Babascript:

Babascript:

匿名で査読を行うため著者名なし *

Summary.
プログラムはコンピュータを制御できるが、人は制御できない。人をプログラムから制御できれば、コンピュータのみでは実現出来なかった処理も実現可能となる。そこで、本論文では、プログラムに人への処理命令を組み込むことのできるプログラミング環境 Babascript を提案する。Babascript では、人とコンピュータはプログラム上で対等な処理ノードとなり、相互補完的に動作することで、人にとっての処理の最適化を実現する。また、Babascript フレームワークによって実現する応用例を示し、起こりうる問題について考察する。

1 はじめに

プログラムはコンピュータを制御できるが、人を制御することはできない。人はコンピュータに比べ、とても柔軟な能力を有する。単純な演算能力や記憶能力だけで言えば、コンピュータのほうが優れていることもあるが、場の空気を読んだり、人とのコミュニケーションは人のほうが得意であると言える。

人は、コンピュータにはない貴重な能力を有していると言えるが、人をプログラムから制御出来るような仕組みの研究はクラウドソーシングの分野に限られている。クラウドソーシングはインターネットを介した不特定多数の人をリソースとして扱うものであり、利用可能な能力は限られている。不特定な人ではなく、特定の人をプログラムから操作できるようになれば、プログラムが干渉可能な領域は更に広がると考えられる。

本論文では、プログラムに人への処理命令を組み込むことのできる Babascript プログラミング環境を提案する。Babascript 環境では、人を処理ノードのうちの一つとして扱い、人を含んだ一連の処理をプログラムとして記述する。コンピュータと人が従順な処理ノードとして、プログラムからの命令を積極的に処理していくことによって、相互補完しながら全体の処理を最適化できると考えられる。つまり、コンピュータはその演算能力を最大限に活かし人を助け、人はその柔軟な思考力や実世界への干渉力を活かしてコンピュータにはできない処理を実現する。こういった仕組みによって、人にとっての処理の最適化を狙う。また、その例として「仕事のプログラム化」と「人力実世界プログラミング」を示す。

* 匿名で査読を行うため所属名なし

2 Babascript プログラミング環境

Babascript 環境では関数呼び出しによって人に命令を送る仕組みによって、プログラム内での人への処理命令を実現する。通常のプログラムでは、関数に引数を与え実行することで、処理結果を返り値として得ることができる。Babascript では、同様に関数呼び出しし、処理を人に実行させ、返り値を処理結果として返す。つまり、同じ関数呼び出しだが、その実行主体を人間にすることによって、プログラム内における人力処理の組み込みを実現する。

このような仕組みを実現するために、人への命令構文を組み込んだライブラリ Babascript と、命令に対して返り値を返すためのクライアントライブラリ Babascript Client、クライアントライブラリを組み込んだ Web アプリケーションを実装した、また、プラグイン機構によってその機能を拡張可能にした。

```
1 BabaScript = require("babascript");
2 baba = new BabaScript("takumibaba");
3 baba.書類整理をする(function(tesult){
4   // 返り値を得た後の処理を記述する
5 });
```

図 1. babascript プログラムの基本命令文

2.1 BabaScript

Babascript は、関数呼び出しによって人に処理命令を送れるオブジェクト (以下、人オブジェクト) を宣言可能にする DSL だ。人オブジェクトにおいて定義されていない全てのメソッドが人への命令として解釈される。また、命令に対して人からの返り値を得ると、実行メソッドの引数で指定したコールバック関数を実行する。

2.1.1 人オブジェクトの宣言と処理命令構文

図 1 のようなプログラムによって、人に処理命令を送ることができる。メソッド名がそのまま命令内容となる。

人への命令構文は、実行されるとメソッド名と引数を元にした json オブジェクトへと変換され、この json オブジェクトが処理タスクとしてクライアントライブラリに配信される。

人オブジェクトは宣言時に ID を指定することによって、誰に対して処理命令を配信するかを決定する。例えば、id:baba に命令を送りたければ、人オブジェクト宣言時の第一引数には baba という文字列を指定する必要がある。また、クライアント側でも同様に、id:baba として宣言する必要がある。

この際、指定した ID を監視するクライアントが複数人だった場合は、人への命令構文を実行時に他のタスクを実行していないクライアントへタスクが配信される。

人への処理命令構文の第一引数には基本的な処理情報に加えてクライアント側に送信するオプション情報を指定し、第二引数には、人から処理命令に対する返り値が得られた際に実行するコールバック関数を指定する。コールバック関数実行時には、その引数に返り値と関連した情報を格納したオブジェクトが与えられる。

オブジェクトに定義されていない全ての関数は人への処理命令構文として解釈される。この機能は methodmissing という、定義されていないメソッドが実行された際にその処理を記述しておく仕組みによって実現する。

2.1.2 オプション情報の付加

人への処理命令構文の第一引数に与えるオプション情報には、返り値の型指定などの情報が考えられる。返り値としてあらゆる型を想定したプログラムを記述することは難しい。例えば Boolean 型で返してほしい、といった時には図 2 のようなプログラムが考えられる。

```
1 BabaScript = require("babascript");
2 baba = new BabaScript("takumibaba");
3 baba.書類整理をする(function(tesult){
4   // 返り値を得た後の処理を記述する
5 });
```

図 2. オプション情報

他にも、指定したリストの中から値を選択してほしい、といった命令の場合は、そのリストをオプション情報として付加することが考えられる。

特別なオプション情報として、broadcast オプションが存在する。broadcast 機能は、指定した ID を監視する全てのクライアントへとタスクを配信し、指定した数の返り値を得られると処理を終了し、コールバック関数を実行するといったものだ。

2.2 Babascript Client

Babascript Client は、Babascript からの命令受信と処理結果の送信機能を実現するクライアントライブラリだ。命令受信のイベントに対してコールバック関数を指定することで、処理命令を受け取ることができる。基本的な利用方法は、図 3 に示す。

```
1 Client = require("babascript-client");
2 client = new Client("takumibaba");
3 client.on("get_task", function(task){
4   // プログラムからタスクを受け取った時の
5   // 挙動を記述する
6 });
```

図 3. クライアントライブラリの基本機能

クライアントオブジェクトの宣言時、ID を指定することによって、ID に対して処理命令が発行された際にタスクを受信することができる。また、クライアントオブジェクトが持つ “returnValue” メソッドを利用することで、返り値を命令発行元に返すことができる。

クライアントライブラリは、UI と完全に分離した実装となっており、かつ利用方もシンプルだ。後述の Web アプリケーションだけでなく、様々なアプリケーションに組み込むことができる。

2.3 Web アプリケーション

BabascriptClient が得たタスクをユーザに提示するためのインタフェースとして、Web アプリケーションを実装した。タスク情報において型指定をすることによって、ユーザに提示する UI を変化させ、型に合った返り値を選択できるようにしている。例えば、Boolean 型を指定していた場合、ユーザには true ボタンと false ボタンが提示され、どちらかを押すと、その結果が返り値としてプログラムに返される。また、String 型を指定すれば、文字列の入力フォームと Submit ボタンが表示される。

スマートフォンに最適化した Web アプリケーションとして実装しており、様々な場面において利用可能であり、実世界におけるタスクを処理しながらでも十分に利用可能である。図 4 のような画面を持つ。

2.4 Plugin

Babascript 及び Babascript Client に組み込むことのできるプラグイン機構を実装した。Babascript 及び BabascriptClient が発行するイベントとデータを受け取り、その都度処理を実行させることができる。現在、以下のようなイベントが存在する

図 4. Web アプリのインタフェース

- init(プラグイン読み込み時)
- connect(ネットワーク接続時)
- send(タスクの送信時)
- receive(タスクの受信時)

具体的には、図 5 のようにプラグインを読み込むことができる。Script 側も Client 側も、対応していれば同じプラグインを読み込ませて動作させることができる。

```
// Script側
baba = new Babascript("baba")
baba.set(new Logger())

// Client側
client = new BabascriptClient("baba")
client.set(new Logger())
```

図 5. プラグインの読み込み

例として、以下のようなプラグインの実装が挙げられる。

- ログコレクター
- データ同期
- ユーザ管理

2.5 script-client 間の通信

script と client のタスク送受信のために、node-linda[2] を利用した。script, client 共に、この node-linda に接続するためのライブラリを組み込んでいる。

script は、プログラム実行時に node-linda サーバに接続し、人への命令構文を実行するごとに、タスクを node-linda サーバに書き込む。書き込み終了後、そのタスクと同じ ID を持った返り値タスクが書き込まれるまで、node-linda サーバの監視を行う。

client は、プログラム起動中は常に node-linda サーバに接続し、指定した ID を監視し、タスク受信まで待機する。node-linda サーバに同一 ID にタスク書き込みが行われた際、そのタスクを取得する。このタスクは、broadcast オプションが付いている場合は、同一 ID を監視している全てのクライアントがタスクを受信する。通常の実行の場合は、キューの最初のクライアントのみがタスクを受信・実行し、他のクライアントは待機を継続する。client は、タスク実行中は通常のタスクを受信することはなく、タスク実行が終わって受信待ち状態になる際に新たにキューに追加される。このため、複数のクライアントが一つ ID を監視している状況であっても、一つのクライアントにタスクが集中するといったことはない。

2.6 処理の流れ

Babascript 環境は、以下のようなフローで一回の人への命令構文が実行される。

1. 人への命令構文を実行する
2. 命令構文に応じたタスクが生成される
3. タスクが Node-Linda サーバを経由してクライアントへと配信される
4. タスクを受け取ったクライアントがユーザに処理を促す
5. タスク実行者が、処理結果を入力する
6. 処理結果を元に返り値データが生成される
7. Node-Linda サーバを経由して実行元プログラムに返り値が送信される
8. 指定されたコールバック関数が実行され、処理が続く

2.7 実装

上記システムは全て Javascript で実装した。Babascript は Node.js 上で動作し、Babascript Client は Node.js と Web ブラウザ上で動作する。全体図は図 6 のとおりだ。

2.7.1 タスク情報の構成

人への処理命令構文は実行によってタスク情報を生成する。このタスク情報は、図 7 のような json オブジェクトとなる。

2.7.2 実行結果の待ち方

人への命令構文に対する返り値が、適切に返り値として戻されるようにするために、命令ごとにユニークな ID を生成している。ユニーク ID は、人への命令構文実行時の UNIX time と、ランダムな数値を結合した文字列である。図 7 における json オブ

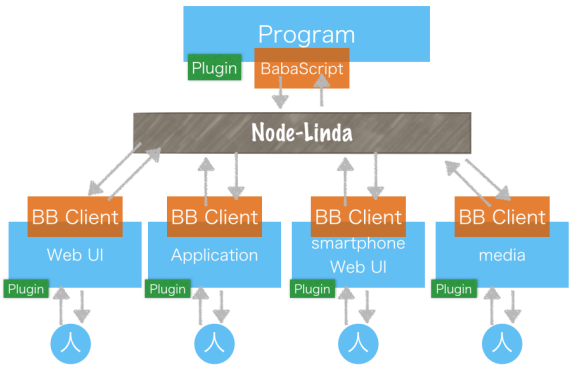

図 6. システム全体図

```
task = {
  "name": "baba", // 命令配信先ID
  "type": "eval", // 配信タイプ
  "key": "部屋の温度は適切ですか", // 処理命令の内容
  "cid": "1409227153_0.869041177676", //タスクID
  "option": { // オプション情報
    "format": "boolean",
  }
};
```

図 7. タスクの JSON オブジェクト

ジェクトの'cid' の項目に示されているものだ。このユニーク ID を伴った返り値が戻ってくるか、指定したタイムアウト時間に到達するか、タスクキャンセルが発生するまで、

3 応用例

以下のような応用が考えられる。

3.1 仕事のプログラム化

人への行動命令がプログラムとして記述可能になることで、人の仕事をプログラム化することができる。プログラム化し、実行することによって、状態をプログラムによって管理することが可能となる。つまり、条件判断や記憶が必要なことは出来るだけコンピュータ側に処理させる、といったことが可能にあんり、人は細かい条件などを覚えておいたり、自分で判断することなく、ただ指示されたことのみを実行することで、仕事を終わらせることができるようになる。ただ指示されたことのみを実行するだけで良いならば、仕事の引き継ぎなども必要なくなり、人の代替も容易となる。

また、人同士がコミュニケーションを取ることなく、複数人を協調させるといったことも可能となる。通常、複数人を協調させるためには、人同士が相談したり、上位の意思決定者が必要となる。しかし、コミュニケーションはコストのかかるものであり、適切に行われない場合、問題が生じることもある。意思決定をプログラムに委ねることは、複数人を効率よく協調させることに繋がるとも考えられる。

全ての仕事をプログラム経由にすることで、詳細な実行ログや実行状況を把握することもできる。仕事の実行量の定量化や、状況監視、状況の可視化などに活かすことができる。

3.2 人力実世界プログラミング

人をセンサーやアクチュエータとして利用することで、人によって実世界を操作する実世界プログラミングが可能となる。現在のセンサーやアクチュエータでは、実世界に干渉するのには限界がある。例えば、その場の雰囲気を数値化したり文字列化するといった、コンテキスト情報を伴ったセンシングは困難である。人というセンサーをプログラムから利用できるようにすることで、今までには実現しなかった処理が実現する。

4 考察

Babascript 環境について、以下のように考察する。

4.1 処理単位としての人

本研究では、人は処理を命令され実行するノードとなり、プログラム上においてコンピュータやセンサーと同等となる。こういったことに対して心理的な拒否感を覚えることも考えられる。しかし、一処理ノードとして扱うことによって、大きなメリットもある。人とコンピュータ、処理に応じて適切なほうに実行させることができれば、人は人にしかできないことや人がやるべき処理にのみ集中するといったことも実現できる。また、ただやるべきことのみを命令されて動くということは、自分で深く考えて行動する必要がないということだ。とても楽なことでもある。

4.2 タスク実行の遅延と実行保障性

Babascript によってタスク実行を依頼しても、人がすぐにタスクを実行し値を返すことを完全に保証することはできない。タスク受信端末を見ていない、受信しても実行できないといった状況の場合、すぐに値を返すことはできない。この際、Babascript による処理が原因で全体の処理が遅延する可能性がある。

また、労働関係にあるなど、タスク実行に強制力がある場合は、タスク実行が確実に行われると考えられるが、強制力がない場合はそもそもタスク受信を無視するといったことも考えられる。タスク実行に強制力がない場合は、金銭などのインセンティブを与えるといった手段によって、実行保障性を確保するといったことが考えられる。

4.3 例外処理

Babascript 環境において、命令文と現実との乖離によって適切な返り値を選択・記述ができなくなるといった可能性がある。これは、現実が刻々と変化していることなどから、完全に避ける事の出来ない問題であると考える。この際、無理やり値を返すといった処理をしてしまうと、本来の状態とは違った判断がなされてしまう危険性がある。命令文とは明らかに現実が異なっている場合などは、タスク実行者から例外としてプログラムに通知出来るような仕組みの実装によって、問題の解決へと繋げられると考える。

4.4 命令内容の粒度

Babascript では、タスクの命令文の記述には制限がないため、自由となっている。抽象度が高すぎる命令は、あいまいな表記となり、タスク実行者にとって理解しづらい文面となり得る。その結果、想定外の処理が実行され、意図しない結果を招く恐れがある。抽象度が低すぎる命令は、全体の処理内容にもよるが、プログラム自体が冗長となり得る。プログラムとタスク実行者の間のやりとりが増え、通信や待機時間などがボトルネックとなる可能性がある。また、タスク実行者にとっても、やりとりが増えることで負担増になると考えられる。処理ごとに異なると考えられるが、命令文は適切な抽象度に設計しなくてはならない。

4.5 複数命令の同時実行

複数のプログラムから同時に一人のタスク実行者へとタスクが配信される可能性がある。この際、異なるコンテキストにある命令が交互に配信され、タスク実行に大きな障害をもたらす可能性がある。例えば、料理プログラムと掃除プログラムが同時に実行された場合、鍋で煮ている途中で「洗剤を投入しろ」などといった命令が配信されることが考えられる。

この問題は、全ての Babascript プログラム中において、一人のタスク実行者は一つのプログラムからのみ、連続してタスクを受信できるような仕組みを用意することによって、解決可能であると考えられる。また、応用アプリケーションでの実装になるが、コンテキストを明示し、どの処理系におけるタスクなのかをタスク実行者に示すといった手段によっても解決可能である。

5 関連研究

計算機では処理が難しいようなタスクを解決するために、人を計算資源として利用する手法はヒューマンコンピュテーション [8] と呼ばれ、様々な研究が行われている。インターネットを介して不特定多数の群衆にタスクを実行させるクラウドソーシングと組み合わせた研究事例も多く存在する。クラウドソーシングのプラットフォームとしては、Amazon Mechanical Turk[1] が存在する。 Barowy らは、CrowdProgramming という概念を提唱し、プログラミング言語内においてクラウドソーシングによる計算とコンピュータによる計算の統合を実現した [3]。Franklin らは、機械だけでは答えられないような DB へのクエリに対する応答を、クラウドソーシングを用いることで返答可能にする CrowdDB を提案している [6]。Morishima らは、人をデータソースとしてプログラムの中で利用する手法を提案している [7]。jabberwocky crowdforge これらの研究では、クラウドソーシングを利用した問題解決手法の提案をしている。本研究は、不特定多数の群衆をプログラムするためのものではなく、実世界の操作なども含むため、家族や職場の人のような特定可能な人物を対象としている。また、人を計算資源やデータソースとしてシステムに組み込むことを前提としているが、本研究は、計算資源やデータソースに限らず、実世界への干渉等も対象としている。本研究はより汎用的な枠組みであると言える。

ユビキタスコンピューティングの研究分野においては、Human as Sensor という概念も提唱されている。PRISM は、スマートフォンを利用したセンシングプラットフォームだ [5]。Liu らは、ソーシャルメディア上の人をセンサーとして扱った Q&A サービス MoboQ を提案し、その検証を行った。Human as Sensor に類する研究では、人をセンサーとして扱うことを対象としているが、本研究ではセンサーのみを対象としていない。

Cheng らは、モーションプラットフォームにおけるモーターやメカニカル機構の代替として人を利用した Haptic Turk を提案している [4]。Haptic Turk はゲームでの利用に特化したものだ。本研究は、用途を限らない汎用的な仕組みとなっている。

加藤らは、人とロボット間でのタスク共有システム Sharedo を提案した [9]。人とロボットのタスク実行における協調は、本研究の主眼である「何かの処理を実現するとき、人とコンピュータは相補的に動作するべき」という考えと大きく類似している。

6 おわりに

本論文では、人への命令構文をプログラムに付与可能なプログラミング環境 Babascript を提案した。

何かの処理を実行していく上で、コンピュータにできることは極力コンピュータに処理させ、人は人にしかできないことや、人がやるべきことだけを処理するというのが本来あるべき処理の切り分け方であるが、現状ではその切り分け方はあいまいである。

Babascript 環境においては、プログラム上において人は、コンピュータと同じ処理ノードとして存在し、関数実行によって処理内容を受け取り、実行して値を返す存在になれる。これによって、プログラ

マブルになりつつある世界において人自身もその一部になることが可能となる。

また、Babascript 環境によって実現する応用例を示すことによって、人をプログラマブルにした際のメリットを示した。

今後は、議論で述べた Babascript の問題点などを改善していく。

参考文献

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

[2] Linda implementation on Socket.IO. https://github.com/node-linda/linda.

[3] D. W. Barowy, C. Curtsinger, E. D. Berger, and A. McGregor. AutoMan: A Platform for Integrating Human-based and Digital Computation. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '12, pp. 639–654, New York, NY, USA, 2012. ACM.

[4] L.-P. Cheng, P. Lühne, P. Lopes, C. Sterz, and P. Baudisch. Haptic Turk: A Motion Platform Based on People. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pp. 3463–3472, New York, NY, USA, 2014. ACM.

[5] T. Das, P. Mohan, V. N. Padmanabhan, R. Ramjee, and A. Sharma. PRISM: Platform for Remote Sensing Using Smartphones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pp. 63–76, New York, NY, USA, 2010. ACM.

[6] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: Answering Queries with Crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pp. 61–72, New York, NY, USA, 2011. ACM.

[7] A. Morishima, N. Shinagawa, T. Mitsuishi, H. Aoki, and S. Fukusumi. CyLog/Crowd4U: A Declarative Platform for Complex Data-centric Crowdsourcing. *Proc. VLDB Endow.*, 5(12):1918–1921, Aug. 2012.

[8] L. Von Ahn. *Human Computation*. PhD thesis, Pittsburgh, PA, USA, 2005. AAI3205378.

[9] 淳 加藤, 大介 坂本, 健夫 五十嵐. Sharedo: To-do リストによる人-ロボット間のタスク共有. WISS '11: 第 19 回インタラクティブシステムとソフトウェアに関するワークショップ, pp. 102–107, 2011.