

---

# ライブ UI プロトタイピング に向けたマルチ言語環境 SOMETHINGit

---

小田朋宏

(株)SRA

中小路久美代

(株)SRA

山本恭裕

東京工業大学

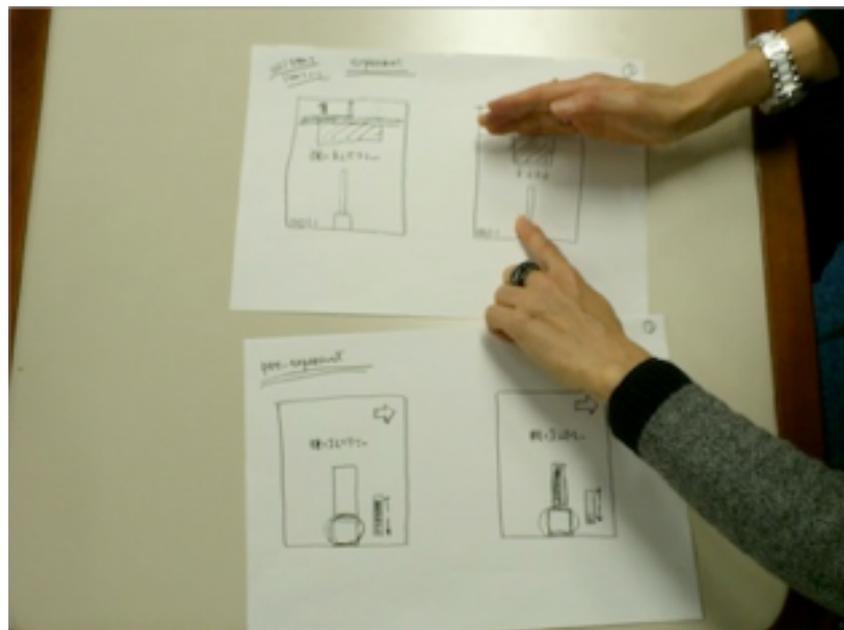
---

# UI design Lightweight Formal Method

# 動機

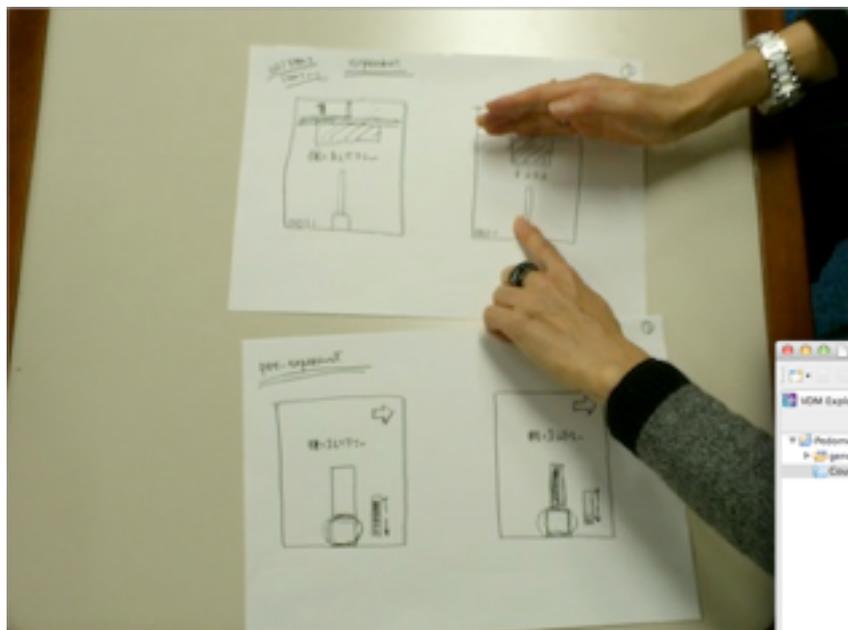
---

- 軽量形式手法をより多くの人を手軽に使える環境を作りたい
  - 例えば、UIデザイナー



# 軽量形式手法とUIデザイン

## それぞれのプロトタイピングの特徴

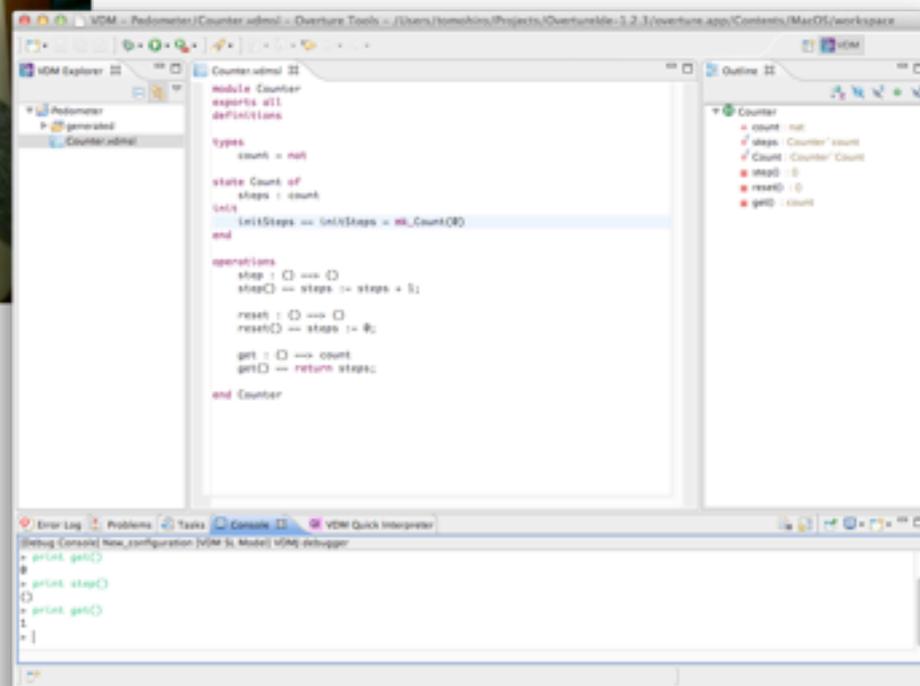


### UIスケッチ

- 具体性
- ニュアンス

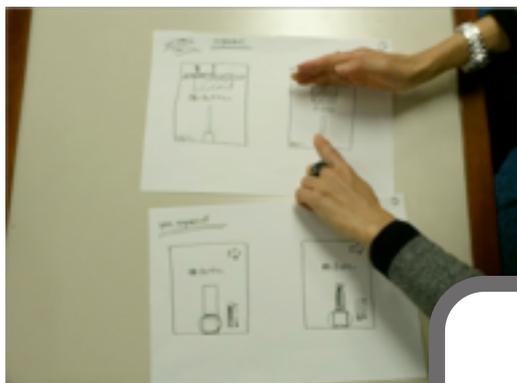
### 実行可能仕様

- 実行可能性
- 厳密性

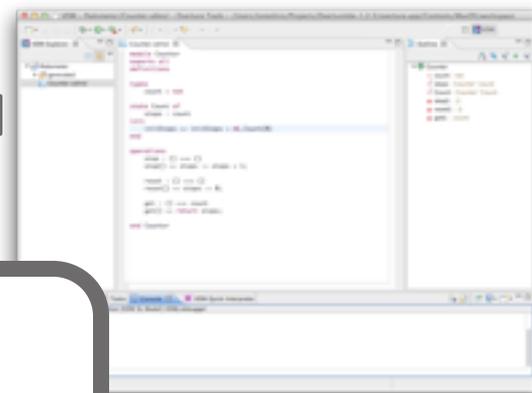


# 軽量形式手法+UIプロトタイピング

UIデザイナー



形式手法技術者



実行可能な  
UIプロトタイプ

デザインデシジ  
ョンの合意

# めざすところ

---

- 「機能モデル的に実現可能なUIデザイン」  
かつ、  
「適切なUIを構築可能な機能モデル」  
ができるようなプロトタイプ環境を作りたい

# 本研究開発のアプローチ

---

UIをデザインすることと機能をモデリングすることを  
共同デザインとして捉える



UIデザイナーと形式手法技術者間の「対話」のデザイン



プロトタイピング環境構築に必要な  
ライブラリの実装



UIプロトタイピング環境の構築

---

LIVE

13年7月9日火曜日

---

# 合意形成のためのプロトタイピング

---

- プロトタイプの実現とビュー双方について
  - 仮説的状況を実現して試行できること
  - 何が起きているのか理解できること
  - 手に取って指し示して議論できること

そこで

- プロトタイプを定義する記述環境
- プロトタイプを使う実行環境

を同一にする = ライブプロトタイピング環境

---

SOMETHINGit

# SOMETHINGit

---

- Smalltalk (Squeak/Pharo) 上のライブラリ
  - 外部インタプリタを呼び出す
    - VDM-SL (VDMJ)
    - Haskell (GHCi)
- ライブプロトタイピングを指向
  - VDM-SLでの  
Smalltalk流のライブな  
プロトタイピングを可能にする

# SOMETHINGitによる

## Smalltalk流プログラミング環境の実現

---

- 情報隠蔽を破るバックドア
- 実行時改変と継続実行
- 自由度の高いFFI

---

# Backdoors

# 情報隠蔽を破るバックドア

---

VDM-SLモジュールのstate内の変数はprivate

- 外部から値を読んだり書き込むためには operation を通す必要がある
- そこでstate内の全変数に一括して読み書きするpublicなアクセサを自動生成する
  - setter
  - getter

# 安全なバックドア

---

VDM-SL仕様中ではバックドアを利用できない

1. バックドア無しの仕様をVDMJにロードする
2. コンパイルエラーが発生しなければ
  - a. バックドアを自動生成し
  - b. バックドア付きの仕様を再ロードする

# 例：state変数の値を直接操作する

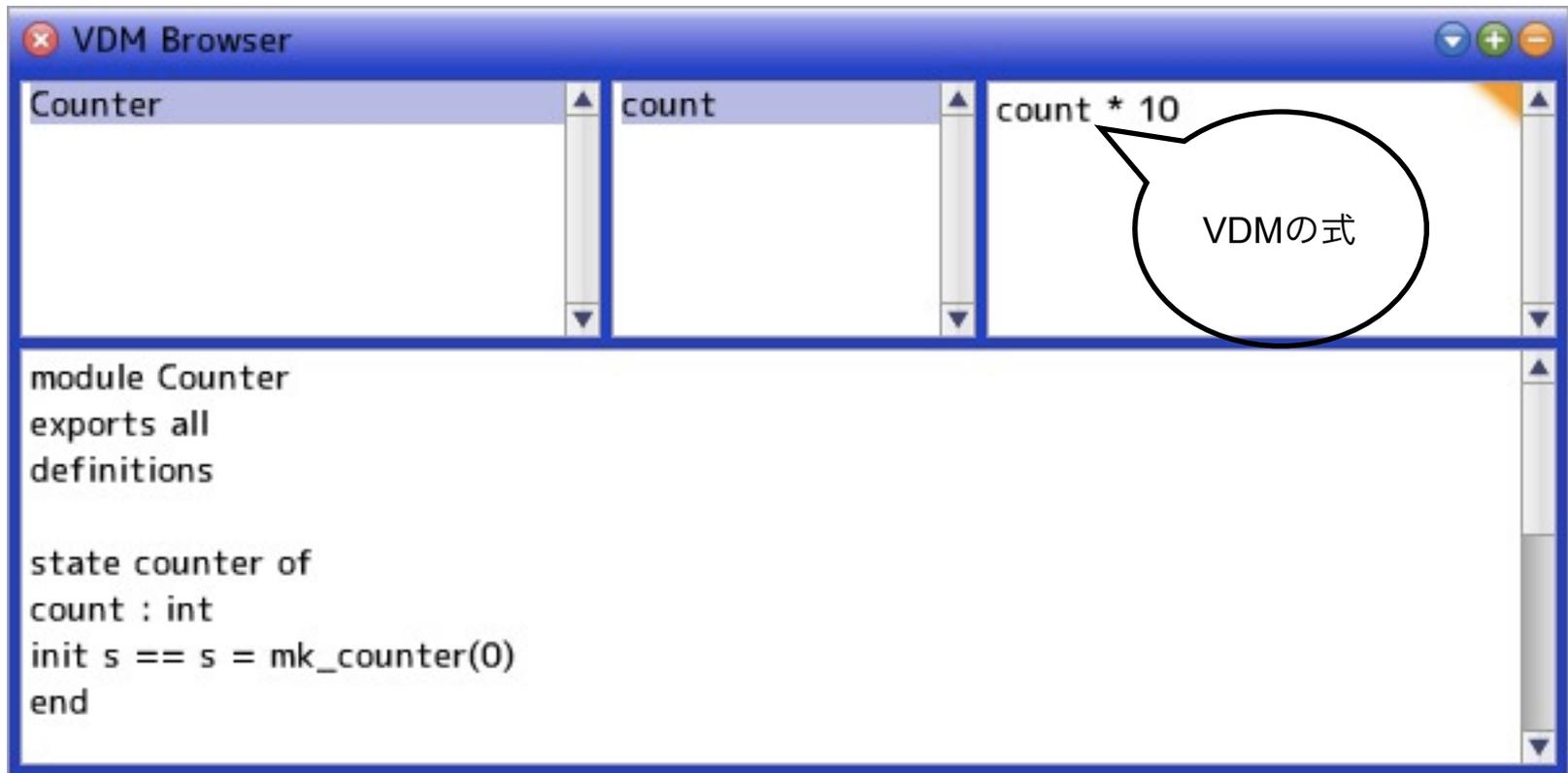
The screenshot shows the VDM Browser interface. The top section displays a list of modules with 'Counter' selected. Below this, a list of variables for the selected module shows 'count' with a value of 14. The bottom section displays the source code for the 'Counter' module, which includes a state declaration for 'count' and an initialization function 'mk\_counter(0)'. Callouts in Japanese identify these elements: 'モジュールリスト' (Module List) points to the 'Counter' entry; '変数リスト' (Variable List) points to the 'count' variable; '値' (Value) points to the number '14'; and 'モジュールの仕様記述' (Module Specification) points to the source code block.

```
module Counter
exports all
definitions

state counter of
count : int
init s == s = mk_counter(0)
end
```

# count \* 10を計算する

---



# count \* 10 の答えは140

The screenshot shows a window titled "VDM Browser" with a table of variables and expressions. The table has three columns: "Counter", "count", and "count\*10". The value for "count" is 14, and the value for "count\*10" is 140. A callout bubble points to the value 140 with the text "評価結果". Below the table, the source code for the "Counter" module is displayed.

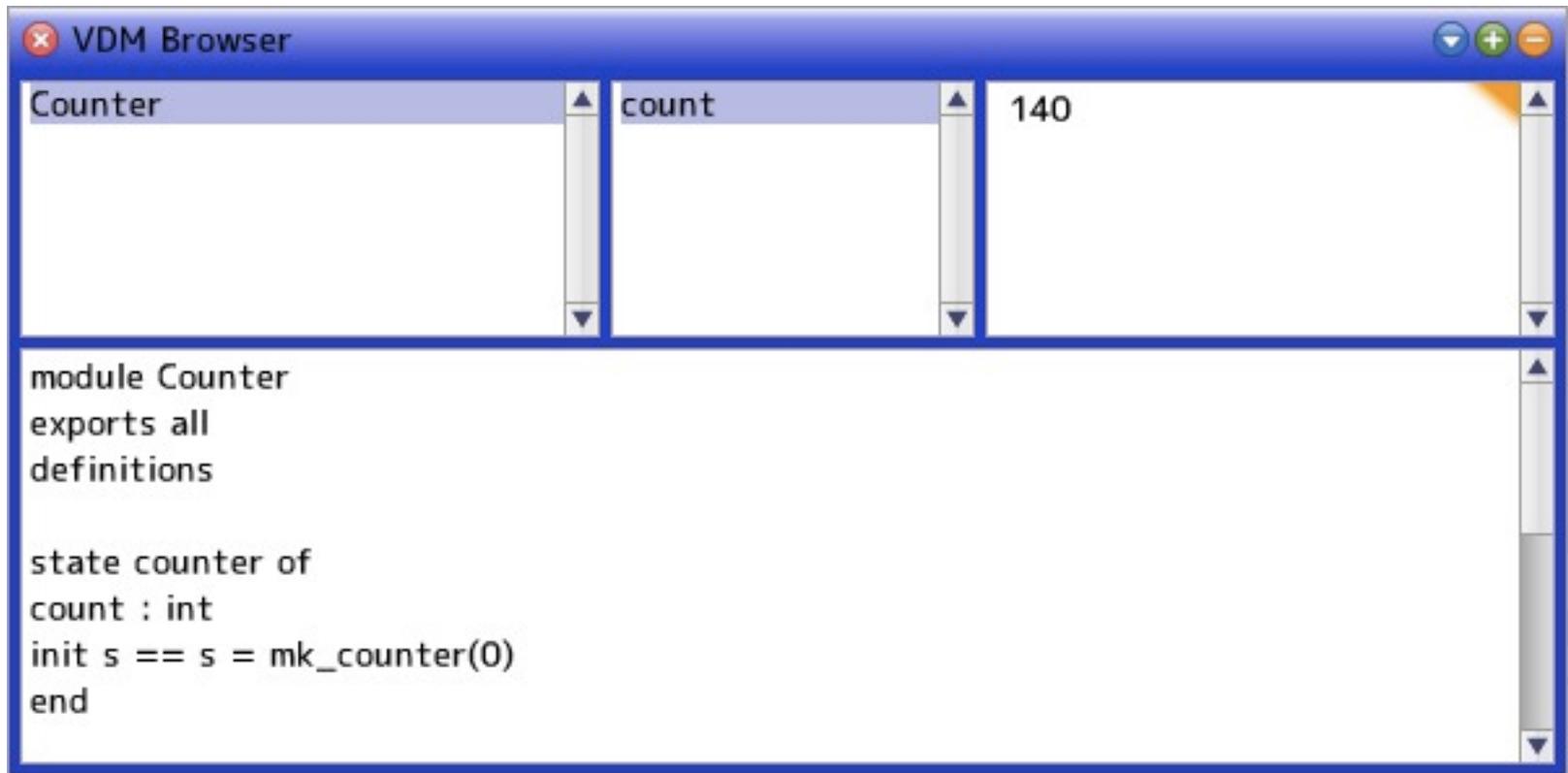
Counter	count	count*10
	14	140

```
module Counter
exports all
definitions

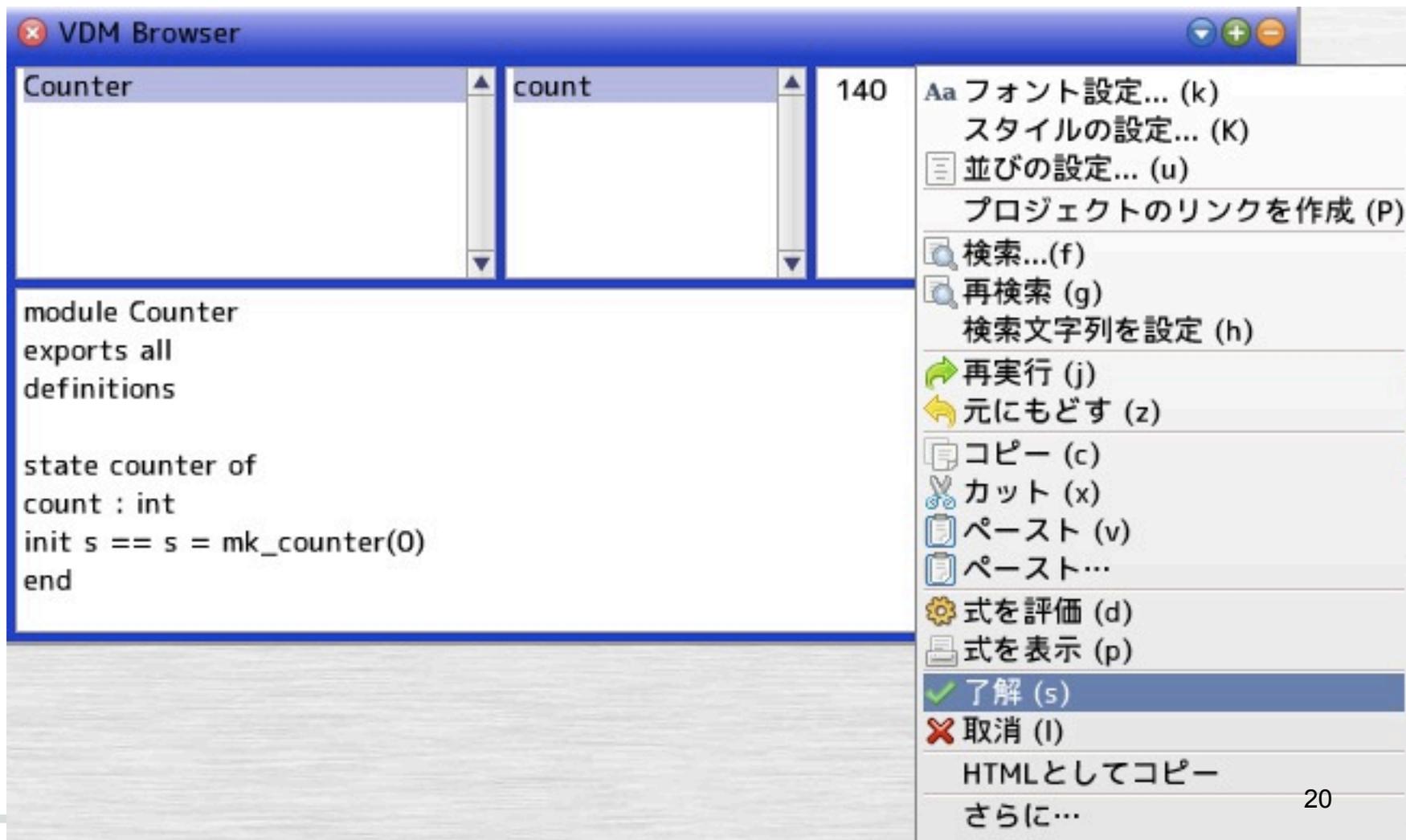
state counter of
count : int
init s == s = mk_counter(0)
end
```

# 140を

---



# 140を変数countの値に設定する



The screenshot shows the VDM Browser interface. The top part displays a table with two columns: 'Counter' and 'count'. The 'count' column has the value '140'. Below the table, the code for the 'Counter' module is visible:

```
module Counter
exports all
definitions

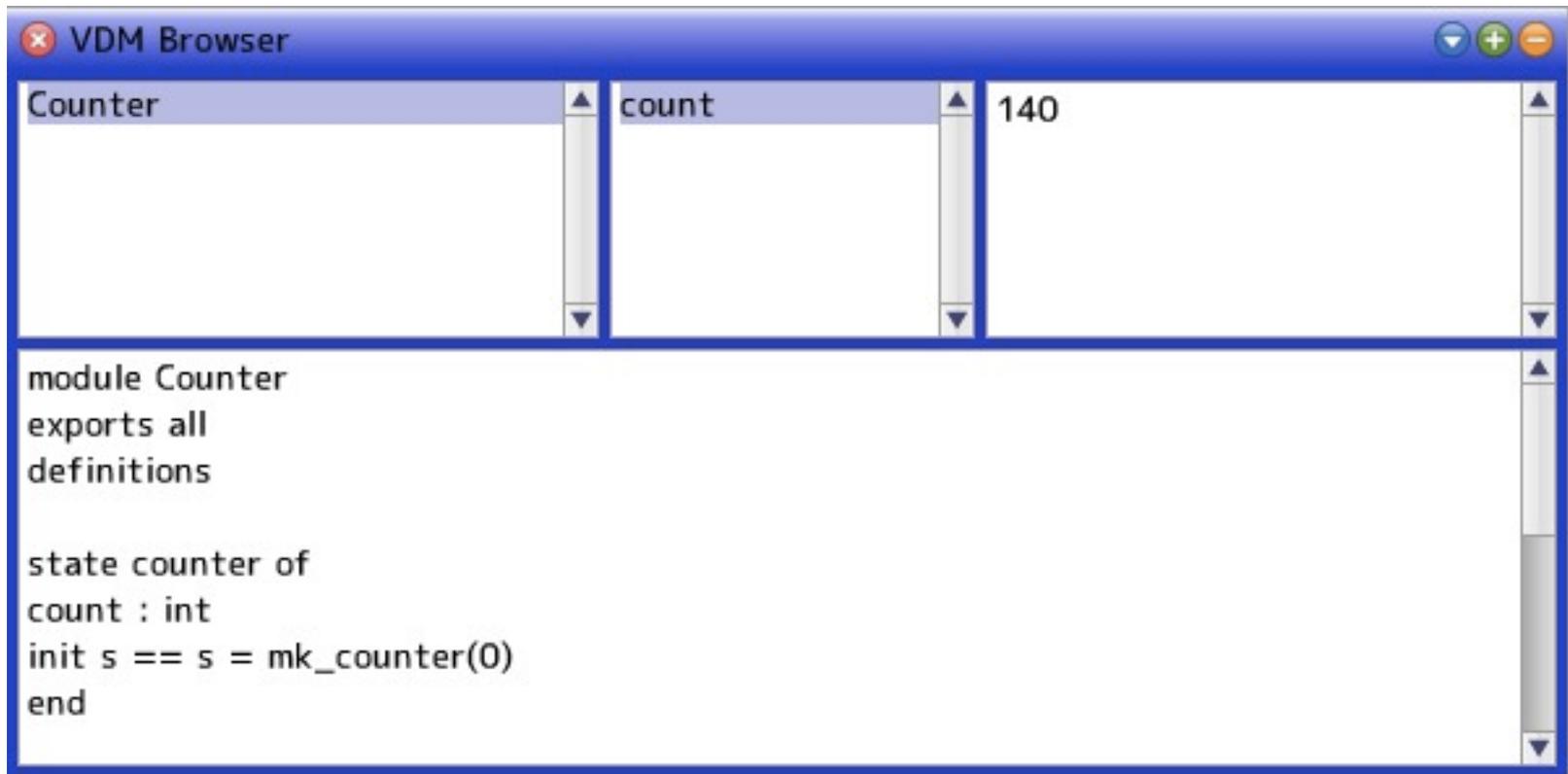
state counter of
count : int
init s == s = mk_counter(0)
end
```

A context menu is open on the right side of the window, listing various actions. The '了解 (s)' (Help) option is currently selected.

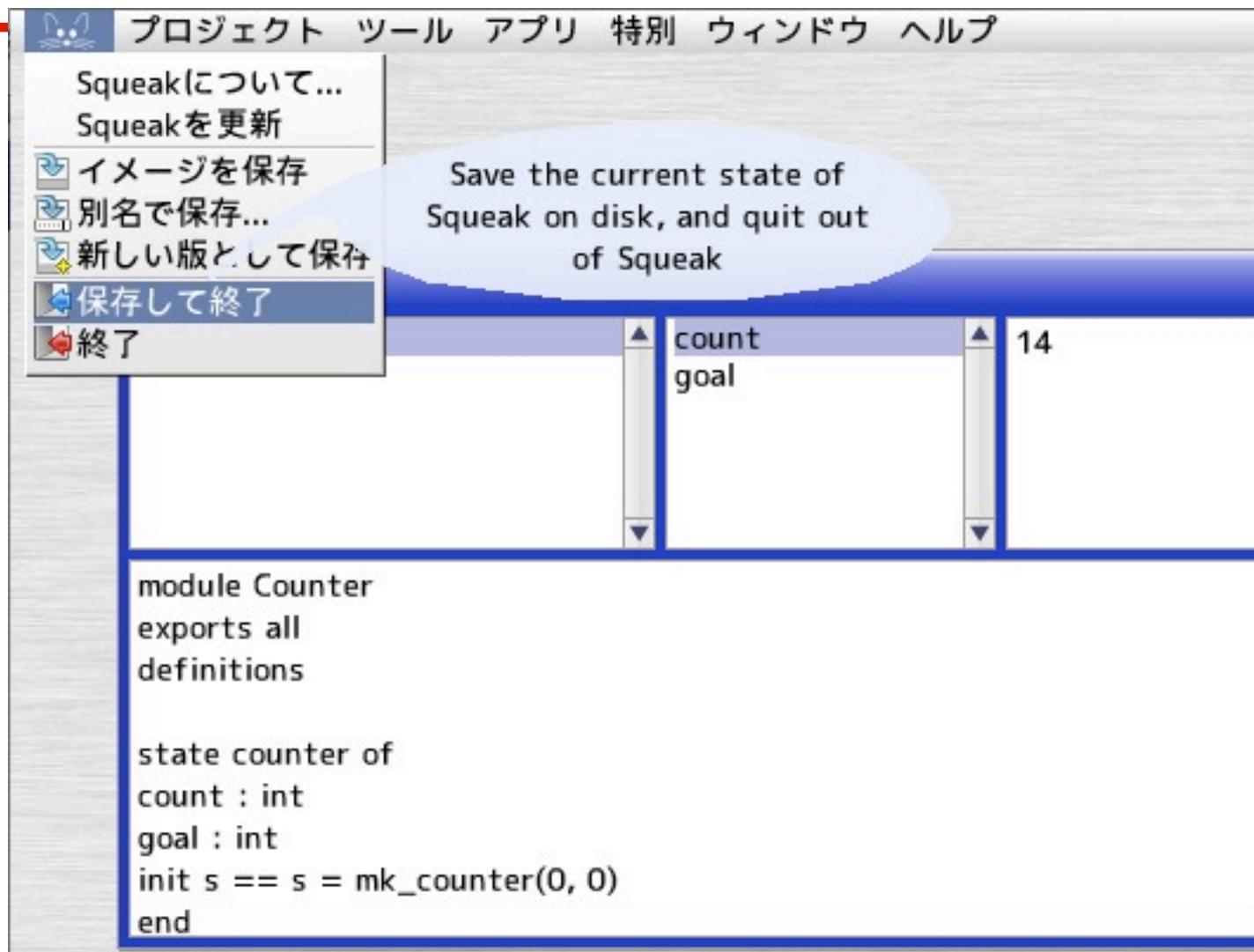
- Aa フォント設定... (k)
- スタイルの設定... (K)
- 並びの設定... (u)
- プロジェクトのリンクを作成 (P)
- 検索...(f)
- 再検索 (g)
- 検索文字列を設定 (h)
- 再実行 (j)
- 元にもどす (z)
- コピー (c)
- カット (x)
- ペースト (v)
- ペースト...
- 式を評価 (d)
- 式を表示 (p)
- ✓ 了解 (s)
- ✗ 取消 (l)
- HTMLとしてコピー
- さらに...

# 変数countの値が140になる

---



# 開発環境を「保存して終了」する



一旦、開発環境を終了した後で

---



# もう一度開発環境を開くと 終了前の状態が回復する



プロジェクト ツール アプリ 特別 ウィンドウ ヘルプ

Counter	count	goal	14

```
module Counter
exports all
definitions

state counter of
count : int
goal : int
init s == s = mk_counter(0, 0)
end
```

値がリセット  
されていない

---

# Liveness

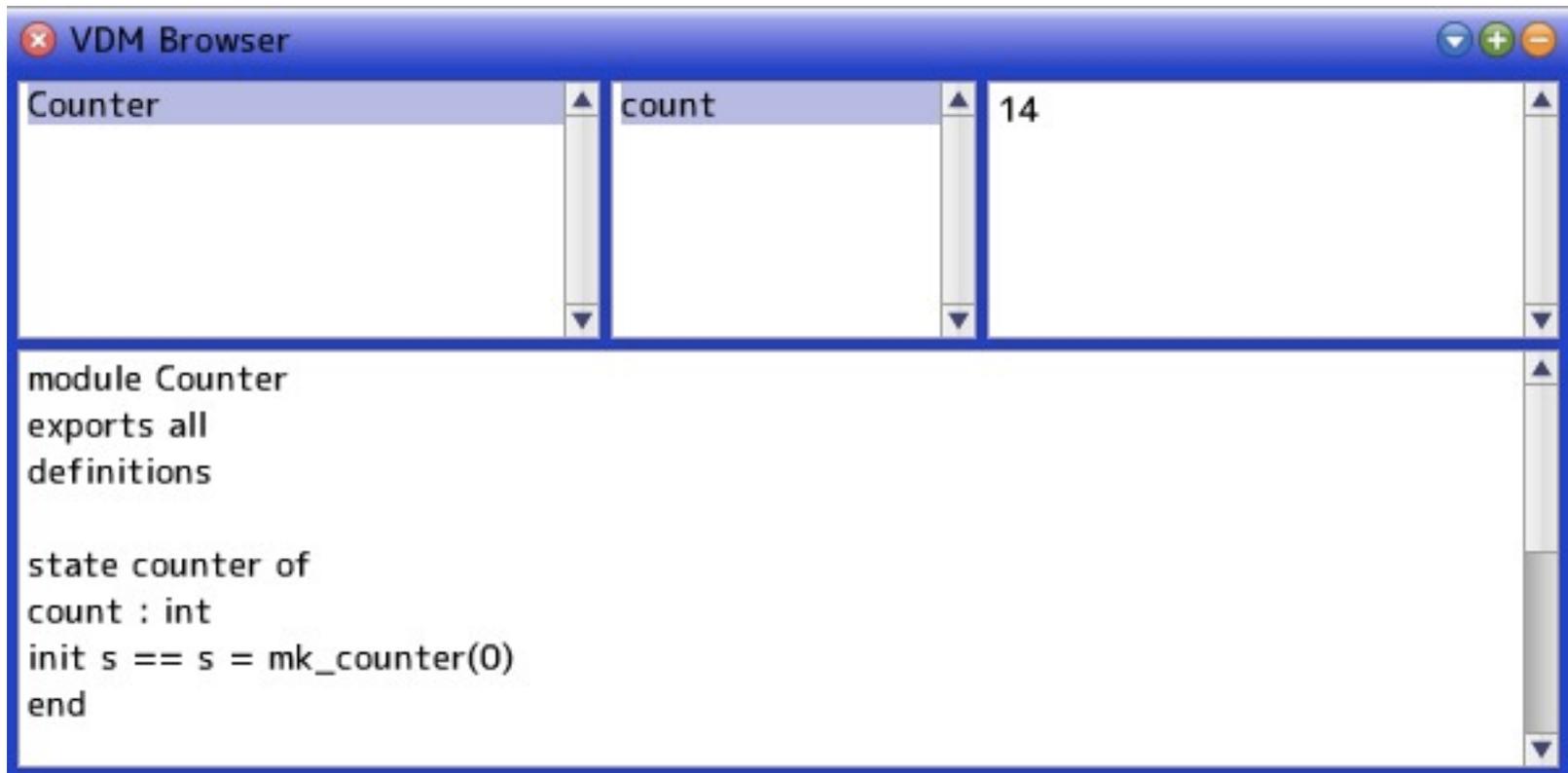
# ライブプログラミング

---

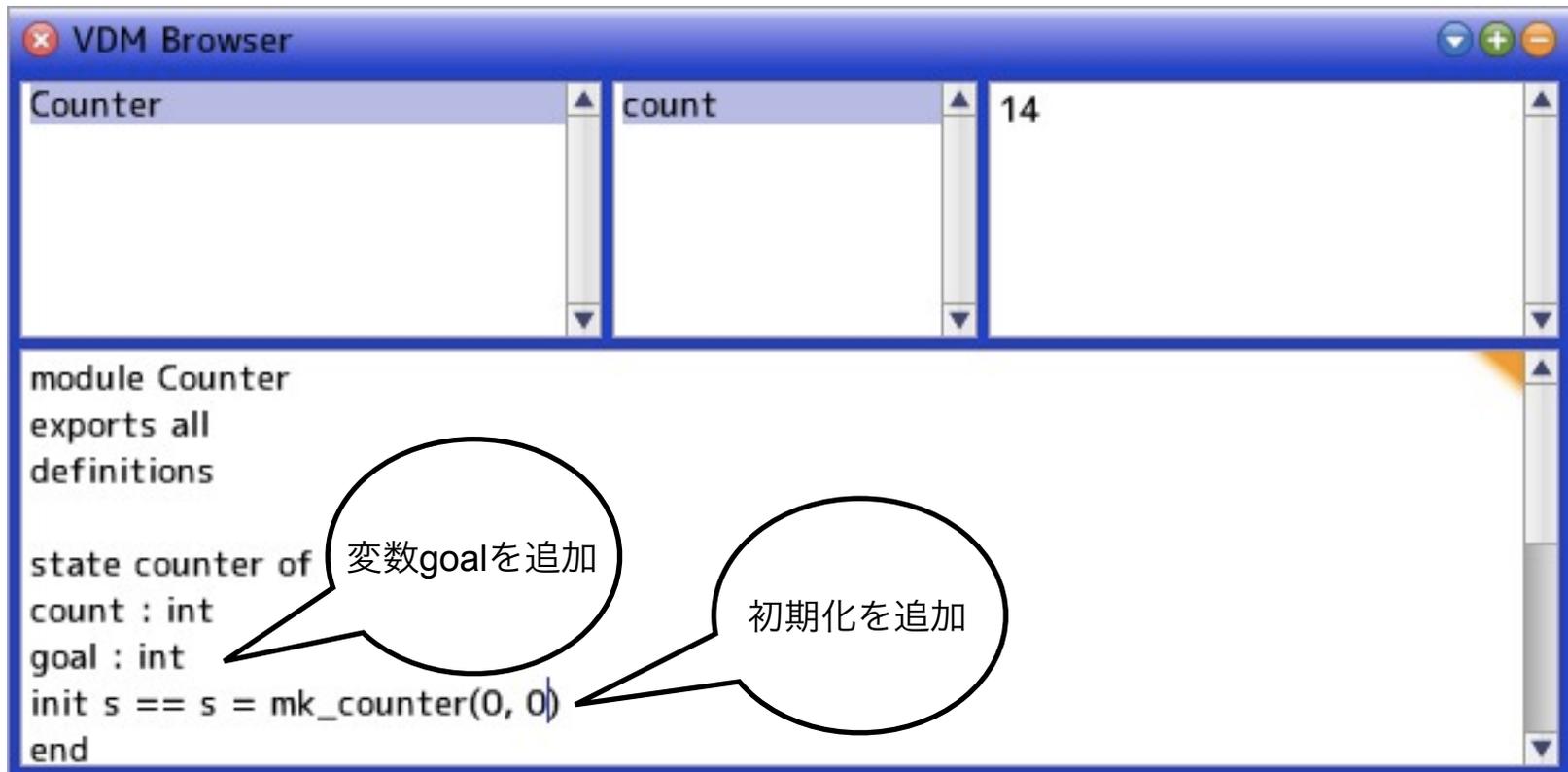
実行中のプログラムのソースコードを  
変更したら即座にその実行に反映される  
プログラミング  
= 脱 Edit-Compile-Run サイクル

# 例：プロトタイプ実行中の仕様変更

---



# state変数としてgoal : intを追加



The screenshot shows the VDM Browser interface. At the top, a table displays the state of the Counter module:

Counter	count	14

Below the table, the module definition is shown:

```
module Counter
exports all
definitions

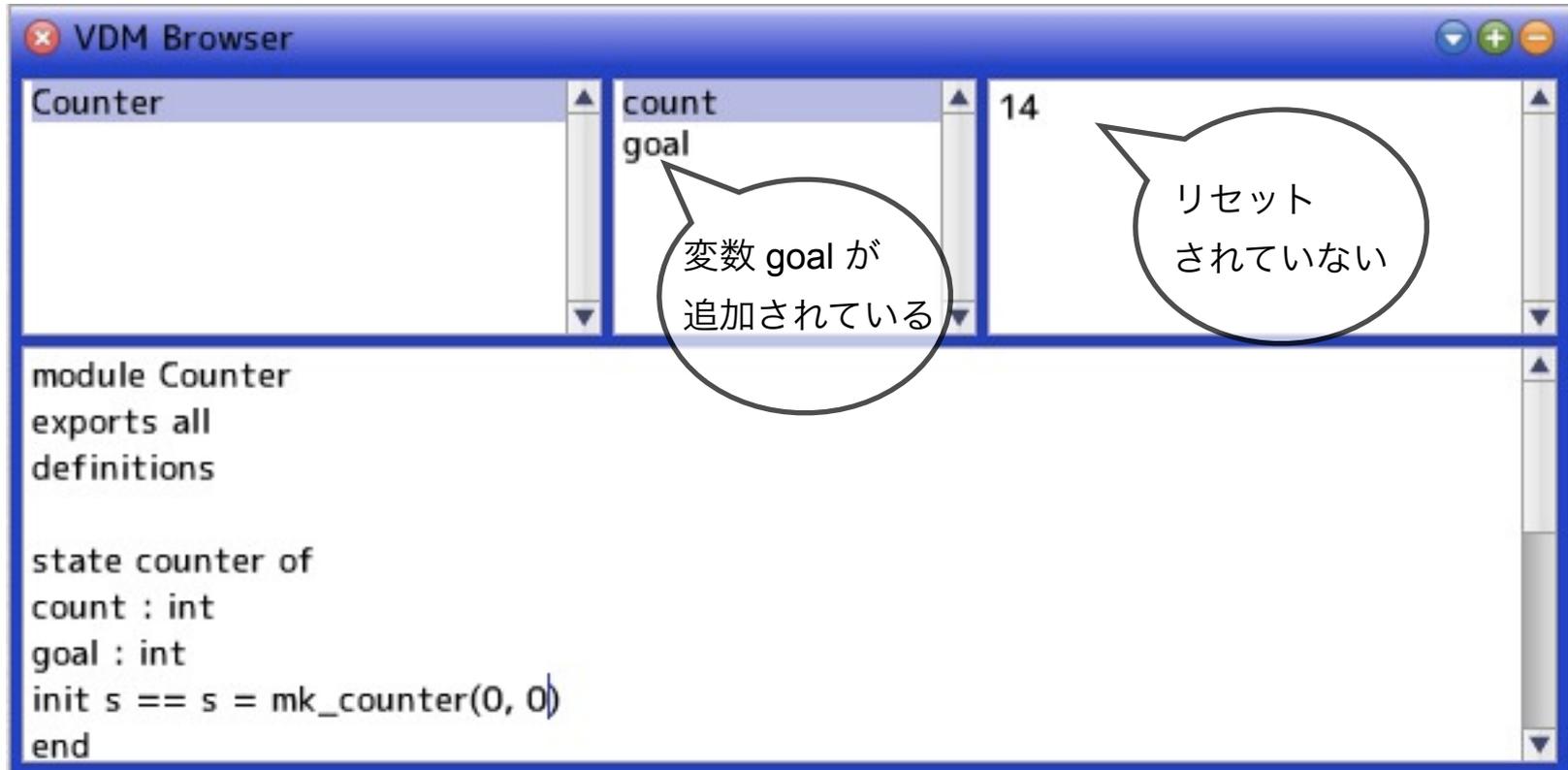
state counter of
count : int
goal : int
init s == s = mk_counter(0, 0)
end
```

Two callout bubbles provide annotations:

- A bubble pointing to `goal : int` contains the text "変数goalを追加" (Add variable goal).
- A bubble pointing to `mk_counter(0, 0)` contains the text "初期化を追加" (Add initialization).

# 変数goalを追加しても

元の変数の値は維持されている



VDM Browser

Counter	count	goal	14

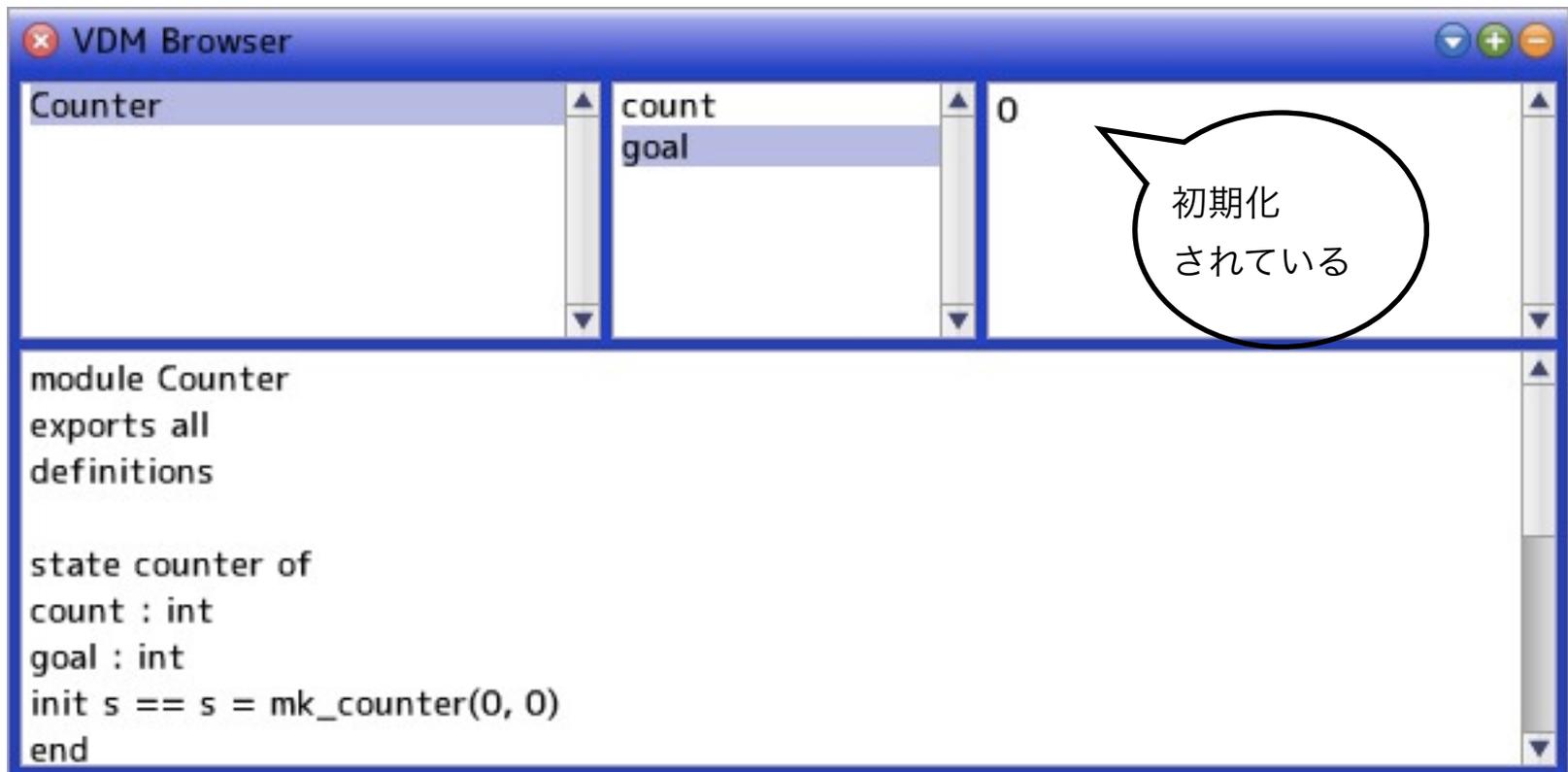
変数 goal が追加されている

リセットされていない

```
module Counter
exports all
definitions

state counter of
count : int
goal : int
init s == s = mk_counter(0, 0)
end
```

# 変数goalの値はinitの定義通り0



VDM Browser

Counter	count	0
	goal	

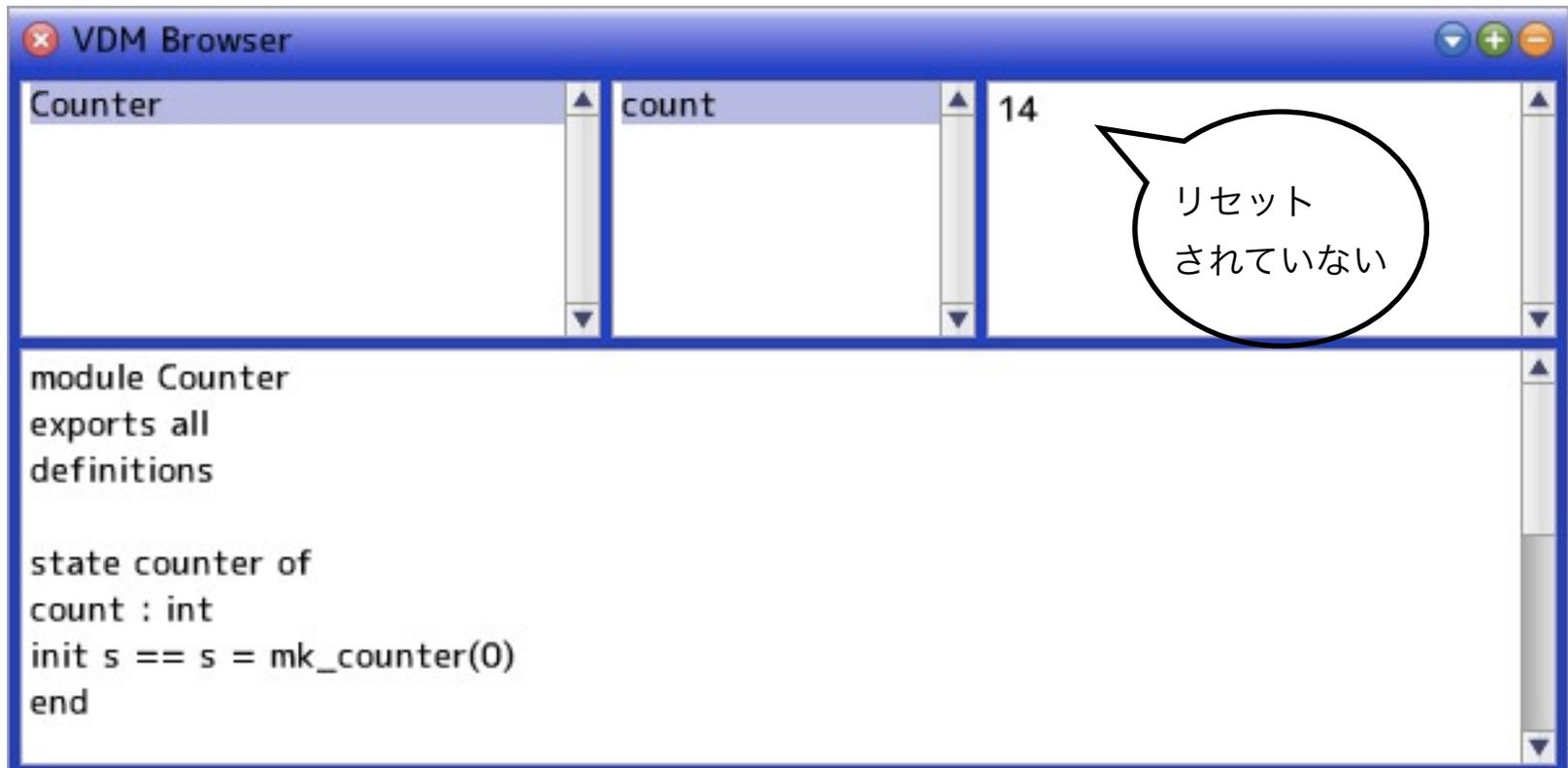
初期化  
されている

```
module Counter
exports all
definitions

state counter of
count : int
goal : int
init s == s = mk_counter(0, 0)
end
```

# 変数goalを削除しても

# 変数countの値は維持される



VDM Browser

Counter	count	14
---------	-------	----

リセット  
されていない

```
module Counter
exports all
definitions

state counter of
count : int
init s == s = mk_counter(0)
end
```

# 実行時改変と継続実行のしくみ

---

1. バックドアを使って全モジュールのstateを読み出す
2. 変更後の仕様をロードする
3. 変更後の仕様のバックドアを生成する
4. バックドア付きの仕様をロードする
5. バックドアを使って、全モジュールのstateに書き込む

---

# Expression-based Foreign Function Interface

# 自由度の高い言語間インターフェイス

---

## 通常の言語間インターフェイス(FFI)

- 関数名、引数の数と型、返り値の型  
を前もって公開する

## SOMETHINGitの表現式ベースのFFI

- 外部DSLとして任意の表現式を埋め込むことが可能
- ホスト言語側でゲスト言語の表現式を合成することで、単なる関数呼び出し以上の表現が可能

# 例：関数double

---

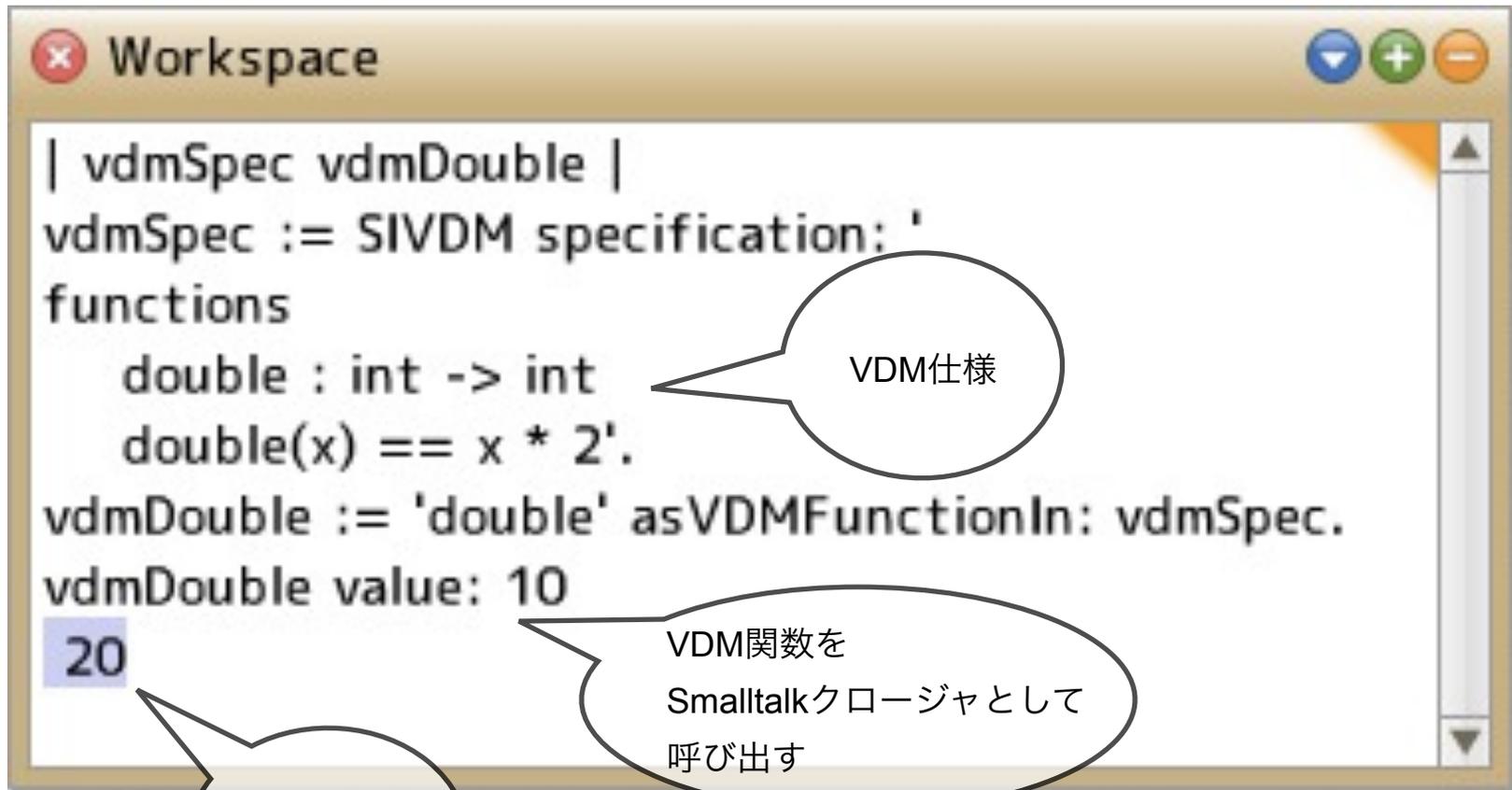
functions

double : int -> int

double(x) == x \* 2

引数の2倍を返す関数

# 例：double(10)をSmalltalkで評価する



```
Workspace  
| vdmSpec vdmDouble |  
vdmSpec := SIVDM specification: '  
functions  
  double : int -> int  
  double(x) == x * 2'.  
vdmDouble := 'double' asVDMFunctionIn: vdmSpec.  
vdmDouble value: 10  
20
```

VDM仕様

VDM関数を  
Smalltalkクロージャとして  
呼び出す

$10 * 2 = 20$

# 例：Smalltalkのクロージャとしての VDM double関数

Smalltalkのクロージャ

```
(1 to: 10) collect: [:x | x * 2]  
#(2 4 6 8 10 12 14 16 18 20)
```

VDM仕様

```
| vdmSpec vdmDouble |  
vdmSpec := SIVDM specification: '  
functions  
  double : int -> int  
  double(x) == x * 2'.  
vdmDouble := 'double' asVDMFunctionIn: vdmSpec.  
(1 to: 10) collect: vdmDouble  
#(2 4 6 8 10 12 14 16 18 20)
```

VDM関数

# 応用：ライブUIプロトタイピング環境

## Lively Walk-Through

The screenshot shows the Squeak4.3-ja.image environment with several windows and callouts:

- VDM Browser:** Displays a VDM specification for a counter. The state is shown as `count : int` with a value of `9`. The operations include `get : () ==> int`.
- Cut Browser:** Shows a sequence of events and states. The event log includes:
  - 15 January 2013 6:31 pm : EVENT : [リセット]
  - 15 January 2013 6:31 pm : State : {Counter: {
  - 15 January 2013 6:31 pm : VDM : Counter'get
  - 15 January 2013 6:31 pm : VDM : Counter'reset

- Sketching Tool:** A vertical toolbar on the right side of the sketching area, containing various drawing tools like pens, eraser, and selection tools.
- UI Sketch Image:** A hand-drawn sketch of a mobile device interface, outlined in red and blue, with a green circle on the screen.
- Event Log:** A callout pointing to the event log in the Cut Browser, labeled "イベントログ".
- UI Sketch Image:** A callout pointing to the hand-drawn sketch, labeled "UIのスケッチ画像".
- Sketching Tool:** A callout pointing to the drawing toolbar, labeled "スケッチングツール".
- GUI Components:** A callout pointing to the red and blue outline of the mobile device, labeled "GUI部品".
- Agreement Description:** A callout pointing to the text in the Cut Browser, labeled "合意事項の記述".

---

# Conclusion

# まとめ

---

- 軽量形式手法とUIデザイン
- UIデザイナーと形式仕様技術者の対話としてのライブUIプロトタイピング
- VDM-SLでライブプログラミング
  - 情報隠蔽を破るバックドア
  - 実行時改変と継続実行
  - 自由度の高いFFI

# 今後の課題

---

- さらに
    - 生き生きとした
    - 技術者やデザイナーの間で会話が進む
    - 多様なUI形態のデザインに使える  
(タッチデバイス、ウェブUI、Kinect等)
    - 使っていて楽しい
- 環境をめざします

---

Thank you.