ソフトウェア工学 115－6
（1997．7．31）

Java[1]プログラムのスレッド視覚化ツール

片山　透　中本幸一　臼井和敏
NECマイコンソフト開発環境研究所

Java 実行環境においてマルチスレッドプログラムの動作把握やデッドロックなど協調動作による問題の解決支援のためのツール，Thread Viewer を作成した．

Java 言語の登場により，プログラムをマルチスレッドを利用して作成することが比較的容易になった．その反面，マルチスレッドプログラムの問題点である挙動の把握の困難さ，協調動作におけるデッドロックの問題などの解決を支援する方法は十分ではない．

そこでそれらの問題の解決を支援するためのツールとして Thread Viewer を作成した．Thread Viewer はターゲットとなる Java プログラムの実行時にスレッド情報を測定してデータを採取するデータ測定部と、そのデータを元にスレッドの振る舞いなどを時間軸上に表示する表示部とで構成される．

このツールを用いることにより，マルチスレッドプログラムの挙動を一目で理解することができ，マルチスレッドプログラムおける諸問題の解決を支援することが可能となる．

Thread visualization tool of Java program

Toru Katayama　Yukikazu Nakamoto　Kazutoshi Usui
Microcomputer Software Engineering Laboratories
NEC Corporation

We developed Thread Viewer as a tool to solve with a problem of multi-thread program.

Programmer became easily made of the multi-thread program after the appearance of Java. On the other hand, there is no method of supporting the solution of the problem of the multi-thread program. The problem is a difficulty of understanding the thread behavior and finding a deadlock, etc.

Therefore, we made Thread Viewer as a tool to support the solution of those problems. Thread Viewer is composed of the following two tools. One is the data measurement part where the data is gathered measuring thread information when the targeted Java program is executed. Another one is a display part where the thread behavior is displayed on the axis of time based on the data.

If you use Thread Viewer at debugging process, you become understanding the behavior of the multi-thread program easily. Thread Viewer strongly supports the solution of the problem of the multi-thread program.

1.はじめに

Java 言語の登場により，プログラムをマルチスレッドを利用して作成することが比較的容易になった．その反面，マルチスレッドプログラムの問題点である挙動の把握の困難さ，協調動作におけるデッドロックの問題などの解決を支援する方法は十分ではない．

既存の技術として Allen D. Malony らの Traceview[1]，片山らの性能解析ツール[2][3][4]がある．これらはリアルタイムシステム向けの性能解析の手法であり，主にシステム全体の性能を把握することを目的としており，ユーザプログラム，特にスレッドを測定の対象にした手法ではない．

そこでユーザプログラム，特にマルチスレッドプログラムを対象としたプログラミングおよびデバッグを支援するツールとして Thread Viewer を作成した．Thread Viewer はターゲットプログラムのスレッド情報を測定して，スレッドの振る舞いなどを時間軸上に表示する．

このツールを用いることにより，マルチスレッドプログラムの挙動を一目で理解することができ，マルチスレッドプログラムの問題の解決を支援することが可能となる．

2.並行プログラムの実現方式

潜在的に並列実行が可能な複数の処理の流れをひとつのプログラムとして記述したものを並行プログラムという[5]．並行プログラムの実現方法のひとつとして，マルチスレッドプログラムが存在する．

並行プログラムには次のような利点がある．

- ●並行実行による実行性能の向上
- ●並行性のあるモデルの直感的な記述
- ●並行分割による実行効率の向上

これらの利点により，並行プログラミングを利用するメリットは多い．

2.1.並行プログラム実現のための機能

並行プログラムを実現するためには，並行にプログラムを実行するための手段と並行プログラムの同期を実現するための手段とが必要である．

並行にプログラムを実行するための手段としてスレッド，プロセス，タスクなどがある．並行プログラムの同期を実現する手段としてセマフォ，モニター，非同期メッセージパッシングなどがある．

2.2.Java の並行プログラム関連機能

Java は並行プログラムを実現するための手段としてスレッド，モニタの機能をそれぞれ有する．

Java プログラムでは Thread クラスの継承か Runnable インタフェースの実装のいずれかの方法でマルチスレッドプログラムを実現できる．また，モニタは synchronized メソッド/ブロックにより実現される[6]．

3.マルチスレッドプログラムの問題

従来，並行プログラムを作成するためには非常に大きなコストが掛かったが，Java は言語仕様としてマルチスレッドをサポートしており，一般プログラマーにも比較的容易に並行プログラムを実現することが可能である．

しかし，簡単に並行プログラムを作成することができるということは，正しいプログラムが作成できることを意味するわけではない．

並行プログラム，特に Java のマルチスレッドプログラムの問題の主なものを以下に挙げる．

- ●同期に関する問題
- ●スケジュールに関する問題
- ●システム起動スレッドの影響

以降では，Java における並行プログラムの

諸問題を考察してみよう．

3.1.同期に関する問題

Java は synchronize メソッド(ブロック)によるモニタを用いて並行プログラムの同期を実現している。これによりオブジェクトが排他制御される．

synchronize ブロック内において，同期制御のためのメソッドである wait/notify，あるいはスレッドの実行制御のためのメソッドである sleep，susupend/resume などを用いたプログラムの場合，この同期の様子とスレッドの振る舞いを関連付けることが困難である．

3.2.スケジュールに関する問題

Java は言語仕様でマルチスレッドをサポートしているが，スレッドのスケジューリングについては定められていない．スレッドのスケジューリングはプラットフォーム依存である．

一般に並行プログラムにおいてスケジューリング方式に依存するコーディングをするべきではなが，特に Java はスレッドのスケジューリング方式については保証されないため，注意が必要である．

```
class MultiThread extends Thread {
  public void run() {
    for (int I = 0; I < 5; I++) {
      System.out.println(getName() + " count:" + I);
    }
  }
  public static void main(String args[]) {
    MultiThread mt1 = new MultiThread();
    MultiThread mt2 = new MultiThread();
    MultiThread mt3 = new MultiThread();
    mt1.setName("mt1");
    mt2.setName("mt2");
    mt3.setName("mt3");
    mt1.start();
    mt2.start();
    mt3.start();
  }
}
```

リスト1 スケジューリングの例題

スケジューリング方式の違いによる問題の例として，3つのスレッドからなり，それぞれのスレッドが非同期に標準出力にそのループ回数を表示するプログラム(リスト 1)と，その実行結果(図 1)を示す[7].

Windows95 では mt1-mt3 が不規則に実行され，Solaris では起動されたスレッドの順序で実行されていることがわかる．これは Windows95 が時分割によってスレッドの切り替えをおこなっているのに対し，Solaris ではそれをおこなっていないことが原因である．

しかし，プログラムがスレッドのスケジュールに依存しているかどうかを確かめるにはスレッドの振る舞いを知る必要があるが，それは困難である．

3.3.システム起動スレッドの影響

Java は実行環境自身が複数のスレッドで動作している．すなわち，Java プログラムの実行時には，ユーザが明示的にスレッドを生成/起動しなくても，複数のスレッドが動作している可能性がある．このようなスレッドをシステム起動スレッドと呼ぶことにする．

システム起動スレッドを，以下のように分類する．

- ●ウィンドウスレッド
  AWT ライブラリのスレッド
- ●システムスレッド
  システムの動作に必要な機能のスレッド

・Windows95	・Solaris
mt1 count:0	mt1 count:0
mt2 count:0	mt1 count:1
mt2 count:1	mt1 count:2
mt3 count:0	mt1 count:3
mt3 count:1	mt1 count:4
mt1 count:1	mt2 count:0
mt1 count:2	mt2 count:1
・・・	・・・

図1 スケジュールの実行結果

なお，ウィンドウスレッドはユーザスレッドと同じスレッドグループであるが，明示的に生成/起動しなくても出現することから，ここではシステム起動スレッドに分類する．

通常，プログラミング時にはウィンドウスレッドやシステムスレッドの存在を意識する必要はない．しかし，これらのスレッドがユーザプログラムに対して影響を及ぼしている場合，それを調べる手段は存在しない．

4.Java プログラム視覚化ツール

Thread Viewer は Java プログラム実行時にスレッドに関する情報を測定し，トレースデータとして保存し，スレッドの振る舞いおよび付加情報を時間軸上にグラフ表現することにより，Java におけるマルチスレッドプログラムの諸問題の解決を支援するツールである．

4.1.Thread Viewer の機能

Thread Viewer の表示機能を以下に挙げる．

●スレッドの振る舞いの表示
●同期メソッドの呼び出しの表示
●モニタロック区間の表示
●スレッドの走行時間のグラフ表示
●イベント時のソースプログラムの表示

これら表示機能により，並行プログラムのさまざまな問題の解決を支援する．図 2に Thread Viewer の表示例を示す．

4.1.1.スレッドの振る舞いの表示

図 2に示す通り，Thread Viewer は縦軸にスレッド名を配置し，横軸は時間を表している．表示例のプログラムは 5 つのスレッドで構成されている．たとえば時刻①から時刻②まではスレッド Saru0 が走行しており，時刻②においてスレッド Saru0 からスレッド Saru3 へスレッドの実行が切り替わったことをあらわしている．①と②の長さはその間の Saru0 の走行時間をあらわしている．

スレッドの振る舞いはプログラムあるいはスケジューリング方式により異なる．通常はこれらスレッドの振る舞いを知ることは難しいが，Thread Viewer ではわかりやすい形で表示され，マルチスレッドプログラムの挙動を知ることが可能である．

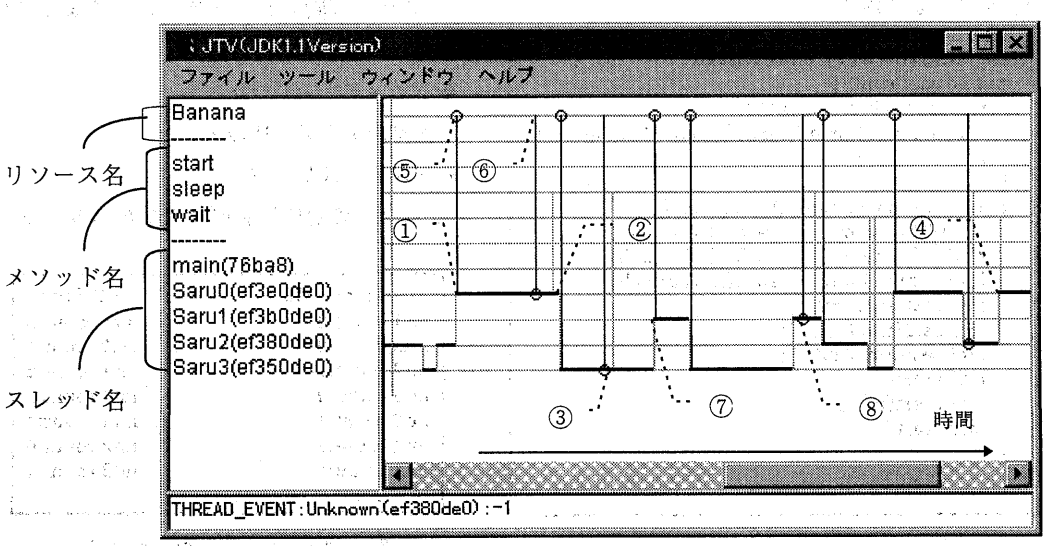

図2 Thread Viewer の表示例

4.1.2.同期メソッドの呼び出し表示

Thread Viewerでは，次に挙げる同期に関するメソッドの呼び出しを表示する．

- sleep，suspend，resumeなど
  スレッドの動作を制御するThreadクラスのメソッド
- wait，notifyなど
  同期をObjectクラスのメソッド

図2の時刻③ではスレッドSaru3がsleepメソッドを，時刻④ではスレッドSaru0がwaitメソッドをそれぞれ呼び出している．

これの機能により，スレッドの振る舞いだけでは分からないスレッド間の同期の様子を知ることができる．

4.1.3.モニタのロック区間の表示

Thread ViewerはJavaのモニタのロック区間，すなわちsynchronizedメソッド(ブロック)の入口と出口を表示できる．

図2の時刻⑤，時刻⑥間はスレッドSaru0がBananクラスのあるオブジェクトのsynchronizedメソッドを実行中であり，そのオブジェクトがロックされている，時刻⑦，時刻⑧間はスレッドSaru1がBananaクラスのあるオブジェクトのsynchronizedメソッドを実行中であり，そのオブジェクトがロックされていることをあらわしている．

これらsynchronizedメソッドの入口と出口をグラフ上に表示することでモニタがロックされている状態を把握することができる．

4.1.4.イベントのソースプログラムの表示

これまで述べたThread Viewerの機能は，スレッドの振る舞いの概要を把握するためには有効な機能であると考えられる．しかし，プログラムのデバッグにおいて，ソースプログラムとの関連を把握することが重要である．

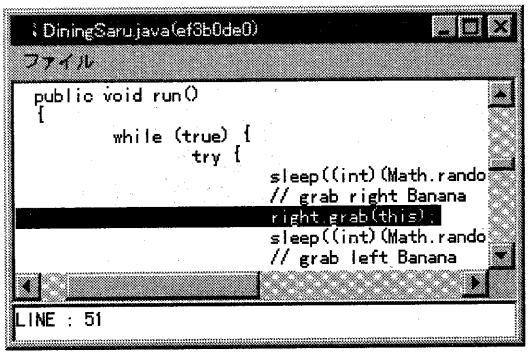

図3 ソースプログラム表示

Thread Viewerでは，グラフ上のイベントとソースプログラムを関連付ける機能を備えている．

図3はあるイベントのソースプログラムの表示の例である．

ソースプログラムはスレッドごとにウィンドウに表示されるので，スレッドの振る舞いと各スレッドのより詳細な動作を結び付けることにより，プログラム全体の動作を把握することが可能となる．

4.1.5.スレッドの走行時間のグラフ表示

2.において並行プログラムの利点として，性能や効率の向上を挙げた．しかし，プログラムの並行化によって，それらが実現されているかどうかを調べる手段はなかった．

Thread Viewerではスレッドの走行時間を全スレッドの走行時間との割合を棒グラフで示すことが可能である．

図4にスレッドの走行時間のグラフ表示の例を示す．

各スレッドの走行時間の割合を知ることにより，プログラムの並行化による性能や効率の向上を確かめることができる．

4.2.測定部の概要

次に，スレッド情報を測定するための測定部の概要を説明する．

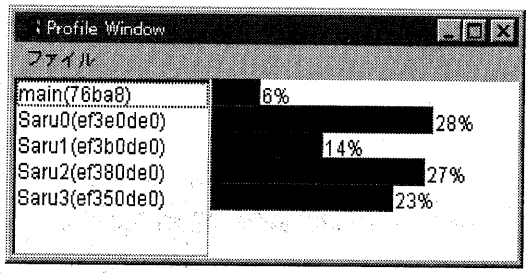

図4 実行時間グラフ

Thread Viewer は JDK(Java Development Kit)の実行環境上で実行されるプログラムを測定対象とする．測定には JDK により提供される JIT(Just in Time) Compiler インタフェース[8]を利用している．JIT Compiler インタフェースを利用して，メソッド呼び出し時に図 5のアルゴリズムのスレッド情報測定ルーチンを実行してスレッド情報を測定する．

測定方式の実現に JIT Compiler インタフェースを利用することにより，次のような利点が考えられる．

●Java 実行系の実装から独立

●ターゲットプログラムへの変更なし

これらの利点により Thread Viewer では JIT Compiler インタフェースを測定方式の実現に利用している．

5.Thread Viewer の効果と課題

デッドロックの例題として有名な「哲学者問題」を取り上げた．

以下では，「哲学者問題」を例に Thread Viewer の効果を示したのち，現状の課題について考察する．

5.1.「哲学者問題」

デッドロックは並行プログラムの典型的な問題である．その主な原因は並行プログラムのアルゴリズムの誤りであるが，発見が困難であることが知られている．デッドロック問題の例として「哲学者問題」が有名である[9].

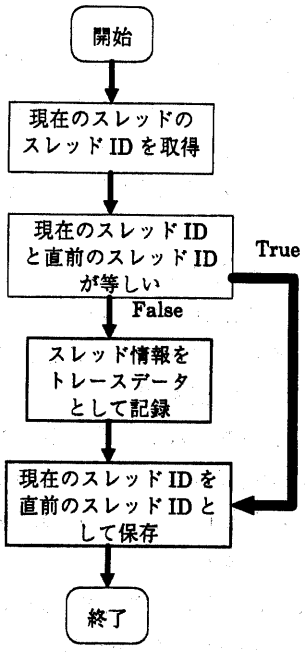

図5 スレッド情報測定ルーチン

ここでは「哲学者問題」を「食事をするサル」に置き換え「食事をするサル」の Java プログラムの概要をソース 1およびソース 2に，デッドロック時の挙動を測定した結果を図 6に示す．

図 6を参照すると，時刻①，②，③，④においてスレッド Saru3, Saru0, Saru1, Saru2 が順次左のバナナを取ろうとして grab メソッドを発行して wait し，ついにはデッドロックに陥っている状態がよく分かる．

通常のデバッグ手法ではデッドロックに至るまでの経過を観察することは不可能だが，このように Thread Viewer を用いれば容易に観察することができる．

5.2.Thread Viewer の課題

Thread Viewer には，大きく分けて次の 2つの課題がある．

●測定方法に起因する問題

●スレッドプログラムに起因する問題

```
class Saru extends Thread
{
  protected Banana left;
  protected Banana right;

  public void run()
  {
    while (true) {
      right.grab(this);
      left.grab(this);
      this.eating();
    }
  }
  protected void eating()
  {
    // eating
    right.release(this);
    left.release(this);
  }
}
```

ソース1 サルプログラム

```
class Banana
{
  protected Saru owner;

  synchronized public void grab(Saru saru)
  {
    if (owner != null) {
      wait();
    }
    owner = saru;
  }
  synchronized public void release(Saru saru)
  {
    if (owner == saru) {
      owner = null;
    }
    notify();
  }
}
```

ソース2 バナナプログラム

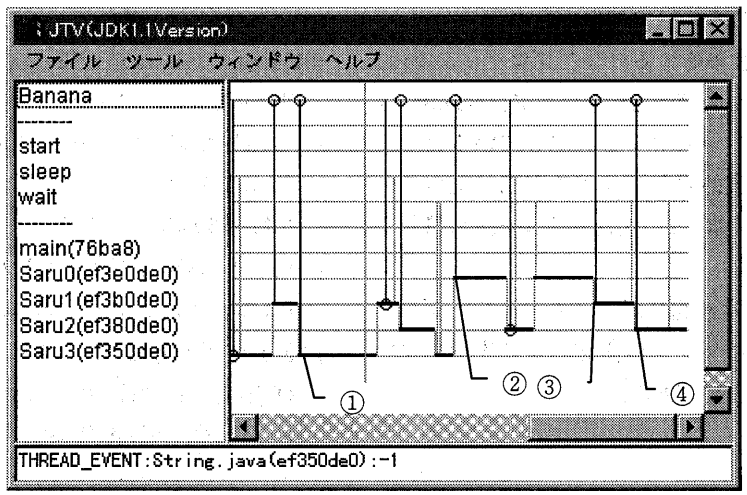

図 6 デッドロック時の挙動

以降，それぞれの問題について考察する．

5.2.1.測定方式に起因する問題

Thread Viewer では，測定に JDK の JIT Compiler インタフェースを利用している．そこで，以下のような問題がある．

●すべてのスレッドを測定できない

JIT Compiler インタフェースを利用しているため，特定のクラスのメソッドコール時にしか測定ルーチンが呼ばれないため．

●スレッド切り替えが正しくない

上述の理由によりすべてのスレッドを測定できないため．

●JDK 以外の実行系で測定できない

JDK 以外の実行系が JIT Compiler インタフェースを実装していないため.

このような制限はあるものの，スレッドの振る舞いを理解することを支援する，JDK のサポートするプラットフォームでは同一の測定方式を実現できるという利点があるため，Thread Viewer では JDK の JIT Compiler インタフェースを利用してスレッド情報を測定する方式を採用している.

5.2.2.スレッドプログラムに起因する問題

マルチスレッドプログラムの本質的な問題として，Thread Viewer では対処が難しいものも存在する.

●論理的な同期処理を表示できない

スレッド間の同期が synchronized だけで実現されるわけではないため.

5.1.の「食事をするサル」プログラムでは，owner 変数および grab メソッド，release メソッドによりセマフォを実現しているが，そのように論理的にスレッドの同期が実現されている場合，現在の Thread Viewer の機能では表現できない.

このような問題に対処するために，プログラマブルなイベントの測定および表示について検討する必要がある.

6.まとめ

Java の出現により，マルチスレッドプログラムの実現は容易になった．その反面，マルチスレッドプログラムの問題の解決を支援する方法は提供されていない.

Thread Viewer はスレッドの振る舞いなどを時間軸上に表示することにより，これらマルチスレッドプログラムの問題のうち，特にスケジューリング，同期処理など主要な問題の解決支援に役立つ.

参考文献

[1]Allen D. Malony, et al."Traceview: A Trace Visualization Tool", IEEE Software, Vol.8 No.5,1991.

[2]片山透他, "ハードウェアトレーサを用いた性能解析ツールの概要", 情処第 45 回全国大会, 7P-1,1992.

[3]上野智美他, "性能解析ツールのユーザインタフェース", 情処第 45 回全国大会, 7P-2, 1992.

[4]寺峯和裕他, "ハードウェアトレーサを用いた性能解析ツールの適用", 情処第 45 回全国大会, 7P-3, 1992.

[5]戸松豊和, "JAVA プログラムデザイン", ソフトバンク, 1997.

[6]James Gosling, "The Java Language Specification", Adison Wesly, 1996.

[7]臼井和敏他, "C/C++プログラマのための Java プログラミング入門", Interface, CQ 出版, 1996/8.

[8]Frank Yellin, "The JIT Compiler API", http://www.javasoft.com, 1996.

[9]Mary Campione, "The Java Tutorial Object-Oriented Programming for the Internet", Adison Wesly, 1996.

---

[1] Java は米国 Sun Microsystems,Inc.の商標または登録商標です.