検索により求めたい属性名を指定する。ここには、計算式を書くことが可能である。データ検索による変換後、変数 out に出力値が入っているので、out を使って Kasuga Script の文を書けば、一般的なロジックの変換内容が記述可能である。

この例の transvalue の設定は次のように行える[8]。対象 tag 名は tag999 とする。

```
tag999.transvalue(get_population, from_gui) ;
tag999.transvalue(put_population, to_gui)
      :trigger gui;
```

5.2 金額形式変換

データベースの実務的応用によく出てくる変換に金額の形式変換がある。この記述に関して、我々が行った検討過程も含めて説明する。例として、"4000" とテキストウイジェットにキーインされると、同じテキストウイジェットに "¥40,000" と表示される処理を考える。キーイン過程におけるウイジェット上の表示については以下のような 2 つの考え方がある。

- ウイジェット上の表示は、1 文字ずつの入力がある度に、金額形式変換が起こるべきである。ウイジェット上での数値編集の時も、1 文字削除などの度に、表示が変わるべきである。
- 入力イベント完了時点、換言すれば、transvalue FromGUI 変換を起こすトリガが来るまで、キーインされた値をそのまま表示し、入力イベントが完了した時点で、形式変換を行うべきである。

また、この形式変換は、transvalue ですべきか、ウイジェットの変換機能として提供すべきか、という議論があった。議論の結果、transvalue を用いて図 3 のように書くのが適切であるとの結論に到った。この方式では、ウイジェットに入力された値が tag にそのまま代入され、その値がウイジェット側に反映される時に、形式変換を起こすようになっている。

まず、tag には金額表示形式の値を入れないようにした。理由を以下に示す。

tag は変数であることから、Kasuga Script では tag を含めた計算式が書ける。もし、tag の値を transvalue によって金額形式に変換すると、計算式の解析部の実装の際、金額形式の型変換機能が必要となる。また整数型と金額型の結果は、どちらの型にすべきか、など、型変換

[8] tag とデータベース属性及び GUI ウイジェットとの対応付けは既に行われたと仮定する。

```
transvalue through(in)(out){
/* 素通し */
};
tag999.transvalue(through, from_gui);
transvalue yen_conv(in)(out){
     /* 金額形式変換関数 */
     out = yen_format("yzz,zzz,zzz", in);
};
tag999.transvalue(yen_conv, to_gui)
     :trigger gui;
```

図 3: 金額表示形式変換の transvalue 例。

についてルール決めをしなくてはならなくなり、ルールが繁雑になる。実装が繁雑になるだけでなく、Kasuga Script のユーザにとっても、変換ルールが繁雑なことは不便である。また、金額表示の形式は国によって変わるものであり、整数型や浮動小数点型に比べて特化しており、演算式でサポートすべきものではないと考えた。よって、tag 値としては金額形式ではない、生の数値データが入るべきと考えた。

また、金額表示変換はウイジェット内で行うのではなく、transvalue で行うことにした。その理由は以下の通りである。

もし transvalue を使わずに、ウイジェット側で変換しようとすると、ウイジェット内部の属性として、「外部表示用ウイジェット値」の他に、「内部用ウイジェット値」をウイジェット属性としてもたせる必要が出てくる。理由は、ウイジェット値の入力モード時及び編集モード時においては、金額形式文字列ではなく、生の数値が必要であるからである。我々は Kasuga Script のウイジェットを、OSF/Motif のウイジェットセットをベースにして作成しているが、Motif の XmText にはそもそも外部表示用の値属性しかないので、上記変換機能をウイジェットで実現するためには、XmText の独自拡張が必要になってしまう[9]。しかし、ウイジェットセットはシステム環境に依存するものであり、Kasuga Script にしても将来違うウイジェットセットと組み合わせる可能性は高い。よって、ウイジェットの機能拡張をするよりは、Kasuga Script として、transvalue を提供するほうが、Kasuga Script のライフタイムを長くすることができると考えた。

[9]データベースの金額形式表示のような応用例を考えれば、テキストフィールドウイジェットに 2 種類の値が必要なことはすぐ推測できることであるが、世の中のウイジェットは応用のことはあまり考えていないため、そのままでは使いにくい。

```
transvalue tr_image(in)(out){
   in = out;
   $out.scale() :width $w
                :height $h;
};
tag 名.transvalue(tr_image, to_gui)
   :trigger db;
```

図 4: イメージサイズを変換する transvalue の例。

5.3 イメージサイズ変換

最後にイメージデータに対して transvalue を適応する例を示す。Kasuga Script の変数はイメージデータも収納できる。例えば、データベースに格納してあるイメージデータを表示する時、GUI 上のイメージ表示ウイジェットのサイズにあわせて表示を行いたいと仮定しよう[10]。その場合、図 4 のような transvalue を設定しておくことにより、表示の度に自動的にスケール変換が行われる。scale() というメソッドは、Kasuga Script 組み込みのイメージ型データに対するメソッドである。

6 tag 実装のための内部機能

ここでは、tag 実装のための内部機能を説明する。図 5は tag を通してデータベース属性に結合されている GUI ウイジェットを表している。図の上部は G-BASE のデータ構造を示している。Kasuga Script のデータ操作機能は G-BASE の機能を呼び出すことにより実現されているので、データベース側の内部構造は G-BASE の構造となる。

G-BASE の問い合わせ処理では、結果のデータタイプは複数のレコード型とリンク型から構成されていて、問合せ結果は、そのオカレンスの集合となる。個々のオカレンスはパス (path) と呼ばれ、その集合はターゲット (target) と呼ばれる。ターゲットにはひとつのカーソルが備えられており、そのカーソルは先頭のパスを差し示している。カーソルは target 全体を上下に動くことができる。つまり record at a time 型の操作が可能となる。カーソルによって差し示されたパスはカレントパスと呼ばれる。図に示されるようにパスデータ構造は上位と下位の 2 レベルから構成されている。上位はレコードオカレンスの Unique Record Identifier(URI) の集合であり、下位は属性値の集合である。このように

[10]縦横の倍率比はオリジナルと変わってもイメージの概要さえ掴めればよいとする。

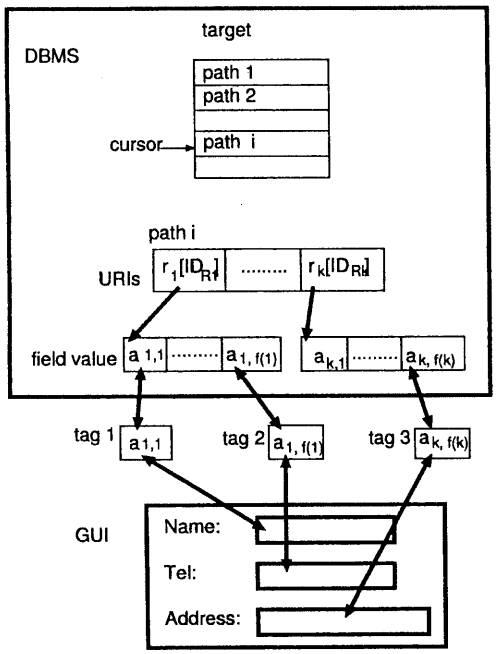

図 5: データベース属性、tag、ウイジェットを含む内部データ構造。

2 レベルにしてある理由は内部のデータサーチをより効率的にするためにである。

tag がデータベース属性名と結合されていると、カーソルが動く度に、tag 値もまた、tag が対応するカレントパスを指すように変化する。

transvalue では、データベース属性値の変化及びウイジェット値の変化により tag 値が変更される。tag 値変更のトリガとなるものは、以下のようになっている。

1. データベース側のトリガ

   ■データベースの更新操作: 挿入、修正の両方を含む。更新されたデータが tag 値に代入される。

   ■カーソルの移動: カーソルの指し示す新たなデータが tag 値に代入される。

2. ウイジェット側のトリガ

   ウイジェットはテキスト型ウイジェットとする。このウイジェットに以下のイベントが発生した時に、ウイジェットの値が tag 値に代入される。

   ■フォーカスの移動

   ■ RETURN の入力

ここでは説明の簡単化のためウイジェットのクラスをテキストに限定するが、他のウイジェットクラス、例えば、スライダー、ラジオボタンボックス、など、データベースの属性を表現可能なウイジェットであれば、どのようなウイジェットクラスにでもこのアーキテクチャは展開可能である。

図 6に、transvalue まわりの内部機構を示す。tag 値の内部構成は、図に示すように、GUI 表示などの外部出力用の ks 型と、内部計算用の data_t 型[11] の 2 つがある。ks 型はプレインテキスト型である。

Kasuga Script におけるデータ型変換について説明する。Kasuga Script の特長は Tcl/Tk などのスクリプト言語のように変数が型をもたず、文脈に応じて内部的に型を決めて、一度型変換してから、演算を施すようにしている。つまり、演算時にはその演算に応じた型変換が起こるが、変数値を参照するときは、必ずプレインテキスト型への変換が起こる。それに対し、データベース側では DBMS に依存した構造体でデータを保持している。このように data_t 型と ks 型の 2 つのデータ形式をもつが、どちらかの更新の度に毎回、両側のデータ変換を行うのでは、効率が悪い。よって tag の中にも 2 つのデータ形式をもたせることにより、できるだけデータ変換の回数を押えるようにしたのである。

tag 内のデータ形式変換は、その変換が必要になる時まで可能な限り遅らせる、という方針をとる。例えば、ウイジェットから検索条件を入力する時、検索条件が確定されるまで、複数回、検索条件文字列の修正を行うことがあるが、その時、毎回 ks 型データを data_t 型に変換していては効率が悪い。データ操作に関するメソッドが実行される時に初めて、data_t 型への変換を行えばよい。

内部的には ks 型データと data_t 型データの双方に各々「更新済みか否か」を示す dirty flag が備えてある。data_t 型が更新されたとき、data_t 型の dirty flag を off にし、ks 型の dirty flag を on にする。逆の場合も同様である。この 2 つの dirty flag により、tag 値のどちら側かがまだ変更されていない場合でも、それが判定可能となる。

図 6は、FromGUI 変換トリガによって発生した一連の内部動作を示している。以下では、順番にその様子を説明する。

(1) FromDB トリガ発生により、データベース属性値が tag data_t 型に代入される。

(2) FromDB 変換ルールを探す。"AAAAA" が設定されていたので、変換ルールデータベースにその本体を探しにいく。変換ルールデータベースは、この tag に限らず、全ての tag の変換ルールを含んでいる。

(3) tag data_t 値に対して、"AAAAA" を実行する。(この時点でも、 ks 型の値は未定義で、その dirty_flag 値

[11] data_t 型とは G-BASE が内部で使っているデータ構造である。

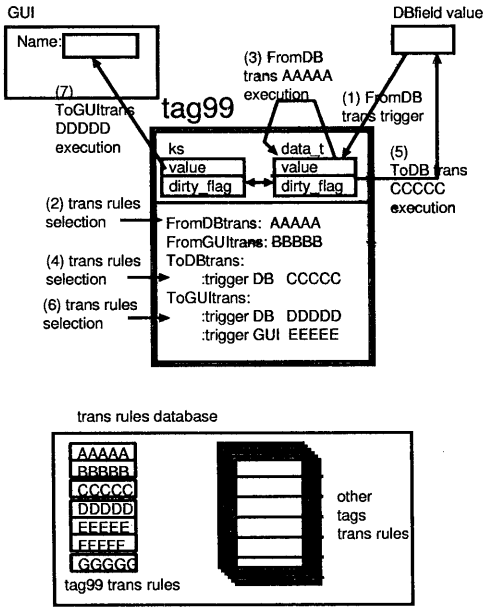

図 6: transvalue の内部機構

は on である。)

(4) 発生元トリガが DB である ToDB 変換ルールを探す。"CCCCC" が設定されていた。

(5) data_t 値に対して "CCCCC" を施した値を計算し、それをデータベース属性値に代入する。data_t 値には代入しない。(この時点でも、ks 型の値は未定義で、その dirty_flag 値は on である。)

(6) 発生元トリガが DB である ToGUI 変換ルールを探す。"DDDDD" が設定されていた。

(7) ここで初めて、data_t 値を ks 型値に代入する。

(8) ks 型値を入力とし、"DDDDD" を計算し、その値をウイジェットに代入する。tag 値には代入しない。

7 まとめ

Kasuga Script とはマルチメディアデータベース応用プログラムを作成するためのスクリプト言語である。Kasuga Script には、プログラム開発効率を向上させるための各種の機能があるが、本論文ではそのうちの transvalue 機構を紹介した。tag とはデータベース属性と GUI ウイジェットを結合させるための変数であり、宣言的に定義しておくことで、そのトリガ発生の度に該当するデータ変換を自動で行ってくれる。

これにより、通常のウイジェットにコールバックを設定する手法に比べて、プログラム全体の見通しがよくなり、プログラミング効率の向上に役立つ。

参考文献

[西尾 96] 飯沢篤志、白田由香利 (西尾章治郎 監修)、「実践 SQL 教科書: 第 2 章 いろいろなデータモデルの比較」、ASCII、1996.

[牧之内 95] 牧之内顕文、"次世代データベースシステム"、電子情報通信学会誌、Vol.78, No.1, pp.65-75.

[リコー 95] (株) リコー、「G-BASE マニュアル『ユーザーズ』(第 3.2 版)」、1995.

[McC88] McCormack, J., "An Overview of the X Toolkit", *Proc. of the ACM SIGGRAPH Symposium on User Interface Software*, pp. 46-55, 1988.

[Shi91] Shirota, Y. and Kunii, T.L., "Action Propagation Model for User Interface Programs", Klinger, A. (ed) *Human-Machine Interactive Systems*, Plenum Press, N.Y., 1991, pp.23-35.

[Shi96] Shirota, Y., Nakayama, H., and Iizawa A., "A New Type of Fourth-Generation Language for Multimedia Databases: Kasuga Script", Terashima, N. and Altman, E. (eds) *Advanced IT Tools*, Chapman & Hall, London, UK, IFIP96: 14th World Computer Congress, 1996, pp.207-214.

[Ston90] M.Stonebraker et al. (The Committee for Advanced DBMS Function), "Third-Generation Data Base System Manifesto", *ACM SIGMOD Record*, Vol.19, No.3, 1990, pp.31-44.

付録

transvalue 設定の文法を以下に BNF で表す。Kasuga Script では、メソッドの適応はドット演算子で記述する。変換ルールは、transvalue メソッド適応に先立ち、関数として定義されている必要がある。

```
<tag 式> = <tag 名>
    [<ドット演算子> <transvalue メソッド文>]
<ドット演算子> = '.'
<transvalue メソッド文> = 'transvalue'
    '(' <変換ルール名> ',' <変換方向> ')'
    [':' <trigger オプション>]
<変換方向> = 'from_db' | 'from_gui'
    | 'to_db' | 'to_gui'
<trigger オプション> = 'trigger'
    ('db' | 'gui'| 'substitute')
```