宣言的記述によるクラウドソーシングシステムの開発支援

権守　健嗣†　　森嶋　厚行††,†††

† 筑波大学 情報学群情報メディア創成学類　〒 305–8550 茨城県つくば市春日 1–2
†† 筑波大学 図書館情報メディア系/知的コミュニティ基盤研究センター　〒 305–8550 茨城県つくば市春日 1–2
††† JST さきがけ
E-mail: †s1113114@u.tsukuba.ac.jp ††mori@slis.tsukuba.ac.jp

あらまし　近年，クラウドソーシングによるデータ収集・整理・処理等が広く行われている．それに伴い，クラウドソーシングシステムの開発を支援するツールも多く登場している．一般に，クラウドソーシングは宣言的記述と相性が良いと考えられるが，既存のツールの多くは，手続き型言語によるクラウドソーシングシステムの実装を支援するものである．本論文では，宣言的記述によりクラウドソーシングシステムの開発を支援する手法を提案する．具体的には，クラウドソーシングシステムを宣言的に記述するための CTD(Condition-Task-Data) 抽象化を提案する．CTD 抽象化は，商用のクラウドソーシングプラットフォームなどで広く使われているタスクテンプレートの一般化になっている．本論文はさらに，CTD 抽象化が大きな表現力を持つ事を示すほか，CTD 抽象化を用いたクラウドソーシングシステム開発支援ツール Crapid についても説明する．

キーワード　クラウドソーシング，開発支援，宣言型プログラミング

1. はじめに

計算機ネットワーク技術の発達に伴い，不特定多数の群衆に仕事を委託するクラウドソーシングが広く行われるようになっている．一般に，クラウドソーシングを行うシステムは，クラウドソーシングシステムと呼ばれている [1]．例えば，画像のタグを人手で付与する ESP Game [2] や，OCR で正しく処理できない画像から人手でテキスト抽出を行う reCAPTCHA [3] といったクラウドソーシングシステムがある．

クラウドソーシングの重要性が認識されてきたことから，クラウドソーシングのための基礎機能を提供するクラウドソーシングプラットフォームが登場してきた．例えば，Amazon Mechanical Turk [4]（以下，MTurk）は，少額の謝金が支払われる作業 (タスク) を群衆に割り当てるための市場を提供するクラウドソーシングプラットフォームである．これらのプラットフォームではソフトウェアから呼び出し可能な API を提供しており，クラウドソーシングシステムのソフトウェアからタスクを登録することを可能としている．

クラウドソーシングシステム実現のためのプログラミングには固有の頻出パターンがあるため，ソフトウェア開発を支援するためのツールがいくつか開発されてきた．例えば，TurKit [5] ではプログラムの再実行の効率化を図るための仕組みやタスクを定義する関数などが定義されている．TurKit をはじめとする既存の支援ツールの多くは，手続き型プログラミングの支援を行う．例えば，手続き型プログラムから呼び出し可能なライブラリを提供するといったものがある．

本研究では，宣言的記述によりクラウドソーシングシステムの開発を支援する手法を提案する．具体的には，ルールベースの言語を用いてクラウドソーシングシステムを記述し，そこから実行可能なコードを生成する．クラウドソーシングシステムは，次の 3 つの理由から，本質的に宣言的記述と相性がよいと考えられる．

(1) 群衆は並列に非同期で作業すると仮定することが自然であるが，手続き型のコードではその記述が容易ではない．例えば，fork 関数等を利用して同時実行を実現するなどの工夫が必要である [5].

(2) クラウドソーシングシステムを実現する手続き自体を記述することが難しい場合が存在する．例えば，つくば市のカフェの情報を集めて，駅に近い順に並べるクラウドソーシングを考える．この場合，つくば市のカフェを入力するタスク A を N 回実行 (クラウドが作業) し，次に，入力されたカフェを近い順に並べるタスク B を実行するという手続きが考えられる．しかし，この手続きの記述において，N をいくつに設定するか，すなわち，いつタスク A を終了してタスク B に移行すれば良いかは自明ではない．一方，宣言的記述では N を指定しない記述が可能であり，インクリメンタルな実行も容易である．

(3) クラウドソーシングは本質的に人手での処理を必要とするため，一般に処理のコストが高い．したがって，計算機だけの処理と比較しても最適化の重要性が高いと考えられる．宣言的記述に関しては最適化の技法が数多く研究されているため，これらの知見を適用することができると考えられる．

以上の理由から，本質的にクラウドソーシングは宣言的記述と相性が良いと考えられるが，Java 等の手続き型言語と比較して，Prolog や Datalog のようなホーン節形式のルールによる宣言的記述は一般に普及していない．したがって，現在の多くのプログラマにとって単純なホーン節形式によるプログラミングは敷居が高いと考えられる．また，SQL でクラウドソーシングを記述するアプローチも存在する [6] が，後述するようにその記述力は高くない．

本論文では，まず，クラウドソーシングシステム開発のため

の宣言的記述として **CTD 抽象化** (Condition-Task-Data Abstraction) を提案する．CTD 抽象化は，クラウドソーシングシステムを，CTD ルールの集合として記述するものである．CTD ルールとは，3 章で説明するように，MTurk を含む多くのクラウドソーシングプラットフォームで広く使われているタスクテンプレートの一般化である．したがって，CTD 抽象化は，既存のクラウドソーシングプラットフォームの利用者にとってなじみやすいと考えられる．

次に，本論文は CTD 抽象化の表現力について議論する．一般に，プログラミング言語の表現力を議論するにはチューリング完全性が議論されるが，本論文では，記述可能な “人とのインタラクション” に着目した記述力の尺度 (**ゲーム完全性**と呼ぶ) を導入する．さらに，CTD 抽象化はチューリング完全であり，かつ，ゲーム完全である事を示す．

最後に，CTD 抽象化を用いた開発支援ツール Crapid の説明を行う．Crapid は，フォームベースで CTD 抽象化によるクラウドソーシングシステムの記述から，実行可能なコードを出力するものである．具体的には，様々なタスクテンプレートのひな形を Crapid 側で用意する事により，容易にプログラムが書けるようになっている．

本論文の構成は次の通りである．2 節では関連研究について述べる．3 節では，CTD 抽象化について説明し，4 節でその表現力について議論し，他の開発支援ツール等と CTD 抽象化の表現力を比較する．5 節では，CTD ルールの集合をプログラミング言語 CyLog に変換するツール Crapid について説明する．6 節はまとめと今後の課題である．

2. 関連研究・関連システム

近年，クラウドソーシングシステムの開発支援について，多くの論文で議論されている．これらは，クラウドソーシングのためのソフトウェアの抽象化の方法が異なる．

(1) 手続き型プログラムとして抽象化：TurKit [5] は，クラウドソーシングのための関数呼び出しと crash-and-rerun モデルを提供する．

(2) MapReduce 風記述として抽象化：CrowdForge [9] は，MTurk 上で複雑で相互依存なタスクを行うための MapReduce 風のフレームワークであり，クラウドソーシングシステムを partition, map, reduce を表すタスクの集合として抽象化する．CrowdForge と本研究との違いは，抽象化の方法とその表現力である．詳細は 4.3 節で議論するが，CTD 抽象化の方が表現力が大きい．

(3) 制御フローとデータフローとして抽象化：CrowdLang [10] は，クラウドソーシングシステムで頻出するパターンを構成要素として制御フロー/データフローの形式で，クラウドソーシングシステムを記述するためのフレームワークである．論文 [10] では，CrowdLang は抽象レベルの高いコードを記述する目的で使われており，直接実行可能なコードを書くための言語には見えない．

(4) Prolog 風ルールの集合として抽象化：CyLog [7] はクラウドソーシングシステムを記述するための Prolog 風ルールベースプログラミング言語である．しかし，単純なホーン節のルールは，必ずしも一般のプログラマが書きやすいものではない．

```
<?xml version="1.0" encoding="UTF-8"?>
<QuestionForm xmlns="http://mechanicalturk.
amazonaws.com/AWSMechanicalTurkDataSchemas/
2005-10-01/QuestionForm.xsd">
  <Question>
   <QuestionIdentifier>1</QuestionIdentifier>
   <QuestionContent>
     <Text>How many movies have you seen this
     month?</Text>
   </QuestionContent>
   <AnswerSpecification>
     <FreeTextAnswer/>
   </AnswerSpecification>
  </Question>
</QuestionForm>
```

図 1　MTurk のタスクテンプレート例

(5) SQL 風記述として抽象化：CrowdDB [6] は，SQL 風言語によるクラウドソーシングの記述をサポートする．詳細は 4.3 節で議論するが，CTD 抽象化の方が表現力が大きい．

本研究で提案する CTD 抽象化に用いる CTD ルールは，ECA (Event-Condition-Action) ルールの組み合わせとして実装可能と考えられる．すなわち，CTD ルールは，条件が成立するとマイクロタスクを生成するための ECA ルール (Action 部でマイクロタスクを生成) と，マイクロタスクが実行されるとデータを格納する ECA ルール (Action 部でデータベースに格納) の組み合わせと見なすことができるが，CTD ルールはタスクテンプレートの自然な拡張として定義していることが特徴である．

3. CTD 抽象化

本節では，本論文で提案する CTD 抽象化について説明する．まず，既存のクラウドソーシングプラットフォームで広く利用されているタスクテンプレートの説明を行い，次に，CTD 抽象化の概要を説明する．最後に，CTD 抽象化の詳細について説明する．

3.1 タスクテンプレート

商用クラウドソーシングプラットフォームでは,「タスクテンプレート」の形式で，クラウドソーシングを行いたいタスクを登録する事が一般的である．例えば，図 1 は，今月何本の映画を見たか作業者に自由記述で入力してもらうタスクの MTurk のタスクテンプレートである．MTurk におけるタスクテンプレートは XML で記述されるが，本質的には，タスクの画面に表示すべき文章や，その結果の型などが書かれているものがタスクテンプレートである．QuestionContent 要素には質問文を書き，AnswerSpecification 要素には結果の値の型を記述する．タスクテンプレートの記述に基づいて**タスクプール**にタスクが登録され，ワーカはそのタスクプールに入っているタスクを処

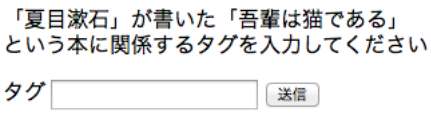

図 2　クラウドソーシングの作業例：本に関するタグ付け

Condition	$Book(id : book\_id, title, author)$	
Task	Type	Entry(tag[“タグ”]:text)
	Question	「$author」が書いた「$title」という本に関係するタグを入力してください
	Count	3
Data	$Tag(book\_id, tag)$	

図 3　本のタグ付けの場合の CTD ルール

理することになる．

MTurk では，さらに，変数 (プレースホルダ) を中に記述することによって，このタスクテンプレートの一部を，書き換えたタスクを複数登録することが出来る．この場合，変数中に書かれた値によって複数のタスクが生成され，それらがタスクプールに登録されることになる．

3.2　CTD 抽象化の概要

本節では，まず CTD 抽象化の構成要素である CTD ルールの直感的な説明を行い，次に複数の CTD ルールを利用したやや複雑なクラウドソーシングシステムの記述を説明する．CTD 抽象化の定義については，3.3 節に示す．

3.2.1　CTD 抽象化と CTD ルール

CTD 抽象化とは，クラウドソーシングシステムを CTD ルールの集合として記述することである．CTD 抽象化では，リレーショナルデータベースの存在を仮定する．次の例で示すように，CTD ルールからデータベースに格納されているリレーションを参照・更新可能である．

例 1．単純なクラウドソーシングシステム

図 2 のようなタスクを用いて，画面に示された本のタイトルと著者名を見て，関係するタグをワーカに入力してもらうクラウドソーシングシステムを考える．具体的には次のようなものである．

- データベースに，本の情報を格納したリレーション $Book(id, title, author)$ が存在する．
- ワーカは，$Book$ リレーションに格納されている本ごとにタグを入力する．
- その結果はリレーション $Tag(book\_id, tag)$ に格納する．

□

例 1 のクラウドソーシングシステムは，図 3 の CTD ルールひとつだけで記述できる．CTD ルールは，タスクテンプレート (Task) に加えて，そのタスクを生成するための条件 (Condition) と，タスクを行った結果として得られるデータ (Data) を合わせた，3 つの要素から構成されたものである．

- Condition には，タスクテンプレートに書かれたタスクを生成し，タスクプールに登録するための条件を記述する．具体的には，DB 中のリレーションに，指定された条件を満たすタプルが存在するときにタスクを生成する事を表す．例えば，図 3 では，$Book$ リレーションにタプルが存在する場合にタスクを生成する事を表す．Condition の記述方法などの詳細については 3.3 節で説明する．

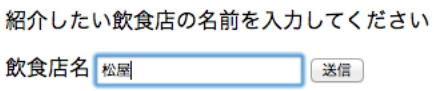

図 4　クラウドソーシングの作業例：飲食店の入力 (タスク 1)

- Task はタスクテンプレートの指定である．タスクテンプレートでは，ワーカへのタスクの指示文と，タスクの結果を格納する変数を指定する．図 3 では，後述するシステム Crapid での構文に従って記述している．この構文においては，タスクの指示文は Question 項目に書かれており，タスクの結果を格納する変数 (tag) は，Type 項目に書かれている．3.3.2 節で説明するが，Type 項目に書かれるものはタスクテンプレートの「ひな形」とよばれ，データの入力方法などを指定することにより，タスクテンプレートの簡単な記述を支援するものである．図 3 では，タスクテンプレートのひな形として，データのキーボード入力を示す Entry が指定されている．ここでは，ひな形の引数として，(1) 入力フォーム画面には “タグ” と表示すること，(2) 値の型は text であること，(3) 結果は変数 tag に格納すること，が指定されている．Count はタスクの生成数である．
- Data には，結果を格納するリレーションとその属性を指定する．図 3 では，本の id と入力されたタグを Tag リレーションに格納することが記述されている．

以上の様に，CTD ルールは，一般的なタスクテンプレート記述の一般化になっている．直感的に言うと，CTD ルールは，次の 4 つのタイプの処理を記述することが可能である．

(1) ある条件によってタスクを生成し，タスク処理の結果をデータベースに保存する処理．

(2) (Condition 部を省略すると) 無条件でタスクが生成され，タスク処理の結果をデータベースに保存する処理．

(3) (Task 部を省略すると) ある条件が満たされたとき，機械処理を行い，その結果をデータベースに保存する処理．

(4) (Condition 部，Task 部を共に省略すると) 無条件で機械処理を行い，その結果をデータベースに保存する処理．

3.2.2　より複雑なクラウドソーシング記述

本節では，複数の CTD ルールによって記述されるより複雑なクラウドソーシングの例として，飲食店の評価を行うクラウドソーシングシステムを考える．作業者が行うタスクは次の 2 つである．

タスク 1:　飲食店の入力 (図 4)．何店舗でも入力できる．

タスク 2:　入力された飲食店に対しての 5 段階での評価 (図 5)．各店につき 3 人が入力する．

ここでは，タスク 1 の結果として得られるデータをリレーション $Restaurant(id, name)$ に格納し，タスク 2 の結果として得られるデータをリレーション $Rating(restaurant\_id, value)$ に格納するとする．このとき，タスク 1 の CTD ルールを図 6 に，

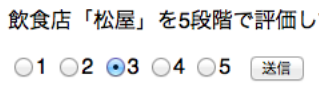

図 5　クラウドソーシングの作業例：飲食店の評価 (タスク 2)

Condition		
Task	Type	Entry(name[“飲食店名”]:text)
	Question	紹介したい飲食店の名前を入力してください.
	Count	*
Data	*Restaurant*(*name*)	

図 6　タスク 1. 飲食店の入力の CTD ルール

Condition	*Restaurant*(*id*, *name*)	
Task	Type	Choice(value,[1, 2, 3, 4, 5])
	Question	飲食店「$name」を 5 段階で評価してください.
	Count	3
Data	*Rating*(*restaurant_id* : *id*, *value*)	

図 7　タスク 2. 飲食店の評価の CTD ルール

タスク 2 の CTD ルールを図 7 に示す.

タスク 1 の Condition 部には，何も指定されていない．この場合，タスクは無条件に生成される．また，Task 部にて Count 属性に*を指定しているが，これは無限回タスクを生成することを表す.

タスク 2 の Condition 部では，*Restaurant* に格納されたタプル毎に，タスクを生成する事を指定している．したがって，タスク 1 で入力された飲食店毎にタスク 2 が生成されることになる.

以上のように，CTD 抽象化では，CTD ルールの集合としてクラウドソーシングシステムを記述する．これらは宣言的な記述であり，各タスクはルールの Condition の条件が成立したときに生成される．クラウドソーシングではワーカによる非同期な並列作業が一般的であるため，この程度の例であっても，宣言型記述が手続き型と比較してより自然であることがわかる.

3.3　CTD 抽象化の詳細

本節では，まず CTD 抽象化の定義を示し，次に本論文において簡潔にルールを書くための構文上の糖衣について説明する.

3.3.1　CTD 抽象化の定義

定義 1. プログラムを CTD (Condition-Task-Data) ルール $R_i = (C_i, T_i, D_i)$ の集合で記述することを CTD 抽象化と呼ぶ. ここで，C_i は T_i を生成するための条件，T_i はタスクの記述，D_i は T_i の処理結果として得られるデータであり，詳細は下記で定義する.

- C_i は 0 個以上の原子式の並び $P_1(x_{11}, \ldots, x_{1n_1}), \ldots, P_m(x_{m1}, \ldots, x_{mn_m})$ である．原子式のいくつかは算術原子式 ($x_{11} = 3$ 等) であっても良い.
- T_i は，3 つ組み $(d, \bar{x}, \bar{y})$ もしくは *null* である．ここで，d はワーカへのタスク指示，$\bar{x}$ は C_i にて束縛されている変数の並び，$\bar{y}$ はタスクの結果を格納する変数の並びである.
- D_i は，原子式の並び $Q_1(y_{11}, \ldots, y_{1n_1}), \ldots, Q_m(y_{m1}, \ldots, y_{mn_m})$．ただし，$y_{jk}$ は C_i にて束縛される変数，$\bar{y}$ 中の変数，もしくは定数のいずれかである．また，各原子式の後ろに/update もしくは/delete をつけることが可能である.

□

3.3.2　CTD ルール構文上の糖衣

式を簡潔に書くため，CTD ルールに対していくつかの構文上の糖衣を導入する.

(1) 原子式の属性. Prolog や Datalog と異なり，CTD ルールの記述における Condition および Data 部に現れる原子式の記述では，その引数の記述において明示的に属性名を記述する．具体的には，次の様に記述する.

- 引数指定では，“属性名:変数名もしくは値” と記述する.
- 属性を参照しない場合，“属性名：変数名” を省略できる.
- 変数名と属性名が同じ場合，属性名を省略できる.

例えば，スキーマ `Restaurant(name, zipcode)` を持つリレーションが存在するとき，`Restaurant(name:x, zipcode:305)` や `Restaurant(name:y)` は，そのリレーションに関する原子式である．また，`Restaurant(name:name, zip:y)` は `Restaurant(name, zip:y)` と省略できる.

(2) Task 部の記述. Task には頻出パターンが存在するため，繰り返しを避けるためにタスクテンプレートのひな形 Type と Count を導入する.

- Type には，利用するタスクテンプレートのひな形の名前と引数を記述する．本論文では，Crapid (5. 節で説明) がサポートするタスクテンプレートのひな形の存在を仮定する．ひな形を利用する事により，実際のタスクテンプレートは，ひな形の指定と，Question 部に書かれた質問文から導出される.
- Count にタスクの生成数 N が指定されているときには，CTD ルールを N 個コピーする.

3.4　CTD 抽象化の計算モデル

CTD 抽象化によるクラウドソーシング記述 d (CTD ルールの集合) が与えられたとする．その時，d の計算とは，データベースの状態に応じた CTD ルールのボトムアップな評価の事である．具体的には次の様に行われる.

- データベースの値を参照し，各 CTD ルール 各 $(C_i, T_i, D_i) \in d$ の C_i の評価を行う．具体的には，C_i の条件を満たすような，変数への束縛が存在しうるタプルがデータベース中に存在するかどうか，C_i 中の左の原子式から順に評価を行う.
- 条件部 C_i の原子式が全て成立するものがある場合，次を行う.
  - $T_i \neq null$ であれば，タスク T_i を生成し，タスクプールに登録する.
  - $T_i = null$ であれば D_i に応じてデータベースへのデータの追加，更新 (/update 指定時)，削除 (/delete 指定時) を行う.
- T_i が処理されれば，D_i に応じてデータベースへのデータの追加，更新 (/update 指定時)，削除 (/delete 指定時) を行う.
- データの変更がある度に，C_i が成立するルールが新たに

Condition	$Image(i, size, type : \text{“}photo\text{”}), Large(size)$	
Task	Type	Choice(category, [landscape, portrait, animal, food, etc.])
	Question	$i のカテゴリを選択して下さい.
	Count	1
Data	$LargePhoto(i, category)$	

図 8　複数の原子式から成る Condition を持つ CTD ルール

ルール 1

Condition	`MachineCurrentStatus(st, head),`
	`Tape(pos, sym),`
	`Rule(st, sym, new_st, new_sym, dire),`
	`new_pos = pos + dire`

Task	-

Data	`MachineCurrentStatus(st, head)/delete,`
	`MachineCurrentStatus(`
	`  st:new_st, head:new_pos`
	`)/update,`
	`Tape(pos, sym:new_sym)/update`

ルール 2

Condition	`MachineCurrentStatus(head)`
Task	-
Data	`Tape(pos:head)/update`

図 9　チューリングマシンを記述する CTD ルール

生じればその処理を行い，そのようなルールがなくなるまで処理を続ける．

例えば，図 8 に示す CTD ルールでは，まず，$Image(i, size, type :$ ”$photo$”) を評価し，$Image$ 中の $type$ が “photo” であるタプルによって i と $size$ が束縛される．次に，束縛された $size$ によって $Large(size)$ が成り立つかを確認する．そして，成り立つ場合には束縛された i に対してカテゴリを選択するタスクが生成され，その処理結果が $LargePhoto$ に格納される．

4. CTD 抽象化の表現力

本節では，CTD 抽象化の表現力について議論する．まず，CTD 抽象化による記述がチューリング完全である事を示す．次に，チューリング完全性とは独立したプログラミング言語の表現力の尺度を導入する．これは，その言語で表現可能な人と計算機のインタラクションに着目したものである．いくつかのクラスを定義し，各クラスで記述可能なプログラムが有意に異なる事を示す．最後に，CTD 抽象化と他のクラウドソーシング開発支援ツールの表現力を比較する．

4.1 CTD 抽象化のチューリング完全性

定理 1. CTD 抽象化はチューリング完全である．　□

証明. 図 9 に CTD 抽象化によるチューリングマシンの記述を示す．

チューリングマシンは，次の 3 つの構成要素から成る．

要素 1　機械の内部状態を記憶するメモリ

要素 2　テープに格納された情報を読み書きするヘッド

要素 3　無限に長いテープ

要素 1 は，リレーション $MachineCurrentStatus(st, head)$ の st に機械の内部状態を格納することによって表現する．

要素 2 は，$MachineCurrentStatus(st, head)$ の head にヘッドの位置を格納することによって表現する．

要素 3 は，リレーション $Tape(pos, sym)$ とルール 2 によって表現する．$Tape(pos, sym)$ はテープ上のある位置 pos に記号 sym が書かれていることを格納する．ルール 2 は無限に長いテープを表現するためのルールである．リレーションにあらかじめ無限個数のタプルを格納することはできないので，ヘッドが初めてその位置を指したときに記号が空白であるタプルを生成することにした．その処理はルール 2 によって表現され，テープを無限に伸ばすことを可能にする．この表現によって無限に長いテープを用意するのと同等の処理が可能である．

図 9 では，ルール 1 に遷移規則に基づいたチューリングマシンの動作を表現している．リレーション $Rule(st, sym, new\_st, new\_sym, dire)$ は遷移規則を表す．機械の内部状態が st かつ，ヘッドの現在位置の記号が sym のとき，現在位置に記号 new_sym を書き込み，内部状態を new_st に移し，$dire$ 方向にヘッドを移動するという規則である．

以上により，CTD 抽象化によってチューリングマシンを記述できるので，CTD 抽象化はチューリング完全である．　□

4.2 人とのインタラクションに着目したプログラミング言語の表現力の尺度

本論文では，プログラムがどのような人とのインタラクションを記述できるかに着目した，プログラミング言語の表現力の尺度を提案する．まず，その言語によるプログラムがどのような人とのインタラクションを記述できるかによって，処理可能なプログラムのクラスを分類し，これらが有意に異なるクラスである事を示す．

定義 2. 全ての入力者が他者の入力に影響されずデータ入力を行うプログラムのクラスを $\mathcal{G}_1$ と呼ぶ．　□

$\mathcal{G}_1$ に属するプログラムの例として，アンケートを行うプログラムが挙げられる．例えば，好きな食べ物を聞くプログラムでは入力者は他者の入力に影響されずにデータ入力を行う．

定義 3. 入力のタイミングが 2 段階に分かれており，後者が何を入力するかが前者の入力に影響されるようなプログラムのクラスを $\mathcal{G}_2$ と呼ぶ．また，その一般化である入力のタイミングが i 段階 $(i > 1)$ に分かれており，i 段階の入力が $i-1$ 段階以前の入力に影響されるようなプログラムのクラスを $\mathcal{G}_i$ と呼ぶ．□

$\mathcal{G}_i$ に属するプログラムの例として，前者の入力の検証を含むプログラムが挙げられる．例えば，好きな食べ物を聞くプログラムでは，各入力者によって表記ゆれが発生し，正しい集計が行うのが難しい．その場合，表記ゆれを正すための検証のタス

クが必要となり，プログラムはアンケートを取るタスクと検証を行うタスクの 2 段階の入力作業を行うため，$\mathcal{G}_2$ に属するものとなる．

定義 4. $\mathcal{G}_N$ に属するプログラムの繰り返しであり，繰り返し回数 (すなわちデータ入力の回数) の最大数を事前に決めることが不可能であって，各入力の内容がそれまでのすべての入力に影響されるようなプログラムのクラスを $\mathcal{G}_*$ と呼ぶ． □

$\mathcal{G}_*$ の例として，リレー小説を作るプログラムが挙げられる．リレー小説を作るプログラムでは各入力者が交代で文章を入力する．このタスクは事前のタスクの結果に影響されるものであり，また，いつ終了するかは事前にわからない．

以上の定義をしたときに成り立つ 2 つの定理とその証明を次に示す．

定理 2. $\mathcal{G}_{i+1}$ は $\mathcal{G}_i$ より真に大きいクラスである． □

証明. 定義より，$\mathcal{G}_i \subseteqq \mathcal{G}_{i+1}$．$\mathcal{G}_i$ に属する任意のプログラム P_i を考えた時，P_i に結果を検証するタスクを加えたプログラム P_{i+1} は $\mathcal{G}_{i+1}$ に属する．$P_{i+1} \notin \mathcal{G}_i$ なので $\mathcal{G}_i \subset \mathcal{G}_{i+1}$． □

定理 3. ある定数 N が与えられた時，$\mathcal{G}_*$ は $\mathcal{G}_N$ より真に大きいクラスである． □

証明. ある有名人 A さんの似顔絵を入力するプログラム P_A を考える．P_A は，次のステップで実行される．

(1) 誰かが元となる似顔絵を描く

(2) 他の誰かがより似るように修正する

(3) 似顔絵が十分似ていると判断されるまで繰り返し，他の誰かがより似るように修正する

(4) 似顔絵が十分似ていると判断され，終了する

P_A は何段階の入力で終了するか決定できず，$\mathcal{G}_*$ に属するが，$\mathcal{G}_N$ には属しない． □

最後に，プログラミング言語の表現力のクラスを次の様に定義する．

定義 5. あるプログラミング言語が次の条件を共に満たすとき，その言語は**ゲーム完全** (Game Complete) である．

- 条件 1: クラス $\mathcal{G}_*$ のインタラクションを表現可能である．
- 条件 2:人に対する報酬を表す値を，入力されたデータの並びからチューリングマシンで計算可能である．

□

ゲーム完全は，ゲーム理論における，長さ不定の逐次ゲームのうち，チューリングマシンで表現可能なものを表すクラスである．ゲーム完全であるためには，プログラミング言語が，クラス $\mathcal{G}_*$ のインタラクションを表現可能であること，および，チューリング完全である事の両方が必要である．

定理 4. CTD 抽象化はゲーム完全である． □

証明. CTD 抽象化は次のことから $\mathcal{G}_*$ を記述可能である．

- 次の理由により条件 1 を満たす
  - $\mathcal{G}_N$ であるプログラムを全て書くことができる．
  - $\mathcal{G}_N$ であるプログラムを再帰で呼び出せる．
- チューリング完全であるため，条件 2 を満たす

□

表 1　クラウドソーシング開発支援ツールの表現力の比較

記法	チューリング完全性	記述可能な「人とのインタラクション」のクラス
MTurk 単体	×	$\mathcal{G}_1$
CrowdForge	×	$\mathcal{G}_N$ の一部（基本的なものは $\mathcal{G}_3$の一部）
CrowdDB	×	$\mathcal{G}_N$ の一部
CTD 抽象化	○	$\mathcal{G}_*$ (ゲーム完全)
MTurk を手続き型言語で書かれた外部プログラムから呼び出す	○	$\mathcal{G}_*$ (ゲーム完全)

4.3 表現力の比較

本節では，CTD 抽象化と他のクラウドソーシング開発支援ツールの表現力を比較する (表 1)．MTurk のように人とのインタラクションを一回だけ行うものは $\mathcal{G}_1$ である．また，CrowdForge のように Partition, Map, Reduce といった 3 回のインタラクションの場合は $\mathcal{G}_3$ である．CrowdForge では Partition, Map, Reduce を複数組み合わせることが出来るが，ループや再帰を表現できないので，その場合には $\mathcal{G}_N$ となる．我々が提案する CTD 抽象化は CTD ルールにより再帰を表現できるので，$\mathcal{G}_*$ クラスを表現可能である．

CrowdForge はチューリング完全ではない．MTurk を手続き型言語で書かれた外部プログラムから呼び出せば，チューリング完全かつ $\mathcal{G}_*$ となり，CTD 抽象化と同じ表現力を持つことが出来る．CrowdDB は SQL をベースとした言語を用いるためチューリング完全でなく，クラスは $\mathcal{G}_N$ である．

5. CTD 抽象化によるクラウドソーシングシステム開発ツール Crapid

本節では，我々が実装した，CTD 抽象化によるクラウドソーシング記述から実行可能なコードを出力する Crapid について説明する．まず概要を説明し，次に，3.2.2 節で紹介したクラウドソーシングシステムの例を用いて，Crapid の使い方を説明する．最後に，現在の Crapid の実装の制限について議論する．

5.1 Crapid の概要

Crapid は CTD ルールの集合を入力として受け付け，実行可能な CyLog [7] のコードを生成する．Crapid では，CTD ルールを直接テキストで入力させるのではなく，選択したタスクテンプレートのひな形に応じて，HTML フォームベースのインターフェースを提供する．

Crapid が現在サポートするタスクテンプレートのひな形を表 2 に示す．CTD ルールを記述するときにタスクテンプレートのひな形を選択すると，各ひな形に応じた引数を記述するための HTML フォームが現れる．

5.2 Crapid の利用方法

Crapid は次の 3 つのステップに従ってシステムを生成する．

表 2　Crapid が現在サポートするタスクテンプレートのひな形

Entry	引数で指定された型のデータの入力．引数は「属性名 [“画面に表示する名前”]:データの型」という形式で指定する．カンマ区切りで複数指定可能である．
Choice	引数で指定された選択肢からの選択．引数は「属性名, [値 1, 値 2, …]」という形式で 1 つ指定する．
Decision	YES, NO の判断．引数は YES, NO の結果を格納する属性名を 1 つ指定する．
Comparison	値の比較．引数は「比較する属性名 1[画面に表示する属性名], 比較する属性名 2[画面に表示する属性名], 結果を格納する属性名」という形式で指定する．

(1) リレーションの登録
(2) CTD ルールの登録
(3) コードの表示

3.2.2 節の飲食店の評価を行う例を用いて，3 つのステップについて説明する．

5.2.1　リレーションの登録

このステップでは，ユーザは CTD ルール中で使用するリレーションの登録を行う．リレーションの登録にはリレーション名とその属性を登録する．属性には，必須項目として属性名，オプション項目としてキー，auto_increment の設定項目がある．

例では，レストランを格納するリレーション $Restaurant(id, name)$ と評価を格納するリレーション $Rating(restaurant\_id, value)$ を登録する．登録画面を図 10 に示す．ユーザは，まず，*Restaurant* の登録のために，「リレーション名」に Restaurant，「属性名」に id, name を入力する．id は主キーかつ auto_increment なので，それぞれのチェックボックスにチェックを入れ，「作成」ボタンを押す．次に，*Rating* の登録のために，「リレーション名」に Rating，「属性名」に restaurant_id, value を入力し，「作成」ボタンを押す．属性を複数入力する際には「属性の追加」ボタンをクリックする．

5.2.2　CTD ルールの登録

このステップでは，ユーザはシステムを記述する CTD ルールの登録を行う．例では，図 6 および図 7 に示した CTD ルールを登録する．登録画面を図 11 に示す．ユーザはまず，レストランの入力のタスクを表す CTD ルールを登録する．Type のセレクトボックスでは「Entry」を選択し，テキストボックスに「name[”飲食店名”]:text」を入力する．そして，Question に「紹介したい飲食店の名前を入力してください」，Count に「*」，Data に「Restaurant(name)」を入力し，最後に追加ボタンを押す．次に，評価のタスクを表す CTD ルールを登録する．Condition に「Restaurant(id, name)」を入力する．Type のセレクトボックスでは「Entry」を選択し，テキストボックスに「value, [1,2,3,4,5]」を入力する．そして，Count に「3」，Data に「Rating(restaurant_id:id, value)」を入力し，最後に追加ボタンを押す．

5.2.3　コードの表示

このステップでは，CTD ルールの集合から生成された CyLog [7] のコードを表示する．例では図 12 に示したコードが表示される．Crapid によって生成された CyLog コードは，クラウドソーシングプラットフォーム Crowd4U [8] で実行可能である．

図 10　リレーションの登録

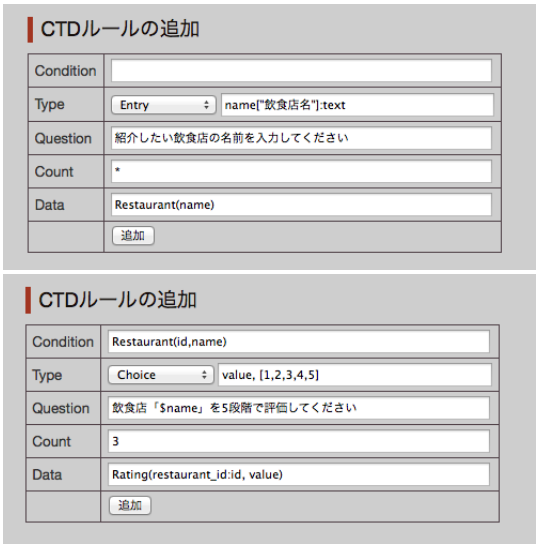

図 11　CTD ルールの登録

5.3　現在の Crapid 実装における制限

現在，Crapid では，各タスクテンプレート毎にワーカへの固定報酬 (ポイント) を指定できる．一般には報酬は，クラウドソーシングシステムを構成する要素として重要な要素であり，入力されたデータなどに応じた報酬の変更が重要であると考えられるが，Crapid における，より柔軟な報酬の指定は今後の課題である．

6.　まとめと今後の課題

本論文は，宣言的記述によるクラウドソーシングシステムの開発を支援するため，CTD 抽象化およびそれに基づく開発支援ツール Crapid の提案を行った．CTD 抽象化は，クラウド

CyLogコードの表示

```
schema:

Restaurant(
  id  int  :auto_increment;
  name  text;
) key (id);

Rating(
  restaurant_id  text;
  value  text;
);

!Task1(
  id  int;
);

!Task2(
  name  text;
  id  text;
);

rules:

  Restaurant(name)/open;

  !Task1(id) <- ?Restaurant(id);

  Rating(restaurant_id:id, value)/open <- Restaurant(id,name);

  !Task2(name,id) <- Restaurant(name, id);

views:

!Task1(id){
  <p>紹介したい飲食店の名前を入力してください</p>
  <form fact=Restaurant(id, name) move=!Task1(id)>
    飲食店<input type="text" value="" name="name" />
    <input type="submit" />
  </form>
}

!Task2(name, id){
  <p>飲食店「$name」を5段階で評価してください</p>
  <form fact=Rating(restaurant_id:id, value) move=!Task2(name, id)>
    1<input type="radio" value="1" name="value" /><br />
    2<input type="radio" value="2" name="value" /><br />
    3<input type="radio" value="3" name="value" /><br />
    4<input type="radio" value="4" name="value" /><br />
    5<input type="radio" value="5" name="value" /><br />
    <input type="submit" />
  </form>
}
```

図 12　生成された CyLog コード

ソーシングプラットフォームにおいてよく利用されるタスクテンプレートを一般化したものであり，十分な記述力を持つ．また，Crapid は様々なタスクテンプレートのひな形を提供することにより，効率よいクラウドソーシングシステム開発を支援する．

今後の課題としては，クラウドソーシングの頻出パターンを考慮したタスクテンプレートのひな形ライブラリの構築が挙げられる．また，CTD 抽象化によって記述されたコードの最適化や，データクオリティなどの要件を考慮したコードの変形等も今後の課題である．

謝　　辞

本論文の一部は JST さきがけ「情報環境と人」による．

文　　献

[1] Doan AnHai, Ramakrishnan Raghu, Halevy Alon Y.. “Crowdsourcing systems on the World-Wide Web. Commun.ACM. 2011, vol. 54, no. 4, p. 86-96.

[2] gwap.com. “ESP Game”. http://www.gwap.com/gwap/gamesPreview/espgame/

[3] reCAPTCHA. “What is reCAPTCHA?”. http://recaptcha.net/learnmore.html

[4] Amazon Mechanical Turk, https://www.mturk.com/.

[5] Little,G.,Chilton, L.B., Goldman, M.,and Miller, R.C. “Turkit: human computation algorithms on mechanical turk”. In Proceedings of UIST (2010), ACM, New York, 57-66.

[6] Franklin Michael J., Kossmann Donald, Kraska Tim, Ramesh Sukriti, Xin Reynold.. “CrowdDB: answering queries with crowdsourcing”. SIGMOD Conference. 2011, p. 61-72.

[7] Atsuyuki Morishima, Norihide Shinagawa, Shoji Mochizuki. “The Power of Integrated Abstraction for Data-centric Human/MachineCurrentStatus Computations.” First International Workshop on Searching and Integrating New Web Data Sources (VLDS2011) Co-located with VLDB 2011, pp. 5-9.

[8] Atsuyuki Morishima, Norihide Shinagawa, Tomomi Mitsuishi, Hideto Aoki, Shun Fukusumi. “CyLog/Crowd4U: A Declarative Platform for Complex Data-centric Crowdsourcing”. PVLDB 5(12): 1918-1921 (2012).

[9] A. Kittur, B. Smus, and R. E. Kraut, “CrowdForge: Crowdsourcing Complex Work”, Technical Report. CMUHCII-11, 2011.

[10] Patrick Minder and Abraham Bernstein.. “A Programming Language for the Systematic Exploration of Human Computation Systems”, Fourth International Conference on Social Informatics (SocInfo 2012), 2012.