活動履歴を活用するシステムの基本設計と漸次的開発

近藤秀樹[†] 小出洋[††] 三宅芳雄[†††]

我々は PC 上でのユーザの活動履歴を包括的に記録し活用する NecoLogger というシステムを開発した．履歴の中から必要に応じて有用な一連の活動の様子を取り出して，当面の問題解決に役立てるためのシステムである．本論文では，開発したシステムの設計と実装方式，複数ユーザの利用から得られた運用実績について報告する．膨大な活動履歴から必要に応じて過去の活動から有益な情報を得るため，人が思い出せる断片的なキーワードでも活動を取り出せるよう，できるだけ多くの検索可能なテキスト情報を記録しておく必要がある．本システムでは打鍵履歴のような PC の操作内容だけでなく，画面に表示されている文字列も検索可能にするために，OS レベルの API をフックし，画面上のテキスト情報をアプリケーションにとらわれずに記録する手法を実現した．人の活動は画面全体に広がっているため，画面の様子から活動内容を読み取れるよう，画面イメージを記録することも重要である．しかし動画で記録し続けると情報量が膨大になり，現実的ではなくなる．そこで人の活動がまとまりを持つことを利用し，画像を断続的に記録することで情報量を削減し，実用的に活動履歴を記録するよう工夫した．また，規模の大きなシステムは漸次的に開発できるほうが良いが，一方で，履歴を扱うシステムは開発初期から一貫して履歴を蓄積し続けなければならないため，開発中にデータ構造を変更するのが困難である．そこで，十分に拡張性のある中間フォーマットを設計し，柔軟なシステム開発を可能にした．これらの工夫によって，2 年以上にわたって，活動履歴を実際に蓄積・活用しつつ，同時に，ユーザの利用実態に適合した有用なシステムを漸次的に開発することが可能になった．

The Basic Design for a System to Utilize the History of Activity and Findings to Develop it Progressively

Hideki Kondo,[†] Hiroshi Koide[††] and Yoshio Miyake[†††]

We have developed NecoLogger, a recording-retrieving system that records the entire activity of personal computer use. Using NecoLogger, users can retrieve useful information and utilize it for their current problem solving. In this paper we report the overall design of the system and the details of its implementation. We also report the operational findings to show the usefulness of this system. In order to retrieve user activities efficently, we record the textual information on the screen regardless of the applications software by utilizing API hooks in OS. We also record screen images intermittently to restore the entire activity of a user. Logging the user's entire activity requires large amount of data area and it is necessary to reduce the amount to a practical level. We reduce the amount by taking screen shots at the optimal interval. Since our system should be developed extensively in the course of the development, we had to modify data structures quite often to store logging data. We adopted the XML format to meet the requirement. By adopting these methods mentioned above, we have been able to accumulate logging data continuously and have developed the system progressively to meet the newly found requirements based on the actual uses of the system.

1. はじめに

我々は，パーソナルコンピュータ（以下 PC）上でのユーザの活動履歴を包括的に記録し，その履歴を当面する問題解決に活用するためのシステムを開発してきた[1),2)]．このシステムは WindowsXP 上で動作し，キーストロークやマウス操作，画面全体のイメージや画面上のテキスト情報などを活動履歴として記録する．そしてキーワード検索や画面イメージのブラウズなど

† 中京大学大学院情報科学研究科
Graduate School of Computer and Cognitive Sciences, Chukyo University
†† 九州工業大学情報工学部
Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology
††† 中京大学情報理工学部
School of Information Science and Technology, Chukyo University

の手法で履歴にアクセスすることで，履歴から有用な一連の活動のまとまりを対話的に取り出し，当面の問題解決に役立てるものである．これまでにも，履歴を活用するシステムは少なくなかったが，その多くはアプリケーションソフトウェアの内部での履歴の活用であり，本システムのように，活動全体を対象にして履歴をとり，それを問題解決に役立てようとするシステムは少なかった．ここでは，開発したシステムの実装方式，および内部構造の詳細を報告し，あわせて，本システムの優位性をこれまでの運用実績から示す．

本システムの基本的な特徴の 1 つは，特定のアプリケーションの履歴に限らず，ユーザの活動を環境全体にわたって包括的に記録し検索可能とすることである．人は多くのアプリケーションを組み合わせて環境全体で意味のある活動を行っていると考えられるため，特定のアプリケーションの記録だけでは不十分であり，長期間にわたり活動全体の履歴を記録し利用可能にすることが必要になる．このことを実現するために，ユーザが使うすべてのアプリケーションで履歴をとり，それを集めて利用するというアプローチもある．しかしユーザの利用するアプリケーションすべてを改造することは現実的に不可能である．

また，長期間の活動を記録すると膨大な量のデータになることが予想される．活動を記録する時点とその記録を活用する時点の両方でシステムのリソースが不足しないような対策を講じる必要もある．

システムの要求仕様が開発の初期段階で十分に定まらないことも問題である．システムの仕様が定まらないため，試作，評価，改良を繰り返す必要があるが，一方で，それまでに蓄積した履歴データの形式を簡単には変更できない．履歴データは莫大な量になり，データフォーマットの変換も容易ではないため，そのまま利用できることが望ましい．

本研究では上記の問題を以下のようなシステム設計上のポイントによって解決した．

- API フックによる画面テキストの記録
- 定期的な画面イメージの記録
- 拡張可能な中間フォーマットによる連携

以下にシステム全体の構成とそれぞれの特徴について説明する．

2. システム構成

システム開発上のポイントを説明するために，本システムの概要を以下に述べる．前章でも述べたように，我々が開発しているシステムは 2 つのサブシステムが連携することによって成り立っている．サブシステム間の関連を**図 1** に示す．

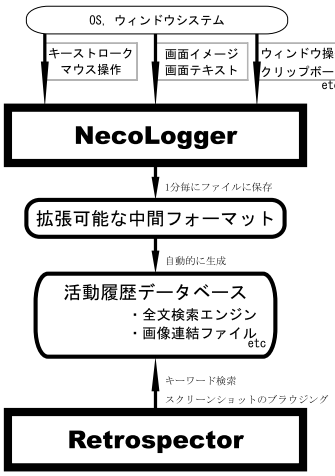

図 1 システムの全体像
Fig. 1 System overview.

図 2 稼働中の NecoLogger
Fig. 2 Running NecoLogger.

NecoLogger はもっぱら活動履歴を記録するためのサブシステムである．NecoLogger の動作している状態を**図 2** に示す．NecoLogger は C++ と VisualBasic.NET で開発されている．OS やウィンドウシステムのレベルからキーストロークやマウス操作，画面イメージ，画面テキスト，ウィンドウ操作などの情報を取り出し，拡張性のある XML ベースの中間フォーマットでログファイルに格納する．画面イメージとともにこうしたテキスト情報も同時に記録するのは，画像データをキーワード検索可能とするためである．

Retrospector は活動履歴を活用するためのインタラクティブな GUI である．VisualBasic.NET で開発

図 3 全文検索のためのインタフェース
Fig. 3 User interface for full text search.

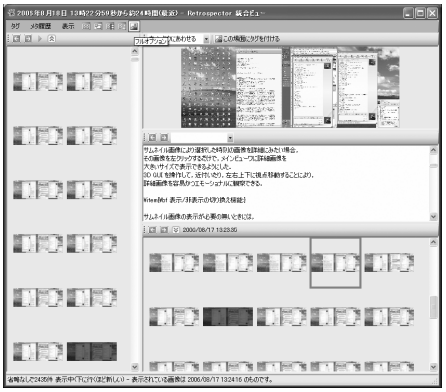

図 4 画面イメージのサムネイルのブラウジング
Fig. 4 Browsing thumbnail of screenshots.

されている．ユーザは Retrospector を操作して，活動履歴のまとまりを検索したりブラウズしたりすることができる．このとき，Retrospector は NecoLogger の出力した中間フォーマットを直接扱うわけではない．

ログファイルは可能な限り情報を詳細に残すことが重要なため冗長になっており，そのまま利用するのには向かない．そのためあらかじめログファイルを処理し，全文検索エンジンや画像連結ファイルなどに格納する．これらの集合体を活動履歴データベースと呼ぶ．活動履歴データベースを作成することで，履歴を大量に蓄積しつつ，検索可能とすることができる．ログファイルから活動履歴データベースへの変換は時間を要するが，Retrospector を立ち上げている間に少しずつ行うことで，利用者には変換していることを意識させないで済む．ユーザは実際には，Retrospector の GUI を通して活動履歴データベースを利用する．Retrospector の全文検索機能を**図 3** に，検索結果や活動履歴を画面イメージを用いてブラウズする機能を**図 4** に示す．

3. 設計のポイント

前章で 2 つのサブシステムがどのようなものか，それぞれの概要を説明した．これらのサブシステムは，以下のような特徴を実現している．

- API フックによる画面テキストの記録
- 定期的な画面イメージの記録
- 拡張可能な中間フォーマットによる連携

これらの特徴は人の活動履歴を包括的に記録し活用するために重要なポイントである．以下それぞれについて議論する．

表 1 フックしている文字列描画 API
Table 1 Hooked text drawing API.

関数名
`DrawText()`
`DrawTextEx()`
`TextOut()`
`TextOutEx()`
`ExtTextOut()`
`TabbedTextOut()`
`PolyTextOut()`

3.1 API フックによる画面テキストの記録

前章で，画面イメージと同時に画面上に表示されているほぼすべてのテキスト情報を記録するという特徴について述べた．この機能を実現するために，OS の文字列描画 API をフックして API 呼び出しの関数パラメータを取得する方法を用いた．たとえば `DrawText()` という文字列描画 API では，描画場所や描画対象となるデバイスコンテキストなどと同時に，描画したい文字列そのものが指定される．本来の API 呼び出しを自作の関数で置き換えれば，ユーザプログラムから渡される描画内容を取得できるため，結果として，画面に表示されるはずの文字列の内容を記録することができる．その後，本来の API を呼び出せば，ユーザプログラムからは通常どおりの動作となる．

API フックそのものは広く知られた技法である．市販の書籍（たとえば文献 3)）にも実現手法が詳しく解説されている．この手法は，動作中のプロセス空間に自作の DLL を注入し，各プロセスに存在する API 呼び出しのアドレステーブルを書き換えて自作 DLL を呼び出させるものである．本システムでもこの方法を用いた．具体的にフックしている API を**表 1** に示す．

このとき，単純に API をフックしただけでは，画面の再描画が発生するたびに描画内容が記録されてしまう．そして画面の再描画は頻繁に発生する．たとえばウィンドウの移動やリサイズなどのたびに重なり部分が再描画される．描画呼び出しを単純にすべて記録するとこうした再描画も記録されてしまい，画面上に表示されているテキストとはかけ離れた情報が読み出されてしまう．

そこで，文字列描画のための API をフックしつつ，すべてのプロセスが共有するフラグによって API パラメータを取得し記録するタイミングを制御するようにした．ふだんはフラグをリセットしておくことで，何も記録しないようにした．そして画面を読み取りたいタイミングでフラグをセットし，その直後に全ウィンドウに対して再描画メッセージ `WM_PAINT` を送信し，

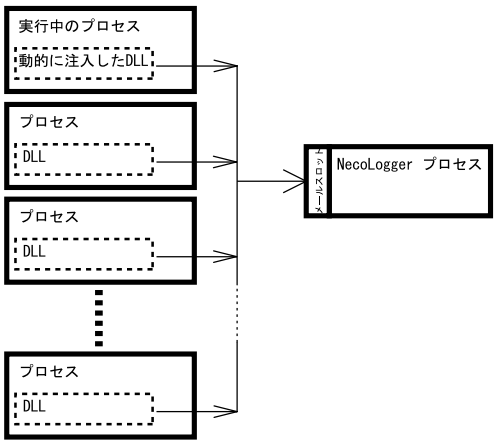

図 5 メールスロット機構を用いたプロセス間通信
Fig. 5 IPC by mailslot mechanism.

フラグがたっている間のみ，API パラメータを記録するようにした．

`WM_PAINT` を受信したウィンドウはメッセージの受信に同期して速やかに内容の再描画を行うことになっている．そのため，このタイミングでウィンドウの内容が書き直される．ウィンドウ再描画の一部として文字列が描画されるが，このとき呼ばれる `DrawText()` などの API はすべて自作の処理にすりかわっており，かつフラグがセットされているため，描画 API 呼び出しのパラメータのテキスト情報が取得される．

取得したテキスト情報は，ウィンドウ操作やキーストロークなどと同様に，プロセス間通信で NecoLogger に送信され，NecoLogger でバッファリングされた後，まとめてファイルに記録される（**図 5**）．本システムでは Win32 のプロセス間通信機構であるメールスロットを用いた．メールスロットは名前付きパイプによく似ているが，多対 1 の通信が可能という特徴がある．システム中で動作するほぼすべてのプロセスから情報を集約する必要があるため，この仕組みを採用した．

自作の処理関数の内部で必要なパラメータを取得しテキスト情報を NecoLogger に送信したあと，本来の API が呼び出されて実際に文字列が描画される．そうした処理を繰り返し，すべてのウィンドウへの再描画メッセージの配信と再描画の完了を待って即座にフラグをリセットすることにより，画面の読み取り処理が完了する．

このように，フラグ操作によるテキスト読み取りと画面イメージの撮影とを同期させることによって，重なり合うウィンドウの移動やリサイズなどによる無駄な再描画が記録されることがなくなり，画面イメージと連動したテキスト情報を取得することが可能となった．

一方で，独自の描画最適化を行っているアプリケーションではテキストを正常に読み取れない場合があると考えられる．また，テキストをスムーズにスクロールさせるアプリケーションでは，同じ内容のテキストが何度も繰り返し再描画されるため，そのつどテキストを読み取ってしまうという問題が起こりうる．

実際に運用してみたところ，Adobe Reader の表示内容がうまく読み取れなかったり，iTunes のスクロールするテキスト内容を何度も記録したりすることがあった．しかし全体としては，多くのアプリケーションが表示しているテキストを実態に近い形で取得することができた．

アプリケーションからテキスト情報を得るために，アプリケーションを拡張するプラグインを実装したりするアプローチがある[4]．このアプローチには，アプリケーションの実状に応じてきめ細かなテキスト情報を精度良く取得できるという利点がある一方で，プラグインを開発したアプリケーションでしかテキストを記録できないという制約がある．そのため，ユーザの活動を環境全体にわたって記録するには，網羅的にプラグインを開発しなくてはならず，プラグイン機構を持たないアプリケーションもあるため，現実的ではない．また，アプリケーションがバージョンアップした場合，そのつどプラグインを改修する必要もあるだろう．本システムのアプローチでは，こうした個別の開発を行わなくても汎用的に利用できる点が特徴である．

3.2 定期的な画面イメージの記録

1 章で，履歴を活用しようとするシステムはユーザの活動内容を環境全体にわたって詳細に記録できなければならないことを述べた．そこで本システムでは，画面全体のイメージとテキストを 10 秒間隔で定期的に記録するようにした．

活動内容を詳細に記録するためには，単なるスクリーンショットではなく，画面や人の動きを動画像として記録する手法も考えられる．活動を絶え間なくビデオで記録しておけば，間欠的なスクリーンショットよりも豊富な情報が記録されるためである．現実の世界では出来事がどんなタイミングで起こるのかを予測することができないため，連続的なビデオ記録は合理的である．しかし記憶領域を消費しすぎることと，実際に撮影した履歴を振り返ることが困難である，という問題があり，現実的ではない．

それに対して間欠的にスクリーンショットを記録することは不十分に見えるかもしれないが，実際にはそ

うではない．なぜなら，PC 上での人の活動はもともとインタラクティブで継続的なものだからである．たとえば画面イメージや表示されているテキスト内容は，人の活動と無関係に頻繁に書き換わるわけではない．人の活動に合わせて徐々に変化するものである．したがって，OS 内のイベントに連動しつつ，環境全体を記録することで，人の活動履歴を十分に活用でき，かつ現実的なシステムを実現できると判断した．10 秒という数値は直感的に決めた既定値であり，実情に応じて秒単位で調整可能である．

また，間欠的な記録に加えて，スクリーンセーバの動作やマウス，キーボードの操作も監視している．ユーザの操作がないと判断できる場合は画面イメージの記録を行わない．これによってさらに記憶領域の消費を抑えることができた．

3.3 拡張可能な中間フォーマットの設計

2 章で説明したように，NecoLogger は OS やウィンドウシステムのレベルからさまざまな情報を収集し，人の活動履歴として記録するシステムである．履歴は XML ベースの中間フォーマットでログファイルに記録される．画面イメージは JPEG 形式で保存するが，JPEG のバイナリデータをそのまま XML に格納することはできないため，それを base64 エンコードしたものを記録している．ログファイルは分ごとに分割され，日時ベースのディレクトリに格納される．たとえば 2006 年 10 月 30 日 23 時 45 分から 46 分のログファイルは，`$NLGDIR/NecoLoggerNeXT/$USER-$HOST/2006/20061030/20061030-23/NecoLog-$USER-$HOST-20061030234500.nlg` という名前になる．`$NLGDIR` はユーザが指定したログ格納ディレクトリ名，`$USER` はユーザ名，`$HOST` は NecoLogger を実行中のホスト名で置き換えられる．

この中間フォーマットは XML であるため，インタラクティブで高速な処理を行うには不適切である．しかし，人の活動履歴を活用するシステムを実現するには，XML のように拡張性があり，手軽に利用できることがより望ましい．なぜなら，履歴情報をできるだけ長期間利用可能な形で蓄積し，さまざまな目的で利用可能にするためである．

履歴システム開発の初期段階では，どんな情報をどのように記録すべきか，仕様を厳密に決めることができない．しかし一方で，開発初期段階に記録した履歴を蓄積し続けることができ，かつ，その履歴は将来にわたっても利用可能でなければならない．なぜなら，履歴を扱うシステムはできるだけ長い期間にわたって多くの履歴を集積する必要があるためである．

この 2 つの特徴を満たすために，本システムでは拡張性の高い XML を中間フォーマットとして採用し，必要に応じてそれを適切な形式に変換しながら利用するという設計を行った．

実際に Retrospector では，中間フォーマットを処理して高速動作のためのデータベースを作り，インタラクティブな処理を実現した．そしてその開発プロセスを進めていくうえで，全文検索エンジンなどのコンポーネントをいくつも乗り換えたり，画像の保存形式を変更したりしてきたが，2 年以上前の開発初期のログファイルが現在のバージョンでも利用可能である．

システムのバージョンアップにあわせてデータフォーマットを変換する，というアプローチも考えられる．しかし中間フォーマットを採用したことにより，変換処理にかかる時間が不要になり，かつ，長期にわたるログファイルの蓄積と管理が容易になるというメリットがある．

また，XML は多くのプログラミング言語で処理が容易なので，独自の最適化したフォーマットに比べて，実験的なシステムを構成しやすいというメリットもある．履歴の活用の可能性を探求するにはシステムを実際に実現し評価することが重要である．XML ベースのログファイルを用いることで，開発者は自分の慣れ親しんだプログラミング言語で履歴を活用するプログラムを実現できる．

4. 評　　価

実装の詳細で述べたような工夫によって，長期間の履歴を蓄積し活用するシステムを現実に実現することができた．以下にシステムの動作に関する評価と，開発と運用の実績について述べる．

4.1 動 作 性 能

履歴を記録するための NecoLogger と履歴を対話的に活用するための Retrospector を 9 名のユーザの環境で実際に運用している．以下に開発者自身が利用している環境での動作について述べる．

4.1.1 記憶領域の消費量

本論文執筆時点での最近の 1 カ月分の運用実績を分析する．2006 年 12 月の運用では，合計 62 GB のディスクが消費された．これは NecoLogger の出力する中間フォーマットのログファイル 36 GB と，Retrospector の作成する活動履歴データベース 26 GB の合計である．NecoLogger の出力する中間フォーマットのログファイルは，1 分間で約 6 MB であった．これは 3200×1200 ピクセルの比較的大きいデスクトップの場合である．UXGA サイズのモニタを 2 枚横に並

べて利用しているため，1 枚の画面イメージはこのサイズになる．ログファイルのサイズは画面イメージのサイズによって決まるといってよいため，ノート PC のような，より小さな画面ではもっと小さくなる．1 日の合算では，1 GB 程度から 3 GB 程度，1 カ月では 36 GB 程度となった．これは 2006 年 12 月の実績である．3.2 節で述べたように，操作していない間は履歴を記録しないため，画面上の情報を精読したり熟考したりする時間，休憩や外出，離席している時間などが含まれていない．そのため，1 分間のサイズには比例していない．

Retrospector の作成する活動履歴データベースは，全文検索エンジンや画像ファイルなどを連携させたものである．中間フォーマットで出力された情報量に応じた大きさとなる．26 GB のうち，全文検索エンジン Lucene.NET のデータベースが約 1.4 GB，画像ファイルを連結したものが約 25 GB であった．

本論文執筆時点で一般に入手できる 3.5 インチハードディスクは 500 GB 程度であるため，8 カ月程度の活動を 1 台のディスクで処理することができる．Retrospector の動作には中間フォーマットのログファイルは不要なため，ログファイルを逐次別のディスクに移動すれば，1 台のディスクで 20 カ月程度の活動履歴を活用することができる．

ハードディスクの記録密度は向上しつつ価格は非常に安価になってきている．そのため著者らは，ハードディスク増設が容易なデスクトップ PC においては，本手法は十分に実用的に働くと考えている．ノート PC など，ハードディスクの増設が難しい環境では改善の余地がある．

4.1.2 動作速度

Retrospector はログファイルの内容を独自のデータベースに格納することで現実的な速度を実現した．Athlon64X2 4400（2.2 GHz），RAM 2 GB の環境で，デバッガ上で動作させたときの動作速度について述べる．

1 カ月分のデータベースについてキーワード検索を行うと，850 ms から 1500 ms 程度で検索と表示を完了した．このときデータベースには，10 秒ごとのタイムスライスが 124072 件格納されていた．キーワード検索の結果は 106 件となった．

また，1 カ月分のスクリーンショットのサムネイルをブラウズしても，スクロールはユーザの操作に追従する．若干のもたつきはあったが，致命的な遅れはなかった．

4.2 運用の実績

現在管理されている最も古い履歴は開発者自身の 2005 年 3 月 5 日のものである．これ以降，可能な限り NecoLogger が実行され，活動履歴が記録されている．このユーザの約 2 年分の XML ログファイルでの総量は 729 GB である．この記録すべてが最新の Retrospector でも利用可能となっている．また，開発者以外にも逐次ユーザを増やしており，論文執筆時点で 9 名のユーザの環境で履歴を記録している．

本システムの運用を通じて，以下に代表されるような実際の活用事例が観察された．

- ソースコードを直そうと編集しているうちに壊してしまったが，壊す前に戻ることができた．
  バグを修正したつもりだったが，結果として動かなくなってしまった．修正直前の様子を呼び出してどのように編集したかを振り返り，ソースコードを壊す前の状態に戻すことができた．
- めったに設定を変更しない機器のパスワードを取り出せた．
  めったに設定を変更しないルータやプリンタサーバのパスワードを忘れてしまった．しかし設定画面に表示されている文字列で履歴を検索し，前回パスワードを入力した場面を取り出して，キーストロークの記録からパスワードを取り出すことができた．
- 以前試行錯誤したフリーソフトのコンパイルにもう一度取り組み，うまくコンパイルすることができた．
  頼まれて試行錯誤しつつ何度も打ち合わせてコンパイルしたフリーソフトのビルドオプションがどうだったか忘れた．しかしフリーソフトの名前で履歴を検索し，その様子を振り返ることで，最終的にどのようなオプションにしたのか，その判断根拠は何であったか，その過程でどんな資料を参考にしたのか，を理解することができた．
- オンライン RPG で他人から案内された道順を振り返り，同じ場所にたどり着くことができた．
  オンライン RPG で他のプレイヤに案内された場所にもう一度行こうとしたが，どうやってそこまで行ったか思い出せなかった．しかし案内されている場面を振り返りながら道をたどることでたどり着くことができた．

ユーザの活動から有益な情報を取り出すシステムはほかにも研究されている[5),6)]が，特定のデータ型のファイルやウェブページを記録し検索するものであった．それに対して本システムでは，人の活動環境全体

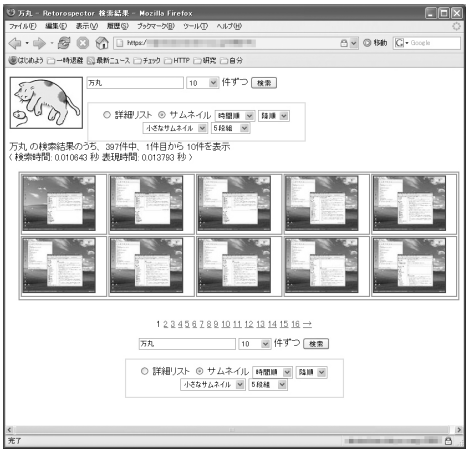

図 6 ウェブアプリケーション版 Retrospector
Fig. 6 Retrospector web application.

にわたる包括的な履歴を検索可能としている．このアプローチによって初めて前述のような履歴の活用が可能となった．

4.3 開発の経過

現実にシステムを運用している間にも開発は継続しており，さまざまな機能拡張を行ってきた．たとえば初期の Retrospector には全文検索エンジンが搭載されていなかった．画面イメージも 1 枚ずつ 1 ファイルでファイルシステムに格納していた．このため，検索速度や表示速度に大きな問題があった．そこで全文検索エンジンを導入したり，画面イメージを連結して独自に管理するなどの手法を取り入れたりして，高速化を行った．

最も新しいバージョンの Retrospector はウェブアプリケーション（**図 6**）として実現され，運用されている．長期の運用を通じて，ウェブ上で履歴を利用することがユーザの利便性を向上させ，履歴をより活発に活用する可能性があることが分かったためである．これは開発初期から予見できたことではなく，また当初からそれだけの開発を行うことも難しかったため，漸次的な開発プロセスが有効に働いた結果である．そして開発の都合にあわせて，開発言語を Ruby に変更し，開発フレームワークも.NET Framework から Ruby on Rails に変更した．全文検索エンジンも Lucene.NET から HyperEstraier に変更した．この結果，開発初期の Retrospector とは別物のようなシステムが実現された．こうした開発期間全体を通じて，活動履歴は XML ログファイルで保存されてきたため，こうした改善を逐次繰り返しても，過去の記録は失われずにつねに利用可能な状態を保つことができた．

4.4 セキュリティ上の問題の検討

本システムは個人の活動履歴を包括的に記録するので，記録内容にはデリケートな情報が非常に多く含まれる．運用に注意しないと，こうした情報が思いがけず流出してしまう可能性がある．

ローカル PC で動作する NecoLogger と Retrospector の場合，取り外し可能なディスクに履歴を蓄積し，PC を利用しないときはディスクを外して安全な場所に保管することで，問題の少ない運用が可能だろう．ファイルシステムレベルでの暗号化も有効に働くと考えられる．

それに対して最新バージョンのシステムでは，ローカル PC には原則として履歴を保持せず，1 分刻みにサーバ側に送信される．ネットワークが利用できない場合はローカル PC に蓄積されるものの，サーバへの接続が確立した時点で蓄積されていた履歴はサーバに移動される．このため，ローカル版のようにローカル PC を特別に保護する必要はないが，通信経路の盗聴への対策や，情報の流出に対する対策が必要となる．

本システムでは通信をすべて HTTPS 経由にすることで盗聴に対処している．すべての通信内容が暗号化されるため，信頼できないネットワークを経由した利用でも問題はないと考えている．

一方，履歴サーバ上のデータは，サーバ側での検索を可能とするため暗号化されていない．そのためサーバの管理者がすべての情報を知りうる立場にあるという制約がある．問題なく利用するためには，少なくともメールサーバと同程度に信頼できる管理者がサーバを運用するという前提が必要だと考えている．電子メールもデリケートな情報を扱うが，運用が信頼できるという前提に成り立っているシステムである．管理者が信頼できる場合には，gmail[7] のように利用される可能性が十分にありうる．

将来，検索に必要なインデックスをサーバから分離してローカル PC に配置することで，技術的に解決できる可能性がある．検索はローカル PC で実行し，履歴の実データをサーバから取り出すようにすることで，サーバ上のデータを暗号化しても問題なく利用できるようになるだろう．

4.5 動画によるイメージ保存の可能性

本システムでは定期的に画面イメージを撮影することで活動内容を記録しているが，動画像で記録することでデータ量をさらに削減できる可能性があると考えられる．画面の表示内容は大きく変化するわけではないため，フレーム間圧縮によってさらに情報量を減らすことが期待できるためである．そこで記録された静

止画像を試験的に 30 フレーム/秒の動画として圧縮してみた．すると，データ量は約 10%程度にまで小さくなったが，概要をつかむことはできるものの，フレームごとの画質は大幅に低下し，表示されている文字列を読み取りにくいものとなった．今後，さらに詳しい検討が必要である．

一方，非常に長い期間の活動を高速再生することで，概要を把握できる可能性があることが分かった．現在 10 秒ごとに 1 枚の画面イメージを記録しているため，30 フレーム/秒の動画として圧縮・再生すると 300 倍速で活動を振り返ることができることになる．8 時間の活動を 100 秒程度で振り返ることができるため，大量の活動履歴を活用するための一手法として今後探索する価値があると考えている．

5. ま と め

人が PC 上で行う活動の履歴を包括的に記録し，当面の問題解決に活用するシステムを設計した．そして長期間にわたって継続的に開発・改修を続けながら運用し，これまでの類似のシステムではカバーできない活用事例を明らかにした．

PC 上での過去の活動から一連の履歴を取り出し活用するためには，特定のアプリケーションだけに注目するのではなく，人の活動環境全体にわたる活動履歴を長期間記録することが必要である．OS の API をフックして関数呼び出しのパラメータを取得したり，ウィンドウシステムのイベント配送から情報を取得したりするなどの技術を組み合わせることで，画面全体のイメージやテキスト，PC の操作履歴などを複合的に記録することができ，これまでにない履歴の活用が可能となった．

詳細に活動を記録しすぎると，情報量が多くなりすぎ，現実的でなくなるが，間欠的に記録することでこの問題を解決できる．動画像のような形で記録すると，格納することも，検索し取り出すことも困難になる．そのため，活動を間欠的に記録することが有効である．PC 上での活動はそもそもインタラクティブであるため，操作履歴に合わせた断続的な記録でも有用に働くと考えられ，かつ情報量を削減することが可能である．

長期にわたって運用しながら開発・改修を続けるには，開発プロセスの初期段階から利用可能な形で履歴を蓄積する必要がある．そのため，拡張可能な中間フォーマットを中心にして開発を行うことが有用であった．このフォーマットには速度性能や記憶領域の消費量に問題があるが，ハードディスクの密度は向上し価格は非常に安価になっているため，データベースなどと併用することにより，現実的に運用することが可能であった．

謝辞 本研究は（独）情報処理推進機構が実施した 2004 年度未踏ソフトウェア創造事業の支援を受けたものである．

参 考 文 献

1) 近藤秀樹，三宅芳雄：作業履歴の記録システム NecoLogger の試作，インタラクション 2004 論文集，pp.81–82 (2004).
2) 近藤秀樹，三宅芳雄：計算機上での履歴を振り返ることによる日常作業支援，第 12 回インタラクティブシステムとソフトウェアに関するワークショップ，pp.151–152 (2004).
3) Richter, J.: Advanced Windows 改訂第 4 版，アスキー，pp.752–769 (2001).
4) 大澤 亮，高汐一紀，徳田英幸：俺デスク：ユーザ操作履歴に基づく情報想起支援ツール，情報処理学会プログラミングシンポジウム，箱根ホテル小涌園 (2006).
5) 森田哲之，日高哲雄，田中明通，加藤泰久：記憶想起支援ツール『Memory-Retriever』，インタラクション 2007 論文集，pp.173–174 (2007).
6) Gemmell, J., Bell, G. and Lueder, R.: MyLifeBits: A personal database for everything, *Comm. ACM*, Vol.49, Issue 1, pp.89–95 (2006).
7) Gmail, Google (2004). http://gmail.com/

付 録

画面テキスト取得機能の動作の概要を説明するための擬似コードを以下に示す．

A.1 画面テキスト取得のためのフック DLL の擬似コード

実行中のプロセスに動的に注入される DLL の擬似コードを次に示す．ここに示したコードでは 1 つの API がフックされている．実際には多くの API 関数がフックされる．

```
/* flag to decide to send redrawing
    text to logger process or not */
BOOL taking_screen_text = FALSE;

/* address of original api,
    initialized by DllMain */
API_FUNC *orig_api;

/* Hooked function */
int hooked_api(char *text, int length
    , int xpos, int ypos)
{
  if(taking_screen_text != FALSE){
    send_texts_to_logger(text, length
```

```
        );
  }
  /* calling original API */
  return (*orig_api)(text, length,
      xpos, ypos);
}

BOOL set_taking_screein_text(BOOL v)
{
  return taking_screen_text = v;
}
```

A.2 画面テキスト取得のためのロガー側の擬似コード

実際に画面テキストを取得するタイミングで実行される処理を擬似コードで示す.

```
void take_screen_text_callback(HWND
    hWnd)
{
  SendMessage(hWnd, WM_PAINT);
}

void take_screen_text()
{
  set_taking_screen_text(TRUE);
  EnumWindows(
      take_screen_text_callback);
  set_taking_screen_text(FALSE);
}

void mainloop()
{
  while(TRUE){
    sleep(INTERVAL_BY_MILLISEC);
    take_screen_shot();
    take_screen_text();
  }
}
```

(平成 19 年 1 月 22 日受付)
(平成 19 年 5 月 18 日採録)

近藤 秀樹（正会員）

1998 年中京大学大学院情報科学研究科修士課程修了．ソフトウェア受託開発会社勤務を経て 2004 年より中京大学大学院情報科学研究科博士課程．個人の活動履歴に注目した問題解決支援システムの研究開発に従事.

小出 洋（正会員）

九州工業大学情報工学部准教授．1997 年電気通信大学大学院電気通信学研究科博士後期課程修了．日本原子力研究所計算科学推進センター研究員，九州工業大学大学院工学研究科講師を経て，2003 年より現職．博士（工学）．並列分散処理，動的記憶管理に関する研究に従事.

三宅 芳雄（正会員）

1974 年東京大学教育学研究科修士．1982 年 University of California, San Diego. Ph.D. in Psychology. 日本電信電話（株）基礎研究所を経て 1992 年より中京大学情報科学部認知科学科教授．2005 年より中京大学情報理工学部情報知能学科教授．思考学習過程とその支援の認知科学研究に従事.