

問題解決オントロジーに基づく概念レベルプログラミング環境 CLEPE

瀬田 和久[†] 池田 満[†] 島 輝行[†] 角所 収^{††} 溝口 理一郎[†]

CLEPE: a Task Ontology Based Conceptual Level Programming Environment

Kazuhiisa SETA[†], Mitsuru IKEDA[†], Teruyuki SHIMA[†], Osamu KAKUSHO^{††}, and Riichiro MIZOGUCHI[†]

あらまし 我々が提案している問題解決オントロジーの概念は、人間にとって自然に受け入れられ、かつ計算機にとっての操作的な意味も明確になるような問題解決概念の体系的定義である。本論文では、問題解決オントロジーの一利用形態としての概念レベルプログラミング環境が提供する様々な機能を紹介しながら、問題解決オントロジーの有用性を示す。問題解決オントロジーをシステムの基盤にすることで概念レベルプログラミング環境では、計算機に馴染みのないエンドユーザが、(a)日頃意識している概念を中心に、自分自身の知識を外化しやすい様な枠組みを設定することが可能になり、(b) エンドユーザが日頃意識していない概念であるが、計算機での実行を考える際に必要な概念については、それを計算機側が読みとる能力を備えることができる。(c) 問題解決知識をエンドユーザの概念レベルで実行することができる。本稿ではこのことを実例を挙げながら詳述する。

キーワード 概念レベルプログラミング環境, 問題解決オントロジー, 知識共有/ 再利用, ヒューマンコンピュータインタラクション

1. はじめに

問題解決モデルの構築作業を、計算機システムの利用に不慣れたエンドユーザでも行うことの出来るような環境の実現を目指して、エンドユーザプログラミング環境に関する研究が様々なアプローチで進められている[1][2][3]。エンドユーザプログラミング環境は、対象世界に関する様々な概念に対応するソフトウェアコンポーネントを準備し、それをを用いてエンドユーザが容易に問題解決モデルを記述(概念レベルプログラミング) できるような表現の枠組みと、そのデバッグ及び実行のための環境を提供する。そこでの「表現」としては線形の言語よりも表現力に富むものが想定されることが一般的である。本研究で開発を進めている概念レベルプログラミング環境の背景にあるアイデアも基本的にこの線上にある。

ここで重要なことは、自分の頭の中にある「知識」を容易にかつ円滑に「表現」へと変換できるようにするための仕組みを考えることである。この問題を考え

るにあたっては、ユーザインタフェイスの表面的な問題として扱うのではなく、エンドユーザが自分の問題解決知識を「モデル」化するのに必要な概念に関する深い洞察に基づいて考えなければならない。筆者らが提案している問題解決オントロジー[4] は、エンドユーザが問題解決過程をモデル化するにあたって必要となる概念(オブジェクト・基本処理プロセス・制御、及びそれらの関係)を分析しこれを体系的に定義したものである。この問題解決オントロジーをモデル化の基盤とすることで、自分の知識を表現へと円滑に変換することができるような環境を実現することができる。本研究では、人間が行う様々な問題解決の内、専門家が日常的に行っているスケジューリングや診断などの処理の手順を中心的に考える問題解決を対象として、その問題解決モデルの記述環境として概念レベルプログラミング環境CLEPE(Conceptual LEvel Programming Envorinment)[4][5]を実現した。CLEPEにおいて、エンドユーザは(1) 日頃使っている平易な言葉を使って問題解決モデルを記述(モデル化) ことができ、(2) 問題解決モデルの実行内容をわかりやすい形で確認/デバッグすることができる。さらに、(3) 平易な言葉を使って記述した問題解決モデルを、記号レベルのプログラムコード(Lispコード)へと変換することができる。

本論文では、主に(1)(2)の特徴について中心に述

[†] 大阪大学産業科学研究所, 大阪府茨木市
Institute of Scientific and Industrial Research, Osaka University, Ibaraki-shi, 567, Japan
^{††} 兵庫大学経済情報学部, 兵庫県加古川市
Faculty of Economics and Information Science, Hyogo University, Kakogawa-shi, 675-01, Japan

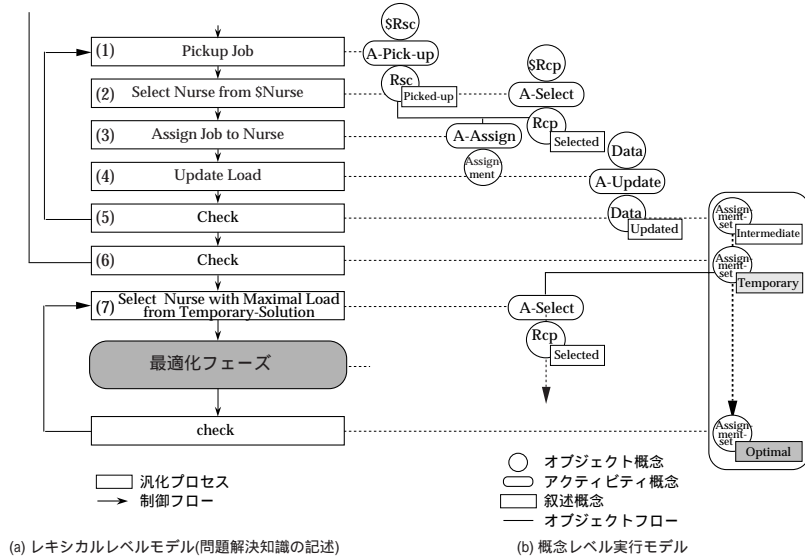


図2 レキシカルレベルモデルと概念レベル実行モデル(タスクフローモデル)
 Fig. 2 Lexical level model and conceptual level execution model (task flow model)

決過程をモデル化するために必要となる概念，すなわちオブジェクト・基本処理プロセス・制御、及びそれらの関係を体系的に定義したものである。問題解決オントロジーに従って、エンドユーザは問題解決知識を問題解決モデルへとモデル化し、システムは問題解決モデルの整合性を検証することが可能になる。

2.3 CLEPEにおける問題解決オントロジーの役割

図1は問題解決オントロジーを中心に捉えたCLEPEの概念図を示している。CLEPEには、問題解決モデル記述環境の側面(図1(a)(b))と問題解決オントロジー構築支援環境の側面(図1(c))がある[4][5]。ここでは前者の側面から前節のI~IVの要求に相当する問題解決オントロジーの役割について述べる。

既に述べたように問題解決知識を問題解決モデルとしてモデル化する際の規約を与えるのが問題解決オントロジーである。問題解決オントロジーは、さらにその役割によって、問題解決知識の記述上の規則を定めるレキシカルレベルオントロジーと、その記述の意味内容を定義するための概念レベルオントロジーに分割されている。二つのオントロジーは、それぞれレキシカルレベルモデル、概念レベル実行モデルの構成上の規約を与えている。問題解決モデルは、厳密に言えばこの二つのモデルを総称する用語である。CLEPE、エンドユーザ、二つのモデルの関係を直感的に書き下すと次のようになる。「エンドユーザが自分の問題解

決知識をレキシカルレベルモデルとして記述し、CLEPEはその意味内容を概念レベル実行モデルとして再構成する」。図2に問題解決モデル記述環境において問題解決オントロジーに基づいて構成されたスケジューリング問題(要員配置問題)の問題解決モデルを示している。図の左側がエンドユーザが記述したレキシカルレベルモデルを表し、右側が概念レベル実行モデルを表している。それぞれのモデルの詳細は後述するとして、問題解決オントロジーの役割を前節で示した要求に対応させてまとめると以下ようになる。

基本的に、レキシカルレベルオントロジーは、以下の規約を公理¹として定めている。

- I. に関して、
 - 問題解決モデルを記述する際にエンドユーザが使う語彙(e.g. Pickup, Jobなど)を提供し、
 - 語の配列に関する性質(図2左側の各ノード内部の文法)を規約として定めている。さらに、
 - レキシカルレベルモデルを構成する各ノードと制御リンクの接続に関する規約を定めている。
- II. に関して、
 - エンドユーザが書いたレキシカルレベルモデルと概念レベル実行モデルの間の対応関係を定めるた

1. 問題解決オントロジーとして形式的に定義された宣言的な知識は、対象世界において証明なしに受けれるべき規約である。この意味で問題解決オントロジーが定める規約を公理と呼ぶことにする。



図4 オブジェクトフロー解釈の実行画面
Fig.4 A screen image of object flow analysis

問題に対する問題解決モデルの事例検索機能がエンドユーザに提供される。事例ベースは汎化語彙によってインデックスづけされており、検索は図3(a)で記述されたゴール等の情報に基づいてなされる。検索結果として得られた事例中の語彙は、汎化語彙を介してエンドユーザに適応した語彙に置換されて提示されることになる(図3(b))。エンドユーザは必要に応じてこの事例を参照、あるいはコピー＆ペーストによって部分的に再利用しながら問題解決モデルを記述する(図3(c))ことができる。図3(d)のブラウザでは、エンドユーザが利用可能な用語がレキシカルレベルオントロジーに従って階層的に提示される。エンドユーザは、その用語をマウス操作によって選択し、組み合わせることによって、問題解決モデルの処理内容(ノードの内容)を記述する(図3(c))。この様にして記述された問題解決モデル(レキシカルレベルモデル)は、問題解決オントロジーが定義するレキシカルレベル公理に基づいて文法的な整合性が検証され、内部的な実行モデルとしての概念レベル実行モデルが構成される。

既に述べた様に、CLEPEでは制御フローを中心とした記述の枠組みを提供している。したがって、例えば図2のレキシカルレベルモデルでは、「(2)Select」プロセスの出力である「Nurse」と「(3)Assign」プロセスの入力である「Nurse」が「同一の」Nurseであるといったオブジェクトフローに関する情報が表現として明示的に表されていない。CLEPEでは、このオブジェクトフローを補完する機能(オブジェクトフロー解釈と呼ぶ)を備えている。図4にこのオブジェクトフローをシステムが解釈する際に出力されるメッセージを示している。このようなメッセージを介したエンドユーザとシステムの対話的な作業を通じて、無矛盾なオブジェ

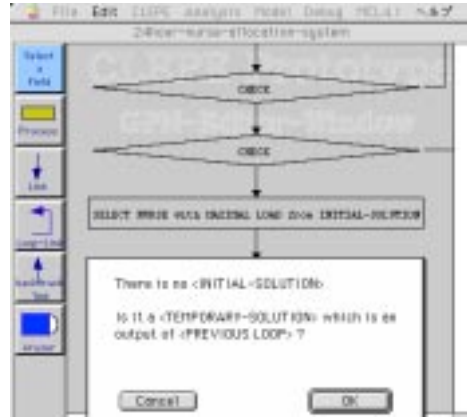


図5 タスクオントロジーに基づく問題解決知識の整合性の検証
Fig.5 Consistency check of problem solving knowledge based on task ontology

クトフローが同定される。

また、図5にエンドユーザが記述したレキシカルレベルモデルに誤りがある場合に、システムから出力されるメッセージを示している。この例では、図2の(7)のプロセスにおいて「暫定解から最大負荷の看護婦を選ぶ」と記述すべき所を、「初期解」から選ぶと記述してしまっている。これに対してシステムは、「初期解に対応するものがない。前のループで作られた暫定解ではないか?」というメッセージを出力している。この情報に基づいて、エンドユーザはレキシカルレベルモデルの修正を行うことができる。このオブジェクトフロー解釈の仕組みについては、4. で述べる。

(Phase 2) 問題解決モデルの実行/修正

エンドユーザが問題解決モデルを記述した後で次に行う作業は、自分が書いた問題解決モデルが自分の意図通りのものであるかを確認し、必要に応じて問題解決モデルの記述を修正することである。自分が書いたモデルが意図通りのものであるかどうかを確認する際の根拠になるのが、モデルの振る舞いに関する情報である。概念レベル実行は、そのような情報を問題解決モデルの記述レベルと同じレベルで提供する機能である。そこでは、問題解決の実行を通じたアクティビティのオブジェクトへの作用、オブジェクトの状態の変化や構成の変化の様子が記述レベルと同一のレベルでエンドユーザに対して提示される。

概念レベルの実行可能性は問題解決オントロジーに基づくシステムが持つ特徴的な機能である。この機能の特徴は従来型のプログラミング環境と対比すると分かりやすい。従来型においてはプログラムの振る舞い

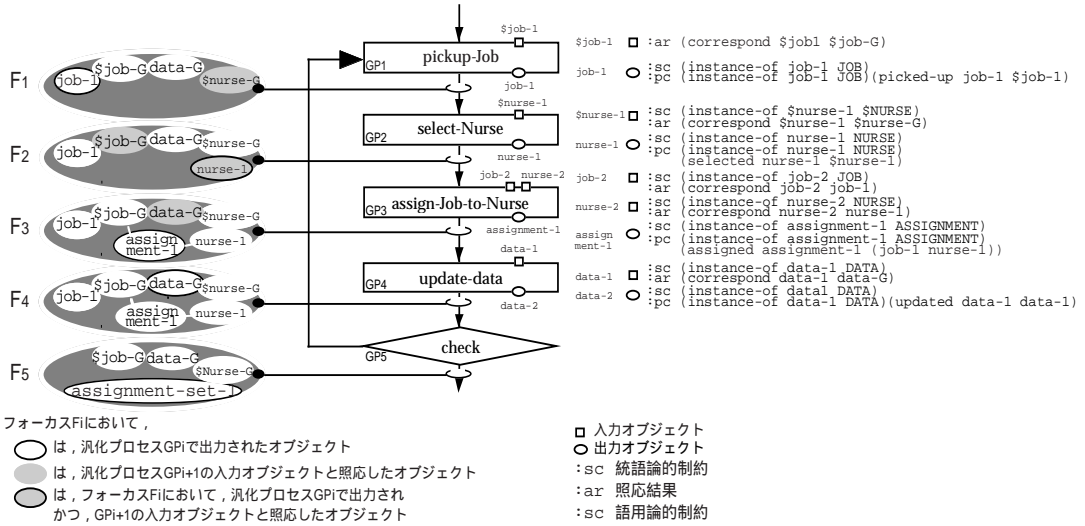


図7 レキシカルレベルモデルの一部とフォーカス
Fig. 7 Focus model which corresponds to the lexical level model

る，図7 左にフォーカスモデルを示している。フォーカスモデルはタスクの制御構造が設定するオブジェクト参照に関するコンテキストをモデル化したものである。図中のフォーカスFiでは汎化プロセスGPi+1の入力オブジェクトと照応する可能性のあるオブジェクトがフォーカスされている。フォーカスFi内の黒枠で囲われた楕円は、GPiで出力されたオブジェクトを表し、シャドウされた楕円は、GPi+1の入力オブジェクトと照応したオブジェクトを表している。

照応関係の同定は、フォーカスされたオブジェクトの内、(1) で作られた統語論的制約を満たすオブジェクトを同定する事によって行われる。そして、照応結果(ar:anaphoric result)に基づいて照応関係にあるオブジェクトは同一化され(例えば、GP3において入力job-2が、GP1の出力のjob-1と照応することからこれらは同一化される)、語用論的制約(pc:pragmatic constraint)が生成される。語用論的制約は、照応結果に基づいて統語論的制約を更新したものであり、統語論的制約で与えられる制約に加えて、オブジェクトの生滅に関する変化を捉えるための概念も含まれている。これは、照応結果とアクティビティの作用に基づいて生成される。例えば、GP3に関するpcでは、入力が「job-1」「nurse-1」と照応すること、assignの作用を定めるアクティビティ「A-Assigned(5.2で後述)」が入力オブジェクトを構成要素とする割り付けオブジェクトを生成することから、「job-1」と「nurse-1」から構成される「assignment-1」が生成される(assigned assign-

ment-1 (job-1 nurse-1))という制約が生成される。

フォーカスFiの更新はGPiの照応結果、アクティビティの作用、及び、レキシカルレベルの制御構造に基づいて行われる。例えばフォーカスF3では、GP3の照応結果つまり、GP3の入力オブジェクトがGP1(「pickup job」プロセス)で出力された「job-1」とGP2(「select nurse」プロセス)で出力された「nurse-1」と照応すること、及びassignプロセスの作用として入力された二つのオブジェクトからなる「assignment」を生成する作用をもつことに基づいて、フォーカスF3がF4へと更新される。さらに、レキシカルレベルモデルの制御構造に基づくフォーカスの更新の例(F5)として、ループのスコープ外ではループ内で生成されたオブジェクトに対するフォーカスが消滅し、ループ全体の実行を通して生成されたオブジェクトassignment-set-1に対するフォーカスが生成される様子が表されている。

このフォーカスはフォーカスマネージャによって管理され、照応解析エンジンによる照応結果は、WMマネージャによって管理される。フォーカスは、照応データの追加、更新に応じてWMマネージャがフォーカスマネージャを起動することで更新され、データ駆動でフォーカスが更新されるメカニズムが実現されている。

最終的に得られた概念レベル実行モデルを元に、エンドユーザはExecutorを介して次節で述べる概念レベル実行を行うことができる。

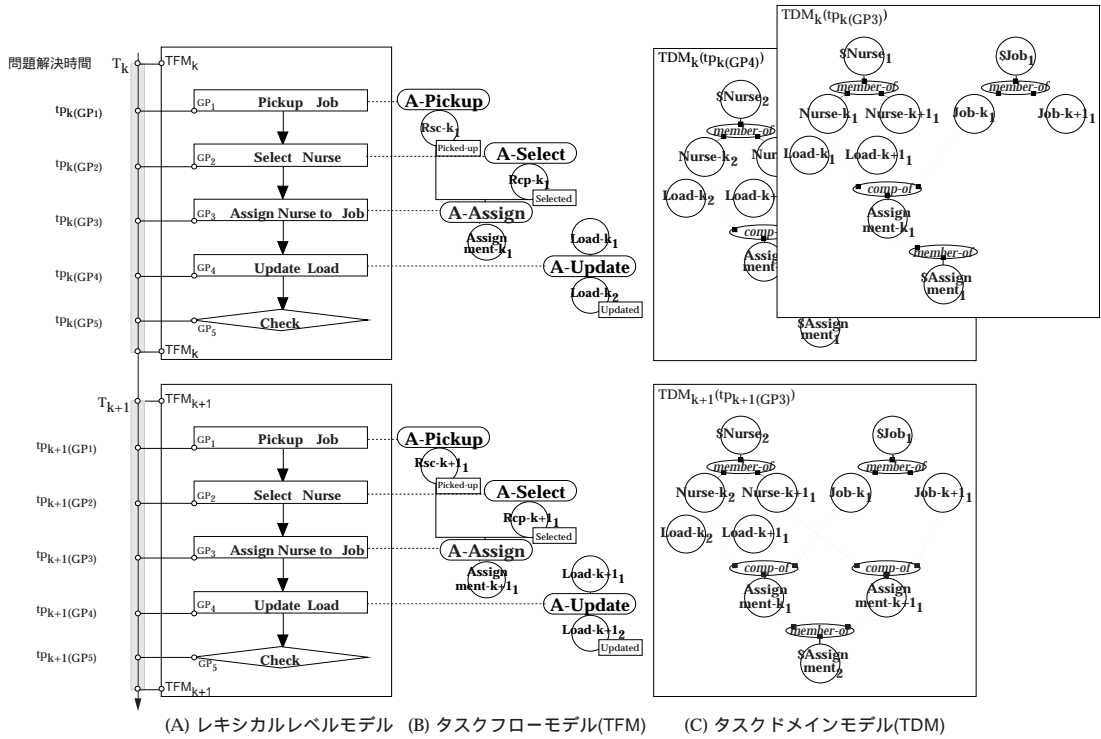


図9 タスクフローモデル(上側)とタスクドメインモデル(下側)
 Fig. 9 An image of task flow model (upper) and task-domain model (lower)

ここで、オブジェクト i は Obj_{i_v} と表されており、 v は後述するバージョンを表している。TFMでの処理によるTDMの変化はT-ドメインオントロジーを通じて起こる。例えば、 $TDM_{k+1}(GP_3)$ では、Assignの作用による対象の世界の変化として、ジョブ($Job-k+1_1$)と看護婦($Nurse-k+1_1$)からなる(component-of)割り付けオブジェクト($Assignment-k+1_1$)が新たに生成され解($Assignment$)の構成要素(member-of)となる様子が示されている。

ここで、オブジェクトの変化を捉えるための概念としてバージョンの概念を導入する。バージョンは問題解決時間を用いて「ある問題解決時間 tp_x からある問題解決時間 tp_y まで、状態 S_{xy} である」という形式で表現される。そして、オブジェクトが変化したことを表す、「バージョンの変化」はそのオブジェクト自体の状態の変化や他のオブジェクトとの関係の変化(e.g.集合の構成要素の変化など)に基づいて行われる。 $TDM_k(tp_k(GP_4))$ では、オブジェクトの構成要素の属性値変化に伴う全体オブジェクトのバージョンの変化が示されている。TFMにおいて看護婦($Rcp-k_1$)の負荷($Load-k_1$)が($Load-k_2$ へと)更新された際には、その変化

がTDMにおいて伝播し、看護婦($Nurse-k_2$)を構成要素とする看護婦の集合($\$Nurse$)のバージョンが($\$Nurse_2$ へと)更新されている。これによってシステムは、例えばSelectプロセスの働きについて単に「看護婦の集合から看護婦を選ぶ」といった実行ではなく「前回の割り付けに基づいて負荷が更新された看護婦の集合から看護婦を選ぶ」といった問題解決のコンテキストを捉えた適切な情報を与えることが可能になる。

5.2 オブジェクトの変化のモデル

問題解決オントロジーが定める様々な概念の内、上で述べた物の変化を捉えるための概念は、アクティビティの作用概念定義である。

図10にスケジュールの受け手(Rcp)に対して資源(Rsc)を割り付けるアクティビティ「A-Assign」の作用である作用概念「A-Assigned」の定義を示している。作用概念は主にアクティビティ定義の作用定義フィールドで使われる。アクティビティが入力に対して処理を行うことによって出す出力と、対象世界における変化が定義される。概念レベル実行でエンドユーザに示される処理の内容や対象世界における変化は、この作用定義に基づいて提供される。

6 関連研究

問題解決オントロジーを対象とし、概念レベル実行に相当する機能を備えた研究であるTOVEプロジェクト[9]については、別稿[4]で比較している。彼らのプロジェクトとの違いを一言で言えば、TOVEプロジェクトでは、エンドユーザを対象としておらず、レキシカルレベルモデルに相当する表現の枠組みを設定していない点をあげることができる。

エンドユーザプログラミングの研究として代表的なものにFischerらが進めているDODE[2]とCypherらが進めているKidSim[3]がある。

DODEは、ネットワークや、キッチン設計などを対象として、領域固有の概念を取り入れたエンドユーザプログラミング環境を実現している。対象領域のオブジェクトを部品として準備し、これを用いてソフトウェアを構成するという点で我々のアプローチと概ね一致している。しかし、問題解決を捉える際の規約となるオントロジーが明確になっておらず、モデルの整合性を検証する枠組みがない。

KidSimは、小学生に対する創発教育を目的としたシステムである。エンドユーザはプロダクションルールの形式で仮想世界を定義し、システムはそれに基づいた仮想世界のシミュレーションを行うことができる。我々のアプローチとKidSimの違いは、KidSimはシミュレーションという一般的な領域の中で対象を限定しておらず、しかも、プロダクションルールという一般的な表現の枠組みを設定しているため、対象世界を定義するための概念や属性をエンドユーザ自身が定義する必要があり、エンドユーザプログラミングを実現しているとは言い難い。これに対して、我々のアプローチでは、知識をモデル化する際のエンドユーザの特性を分析し、オブジェクトフロー解釈や概念レベル実行などのエンドユーザと計算機間の概念ギャップを埋めるための機能を実現している。また、問題解決を記述するために必要な概念や属性を問題解決オントロジーとしてあらかじめ準備しておくことで、問題解決モデルを記述するにあたって、エンドユーザが概念を定義するという作業プロセスに煩わされることがない。

Multis[6][7]は本研究の前身であり、レキシカルレベルモデルから具体的な問題解決を行う記号レベルのプログラムコードへの連続性が実現されており本研究の

基盤になっている。本稿で述べた枠組みにおいて、エンドユーザが自分の書いた問題解決モデルの動作を概念レベル実行により確認した後、制約などのドメイン固有の知識に関するインタビューが行うことによって、記号レベルのビルディングブロックと対応づけられる。最終的に、システムは具体的な問題を解くプログラムコードを生成する。Multisでは明らかになっていなかった公理[4]を利用することで、オントロジーに基づくモデルの整合性の検証や、機能的な側面からは、オブジェクトフロー解釈、概念レベル実行といった機能を実現することが可能になっている。

7 結論

問題解決オントロジーは、人間にとって自然に受け入れられ、かつ計算機にとっての操作的な意味も明確になるような問題解決概念の体系的定義である[4]。本稿では、問題解決オントロジーの有効性を明らかにすることを目的として、概念レベルプログラミング環境CLEPEの構成について論じながら、そこで提供される様々な機能を紹介した。特に、モデルの整合性の検証、オブジェクトフロー解釈、概念レベル実行といった機能は、問題解決オントロジーを基礎におくことによって可能になっている。

本稿では、エンドユーザプログラミング環境において、頭の中にある「知識」の「モデル」化を支援するための基本となる考え方、問題解決オントロジーを基盤とすることによって実現される機能的な側面を明らかにした。問題解決オントロジーによって、エンドユーザが捉えている問題解決の過程と、システムの内部的なモデルとのギャップを縮めることが原理的に可能になっている。エンドユーザプログラミング環境のもう一つの問題であるインタフェースの受け入れやすさ(例えば、エンドユーザにとって受け入れやすい概念レベル実行の表示方法(GUI))について、オントロジーは直接的な解を与えていない。現状では、問題解決オントロジーによって「何を」「どう」表現すべきかということの基礎が与えられた段階にある。この基礎の上に、エンドユーザにとって受け入れやすい視覚的な表現に関する検討をタスクタイプ毎に行っていく予定である。

問題解決オントロジーの評価に関して大きく2つの段階が必要になる。第一は、ある特定のタスクタイプに対する有効性であり、第二は、様々なタスクに対す