

# GUI とプログラミング言語の 利点を兼ね備えたスクリプト言語:

## **guript**

日本ソフトウェア科学会第21回大会ポスター・デモセッション資料

産業技術総合研究所

情報技術研究部門

一杉裕志、古川浩史

2004年9月16日

# guript = GUI + script

- guript IDE (統合開発環境) のスクリーンショット

The screenshot shows the guript IDE interface. At the top is a menu bar with options: ファイル, 編集, 表示, リファクタリング, ツール, オプション, 実行, ヘルプ. A 'sys reload' button is on the right. The main workspace is divided into three sections:

- デモプログラム (Demo Programs):** A list of demo programs on the left: デモ1: 引数制約, デモ2: 高階関数, デモ3: if文, デモ4: 変数.
- Script Editor:** The central area contains a script with the following steps:
  1. **ファイル情報一覧を得る** (Get file information list) フォルダ名: [ ] 参照 逆順に並べる:
  2. **列を2つ選択する** (Select 2 columns) フィールド名1: [名前] フィールド名2: [サイズ]
  3. **並び替える** (Sort) 降順に並べる:  キーとなるフィールド名: [サイズ]
  4. **XML に変換する** (Convert to XML)
  5. **結果を表示する** (Display results) 出力先: [ダイアログ]
- 編集メニュー (Edit Menu):** A sidebar menu on the right with options: 削除, print フラグ on/off, ファイル情報一覧を得る, Amazon トップセラーを得る, 複数のファイルを操作する, 列を2つ選択する, 並び替える, リストを逆順にする, 集計する, 結果を表示する, XML に変換する.
- (システムデバッグ用) (System Debugging):** A console area at the bottom right showing the output of a script: 2+3, eval, alert(HTML), SELECT(up)38:1:1, SELECT(down)38:1:1, SELECT(change)39:1, SELECT(up)39:1:1, SELECT(down)39:1:1.

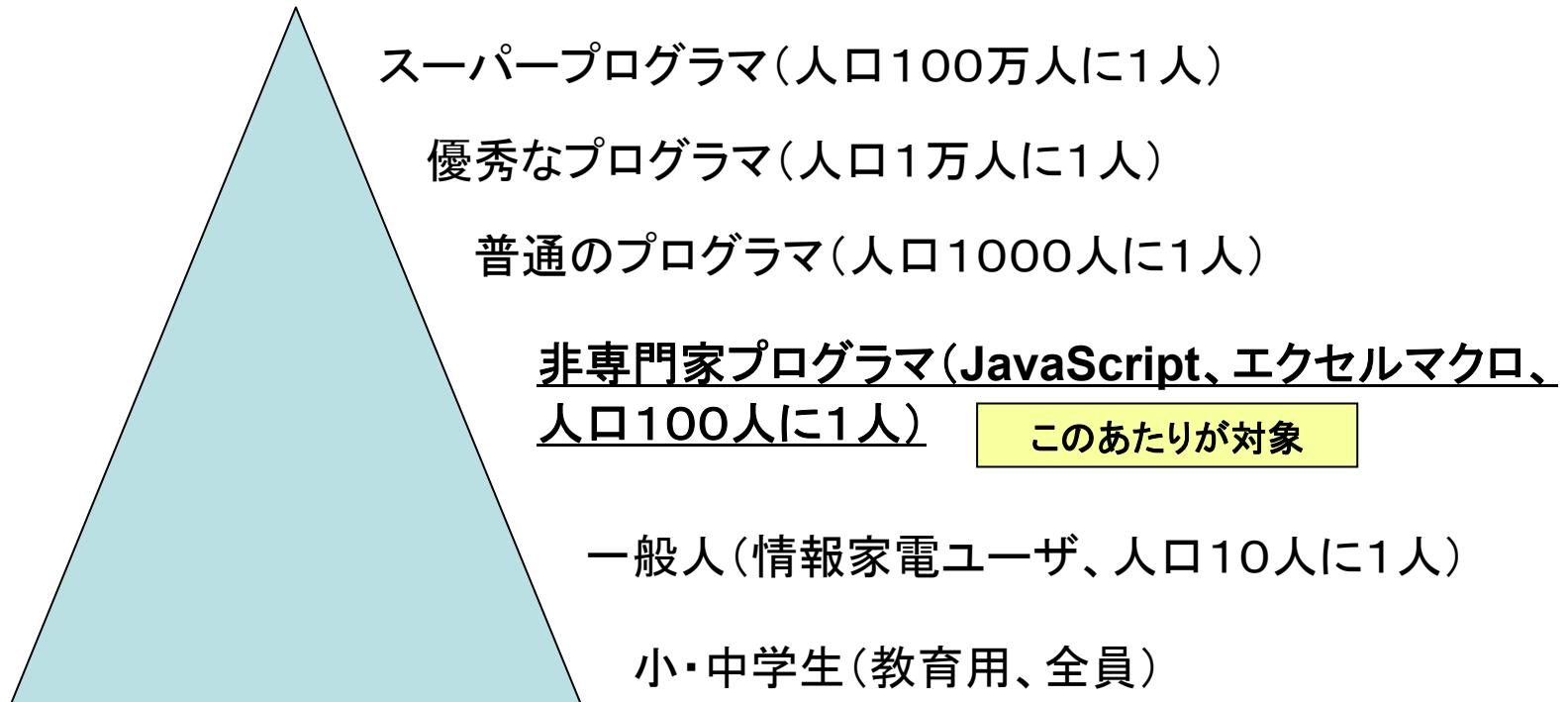
At the bottom of the window, a status bar contains the text: テーブルの中から、指定された2列だけを抜き出します。

# guript の特徴

- 非プログラマでも使える「スクリプト言語」
  - Unix のコマンドラインのような処理がマウスだけでできる
- 日本語化された構造エディタ
- 手続き型言語 (⇔ Visual language)
- IDE (統合開発環境)
- 使いやすい標準ライブラリ
- 高い拡張性

# 対象とするユーザ

- プログラマの階層



# プログラム例

- Unix のパイプのように、データが上から下に流れる

The screenshot shows the Guript IDE interface. The main window contains a menu bar (ファイル, 編集, 表示, リファクタリング, ツール, オプション, 実行, ヘルプ) and a toolbar (sys reload). The main area is divided into several sections:

- デモプログラム**: A list of demo programs on the left side.
- 使用パッケージ**: A section for installed packages (標準/ファイル処理, 標準/テーブル処理).
- プログラム**: A section for running programs, containing a list of commands:
  1. **ファイル情報一覧を得る** (Folder name, reference, reverse order)
  2. **列を2つ選択する** (Field name 1: 名前, Field name 2: サイズ)
  3. **並び替える** (Sort order, key field name: サイズ)
  4. **XML に変換する**
  5. **結果を表示する** (Output destination: ダイアログ)
- 編集メニュー**: A menu on the right side with options like 削除, print フラグ on/off, etc.
- (システムデバッグ用)**: A section for system debugging, showing a command prompt with the text "2+3" and buttons for "eval" and "alert(HTML)".

Annotations with red boxes and arrows point to specific features:

- コマンド列**: Points to the list of commands in the "プログラム" section.
- 挿入可能なコマンド一覧**: Points to the dropdown menu for field selection in the second command.
- パラメタ入力も GUI だけで行う**: Points to the input fields for field names and sort order.
- コマンドの簡潔な説明を適宜表示**: Points to the descriptive text at the bottom of the window: "テーブルの中から、指定された2列だけを抜き出します。"

# guript は静的型付言語

- 編集集中に逐次型チェック、表示オプションで型情報を表示
- 型情報は、パラメタの入力支援にも使われる: 「引数制約」

The screenshot shows the guript IDE interface. The main editor displays a script with the following content:

```
使用パッケージ:  
標準/ファイル処理  
標準/テーブル処理  
  
プログラム:  
// このコマンドの型: ["list", ["record", ["fieldSet", "名前", "パス", "作成日時", "更新日時", "アクセス日時", "サイズ"]]]  
1. ファイル情報一覧を得る フォルダ名:  参照 逆順に  
並べる:   
// このコマンドの型: ["list", ["record", ["fieldSet", "名前", "サイズ"]]]  
2. 列を2つ選択する フィールド名1:  フィールド名2:   
// このコマンドの型: ["list", ["record", ["fieldSet", "名前", "サイズ"]]]  
3. 並び替える 降順に並べる:  キーとなるフィールド名:   
// このコマンドの型: "xml"  
4. XML に変換する  
// このコマンドの型: "unit"  
5. 結果を表示する 出力先:   
// このコマンド列の型: "unit"
```

The dropdown menu for the second field name is open, showing the options "名前" and "サイズ". The "名前" option is highlighted in blue. A red circle highlights the dropdown menu, and a red arrow points from a text box below to it.

直前のコマンドの型が、  
パラメタの選択肢を決める例

デモプログラム  
-----  
デモ1:  
引数制約  
  
デモ2:  
高階関数  
  
デモ3:  
if文  
  
デモ4:  
変数

編集メニュー  
削除  
print フラグ on/off  
----  
ファイル情報一覧を得る  
Amazon トップセラーを得る  
複数のファイルを操作する  
列を2つ選択する  
並び替える  
リストを逆順にする  
集計する  
結果を表示する  
XML に変換する

(システムデバッグ用)  
2+3  
eval alert(HTML)  
SELECT (up)39:1:1  
SELECT (down)39:1:1  
SELECT (down)39:1:1  
SELECT (up)39:1:1  
SELECT (down)39:1:1

テーブルの行を、キーに指定されたフィールドの大きさにしたがって並び替えます。

# コマンドのネスト

- コマンドの種類によっては、ネストが可能
  - 制御構造、高階関数(block 引数)

指定したフォルダ内に存在する各ファイルに対して、コマンド列で指定した操作を行います。

# 変数

- 変数宣言不要、型宣言不要

The screenshot shows the Guript IDE interface. The main editor contains a script with the following steps:

```
使用パッケージ:  
標準/ファイル処理  
標準/テーブル処理  
-----  
プログラム:  
// このコマンドの型: ["list", ["record", ["fieldSet", "名前", "パス", "作成日時", "更新日時", "アクセス日時", "サイズ"]]]  
1. ファイル情報一覧を得る フォルダ名:  参照 逆順に  
並べる:   
// このコマンドの型: "unit"  
2. 結果を変数に代入する 変数名:   
-----  
// このコマンドの型: "bool"  
3. 確認ダイアログを表示 メッセージ:   
// このコマンドの型: "unit"  
4. 結果を変数に代入する 変数名:   
-----  
// このコマンドの型: ["list", ["record", ["fieldSet", "名前", "パス", "作成日時", "更新日時", "アクセス日時", "サイズ"]]]  
5. 変数の値を参照する 変数名:   
// このコマンドの型: "unit"  
6. 結果を表示する 出力先:    
// このコマンド列の型: "unit"  
-----
```

The console on the right shows the output of the script:

```
(システムデバッグ用)  
2+3  
eval alert(HTML)  
SELECT(up)60:0:1  
SELECT(down)60:0:1  
SELECT(down)60:0:1  
SELECT(up)60:0:1  
SELECT(down)60:0:1
```

At the bottom of the IDE, a message reads: 指定された変数に代入されている値を参照します。



# guript の変数の特徴

- 変数のスコープは、1つのサブルーチン全体
- 代入文では、自動的にデフォルトの変数名 x1, x2, ... が付けられる
- 変数の参照では、変数名をセレクタで選ぶ
- 変数の型は代入の直前のコマンドの型で決まる
- 同じ変数に2度以上代入するときは、型が同じでなければ型エラー
- 代入されない変数を参照するとエラー
  - (Java でいう「確実な代入」)

# 条件分岐

- 条件式の値によって then か else に値が流れる
- 型: 'a->(unit->bool)->('a->b)->('a->'b)->'b

The screenshot shows the Guript IDE interface. The main window displays a program with the following structure:

```
使用パッケージ:  
標準/ファイル処理  
標準/テーブル処理  
-----  
プログラム:  
1. ファイル情報一覧を得る フォルダ名:  参照 逆順に  
並べる:   
2. 条件によって動作を変える  
条件式:  
    2.1. 確認ダイアログを表示 メッセージ:   
-----  
成り立つ場合の動作:  
    2.1. リストを逆順にする  
-----  
成り立たない場合の動作:  
-----  
3. 結果を表示する 出力先:   
-----
```

The right sidebar contains a list of actions: 複数のファイルを操作する, 列を2つ選択する, 並び替える, リストを逆順にする, 集計する, 結果を表示する, XMLに変換する, CSVに変換する, 結果を変数に代入する, 変数の値を参照する, **条件によって動作を変える**, 確認ダイアログを表示.

At the bottom of the IDE, a status bar reads: 指定した条件式が成り立つかどうかによって、異なるコマンド列を実行します。

# 構造エディタの利点

- シンタックスエラーが起きない
- コマンド名、パラメタ名の日本語化(国際化)
- 引数制約などによる入力支援
- さまざまな付加情報の提示
- 文字列リテラル内のエスケープが不要

→ **非プログラマの人には必須の機能**  
(プログラマにもありがたい)

# 表示オプション: 説明を表示

- 各コマンドの上に簡潔な説明を表示
- マニュアルがなくても、印刷したソースコードが読める

The screenshot shows the Guript IDE interface. The main window displays a command menu with the following options and descriptions:

使用パッケージ:  
標準/ファイル処理  
標準/テーブル処理

プログラム:  
// 指定したフォルダ内の各ファイル・フォルダの詳細情報(名前、更新日時など)を表すテーブルを作成します。  
1. **ファイル情報一覧を得る** フォルダ名:  参照 逆順に  
並べる:   
// テーブルの中から、指定された2列だけを抜き出します。  
2. **列を2つ選択する** フィールド名1:  フィールド名2:   
// テーブルの行を、キーに指定されたフィールドの大きさにしたがって並び替えます。  
3. **並び替える** 降順に並べる:  キーとなるフィールド名:   
// テーブルをXML形式に変換します。結果はXML形式の文字列です。  
4. **XMLに変換する**  
// 結果を指定した方法で表示します。  
5. **結果を表示する** 出力先:

The right sidebar shows a menu with options like "削除", "print フラグ on/off", and "ファイル情報一覧を得る" (highlighted). Below it, a "システムデバッグ用" section shows a terminal output with commands like "eval" and "alert(HTML)".

指定したフォルダ内の各ファイル・フォルダの詳細情報(名前、更新日時など)を表すテーブルを作成します。

# 表示オプション: 型エラーメッセージ

- 種々のエラーを編集集中に随時表示
- エラーがあっても実行可能

The screenshot shows the Guript IDE window. The main editor contains the following script:

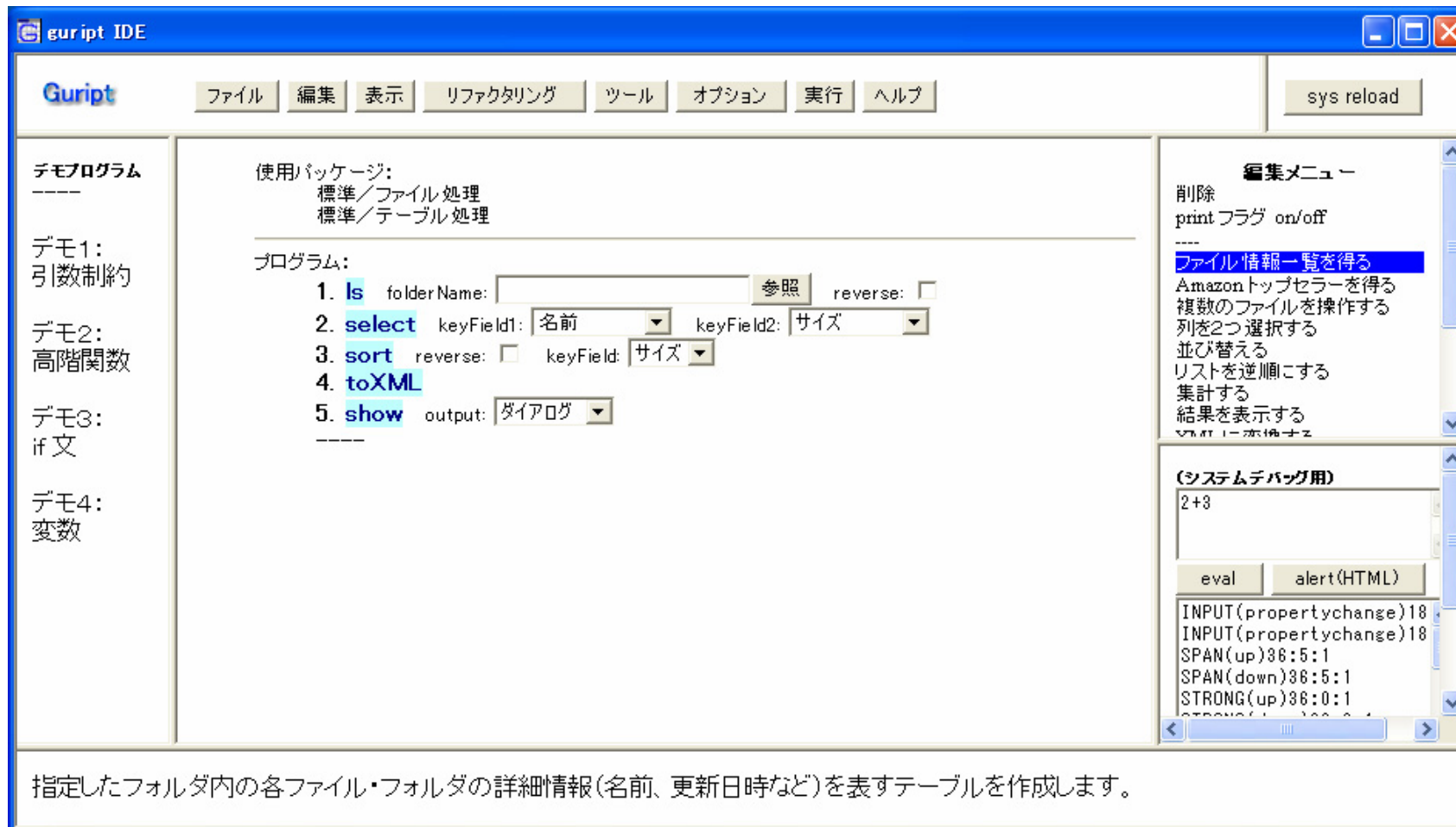
```
使用パッケージ:  
標準/ファイル処理  
標準/テーブル処理  
  
プログラム:  
// このコマンドの型: ["list", "bool"]  
1. 複数のファイルを操作する フォルダ名: C:\* 参照 サ  
  プフォルダ以下のファイルも対象とする:   
  各ファイルに対する操作の内容:  
  // 問題があります: このコマンドの操作対象は型 "unit" であるべきですが、直前のコマンドの型は "string" です。  
  // このコマンドの型: "bool"  
  1.1. 確認ダイアログを表示 メッセージ: よろしいですか?  
  // このコマンド列の型: "bool"  
  -----  
  // このコマンドの型: "unit"  
  2. 結果を表示する 出力先:    
  // このコマンド列の型: "unit"  
-----
```

The right sidebar shows a list of actions, with "確認ダイアログを表示" (Show confirmation dialog) highlighted. Below it, the system debug console shows the output of the script, including the error message and the dialog box content.

指定した文字列を使って確認ダイアログを表示する条件式です。

# 表示オプション: 簡潔表示モード

- 上級者向け
- 慣れればキーボードで直接コマンドを打てる(未実装)



The screenshot shows the Guript IDE interface. The main window has a menu bar with 'ファイル', '編集', '表示', 'リファクタリング', 'ツール', 'オプション', '実行', and 'ヘルプ'. Below the menu bar is a 'sys reload' button. The main area is divided into three sections:

- デモプログラム (Demo Programs):** A list of demo programs on the left side, including 'デモ1: 引数制約', 'デモ2: 高階関数', 'デモ3: if文', and 'デモ4: 変数'.
- 使用パッケージ (Used Packages):** A section showing '標準/ファイル処理' and '標準/テーブル処理'.
- プログラム (Program):** A list of commands with their configurations:
  1. `ls` folderName: [input field] 参照 reverse:
  2. `select` keyField1: [名前] keyField2: [サイズ]
  3. `sort` reverse:  keyField: [サイズ]
  4. `toXML`
  5. `show` output: [ダイアログ]

On the right side, there is a '編集メニュー' (Edit Menu) with various actions like '削除', 'print フラグ on/off', and 'ファイル情報一覧を得る'. Below that is a '(システムデバッグ用)' (System Debug) section with a text area showing '2+3' and buttons for 'eval' and 'alert(HTML)'. The bottom of the window contains a footer text: '指定したフォルダ内の各ファイル・フォルダの詳細情報(名前、更新日時など)を表すテーブルを作成します。'

# 実装

- 現在のところ IE6 上の JavaScript で記述 (約1800行)
  - LiveConnect で Java と連携可能
    - Webサービス、データベースなどへのアクセス
- Windows HTA(HTML application) として起動
  - ActiveX が利用可能
    - Excel, IE, Outlook などの自動制御
- guript コマンドの追加は容易

# 応用の可能性

- 一般のパソコンユーザの作業の自動化
  - ファイル整理、メール自動処理、フォーム自動入力、Webサイトの自動更新
  - Webサービス
- エンドユーザコンピューティング
  - CMS、グループウェア、ワークフローシステムなどの動作設定
- 情報家電
  - 膨大な録画データからの情報検索
  - エアコン、ロボット掃除機などの動作設定
- 教育用



# 今後の研究課題

- 実用的ライブラリの充実
  - Webサービス、表処理、XML処理、...
- 引数制約のフレームワーク確立
  - 型推論、mediator pattern などを検討
  - 入力の快適さと部品のリ利用性とのトレードオフ
- サードパーティによるコマンド作成の支援
  - 型チェックルーチンの定義支援等
- 複雑な式も快適に編集可能な構造エディタ
  - パラメタに定数リテラル以外の式を書けるように

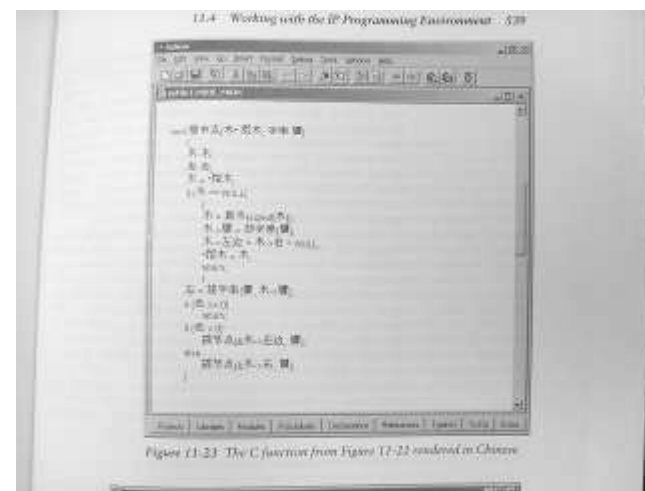
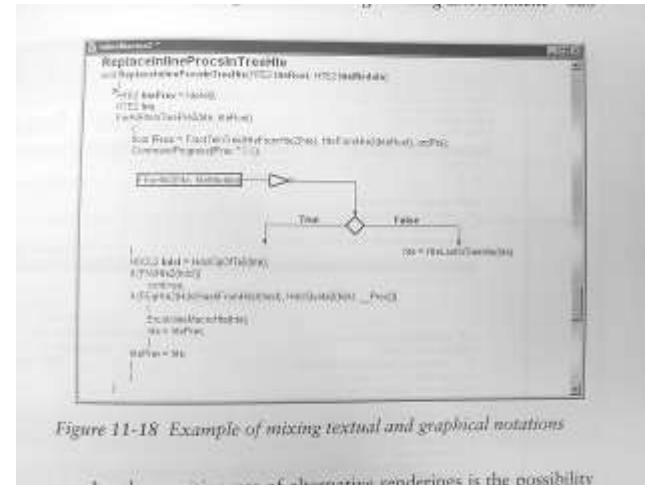
# 関連研究1:SqueakToys (eToys)

- Alan Kay
- 教育用言語
  - ゲームなどが簡単に作れる
  - 構造エディタ
  - 日本語化されている



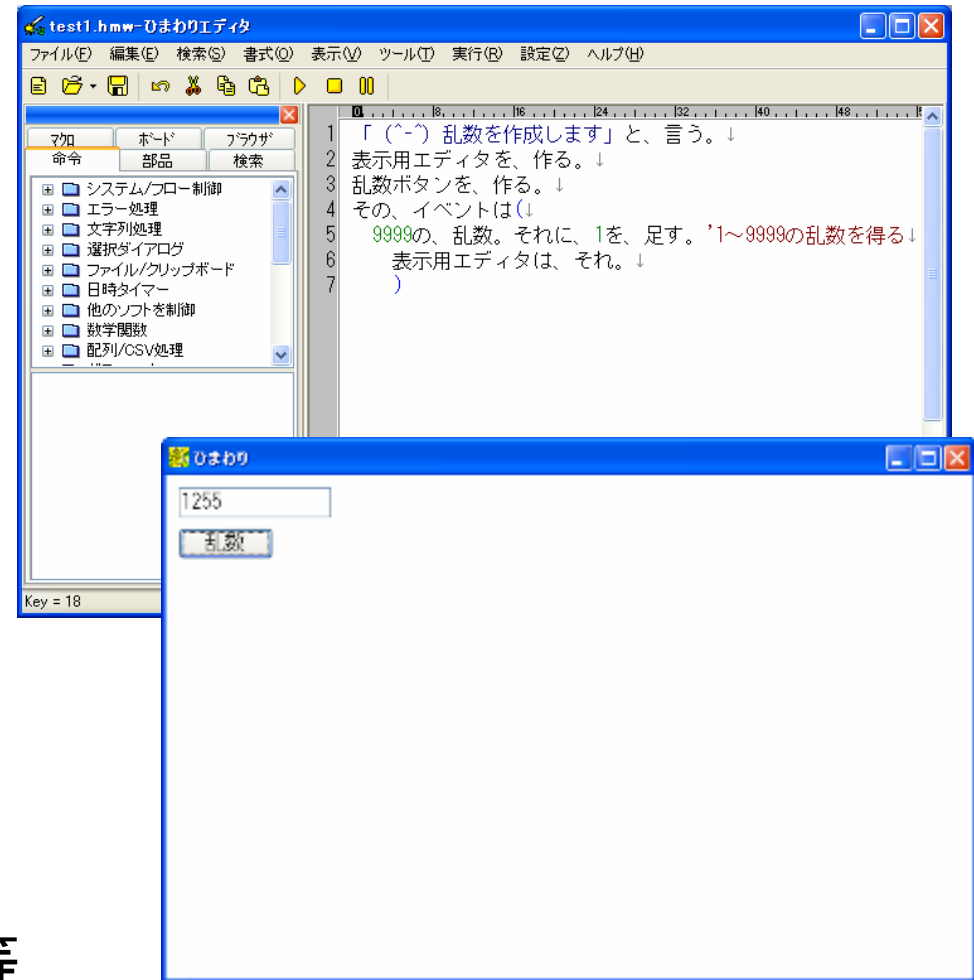
# 関連研究2: Intentional Programming

- **Charles Simonyi**  
(Microsoft Research)
  - "The WYSIWYG"
- representation と notation の分離
- 構造エディタ
- 問題領域に適した表示
- 公開されていないため、使い勝手は不明



# 関連研究3: 日本語プログラミング

- ひまわり、TTSneo
  - フリーソフト
  - とっつきやすい
  - わかりやすいマニュアル
  - すぐれた開発環境
  - 実用的なライブラリ
- 他に、Mind、言霊、旧AppleScript など
- 入力に多少難あり
  - 語順、送り仮名、句読点等に関する自明でない規則



ひまわりエディタと  
プログラム実行例

# 構造エディタの可能性

- テキストベース言語の限界
  - block 引数など、構文解析に苦勞(ruby, groovy)
  - HTML、XML リテラルは  
テキストエディタで扱いづらい

```
html {
  head {
    title("XML encoding with Groovy")
  }
  body {
    h1("XML encoding with Groovy")
    p("this format can be used as an alternative markup to XML")
    a(href:' http://groovy.codehaus.org') ["Groovy"]
    p [
      "This is some", b("mixed"),
      "text. For more see the",
      a(href:' http://groovy.codehaus.org') ["Groovy"],
      "project"
    ]
    p("some text")
  }
}
```

groovy による HTML の記法

- すでに汎用エディタより IDE が主流に
  - コード補完、リファクタリング、...
- Microsoft と Intentional Software の動向

# かつての構造エディタの問題点

- 1970s～1980sの構造エディタはなぜ使いにくかったのか？
- 単に作りこみが甘い
  - キー割り当てや編集機能に、emacs/vi ほどのこだわり・工夫が見られる構造エディタがなかった
- 既存のテキストベース言語を扱っていた
  - テキストエディタに最適化された文法は、構造エディタで扱いにくいのは当然
- 四則演算など式の入力・編集に困難
  - 最もチャレンジングな問題、現在取り組み中