対話型視覚的アプリケーションのための制約命令型プログラミング言語

細部 博史

グラフィカルユーザインタフェースやウェブなどの対話型視覚的アプリケーションの構築は，プログラミングにおける古くからの重要な課題である．この目的では命令型プログラミングが用いられることが多い．一方で，制約プログラミングをこの目的に適用する試みも行われており，制約命令型プログラミングの枠組みも提案されているが，普及には至っていない．本研究では，対話型視覚的アプリケーションのための新しい制約命令型プログラミング言語 P5CP を提案する．本言語は，命令型プログラミングで使われるイベントの概念と，制約プログラミングで使われるガードの概念を利用することで，これら 2 つのプログラミング方式を統合する．さらに本研究では本言語の計算モデルと実装を示す．

1 はじめに

グラフィカルユーザインタフェースやウェブなどの対話型視覚的アプリケーションの構築は，プログラミングにおける古くからの重要な課題である [10] [11] [25]．この目的では命令型プログラミングが用いられることが多く，C++や Java，JavaScript などの言語がよく利用される．

一方で，制約プログラミングをこの目的に適用する試みも多く行われている [1] [3] [4] [9] [23]．制約は，内部データとその視覚的表現の間の一貫性維持や，画面上での視覚的配置の指定，アニメーションのためのイベントの指定などを容易にするため，対話型視覚的アプリケーションの構築に有効である．

さらに，命令型プログラミングと制約プログラミングを統合する制約命令型プログラミングの枠組みも提案されている [6] [7] [8] [15] [16] [19] [20]．しかし，現状で普及には至っていない．

本研究では，対話型視覚的アプリケーションのための新しい制約命令型プログラミング言語 P5CP を提案する．本言語は，命令型プログラミングで使われるイベント [11] [24] の概念と，制約プログラミングで使われるガード [26] の概念を利用することで，これら 2 つのプログラミング方式を統合する．さらに本研究では本言語の計算モデルと実装を示す．

2 関連研究

本研究に関連する先行研究について述べる．

2.1 リアクティブシステム

本研究が対象とする対話型視覚的アプリケーションはリアクティブシステムの 1 種である．リアクティブシステムは変換型システムと対になる概念である．変換型システムが入力を受け取って，それを関数的に変換し，出力を生成して終了するのに対して，リアクティブシステムは外界との相互作用を繰り返し，外界からの入力に継続的に反応することをその役割とする [14]．リアクティブシステムの設計に関する初期の研究には，時相論理 [21] やステートチャート [13] を用いるものがある．

A Constraint Imperative Programming Language for Interactive Graphical Applications. (This is an unrefereed paper. The author holds the copyright.)

Hiroshi Hosobe, 法政大学情報科学部, Faculty of Computer and Information Sciences, Hosei University.

2.2 イベント駆動プログラミング

対話型視覚的アプリケーションの開発にはイベント駆動プログラミング (event-driven programming; EDP) が用いられることが多い．EDP では，ユーザ入力やタイマ起動などに呼応してイベントが発生すると，その種類に応じたイベントハンドラが呼び出され，処理が行われる．EDP を明確に導入した初期のシステムに，アルバータ大学ユーザインタフェース管理システム [11] がある．

EDP の実装方式の 1 つとして Java の委譲イベントモデル [24] がある．この方式では，EventListener インタフェースのメソッドを実装し必要な設定を行うことで，イベントハンドラを簡明に実現できる．

Processing [22] は視覚的アプリケーションのプログラミング学習を目的として開発されたプログラミング言語・環境である．Processing は Java を基礎とするが，イベントハンドラの実装は規定の関数に関して必要な範囲で行うだけでよく，Java の委譲イベントモデルよりも単純である．また，Processing の主要なプログラミング方式であるアクティブモードでは，プログラムの実行開始直後に `setup` 関数が呼ばれ，その後，一定時間 (デフォルトで 1/60 秒) ごとに `draw` 関数が呼ばれるようになっており，これらもイベントハンドラと見なすことができるため，Processing の計算モデルは EDP を基本としているとも言える．

p5.js [18] は Processing のプログラミング方式を JavaScript 上に実現したライブラリである．p5.js を用いることで，ウェブブラウザ上で実行できる視覚的アプリケーションの作成が容易になる．

2.3 制約プログラミング

ユーザインタフェースなどの対話型視覚的アプリケーションの開発には制約が有用であることが古くから知られている [25]．具体的には，内部データとその視覚的表現の間の一貫性維持，複数の視覚的表現の間の一貫性維持，画面上での視覚的配置の指定，アニメーションのためのイベントの指定，アニメーションに関するオブジェクトの属性の指定などに制約は利用可能である [3]．

初期のシステムでは，制約の変数の間の入出力関係が事前に定められている単方向制約 [19] や，制約の接続関係に基づいて入出力関係が自動選択される多方向制約 [3] が用いられることが多かった．その後，事前に指定された制約間の優先度に基づき，制約過多な場合も扱うことのできる制約階層 [4] が提案され，DeltaBlue [9] などの制約解消系に関する研究が多数行われた．特に線形等式・不等式からなる制約階層を扱うことのできる Cassowary [1] 制約解消系は，Apple の Cocoa フレームワークの Auto Layout システムで採用されている [23]．

リアクティブプログラミング [2] は，近年普及しつつある対話型視覚的アプリケーションの開発手法である．これは，時間の進行に伴い変化する値の依存関係を自動的に維持する機構を用いるものであり，単方向制約や多方向制約に基づく制約プログラミングに類似している．

2.4 制約命令型プログラミング

対話型視覚的アプリケーションのための制約プログラミング言語の開発も行われており，その多くは命令型プログラミング言語に対して制約の処理機能を追加したものである．Garnet [19] は，Common Lisp 上にプロトタイプベースのオブジェクトシステムと，単方向制約の処理機能を実現した言語である．また，Garnet の後継として，C++上に単方向制約の処理機能を実現した言語 Amulet [20] も開発されている．他にも C++上に単方向制約の処理機能を実現した言語として Eval/vite [15] がある．これらの言語では全ての変数に対して制約を付加できるようにはなっておらず，その意味では制約プログラミングと命令型プログラミングの統合は部分的であると言える．

Kaleidoscope'90 [8] は，制約プログラミングと命令型プログラミングを密接に統合した言語であり，全ての変数が制約可能である．命令型プログラミングにおける破壊的代入を可能にするために，変数は時刻に応じた値を持ち，代入の際に時間が進むという意味論を導入している．Kaleidoscope'90 の後継として，制約処理の基本モデルを変更して効率化し，専用の仮想機械を提供した Kaleidoscope'93 [16] がある．さらにその発展として，制約生成の機構を単純化して効率化

した Babelsberg [6] がある．また，JavaScript を拡張して Babelsberg の制約プログラミングの機能を実現した言語として，Babelsberg/JS [7] がある．

2.5 ハイブリッド並行制約プログラミング

命令型でない制約プログラミング言語の枠組みの 1 つとして，ハイブリッド並行制約プログラミング (hybrid concurrent constraint programming; HCCP) [12] [26] がある．その主な特徴は，常微分方程式を制約としてプログラム中に直接記述できる点である．並行制約プログラミングでは，制約を採用すべき条件をガード [5] として記述することができる．HCCP では，ガードを常微分方程式と組み合わせることで，現在の状態に応じて常微分方程式を切り替えられるハイブリッドシステムを記述できる．Hybrid cc [12] は最初の HCCP 言語である．

HydLa [26] は，HCCP に制約階層 [4] を新たに導入しながら，簡明な構文と計算モデルを実現し，ハイブリッドシステムの簡潔な記述を可能にした言語である．HydLa には，実行結果の可視化機能を備えたプログラミング環境 HIDE [17] も開発されている．

3 提案言語の概要

本論文で提案するプログラミング言語 P5CP は，JavaScript 言語と p5.js ライブラリの組合せに対して，制約プログラミングの機能を加えたものである．JavaScript 自体は命令型プログラミング言語であるため，制約プログラミングの機能をさらに備えた本言語は，制約命令型プログラミング言語と見なすことができる．

本言語は，Kaleidoscope'90 などの制約命令型プログラミング言語とは異なり，制約プログラミングと命令型プログラミングの密接な統合を行わない．本言語では 1 つのプログラムの中に制約プログラム (constraint program; CP) と命令型プログラム (imperative program; IP) が混在するが，それぞれが明確に区別できる形で存在する．プログラムの実行も，CP を処理する CP フェーズと，IP を処理する IP フェーズが交互に動作することで進められる．フェーズ内で計算を進める基本機構が CP フェーズではガードであり，IP フェーズではイベントである (5 節で詳細を述べる)．

プログラム実行中の CP，IP の状態はそれぞれ CP 変数，IP 変数によって保持される．CP，IP のいずれも CP 変数，IP 変数の両方の値を読み取ることができるが，基本的に CP フェーズ，IP フェーズの実行で変更されるのはそれぞれ CP 変数，IP 変数の値である．CP 変数，IP 変数の区別は変数名によって行われ，`$x` や `$foo` のようにドル記号で始まるものを CP 変数とし，それ以外の変数を IP 変数とする．

JavaScript では本来，`var` 文で宣言されたもののみを変数と呼ぶべきであるが，本論文では JavaScript における変数，オブジェクトのプロパティ，関数の仮パラメータを区別せず，単に変数と呼ぶ．ただし，本言語で CP 変数にできるのは，JavaScript におけるオブジェクト (グローバルオブジェクトを含む) のプロパティに相当するもののみである．例えば単に `$x` のように書かれた場合，これはグローバルオブジェクトのプロパティを指し，`bar.$foo` のように書かれた場合，これはオブジェクト `bar` のプロパティ `$foo` を指す．

本言語のプログラムでは IP が主体であり，その中に CP の断片が埋め込まれる．IP フェーズの実行中に CP の断片が現れると，CP 変数の生成・初期化，または (ガード付き) 制約の生成が行われる．

CP は CP 変数宣言文と `always` 文からなる．CP 変数宣言文は CP 変数を生成して初期化する．構文的には，変数名がドル記号で始まることを除き，通常の JavaScript における等号記号を用いたプロパティの宣言文と同様である．具体的には，グローバル環境で，新しい変数名 `$x` に対して "`$x = 1;`" のように記述するか，オブジェクトのコンストラクタで，新しい変数名 `$foo` に対して "`this.$foo = 100;`" のように記述することで，CP 変数を宣言できる．

`always` 文は CP 変数の初期化後に成立すべき制約を宣言する．`always` 文にはガード付きの制約を記述でき，この場合，ガードが成立するときにのみ制約が採用される．ガード付きの制約は，JavaScript における `if` 文と同じ構文を用い，条件部にガードを，then 節に制約 (ガード付きでもよい) を記述する．さらに

if 文に else 節を付けることもできる．ガードとして記述できるのは，JavaScript における条件式のうち，副作用のないものである．一方，制約として記述できるのは，単方向制約と常微分方程式制約である．

IP の記述は JavaScript とほぼ同様であるが，IP の中で CP 変数を記述することでその値を読み取ることができる．一方，IP の中で CP 変数に代入することはできない．

4 プログラム例

前節で述べた提案言語 P5CP は，Hybrid cc や HydLa と同様，常微分方程式をサポートするため，物理シミュレーションを有望な応用領域とする．特に本言語の場合，p5.js の機能を用いて，物理シミュレーションの実行状況をアニメーション表示することができるため，シミュレーション結果をより直観的に理解できる利点がある．

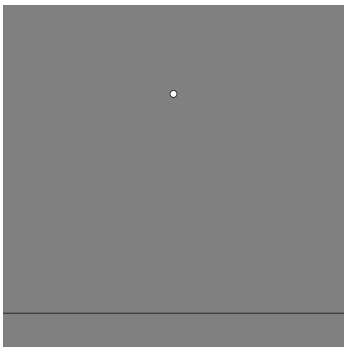
本言語のプログラム例と実行画面を図 1 に示す．これは，HydLa の例題[26] としても用いられている，床で弾むボールのシミュレーションのプログラムを本言語向けに書き直し，さらに画面描画処理を加えたものである †1．

図 1(a) のプログラムの中で，1〜2 行および 12〜19 行が IP，3〜11 行が CP である．CP 変数はボールの高さを表す $y の 1 個のみであり，これは 3 行の CP 変数宣言文によって生成され，値 10 に初期化される．一方，4〜11 行は always 文である．これは 1 個の if 文 (5〜10 行) からなり，$y >= 0 (5 行) をガードとする常微分方程式制約 $y'' == -G (6 行) と，$y >= 0 の否定をガードとする 2 個の単方向制約 $y = 0 (8 行) と $y' = -E * pre $y' (9 行) を生成する．直観的には，$y >= 0 が成立するとき，$y'' == -G が採用され，ボールは重力によって加速される．一方，$y >= 0 が成立しないとき，$y = 0 と $y' = -E * pre $y' が採用され，ボールの高さが 0 に設定されて，速度が下向きから上向きに変更される．

12〜14 行で setup 関数を，15〜19 行で draw 関数

```
 1  const G = 9.8;  // 重力加速度 (定数)
 2  const E = 0.5;  // 反発係数 (定数)
 3  $y = 10;        // ボールの高さ (CP 変数)
 4  always {
 5    if ($y >= 0)  // ガード
 6      $y'' == -G; // 常微分方程式制約
 7    else {
 8      $y = 0;             // 単方向制約
 9      $y' = -E * pre $y'; // 単方向制約
10    }
11  }
12  function setup() {
13    createCanvas(500, 500);
14  }
15  function draw() {
16    background(128);            // 背景
17    line(0, 450, 499, 450);     // 床
18    ellipse(250, 450 - 40 * $y,
              10, 10);            // ボール
19  }
```

(a)



(b)

図 1　床で弾むボール：(a) プログラム；(b) 実行画面

を定義している．これらは p5.js における基本的な関数であり，setup は初期設定のためにプログラム起動直後に 1 回だけ呼び出され，draw は画面描画のために一定時間 (デフォルトで 1/60 秒) ごとに繰り返し呼び出される．本プログラムでは setup で描画領域の大きさを定め (13 行)，draw で背景色による塗りつぶし (16 行)，床の描画 (17 行)，ボールの描画 (18 行) を行っている．ボールの描画の際には CP 変数 $y の値を読み取り，ボールの高さに応じた位置に円を描いている．

†1　6 節にも述べる通り，本論文執筆時点で本言語は実装中であり，このプログラムをそのまま実行できる完成度には至っていない．

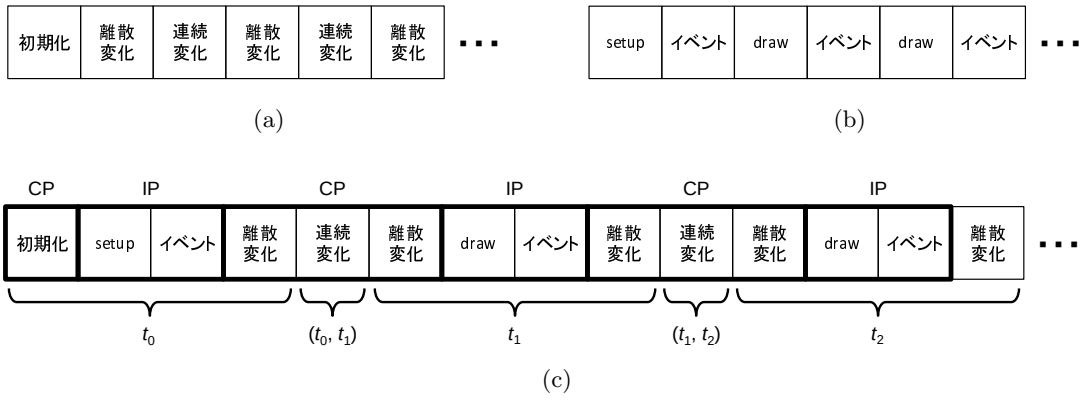


図 2　(a) 通常のハイブリッド並行制約プログラミング；(b) p5.js におけるイベント駆動プログラミング；(c) P5CP の計算モデル

5　計算モデル

3 節で述べた通り，提案言語 P5CP では，CP を処理する CP フェーズと，IP を処理する IP フェーズが交互に動作する．基本的に CP フェーズは従来のハイブリッド並行制約プログラミング (HCCP) に相当し，IP フェーズは従来のイベント駆動プログラミング (EDP) に相当しており，P5CP の計算モデルはこれらの処理を分割した上で，交互に動作するようにしたものである．図 2 に，通常の HCCP，p5.js における EDP，P5CP の計算モデルを模式的に示す．

通常の HCCP (図 2(a)) では，初期化の後，時間が進行しない状態変化である離散変化 (複数の離散変化が続けて生じる場合も含む) と，時間が進行する状態変化である連続変化が交互に繰り返し生じる．連続変化は特に常微分方程式を解く (例えばボールが落下する) ことで生じ，離散変化は連続変化の終了直後 (例えばボールが床で弾む) に生じる．このように離散変化と連続変化の切り替えが生じるのは，状態の変化に応じてガードの真偽が変わり，そのために採用される制約も変わるためである．

p5.js における EDP (図 2(b)) では，最初に `setup` 関数が呼び出され，その後，一定時間ごとに `draw` 関数が呼び出されることで処理が進行する．さらに，これらの関数呼び出しの合間にも，他のイベント (例えばキーの入力やマウスの移動) の発生に応じて，対応する関数が呼び出される．ここで `setup`，`draw` の呼び出しも特殊なイベントによるものであると見なせば，全ての処理はイベントに応じた関数呼び出しによって行われると言える．

P5CP の計算モデル (図 2(c)) では，EDP における `draw` の各呼び出しの直前に区切りを入れ，それぞれのまとまりを HCCP における離散変化の途中に差し込むことで，CP フェーズと IP フェーズが交互に動作する全体の処理を構成する．ここで，時間の進行は連続変化のみで生じるものとする．例えば図 2(c) において，初期化から直後の `setup` とイベントの処理，最初の離散変化までは時刻 t_0 で生じるとし，その次の連続変化で初めて時間が進行し，時刻 t_1 まで進むものとする．

本計算モデルでは，時間の扱いに関して以下の 2 つの近似を行っている．1 つ目は，`setup`，`draw` 以外のイベント処理が直前の `setup`，`draw` と同じ時刻に行われるとしている点である．これらのイベント処理はキーの入力やマウスの移動に応じたものであり，本来はこれらのイベントの発生までに時間が進行しているはずであるが，本計算モデルでは時間の進行を考えない．2 つ目は，連続変化で次の `draw` の呼び出しの直前まで時間が進行するとしている点である．p5.js と同様，`draw` の呼び出しは一定時間ごとに行われるため，連続変化ではその分だけ時間が進行する．通常の HCCP で，連続変化の終了は，採用されていた制

約のガードが偽になることで生じるため，連続変化の長さが一定になるとは限らないが，本計算モデルでは連続変化も一定時間ごとに区切り，若干の時間のずれは許容する．

これらの時間の扱いに関する近似を行っていても，CP フェーズにおけるガードに応じた処理と，IP フェーズにおけるイベントに応じた処理が可能である．また，これらのフェーズは他方の変数の値を読み取ることで情報を交換できるため，協調動作することが可能である．

6 実装

提案言語 P5CP の処理系は現在実装中である．本処理系は P5CP のプログラムから JavaScript のプログラムへのトランスレータであり，このために Scala 言語とパーサ生成系 ANTLR を用いている．CP の実行で必要となる制約解消系は新たに JavaScript で開発したものであり，常微分方程式の解消には改良 Euler 法を採用している．また，IP の実行のために p5.js を用いている．

7 おわりに

本論文では，JavaScript と p5.js の組合せに対して制約プログラミングの機能を加えることで，新しい制約命令型プログラミング言語 P5CP を構築した．本言語では，命令型プログラミングで使われるイベントの概念と，制約プログラミングで使われるガードの概念を利用することで，これら 2 つのプログラミング方式を統合した．

今後の課題は，多様な対話型視覚的アプリケーションを実現できるように本言語を拡張することである．特に現在は検討の十分でない制約の動的生成や，構文的に対応していない制約の消去，コレクションを通じた不定個の CP 変数に関する制約の定義については検討の必要がある．また，記述できる制約の種類を増やすことも今後の課題である．

謝辞 本研究は JSPS 科研費 25540029 の助成を受けたものである．

参 考 文 献

[1] Badros, G. J., Borning, A., and Stuckey, P. J.: The Cassowary Linear Arithmetic Constraint Solving Algorithm, *ACM Trans. Comput.-Human Interact.*, Vol. 8, No. 4 (2001), pp. 267–306.

[2] Bainomugisha, E., Carreton, A. L., van Cutsem, T., Mostinckx, S., and de Meuter, W.: A Survey on Reactive Programming, *ACM Comput. Surveys*, Vol. 45, No. 4 (2013), pp. 52:1–34.

[3] Borning, A. and Duisberg, R.: Constraint-Based Tools for Building User Interfaces, *ACM Trans. Gr.*, Vol. 5, No. 4 (1986), pp. 345–374.

[4] Borning, A., Freeman-Benson, B., and Wilson, M.: Constraint Hierarchies, *Lisp Symbolic Comput.*, Vol. 5, No. 3 (1992), pp. 223–270.

[5] Dijkstra, E. W.: Guarded Commands, Nondeterminacy and Formal Derivation of Programs, *Comm. ACM*, Vol. 18, No. 8 (1975), pp. 453–457.

[6] Felgentreff, T., Borning, A., and Hirschfeld, R.: Babelsberg: Specifying and Solving Constraints on Object Behavior, *J. Object Tech.*, Vol. 13, No. 4 (2014), pp. 1:1–38.

[7] Felgentreff, T., Borning, A., Hirschfeld, R., Lincke, J., Ohshima, Y., Freudenberg, B., and Krahn, R.: Babelsberg/JS: A Browser-Based Implementation of an Object Constraint Language, *Proc. ECOOP*, LNCS, Vol. 8586, 2014, pp. 411–436.

[8] Freeman-Benson, B. and Borning, A.: The Design and Implementation of Kaleidoscope'90, A Constraint Imperative Programming Language, *Proc. ACM ICCL*, 1992, pp. 174–180.

[9] Freeman-Benson, B. N., Maloney, J., and Borning, A.: An Incremental Constraint Solver, *Comm. ACM*, Vol. 33, No. 1 (1990), pp. 54–63.

[10] Goldberg, A. and Robson, D.: *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley, 1983.

[11] Green, M.: The University of Alberta User Interface Management System, *Proc. ACM SIGGRAPH*, 1985, pp. 205–213.

[12] Gupta, V., Jagadeesan, R., and Saraswat, V.: Computing with Continuous Change, *Sci. Comput. Program.*, Vol. 30, No. 1–2 (1998), pp. 3–49.

[13] Harel, D.: Statecharts: A Visual Formalism for Complex Systems, *Sci. Comput. Program.*, Vol. 8, No. 3 (1987), pp. 231–274.

[14] Harel, D. and Pnueli, A.: On the Development of Reactive Systems, *Logics and Models of Concurrent Systems*, Springer, 1985, pp. 477–498.

[15] Hudson, S. E.: A System for Efficient and Flexible One-Way Constraint Evaluation in C++, Tech. Rep. GIT-GVU-93-15, GVU Cent., Georgia Inst. Tech., 1993.

[16] Lopez, G., Freeman-Benson, B., and Borning, A.: Implementing Constraint Imperative Programming Languages: the Kaleidoscope'93 Virtual Machine, *Proc. ACM OOPSLA*, 1994, pp. 259–271.

[17] 松本翔太, 河野文彦, 上田和紀: ハイブリッド制約言

語 HydLa を用いたハイブリッドシステムの解析, 人工知能学会全国大会論文集, 2014, pp. 3O1-3in:1–3.
[18] McCarthy, L.: P5.js Overview, 2014. https://github.com/processing/p5.js/wiki/p5.js-overview .
[19] Myers, B. A., Giuse, D. A., Dannenberg, R. B., Zanden, B. T. V., Kosbie, D. S., Pervin, E., Mickish, A., and Marchal, P.: Garnet: Comprehensive Support for Graphical, Highly Interactive User Interfaces, *IEEE Comput.*, Vol. 23, No. 11 (1990), pp. 71–85.
[20] Myers, B. A., McDaniel, R. G., Miller, R. C., Ferrency, A. S., Faulring, A., Kyle, B. D., Mickish, A., Klimovitski, A., and Doane, P.: The Amulet Environment: New Models for Effective User Interface Software Development, *IEEE Trans. Softw. Eng.*, Vol. 23, No. 6 (1997), pp. 347–365.
[21] Pnueli, A.: Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends, *Current Trends in Concurrency*, LNCS, Vol. 224, 1986, pp. 510–584.
[22] Reas, C. and Fry, B.: *Processing: A Programming Handbook for Visual Designers and Artists*, MIT Press, 2007.
[23] Sadun, E.: *iOS Auto Layout Demystified*, Addison-Wesley, 2013.
[24] Sun Microsystems: Java AWT: Delegation Event Model, 1997.
[25] Sutherland, I. E.: Sketchpad: A Man-Machine Graphical Communication System, *Proc. AFIPS Spring Joint Conf.*, 1963, pp. 329–346.
[26] 上田和紀, 細部博史, 石井大輔: ハイブリッド制約言語 HydLa の宣言的意味論, コンピュータソフトウェア, Vol. 28, No. 1 (2011), pp. 306–311.