プログラミング言語処理系 Squeak の SHARP Zaurus への移植とその評価

大島芳樹[†] 脇田建[†] 佐々政孝[†]

高機能な Smalltalk 処理系である Squeak を携帯情報端末 Zaurus に移植し，Zaurus の持つ独自のハードウェアにあわせて拡張を施したので，システムの実現方法，性能，可用性の評価について報告する．我々の行った実装作業は大きく 2 つに分類される．まず，基本的な機能の性能を向上させるために，Zaurus 上でのイベント処理方式やビットマップ変換方式の性能について定量的評価を行い，それに基づいて最適化を行った．さらに，Zaurus 独自の拡張を施すことによって，ペン入力，音声機能，デジタルカメラ，TCP/IP ネットワーク，シリアル・IrDA ポート，電力消費制御，およびフラッシュメモリファイルシステムのような PDA 独自のハードウェアを Squeak から操作可能とした．この移植および拡張は Zaurus 依存の部分を記述した 4000 行程度の C 記述および 600 行ほどの Squeak 記述によって達成された．我々は大小各種のベンチマークによる性能評価およびいくつかのアプリケーションによる可用性評価を行った．Zaurus シリーズの最上位機種である MI-EX1 上の Squeak は，400 MHz G3 Mac に対して最高 1/8 程度の性能を示した．このベンチマークの数字および実際の使用テストから，このシステムは十分実用に耐える性能を持つことが分かった．我々の実装によって，さまざまな周辺機器を操作できる実用的なプログラミング環境を手のひらサイズの機器上で運用できることが示された．

A Report on Porting the Programming Environment Squeak to SHARP Zaurus and Its Evaluation

Yoshiki Ohshima,[†] Ken Wakita[†] and Masataka Sassa[†]

This paper reports our effort to implement Squeak Smalltalk on a family of personal digital assistant (*PDA*) devices called Zaurus. The Squeak on Zaurus (*Squeak/Zaurus*) offers programmable interfaces to various Zaurus specific devices. The system is evaluated in terms of its usability and performance. Two important issues in implementation of Squeak/Zaurus are as follows. Firstly, we measured performance of several implementation strategies for event-handling and bitmap conversion, which are dominant performance issues, and adopted the best ones in Squeak/Zaurus. Secondly, we have extended Squeak to support PDA specific hardware such as pen input, audio, digital camera, TCP/IP network, serial and IrDA communication, power consumption control, and file I/O on flash memory. These PDA specific devices are made directly accessible to the programmer. Those Zaurus specific functions are written in 4,000 lines of C code and 600 lines of Squeak code. The performance of Squeak/Zaurus has been evaluated by various micro- and macro-benchmarks and its usability has been tested by applications. MI-EX1, the highest model of Zaurus family, marked 1/8 of the performance of Squeak running on 400 MHz G3 Mac at best. This result and the usability tests indicate that it performs sufficiently well for practical use. The Squeak/Zaurus implementation demonstrates feasibility of practical programming environment on palm size devices that can operate PDA specific hardware.

1. はじめに

近年は小型で日常的に持ち運ぶ電子機器（**携帯情報端末**または **PDA** *Personal Digital Assistant*）が目覚ましく発展している．将来にわたって PDA 上で動作するプログラムへの需要が高まっていくにつれて，PDA 用プログラム開発の効率化がますます重要になっていくものと思われる．

この問題に対処するためには，PDA 上に言語処理系を実装し，他の環境で動作するプログラムをでき

† 東京工業大学大学院数理・計算科学専攻
Department of Mathematical and Computing Sciences, Tokyo Institute of Technology

Zaurus, Palm, Windows CE, Java, REBOL など，文中の商品名は各社の登録商標です．

るだけ変更することなく動作させる，という方法が提案され，このような方向性のもとにすでにいくつかのPDA 用言語処理系が作成されている[4),6),12)]．

上記のような言語処理系のいくつかは，PDA 上でプログラム（断片）を「書ける☆」機能を持っている．ある程度のハードウェア性能を持つ PDA であれば，デスクトップ計算機などで作成したプログラムをダウンロードして実行するだけにとどまらず，デスクトップ機の補助がない環境でもプログラムの修正や記述を許すことによって，さらなるプログラム開発の効率化が期待できる．

このような言語処理系のさらなる応用として，PDA を単なるプログラム実行のプラットフォームとして見るのではなく，個人的な対話型メディア，すなわち Kay が **Dynabook**[5)] と呼んだような**動的メディア**（*dynamic media*）として扱ってしまうことが考えられる．すなわち，PDA 上で動作しているシステムのできる限り多くの部分を対話的な言語処理系の上で動作させることによって，PDA 上のシステム全体が 1 つの拡張言語で記述されているかのような状態にしてしまうのである．このように固定的な部分を極力減らすことによって，それぞれのユーザが PDA を自分自身の手に馴染むように拡張・変更できるようになる．

たとえば，PDA 上に格納されているデータを検索するような際に，固定的に与えられた検索ルールでは不十分な場合がしばしばある．このような場合には，ユーザが必要に応じて検索ルールを定義できるようになっていると便利である．

さらに，PDA にはタッチペンやデジタルカメラ，マイク，無線ネットワークなど PDA 特有の周辺機器を持つものが多い．それらの周辺機器を言語から操作できるようにすれば，ユーザが任意の時点でそれらを操作するプログラムを書き，得られたデータを自由に組み合わせて操作することもできるようになる．このように，PDA 上に対話的なシステムを構築することによって，ユーザが PDA を使用するときの自由度を大きく高めることができる．

ここで，このような動的メディアを構築するための道具にどのような特性が求められるか考察してみよう．

PDA とデスクトップで使われるような計算機とを比較すると，一般に PDA では CPU 性能やメモリ搭載量に制限がある，OS の機能が限られている，入出力装置が貧弱であるなどの制限がある．一方では，デスクトップ機には見られない特殊な周辺機器が使用可能であるといった特長もある．

このようなハードウェア上で動作する言語処理系に要求される特性としては，(1) ランタイムシステムが小さいこと，(2) 用途に特化したシステムを柔軟に構築できること，(3) 他の環境向けに書かれたプログラムの流用が容易であること，(4) GUI 指向であり，複雑なキー操作を仮定しないこと，(5) 複雑なファイルシステムを必要としないこと，などがあげられる．候補となる既存の処理系はいくつかあげられるが，なかでも，Smalltalk の一実装である Squeak[4)] は，もともとこのような用途も想定して設計されていることもあって上記の条件をよく満たす処理系である．

一方，そのような言語を動かすための PDA には，可能な限り高性能であり，かつその上で動作するユーザプログラムを自由に作成できることが求められる．実験用の PDA としては，Itsy[14)] などのように自由度が高く高性能なものが存在しており，また市販されている製品としては，Casio E-500 など Windows CE を搭載したいくつかの機種が高性能なものとして知られている．

そのような PDA が存在する一方，シャープの製品である Zaurus のラインナップには，高性能な CPU を搭載し，640 × 480 サイズの液晶や反射型液晶を持つものなどユニークな製品が存在する．さらに，デジタルカメラや IrDA デバイス，PPP を用いたネットワーク機能などを搭載しており，それらを使うためのAPI も公開されている．これらのことを考慮すると，Zaurus はプログラムを記述するためのプラットフォームとしてはかなり魅力的であるといえる．

筆者は，上記のような考察から，現時点で最も可用性が高いと思われる Squeak と Zaurus との組合せによって，高性能な動的メディア実現の向けての評価を行った（**図 1**）．以下，このシステムを Squeak/Zaurus と呼ぶ．このシステムは単なる Squeak の移植のみで

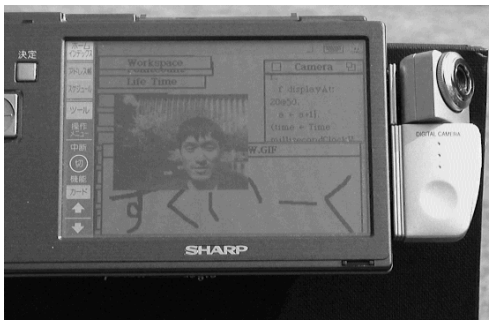

図 1 MI-C1 上の Squeak
Fig. 1 Squeak on MI-C1.

☆ 必ずしも文字列としてのプログラムを意味しない．

はなく，Zaurus 固有のハードウェアに対応するための拡張を行ったものである．

本論文では Squeak/Zaurus の性能と可用性を評価する．筆者による移植版は MI-シリーズと呼ばれる Zaurus のほとんどで動作するが，本論文では，最も実用性の高い 2 機種である MI-EX1 と MI-C1 にしぼって評価を行う．Squeak/Zaurus のソースコードは 8) から入手可能である．

以下の各章は次のように構成されている．2 章では Squeak の特長について紹介する．3 章では Zaurus の特徴について説明する．4 章では，Squeak/Zaurus の機能について説明する．5 章では，Squeak/Zaurus 仮想機械の高速化について述べる．6 章では，Smalltalk の伝統的なベンチマークである Green Book Benchmark の実行結果について述べる．7 章では，関連するシステムや研究について述べる．付録には Zaurus のメモリシステム性能の測定方法と，移植に至る経緯およびその後行った作業を紹介する．

2. Squeak

Squeak とは，Alan Kay や Dan Ingalls など，Smalltalk-80 の開発者たちによる Smalltalk の新しい実装である．音声処理や 2D，3D グラフィックス，ネットワーク，Morphic と呼ばれる GUI フレームワーク[10] などの機能を持ち，OOPSLA '99 では招待講演として Squeak のデモが行われるなど，近年注目を集めている実用的システムである．

Squeak の実行形態は，仮想機械の上で動作する仮想イメージという Smalltalk-80 の伝統的なものに則っている．仮想イメージはほぼ完全に機種独立であるため，新しいプラットフォームへの移植は仮想機械を移植することのみとなる．

Squeak 仮想機械自身も非常に高い移植性を持った形で記述されている．移植の際にプラットフォームが満たすべき条件は，32 ビット整数演算を持つということと，連続したヒープ領域が確保できる，ということのみである．Squeak 仮想機械は Squeak 自身で記述されており，Squeak で記述された Squeak から C への変換系を経て実行可能形式となる．

Squeak の実行を補助するために，**プリミティブ**（*primitive*）と呼ばれる機械語で書かれたルーチンが仮想機械内に存在する．プリミティブには，プラットフォーム非依存のものとプラットフォーム依存のものがあるが，Squeak の場合，前者は仮想機械と同様に Squeak 自身で記述されている．一方，後者は移植者が C 言語で記述した**サポートコード**（*support code*）と呼ばれる関数を呼び出す形で実装される．Squeak を対話的なシステムとして動作させるためは，ユーザからの入力を扱うイベントハンドラと，Squeak の持つビットマップの画面への表示を受け持つサポートコードが最低限必要となるが，これらは 200 行程度の C コードで記述することが可能である．このようにして作成された仮想機械の大きさは，条件にもよるが 100 kB から 400 kB 程度となる．

Squeak は比較的小さなメモリ使用量で動作する．Squeak の最新版には，仮想機械自身のソースや 3D アニメーション・オーサリングツール，音声合成，ベクトル・グラフィックス・ライブラリ，Web server，Morphic フレームワークとそのさまざまな GUI 部品などが含まれており，これはクラス数にして 1160 個程度，あるいはメソッド数にして 25000 個程度の大きさであるが，このイメージはたかだか 8 MB 程度のヒープ使用量で起動することができる．

さらにメモリ制限が厳しい環境で Squeak を実行する場合には，上記の仮想イメージから自分のアプリケーションに必要のないクラスやメソッドなどを除去することによって必要とするメモリ量を減らすことができる．Morphic フレームワークは比較的大きなものであるため，区別のために **MVC フレームワーク**と呼ばれている伝統的な Smalltalk の GUI フレームワークのみを使用するようにすれば，Morphic フレームワークを除去してしまうこともできる．

Squeak のためのデータ領域として 800 kB 程度が確保できれば，MVC フレームワークを用いた小さなアプリケーションを動作させることができ，4 MB 程度が確保できれば，330 個程度のクラスを含むフルセットの MVC フレームワークを含むイメージ，言い換えれば Morphic フレームワークのみを除去したイメージ上で大きなアプリケーションを動作させることができる．ちなみに，Morphic フレームワークを用いたアプリケーションであっても，そのアプリケーションに不必要なクラスを除いたイメージを作成すれば，4 MB でも動作させることは十分可能である．

一般には Morphic フレームワークの方が動的な特性や拡張性の面で優れているといえるので，できればそちらを使用したいところであるが，PDA 用程度の大きさを対象とした場合には MVC フレームワークでもほぼ十分な記述力を持っている．また，後述するような新たな GUI フレームワークを構築することを考えると MVC フレームワークの上で行ったほうが容易なので，Morphic フレームワークを除去してしまっても PDA 用プログラムを作成するうえでは大きな障害

表 1 Zaurus MI シリーズのハードウェア仕様
Table 1 Hardware specification of Zaurus MI-series.

モデル名	DRAM	ユーザ領域	CPU	キャッシュサイズ	液晶
MI-10	1 MB	約 400 kB	SH-3 45 MHz	—	320×240 カラー
MI-506, MT-200 など	2 MB	約 800 kB	ほとんどのものは SH-3 60 MHz	—	320×240 カラーまたは白黒
MI-EX1	8 MB	約 5 MB	SH-3 7709A 120 MHz（クロックは推定）	16 kB	640×480 カラー
MI-C1			SH-3 60 MHz（推定）	8 kB（推定）	320×240 反射型カラー

とはならない．

3. Zaurus

Zaurus とはシャープが発売している一連の PDA に付けられた名称である．Palm などの他の PDA と比較すると，高価格ながら高機能なハードウェアを方向性としたラインナップになっている．最近では Zaurus 用のユーザアプリケーション（**MORE ソフト**）を書くための **SZAB** と呼ばれる開発環境が提供されているため，ユーザが自由にアプリケーションを作成できるようになっている．また，周辺機器であるデジタルカメラやマイク，IrDA デバイス，PPP を用いたネットワーク機能などを扱うプログラムも作成できる．これらの点から，Zaurus は研究者がプログラムを動かすための PDA としてかなり有望なものといえる．

以下では，Zaurus のハードウェアとソフトウェアについて述べる．

3.1 Zaurus のハードウェア

Zaurus と呼ばれる製品はバイナリ互換性に基づく分類によって大きく 2 つの世代に分けることができる．1 つ目は 1993 年に発売開始された「PI-シリーズ」と呼ばれるものであり，2 つ目は 1996 年に発売になった「MI-シリーズ」と呼ばれるものである．

MI-シリーズは，DRAM の搭載量に応じてさらに 3 つに分類することができる．**表 1** に MI-シリーズのハードウェア仕様を示す．ただし，MI-10 はすでに市場から姿を消している．それぞれ OS がある程度の DRAM 領域を使用しているため，ユーザアプリケーションがデータ領域として使用できるメモリ（表中ユーザ領域となっているもの）は，搭載されている DRAM 容量よりは小さくなる．

筆者による Squeak/Zaurus は，メモリ制限の厳しい MI-10 以外の機種で動作する．ただし，前章で述べたようにデータ領域が 800 kB しか確保できない第 2 分類の Zaurus では簡単なアプリケーションを動作させるのがやっとであり，Squeak を実用的に動作させるためには，最低 4 MB 程度のユーザメモリ領域が確保できることが望ましい．そのため，実用的に Squeak を実行できるのは，2000 年 2 月現在で，ICRUISE と呼ばれる **MI-EX1** と **MI-C1** の 2 機種のみである．

以下では，この 2 機種に絞って説明を行う．

3.1.1 MI-EX1

MI-EX1（商品名 ICRUISE）は表 1 にあげたような仕様を持つ．MI-EX1 の最も大きな特徴としては 4 インチの VGA 液晶を搭載していることである．

MI-EX1 に搭載されている CPU の型番は公開されていないが，SH シリーズの仕様[16]，筆者による性能測定やそのほかいくつかの情報☆から，Hitachi SH-3 7709A 120 MHz であると推定されている．

3.1.2 MI-C1

MI-C1 は表 1 にあげたような仕様を持つ．陽光の下でもはっきりと画面を視認できる反射型液晶を搭載していることが大きな特徴である．PDA 上の運用範囲として屋外使用の可能性を無視することはできないため，屋外での視認性は実用上非常に重要である．

MI-C1 のもう 1 つの特長は，音声再生・録音機能を持つことである．モノラル 8bit サンプル，周波数 24 kHz 以下と音質はそれほど高くはないものの，PDA が持つ機能としてサウンド機能は非常に重要である．

MI-C1 に搭載されている CPU も公表されていない．筆者や小笠原博之氏による CPU 速度の測定によって，60 MHz のクロックで動作しているものと推定されているが，具体的な型番は不明である．

3.2 Zaurus のソフトウェア

Zaurus 用のユーザアプリケーションは **MORE ソフト**と呼ばれる．SZAB と呼ばれる MORE ソフト開発環境が市販されており，誰でもアプリケーションを作成することができる[15]．

開発環境には，Windows 上で動作する SH-3 のクロスコンパイラと GUI 部品を配置するためのエディタ

☆ 富樫吉徳氏による MI-EX1 の分解写真．ざうまが Web（http://free01.plala.or.jp/~mobile/zaumaga/）
小笠原博之氏による Zaurus Speed Test（http://hp.vector.co.jp/authors/VA004474/zaurus/zst.html）

が付属している．開発者はアプリケーションを C で記述し，クロスコンパイラを用いて Zaurus（SH-3）用のバイナリを生成する．その後，生成されたバイナリをシリアルケーブル経由で Zaurus にダウンロードし実行する．デバッグも Windows 上で動作するリモートデバッガを使い，シリアルケーブル経由で実行することができる．

Zaurus の OS は **ZaurusOS** と呼ばれている．ZaurusOS はメモリ保護機構やカーネルスレッドなど高度な機構を提供しながらも，1 MB から 2 MB 程度と思われる比較的小さなメモリ使用量で動作している．MI-EX1 および MI-C1 上で MORE ソフトを実行する際には，ZaurusOS はアプリケーションのコード領域としてユーザプログラムからは書き換え不可能な 512 kB の DRAM を固定的に割り当てる．

4. Squeak/Zaurus の実現

Squeak/Zaurus は MORE ソフトとして Zaurus 上で動作する，筆者による Squeak の移植版である．以下では，我々が移植・拡張した Squeak/Zaurus の機能および応用例について述べる．Squeak/Zaurus の機能には，デスクトップなど他の環境で動作している Squeak と同等の機能を提供している部分と，Zaurus が持つ周辺機器への対応など Squeak/Zaurus 独自の機能とが含まれている．以下ではそれらについて説明を行う．

4.1 他の環境との互換性

Squeak プログラムは，どんなプラットフォーム上で実行しても，1 ドットの違いもない形で同一の振舞いをすることが期待されている（*dot-identical behavior*）．Squeak の哲学であるこのような高移植性を維持するために，Zaurus 版の仮想機械を作る際には，ドット同一の振舞いを維持し，デスクトップ機などの環境を仮定して書かれたプログラムを極力変更せずに動作させることを目標の 1 つとした．

Squeak のユーザインタフェースは，ビットマップディスプレイと 3 ボタンポインティングデバイス（マウス），文字入力用のキーボードがあることを仮定している．そのため，Zaurus への移植の際には，これらを Zaurus のハードウェアに対応付ける必要があった．

Zaurus はタッチペンと OS 組み込みのソフトウェアキーボードを持っているので，タッチペンをポインティングデバイスとして用い，ペンによる画面タッチをクリックとして扱うことにした．3 ボタンへの対応は，Zaurus が持つ「機能」と呼ばれる前置キーが押された後での画面タッチを中ボタンとし，「メニュー」キーが押された後での画面タッチを右ボタンとして扱うようにした．

キーボードに関しては，OS 組み込みのソフトウェアキーボードあるいは外部接続用のキーボードからの入力を Squeak から使用できるようにした．また，ビットマップディスプレイは Zaurus の液晶画面をそのまま使うことができる．

これらの対応付けによって基本的にはほとんどのプログラムが問題なく動作した．

問題となるのは，デスクトップ機でマウスのボタンを押さずに画面上を動かす “mouseOver” と呼ばれる動作への対応である．Zaurus のようにペンをポインティングデバイスとして使う場合には，画面に触れずには，すなわちマウスボタンを押していない状態では，マウスカーソルを移動することができなくなってしまう．しかし，MVC フレームワークで書かれた既存のアプリケーションでは，mouseOver を仮定して書かれたコードは数カ所のみであったので，mouseOver をクリック後にドラッグする操作に書き換えることによって対処した．一方，Morphic フレームワークは多くの部分が mouseOver に頼っているため，ペンベースの操作体系ではかなり操作性が落ちることが判明した．

対案としては，特定のハードウェアキーにマウスボタンの on/off を切り替える機能を割り当て，画面タッチをクリックと見なさない操作モードを作る，というものが考えられる．しかし，このようなモードの存在はユーザが把握しておくべき状態を増やすことになり，操作が複雑になるという問題がある．また，仮想機械に固定的に組み込まれた機能は極力少ないほうがよいという哲学にも反することになる．このようなことは望ましくないと考え，上記の方法にした．

ただし，PDA 用の現実的なアプリケーションを作る場合には，ペンベースの操作であることを仮定した設計にすべきである．将来はそのようなアプリケーションの構築を行うためのペン指向 GUI フレームワークの構築が必須である．

4.2 Squeak/Zaurus に特有な機能

さて，ここまで述べてきたようなペンと画面表示とソフトウェアキーボードという基本的なものだけではなく，Zaurus には PDA ならではの特徴を持った周辺機器がいくつか用意されている．これらの機器を抽象化した Squeak のオブジェクトを定義することによって，対話的なユーザプログラムからそれらの機器に簡単にアクセスできるようになる．

以下では，そのような周辺機器の機能と，それぞれに対して行った性能評価の結果，およびそれぞれの機

表 2 デジタルカメラからの Form 取得速度（単位：こま数/秒）
Table 2 Frame rate of converting the image from camera to Form (frame/sec).

ハードウェア	画像サイズ	結果
MI-EX1	80 × 60	4.94
	160 × 120	4.93
	320 × 240	3.28
MI-C1	80 × 60	5.75
	160 × 120	13.75
	320 × 240	5.78

能の実現に要したサポートコードの量について述べる．

4.2.1 デジタルカメラ

Zaurus の最も特徴的な周辺機器としてはデジタルカメラがあげられる．MI-EX1 には 85 万画素の，MI-C1 には 35 万画素のカメラが接続可能であり，それらを制御するための API も ZaurusOS に用意されている．

筆者は，ZaurusOS の API に対応するいくつかのプリミティブと，それらを呼び出すための DigitalCamera という Squeak のクラスを定義した．これによって，ユーザプログラムからの指示があった時点で，デジタルカメラの CCD に写っている画像を Squeak の画像オブジェクトである Form オブジェクトに変換できるようになった．Squeak/Zaurus の DigitalCamera では，大きさ 80 × 60，160 × 120 および 320 × 240 の深度 16 ビットの画像を取得することができる．条件によっては大きさ 640 × 480 のものも取得可能であるが，通常は画像を格納するための一時メモリの確保に失敗してしまうことがある．

表 2 に，デジタルカメラから 1 秒あたりに取り出せる Form の枚数を測定した結果を示す．おそらくは特殊な最適化が行われている MI-C1 での大きさ 160 × 120 のものでは 14 枚弱，その他でもおおむね毎秒 4 から 6 枚程度というように，Squeak/Zaurus ではある程度連続的にカメラから画像を取得できる．Zaurus のデジタルカメラは本来静止画カメラであるにもかかわらず，この機能によって擬似的な動画カメラとしても使用できるようになった．

デジタルカメラ機能を実現するために必要となったサポートコードは 300 行程度であり，DigitalCamera クラスの行数は 50 行程度である．

4.2.2 ネットワーク

ZaurusOS は，内蔵モデムや携帯電話・PHS を用いた PPP 接続と，その上で動作する TCP/IP ソケットを扱う API を用意している．筆者によるサポートコードの記述によって，Squeak が持つ Socket クラスを使用したプログラムを作成することができるようになった．

本論文では詳述しないが，ネットワークのサポートコードの記述はやや困難であった．これは，このようなプログラムでは，OS が提供する API を呼び出してもインタプリタがブロック（停止）しないように記述する必要があるため，ネットワーク機能を非同期的に使用する必要がある．しかし，ZaurusOS が提供しているネットワーク API は，おおむね BSD socket に準じているものの特に非同期的な通信に関しては独自性の強い仕様を持ったものになっているうえに，ドキュメントもかなり不十分なものであったためである．

Squeak/Zaurus の Socket を使用して通信した場合のネットワークバンド幅性能はかなり低く，32 kbps の PHS を使用した場合約 1 kB/秒程度と理論値の 1/3 程度になってしまっている．この部分にはさらなる高速化が必要である．

TCP/IP ネットワークのためのサポートコードは 1100 行程度である．

4.2.3 音声録音・再生

3.1.2 項で述べたように，MI-C1 では音声の録音および再生を行うハードウェアを持っている．筆者らの記述したサポートコードとそれを呼び出すための Squeak コードによって，Squeak プログラムで音声の録音・再生を制御できるようになった．Squeak/Zaurus の録音・再生機能は，Squeak 内で動いている他の Squeak プロセスと並行に動作させることができる．この機能を用いて画面上でアニメーションを行いながらそれに合わせて音声を出力するようなアプリケーションも簡単に構築できる．

ただし，ZaurusOS が提供している音声 API はやや貧弱であり，Squeak が仮定しているだけの機能を提供することができない．そのため，筆者らが実装した音声機能は，もともと Squeak が持っている音声クラスライブラリとは非互換なものとなっている．

音質は 3.1.2 項で述べたようにあまり高くはないが，録音時間に関してはフラッシュメモリ上のファイル容量が許す限りの時間にわたって（数十分から数時間程度）録音・再生を行うことができる．

音声録音・再生機能のためのサポートコードは 400 行程度であり，Squeak コードは 150 行程度である．

4.2.4 シリアル・IrDA ポート

Zaurus は，115200 bps までをサポートするシリアルポートと IrDA ポートを持っており，ZaurusOS はそれらを制御するための API を提供している．筆者の記述したサポートコードによって，シリアルポートおよび IrCOM ポートと見なした IrDA ポートを Squeak の持つ SerialPort として操作できるようになった．

これらのポートを通じたデータ転送を Squeak から行った場合には，ほぼハードウェア仕様どおりの性能を引き出すことができる．ただし，ハードウェアの通信速度以上のペースで Squeak コードからデータを送信すると，OS の持つバッファがあふれてしまう．この現象は，Squeak にはプリミティブの失敗（*primitive failure*）として報告されるので，一般に Squeak/Zaurus でシリアルポートを使用したプログラムを作成する場合には，Squeak 側でプリミティブの失敗に対処しながらデータを送受信する，というスタイルで記述する必要がある．

シリアルポートと IrDA ポートのためのサポートコードは 250 行程度である．

4.2.5 ファイルシステム

Zaurus はフラッシュメモリを用いた 2 次記憶装置を搭載している．ZaurusOS は階層ディレクトリこそ持たないものの，内蔵フラッシュメモリおよび PC カードスロットに挿入したカード上にバイトストリームとしてのファイルを作成し，UNIX のシステムコールにほぼ対応したファイル操作 API を用いて操作することができる．

筆者によるファイル操作 API を呼び出すサポートコードによって，Zaurus のファイルを Squeak の FileStream として扱うことができるようになった．

ZaurusOS では，内蔵のフラッシュメモリを “F0:” ドライブ，外部カードに挿入されたフラッシュメモリを “F1:” ドライブとして参照することができる．Squeak/Zaurus ではそれら 2 つを擬似的にディレクトリと見なし，FileDirectory オブジェクトとして操作できるようになっている．

ファイルシステムの性能を評価するために，C および Squeak プログラムによって MB 単位のファイルを直線的に読み書きした場合の性能を**表 3** に示す．使用した外部カードは SunDisk 社の 48 MB コンパクトフラッシュである．測定の結果，読み出しの性能と書き込みの性能は 1%内外の違いしかなかったため，表には両者を平均した値が示されている．表中 Squeak と書かれている項は，Squeak の FileStream を用いて同等の操作を行ったものでありる．C で書いたものに加えてバッファのためのメモリ割当てとプリミティブ呼び出しが必要となっているが，そのオーバヘッドは最大でも 30%程度におさえることができた．

表 3 ファイル操作のバンド幅
Table 3 File operations bandwidth.

ハードウェア	対象	バンド幅（MB/秒）
MI-EX1	内蔵	1.70（C）
	外部	0.51（C）
	内蔵	1.32（Squeak）
	外部	0.46（Squeak）
MI-C1	内蔵	0.50（C）
	外部	0.44（C）
	内蔵	0.40（Squeak）
	外部	0.40（Squeak）

ファイルシステムのためのサポートコードは 250 行程度である．

4.3 電　　源

Zaurus 本体が持つ省電力機能として，電源が入った状態でユーザが操作をせずに放置したときに，設定された時間（2〜20 分）が経過すると自動的に電源が切れるもの（**自動電源断機能**）がある．しかし，Squeak アプリケーションの中には，Web サーバのようにユーザとは対話を行わずに長時間継続して実行するべきものがある．このようなアプリケーションの存在を考えると，Squeak から自動電源断機能を操作できるようになっていることが望ましい．

そこで，筆者は Squeak/Zaurus で自動電源断機能を制御するためのいくつかの機構を記述した．現在の実装では，Squeak から自動電源断機能の有効化・無効化を変更することができる．ただし，電池が完全に消耗してしまうことは望ましくないので，電池残量が残り少なくなった場合には，自動電源断機能を無効化していても自動的に電源が切れるようになっている．

Squeak を実行している場合の電池持続時間に関しては，電池消費量は実行する内容にも大きく依存するため厳密な計測は容易ではない．特に MI-EX1 のようにバックライトを持つ機種の場合には，バックライトの使用状況によって大きく変化する．そのため，現在のところ筆者による電池持続時間の計測は初歩的なものにとどまっているが，MI-EX1 の場合は 3 時間程度，MI-C1 の場合には 7 時間程度の連続実行が可能であると思われる．

電源関係のサポートコードは 10 行程度であり，それを呼び出すための Squeak 側のコードは 20 行程度である．

4.4 名前呼びプリミティブへの対応

Squeak は伝統的な番号によるプリミティブ呼び出しだけではなく，**名前呼びプリミティブ**（*named primitive*）と呼ばれる,（文字列の）名前が付けられたプリミティブを呼び出す機能を持っている．ただし，その実装では OS が持つ動的リンク機構（*dynamic linking* あるいは *dynamic loading*）を仮定しているため，それを持たない Zaurus では使用できなかった．

これを解決するため，筆者は名前呼びプリミティブ

を静的に仮想機械にリンクしておき，実行時にそれを呼び出せるようにする機構を実装した．

現在は，Squeak 仮想機械を拡張する方法として名前呼びプリミティブが使用されることが多くなっているので，Squeak/Zaurus からそれらの機能を呼べるようにすることは，将来における仮想機械の互換性を極力維持するために非常に重要である．

名前呼びプリミティブへの対応は，340 行程度の Squeak コードである．

4.5 本章のまとめ

Squeak/Zaurus によって，ユーザがデスクトップ機上で作成したアプリケーションは 1 ドットの違いもなく Zaurus 上で再現され，それを Zaurus 上で修正することもできるようになった．その結果，効率良くアプリケーション開発ができ，PDA を持っていった先で，その場に応じてアプリケーションを修正するようなこともできるようになった．

ネットワークやシリアルポートなどはデスクトップ機とまったく同一の Squeak プログラムであり，DigitalCamera クラスも単純なものに置き換えればよいので，Squeak/Zaurus を仮定したプログラムをデスクトップ機上でデバッグすることもできる．

また，Squeak の特長からすべてのデータは Squeak のオブジェクトとして操作できるので，簡単な Squeak プログラムによって，PDA 上にあるデータをユーザが自由に組み合わせて操作できる環境として使うこともできる．

各機能のサポートコードと，ペン操作や画面表示，ソフトウェアキーボードなどの基本的な機能を実現するためのサポートコードを合計すると，およそ 3600 行程度であった．

仮想機械のバイナリの大きさは現在のところ 360 kB 程度である．このうち，ZaurusOS のネットワークライブラリが 110 kB 程度，Squeak で記述され C に変換されているベクターグラフィックス機能が 80 kB 程度を占めており，これらをリンクしなければ 170 kB 程度，さらに仮想機械ソースプログラムのインライン化をしなければ 120 kB 程度まで小さくすることができる．

5. 仮想機械の性能向上

本章では，Squeak/Zaurus 仮想機械に筆者が施した性能向上のための技術について説明する．

2 章で述べたように，Squeak 仮想機械の中は，Squeak で記述された機種独立な仮想機械本体と，サポートコードと呼ばれるハードウェアや OS との界面を記述した部分とに分割される．性能を向上させるために仮想機械本体にも修正を施すことが考えられるが，Squeak 仮想機械はバイトコードインタプリタとしてはすでにかなり注意深く高速化されていること，かといって ZaurusOS がヒープやスタックから命令をフェッチすることを許していないため動的コンパイルは行えないこと，また何よりも仮想イメージの高い移植性を維持することが重要であることから，仮想機械内部は修正せず，サポートコードで達成できることのみを行った．

サポートコードの中には，頻繁に実行され，仮想機械全体の性能に影響を及ぼすものがある．このようなものとしては，Display オブジェクト（画面そのものを抽象化したオブジェクト）を実際の画面に表示するものと，ユーザからの入力を受け付け，仮想機械本体にイベントとして渡すためのものがある．

以下では，上記 2 つのサポートコードの性能向上と，バイトコード分岐の範囲チェックを除去することによって得られた性能向上について説明する．ただしその前に，一般にオブジェクト指向言語処理系の性能に大きな影響を与えるといわれているメモリシステムの性能について，Zaurus 上で計測した結果を述べる．筆者が行ったメモリシステム性能の具体的な測定方法を付録 A.1 に載せる．

Zaurus および ZaurusOS の仕様に関しては多くの部分が非公開になっており，シャープからの支援も非公式なもので，かつハードウェアの仕様に関してはあまり多くの情報を入手することはできなかった．そのため，以下の議論はかなりの部分が推測に基づくものであることをお断わりしておく．

5.1 Zaurus のメモリシステム性能

一般に，プログラムの実行性能には，メモリシステムの性能が大きくかかわっているといわれている．特に，後述するような画面表示処理のように，大きなメモリ領域間でデータをコピーするような場合にはそれが顕著である．そのため，まず Zaurus のメモリシステム性能を計測した結果について述べる．Squeak/Zaurus では画面書き換えのために非公開 API を用いて VRAM への直接アクセスを行っている．そのため，以下では通常のメモリと VRAM とを区別して測定を行った．

表 4 は，MI-EX1 および MI-C1 におけるメモリ転送性能の筆者による測定結果である．MI-EX1 と MI-C1 では，CPU クロックは 2 倍の開きがあると推定されているが，どちらも通常のメモリに対するアクセスではほとんど差が見られない．これは，このテストによってメモリ転送性能の上限に達してしまっているた

表 4 メモリ転送ベンチマークの結果（単位：MB/秒）
Table 4 Memory transfer benchmark results (in MB/sec).

write	MI-EX1	5.81
	MI-C1	5.69
read	MI-EX1	10.06
	MI-C1	10.04
vram-write	MI-EX1	7.90
	MI-C1	5.80
vram-read	MI-EX1	6.87
	MI-C1	5.78

めであり，通常のメモリに対するバンド幅は，MI-EX1 でも MI-C1 でもほぼ同じであるからと考えられる．

5.2 画面表示の性能向上

以上のような性能評価を基に，画面表示の性能向上を図った．まず，Squeak が行っている画面表示の方式について説明する．

Squeak では，高い移植性を保つ必要性と，「計算機内部のすべてがオブジェクトとして操作できる」という抽象化をユーザに与えるため，画面そのものを表したオブジェクトをメモリ内に用意している．このようなオブジェクトは Display と呼ばれ，Squeak 内で他の Form オブジェクトと同様に操作できる．Squeak では深度 1，2，4，8，16，32 の Form を操作することができ，Display もこれらの深度の Form として扱うことができる．

Squeak における画面への表示処理は，本質的にはメモリ上の画像を表現した配列を別のメモリ領域へ転送することである．ZaurusOS は VRAM のアドレスを取得する非標準 API を提供しており，Squeak/Zaurus はこの API を用いて VRAM への直接書き込みを行っているので，一般に画面書き換えは表示すべきデータを Display へ書き込み，次に Display から VRAM へデータを転送する，という操作となる．

Display のようなオブジェクトがユーザから見える形になっていない言語でも，ダブルバッファリングを行って画面表示をする場合には同様の処理が必要となるので，以下の議論はそのような言語にも適用できる．

Squeak において，Display を含めた Form とは，いくつかの付加的な情報を除けば，連続したメモリ領域に格納された画素の配列である．一方 Zaurus のカラー液晶画面は深度 16 ビットのものであり，16 ビット画素の配列となっている VRAM の内容が表示される．

Display オブジェクトを Zaurus の液晶画面に表示するために，Squeak 仮想機械は `ioShowDisplay()` というサポートコードを呼ぶ．`ioShowDisplay()` では，機種独立な表現になっている Display を機種依存の表現に変換して画面に表示する．上記のように，Squeak の Display はいくつかの深度をとりうるが，以下では，Display の深度が 16 ビットのときに話を絞って説明する．

表 5 画素変換方式の比較（単位：ミリ秒）
Table 5 Comparison of pixel conversion scheme (in milliseconds).

ベンチマーク	変換方式	機種	結果
bw	シフト	MI-EX1	9350
	表引き	MI-EX1	9240
	シフト	MI-C1	10990
	表引き	MI-C1	10390
copy	シフト	MI-EX1	12370
	表引き	MI-EX1	12270
	シフト	MI-C1	16130
	表引き	MI-C1	16010
gradient	シフト	MI-EX1	12060
	表引き	MI-EX1	16060
	シフト	MI-C1	15790
	表引き	MI-C1	29350

ZaurusOS の深度 16 ビットの画像とは，RGB 成分がそれぞれ 5-6-5 のものである．一方，Squeak でいう深度 16 ビットの Form とは RGB 成分がそれぞれ 5-5-5 のものである．そのため，Squeak の Form 形式となっている Display から Zaurus の VRAM 形式に変換するために，各画素に対して

```
((pixel&0x7FE0)<<1)|(pixel&0x1F);
```

という変換を施す必要がある．

このような変換を行う方法としては，実際に各画素に対し上記のような式を適用して値を変換するものと，あらかじめ変換表を用意しておき，実行時にはその表によって変換するものとが知られている．一般に，メモリアクセスのコストにそれほど気を使う必要のないプラットフォームであれば，表引きの方が高速であることが知られている．

しかし，Zaurus の場合はメモリアクセスのコストが非常に高いため，メモリアクセスが多い表引きは不利である可能性が高い．これを評価するために，いくつかの Squeak コードによって実行時間を測定した．結果を**表 5** に示す．

表中，`bw` は Display 内の矩形領域を単一色で塗り潰すもの，`copy` は単一色で塗り潰された Form を Display にコピーするもの，`gradient` は複雑なグラデーションのかかった Form を Display にコピーするものである．操作の対象となる画像の大きさはどれも 224×224 であり，深度は 16 ビットである．

表中，シフトとは上記のようなマスクとシフトによる画素変換方式であり，表引きとはその名のとおりあらかじめ求めておいた表に従って画素を変換する方式である．

図 2 インタプリタとイベントハンドラ
Fig. 2 Interpreter loop and the event handler.

bw および copy の結果を見ると，どちらの画素変換方式でも表引き方式の方が高速であるもの，差はたかだか 1，2%程度と大きな差はないことが分かる．この例のようにまったく同じ色のみを使用する場合には，変換表の 1 つの要素のみが参照される．その場合，その要素がキャッシュに載って表引きのためのメモリアクセスが行われなくなる結果，実行時間を支配している実際のメモリアクセスの回数がどちらの方式でも同じになっていると思われる．

一方，gradient で行ったように複雑なパターンを表示する場合は，表引きの方が MI-EX1 では 33%，MI-C1 では 85%遅くなっている．gradient では，各画素を順に変換していく際に毎回異なる色を変換する必要があるが，これを表引きで行った場合には，毎回異なる表の要素が参照されるため，表引きのためのメモリアクセスが顕在化し，低速化の原因になったと考えられる．MI-C1 が MI-EX1 と比較して大きなオーバヘッドとなっているのは，CPU キャッシュの大きさが影響していると考えられる．

この結果から，Squeak/Zaurus では，多少複雑な画像を表示する場合にも性能が劣化しないシフト方式を採用した．

なお，Squeak の公式開発チームが，マシン依存な画素形式と VRAM を Display として直接使うためのインタフェースなどを実装中であることを公表している．これを使用すれば，Zaurus での画面表示は 2〜3 倍程度高速化されることが期待される．

5.3 イベントハンドラ

Squeak のようにユーザレベルでの並行プログラミングを許すような言語処理系では，ユーザイベントがない場合にもインタプリタは動作を続けなくてはならない．**図 2** は図中左側の関数として表されているインタプリタが，適切な時点でイベント処理を行う ioProcessEvents() 関数をときどき呼び出す（**ポーリングする**）という状況を表している．もし呼び出された ioProcessEvents() がイベント待ちでブロックしてしまうとインタプリタ全体が停止してしまうことになる．また，Squeak は対話的に使用されることを想定しているため，ユーザからの入力をできるだけ滑らかに処理できることも要求される．

しかし，Zaurus OS 用アプリケーション（MORE ソフト）の標準的なフレームワークでは，アプリケーションをイベント駆動型で書き，ユーザからのイベントがない場合には OS の中でアプリケーションがブロックすることを想定しているため，これに従って仮想機械を構築することはできなかった．

また，イベント取得用の関数として当初から公開されていたものは，ユーザプログラムからは直接呼ぶべきではないとされている GetMessage() と呼ばれるものだけであった．この関数は呼び出したときに配送されるべきイベントがない場合にはブロックしてしまうものであるため，なんらかの対策を講じる必要があった．

幸い，FastTick と呼ばれる 30 ミリ秒ごとに発生するとされている☆イベントの発生を指示する API が存在し，OS 内で GetMessage() がブロックしてしまっても，FastTick が発生することによって復帰することが判明した．図 2 の ioProcessEvents() の擬似コードは，OS の API である GetMessage() を呼び出し，イベントの取得を試みる．もしイベントがなくてブロックしてしまっても，FastTick が発生した時点で復帰し，何もイベントがなかったかのようにイン

☆ FastTick イベントが発生する実際の条件はなお不明である．

タプリタに帰る，ということを表している．

`ioProcessEvents()` 内の `switch` 文の各 `case` 節には，ZaurusOS が定義しているイベントのうち，機能が判明していて，Squeak を動作させるうえで必要なものが書かれている．`GetMessage()` によって意味のあるイベントが取得された場合は，そのイベントがインタプリタと共有しているリングバッファに格納されることによって，イベントハンドラからインタプリタにイベントが通知される．

当初は上記のような方式で記述していたが，その後，配送されるのを待機しているイベントの有無を（ブロックせずに）調べる `LookEventMsg()` という API の存在も非公式にシャープから示された．`ioProcessEvents()` の先頭でこの API を用いて不要なチェックを除去することにより，対話的性能を大きく向上させることができた．

一方，シャープからはカーネルスレッドの使用法も正式にアナウンスされた．カーネルスレッドを用いれば，イベントハンドラとインタプリタ自身を別のスレッドとして動作させることによって，Zaurus アプリケーション開発環境が想定しているイベント駆動スタイルを崩すことなく仮想機械を実装することも原理的には可能となる．

そこで，ポーリング方式とカーネルスレッドのどちらが好ましいかを，ユーザの行った操作がどの程度の頻度で Squeak に報告されるかを調べることによって測定した．測定には，画面上でペンを滑らせたときに，Squeak が 1 秒間に認識した異なる点の個数を使用した．**表 6** にその結果を示す．表中，ポーリングとは `ioProcessEvents()` 中で上記の `LookupEventMsg()` を使用している方式であり，カーネルスレッドとはイベント処理のみを行うカーネルスレッドを使用する方式である．

表 6 の結果から，ポーリング方式の場合 MI-EX1 および MI-C1 では毎秒 40 個程度の異なる点を取得可能であることが分かる．しかしカーネルスレッドを使用した場合には，MI-EX1 では 3 個程度，MI-C1 では 9 個程度とかなり少なくなってしまう．これは複数スレッドを使用したときに，それらの間での並行性制御が十分に効率化されていないためであると考えられる．この結果に基づいて，現在の Squeak/Zaurus ではポーリング方式を採用している．

表 6 1 秒あたりに取得できる異なるペン位置の個数
Table 6 Number of pen positions available per second.

方式	ハードウェア	個数/秒
ポーリング	MI-EX1	39.54
	MI-C1	40.0
カーネルスレッド	MI-EX1	2.8
	MI-C1	9.1
ポーリング	G3 400 MHz Mac	65.0

Squeak に限らず，Zaurus 上でこのような仮想機械を実装する際には，イベント処理のオーバヘッドを小さくし，より滑らかな操作感を提供するためには，イベントハンドラもユーザアプリケーションの管理下にあるポーリング方式で記述した方が優れているといえる．MI-C1 で個数がそれほど大きく減少しなかったのは，OS の改良によってカーネルスレッドのコンテキストスイッチのコストが減少したためだと思われる．

現在の Squeak/Zaurus のようにペン位置が毎秒 40 個ほど報告されていれば，画面上に図形などを描こうとしてペンを動かす場合には，知覚できるような遅れはまったくなく操作できることが分かった．ただし，極端に速い速度でペンを動かした場合には，サンプリングしていることをユーザに露呈してしまうことも分かった．筆者の経験ではマウスで図形を描くよりもタッチパネルにペンで描いたほうが高速な操作が行える．そのような環境でも完全にユーザの動作に追従するためには，対照として測定した Mac 上の結果である毎秒 60 個程度以上の性能が必要であると思われる．

5.4 バイトコード範囲チェックの除去

広く知られていることであるが，**図 3** のようにバイトコード分岐のための `switch` 文を書いた場合，変数 `currentBytecode` は 0〜255 までの値しかとらないことが保証されているにもかかわらず，通常の C コンパイラはこの変数の値が飛び先テーブルの範囲からはみ出していないかどうかをチェックする命令列を出力してしまう．このチェックはほんの数命令ではあるが，20〜30 サイクルで 1 周するようなループの中に含まれているため，性能に与える影響は大きなものがある（`currentBytecode` の型を `unsigned char` にしても状況は同じである）．

これに対処する方法としては，C コンパイラが出力

```
int currentBytecode;
unsigned char *ip;
while (1) {
  currentBytecode = *ip++;
  switch (currentBytecode) {
    case 0:
      ...
    case 255:
      ...
  }
}
```

図 3 インタプリタの骨格
Fig. 3 Byte code dispatch loop.

表 7 バイトコード範囲チェック除去の効果
Table 7 Effectiveness of removing bytecode range check.

機種	除去前	除去後	効果
MI-EX1	2,486,648	2,672,632	7.5%
MI-C1	1,333,333	1,470,588	10.3%
G3 400 MHz Mac	28,131,873	30,245,764	7.5%

したアセンブリコードをさらに修正するか，GCC などが持つ 1 級ラベルの機能[3] を使用して，switch 文を使用しない形で記述するかという 2 通りが知られている．Zaurus 用のコンパイラは 1 級ラベルの機能を持たないため，Squeak/Zaurus では C コンパイラの出力したアセンブリコードを編集することによってこの範囲チェックを除去した．

範囲チェックを除去の効果を調べるために，小さなバイトコードのみを繰り返し実行するベンチマークを使って速度比較を行った．その結果を**表 7** に示す．ベンチマークの数値は 1 秒間に実行されたバイトコード数の概算となっている．範囲チェックを除去することによって，MI-EX1 では 7%程度，C1 では 10%程度の速度向上となっている．

6. Green Book ベンチマーク

本章では，MI-EX1 および MI-C1 上における Squeak の基本的な性能を，Green Book Benchmark[7] と呼ばれる Smalltalk における伝統的なベンチマークを使って評価する．結果を**表 8** および**表 9** に示す．表 8 は Green Book Benchmark のなかの小さなベンチマークの結果であり，表 9 は大きなベンチマークの結果である．また，a は MI-C1 での結果，b は MI-EX1 での結果，c は対照として測定した G3 Mac 400 MHz での結果である．紙面の都合上，表にある各項目の名前が長いものは，適宜短縮したものを示してある．

今回行った評価では，意味のある数値を得るために，高速なハードウェア上で実行しても各ベンチマークが 1 秒以上持続するように繰り返し回数を調整している．そのため，表にある数値を文献 7) のものと直接比較できないということを注意しておく．Green Book Benchmark の中で，項目名に Ref が含まれているようなものは参照カウント方式の GC を仮定しておりマーク&コンパクト方式の GC を採用している Squeak では冗長なベンチマークとなっているが，ここでは比較のために掲載する．また，Squeak では SmallInteger が 15 ビットから 31 ビットに拡張されているため，16bitArith の代わりに 32bitArith というベンチマークを新たに定義して測定した．同様に，LargeIntArith も演算結果が 32 ビット以上になるように変更した．

まず，MI-EX1 と MI-C1 との比の変化に着目する．この比はいくつかの要因によって変動している．5.2 節で述べたように，MI-C1 が使用している CPU は，キャッシュサイズが MI-EX1 のものよりも小さいと推測されるため，メモリ空間中に散在した多くの個所のデータを使用するようなプログラムの場合には CPU のクロック比以上に性能が劣化する可能性がある．一方，連続的に大量のデータを転送するようなプログラムの場合には，メモリバンド幅が支配的となり，CPU クロック数の比にはほとんど依存せずに MI-C1 と MI-EX1 の比が小さくなると考えられる．ただし，実際には明らかな特徴のあるもの以外では，いくつかの要因が関連して性能に影響を及ぼしているために，性能に与える影響の厳密な切り分けは容易ではない．

Mac との比の大小によって小さなベンチマークを分類することもできる．おおまかにいえば，転送されるデータが少ないときは Mac と MI-EX1 との比が小さく（10.0 倍以下程度）になり，データが多いときは大きく（10.0 倍以上）になっているといえる．また，浮動小数点数演算が関与するようなものも Mac と Zaurus の CPU 性能比が大きく現れていると思われる．

大きなベンチマークでは，ビットマップ転送を主とする TextDisplay を除けば，MI-C1 と MI-EX1 との比は 2 から 4 倍，MI-EX1 と Mac との比も 15 倍から 34 倍程度と大きなものになっている．オブジェクト指向言語の実行時には，広いメモリ空間内に広がったデータを少しずつ使用するという局所性の低いデータ参照になる傾向がある．そのため，2 次キャッシュなどを持たない貧弱なメモリシステムである PDA では，大きなベンチマークを実行したときに性能の劣化が如実に表れたものと考えられる．

MI-EX1 上で測定した大きなベンチマークの結果を，対照として測定した G3 Mac 400 MHz 上のものと比較すると 15 倍から 34 倍程度の開きとなっている．この数値だけを見ると非常に遅いように感じられるが，MI-EX1 や MI-C1 上の Squeak は，それほどストレスを感じることなく操作できるということを申し添えておく．

7. 関 連 研 究

携帯機器に対する興味が急速に高まっている一方，それらの上で動作する高機能なソフトウェア作成は多大なコストを必要とするため，携帯機器用のソフトウェア需要に応えるために，PDA 用のさまざまな言語処理系が作成されている．たとえば，市場規模の大

表 8 Green Book Benchmark，小さなベンチマークの結果（単位：秒）
Table 8 Result of Micro-benchmarks in Green Book Benchmark (in seconds).

名前	a	b	c	a/b	b/c	名前	a	b	c	a/b	b/c
LoadInstVar	30.79	19.45	2.32	1.58	8.38	StringAtPut	49.79	25.29	2.53	1.97	10.00
LoadTempNRef	46.43	19.77	1.52	2.35	13.01	Size	49.42	22.66	2.91	2.18	7.79
LoadTempRef	45.91	19.80	2.24	2.32	8.84	PointCreation	24.44	15.49	1.17	1.58	13.24
LoadQConst	48.76	18.48	2.11	2.64	8.76	StreamNext	55.74	23.64	2.64	2.36	8.95
LoadLiteralNRef	40.71	19.44	2.33	2.09	8.34	StrmNextPut	75.92	29.68	3.17	2.56	9.36
LoadLitIndirect	42.86	21.17	2.43	2.02	8.71	EQ	49.92	24.93	2.82	2.00	8.84
PopStoreInstVar	56.56	27.28	2.59	2.07	10.54	Class	77.18	31.76	3.71	2.43	8.56
PopStoreTemp	49.66	23.22	2.67	2.14	8.70	Value	51.92	25.42	3.00	2.04	8.47
3plus4	40.16	19.11	2.37	2.10	8.06	Creation	37.48	19.29	1.63	1.94	11.83
3lessThan4	46.71	22.21	2.54	2.10	8.74	PointX	31.64	15.64	1.67	2.02	9.36
3times4	65.75	31.45	3.51	2.09	8.96	LoadContext	42.85	20.48	2.44	2.09	8.39
3div4	52.40	19.90	2.19	2.63	9.09	BasicAt	44.69	19.08	2.72	2.34	7.01
32bitArith	38.83	19.44	2.23	2.00	8.72	BasicAtPut	47.51	22.07	2.86	2.15	7.72
LargeIntArith	101.93	29.86	2.45	3.41	12.19	Perform	56.66	25.54	3.60	2.22	7.09
ActReturn	45.55	22.35	2.43	2.04	9.20	StringReplace	52.70	24.85	3.57	2.12	6.96
ShortBranch	33.59	16.87	2.03	1.99	8.31	AsFloat	41.96	22.32	1.63	1.88	13.69
WhileLoop	43.71	21.58	2.49	2.03	8.67	FloatAddition	67.15	29.19	1.66	2.30	17.58
ArrayAt	59.36	29.23	3.44	2.03	8.51	BitBLT	113.95	75.19	3.96	1.52	18.99
ArrayAtPut	41.22	21.20	2.13	1.94	9.95	TextScanning	146.63	47.82	3.22	3.07	14.85
StringAt	66.51	32.53	3.54	2.04	9.19						

表 9 Green Book Benchmark，大きなベンチマークの結果（単位：秒）
Table 9 Result of Macro-benchmarks in Green Book Benchmark (in seconds).

名前	a	b	c	a/b	b/c
ClassOrganizer	62.03	21.74	1.16	2.85	18.74
PrintDefinition	79.88	22.91	1.44	3.49	15.91
PrintHierarchy	173.09	60.88	2.84	2.84	21.47
AllCallsOn	64.15	18.77	0.98	3.42	19.15
AllImplementors	43.11	16.81	0.85	2.56	19.78
Inspect	225.59	101.52	2.96	2.22	34.30
Compiler	238.46	100.82	2.95	2.37	34.18
Decompiler	83.97	31.01	1.15	2.71	26.97
KeyboardLookAhead	96.84	45.68	2.27	2.12	20.12
KeyboardSingle	101.93	49.19	2.35	2.07	20.93
TextDisplay	53.55	33.83	3.29	1.58	10.28
TextFormatting	175.90	78.32	2.50	2.25	31.33
TextEditing	106.97	51.57	2.74	2.07	18.82

表中，a は MI-EX1 での結果，b は MI-C1 での結果，c は G3 Mac 400 MHz, MacOS 8.5.1 での結果である．

きい 3Com の Palm 用には KVM などの Java の実装や PocketSmalltalk と呼ばれる Smalltalk の実装が存在する[1),12)]．これらは Palm の限られたハードウェア性能を考慮して，携帯機器上でのプログラム記述は許していない．そのため，携帯機器上でプログラムを書いたりデータを組み合わせたりするような動的な環境を構築するために使用することはできない．

Spotless[13)] はメモリ要求を極力小さくすることを目標として設計された Java 仮想機械であり，技術的には興味深いものである．また，Quartus Forth[2)] は Palm 上でプログラミングが行えるシステムである．これらは，言語の構文や基本的な意味は Java や Forth などの既存の言語のものを用いているものの，その携帯機器に特化した特殊なライブラリを使用する必要があるため，実際にはプログラムの移植性や記述性はそれほど高くはなく，既存のプログラムや他の環境用のプログラムを動かすためには大きな変更が必要となる．

Palm 以外の高性能な PDA としては Windows CE を搭載したものや Psion があげられる．特に，Windows CE 機のいくつかは Zaurus に匹敵するような性能を持っており，REBOL や Python など，携帯機器上でプログラムの作成が可能な言語処理系も存在する[6),9)]．REBOL は移植性の高さなど Squeak と共通するものを持つが，グラフィックスをサポートして

いない点に問題がある．Windows CE 上で動作する Python も互換性のあるグラフィックスライブラリを持っていない．

Zaurus では，ル・クローンと呼ばれる開発環境も存在する[11]．ル・クローンではインタプリタを Zaurus とデスクトップ機上に用意してアプリケーション開発の高速化を図っている．この点は Squeak/Zaurus に近いが，Zaurus 上でのプログラム編集を許していない．

8. ま と め

本論文では，プログラミング環境 Squeak の Zaurus への移植と拡張について，移植にあたっての留意点と実装の概要について述べ，性能と可用性を評価した．Squeak の性質である小さなメモリ要求量や注意深く高速化されたバイトコード インタプリタという性質を生かして，PDA 上で動作するように移植し，PDA 特有の周辺機器を Squeak のオブジェクトとして扱えるようにすることによって，PDA 上に動的なオブジェクト指向言語を構築することができた．

PDA 上の実装では，ハードウェア特性がデスクトップ機とは異なっていることを考慮に入れる必要がある．Squeak/Zaurus では画面表示処理とイベント処理について，ハードウェア性能の実測値に基づいた実装方針の比較を行った．

今後の課題としては，これまで述べてきた，ネットワーク性能の向上，ペンベースでの操作に向いた GUI フレームワークの開発があげられる．また，起動・終了時間の短縮を図る必要もある．

さらなる課題としては，電池持続時間と性能とのより良いバランスを見つけることがある．一般にこれらはお互いに背反する関係があり，Zaurus では `FastTick` イベントの有効化・無効化が両者の性能に最も大きな影響を及している．現在の実装ではつねに `FastTick` が有効になっているが，これを無効化すると，対話的性能はおよそ 50%程度低下するが電池持続時間は 30%ほど向上させることができる．今後は，GUI フレームワークとの連携を図って，性能が必要なときのみに `FastTick` を有効にするような工夫をすることが考えられる．

今後とも Squeak/Zaurus の完成度をより高めるための努力を行っていきたい．

謝辞 Zaurus に関する情報収集に協力してくださった富樫吉徳氏および小笠原博之氏，ベンチマークの提案と結果の検討に協力してくださった John Maloney 氏，および論文に対する修正意見をいただいた匿名の査読者に感謝する．

参 考 文 献

1) Arseneau, E.: PocketSmalltalk. `http://www.pocketsmalltalk.com`.
2) Bridges, N.: Quartus Forth. `http://www.quartus.net/products/forth/`.
3) Free Software Foundation: Using and Porting GNU CC, for Version 2.95.
4) 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, *Object-Oriented Programming, Systems, Languages, and Applications*, pp.318–326 (1997).
5) Kay, A. and Goldberg, A.: Personal Dynamic Media, *IEEE Computer*, Vol.10, No.3, pp.31–41 (1977).
6) Lutz, M.: *Programming Python*, O'Reilly & Associates (1996).
7) McCall, K.: The Smalltalk-80 Benchmarks, *Smalltalk-80: Bits of History, Words of Advice*, Krasner, G. (Ed.), pp.251–270, Addison-Wesley (1981).
8) Ohshima, Y.: `http://www.is.titech.ac.jp/~ohshima/squeak/`.
9) REBOL Technologies: REBOL. `http://www.rebol.com`.
10) Smith, R.B., Maloney, J. and Ungar, D.: The Self-4.0 User Interface: Manifesting a System-wide Vision of Concreteness, Uniformity, and Flexibility, *Object-Oriented Programming, Systems, Languages, and Applications*, pp.47–60 (1995).
11) SOAR Systems：ル・クローン．`http://www.soar.co.jp/`.
12) Sun Microsystems: KVM. `http://java.sun.com/products/kvm/index.html`.
13) Taivalsaari, A., Bush, B. and Simon, D.: The Spotless System: Implementing a Java System for the Palm Connected Organizer, Technical Report TR-99-73, Sun Microsystems Laboratories (1999).
14) Viredaz, M.A.: The Itsy Pocket Computer Version 1.5: User's Manual, Technical Report TN-54, Compaq WRL (1998).
15) シャープ：SZAB プログラミングガイド．
16) 日立製作所：SuperH information. `http://www.super-h.com`.

付 録

A.1 Zaurus のメモリシステム性能の測定

Zaurus のメモリシステム性能を計測するために，まずいくつかの C 言語で記述されたプログラムを作成

```
int result;
for (count = 0; count < 200; count++) {
  int i, j;
  unsigned short color =
    (count % 2) ? 0xFFFF : 0x0;

  for (j = 0; j < 224; j++) {
    for (i = 0; i < 224; i++) {
      result += color;
    }
  }
}
```

図 4 ベンチマーク add の擬似コード
Fig. 4 Pseudo code of add benchmark.

した．それらは

add メモリアクセスをいっさい行わず，レジスタ間の足し算のみを繰り返すもの

write メモリ領域への定数書き込みを行うもの

read メモリ領域からの読み出しを行うもの

vram-write VRAM への定数書き込みを行うもの

vram-read VRAM からの読み出しを行うもの

である．時間の計測は ZaurusOS の API である GetFastTickCount() を用いて行った．この関数で得られる結果の解像度は 10 ミリ秒である．

add のコード断片はおおむね**図 4** のようになっている．プログラム中 224 という定数は，QVGA 画面上に収まるような領域を想定した値である．図 4 のプログラムを 4 回アンローリングを指定してコンパイルすると，出力されたアセンブリコードの最内ループは**図 5** のようになる．ただし，dt r0 は r0 レジスタを 1 減算する命令であり，bf/s はその演算の結果による条件ジャンプである．

図 5 のコードは，SH-3 のパイプラインが理想的に流れた場合 1 周 6 マシンサイクルで実行される．図 4 では $10035200 = 200 \times 224 \times 224$ 回の加算を行うので，すべての加算が最内ループで実行されると仮定すると，CPU クロックが 120 MHz（推定）の MI-EX1 では 125 ミリ秒程度，60 MHz（推定）の MI-C1 では 250 ミリ秒程度で実行できるはずである．

add の実際の実行時間は MI-EX1 上で約 160 ミリ秒程度，MI-C1 で 310 ミリ秒程度であった☆．最内ループ以外での実行を考慮に入れれば，ユーザアプリケーションでほぼ推定どおりの CPU 性能を引き出せるといえる．

これをふまえて，次に write から vram-read までのベンチマークを EX1 上および C1 上で実行した．

☆ これらの経過時間は 10 msec という測定誤差に対して小さすぎるが，ここでは議論に影響を与えない．

```
.L2978:
        add     r3,r8
        add     r3,r8
        add     r3,r8
        dt      r0
        bf/s    .L2978
        add     r3,r8
```

図 5 add の最内ループに対応するアセンブリコード
Fig. 5 Assembly code for the inner-most loop of add.

表 10 メモリブロック転送ベンチマークの結果（単位：ミリ秒）
Table 10 Memory block transfer benchmark results (in milliseconds).

項目	ハードウェア	結果
add	MI-EX1	160
	MI-C1	310
write	MI-EX1	3450
	MI-C1	3530
read	MI-EX1	1900
	MI-C1	2000
vram-write	MI-EX1	2540
	MI-C1	3460
vram-read	MI-EX1	2920
	MI-C1	3470

結果を**表 10** に示す．プログラムのおおまかな構造は add と同じであるが，ループの中では加算の代わりにメモリの連続領域に値を書き込んでいく．メモリアクセスの単位として 8 ビット単位，16 ビット単位，32 ビット単位のそれぞれに対して計測したが，それほど大きな差は見られなかったので，表には 32 ビット単位（int 単位）で行ったもののみを示している．

write では，$20070400 = 200 \times 224 \times 224 \times 2$ バイトのデータを書き出している．この処理におよそ 3.49 秒かかっていることから，MI-EX1 および MI-C1 の実効メモリ書き込み性能は 20.07 (MB)/3.49 (sec) ≃ 5.7 (MB/sec) 程度であり，実効読み出し性能は同様に 20.07 (MB)/1.95 (sec) ≃ 10.3 (MB/sec) 程度であると推定できる．

MI-EX1 においては，VRAM への書き込みは通常のメモリへの書き込みと比較して有意な差があるが，MI-C1 ではほとんど差が見られない．正確な理由は不明であるが，SH シリーズのマニュアルによれば，MI-EX1 に搭載されている 7709A はメモリ空間のセグメントごとに異なる転送速度を設定する機能があるので，MI-EX1 ではこの機能を使って VRAM にマップされたアドレスへの転送が高速化されているか，あるいは異なるキャッシュの振舞いを示すように設定されている可能性がある．

キャッシュサイズに関しては，1 kB から 32 kB までの大きさのメモリ領域に対して繰り返しアクセスを行うようなプログラムを作成し実行時間を測定した．

測定値からは，MI-EX1 は 16 kB，MI-C1 は 8 kB のキャッシュを搭載しているという結果が得られた．

A.2 移植に至る歴史

そもそもの経緯は，筆者の 1 人が，97 年 10 月に行われた OOPSLA に参加したことである．OOPSLA の exhibition などで Smalltalk が活発に扱われていることを目の当たりにするとともに，Dan Ingalls による文献 4) の発表を聞き，実際に使用されているシステムとしての Smalltalk に興味を持った．また，Squeak の公式開発メンバーと東工大の松岡氏との間に面識があり，雑談の中で松岡氏の持っていた Zaurus（MI-506）への移植も話題としてあがったため，その実現に強い興味をいだいた．

帰国後，Zaurus 用アプリケーション（MORE ソフト）の作成に関し情報を収集したものの，当時は ZAB と呼ばれていた開発環境はシャープと特殊なライセンスを結んだ企業に対してのみ提供されており，個人が研究目的で入手することは不可能であったため，開発作業は停止状態のまま数カ月を送ることとなった．

98 年 5 月になり，シャープは開発環境を一般に市販する方針に転換し，東京ビジネスショウおよびその他の機会で「SZAB 4.0 お試し版」の配布が開始された．その後，6 月から 7 月の実働 2 週間ほどでペン入力と画面表示のサポートコードを記述し，当時筆者が所有していた MI-506 上で動作させることに成功した．

7 月の半ばに，タイミング良く Squeak Central のメンバーである Alan Kay と John Maloney が来日したので，そこで MI-506 版のデモを行った．その後細かな修正を行い，9 月にダウンロード可能な形のものを作成し公開した．筆者の知る限り，この時点では MORE ソフトを公開しているのは筆者らのグループのみであった．

98 年 10 月には OOPSLA に MI-506 版を持参して Squeak BOF においてデモを行った．

1999 年の 4 月に MI-EX1 が発売になった．MI-EX1 は VGA サイズの画面を持つ驚異的な仕様の PDA として当初注目された．主にメモリ容量の拡大によってさまざまなことができるようになったため，1999 年 5 月から 10 月にかけて，デジタルカメラやネットワークなど，その他のデバイスへの対応を行った．11 月には再び OOPSLA の Squeak BOF において，MI-EX1 版のデモを行った．1999 年の 12 月には MI-C1 が発売となった．MI-C1 は API としては MI-EX1 からほとんど変化がなかったため対応は非常に簡単であった．

現在は，VM の機能はおおむね完成に近づいているといえるところまで来つつある．今後はその上で動く実際のアプリケーション構築を目指したい．

(平成 12 年 3 月 7 日受付)
(平成 12 年 5 月 24 日採録)

大島 芳樹（正会員）

1972 年生．1994 年東京工業大学理学部情報科学科卒業．1996 年同大学大学院博士前期課程修了．現在同大学院博士後期課程在学中．プログラミング言語や携帯情報機器等に興味を持つ．

脇田 建（正会員）

1965 年生．1989 年東京大学理学部情報科学科卒業．1991 年同大学大学院理学系研究科情報科学専攻修了．東京工業大学大学院情報理工学研究科数理・計算科学専攻講師．博士（理学）．プログラミング言語，分散処理等に興味を持つ．日本ソフトウェア科学会，ACM，IEEE 各会員．

佐々 政孝（正会員）

1948 年生．1970 年東京大学理学部物理学科卒業．1974 年同大学大学院博士課程中退，東京工業大学理学部情報科学科助手．1981 年筑波大学電子・情報工学系．1992 年東京工業大学理学部．現在同大学情報理工学研究科数理・計算科学専攻教授．理学博士．プログラミング言語，コンパイラ生成系，属性文法，プログラミング環境に興味を持つ．著書「プログラミング言語処理系」（岩波書店）．日本ソフトウェア科学会，ACM，IEEE 各会員．