データベースシステム 92−10
（1993. 3. 22）

ウインドーベースのデータベース操作用アプリケーションの開発システム

白田 由香利　飯沢 篤志
株式会社リコー ソフトウェア事業部 ソフトウェア研究所
Arturo Pizano
Software Research Center, Ricoh Corp.

ウインドーベースのGUI(Graphical User Interface)は表示の美しさ及びダイレクトマニピュレーションによる操作性の良さに特長を持ち、特にイメージデータベース操作アプリケーションには必須である。しかしウインドーベースのプログラミングスタイルはイベント駆動型であるため、アプリケーション本来の処理順序と異なる低レベルのイベント主導型となる傾向が強く、その結果、プログラムの構成がわかりにくくなる。本稿ではE-R型データベース操作のためのGUIを例にあげ、GUI部とアプリケーション本体部の関係を出来る限り切り離すための要求仕様定義の方式を提案する。アプリケーション開発者は、コールバックオブジェクトという、より抽象化されたGUIの動作定義手続きを直列に並べることで自分の要求する仕様を記述する。特にE-Rモデルに基づくデータベースの場合、実体間の多対多関係をGUI上でいかに表現するかは種々多様な方式があり、アプリケーションの仕様を記述する際の困難性を招く。こうした問題を解決するためにもコールバックオブジェクトを用いたこの記述方式は役立つ。

A Framework for the Design of Window-Based Database User Interfaces

Yukari Shirota and **Atsushi Iizawa**
Software Research Center, Software Division, RICOH Company, Ltd.
Arturo Pizano
Software Research Center, RICOH Corporation

This paper describes a database user interface development system that allows the application developers to create their user interface without writing the majority of the callback routines. The application developers write scenarios of the GUI to define how to prepare the data for a database query and how to display the query results on the screen. As a case study, we use an entity-relationship model and create a form-oriented user interface on the model. To define the operations over M:N relationships on the GUI. The paper presents also the master/slave interaction model.

1 まえがき

X Window System[1]を始めとするマルチウインドーベースの GUI(Graphical User Interface) は表示の美しさ及びその操作性の良さのため、データベースのアプリケーションでも広く普及している。特にイメージデータベース・アプリケーションはキャラクタベースの UI(User Interface) ではイメージ処理ができないため GUI は必須である。しかし GUI のソフトウェア作成・保守は一般のソフトウェアに比べて難しくソフトウェア生産性の向上がなかなかあがらない、という問題点がある。この原因は以下の2つと考えられる:

- プログラムスタイルがイベント駆動型であるためプログラムの構成がわかりにくい
- キャラクタベースの UI に比べて設計の自由度が高い

1.1 問題の分析 1

上記の第1の原因であるが、これは「ウインドープログラミングスタイルはイベント駆動型であるため、マウスボタンの押し下げ、ボタンのクリックといったアプリ本来の処理順序とは異なる低レベルのイベント主導型の書き方になってしまう」ためである。換言すると、アプリケーション本来の制御の流れが GUI 部に引きずられてしまい、アプリケーションの意図するひとまとまりの機能（例えばデータベースの検索）が、GUI の複数の**コールバック**[2]に分散されるので結果としてプログラムの設計、デバッグ、修正及び保守が困難になる。

また2つの異なる処理において、同じ操作列が共有されたり、ある処理の操作中に他の処理の操作列を交錯させるといったこともある。例えば、複数のテキストフィールドに検索条件を入力する操作列と、レコードオカレンス更新のためにフィールド値を編集する操作列は、両方とも「複数へのテキストフィールドへのデータの入力」という操作列に対応している。

またコールバックは互いにフラットであるため、特定のモードを設定しにくいという問題もある。モードを設定するため、よく**フラグ変数**が用いられるが、これもプログラム構造を見えにくくするばかりでプログラムから状態遷移の様子を読みとるのは難しい。

以上のような問題は多くの GUI 研究者により指摘されてきた [Blew92, 古田 92, Myer91, Hagi90]。この対応策としては「オブジェクト指向を取り入れた高度なフレームワークをユーザに提供し、ユーザはそのクラスライブラリから自分の要求にあったオブジェクトをカスタマイズしていく」というものがある。こうした高機能フレームワークとしては、Garnet[Myer90], Picasso[Rowe91], 商品としては NeXT の Next Step, MacApp などが著名である。

1.2 問題の分析 2

第2の問題点「キャラクタベースの UI に比べて設計の自由度が高い」とは、画面定義、動作定義の両方においてアプリ開発者が定義しなくてはいけない要素が多数あることを意味する。ボタンやラベル、イメージウインドーといったひとまとまりの機能に対応したグラフィカルオブジェクトのことを X Window System では**ウイジェット (widget)** と呼ぶ。アプリ開発者は

- ある機能を実現するためにどのウイジェットを使うか
- ウイジェットの持つ多数の変数値[3]をいくつに設定するか
- 多数のウイジェットをどのように画面にレイアウトするか

といったことを決める必要がある。これはアプリケーション開発者のセンスと経験が大きく反映されるものであり、アプリケーション開発者には、使用するウイジェットセット (OSF/Motif, XViews など) のオブジェクト階層及びそのクラス間の制約事項、各ウイジェットの機能とリソース、及びそのウイジェットセットのスタイルガイドなどの知識を習得していることが求められる。

本稿では E-R 型データベース操作のための GUI を対象とし、アプリケーション開発者の工数を削減するアプリケーション開発システムを提案する。次節では、上記の問題点への解決方法ならびにアプリケーション開発の手順を概説的に説明する。第3節ではシステムの概要、及び記述方式のポイントとなるコールバックオブジェクトを説明する。第4節では E-R モデルに基づくデータベースの操作における問題点を分析し、それの解決方法としてマスター・スレーブモデルを提案する。第5節はまとめとする。

2 開発プロセス

ここでは我々の提案するシステムによって、アプリケーション開発者がどのような作業により DB 操作用 GUI を作成するのか、その全体的手順を示す。

[1] X Window System は米国マサチューセッツ工科大学の商標である。

[2] イベントにより呼び出される手続きを X Window System ではコールバック (callback) と呼ぶ。

[3] 枠のマージンや背景色といったウイジェットの形態を決める変数。これをウイジェットのリソースと呼ぶ。

1. 開発者はDBの論理的スキーマから興味の対象となる部分を視覚的に選択し、それに対応する画面上のウイジェットの数と種類を指定する。

2. システムがデータ辞書の情報を用いて各ウイジェットのサイズを決める。

3. ウイジェットのサイズと種類が決まるとシステムの**自動レイアウト機能**によりウイジェットが画面に配置される。

4. 結果の画面を使って開発者はGUIの動作を**コールバックオブジェクト**を直列に並べることで定義する。このコールバックオブジェクトの列を**シナリオ**と呼ぶ。

ここでは、我々のとる上記の問題解決アプローチを述べる。まず、「キャラクタベースのUIに比べて設計の自由度が高い」という問題に対しては、ウイジェットの画面上でのレイアウトを半自動的にシステムにやらせることにより、アプリ開発者の作業を削減する。この**自動レイアウト機能**は、ランダムに生成される多数の配置の候補群に対して、システムが人間にとっての見易さを基準に評価を行ない、準最適な解を求めるものである。これについては[Piza92]が詳しい。

もう一つの問題「プログラムスタイルがイベント駆動型であるためプログラムの構成がわかりにくい」に対しては、**コールバックオブジェクト**という、ある一まとまりのユーザの意図する処理を実現するためのオブジェクトを提供することにより解決をはかる。

例えば「検索条件の入力する」という処理を一つのコールバックオブジェクトFillInConditionで実現する。そのオブジェクトの中には図1に示すように、テキストフィールド、トグルスイッチ、リストボックス等の複数のウイジェットが関連し、それらの間には複数のプリミティブなイベントとコールバックのペアが存在する。しかし、高機能なオブジェクトとしてFillInConditionを見た時、内部のウイジェット間の関係は隠蔽され外部からは見えない。このような高機能オブジェクトをアプリのフレームワークをして開発者に提供することは以下の利点を生む:

1. 部品の再利用がはかれる

2. 一から作ることなく、既存のクラスライブラリに修正を加えるだけですむ

3. イベントループの巨大化を防ぐ

4. アプリケーション全体の見通しが良くなる

5. アプリケーション設計の定式化が進み、開発期間が短縮できる

記述能力 (descriptive power):
全てのUIの状態遷移を表現するためには、オブジェクト間のメッセージのやりとりには規制があってはいけないが、本システムの記述仕様では、開発者にとっての構造の見通しの良さを優先し、コールバックオブジェクトは直列にのみ配置できる、と制限した。このため、開発者はUNIXのパイプフィルタコマンドのように、コールバックオブジェクトを接合していくことでGUIの動作を定義できる。

3 GUI部とアプリケーション本体の分離

では次に、ソフトウェアの構成を分りやすくするために、どのようにGUI部とアプリケーション本体の分離を行ない、コールバックオブジェクトとしてまとめていけばよいかについて述べる。E-Rモデルに基づくデータベースアプリケーションの場合、制御は以下の繰り返しとなる。

1. DB処理に渡すデータの準備 (GUI部)

2. DBフロントエンドプロセッサー (FEP) によるDB問合せ文の生成

3. DB処理 (アプリケーション本体)

4. 処理結果のデータ構造の変換

5. 処理結果の表示 (GUI部)

上記1及び5は画面上のウイジェット間の関係だけを定義するもので、この間アプリケーション本体との連絡はとらないものとする。開発者はこの2つのGUI内部制御部に対してシナリオを定義する。それぞれをPrepare Data Scenario, Display Data Scenarioと呼ぶ。上記2及び4のプロセスはGUI部とアプリケーション本体とのインタフェース部分である。複数のDBMSに対応させるため、DBMSの言語インタフェースの違いはDB FEPで吸収させる。DB FEPはGUI部から渡されたデータから個々のDBMSに合った問合せ文を生成する。またDB処理の結果も個々のDBMSのデータモデルに依存しないようにするため、GUI部に渡す前にすべて論理的な表形式に変換する。エラーの通達を含め、GUI部とアプリケーション本体の間のプロトコルは標準化することも必要である。

図2に例を示す。これは人事データベースの操作のためのGUIで、以下の3つのレコード型から構成されている。

- 従業員 (EMPLOYEE)

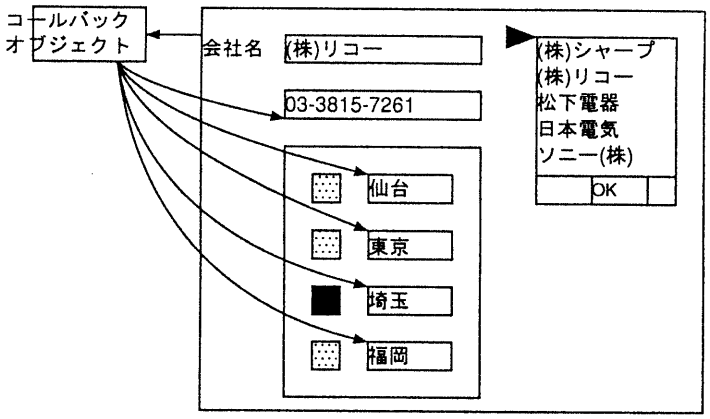


図 1: 高機能化されたオブジェクトの例: FillInCondition

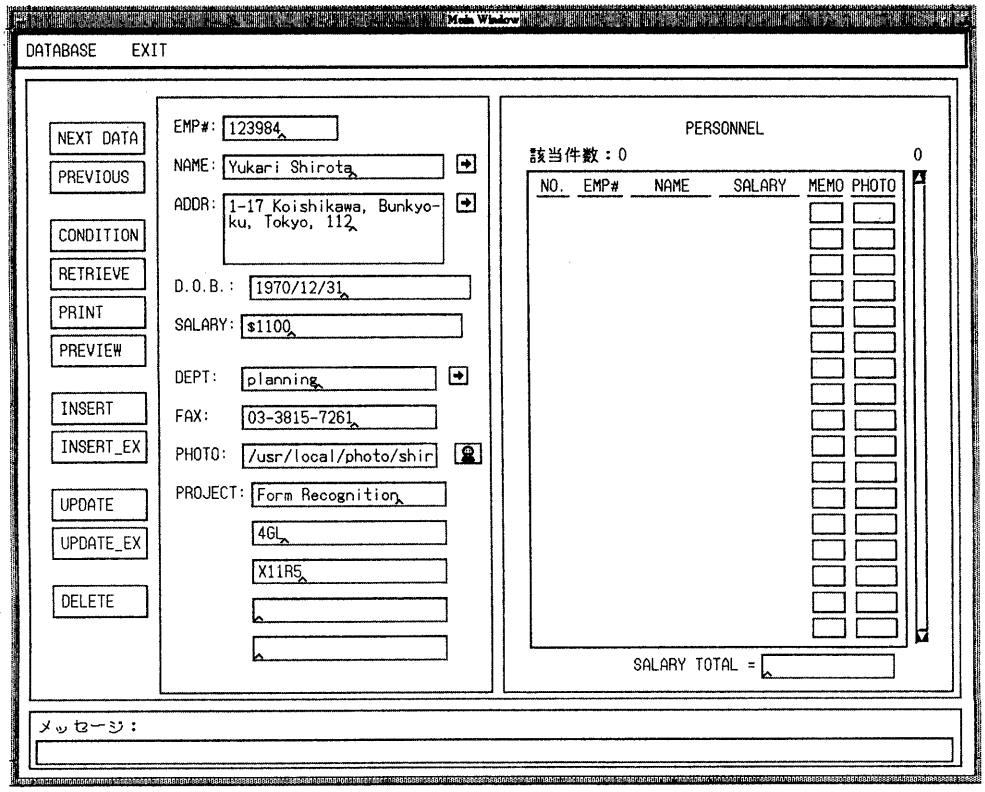


図 2: 人事データベース操作用 GUI

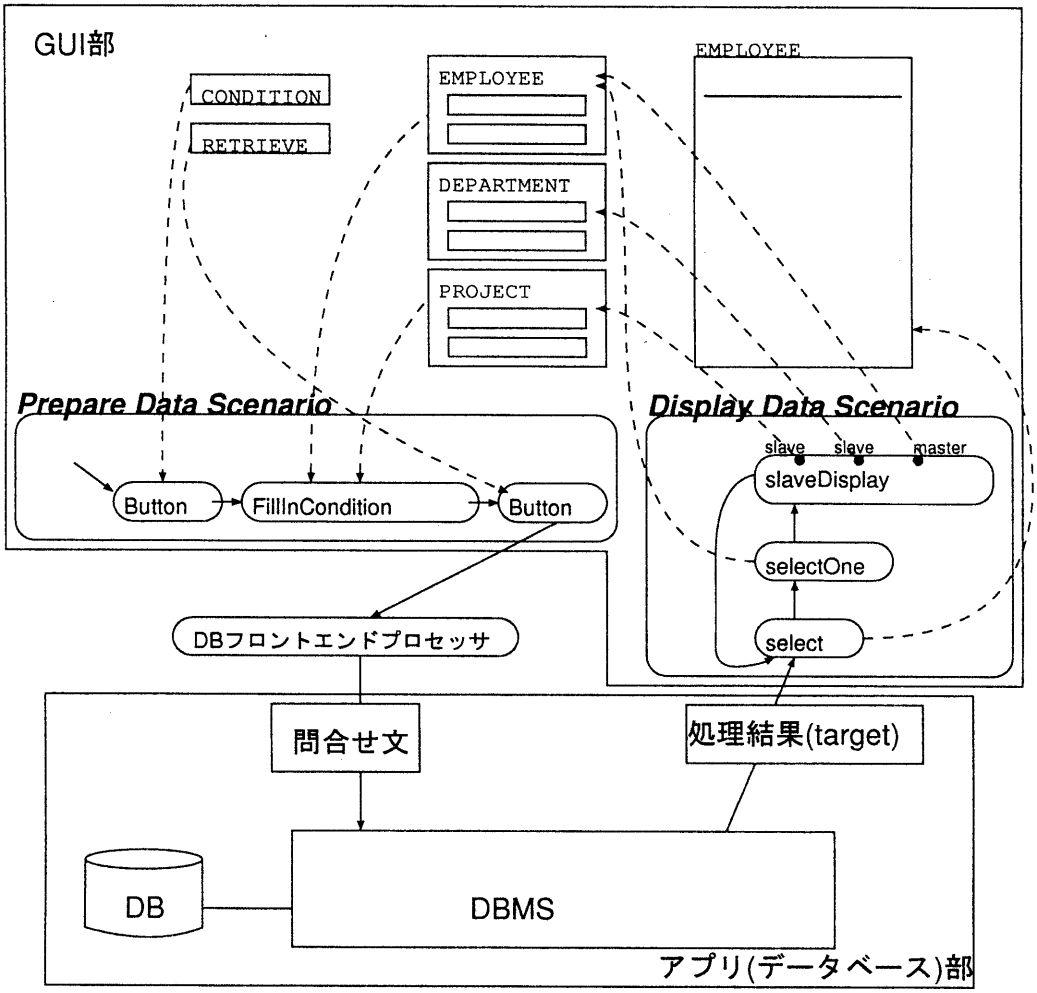


図3: 図2のGUIの動作に対応するシステム内部の概要

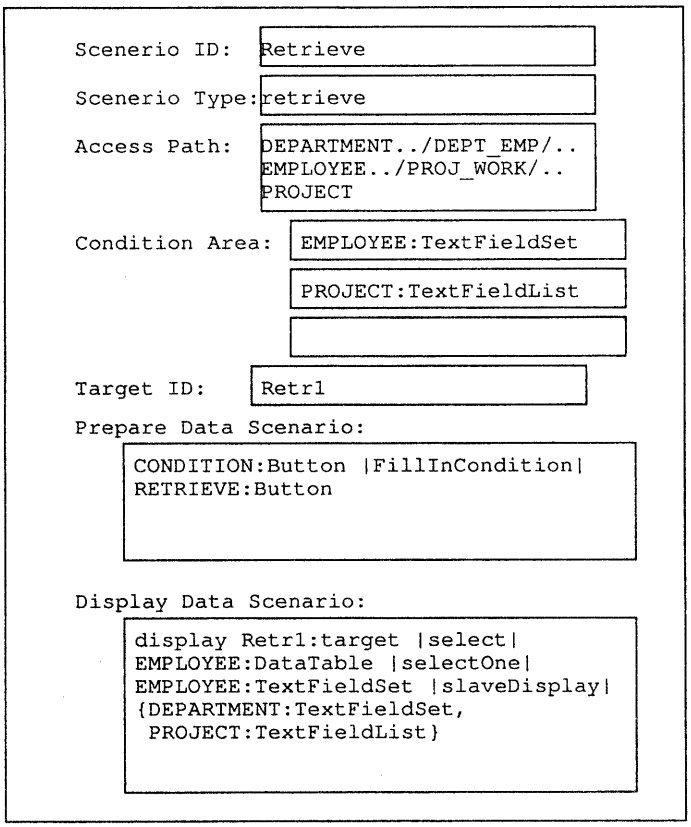


図4: シナリオ定義後に生成される内部データ(図2のGUIの動作に対応するもの)

- 所属 (DEPT)
- プロジェクト (PROJECT)

「4GL のプロジェクトに参加している、東京在住の従業員は？」という問合せに対して、検索が実行され、画面右側にあるデータテーブルに複数の検索結果が表示される。そのうちのひとつのデータをマウスでクリックすると、そのデータの詳細情報が中央にある、テキストフィールドに表示される。次のデータベース操作を発するまで、エンドユーザはデータテーブルのスクロールバーを移動させたり、欲しいデータをクリックしたりで自分の見たいデータを表示させる。

この時のシステムの概要を図3に示す。ここに示されるように制御の流れは、大きく GUI 部とアプリケーション部の2つに分離されている。そして GUI 部では前述した Prepare Data Scenario と Display Data Scenario の2つが実行される。図4に示したダイアグラムは、この GUI を開発者が定義したときに生成される内部データである。

4 マスター・スレーブモデル

E-R 型 DB 操作のためのユーザーインタフェースは従来から多数提案されているが「E-R ダイアグラムを使ったスキーマの扱いには便利であるが、多対多の関係を表現する能力が不十分であるため問合せ処理や処理結果のブラウジングには適していない」という問題点がある。多対多関係の表示及び操作に対するエンドユーザーの要求は多種多様であり、効率的操作のため多段階の多対多関係を一度に表示したいなど、実システムの制御構造は論文に出てくる GUI の例に比較してはるかに複雑であり、現状の 4GL 商品の記述能力ではカバーしきれないものも少なくない。

例えば図5の E-R ダイアグラムに示すような2組の多対多関係があると仮定する。この意味するところは「従業員が複数の所属部署に属する場合があり、各部署は複数のワークステーションを所有している」となる。これを GUI で表現する場合、ある開発者は第1の例のように、データテーブルで所属を表示し、その中から選択された一つの部署の所有するワークステーションを隣のデータテーブルで表示したい。また、所属部署とその所有するワークステーションを出来る限り一度に表示したいという要求もあるだろう。第2の例ではテキストフィールドとデータテーブルの要素3の配列でそれを実現している。第3の例では WORKSTATION もテキストフィールドの要素3の配列で表している。

本システムでは多対多関係の操作を明確に定義するためマスター・スレーブモデルを導入し、これに基づきコールバックオブジェクトの動作を記述する。多対多関係を表示するためには、どちらか一方を一つのオカレンスに限定する必要があるが、その側をマスター側と呼び、残る片側をスレーブと呼ぶことにする。図5に示した3例は以下のようなシナリオによって記述される。イタリックフォントで記述された名称がコールバックオブジェクトである。この例でのアクセスパスは

EMPLOYEE ../DEPT_EMP/.. DEPT ../HAS/.. WORKSTATION

と仮定する。

例1

display Retr2:target | *selectOne* | EMPLOYEE:TextFieldSet | *slaveDiplay* | DEPT:DataTable | *selectOne* | *slaveDisplay* | WORKSTATION:DataTable

Target というのは検索結果のことである。始めはマスターが EMPLOYEE、スレーブが DEPT である。一般に複数の検索結果があるので、その中から1つのデータを選択して EMPLOYEE に表示する。これがコールバックオブジェクト *selectOne* の役割である。この EMPLOYEE データに対し、スレーブである DEPT データを表示する。それが *slaveDisplay* の動作である。表示先は DEPT:DataTable である。次は DEPT が今度はマスター側となり、WORKSTATION がそのスレーブ側となる。

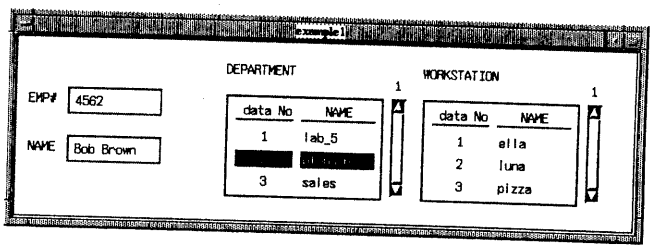
例2

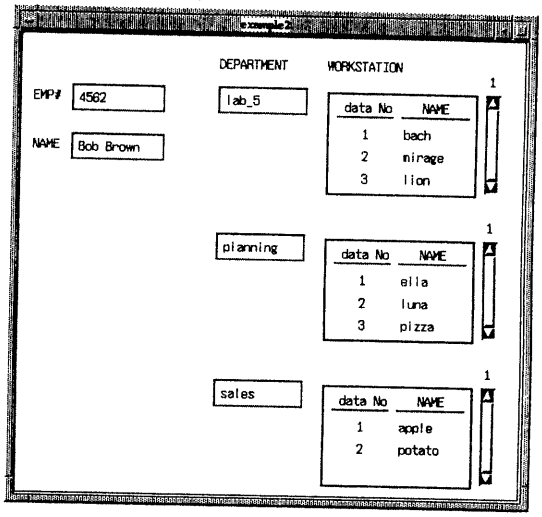
display Retr2:target | *selectOne* | EMPLOYEE:TextFieldSet | *slaveDiplay* | DEPT:TextField[1..3] | *slaveDisplay* | WORKSTATION:DataTable

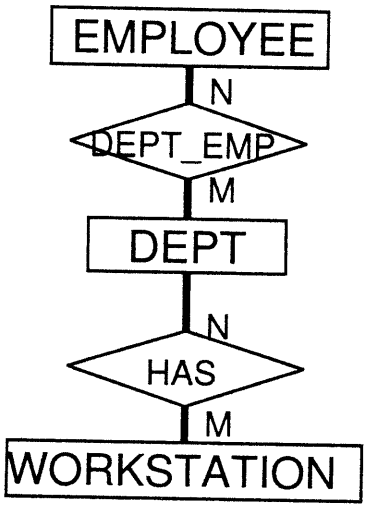
例1と異なる点は EMPLOYEE のスレーブになった DEPT の表示先が3つのテキストフィールドであることだ。そして各々のテキストフィールドに対して、一つのデータテーブルがスレーブ表示のために付けられている。

例3

display Retr2:target | *selectOne* | EMPLOYEE:TextFieldSet | *slaveDiplay* | DEPT:TextField[1..3] | *slaveDisplay* | WORKSTATION:TextField[1..2]







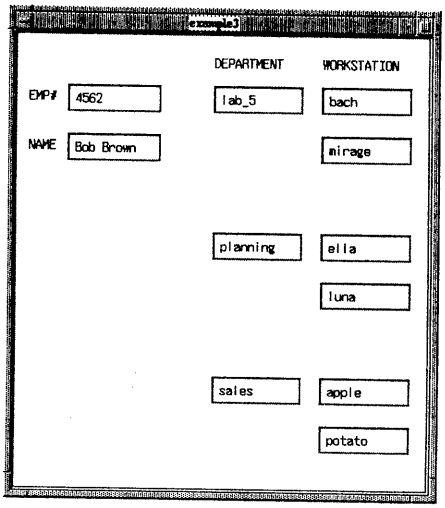


図 5: 多対多関係の GUI での多様な表現方法 (2 つの GUI は両方とも上部の E-R ダイアグラムに対応している)

例2ではDEPTのスレーブになったWORKSTATIONの表示ウイジェットはデータテーブルであったが、それを要素2のテキストフィールドの配列にした点が異なっている。

第2及び3の例のように、スレーブデータ数が画面に用意したウイジェットの数より大きい時は、以下の2通りの対処の方法があるが、このような仕様も記述できるようにしている。

1. エラー: DEPT表示後、そこで実行を中止しユーザの指令を待つ。

2. 警告: 出来る限り多くDEPT及びWORKSTATIONを表示し、実行は中止しない。

5 結論

本稿では、アプリケーション生産性向上を目標とする、データベース操作用GUIの開発システムの基本設計について述べた。特長は以下の通りである:

- コールバックオブジェクトによるGUI動作の抽象化をはかる
- マスター・スレーブモデルによる多対多関係上の仕様記述を明確にする
- 自動レイアウト機能による画面レイアウト作業の削減をはかる

本稿で示したシナリオ記述の方式は始めたばかりである。アプリケーションの仕様記述に関しては、シナリオ記述ツールの開発などを通じて発展していくべき性質のものであるため、今後さらに検討すべき点が見つかると考えている。

参考文献

[Blew 92] Blewett, D., Anderson, S., et al.: "X Widget Based Software Tools for UNIX," *Proc. USENIX '92 Winter*, 1992, pp. 111-123.

[Yosh 92] 吉田、新田: 「アプリケーションをGUIから解放しよう!」jus設立10周年記念UNIX国際シンポジウム論文集、pp. 96-104.

[Myer 91] Myers, B. A.: "Separating Application Code from Toolkits: Eliminating the Spaghetti of Call-backs," *Proc. UIST '91*, ACM SIGGRAPH Symp. on User Interface Software and Technology, Hilton Head, SC, Nov. 1991, pp. 211-220.

[Hagi 90] Hagiya, M. and Ohtani, K.: "Parallel Object-Oriented UIMS with Macro and Micro Stubs," *Proc. of USENIX '90 Winter*, 1990, pp. 259-274.

[Myer 90] Myers, B. A., et al.: "Garnet Comprehensive Support for Graphical, Highly Interactive User Interfaces," *IEEE Computer*, Nov. 1990, pp. 71-85.

[Rowe 91] Rowe, L. A. et al.: "The piccaso application framework," *Proc. of the ACM Symp. on User Interface Software and Technology*, Nov. 1991, pp. 95-105.

[Piza92] Pizano, A., Iizawa, A., and Shirota, Y.: "An Automatic Screen Layout Generator for Database Applications," *Proc. First Intl. Conf. on Information and Knowledge Management*, Nov. 8-11 1992, Baltimore, Maryland, pp. 239-247.