教育用プログラミング言語におけるオブジェクト共有機能の導入

兼宗　進[†]　中谷　多哉子[††]　御手洗　理英[†††]
福井　眞吾[†]　久野　靖[†]

近年，教育課程の改定により，小学校から高等学校までの初中等教育において，プログラミングを含む情報教育の導入が進められている．筆者らは，初中等教育で活用可能な教育用オブジェクト指向言語「ドリトル」を開発し，現場での評価を行ってきた．本稿では，プログラムの中からオブジェクトをネットワーク間で複製・共有して扱うドリトルの拡張について報告する．この機能により，ある生徒がドリトルの任意のオブジェクトに名前を付けて公開したときに，他の生徒はそのオブジェクトを自分のプログラムに取り込んで再利用したり，共有して使うことが可能になった．共有を使うことで，生徒が個人ごとに独立したプログラミングを行うだけでなく，複数の生徒が共同で作業する形のプログラミングを行うことが可能である．共有機能の実装は，Java2 で記述されたドリトルの処理系に，RMI (Remote Method Invocation) を用いて行った．

Design and Implementation of Object Sharing for Dolittle Language

Susumu Kanemune ,[†] Takako Nakatani ,[††] Rie Mitarai ,[†††]
Shingo Fukui [†] and Yasushi Kuno [†]

Recently, IT education curriculum at K12 (kindergarten and 12-year-education) schools are being started in Japan. In this presentation, we describe design and implementation of object sharing for “Dolittle” programming language. Students can release their objects into network in the classroom. Then other students can copy or share objects in their programs. By using object sharing, students not only can make program by oneself, but also can make program with collaboration. We implemented object sharing using Java RMI (Remote Method Invocation).

1. はじめに

教育課程の改定により，小学校から高等学校までの初中等教育において，情報教育が開始された．筆者らはこれまで,「プログラミングの体験により計算機の動作を体験的に学ぶ」学習環境を提案し，教育用オブジェクト指向言語「ドリトル」の配布を行ってきた.[9)10)13)]

また，近年ではインターネット環境の普及が急速に進んでいる．平成 15 年に公開された総務省の統計[12)]によると，平成 9 年末にわずか 6.4%だったインターネットの世帯普及率☆は，平成 14 年末現在で 80%を超え，特に 13〜29 歳の若年層の利用率は 90%近くに達している．

このような背景から，今後の情報教育においては，スタンドアロンの計算機環境だけでなく，日常接する情報通信の中で計算機の果たす役割を理解することが重要になってくると考えられる．

そこで，教育用のオブジェクト指向言語「ドリトル」の拡張により，オブジェクトをネットワーク環境で活用する学習環境を提案する．

2. 情報教育におけるプログラミング言語

2.1 求められる特徴

ドリトルの開発に当たっては，入門用のプログラミング言語に求められる特徴として次の条件を考慮した.[9)]

† 筑波大学大学院 ビジネス科学研究科
Graduate School of Systems Management, University of Tsukuba, Tokyo.
†† （有）エス・ラグーン
S-Lagoon Co.,Ltd.
††† （株）アーマット
Armat Corporation

☆ パーソナルコンピューターや携帯電話等の電子機器により，WWW の閲覧や電子メールを使用している個人のいる世帯の比率を示している．

- オブジェクト指向言語であること
- プロトタイプ方式であること
- テキストに基づくソースコード
- 簡潔であること
- 日本語との対応性
- 階層的な構文を避ける

また，画面上の図形，タートル，GUI 部品などのオブジェクトにメッセージを送って操作するモデルにより，プログラミングの初心者が興味を持ってプログラミングに取り組むことができ，オブジェクトを主体としたプログラミングを扱えることを示した.[10)]

今回提案する分散共有ドリトルでは，ドリトルの入門用言語としての利点を維持しつつ，新たにネットワーク上でオブジェクトを扱う機能を加えることで，個人ごとのプログラミングだけでなく，他の生徒とコラボレーション作品を作成したり，ネットワークでの通信をプログラムを通して体験することを可能にする.

ドリトルをネットワークに拡張するに当たり，次の特徴を取り入れることに留意した.

- ネットワークプログラミングの未経験者が扱えること
- プログラミングを通して，日常接する情報通信の原理を類推できること
- ネットワークプログラミングや分散プログラミングを学ぶことは目的としない．教員の適切な助言により，プログラミングを通して「なるほど，こういう原理で動いているのか」と思えることが重要と考える
- ドリトルのオブジェクトモデルを維持する．画面上のオブジェクトにメッセージを送り結果を視覚的に確認するモデルはネットワーク上のオブジェクトに対しても扱えるようにする

2.2 既存言語における分散共有機能の検討

分散共有ドリトルの設計にあたり，既存の言語について分散共有機能を検討した.

LOGO[7)] は教育用に設計された言語である．画面に視覚的な操作対象 (タートル) を置き，動かした軌跡を画面に残すタートルグラフィックスにより，学習者は自分の行った操作を確認しながらプログラムを作ることができる．ドリトルは LOGO のタートルグラフィックスを取り入れ，軌跡を図形として扱えるように拡張して利用している．しかし，LOGO 自体はオブジェクト指向言語ではなく，タートルなどの画面上の操作対象をネットワークを介して扱う機能も提供されていない.

Smalltalk[2)] の実装である Squeak[5)] の上に構築された子供用のプログラミング環境として，SqueakToys[8)] がある．SqueakToys では，画面上のパレットと呼ばれる領域に，値や制御構造を表すタイルを置く形でプログラムを記述する．タイルを用いることでテキストの入力を不要としていることが特徴である．SqueakToys ではオブジェクトをプログラミングの基本とするが，ネットワークを介した操作を提供するものではない.

```
require 'drb/drb'

DRb.start_service

uri = "druby://myhost:8470"
counter = DRbObject.new(nil, uri)
p counter.up
p counter.up
counter.reset
p counter.up
```

図 1 dRuby のプログラム例 (文献 11) より引用して改変)
Fig. 1 An example program in dRuby

Java[3)] はネットワークでの利用を考慮して設計されたオブジェクト指向言語である．Java には RMI をはじめ，JINI，CORBA[6)]，HORB[4)] など各種の分散オブジェクト指向環境が提供されている．しかし，教育を想定した言語ではないため，初心者が利用するには敷居が高いという問題がある.

スクリプト言語である Ruby[14)] を分散環境に拡張したプログラミング環境として，dRuby[11)] がある．dRuby は Ruby で記述されており，リモートに存在する Ruby オブジェクトのメソッド呼び出しを可能にする．**図 1** に dRuby のサンプルを示す．この例では，サーバーに `"druby://myhost:8470"` で接続し，サーバー上の counter オブジェクト (数の数え上げを行うオブジェクト) にいくつかのメッセージを送信している．サーバー上のオブジェクトにメッセージを送り実行する dRuby のモデルは分散共有ドリトルと共通する部分が多いが，汎用言語である Ruby が画面に表示されないオブジェクトを基本とするのに対し，ドリトルでは画面上に表示されるオブジェクトを中心に操作する点が異なっている.

3. プログラミング言語「ドリトル」

図 2 にドリトルの構文を，**図 3** にドリトルのプログラム例を示す．以下ではこの例を使い，ドリトルの構文を解説する.

(1) で，“タートル” は広域変数名であり，そこには予め用意されたタートルオブジェクトが格納されてい

```
プログラム ::= (文 '。')…
文 ::= 代入文
  | メソッド定義
  | 式
代入文 ::= 変数 '=' 式
メソッド定義 ::= 変数 '=' ブロック
変数 ::= [項 ':']  名前
式 ::= 単純式 | メッセージ送信
メッセージ送信 ::= [レシーバ] '!'  メッセージ
レシーバ ::= 項
メッセージ ::=
  引数… メソッド名 ([';'] 引数… メソッド名)…
引数 ::= 単純式
メソッド名 ::= 名前
括弧 ::= '(' 中置式 ')' | '(' メッセージ送信 ')'
単純式 ::=
  数値リテラル | 文字列リテラル | 括弧 | ブロック
ブロック ::=
  '「' ['|' 名前… '|'] 文 ('。' 文)… '」'
中置式 ::= 中置式 演算子 中置式 | 項
項 ::= 単純式 | 名前
演算子 ::= '+' | '-' | '*' | '/' | '>' | '<'
  | '>=' |'<=' | '==' | '!='
```

図 2 ドリトルの構文
Fig. 2 Syntax of Dolittle

```
カメ太=タートル!作る。                          (1)
カメ太!100 歩く 120 左回り 100 歩く 閉じる。    (2)
三角形=カメ太!図形にする（赤）塗る。            (3)
時計=タイマー!作る 1 間隔 10 時間。             (4)
三角形:ぐるぐる
    =「|x| 時計!「! (x) 右回り」実行」。        (5)
実行ボタン=ボタン!"実行" 作る。                 (6)
実行ボタン:動作=「三角形! 36 ぐるぐる」。       (7)
```

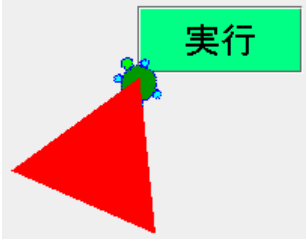

図 3 サンプルプログラムと実行例
Fig. 3 An example program in Dolittle

る．☆ "作る" メソッドは新しいオブジェクトを生成し，そのプロトタイプをタートルオブジェクトにする．これにより，以後このオブジェクトはタートルオブジェクトのプロパティおよびメソッド一式と同じものを予め持つかのようにふるまう．"カメ太" はここで新たに作成する広域変数名であり，そこに新しく作ったオブジェクトを格納する．ドリトルでは変数（オブジェクトのプロパティ）は最初に値を格納した時に作られ，予め宣言する必要はない．

メッセージは任意個数の引数を持てる．引数のうち数値リテラルや文字列リテラルはそのまま書けるが，変数参照や一般の式は「()」で囲む必要がある．識別子が来るとそれがメッセージセレクタとして認識され，そこまでが 1 つのメッセージ送信式となる．メッセージはオブジェクトを値として返すので，その右側に続けて引数とメッセージセレクタを書くことで，(2) のように返されたオブジェクトに対するメッセージを続けて指定できる．これをカスケード（直列）送信と呼ぶ．カスケード送信はいくつでも書くことができ，「。」でその最後を表す．

数字で始まるトークンは数値リテラルである．上の例には現れていないが，文字リテラルは文字を「""」または「『』」で囲んだものとして表す．

それ以外のリテラルは用意していないが，色などはその色名に対応する広域変数に値を格納することでプログラマの便宜を計る．たとえば，広域変数「赤」には赤色を表す色オブジェクトが格納されている．ここで，「()」の中に書くことでメッセージセレクタではなく変数への参照を表していることに注意されたい．

(3) で，タートルオブジェクトは "図形にする" メッセージを受け取ると，移動による軌跡から図形オブジェクトを生成する．生成されたオブジェクトに，色オブジェクトを引数として "塗る" メッセージを送っている．

(4) タイマーは，ブロックを一定間隔で実行するオブジェクトである．内部に実行間隔と実行時間の状態を持つ．ブロックは非同期に実行され，ブロックを起動した処理はブロックの終了を待たずに先に進む．1 つのタイマーに複数の処理を頼んだときは，タイマーはそれらを直列に実行する．タイマーの実行完了を待ちたければ，タイマーのメソッド「待つ」により同期を取ることができる．

回数を「1 回」とすれば，起動した処理と指定したブロックの並行実行の機能としても使うことができる．

(5) メソッドはオブジェクトのプロパティとしてブロックを格納することで定義する．ブロックは [...] または「…」で表し，その内側のコード列は後でブロックが評価される時に実行される．

ブロックは任意個数のパラメタを持つことができる．パラメタはブロックの先頭に「| 識別子…|」という形で指定する（指定がない場合は引数のないブロックとなる）．

パラメタはブロックの実行中だけ存在する無名のコ

☆ 広域変数はトップレベルオブジェクト「ルート」のプロパティとして格納されている．

ンテキストオブジェクトのプロパティとして扱われ，初期値としてブロック評価時に渡された実引数値が格納されている．

ブロックを格納しているプロパティはメソッドとして実行可能である．その際，メッセージセレクタより前に書かれた実引数値が 1 つずつパラメタに対応する．ブロックが持つパラメタより多い場合はそれは捨てられ，少ない場合はパラメタは未定義オブジェクトを初期値として持つ．ブロック内で最後に評価された式の値がメソッドの返値となる．

(6) “実行ボタン” という名前のボタンオブジェクトを作り，押されたときの動作を定義している．

ドリトルでは，ボタンオブジェクトの “動作” やタートルオブジェクトと図形オブジェクトの “衝突” など，特定の名前のメソッドを定義することで，そのオブジェクトのイベントを受け取ることができる．衝突メソッドは，タートルや図形オブジェクトが他のオブジェクトと重なったときに実行される．

このプログラムでは，画面に表示された “実行” ボタンを押すことにより，三角形に定義された “ぐるぐる” というメソッドが実行される．その結果，三角形が画面上で 1 秒ごとに 36 度ずつ回転するアニメーションが表示される．

4. ネットワーク機能

4.1 基本的な考え方

ドリトルでは，画面上のオブジェクトにメッセージを送り，操作の結果を視覚的に確認するモデルにより，初心者がオブジェクトを単位としたプログラミングを行える．また，グラフィックスや GUI 部品を使用したプログラミングを体験することを通して，日常接するさまざまな計算機の中でプログラムが動作していることを学ぶことができる．

ドリトルのオブジェクトは，変数 (プロパティ) や配列に格納し，取り出して使うことができる．分散共有ドリトルでは，オブジェクトの格納をネットワーク上に拡張する形で，初心者がオブジェクトの転送を容易に扱えるようにした．分散共有ドリトルのプログラミングを体験することを通して，日常接するさまざまなデータ通信の中で，プログラムが動作していることを学ぶことができる．☆

分散共有ドリトルの実行時には，ネットワーク上にオブジェクトサーバーのプロセスを起動しておく．オブジェクトサーバーには，名前を付けてドリトルのオブジェクトを登録することができる．登録したオブジェクトは，複製と共有という 2 通りの活用を行えるようにした．

オブジェクトの複製 (**図 4**) は，サーバからオブジェクトの複製を取り出す．取り出されたオブジェクトは通常のローカルなオブジェクトとなる．この機能はサーバを介して複数のドリトル環境 (クライアント) 間でオブジェクトや値をやりとりするために用いる．

オブジェクトの共有 (**図 5**) は，サーバ上のオブジェクトを共有する形で取り出す．この場合，取り出されるのはサーバ上のオブジェクトを参照する「共有オブジェクト」となり，共有オブジェクトに対する操作は，そのサーバ上のオブジェクトに対応するすべての共有オブジェクトに影響を及ぼす．

ドリトルのオブジェクトは画面上に見えるオブジェクトと見えないオブジェクトに大別される．見えないオブジェクトは内部的に値やメソッドを保持して動作するだけなので、見えないオブジェクトを共有する場合はサーバ上のオブジェクトが唯一の実体となり，取り出された共有オブジェクトは単なるプロキシとなってすべての操作をサーバに中継するだけである．

一方，見えるオブジェクトの場合は，オブジェクトの変化が各クライアントでの画面の変化として現れ，また画面上のオブジェクトへの操作がサーバ上のオブジェクトに伝えられる必要がある．このため，サーバ上のオブジェクトが MVC(Model-View-Controller) モデルでいう Model，各共有オブジェクトが View+Controller になるように切り分けを行った．Model に対する変更は Observer パターンと同様の仕組みで View+Controller に伝達されるが，これをプログラミング初心者が理解し制御するのは難しいと考えたので，サーバから共有オブジェクトを取り出すだけで View+Controller の生成と Observer の設定が自動的に行われるようにしている．現在のところ、見えるオブジェクトとしては予め用意されているタートル，図形オブジェクト，GUI 部品オブジェクトのいずれかをプロトタイプに持つものだけが使えるが，将来的にはユーザが独自の見えるオブジェクトを定義できるようにしたいと考えている．

以下では，登録，複製，共有の機能を順に説明する．

4.2 オブジェクトの登録と複製

図 6 に，オブジェクトサーバーにオブジェクトを登録・複製するプログラム例を示す．

あるクライアント A で実行することを考える．

(1) で，タートルオブジェクトを生成する．

☆ 分散共有ドリトルでは,「ネットワークプログラミング技術の習得」を主要な目的とはしていない．そこで，ソケット通信などの習得が難しいと思われるモデルは採用しなかった．

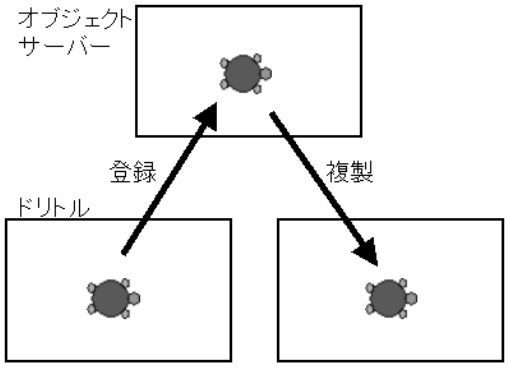

図 4 オブジェクトの登録と複製
Fig. 4 Registration and duplication of objects

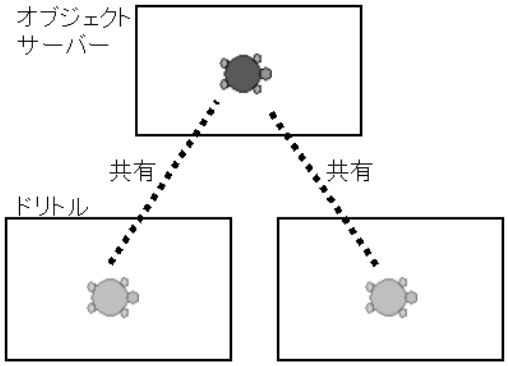

図 5 オブジェクトの共有
Fig. 5 Sharing of objects

(2) で，サーバー “sv1” に接続する．引数には IP アドレスまたはホスト名を指定できる．

(3) で，サーバーにオブジェクト “カメ太” を “kame1” という名前で登録する．

(4) で，サーバーから “kame1” を複製し，カメ吉という名前にする．

(5) で，画面上の 2 つのオブジェクト (カメ太とカメ吉) を同時に動かす．サーバーから複製したオブジェクトは，ローカルのオブジェクトと区別することなくメッセージを送り操作することが可能である．

4.3 オブジェクトの共有

図 7, **図 8** にプログラム例を示す．

クライアント B で図 7 を，クライアント C で図 8 を実行することを考える．

(1) で，サーバー “sv1” に接続する．

(2) で，サーバー上のオブジェクト “kame1” を共有し，“カメ助” というローカルの名前にする．

(3) で，クライアント C から共有オブジェクトを操作する．クライアント C の共有オブジェクトに送られたメッセージはサーバー上のオブジェクトに転送され，

```
// ローカルオブジェクトの生成 (1)
カメ太=タートル!作る。

// サーバーへの接続 (2)
サーバー!"sv1" 接続。

// オブジェクトの登録 (3)
サーバー!"kame1"（カメ太）登録。

// オブジェクトの複製 (4)
カメ吉=サーバー!"kame1" 複製。

// オブジェクトの操作 (5)
時計=タイマー!作る
時計!「カメ太!10 歩く。カメ吉!15 歩く」実行。
```

図 6 オブジェクトの登録と複製
Fig. 6 An example of registration and duplication of objects

```
// サーバーへの接続 (1)
サーバー!"sv1" 接続。

// オブジェクトの共有 (2)
カメ助=サーバー!"kame1" 共有。
```

図 7 オブジェクトの共有 (1)
Fig. 7 An example of sharing of objects (1)

```
// サーバーへの接続 (1)
サーバー!"sv1" 接続。

// オブジェクトの共有 (2)
カメ助=サーバー!"kame1" 共有。

// オブジェクトの操作 (3)
時計=タイマー!作る
時計!「カメ助!10 歩く」実行。
```

図 8 オブジェクトの共有 (2)
Fig. 8 An example of sharing of objects (2)

実行が行われる．サーバー上のオブジェクトの状態が変化すると，クライアント B, クライアント C の共有オブジェクトも画面上で位置の変化などを反映する．

5. 実　　装

5.1 分散共有ドリトルの構成と動作環境

ドリトルの処理系は Java 2[3] により記述されており，Java 2 が動くさまざまな環境で動作する．字句

表 1 ドリトルのオブジェクト
Table 1 Library objects of Dolittle

オブジェクト	説明
タートル	タートルグラフィックスの機能を提供するオブジェクト
図形	図形を表すオブジェクト．タートルに“図形にする”メソッドを送ることで生成される
GUI 部品	ボタン，ラベル，フィールド，リスト，選択メニューを表すオブジェクト
配列	一連の値の並びを表すオブジェクト
タイマー	ブロックを指定した“実行”メソッドにより，一定間隔で一定時間，操作を実行するオブジェクト
数値	数値を表すオブジェクト (immutable)
文字列	文字の並びを表すオブジェクト (immutable)
論理値	真偽値を表すオブジェクト (immutable)
色	色を表すオブジェクト (immutable)
ブロック	動作の列を表すオブジェクト (immutable)
サーバー	オブジェクトサーバーとの通信を行うオブジェクト
シリアルポート	外部機器を制御するオブジェクト

図 9 ドリトルの実行画面
Fig. 9 Execution of Dolittle

解析，構文解析は SableCC[1] による生成コードを利用し，これに加えて意味解析，インタプリタ，ユーザインタフェース，および標準オブジェクト群を実装している．**表 1** に，ドリトルの提供するオブジェクトを示す．

図 9 にドリトルの実行画面を示す．実行画面と編集画面を画面上部のタブで切り替えて操作する．画面下部には実行などのボタンが並んでいる．

図 10 に分散共有ドリトルの構成を示す．ネットワーク上にオブジェクトサーバーを置き，ドリトルのオブジェクトを格納する．オブジェクトサーバーは画面を持ったプロセスである．ドリトルと同一のホスト上で動作してもよく，また，ネットワーク中に複数存在してもよい．

図 11 にオブジェクトサーバーの実行画面を示す．

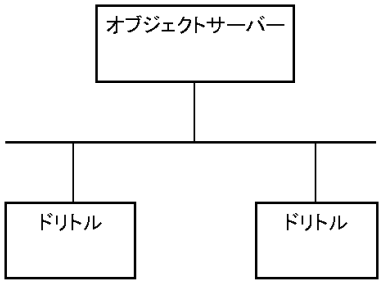

図 10 分散共有ドリトルのシステム構成
Fig. 10 Topology of Dolittle and Object Server

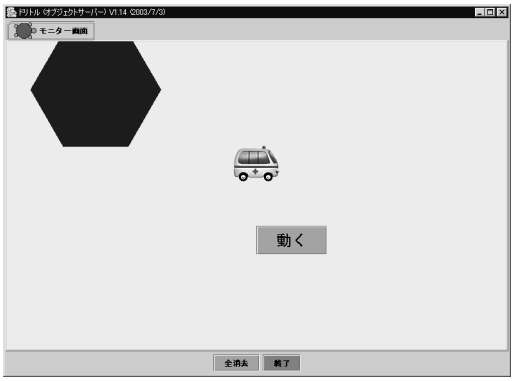

図 11 オブジェクトサーバーの実行画面
Fig. 11 Execution of Object Server

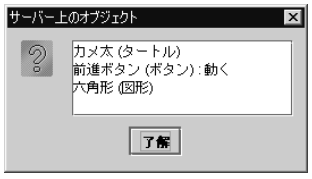

図 12 オブジェクトリストの表示
Fig. 12 An object dialog

モニター画面には，サーバーに登録されたオブジェクトが表示される．この例では，3 種類のオブジェクト (図形，タートル，ボタン) が表示されている．

オブジェクトサーサーバーはドリトルと似た画面インターフェースを使用しているが，オブジェクトサーバーにはプログラムを編集・実行するインターフェースは用意されていない．また，画面上のオブジェクトは表示されるだけであり，画面上のボタンを押してもオブジェクトにメッセージは送られない．

サーバーに登録されたオブジェクトの一覧は，クライアント側のドリトルから確認できる．**図 12** に，サーバー上のオブジェクトを表示するダイアログを示す．

5.2 ネットワークへの拡張

ドリトルのオブジェクトをネットワーク経由で動か

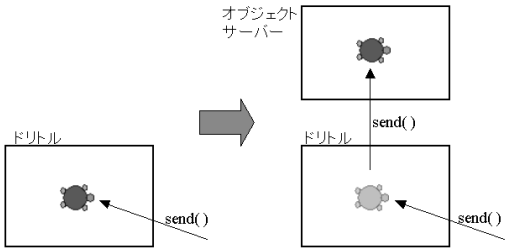

図 13 ローカル実行とリモート実行
Fig. 13 Local execution and remote execution

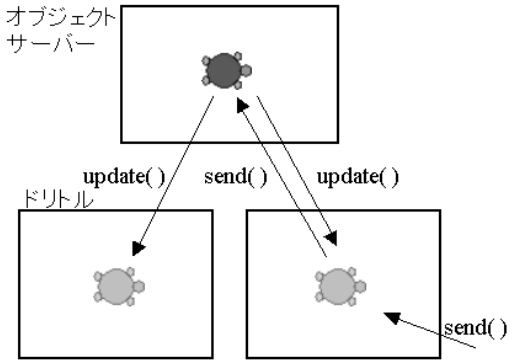

図 14 オブジェクトの共有
Fig. 14 Message sending to sharing objects

す場合について説明する (**図 13**)．通常の (ローカルな) オブジェクトに対するメッセージ送信は直接ローカルな環境内で実行される．これに対し，共有オブジェクトに対するメッセージ送信はすべてサーバ上の対応するオブジェクトに転送され，そこで実行される．すなわち，共有オブジェクトは単純なプロキシとして動作する．

サーバ上のオブジェクトは複数のクライアントによって共有可能である (**図 14**)．共有オブジェクトの 1 つにメッセージが送られると，上と同様にして実際の動作はサーバ上のオブジェクトが行い，サーバ上のオブジェクトの状態が変化させられる．この変化は他のクライアントから (その共有オブジェクトを通じて) 観測可能であり，これにより情報の交換が行える．

タートルや図形など，画面に見えるオブジェクトの場合，共有オブジェクトは単なるプロキシではなく，View+Controller の役割を持ち，画面への表示やマウスクリック，他のオブジェクトとの衝突検知など画面にかかわる動作を受け持つ．一方，オブジェクトの位置や色など Model が持つ情報はサーバ側のオブジェクトに保持され，これらを取り扱うメソッドに関しては共有オブジェクトはプロキシとして動作する．Model の状態が変化した場合は Oberver パターンによるコールバックがなされ，共有オブジェクトはそれに応じて画面上のオブジェクトの表示を更新する．

表 2 複製・共有可能なオブジェクト
Table 2 The objects which can duplicate and share

オブジェクト	登録・複製	共有
タートル	○	○
図形	○	○
GUI 部品	○	○
配列	○	○
タイマー	○	○
数値	○	×
文字列	○	×
論理値	○	×
色	○	×
ブロック	○	×
サーバー	×	×
シリアルポート	×	×

ドリトルのオブジェクトは原則として登録・複製の対象となるが (**表 2**)，機器の入出力や通信を担うオブジェクト (サーバー，シリアルポート) は除外している．また，共有は状態を持たない (immutable) なオブジェクトに対しては複製と区別できないため，そのようなオブジェクトに対して共有を指示した場合は単に複製を返すようになっている (表 2 で左の欄が○，右の欄が×のもの)．

以下では，図 6〜図 8 のサンプルを使い，接続，登録，複製，共有，共有実行の内部動作を説明する．

5.3 接　　続

サーバー！`"sv1"` 接続。

ローカルの “サーバー” オブジェクト (OxStore) は，“接続” メッセージを受け取ると，次の手順でサーバー側のオブジェクト (RemoteStoreServer) と接続する．

- 次の形式でサーバーを lookup する．
  `rmi://`ホスト`:`ポート`/bind`名
- 得られたサーバーオブジェクト (RemoteStoreServer) の参照をローカル変数 store に保持する

ここで，ホストは “接続” メッセージに引数として指定された値である．引数が省略された場合は “localhost” が使われる．ポートは，ドリトルの初期化ファイルに定義された値を使う．デフォルトは “2020” である．bind名は，ドリトルの初期化ファイルに定義された値を使う．デフォルトは “d0” である．

サーバー側では，接続を受け付けるオブジェクト (RemoteStoreServer) を bind して待機する．

5.4 登　　録

サーバー！`"t1"`（カメ太）登録。

クライアントの “サーバー” オブジェクトは，“登録” メッセージを受け取ると，次の手順でサーバーにオブ

ジェクトを登録する．

- オブジェクトを前処理する
- オブジェクトのプロパティを平滑化する
- サーバーに登録する
- オブジェクトを後処理する

前処理では，Java で直列化 (serialize) できない要素を直列化可能なオブジェクトに変換する処理を行う．この処理の例としては，タートルオブジェクトの画像情報である Image を ImageIcon に変換する処理と，図形オブジェクトの幾何学情報である GeneralPath を転送用の配列データに変換する処理がある．直列化できないデータは transient として宣言されており，サーバーには転送されない．☆

プロパティの平滑化では，転送するオブジェクトに親オブジェクトのプロパティを格納する処理を行う．平滑化の詳細は後述する．

登録処理では，サーバーオブジェクトに対して put() メソッドを実行することにより，サーバーに名前とオブジェクトを転送し，オブジェクトを登録する．

オブジェクトの後処理では，前処理で生成した転送用のデータをクリアする．

サーバー側では，次の手順で登録を行う．

- 登録名とオブジェクトを名前テーブル (objmap) に登録する
- オブジェクトに後処理をする
- オブジェクトに親オブジェクトを登録する
- ドリトルの広域変数として，登録名でオブジェクトを登録する

名前テーブルは，サーバーに登録されたオブジェクトを管理するテーブルである．名前からオブジェクトを参照する辞書を提供する．名前テーブルへの登録では，登録名でオブジェクトを登録する．すでに同名のオブジェクトが存在した場合には上書きされる．

オブジェクトの後処理では，転送用に加工されたデータを戻す処理を行う．具体的には，転送用のデータからタートルオブジェクトの画像と図形オブジェクトの幾何学データを復元する．この処理により，登録したオブジェクトがサーバー画面に表示される．

登録したオブジェクトの親オブジェクトを設定する．親オブジェクトには，存在すればプロトタイプオブジェクトを設定し，そうでない場合はルートオブジェクトを設定する．この処理により，共有時にサーバー上でオブジェクトを実行できる．

☆ Java で transient 修飾子のついたインスタンス変数は直列化されず，復元されたオブジェクトにおいてそのような変数の値は null になる．

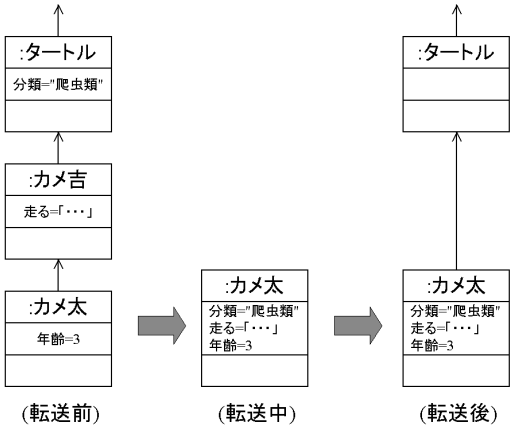

図 15 オブジェクトの平滑化
Fig. 15 Flatten of an object

広域変数への登録では，サーバー上のドリトルの名前空間にオブジェクトを登録する．この処理により，共有時にサーバー上でオブジェクトを実行できる．

ドリトルのオブジェクトは，親リンクとプロパティにより，他のオブジェクトとのリンク関係を持つ．あるオブジェクトをサーバーに登録するとき，転送される範囲は次のように決定される．

- プロパティは転送する．
- 親オブジェクトは転送せず，代わりに平滑化を行う．

  平滑化とは**図 15** に示すように，親オブジェクトのプロパテイを集めて来て子オブジェクトに格納する操作である．平滑化を行うのは，ドリトルのようなプロトタイプ方式のオブジェクト言語では，オブジェクトのプロパティは概念的には親オブジェクトのプロパティまでを含めた全体であるため，これらを合わせて転送しなければ転送先でオブジェクトが正しく動作させられないためである．親が多段になっている場合にも平滑化は再帰的に行うが，システム標準のオブジェクト (ルートオブジェクトを含む) や共有オブジェクトが現れた場合はそこまでで止める．

5.5 複　　製

カメ吉＝サーバー！`"t1"` 複製。

クライアントの “サーバー” オブジェクトは，“複製” メッセージを受け取ると，次の手順でサーバーからオブジェクトを複製する．

- サーバーからオブジェクトを複製する
- オブジェクトを後処理する
- オブジェクトに親オブジェクトを登録する

複製処理では，サーバーオブジェクトに対して get-

Copy() メソッドを実行することにより，サーバーからオブジェクトを転送する．

オブジェクトの後処理では，転送用に加工されたデータを戻す処理，画面に登録する処理，イベントリスナーを設定する処理を行う．タートルオブジェクトと図形オブジェクトでは，転送用のデータから画像と幾何学データを復元する．ボタンなどの GUI オブジェクトとタートルオブジェクト，図形オブジェクトでは，画面に登録する処理が行われる．ボタンオブジェクトとテキストフィールドオブジェクトでは，マウスやキーボードからのイベントを受け取れるようにリスナーの設定を行う．これらの処理により，複製したオブジェクトがクライアント画面に表示され，適切な動作を行えるようになる．

複製したオブジェクトの親オブジェクトを設定する．親オブジェクトは，存在すればプロトタイプオブジェクトを設定し，そうでない場合はルートオブジェクトを設定する．この処理により，複製したオブジェクトをローカルで実行できるようになる．

複製処理はオブジェクトをローカル側で生成する処理までを行う．広域変数への登録は，複製の結果を変数に代入する形でプログラム中で明示的に行う必要がある．

サーバー側では，次の手順で複製を行う．

- 名前テーブル (objmap) からオブジェクトを検索する
- 検索されたオブジェクトを返す

サーバー上のオブジェクトは，平滑化や直列化できないデータの変換など，登録時に転送に適した形式に加工されている．よって，複製時の前処理は不要である．

5.6 共　　有

カメ吉＝サーバー！`"t1"` 共有。

ドリトルのオブジェクト共有は，サーバー上のオブジェクトを複数のクライアントから共有する．基本的な考え方は，サーバー側のオブジェクト (リモートオブジェクト) の参照をクライアントに置き，それにメッセージを送り実行するモデルである．

実際の実装では，次の特性を考慮し，メッセージを中継するプロキシーだけではなく，クライアント側にドリトルのオブジェクトを生成することで，参照と実体を混在させる形を取った．

- ドリトルのプログラムでは，画面に表示されるオブジェクトが使われることが多い．
- 画面上で他のオブジェクトとの衝突や，ボタン押下によるイベント処理などが必要になる

図 16 に，オブジェクト共有の構成を示す．クライアントの “サーバー” オブジェクトは，“共有” メッセージを受け取ると，次の手順でサーバー側のオブジェクトからローカルの共有オブジェクトを生成する．

- サーバーに getShared() メソッドを実行する
- サーバーから，リモートオブジェクトの参照が返る
- ローカルオブジェクトの核となるプロキシーオブジェクト (OxRemote) を生成し，リモートオブジェクトへの参照 (sv) を格納する
- リモートオブジェクトに getClass() を実行し，リモートオブジェクトのクラスを調べる
- ローカル側に，リモートオブジェクトと同じクラスのインスタンスを生成する．これが共有オブジェクトとなる
- 生成した共有オブジェクトへのメッセージが OxRemote オブジェクトに転送されるようにする．以後，共有オブジェクトへのメッセージやプロパティ操作は OxRemote を通してサーバー側で実行される
- 共有オブジェクトのスケルトンであるコールバックオブジェクト (RemoteCallbackServer) を生成し，サーバーに伝える
- リモートオブジェクトの内部状態を共有オブジェクトに複製する
- 共有オブジェクトに複製後の後処理をする．後処理では，転送用に加工されたデータを戻す処理，画面に登録する処理，イベントリスナーを設定する処理を行う
- ローカルサーバーオブジェクト (OxStore) に，共有オブジェクトとコールバックオブジェクトを登録する

サーバー側では，次の手順で共有処理を行う．

- 名前テーブル (objmap) からオブジェクトを検索する
- 検索されたオブジェクトのスケルトンを作り，スケルトンテーブル (sharemap) に入れる
- スケルトンをクライアントに返す

また，クライアントが生成したコールバックオブジェクトは，スケルトン中のテーブル (callbacks) に格納される．

5.7 共 有 実 行

カメ吉！`100` 歩く。

共有オブジェクトに送られたメッセージは，次の手順で実行される．

- クライアントの共有オブジェクトは，リモートオブジェクトのスケルトン (RemoteObjectServer)

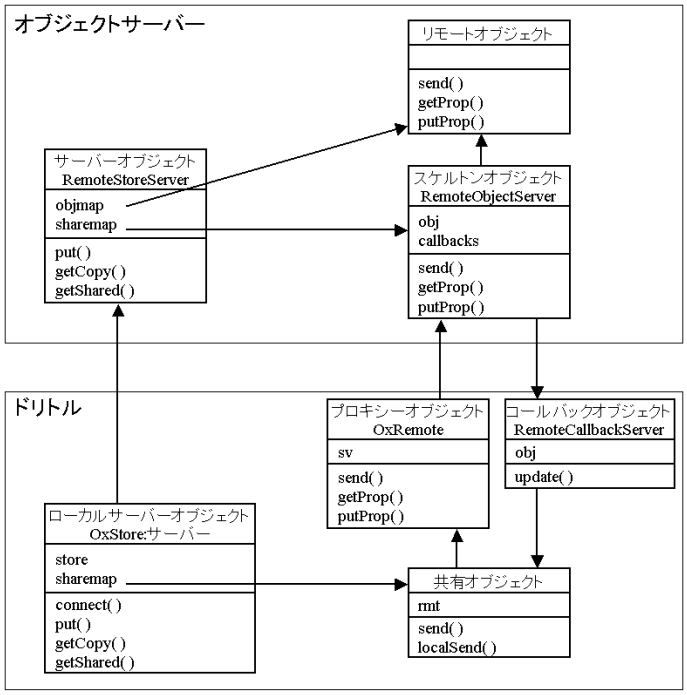

図 16 オブジェクトの共有
Fig. 16 An diagram of object sharing

に send() メッセージを転送する

- リモートオブジェクトのスケルトンは，リモートオブジェクトを共有しているクライアントのコールバックオブジェクトに対して,「順に」update() メッセージを転送する☆
- クライアント側のコールバックオブジェクトは，スケルトンからメッセージを受け取ると，共有オブジェクトの localSend() を実行し，クライアント側でのメソッド実行を行う
- リモートオブジェクトのスケルトンは，リモートオブジェクトに send() メッセージを送り，サーバー側での実行を行う
- 実行の結果がクライアントに返る

5.8 最 適 化

共有オブジェクトの実行は，当初は実行速度が遅く実用性に問題があった．そこで，2 種類の最適化を行うことで改善を行った．

5.8.1 リモート呼び出しの最適化

当初はクライアント・サーバー間のすべての通信でオブジェクトを転送していたが，次の 2 つの問題があった．

- 実行速度の問題．タートルオブジェクトなど，画像のように大きなデータを持つオブジェクトを毎回転送するのはシステムへの負荷が高い
- カスケード実行の問題．実行の結果としてローカルオブジェクトが作られると，カスケード送信を行えない．次のプログラム例で，“100 歩く” はリモートオブジェクト (カメ吉) に送られるが，“90 右回り” は実行の戻り値であるローカルオブジェクトに送られてしまう．

```
カメ吉！100 歩く 90 右回り。
```

そこで，リモート実行の結果を参照で返すよう修正を行った．その結果，共有オブジェクトに対するメッセージのカスケード実行が可能になった．

リモートで実行された結果として返るオブジェクトは，共有オブジェクトとして各クライアントに生成される．クライアント側では，共有オブジェクトの参照テーブルを管理する．同じリモートオブジェクトに対応する共有オブジェクトは，テーブルから引いて返される．新たに共有オブジェクトを生成する処理は，画像などの転送が各クライアントに対して行われるため重い処理になるが，ドリトルのプログラムでは自分自身 (self) を返すメソッドが多いため，実際のプログラムにおいて大きな速度低下が発生することはなかった．

☆ 現在の実装では，メッセージはクライアントとサーバーでシリアルで実行される．あるオブジェクトを 2 つのクライアントから共有している場合には，メッセージの処理は「クライアント (× 2) + サーバー (× 1)」の時間が必要である．

5.8.2 プロパティ参照の最適化

ドリトルでは，オブジェクトのプロパティにブロックを格納することでメソッドを定義する．また，タートルオブジェクトと図形オブジェクトは，画面上で位置を移動したときに他のオブジェクトと重なると，相手から “衝突” というメッセージを受け取る．

共有オブジェクトは衝突メッセージを受け取ると，サーバーにメッセージを転送する．サーバーはオブジェクトを共有しているクライアントにコールバックする形で，各クライアント上でメッセージを実行する．クライアントでの実行時には，メソッドの定義を取得するために，サーバーに対して衝突という名前のプロパティを参照する．この処理において，衝突プロパティの値であるブロックの転送に時間がかかるという問題があった．

- 画面でオブジェクト同士が重なるたびに衝突メソッドが実行される．(回数が多く頻繁)
- 共有されたクライアントの数だけブロックの転送が行われる
- ブロックの転送は重い処理である

そこで，ブロックが immutable なオブジェクトであることを利用し，キャッシュすることでパフォーマンスの向上を図った．改良したプロパティ値の参照手順を示す．

- サーバーにプロパティの型を問い合わせる
- ブロック以外なら，値を転送する
- ブロックであれば，値の代りにハッシュ値を転送する．ローカル側では，ブロックのキャッシュを参照し，ヒットしない場合のみブロックを転送する

ブロック以外の immutable なドリトルのオブジェクトには数値，文字列，色，論理値などが存在するが，これらはデータ量が小さく転送のコストが軽いため，今回のキャッシュの対象とはしなかった．

5.8.3 最適化の効果

リモート呼び出しの最適化とプロパティ参照の最適化の効果を測定した．使用したプログラムを**図 17** に示す．このプログラムでは，サーバー上のタートルオブジェクトを共有し，そのオブジェクトに “歩く” メッセージを 200 回繰り返し送信する．結果を**表 3** に示す．測定に用いた環境は，クロック 750MHz の PentiumIII プロセッサを備えたパーソナルコンピュータを使用した．メモリは 384MB を搭載し，OS は Windows 2000 Professional である．リモートオブジェクトでの実行で，サーバーを localhost に置く測定では，このマシン上でドリトルとオブジェクトサーバーを同時に実行した．

```
// リモートオブジェクトでの実行
サーバー!"localhost"接続。
t=サーバー!"t"共有。
b=ボタン!"実行"作る 100 0 位置。
b:動作=「「t!1 歩く」!200 繰り返す」。
```

```
// ローカルオブジェクトでの実行
t=タートル!作る。
b=ボタン!"実行"作る 100 0 位置。
b:動作=「「t!1 歩く」!200 繰り返す」。
```

図 17 共有実行速度測定プログラム
Fig. 17 An benchmark program

リモートオブジェクトでの実行では，別ホストにサーバーを置く測定も行った．サーバーを置いた環境は，クロック 1.8GHz の Celeron プロセッサを備えたパーソナルコンピュータを使用した．メモリは 512MB を搭載し，OS は Windows 2000 Professional である．2 台のマシン環境は 100BASE-TX のネットワークで接続した．

ドリトルのプログラムでは，タイマーオブジェクトを利用した定期的な繰り返し処理が用いられることが多い．スムーズなアニメーション表示を行うためには 0.1 秒程度の間隔でメッセージを処理する必要がある．そこで，今回の測定では 0.1 秒以内で実行できることを評価の目安とした．☆

最適化前の結果を見ると，1 回のメッセージ処理に約 0.5 秒を要している．メッセージを 0.1 秒以内に処理できないことから，共有オブジェクトを実用的なプログラムで使用することは困難であったことがわかる．

最適化後の結果を見ると，1 回のメッセージ処理に約 0.024 秒を要している．メッセージを 0.1 秒以内に処理できることから，実用的なプログラムで使用可能な速度に改善されたことがわかる．サーバーを別ホストに置いた場合でも，1 回のメッセージ処理は約 0.026 秒であり，大きな速度の変化は見られなかった．

参考までに，共有オブジェクトの代りにローカルオブジェクトで実行した結果を測定した．ローカルオブジェクトと比較したリモートオブジェクトの速度比は約 70 倍であった．

☆ ドリトルに付属する 9 種類のサンプルプログラムを調べたところ，タイマーの繰り返し間隔は 0.1 秒以上のプログラムが多いことがわかった．ブロック崩しゲームなど，一部の速度が要求されるプログラムを含めても，タイマーの繰り返し間隔は最短で 0.05 秒であった．

表 3 共有実行速度の測定結果
Table 3 Result of benchmark

テスト	実行時間		説明
	200 回 (秒)	1 回あたり (m秒)	
最適化前	104.0	520	リモートオブジェクト (サーバーは localhost で実行)
最適化後	4.7	23.5	リモートオブジェクト (サーバーは localhost で実行)
最適化後	5.2	26	リモートオブジェクト (サーバーは別ホストで実行)
ローカル	0.068	0.34	ローカルオブジェクト

6. ま と め

本論文では，教育用オブジェクト指向言語「ドリトル」を拡張し，ネットワーク上でオブジェクトを活用することが可能な分散共有ドリトルを提案した．

分散共有ドリトルを使うことで，たとえばタートルオブジェクトを共有して操作しながら，オブジェクトがネットワークを介して動作する様子を視覚的に観察することが可能になる．このような，ひとつの実体 (Model) を複数の参照 (View) から参照するモデルを，Objserver パターンのようなコードを記述せずに扱う機能を提供した．

以下は，分散共有ドリトルで今後検討すべき課題である．

- 情報教育での活用．教育現場での活用は，カリキュラムを含めて今後の課題である．教員と連絡を取りながら検証を進めたい．
- 高速化．ブロック転送の最適化を紹介したが，画像など転送にコストがかかるデータについて，改善の余地が残されている．また，現在はコールバック処理をシリアルに行っているため，メッセージの実行時間はクライアント数に比例して遅くなる．コールバック処理を非同期に行うことで速度の改善が期待できる．
- 実用的なプログラムの記述．学習用の教材として実用的なシステムを構築する場合には，オブジェクトの排他制御やトランザクション管理，メッセージの順序実行の保証などが必要になると考えられる．
- 実行結果の複製による返戻．現在は，共有オブジェクトへのメッセージ送信の結果はメソッドが返したオブジェクトが共有可能であれば常に共有オブジェクトとして返される．この仕様はカスケード実行の実現と実行速度向上の上で効果があったが，共有を望まない場合でも共有が強制されてしまうことになる．この部分を制御可能にすることを今後検討していきたい．

今後は，これらの課題への対応を行いながら，情報教育での活用を進めていきたい．

参 考 文 献

1) Gagnon, E.: SableCC, An Object-Oriented Compiler Framework, Master's thesis, McGill University (1998). http://www.sablecc.org/.
2) Goldberg, A. and Robson, D.: *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley (1983).
3) Gosling, J., Joy, B. and Steele, G.: *The Java Language Specification*, Addison-Wesley (1996).
4) Hirano, S.: HORB: Distributed Execution of Java Programs, *Worldwide Computing and Its Applications*, Lecture Note in Computer Science, No. 1274, Springer-Verlag, pp. 29–42 (1997).
5) Ingalls, D., Kaehler, T., Maloney, J., Wallace, S. and Kay, A.: Back to the future: the story of Squeak, a practical Smalltalk written in itself, *Proceedings of the 1997 ACM SIGPLAN conference on Object-oriented programming systems, languages and applications*, pp. 318–326 (1997).
6) Object Management Group: CORBA/IIOP 3.0 Specification (2002). http://www.omg.org/technology/documents/formal/corba_iiop.htm.
7) Papert, S.: *Mindstorms : children, computers, and powerful ideas*, Basic Books (1980).
8) Steinmetz, J.: Computers and Squeak as Environments for Learning, *Squeak: Open Personal Computing and Multimedia*, Prentice Hall (2001).
9) 兼宗進, 御手洗理英, 中谷多哉子, 福井眞吾, 久野靖: 学校教育用オブジェクト指向言語「ドリトル」の設計と実装, 情報処理学会論文誌, Vol. 42, No. SIG11(PRO12), pp. 78–90 (2001).
10) 兼宗進, 中谷多哉子, 御手洗理英, 福井眞吾, 久野靖: 初中等教育におけるオブジェクト指向プログラミングの実践と評価, 情報処理学会論文誌 (印刷中).
11) 関将俊: dRuby による分散オブジェクトプログラミング, アスキー (2001).
12) 総務省: 平成 14 年通信利用動向調査の結

果. http://www.soumu.go.jp/s-news/2003/030307_1.html.
13) 中谷多哉子, 兼宗進, 御手洗理英, 福井眞吾, 久野靖: オブジェクトストーム: オブジェクト指向言語による初中等プログラミング教育の提案, 情報処理学会論文誌, Vol. 43, No. 6, pp. 1610–1624 (2002).
14) まつもとゆきひろ, 石塚圭樹: オブジェクト指向スクリプト言語 Ruby, アスキー (1999).