ソフトウェア工学 120－12
（1998. 7. 24）

プログラム実行情報の視覚化による理解支援ツール

鈴木 宏紀† 山本 晋一郎‡ 阿草 清滋§

†名古屋大学大学院工学研究科
464-8603 名古屋市千種区不老町
‡愛知県立大学情報科学部
§名古屋大学情報メディア教育センター
hironori@agusa.nuie.nagoya-u.ac.jp
yamamoto@ist.aichi-pu.ac.jp
agusa@nuie.nagoya-u.ac.jp

概要

プログラムを理解するには、ソースプログラムから静的に得られる情報だけではなく、関数呼出し履歴や制御の流れや変数値の変化などプログラムを実行して初めて得られる情報も不可欠である。しかし、これらの情報は膨大な量となる。このような膨大な量の情報を視覚化することで直観的な理解を促し、プログラムの理解を支援することができる。本稿では、プログラムの構造の情報の視覚表現に重ねて実行情報を視覚化する手法を提案する。そして、その手法を実装したツールを作成し、その効果を確認する。

キーワード　　ソフトウェア理解支援　視覚化　動作状況解析

A Program Execution Information Visualizer

Hironori Suzuki†, Shin-ichiro Yamamoto‡ and Kiyoshi Agusa§

†Graduate School of Engineering, Nagoya University
Furo-cho, Chikusa-ku, Nagoya-shi, 464-8603
‡Faculty of Information Science and Technology, Aichi Prefectural University
§Center for Information Media Studies, Nagoya University
hironori@agusa.nuie.nagoya-u.ac.jp
yamamoto@ist.aichi-pu.ac.jp
agusa@nuie.nagoya-u.ac.jp

Abstract

For program understanding, not only the static analysis of a source code but also the dynamic analysis of program execution, such as function call history, control flow and the change of variables, is essential. However, a mount of information collected in the dynamic analysis is so large that we need visualizing to understand intuitively. We propose a method visualizing program execution information by piling on graphical representation of program structure. We make a tool with this method and confirm this benefit.

Keywords　Software Understanding　Visualization　Behavior Analysis

1 はじめに

近年のソフトウェアプロジェクトの大規模化に伴い開発や保守のコストが増大している。ソフトウェアライフサイクルにおいて、保守のコストが最も多くの割合を占め、プロジェクトの大規模化に伴いその割合は更に増加している。その保守の作業内容のうち、プログラムの理解に最も時間を費やしている。そのため、ソフトウェアのコストを減少させるにはプログラム理解を容易にする必要がある。

プログラムを理解するには、プログラムの様々な側面を理解する必要がある。具体的には、プログラムを静的に解析した情報だけではなく、プログラムを実行させることで初めて得られる情報も不可欠である。しかし、一般的にはプログラム実行情報は膨大な量になり、それを理解するのは困難である。そのような膨大な量の情報は、視覚化することで利用者の直感的な理解を促すことができる。

本稿では、まずプログラムを解析して得られる情報を分類し、各情報間の関係を明確にする。次に、その分類に基づいてプログラム実行情報を視覚化のために分類し、構造情報の視覚表現を生かしたプログラム実行情報の視覚化手法を提案する。最後に、その手法を用いたツールを作成し、その効果を確認する。

2 関連研究とその問題点

プログラムの実行情報を視覚化することでデバッグや保守に役立てようとする研究は数多く行なわれ、ソフトウェア・ビジュアライゼーションやプログラム・ビジュアライゼーションという分野として確立している。

K. C. Cox と G. C. Roman による Pavane[1] は、アルゴリズムアニメーションシステムである。このシステムでは宣言的記述により、利用者は対象プログラムに視覚化手続きを挿入する必要がない。しかし、アルゴリズムアニメーションシステム全般に言えることだが、視覚表現が対象とするアルゴリズムに依存する。そのため、異なるアルゴリズムには新たな視覚表現が必要となるため汎用性に乏しい。さらに、実際のプログラムのアルゴリズムは効果的にアニメーションとして視覚化できるものが少ないのも汎用性が乏しくなる要因である。

高田哲司らによる VisuaLinda[2] は、並列プログラミング言語 Linda[3] の実行状態を視覚化するシステムである。このシステムは3次元視覚化により x-y 平面にプロセス関係を表示し、奥行き方向にプロセスのガントチャートを表示する。これにより、プロセス間の関係と動作を理解しやすく、従来の2次元ガントチャートと比較して、より多くのプロセスの視覚化を可能にしている。また、Linda サーバ内に視覚化手続きを組み込むことによりクライアントプログラムを変更せずに視覚化することができる。しかし、このシステムが Linda 言語という特殊な言語を対象としているため適用範囲が狭い。

Imagix 社の Imagix 4D[4] は、C/C++ 言語のプログラムを静的解析した情報の視覚化による理解支援システムである。このシステムではクラス継承図や関数呼出しグラフなどを3次元視覚化し、プロファイラの出力結果をそこに重ねて表示することができる。しかし、プログラム構造の視覚化のみを提供しているため、プロファイル以外にプログラムの動作に関する情報を得られない。

本研究では、現在、システム開発に広く使用されている C 言語を対象とする。そして、特定のアルゴリズムなどを対象とせず、実行情報全般を視覚化し、その表現に構造情報の既存の視覚表現を用いる手法を提案する。

3 プログラム情報

プログラムを解析して得られる情報を**プログラム情報**と呼ぶ。プログラム情報は、プログラムの静的な構造を示す構造情報と動作や値の変化を示す実行情報に分類される。

また本稿における**プログラム理解**とは、利用者がプログラム情報からシステムの構造や動作を理解することである。

以下の3.1節と3.2節でプログラムの構造情報と実行情報について詳しく述べる。

3.1 構造情報

ソースプログラムの制御依存関係とデータ依存関係を静的に解析することによって得られる情報を**構造情報**と呼ぶ。構造情報はプログラム

の静的な構造を示し、制御構造情報とデータ構造情報に分類される。

制御構造情報はプログラムの制御依存関係や関数呼出し関係などの情報を含む。

データ構造情報は、データの型情報やデータ依存関係の情報を含む。

3.2 実行情報

プログラムを解析する際に実行しながら解析することによって得られる情報を**実行情報**と呼ぶ。実行情報も構造情報と同様に、制御流れ情報と値変化情報に分類される。

制御流れ情報はプログラムの実行においてどのような順序で実行制御が行われているかを示す情報である。制御流れ情報は、関数レベルの実行や文レベルの実行などのようなプログラムのどのようなレベルの実行に注目しているかによって分類される。その制御流れ情報は粒度の細かいものから順に以下のように分類される。

1. 式実行履歴
2. 文実行履歴
3. 関数実行履歴
4. モジュール実行履歴

プログラムを観察するレベルに応じた1つのオブジェクトの実行のことを**単位実行**と呼び、各実行履歴の情報は単位実行の列で構成される。

値変化情報は、ある構造のデータがどの実行に影響を受けてどんな値に変化しているかを示す情報で、プログラムの変数値の変化やメモリ上の値変化の状況を示し、以下のように分類される。

- 局所変数値変化
- 大域変数値変化
- ヒープ領域値変化

また、値の変化は単位実行時に起こるため、制御流れ情報と値変化情報には関係が存在する。図1は制御流れ情報と値変化情報の対応関係を示している。

局所変数値変化が式と文の実行のみに関連を持っているのは、プログラムのスコープにより関数やモジュールのレベルからはその変化が見えないためである。

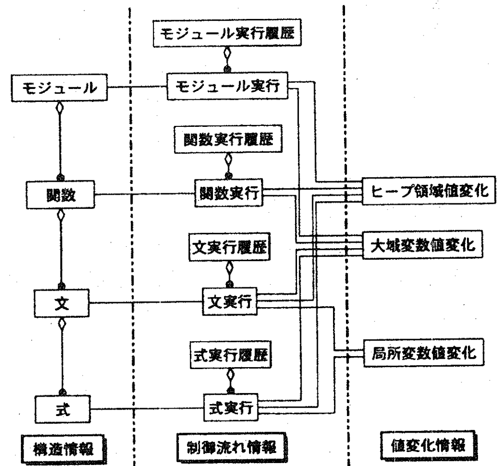

図 1: 制御流れ情報と値変化情報の対応関係

4 視覚化手法

構造情報の視覚化に関しては4.2節で述べるような表現が一般に利用され、設計やリバースエンジニアリングによる理解支援のために役立っている。しかし、実行情報の視覚化に関してはガントチャートまたはダイアグラムを用いる方法やアニメーションを用いる方法など一般的に用いられている表現が存在しない。そこで我々はプログラム実行情報を視覚化するための独自の手法を提案する。

プログラムの実行はその構造に束縛されて行われるため、実行情報を理解する際、実行情報だけを得るのではなく、構造情報を踏まえた上で実行情報を得る方が理解が容易である。そのため、実行情報を視覚化する場合も構造情報と実行情報を組み合わせて視覚化するべきである。また、構造情報の視覚化において新たな視覚表現を考えると、その見方を覚えなければならないため、既存の表現に比べて理解の妨げになる。我々は構造情報の既存の視覚表現を用いて実行情報を視覚化する手法を提案する。

4.1 視覚化のための実行情報の分類

まず、プログラム実行情報を図1に示した構成関係からモジュール層, 関数層, 文層, 式層の4つの層に分類する。この分類により、実行情報

の視覚化を簡潔に行うことができる。

モジュール層ではモジュールを実行の基本粒度としている。しかし、C 言語にはモジュールという概念が明確にないため、本稿ではプログラム中の関数群をある基準で分割した関数の集合のことをモジュールと呼ぶ。

また、関数層, 文層, 式層では、それぞれ関数, 文, 式を実行の基本粒度としている。

更に以上の 4 つの層に加えて、関数層と文層の間にブロック層も定義する。ブロック層では、文層における分岐や反復を含まない順次実行の列を 1 つのオブジェクト (**ブロックオブジェクト**) とし、そのブロックと分岐, 反復を実行の基本粒度とする。この層では、分岐や反復による制御の変化だけに注目して実行情報を得ることができる。

分類された 5 つの層の制御流れ情報と値変化情報の対応関係に基づいて、各層に含まれる情報を以下に示す。

- **モジュール層** モジュール実行履歴, ヒープ領域値変化, 大域変数値変化
- **関数層** 関数実行履歴, ヒープ領域値変化, 大域変数値変化
- **ブロック層** 文実行履歴, ヒープ領域値変化, 大域変数値変化, 局所変数値変化
- **文層** 文実行履歴, ヒープ領域値変化, 大域変数値変化, 局所変数値変化
- **式層** 式実行履歴, ヒープ領域値変化, 大域変数値変化, 局所変数値変化

このように分類することで各層の実行を基本粒度オブジェクトの順次実行とすることができるので、同様に視覚化できる。また、モジュール層から式層へと実行の粒度が細かくなっているため、粒度の粗い層ではプログラム動作の概要、粒度の細かい層ではプログラム動作の詳細の情報を得ることができる。

4.2 構造情報の視覚表現の利用

プログラム構造情報の視覚表現は、一般的にグラフ表現が用いられることが多い。プログラム構造情報はグラフ構造を用いて表すことができ、以下に各層に対応した構造情報の視覚表現をあげる。

- **モジュール** モジュール構成図
- **関数** 関数呼出しグラフ
- **文** フローチャート, PAD, NS チャート
- **式** データフローグラフ, 構文木

これらのグラフの各節点は、4.1節の各層の基本粒度のオブジェクトを示しているため、どのオブジェクトが実行されているかを示すことでプログラムの実行を視覚化することができる。

具体的には平面上に構造情報を視覚化し、その平面の奥行き方向に 4.3節で述べるプッシュダウン方式を用いて履歴を表現する。

4.3 プッシュダウン方式

制御流れ情報を理解する場合、ある実行がどの実行の後に起きていて、どの実行に影響されているかを考える。言い換えると、ある実行までの履歴情報を考慮するのである。そのため、制御流れ情報を視覚化する場合、一時点のみの情報を視覚化するのではなく、その時点までの履歴を視覚化する必要がある。

我々はオブジェクトが実行される度に奥行き方向にオブジェクトを示す図要素が押し出されていくことで履歴を表現する方式を提案し、この方式をプッシュダウン方式と呼ぶ。

この方式では、図 2のように単位実行が完了したオブジェクト (**実行完了オブジェクト**) を奥行き方向に押し出すことでオブジェクトの実行頻度を表現する。そのため、各層における実行回数の多いオブジェクトほど多くの実行完了オブジェクトが押し出されることになる。また、実行完了オブジェクトは古いオブジェクトほど奥に押し出されている。

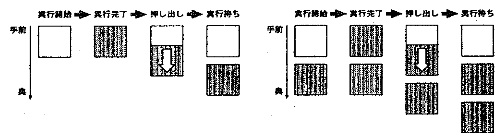

図 2: 基本動作

また、関数層における再帰呼出しのように実行中のオブジェクトが再び実行されることがあるが、プッシュダウン方式では再帰呼出しを図 3のように表現する。ここで、押し出された実行

中のオブジェクトを**再帰オブジェクト**と呼び、押し出された再帰オブジェクトの個数で再帰の深さを表現することができる。

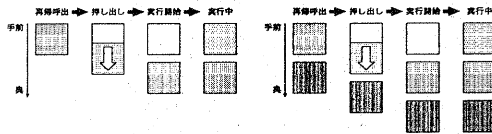

図 3: 再帰呼出し時の動作

押し出された再帰オブジェクトは図 4のように手前のオブジェクトの再帰実行が完了した時点で引き上げられる。このとき再帰実行が完了したオブジェクトは、再帰オブジェクトの奥まで押し出される。

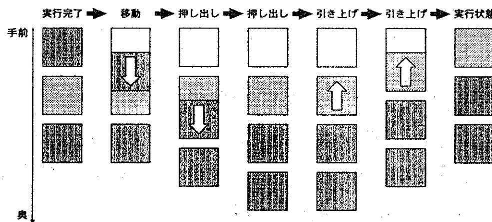

図 4: 再帰実行時の動作

オブジェクトの単位実行の進行は、そのオブジェクトの色 (**実行色**) が変化することで示す。あるオブジェクト A の実行から次のオブジェクト B の実行への制御の移動の視覚表現は、制御移動が起きた時点での A の実行色で A から B への曲線を引くことで示す。この曲線によって、A から B への制御移動が A の実行途中に起きた場合でも、A の実行がどの程度進行した時点で制御移動が起きたかを判断することができる。このプッシュダウン方式のアルゴリズムを図 5に示す。

4.4 値変化情報の視覚化

プログラム中に出現する値には、数値, 文字定数, 文字列, ポインタ値など様々な種類があるため、値変化情報を視覚化する際にはその種類に適した視覚表現が必要となる。変数が静的データ構造の場合は、値の変化のみを視覚化すれば

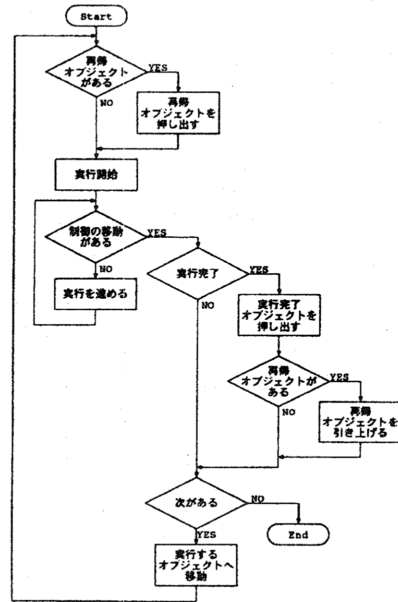

図 5: プッシュダウン方式のアルゴリズム

よいが、動的データ構造の場合にはその値の変化とともにデータ構造の変化を視覚化する必要がある。

そこで、各値の領域に**領域オブジェクト**を表示することで値変化情報を視覚化する手法を提案する。そして、その領域に入っている値のそれぞれの型について表 1のように視覚化する。

表 1: 値の型と視覚表現

値の型	視覚表現
数値	色の変化とテキスト (数値)
文字定数	色の変化とテキスト (文字)
文字列	配列領域を指す矢印とテキスト
ポインタ	領域オブジェクトを指す矢印

動的データ構造はポインタを用いて表されるので、矢印を使うことで表現できる。また、構造体や配列の視覚表現は、領域オブジェクトを適切に組み合わせることによって表現する。図 6に領域オブジェクトの視覚表現例を示す。

表 1の表現を用いて、値変化情報を視覚化する。値変化情報を理解する場合、一般的に以下

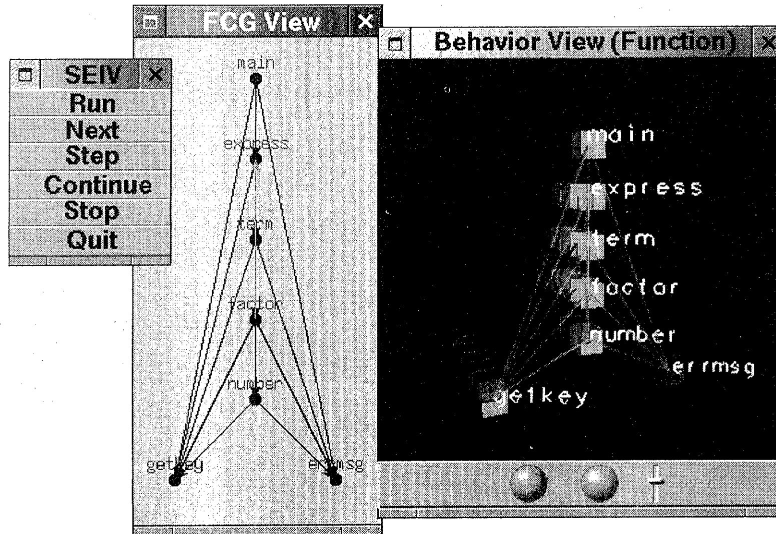

図 7: SEIV の実行画面

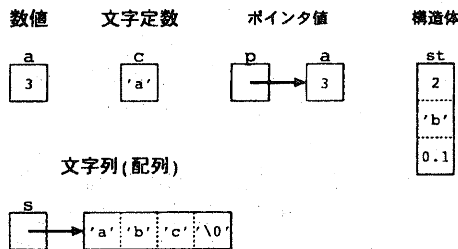

図 6: 領域オブジェクトの視覚表現例

の点に注目している。

- どの実行に影響を受けているか
- どの変数の値を参照しているか

これらの情報を実行オブジェクトと領域オブジェクトの間を値オブジェクトが行き来することで表現する。

また、変数は制御に影響を与える変数 (**制御変数**) とそれ以外の変数 (**データ変数**) に分類できる。 4.1節のブロック層では制御の変化に注目しているので制御変数を効果的に視覚化する必要がある。例えば、制御変数の値の変化に応じて制御の流れが予測できるので、その予測したパスを視覚化することが考えられる。

5 プログラム理解支援ツール

我々の研究室では細粒度リポジトリに基づくCASE ツールプラットフォーム Sapid[5] が開発されている。 Sapid では、 C 言語のソースプログラムを解析した結果をリポジトリに蓄えることができる。この Sapid を用いて関数実行履歴と文実行履歴の視覚化ツール SEIV(Sapid Execute Information Visualizer) を実現した。

5.1 システム構成

SEIV のシステム構成は図 8, 実行画面は図 7のようになっている。

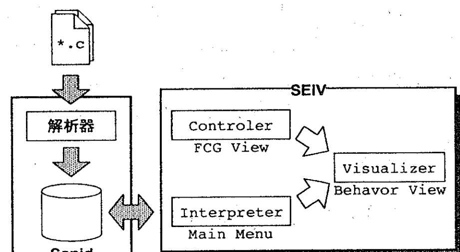

図 8: SEIV のシステム構成図

Controler は、 FCG View ウィンドウにおける利用者の入力を受け取り、 Visualizer の制御を行う。具体的には、関数呼出しグラフ (FCG : Function Call Graph) を表示し、配置の変更や FCG の節点に相当する各関数の実行をどの層の粒度で行うかの設定をすることができる。

Interpreter は、メインメニューにおける利用者の入力を受け取り、 Sapid によって構築されたソースプログラムリポジトリの情報を用いて単位実行を行い、その実行情報を Visualizer に出力する。

Visualizer は、 Interpreter と Controler からの情報を基に Behavior View ウィンドウに4節の視覚化手法を用いて視覚化を行う。

6 実行例

簡易電卓プログラム (144 行) のプログラム実行情報を SEIV で視覚化した。入力データは `1 + (1 * (1 + 1) + 1 + (1 - 1))` である。図9は、 express 関数の再帰実行を横から見た図である。この図から、 factor 関数が express 関数を呼び出すことで再帰呼出しが起きていることを知ることができる。また、再帰呼出しの深さが 2 であることも容易に読み取れる。

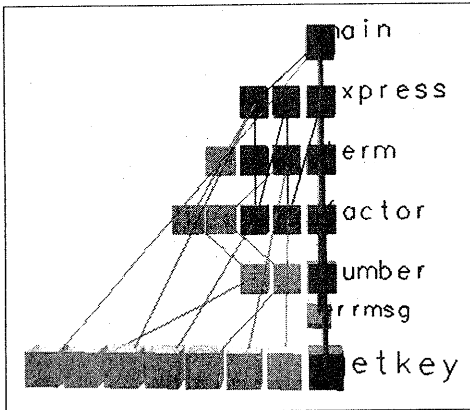

図 9: 実行途中

図 10は実行が正常に終了したときの表示例である。この図からは、まず getkey 関数が頻繁に呼び出されていることが理解できる。そのため、関数 getkey が汎用関数であると考えられる。

しかし、汎用関数は頻繁に呼び出されるために、概要の理解を妨げることもある。そのような場合、 getkey 関数の実行情報を表示しないように設定することで図 11のような結果を得ることができる。この結果から、 express 関数は main 関数から 1 回だけ呼出されて、 factor 関数から再帰的に 4 回呼出されていることが分かる。

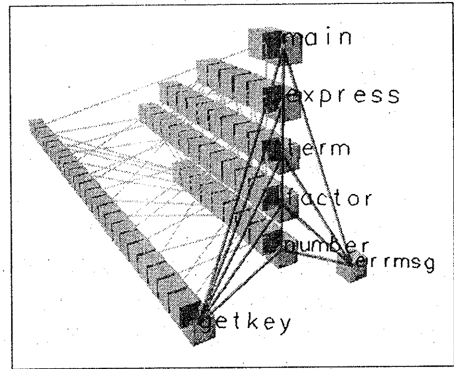

図 10: 実行終了

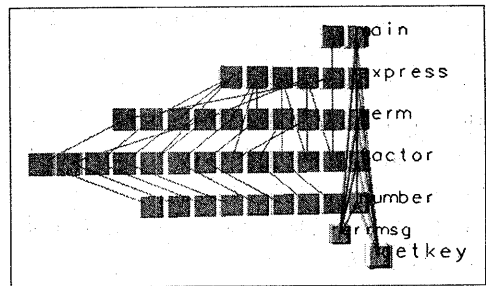

図 11: 関数 getkey を非表示

7 まとめ

本研究では、プログラム実行情報を視覚化するための分類について述べ、その分類に基づいた実行情報の視覚化手法を提案した。また、その手法を用いた実行情報視覚化ツール SEIV を実装し、それを使用することでプログラム理解を支援できることを示した。

現段階では、 SEIV は関数層とブロック層と文層の視覚化のみを行っている。今後は、モジュール層と式層の視覚化も行わなければならないが、そのためには以下のような問題点がある。

1. 関数群のモジュール分割の基準が未決定
2. モジュール実行情報の取得ができていない
3. 式層にプッシュダウン方式を適用できるかの検討

1の問題は、節4.1で述べたようにC言語に明確なモジュールの定義がないため、ソースプログラムの関数群をモジュールに分割することから行わなければならない。しかし、利用者がSEIVのFCG View(図8)ウィンドウでいくつかの関数の選択によりモジュールの設定を可能にすることで、モジュール分割の必要がなくなる。

2の問題は、関数の実行情報を用いることでモジュールの実行情報を取得することが考えられるが、モジュールの実行進度を取得するのが困難である。関数集合をモジュールと定義したために、モジュールの何割が実行されているかを判断しにくい。このために実行色でオブジェクトの実行進度を表すことが困難になる。

3の問題は、4.1節で各層を同様に視覚化できると述べたが、式層をプッシュダウン方式で視覚化しても理解が用意になるとは考えられないためである。例えば、データフローグラフは式の構造を視覚化できるが、制御構造だけでなくデータ構造も視覚化する。そのため、制御構造に着目して視覚化するプッシュダウン方式にデータ構造の視覚表現を組み合わせる方法を検討しなければならない。

また、プッシュダウン方式による視覚化結果に対して、以下のような操作を考えている。

- **制御流れ情報のスライス**
  ある実行オブジェクト以降の実行オブジェクトを取得する

この操作によって、ある実行以下の制御流れ情報を容易に取得することができるようになる。

謝辞 本研究にあたり、さまざまな助言を頂いた阿草研究室の諸氏に感謝します。

参考文献

[1] K. C. Cox, G. C. Roman, “Visualizing concurrent computations”, Proceedings of 1991 IEEE Workshop on Visual Languages, pp.18-24, IEEE CS Press, 1991.

[2] 高田哲司, 小池英樹, “VisuaLinda: 並列言語lindaのプログラムの実行状態の3次元視覚化”, 竹内彰一編, インタラクティブシステムとソフトウェアII: 日本ソフトウェア科学会WISS’94, pp.215-223, 近代科学社, 1994.

[3] N. Carriero, D. Gelernter, “Linda in context”, Communications of the ACM, Vol.32, No.4, pp.444-458, 1989.

[4] Imagix Corporation, “Imagix - Imagix 4D”, http://www.imagix.com/products/imagix4d.html

[5] 福安 直樹, 山本 晋一郎, 阿草 清滋 “細粒度リポジトリに基づいたCASEツール・プラットフォームSapid”, 情報処理学会論文誌, Vol.39, No.6, pp.1990-1998, Jun 1998.