A perception/action substrate for cognitive modeling in HCI

ROBERT ST AMANT AND MARK O. RIEDL
Int. J. Human-Computer Studies (2001) 55, 15-39

1. Introduction

認知モデル研究と HCI (Human-Computer Interaction)研究の関係
　認知モデルから HCI 研究への貢献
　　ユーザの行動を予測
　HCI 研究から認知モデル研究への貢献
　　現実的な課題(アプリケーションの操作)をモデルへ
　　GUI を介してモデルが課題を遂行
だが，モデルとユーザインタフェースのダイレクトなインタラクトは可能ではない
　ファイルの媒介
　現実のインタフェースのシミュレータを使用
　ユーザインタフェース・マネージメントシステムの媒介
ダイレクトなモデルとユーザインタフェースのインタラクトは以下の観点から重要

1. 生態学的妥当性
   インタフェースのシミュレータは人間が本当に使うものとは異なる
   タイミングの変化，予測性の変化，属性の信頼性などは捨象
2. 現実の問題との関連性
   認知モデルの目的は一般的な問題解決のプロセスへの知見を得ること
   課題に特化したモデルでは意味がない
3. 比較のための外的基準
   複数のモデルが同一の課題を解くことも重要（eg., EPIC vs. ACT-R/PM）
   モデルに特化したインタフェースを作っていたら，比較ができない
4. 開発の労力
   認知モデルに入力するインタフェースシナリオを作ることは大変

過去の著者達の試み：画像処理，マニュアル操作の認知モデルへの組み込み
　Interface softbot (Zettlemoyer & St. Amant, 1999; Zettlemoyer, St. Amant & Dulberg, 1999)
　API，ソースコードへのアクセスなしにユーザインタフェースとモデルがインタラクト
今回の論文は Windows に特化したプログラム可能な下地(Programmable substrate)
　→VisMap (Visual Manipulation)
　市販のアプリケーションとのやりとりを可能にする関数の集合

2. Indirect Interaction with the interface

VisMap 開発の背景となった研究…視覚と運動処理の HCI 研究

- LICAI (Kitajima & Polson, 1995, 1997)

GUI 上での探索課題をモデル化
ユーザのアクションによって動的に変化する環境上での探索行動
しかし，ディスプレイ上の情報は静的(述語)に表現
“Graph”とタイトルがつけられたアイコンの表象

```
(is-on-screen $object-123)
(is-a-kind-of $object-123 display-object)
(is-a-kind-of $object-123 graph-menu-item)
(is-pointed-at $object-123)
(not (is-highlighted $object-123)
(not (is-grabbed $object-123).
```

はじめ 3 つ：オブイジェクトの記述

残り 3 つ：オブジェクトの状態(オブジェクト上にマウス、クリックされていない)

アクション(条件節-実行節)　条件節，実行節ともに述語の組み合わせで表現

Move-mouse-cursor：マウスカーソルを特定の位置に移動

Single-click, Double-click：マウスのボタンをクリック

Press-and-mouse-bottom-down，Release-mouse-botton：マウス操作のよりプリミティブなアクション

Type：キーボードからの文字列の入力

LICAI ではアクションが静的な表象を変化させることが容易にできた．

だが，モデルを作成する上ではプログラミングのコストがかかる(あらかじめアクション系列を定めなければならない)

- GLEAN GOMS Language Evaluation and Analysis

GOMS(巻末)によって課題分析されたインタフェースのシミュレータを計算機に実装して評価するツール

GLEAN による視覚表象：属性(位置，ザイズ，色)

```
Visual object: Start-button
    Type is Button
    Label is Start
    Color is Red
```

GLEAN によるアクション

Keystroke：キーボードからの文字の入力

Type-in：文字列の入力

Hold-down, Release：ローレベルのキーボードイベント

Click, Double-click：マウスボタンのイベント

Point-to：マウスカーソルをオブジェクトへ移動

Home-to：キーボード上の手の動き

Look-for-object-with-property-and-store…：特定の属性値をもつオブジェクトを探索し，タグ付け

Wait-for-object-with-property-and-store…：特定の属性値をもつオブジェクトの出現をまつ

GLEAN は LICAI よりも詳細な知覚，アクションが可能．インタフェースはハンドコードされた表象ではなく，シミュレータとしてモデルに入力される．だが，インタフェースに含まれる視覚表象の属性が動的に生成されることはない(オブジェクト認知はモデルの範囲外)

- EPIC

知覚とアクションの詳細なモデル化を実現

知覚：網膜像の構造を反映，注視

アクション：パンチ，指示，停止，つつく

GLEAN よりも明かに詳細．だが，環境との相互作用は GLEAN と同様(オブジェクト認知を行わな

い)

- ACT-R/PM
  デフォルトではオブジェクト認知を行わない(オブジェクト認知におけるエラーは滅多に起きないという仮定)
  オブジェクト認知をするモードも存在

```
(Letter-E
    is a Abstract-Letter
    value ''E''
    line-pos ...)
```

  文字の特徴を 16 のラインセグメントで表現
  オペレータの例
  Press-key: キーボードのキーを叩く
  Move-mouse, Click-mouse: マウスのカーソルを移動，ボタンイベントを生成
  Find-location: オブジェクトのスクリーン上の位置を同定
  Move-attention: 注意のシフト

- Sim-eyes Sim-hand(Ritter et al., 2000)
  ACT-R/PM の概念を更に発展
  ユーザインタフェースのシミュレータに知覚とアクションの関数を組みこんで開発
  モデル化のスキーマを多数定義
  認知モデルインタフェースマネージャーにより複数のアーキテクチャに適用可能

- 既存の研究の問題点

直接的ではない．認知モデルが扱う情報はディスプレイから直接的に抽出されない
プログラマの手腕によってシミュレーション結果が変化してしまう

3. Direct interaction with the interface

ユーザインタフェースと認知モデルのやりとりを可能にする関数の集合
特定のモデル，ユーザインタフェースに特化していない
オブジェクトの認知によってユーザインタフェースとモデルを繋ぐ
必ずしも正確に人の知覚プロセスを反映していない(eg., 並列性)
3 次元物体の知覚が目的ではなく，2 次元のユーザインタフェースの知覚
ユーザインタフェースは良く構造化．パターンが定型化
比較的簡単にモデルと課題を繋げることが可能

4. VisMap

画像処理の技術(Gonzales & Woods, 1992）と AI でのプランニングの研究（Hendler, Tate, & Drummond, 1990）を基礎

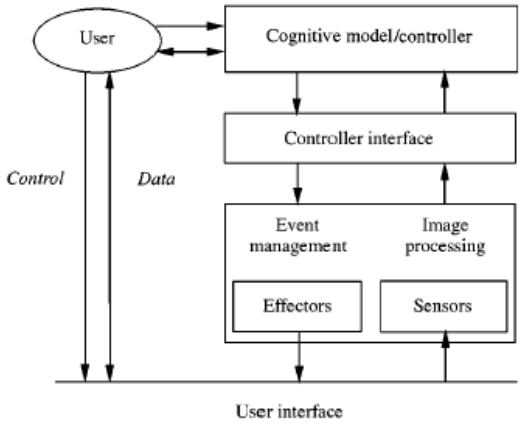

FIGURE 1. Genetic ibot architecture.

4.1. Visual perception in VisMap

- インプット：ディスプレイ上のローコンテンツ(ピクセルレベル)
- アウトプット：高次のユーザインタフェースのコンポーネント(コントロールインタフェース…過去の HCI 研究で用いられてきた抽象化されたインタフェース)

大まかに生体の視覚処理を参考：Ullman (1996)による高次視覚の計算論的説明、Chapman (1991)による Ullman 理論のインプリメンテーション

視覚モデルの研究における仮定を反映(Marr, 1982)

- 生体と計算機はハードウェアでは別．アルゴリズム的にはシミュレート可能
- 視覚は単体のプロセスではなく，蓄積的に環境の表象を構築する複数のプロセス
-

4.1.1. **低次視覚**

- 生体：オブジェクトの属性(色彩，奥行き，運動，テクスチャ)の解析
- VisMap：ローデータ(2 次元のピクセル)を変換→領域分割/運動解析

```
GrowGroupFrom (location)
    Set groupColor = color of pixel at location
    Add point at location to ExpansionList
    while (length of ExpansionList > 0)
        ExpandAbout (location)

ExpandAbout (location)
   Increment pixelCount
   For each neighboringPixel around location
       If (color of neighboringPixel = GroundColor)
           Call ComputerType (NeighboringPixel)
           if (NeighboringPixel isa BorderType)
              Add NeighboringPixel to BorderList
           Call UpdateBoundingBox ( )
           Add NeighboringPixel to ExpansionList
```

FIGURE 2. Region segmentation algorithm.

- 領域分割

  スクリーンのキャプチャ→特定のピクセル位置からスタート→ピクセル情報(色)を取得→隣接したセルが同じ色なら同じ領域→内部が一様の直線で区切られた領域が抽出
- 運動解析(×物体追尾)

  環境の変化(ウインドウの出現，文字の追加など)を認識(標準のグラフィックハードウェアによって

提供されるビット単位比較)→領域分割

◆生体との差異

- エッジ抽出，テクスチャ処理は行わない
  ユーザインタフェースでは領域は同じ色で構成されるため，不必要
- 並列処理ではなく系列処理
- 3 次元的な視覚は扱わない

◆VisMap による低次視覚の重要な点

- 単純な出力
- 知識やゴールが必要ない完全なボトムアップ処理
- 選択的注意のメカニズムが実装
  フォーカスの当っている個所のみのピクセル情報を取得
  フォーカスは認知モデルから

4.1.2. 高次視覚

- 生体：低次視覚の結果から物体認知，分類
- VisMap：アイコンの認知

VisMap では低次視覚からオブジェクト認知までプロセスを 2 段階に分割

- 中間段階： 空間的関係に関する一般的知識(上下左右)を利用

ボトムアップに特徴抽出(幅・高さ・エリア・矩形領域)

矩形について，ほとんどのユーザインタフェースは線で構成されるから，この仮定は妥当

```
Width:
  region.lowerRightX ()− region.upperLeftX() + 1

Height:
  region.lowerRightY ()− region.upperLeft() + 1

Area:
  MaxPossibleNumPixels = Width()* Height ()
    area = ActualNumPixels ()/MaxPossibleNumPixels

ContainedIn:
  inner.upperLeftX () ⩾ outer.upperLeftX () and
  inner.upperLeftY () ⩾ outer.upperLeftY () and
  inner.lowerRightX () ⩽ outer.lowerRightX () and
  inner.lowerRightY () ⩽ outer.lowerRightY ()
```

FIGURE 3. Examples of intermediate features.

- 高次視覚：物体に特化したテンプレートの利用

領域間の構造的関係を記述したルール…異なるオブジェクトに対応

```
If object Obj is a downArrow ()
      and Obj is containedIn () object RB
          such that RB is a raisedButton ()
          and RB is toTheRightOf () object RTA
              such that RTA is a rectangularTextArea ()
              and RTA is recessed () and has width () > height ()
   Then Obj is a component of a dropBox
```

FIGURE 4. A description rule for drop boxes.

他のテンプレート：角度，四角，円，チェックマーク，上，した，左，右の三角形→ボタン，ウインドウ，チックボックス，ラジオボタン，リストボックス，スクロールバー，アルファベット

◆生体との差異

オブジェクトの全体が見えなければ認知されない

3 次元の世界では困るが、ユーザインタフェースでは充分
矩形抽出
柔軟性に欠ける

4.1.3. Cognitive modeling interface

高次視覚と認知モデルのインタフェース(既存研究でのインタフェースシミュレータ)
オペレーション

Get-widgets: スクリーン上の全ての widget(ボタン，スクロールバー(スクロールボックス，スクロールアロー，バックグラウンド)，チェックボックス，ラジオボタン，ウインドウ，リストボックス，メニューアイテム)を返す

Find-widget: widget のプロパティ(位置，幅，高さ)を返す

Get-pixel-groups, Find-pixel-group: widget 以外のピクセルのかたまりに適用．入力として，矩形領域として構成されるオブジェクト(ウインドウ，メニューバー)．矩形内部のピクセルのかたまりを抽出(位置，色)

Get-strings, Find-string: スクリーン上の文字列を全て返す. 文字列の定義は複数の文字が空間を伴って並んでいること．文字列が抽出されたら，文字列を構成する文字と空間位置を返す

Get-letters: スクリーン上の全ての文字を返す.

Get-screen-objects: 以上で抽出されたオブジェクト間の関係を返す.

Set-bound: オペレータが適用される領域を制限する(注意の焦点)

Wait-for-object: 新たなオブジェクトが構成されたときにその時間を返す

◆生体との差異

網膜像の制約，オブジェクトの特徴の妥当性，注意のメカニズム
非常にチープ
だが，チープさが汎用性のあるツールとしては重要
より詳細な視覚的制約を考えてしまうと適用できるタスクが制約

◆現在までの実装

特徴の計算関数が 30/テンプレートが 80

◆問題点

マウスカーソルの動きは今のところ検知できない
文字認識の正答率が低い，Kerning によって上の文字との弁別ができなくなる

4.2. Motor behavior VisMap

ユーザインタフェースにおける運動のためのモジュール

Key-down, Key-up: キーボードのキーを押す/離す．遅延時間のデフォルト値は 200ms

Mouse-up, Mouse-down: マウスの左ボタンを押す/離す．遅延時間のデフォルト値は 100ms

Mouse-click, Mouse-double-click: Mouse-up, Mouse-down を組み合わせることで構成．遅延時間のデフォルト値はコンポーネントとなるアクションの合計

Mouse-to-point: マウスポインタを特定の位置に移動．遅延時間 T は Fitts の法則に従う

$$T=a+b\log_2(A/W+1)$$

定数のデフォルトは a=230, b=166
A は距離，W は名義的な定数

Key-down, Key-up: オブジェクトの場所を返し，Mouse-to-point のパラメータ A に入れる

Drag-to-point: Mouse-up, Mouse-down: Mouse-to-point と同じ内容だが，パラメータが異なる．a=135, b=249

各オペレータの遅延時間はモデルに即して変更可能
VisMap の目的は関数の集合を定義することで，特定の課題に特化したモデルを作ることではない

5. VisMap in practice

- 例 1：どのようにして VisMap が静的なシナリオ記述を作るか
- 例 2：ACT-R/PM と繋げる

5.1. Generating Scenarios with VisMap

今までの認知モデルは静的なインタフェースの記述をモデルに入力→VisMap では自動生成

例：number.text を開いて，記述されている文字の中から Five を選択(Figure 5)

one, two three, four, five, six, seven, eight, nine, ten

VisMap に含まれるオペレータ，Windows のディスプレイに関する知識をもとに Windows のユーザインタフェースとインタラクトするコントローラー(ゴール系列)を作成

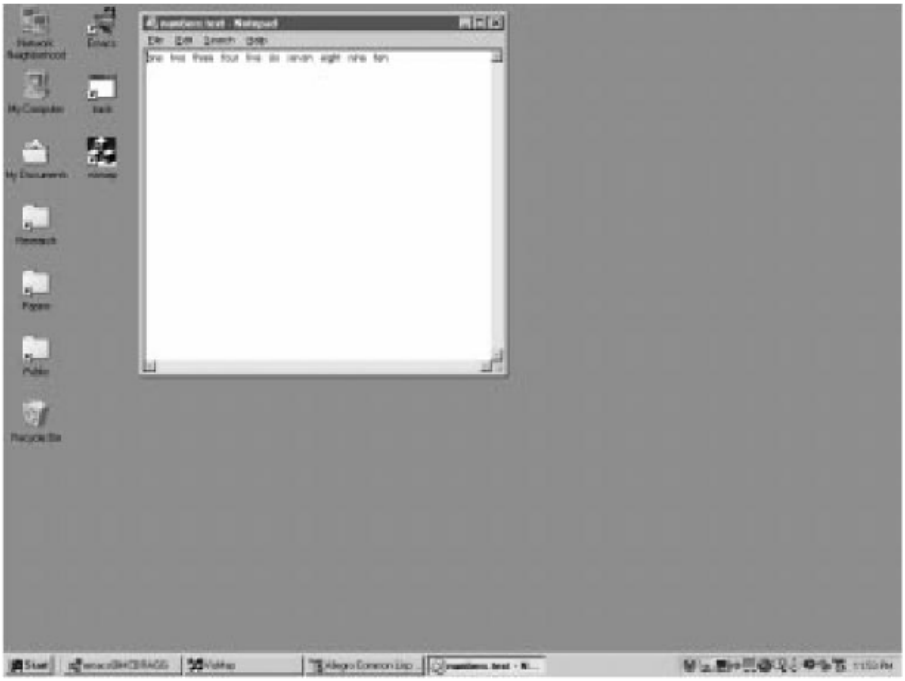

FIGURE 5. The Notepad example environment.

各ステップにおいて，ディスプレイの状態を表現する記述を生成

```
(MOUSE-POSITION 443 264)
(BUTTON :LEFT 3 :TOP 745 :RIGHT 53 :BOTTOM 763)
(BUTTON :LEFT 554 :TOP 16 :RIGHT 566 :BOTTOM 26)
(BUTTON :LEFT 536 :TOP 16: RIGHT 548 :BOTTOM 26) ···
(VSCROLL :LEFT 558 :TOP 55 :RIGHT 566 :BOTTOM 404)
(HSCROLL :LEFT 163 :TOP 409:RIGHT 553 :BOTTOM 553)
(WINDOW :LEFT 157 :TOP 11 :RIGHT 572 :BOTTOM 423) ···
(TEXT :LEFT 370 :TOP 754 :RIGHT 402 :BOTTOM 754 :STRING ''AIIERGO'')
(TEXT :LEFT 405 :TOP 754 :RIGHT 446 :BOTTOM 754 :STRING ''COMMON'')
(TEXT :LEFT 449 :TOP 754 :RIGHT 468 :BOTTOM 754 :STRING ''LISP'')
(TEXT :LEFT 12 :TOP 495 :RIGHT 51 :BOTTOM 495 :STRING ''RECYCIE'')
(TEXT :LEFT 54 :TOP 495 :RIGHT 69 :BOTTOM 495 :STRING ''BIN'') ···
(TEXT :LEFT 166 :TOP 60 :RIGHT 184 :BOTTOM 60 :STRING ''ONE'')
(TEXT :LEFT 191 :TOP 60 :RIGHT 210 :BOTTOM 60 :STRING ''TWO'')
(TEXT :LEFT 217 :TOP 60 :RIGHT 243 :BOTTOM 60 :STRING ''THREE'')
(TEXT :LEFT 250 :TOP 60 :RIGHT 271 :BOTTOM 60 :STRING ''FOUR'')
(TEXT :LEFT 277 :TOP 60 :RIGHT 296 :BOTTOM 60 :STRING ''FIVE'')
```

FIGURE 6. Partial description generated from the starting state.

(1) ノートパッドを立ち上げる

(a) タスクバーから notepad を探索(set-bounds, find-string). タスクバーが画面下にあることをコントローラーに記述済み. タスクバー中のストリングからノートバットを探索

(b) アプリケーションを開く(get-widgets; mouse-click; wait-for-objects). アプリケーションのボタンを押す→一定時間待機→以前のスクリーンの状態と現在の状態を比較し，ノートパッドのウインドウを領域抽出

(2) number. text を開く

(a) ノートパッドのウインドウからファイルメニューを探す(set-bounds, find-string). アプリケーションウインドウの上部にあるメニューバーのみを探索範囲. “ File ” の文字列を探す

(b) メニューのプルダウン: (move-to-point; get-widget; mouse-click; wait-for-objects). ファイルを押してから一定時間待機

(c) メニューから “ Open ” を探す (set-bounds; find-string; move-to-point; mouse-click; wait-fo-object). ダイアログボックスが出るのをまつ

(d) number.text を探す (set-bounds; find-string; move-to-point; mouse-click; wait-for-object; type-string; press-key). ダイアログボックスから number.box を探す，もしくは，テクストボックスに nunber.box をうつ

(3) ドキュメントから “ five ” を選択

(a) “ five ” を探す (set-bounds, find-string). フォントが違う場合は余計な処理が必要，単語間は半角 3 文字分のスペースで区切られていなければならない

(b) 単語の選択 (move-to-point, mouse-down, move-to-point, mouse-up). 単語をドラッグして選択. ショートカットとしてはダブルクリック

(c)

5.2. Executing modeling actions with VisMap

Byrne (1997) による ACT-R/PM のデモ：単語列から名詞を選択し，消去する

- プロダクション:attend-word, move-to-noun, click-noun, delete-noun, skip-word
- 単語の知識 :“ dog ”“ girl ”“ the ”“ hit ” +文法的カテゴリ

◆Byrne による ACT-R/PM：単語のラベルが貼られた領域が定義されたダイアログウインドウ. 視覚モジュールによって注意が領域を推移. 注意が向けられた領域が名詞なら, move-to-noun が活性化し，モーターモジュールがマウスポインタを動かし，消去. 名詞以外はスキップ

◆著者らによる ACT-R/PM + VisMap: インタフェースが Windows のノートパッド. 余分なインタフェースの記述は使用せずにダイレクトにユーザインタフェースとインタラクト

6. Discussion

特徴の計算やルールの作成に労力がかかる→特徴やルールを効率良く作成するためのツールを開発

- 特徴とルールの定義：ドローイングツールを利用して，効率的に特徴を定義できるツールを開発中．複数の特徴を空間的に結合することでルールを作るエディターを開発中
- シナリオの抽出：ACT-R などでは特殊な言語が使用される．VisMap のオペレータを ACT-R の言語で記述することは大変．簡単に翻訳のできるツールを開発中
- 回帰テスト：システム変更のログをとり，ユーザに提示する機能を開発中

より洗練された認知モデルとの統合が必要．VisMap なら可能

ACT-R 以外のアーキテクチャとの統合が必要．VisMap なら可能

```
; Loading ACTR-DEMO: actr-demo; example1.lisp
; (/Research/systems/vismap/domains/actr-demo/example1.lisp)
; Loading ACTR-DEMO: actr-demo; example1.actr
; (/Research/systems/vismap/domains/actr-demo/example1.actr)

Running ACT-R at time 0.000. Cycle 0 Time 0.000: Attend-Word
Running ACT-R at time 0.185. Cycle 1 Time 0.185: Skip-Word
PM: Skipping word: ''the''
Running ACT-R at time 1.235. Cycle 2 Time 1.235: Attend-Word
Running ACT-R at time 1.420. Cycle 3 Time 1.420: Move-To-Noun
PM: Going to delete word: ''boy''
VisMap: Move cursor to (126 104).
Running ACT-R at time 5.106. Cycle 4 Time 5.106: Click-Noun
VisMap: Key press: Mouse-Double-Click.
Running ACT-R at time 5.806. Cycle 5 Time 5.806: Delete-Noun
Running ACT-R at time 5.856. Cycle 6 Time 5.856: Attend-Word
VisMap: Key Press: Delete.
Running ACT-R at time 6.256. Cycle 7 Time 6.256: Skip-Word
PM: Skipping word: ''the''
Running ACT-R at time 7.306. Cycle 8 Time 7.306: Attend-Word
Running ACT-R at time 7.491. Cycle 9 Time 7.491: Skip-Word
PM: Skipping word: ''hit''
Running ACT-R at time 8.541. Cycle 10 Time 8.541: Attend-Word
Running ACT-R at time 8.726. Cycle 11 Time 8.726: Skip-Word
PM: Skipping word: ''hit''
Running ACT-R at time 9.776. Cycle 12 Time 9.776: Attend-Word
Running ACT-R at time 9.961. Cycle 13 Time 9.961: Skip-Word
PM: Skipping word: ''the''
Running ACT-R at time 11.011. Cycle 14 Time 11.011: Attend-Word
Running ACT-R at time 11.196. Cycle 15 Time 11.196: Move-To-Noun
PM: Going to delete work: ''girl''
VisMap: Move cursor to (166 104).
Running ACT-R at time 14.882. Cycle 16 Time 14.882: Click-Noun
VisMap: Key press: Mouse-Double-Click.
Running ACT-R at time 15.582. Cycle 17 Time 15.582: Delete-Noun
Running ACT-R at time 15.632. Cycle 18 Time 15.632: Attend-Word
VisMap: Key press: Delete.
...
VisMap: Key press: Return.
...
```

FIGURE 7. A trace of ACT-R/PM and VisMap on the noun selection task.

LICAI モデル：マニュアルやオンラインヘルプの理解に基づいて操作を選択してインタフェース上で実行する過程の認知モデル。Kintsch の文章理解の認知モデル（Construction-Integration モデル）に基礎を置いている

GOMS：ユーザインタフェースに関して，ユーザの課題遂行行動を階層的に分解していくタスク分析の手法の１つ．この手法では，ユーザの課題遂行行動を，まず解決すべき目標（goal）の階層として分解していき，最終的に，それぞれの下位目標を達成するための方法（method）や，その方法を構成する最小単位の操作（operator）に分解していく．また目標達成の際に複数の方法がある場合には，それを選択するための規則（selection rule）が設定される．GOMS という名称は，これら 4 つのキー

ワードの頭文字を組み合わせたもの．構築されるモデルは，目標や方法などの名称を要素とする木構造の形になるが，この木構造の形が縦に長くなりすぎたり，横に深くなりすぎたりしていると，ユーザの認知的負担が高くなると予想される．また，あるシステムの複数の操作に関する木構造が類似の形をしているかどうかで，操作の一貫性を確認することもできる．