B-49

SoftwarePot/iPAQ: PDAの環境に適応するソフトウェアパッケージの作成と実行を支援するミドルウェア

SoftwarePot/iPAQ: A Middleware for Software Packages that Adapt to Computing Environment of PDAs

大山 恵弘[†] 神田 勝規[‡] 加藤 和彦[†§]
Yoshihiro Oyama Katsunori Kanda Kazuhiko Kato

1. SoftwarePot

携帯情報端末（PDA）が急速に広まっている。PDAには二次記憶等の資源が貧弱であるという欠点がある。その欠点を補うために、PDAと外部のPCを連携させてPDAの可用性を高める試みがなされている。

SoftwarePot/iPAQは、外部の計算機上の資源を利用するPDA用ソフトウェアパッケージの作成を支援する。SoftwarePot/iPAQは、SoftwarePot [1, 3] をiPAQ上のLinux向けに実装し、PDA用の機能で拡張したものである。SoftwarePotは、ユーザの意図とパッケージの設定に従って仮想的な実行環境（ポット）を構築し、その中でソフトウェアを実行するシステムである。ポットは仮想的なファイルシステムを中に含むという特徴を持つ。SoftwarePotはアプリケーションとOSとの間に位置するミドルウェアであり、かつ、それ自身が単体のアプリケーションである。SoftwarePotはソフトウェアが発行するシステムコールの横取りによってポットを構築する。そのためSoftwarePotはソフトウェアが記述される言語を限定しない。例えばユーザは既存のバイナリをポット内で動かすことができる。

1.1 仮想的なファイルシステムの設計

図1にSoftwarePotの概要を示す。パッケージ作者はパッケージに仮想的なファイルシステムの情報を含ませ、パッケージ使用者はソフトウェアをその仮想的なファイルシステム上で実行する。パッケージ作者は、仮想的なファイルシステムを構成する各ファイルの実体をパッケージに含ませても含ませなくてもよい。実体が含まれないファイルに対しては、実体の代わりに、そのファイルから他のファイルへのマッピングを作るための情報が格納される。マッピング先として指示できるものは、ソフトウェアを実行する計算機上に存在するファイル、遠隔計算機上に存在するファイル、そして、ファイルであるかのように振る舞う実行主体（すなわちプロセス）である。極端な例では、パッケージ内の全ファイルが他のファイルにマップされておりファイルの実体を一つも含まないソフトウェアパッケージを作ることもできる。

マッピングの指示はパッケージの実行時に読み出され、SoftwarePotの実行時システムに対してマッピングの要求が発行される。要求されたマッピングが実際に作られるかどうかは、パッケージの実行者が与えるマッピングポリシーによって定まる。マッピングポリシーは、マッピングの要求のパターンと、許可か禁止の指示の組の集合である。マッピングポリシーの例は、「/usr/lib以下のローカルファイルへのマッピングは許可」や、「このURLのファイルへのマッピングは禁止」である。¶

1.2 仮想的なファイルシステムの実装

仮想的なファイルシステムはシステムコールの引数の書き換えによって実装されている。SoftwarePotの実行時システムはパッケージごとに一時ディレクトリを作り、ファイルの実体をそのディレクトリに展開する。そして、展開されたファイルをアクセスするようにシステムコールの引数を書き換える。システムコールの横取り処理と書き換え処理は、iPAQ上のLinux用の我々が開発したカーネルモジュールによって実現されている。

遠隔計算機上のファイルにマッピングされたファイルは、そのファイルへの最初のアクセス時に遠隔計算機からファイルを転送することによって実装されている。現在は、webサーバを利用した読み出し専用ファイルのマッピングのみが実装されている。遠隔計算機から転送されたファイルは次回以降のアクセスに備えてキャッシュされることもあればされないこともある。

SoftwarePotは管理者でない一般ユーザが利用可能なシステムである。カーネルモジュールのインストール以外では管理者権限は必要ない。

2. PDAに適した機能

2.1 要求駆動ファイル抽出

SoftwarePotはパッケージに含まれる全ファイルの実体を一時ディレクトリに一度に展開することはしない。各ファイルの最初のアクセス時に初めて、ファイルの実体がパッケージから抽出される。そして、ソフトウェアの実行終了後に一時ディレクトリは消去される。この機構はPDAの二次記憶の恒常的な消費量を減少させる。

2.2 ソフトウェアごとのファイル容量制限

SoftwarePot/iPAQでは各ポットに含まれるファイルの総容量を制限できる。ソフトウェア実行者はポット内のファイルの総容量の上限を指定できる。もし上限を越えたら、実行時システムはポット内のファイルのいくつかを消去する‖。消去する候補になるのは以下の二種類のファイルである。

- 遠隔から取得されてPDA上にキャッシュされている読み出し専用ファイル
- パッケージから一時ディレクトリに展開された読み出し専用ファイル

[†]科学技術振興事業団さきがけ研究21
[‡]筑波大学大学院博士課程システム情報工学研究科
[§]筑波大学電子・情報工学系

¶マッピングポリシーの他にセキュリティポリシーを与えることによって資源へのアクセスの制御もできる。例えば、ポット内で走るソフトウェアがconnectできる相手ホストを制限できる。

‖PDAが外部二次記憶装置を備えている場合には、ファイルを消去する代わりに外部二次記憶装置に移動させる可能性もある。

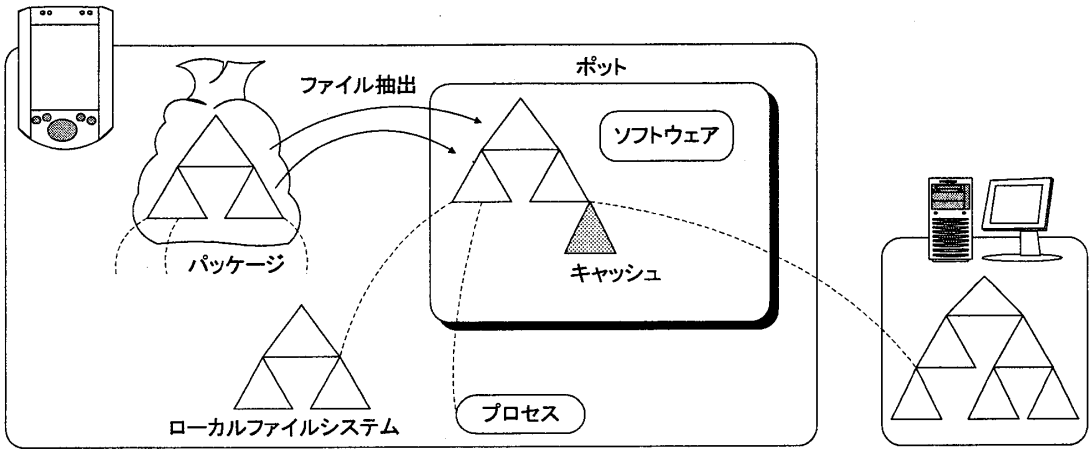

図 1: SoftwarePot/iPAQ の概要

消去されたファイルが再度アクセスされたら、再び遠隔からそのファイルを転送したり、パッケージから抽出したりする。現在は、削除するファイルは削除候補の中からランダムに選択する。今後、より良い選択戦略を考えていく予定である。例えば [2] のように、過去の参照履歴やユーザが与えるキャッシング優先度を利用することが考えられる。

ポットに含まれるファイルの総容量を実時間で正確に把握するのは簡単ではない。現在の実装では、上で述べた二種類のファイルのみを総容量の計算の対象にしている。

この容量制限の機構は、PDA の記憶容量に余裕があれば大きな記憶容量を使って高速化を図り、余裕がなければ記憶領域を節約しつつネットワークを活用して実行を続ける処理を可能にする。

2.3 動的に決定するファイルマッピング

ポット内で動くプログラムはマッピングの要求を動的に発行できる。例えば、まず PDA 上のライブラリファイルへのマッピングを要求し、もし拒否されたら、遠隔計算機上のライブラリファイルへのマッピングを要求する等の処理ができる。どのようなマッピング要求を出すかの戦略はソフトウェア作成者またはパッケージ作成者が記述する。将来にはその記述を支援する機構を SoftwarePot が提供していきたい。例えば、autoconf のような、PDA の環境の検査を簡単化する機構を作り、その機構で戦略の記述を支援したい。

動的に決定するファイルマッピングは、PDA の環境の差異や実行者の好みの多様性にソフトウェアを適応させて動かすのに役立つ。

3. 実験と関連研究

実験 SoftwarePot/iPAQ のオーバヘッドを計測した。実験に使用したベンチマークは、あるディレクトリ（ファイル数 89、ファイル総容量 5.6MB）全体を cp -r により別のディレクトリにコピーするものである。コピー元とコピー先ともに、ポット外にマップされていないディレクトリを用いた。ファイルの容量制限の機構は使っていない。上記のベンチマークをポット内で実行することによる実行時間の増加は 5.4%であった。

一般に、システムコールの実行頻度が低いアプリケーションに対し SoftwarePot/iPAQ が加えるオーバヘッドは小さい。極端な例では、フィボナッチ数列を計算するプログラムをポット空間内で実行したところ、オーバヘッドは 0%であった。

SoftwarePot/iPAQ のバイナリの大きさは 131KB、カーネルモジュールの大きさは 52KB であり、二次記憶が小さい iPAQ でも問題なく実行できる。

関連研究 NFS や samba でも、ネットワークをまたがるファイルシステムが構成できる。Coda [2] は、サーバ計算機に存在するファイルをクライアント計算機にキャッシュして使うことに注意を払って設計された分散ファイルシステムである。それらは SoftwarePot と異なり、ソフトウェアごとに異なるファイルマッピング、ソフトウェアごとのファイルキャッシュ容量制限の機能を持たず、また、通常、非管理者はファイルマッピングを作れない。

SoftwarePot に関する詳しい情報は http://www.osss.is.tsukuba.ac.jp/pot/ で得られる。

参考文献

[1] K. Kato and Y. Oyama. SoftwarePot: An Encapsulated Transferable File System for Secure Software Circulation. Technical Report ISE-TR-02-185, Institute of Information Sciences and Electronics, University of Tsukuba, January 2002.

[2] J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, 1992.

[3] 大山恵弘, 神田勝規, 加藤和彦. 安全なソフトウェア実行システム SoftwarePot の設計と実装. 第 5 回プログラミングおよび応用のシステムに関するワークショップ (SPA '02) 論文集, 2002 年.