matereal: インタラクティブなロボットアプリケーションのプロトタイピング用ツールキット

matereal: A Toolkit for Prototyping Interactive Robot Applications

加藤 淳　　坂本 大介　　五十嵐 健夫*

Summary.　ユーザの指示に従って料理や服畳みといった実世界におけるタスクを実行するインタラクティブなロボットアプリケーションが実現しつつある．また，このような高度なアプリケーションの効果的な開発には，プロトタイピングが重要である．しかし，ロボット工学の前提知識を持たないソフトウェアプログラマにとって，そのようなアプリケーションのプロトタイピングは簡単ではない．その理由として，指定した場所への移動や物体の運搬などの高レベルな動作指示を低レベルなモータ動作の組み合わせに変換して実行する必要があること，複数のタスクを複数のロボットで並列実行するときにタスクを管理する必要があること，などが挙げられる．我々は，そのような問題を解決する手段として，平面上でのロボットの移動機能を 2 次元のベクトル場を用いてタスクの一種として抽象化した API と，実行時にシステムがソースコードからアクティビティ図を動的に構築してタスク管理を実現する API を提供するツールキット “matereal” を開発した．

1　はじめに

ユーザの指示に従って料理や服畳みといった実世界におけるタスクを実行するインタラクティブなロボットアプリケーションが実現しつつある [7][8]．一方で，既存のロボット研究の多くは，実用イメージのデモンストレーションにおいてロボットの動作シナリオがハードコーディングされている．エンドユーザがインタラクティブにロボットの動作シナリオを決められるシステムに関する研究は，未だ黎明期にあると言える．

センサやアクチュエータを用いたインタラクティブシステムに関する研究分野の先達である Physical Computing では，Phidgets[4] やプロトタイピング用のツールキット [5] が開発されたことによって，電子工作の知識を持たないソフトウェアプログラマも研究に参加できるようになり，研究が推進した．インタラクティブなロボットアプリケーションの研究開発においても，ロボット工学の前提知識を持たないソフトウェアプログラマがプロトタイピングできるような開発環境が整えば，分野の発展が見込めると考えられる．例えば，Human-Computer Interaction における肥沃な研究成果をロボット向けに援用することで，より多様な Human-Robot Interaction を実現するアプリケーションが生まれる可能性がある．

しかし，ロボット工学の前提知識を持たないソフトウェアプログラマにとって，そのようなアプリケーションのプロトタイピングは容易ではない．そこで我々は，その困難を解消するべくツールキット “matereal” を開発した．本稿では，ツールキットの開発に至った動機を説明して要求仕様を明確にし，matereal の概要を紹介する．また，matereal の利用例を通して実用性を評価し，考察を加えて現状の課題を述べた後，関連研究を紹介する．

2　matereal の要求仕様

エンドユーザの指示に応えるインタラクティブなアプリケーション例として，一台のロボットで指定された複数の場所を順々に巡ったり，一台のロボットがコーヒーカップを運搬して他の一台がタイミングを合わせてお湯を注いだり，複数台のロボットで分担して掃除を行うといった例が考えられる．このようなアプリケーションでは複数のタスクを複数のロボットで並列実行することになるため，相応の困難が生じる．まず，指定した場所への移動や物体の運搬などの高レベルなタスク指示を低レベルなモータ動作の組み合わせに変換し，タスクが完了するまでの間は実行し続けなくてはならない．また，どのロボットがどのタスクをどのタイミングで実行するかというワークフローを管理する必要がある．ワークフローの管理にはマルチスレッドプログラミングにまつわる問題がついてまわるため，ともすればプログラムがデッドロックを起こして固まったり，ロボットがどのコードに従って動作しているのか分からなくなってデバッグが難しくなりかねない．

我々はこれらの困難を解消し，素早く実働するロ

* Jun Kato and Takeo Igarashi, 東京大学 / JST ERATO 五十嵐デザインインタフェースプロジェクト, Daisuke Sakamoto, JST ERATO 五十嵐デザインインタフェースプロジェクト

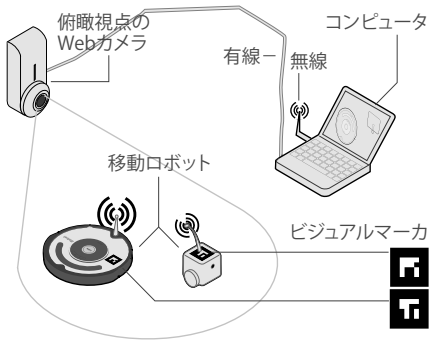

図 1. カメラと視覚マーカによる物体の 2 次元座標検出

ボットアプリケーションを開発できるプロトタイピングのための環境を構築したいと考えた．そのためには，高レベルなタスクが再利用可能なプログラム片として定義できるようになっている必要があるだろう．とくにロボットの基本的な機能であると考えられる環境内での移動に関する機能は，使いやすい組み込み API が提供されるべきである．また，マルチスレッドプログラミングを意識せず使えるワークフロー管理のための API が必要だろう．以降，これらの要求仕様に応える “matereal” の API の概要と実装を説明していく．

3 matereal

“matereal” は Java 用のライブラリとして実装されており，インタフェースや抽象クラスを用いて容易に拡張可能な構成となっている．本節ではその概要と実装について説明する．なお，仕様の詳細についてはツールキットの公式サイト[1]を参照されたい．

3.1 概要

本稿で提案するツールキット “matereal” は，図 1 のように一般的な Web カメラと視覚マーカを用いて物体の床面上の二次元絶対座標値を取得する機能が組み込まれている．この面上での移動に関しては，ツールキットが提供する高レベル API によって，ロボットのアクチュエータに継続的に低レベルコマンドを送出する処理が隠蔽される．また，どのロボットがどのタスクをどのタイミングで実行するかというワークフローを適切に管理する機構により，プログラマがマルチスレッドプログラミングを意識する必要がなくなり，容易にロボットアプリケーションをプロトタイピングできるようになる．

3.2 ロボットとタスク

本ツールキットでは，ロボットはリソースと呼ばれる機能単位の集合として扱われ，種類ごとにクラスとして定義されている．タスクも種類ごとにクラスとして定義されており，そのインスタンスはロボットのリソースを利用して実行される．タスクは，場合によってはリソースを排他利用する．例えば，アクチュエータの動作状況を読み取るだけなら排他制御は不要だが，アクチュエータへの出力をコントロールするタスクは一度に一つでなければ誤動作を招く．このような場合，読み取りに用いられるインタフェースと制御に用いられるインタフェースが，車輪の場合は Wheels と WheelsController のように別々に定義されている．排他制御を要するインタフェースは ExclusiveResource インタフェースを継承するように定められており，ツールキットが適切に排他制御の状態を管理する．これらの仕組みにより，一台のロボットに複数の競合しないタスクを安全に割り当てることができる．例えば，ソースコード 1 のように移動しながら周囲の映像を撮影するプログラムを書くことができる．

ソースコード 1. 前進しながらカメラ映像を撮る

```
1  Robot robot = new NetTansor();
2  GoForward gf = new GoForward();
3  Capture cap = new Capture();
4  if (gf.assign(robot)) gf.start();
5  if (cap.assign(robot)) {
6    cap.addImageListener(new ImageListener() {
7      // (略)
8    });
9    cap.start();
10 }
11 // 以降ロボットは gf.stop() を呼ぶまで前進,cap.
       stop() を呼ぶまでカメラ映像を撮り続ける.
```

プログラマが自作したロボットを本ツールキットで使いたい場合，まず自分のロボットを表すクラスを，ロボット全般を表す抽象実装クラスを継承して定義する．その際，ロボットが持つリソース一覧を返すメソッドを実装する必要がある．リソースは，多くの場合ロボットの内部クラスとして記述するのが適切である．ここで，リソースを予め定義されたインタフェース（例えば移動機能なら WheelsController）に実装させることによって，自作ロボットが既存のプログラムからでも利用可能になる．独自の機能を提供するリソースの場合はとくに既存のインタフェースを実装する必要はないが，一方でそれを利用するタスクのクラスも自分で定義する必要がある．タスクを表すクラスは自分の必要とするリソースの型一覧を返すメソッドを実装することが求められる．例えば，ロボットの車輪による移動機能を用いるタスクは WheelsController を含む一覧を返す．

3.3 2 次元ベクトル場を用いた移動・運搬タスク

本ツールキットは，天面に視覚マーカを貼り付けたロボットや物体を俯瞰視点で撮影することによっ

[1] http://mr.digitalmuseum.jp/

て，ロボットや物体の床面上での位置と方向を取得できる．この面上での移動に関するタスクは，ロボットが指定した位置へ移動するタスク Move と，指定した物体を指定した位置へ押し運ぶタスク Push が組み込みで用意されている．前者は指定位置に向かって落ち込むアリ地獄のような単純なベクトル場，また，後者はダイポール場を表すベクトル場を用いて実装されている [6].

プログラマは，独自のベクトル場を表すクラスを定義して移動アルゴリズムを新しく実装することもできる．ベクトル場を表すクラスは，cm 単位の実世界座標系において，任意の位置に対してロボットが進むべき方向を示す 2 次元ベクトルを返すメソッドを実装することが求められる．

3.4 アクティビティ図

一つのタスクを一台のロボットに実行させるだけであれば，これまでに述べた機能で十分に記述できる．しかし，実際に役立つロボットアプリケーションを開発する際には，しばしば複数のタスクを順番に実行したり，複数台のロボットを同時に扱う必要が生じる．本ツールキットでは，このようなシナリオを API を通じてアクティビティ図として組み上げ実行できるようにすることで，マルチスレッドプログラミングなどタスク管理のために生じる煩わしさからプログラマを解放することを目指した．

アクティビティ図は，UML（Unified Modeling Language）1.0 で定義されたワークフローを表現するための様式である．表 1 に示した現在のツールキットの実装の範囲では，アクティビティ図は概ねフローを複数並列実行できるようにしたフローチャートの拡張とみなすことができる．なお，実世界におけるアプリケーションでは，3 分以内に移動が終わらなかったら諦めるなど，経過時間に応じた処理が必要になることが多い．そこで，タスクの終了を待って状態遷移するアクティビティエッジに加え，タスクの実行開始から一定時間が経過すると状態遷移する例外処理用の有向エッジを新しくタイムアウトエッジと名付け，実装した．

なお，本ツールキットにおいてアクティビティ図はあくまでソースコードから生成するものであり，Visual Programming Language のように視覚的なプログラミング用インタフェースとして使われることはないが，現在アプリケーション内で走っているアクティビティ図を可視化するビューワが組み込みで用意されている．これによりロボットの現在の状態を把握でき，エラー発生時にソースコードのどの部分を見ればよいのか見当がつきやすくなっている．

4 サンプル・アプリケーション

本章では，これまでに説明した API を利用したアプリケーションの例をソースコードと共に紹介し，ツールキットの有用性と実用性を確認する．

4.1 Bring it here!

本例は運搬タスクを利用する．画面に机上の俯瞰映像が表示されており，視覚マーカを貼り付けた物をクリックするとロボットがその物を押して人の目の前に運んできてくれる．以下は俯瞰映像を表示するパネルに追加するマウスリスナのソースコードである．スクリーン座標 (x,y) にある物をマーカ検出結果から探し出して返すメソッド getClickedEntity(x,y) が与えられているものとする．

ソースコード 2. Bring it here!

```
1  panel.addMouseListener(new MouseAdapter() {
2    ScreenPosition goal = null;
3    public void mouseReleased(MouseEvent e) {
4      int x = e.getX(), y = e.getY();
5      if (goal == null) {
6        goal = new ScreenPosition(x, y);
7        return;
8      }
9      Entity entity = getClickedEntity(x, y);
10     if (entity != null && entity != robot) {
11       Push push = new Push(entity, goal);
12       if (push.assign(robot)) push.start();
13     }
14   }
15 });
```

表 1. アクティビティ図の構成要素一覧

構成要素	対応するクラス	用途
アクティビティ図	ActivityDiagram	
ノード	Node	全ノードの基底クラス
動作ノード	Action	ロボットによるタスクの実行を表す
分岐ノード	Decision	複数の遷移先候補から一つを選択する
フォークノード	Fork	複数の遷移先ノードを開始して自身は終了する
ジョインノード	Join	複数の遷移元ノードの終了を待って自身も終了する
アクティビティエッジ	Transition	ノード間の遷移を表す
タイムアウトエッジ	TimeoutTransition	ノード間の時間切れによる遷移を表す

まずエンドユーザが自分の目の前に相当する位置をクリックすると，変数 goal に物を運ぶ先が代入される．その後ユーザが物をクリックすると，運搬タスク Push がインスタンス化されてロボットがクリックされた物を goal まで押し運ぶ．

4.2 ロボ書道

本例はアクティビティ図を利用する．ユーザがマウスカーソルで文字列を描くと，ペンの上げ下ろしができる移動ロボットが床に敷いた模造紙の上でそれを真似て描く．複数台の同じ機能を持ったロボットがいて，書く文字を分担する．ユーザインタフェース部分のソースコードを省略するため，ユーザが描いた文字は一文字ずつ連続したパスの集合 List<Path> として保持されており，その全体のセットが Set<List<Path>> string として取得できているものとする．また，連続したパスを描くタスクを表す DrawPath クラス，ロボットの配列 robots が用意されているものとする．ペンの上げ下ろしはツールキットの組み込み機能ではないため，ペンを表すインタフェースとリソースの実装クラスおよびこれらに対応するタスクも別途用意する必要がある．

ソースコード 3. ロボ書道

```
1  ActivityDiagram ad = new ActivityDiagram();
2  Action[] inits = new Action[robots.length];
3  Action[] as = new Action[robots.length];
4  int i = 0;
5  for (List<Path> c : string) {
6    for (Path p : c) {
7      Action a = new Action(robots[i], new
           DrawPath(p));
8      if (as[i] == null) {
9        inits[i] = a;
10     } else {
11       ad.addTransition(new Transition(as[i], a));
12     }
13     as[i] = a;
14   }
15   i = (i + 1) % robots.length;
16 }
17 Fork fork = new Fork(inits);
18 ad.setInitialNode(fork);
19 ad.start();
```

文字 c を得るループのなかでロボットを選択するインデックス i がインクリメントされ，文字ごとに違うロボットが描画を行うようにアクティビティ図が構築されていく．パスを描く動作ノード a を得るループのなかでは，まず各ロボット robots[i] が最初にパスを描くタスクを inits[i] に格納し，以降は一つ前のパスを描くタスクが as[i] に格納されているので，それと a を Transition で結んでいる．全ての文字を描くタスクを繋ぎ終わったら，フォークノードによって各ロボットが最初にパスを描くタスクが並列実行されるようにしている．これにより，各ロボットが同時並行で各々の担当している文字を描くようになる．最後にフォークノードを初期ノードに指定し，アクティビティ図を実行すれば，ロボット総出で文字が描かれる．

4.3 日本の食卓

本例では，システムがユーザの指示に従って味噌汁を好みの手順で調理するほか，ご飯を炊飯器からよそい，日本茶を淹れてくれる．下表 2 に挙げたロボットと，ご飯をよそうお椀と，味噌汁を調理する IH 対応の小さなお椀兼用鍋と，日本茶パックの元々入っている湯呑みを用意する．

これらのロボットを操作して食卓を準備するソースコードは長大になるため省略するが，これまでに紹介したサンプルアプリケーション例で利用した機能を用いれば実装可能である．例えば IH 調理器や湯沸かし器，炊飯器は，マイコンと Bluetooth チップを用いて遠隔操作できるようにハックしたものを用意することにより，ソフトウェアにおいてはロボ書道で通常のロボットにペンの上げ下ろしのタスクを実装したようにタスクを介して操作できるようになる．他の独自ロボットも同様である．そして，エンドユーザが味噌汁の好みの調理方法をユーザインタフェース [8] で指示すると，順次対応するワークフローが構築され，他のロボットのタスクも同じアクティビティ図に組み込んで実行されることで目的が果たされるだろう．最終的には，例えば図 2 に示したようなアクティビティ図が構築される．

表 2. ロボット一覧

ロボット名	役割
IH 調理器	鍋の温度の管理
おたまロボット	鍋の攪拌
具材投入ロボット	具材の鍋への投入
炊飯器	よそうための蓋の開閉
ご飯よそいロボット	ご飯をよそう
湯沸かし器	湯沸かしと給湯
給仕ロボット	お椀や湯呑みの運搬

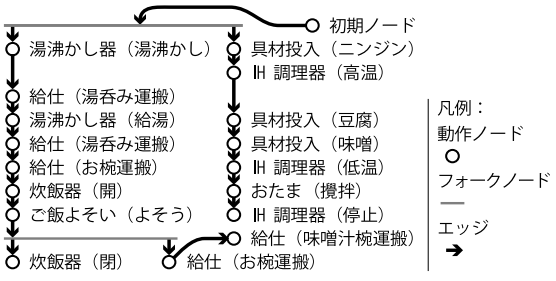

図 2. 日本の食卓を準備するアクティビティ図

5 議論

5.1 タスク管理の仕組みに関する議論

我々は，本ツールキットでタスクを管理する仕組みとしてソースコードからアクティビティ図を生成する API を提供したが，GUI でアクティビティ図を生成，編集して同等のワークフローを設計・実行できる Visual Programming Language（VPL）を開発することもできるだろう．しかし，その場合のアクティビティ図はエンドユーザの入力に応じて組み替えたりできない静的なものである．VPL はエンドユーザ向けのワークフロー設計用ユーザインタフェースとして利用でき，“matereal” を用いて開発できるアプリケーションの一つだが，本ツールキットの代替にはなり得ない．

本ツールキットが主なユーザ層として期待しているソフトウェアプログラマが慣れ親しんだプログラミング手法として，GUI で事実上標準となっているイベント駆動型モデルが挙げられる．本ツールキットは，タスクにイベントリスナを追加してタスクの完了通知を受けるなど，イベント駆動型モデルによるプログラミングをサポートしているが，それだけではツールキットの要求仕様は満たせないと考えている．なぜなら，複数のタスクを順次実行する場合などに，リスナの中でタスクのインスタンスを生成し，それに対してリスナを追加するといったリスナの入れ子構造が生じるからである．リスナの入れ子はソースコードの可読性を損ねるだけでなく，アクティビティ図のように処理の流れが自明ではないため，ロボットが今プログラムのどの部分にしたがって動作しているのか簡単には分からずデバッグが困難になるという問題点を抱えている．

5.2 将来的なデバッグ支援機能の拡充

3.4 節で紹介したように，本ツールキットにはアクティビティ図を用いたことで可能になったデバッグ支援機能が実装されている．我々は，今後この機能を拡充する必要があると考えている．まず，プロトタイピングにおいてはエンドユーザがどのようにシステムとインタラクションしているか観察することが大切とされており，ツールキットがカメラの映像とアクティビティ図上の状態遷移を同期して記録しておけば，そのようなことが容易になる．また，記録したデータをあとでシミュレータ上あるいは実機上で再現できるようにすれば，ロボットの振る舞いを細かく見直すことができるだろう．

6 関連研究

ロボットの開発を支援する環境はこれまでにも多数提案されてきた．RT-Middleware[2] などのミドルウェアは，分散ネットワーク環境において，ロボットやセンサ，それらを用いるプログラムなどのコンポーネントとコンポーネント間の通信様式を標準化することを目指している．プログラマがミドルウェア上でアプリケーションを開発するためには，利用するコンポーネントを選び，コンポーネント間通信について学ぶ必要がある．一方，本ツールキットは必要な機能をワンストップで提供するライブラリとして開発されたため，一台の PC 上でのソフトウェアプログラミングと同様にロボットアプリケーションを開発できる．

大規模なミドルウェアを除けば，Urbi[3] などほとんどのツールキットは一台のロボットの動作計画をプログラムするための機能しか用意していない．ただし，Urbi は最新版ではロボットの部品やタスクを表すクラスを提供しており，ロボットのハードウェア構成を body.arm.grip のようにツリー構造で表現できる．これは，ロボットが単にリソースの集合として表される本ツールキットより複雑だが高度である．本ツールキットも今後このような仕組みを取り入れる必要があるだろう．なお，一台のロボットに備わったセンサのみで床面上の絶対座標を取ることは難しい．したがって，Urbi を含むほとんどのツールキットは距離センサの値を取得できる機能のみ提供し，ロボットの測位に関する他の処理はプログラマに任せている．移動指示に関しても，今いる位置からの相対速度と角度を指定できるだけのことが多い．移動した距離を自分でモニタリングできるロボットの場合は相対座標への移動指示を出せることもある．例えば Tekkotsu[9] は目標地点のリストを与えて移動ロボットを導けるが，精確な絶対座標を得る機能がないためエラーが蓄積し，目標が徐々にずれていく．静的な環境であれば事前に環境の地図を与え，地図上の絶対座標系を用いることができるが，物体の運搬などで地図が変化すると問題が生じる．一方，本ツールキットでは天井カメラさえ固定しておけば環境の変化に影響を受けず，地図を事前に用意する必要もない．本ツールキットは，床面上でのロボットと物体の測位を行い，さらにベクトル場を用いて移動指示を抽象化した点に新規性があると言える．

時間軸に沿ってタスクを実行していく研究は，もともと並列コンピューティング向けになされており，初期にデータフロー言語 [1] というものが提案された．類似したものとして，システムのワークフローをモデル化するワークフロー言語がある [10]．これらの概念は，研究と並行して，オブジェクト指向の考え方と共にビジネス向けに UML の一部，すなわちアクティビティ図として再構成され，標準化された．我々の研究は，UML におけるアクティビティ図に実時間の概念を加えて実世界でのタスクのワークフローを直接表現できるように設計し，実際にロボットを用いてタスクを実行させられるように実装した初めての試みである．

7 まとめ

本稿では，ロボット工学の専門家でないソフトウェアプログラマでもインタラクティブなロボットアプリケーションをプロトタイピングできるツールキット “matereal” を提案した．これにより，既存の Physical Computing の範疇に留まらない，実世界を動き回れるロボットの特徴を生かした実世界指向アプリケーションをプロトタイピングするための環境を整えることができたと考えている．

参考文献

[1] W. Ackerman. Data flow languages. *Computer*, 15(2):15–25, 1982.

[2] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and W.-K. Yoon. RT-Middleware: Distributed Component Middleware for RT (Robot Technology). In *Proc. of IROS '05*, pp. 3555–3560. IEEE, 2005.

[3] J. Baillie. Urbi: Towards a universal robotic low-level programming language. In *Proc. of IROS '05*, pp. 820–825. IEEE, 2005.

[4] S. Greenberg and C. Fitchett. Phidgets: easy development of physical interfaces through physical widgets. In *Proc. of UIST '01*, pp. 209–218, New York, NY, USA, 2001. ACM.

[5] B. Hartmann, S. R. Klemmer, M. Bernstein, L. Abdulla, B. Burr, A. Robinson-Mosher, and J. Gee. Reflective physical prototyping through integrated design, test, and analysis. In *Proc. of UIST '06*, pp. 299–308, New York, NY, USA, 2006. ACM.

[6] T. Igarashi, Y. Kamiyama, and M. Inami. A Dipole Field for Object Delivery by Pushing on a Flat Surface. In *Proc. of ICRA '10*, pp. 5114–5119. IEEE, 2010.

[7] Y. Sugiura, T. Igarashi, H. Takahashi, T. A. Gowon, C. L. Fernando, M. Sugimoto, and M. Inami. Graphical instruction for a garment folding robot. In *ACM SIGGRAPH 2009 Emerging Technologies*, pp. 1–1. ACM, 2009.

[8] Y. Sugiura, D. Sakamoto, A. Withana, M. Inami, and T. Igarashi. Cooking with robots: designing a household system working in open environments. In *Proc. of CHI '10*, pp. 2427–2430. ACM, 2010.

[9] D. Touretzky and E. Tira-Thompson. Tekkotsu: A framework for AIBO cognitive robotics. In *Proc. of AAAI '05*, pp. 1741–1742, 2005.

[10] W. Van Der Aalst and A. Ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

未来ビジョン

我々はロボットが何であるかを知らない．自分たちで作った言葉であるにも拘らず，言葉が独り歩きし，各々が勝手なイメージを持っている．日本においては，とくに大型で人型の印象が強いようである．しかしながら，既存のロボット研究において一般的に用いられてきた大型のロボットは，ロバストに動作させるためにハードウェアおよび制御用ソフトウェアに高度な作りこみが求められ，かつ高価なため，そもそも多くのソフトウェアプログラマが満足にアプリケーション開発できるだけの環境を用意するのが困難である．また，空間知能化などのキーワードを考えたとき，大型ロボットは動作空間を用意すること自体に手間がかかる．

一方，小型のロボットであれば Arduino 等の Physical Computing 用ツールキットを利用して比較的安価に自作することが容易であり，さまざまな種類のロボットを試作できることが見込まれる．小型であれば机上など多くの場所で動作させることができるだろう．アプリケーションが確立すれば，小型ロボットなら大量生産もしやすく一層安価になることが期待できる．そもそも人間の生活環境における小型ロボットの存在は既存の家電製品との境界が曖昧であり，とくに家電製品をネットワーク越しに操作できるようになったとき境界はほぼ消失すると考えている．このように我々は，ネットワーク家電を含む小型ロボットが複数台・複数種人間の生活環境に溶け込んでいる未来ビジョンを念頭においてツールキットを開発してきたため，本文中のサンプルアプリケーションでも多くの小型ロボットや遠隔操作できるようハックした家電製品を登場させた．

我々が描く未来は，多くの人々，最初は研究者や科学・技術に強い興味のあるプログラマなどに，まずはロボットに触れてもらうところから始まると考えている．触れるということは，そのプログラムやアプリケーションを開発することであり，それによってロボットで何が出来るのか，どうしたら良くなるのかを，広く議論して行きたいということである．そこで我々は，手軽にプロトタイピングを行える共通のスタートラインとしてのツールキットが必要であると考え，“matereal” の研究開発を行ってきた．現時点では Java 用ライブラリとして提供しているが，Processing や他の言語の開発環境でも利用できるようにして，さらに幅広い層のプログラマにとって魅力的な環境構築を目指していきたい．