

トレースログ可視化ツール (TLV) に対する 機能追加とリファクタリング

350802246 水野 洋樹

要旨

近年、組込みシステムの分野においてもマルチコアの導入が進んでいる。マルチコア環境では、各コアが独立して並列に動作するため、ブレークポイントやステップ実行を用いた実行時のデバッグが困難である。そのため、RTOS等のトレースログを用いたデバッグが有用と考えられるが、サイズの大きなトレースログを直接開発者が扱うのは困難である。そこで本OJLでは、トレースログを可視化表示することで解析を容易にし、マルチコア環境でのアプリケーション開発を支援である **TraceLogVisualizer(TLV)** の開発を目的とする。

TLVは、各種RTOSやシミュレータ、エミュレータなどが outputするトレースログを可視化表示するツールである。TLVは、トレースログを変換ルールに従い標準形式トレースログに変換し、標準形式トレースログに対して可視化ルールを適用し、図形データの生成を行う。TLVでは、変換ルールと可視化ルールを外部ファイルとして与えることで、汎用性と拡張性を実現する。

本OJLはフェーズに分割して開発を行なった。フェーズの終了ごとに TLV のリリースを行ない、要求の収集を行なった。

リリースを通じて要求を収集したところ、(1) 従来の変換ルール、可視化ルールでは行なえない複雑な可視化を行いたい、(2) トレースログの解析が遅いため高速化して欲しい、という要求が強かった。

(1) の複雑な可視化を実現するために、①変換ルールと可視化ルールを拡張する②変換用の言語を開発する③外部プロセスで変換・図形生成を行なえるようにする、の3つの方針を比較検討した。実装コストを利用者の学習コストを考慮し、③の方法を選択し、変換と図形生成を外部のプロセスで行なえるようにするスクリプト拡張機能を実装した。外部プロセスは任意のプログラミング言語で記述できるため、複雑な可視化を行なえる。

(2) の変換の高速化の行なうにあたり、トレースログへの変換に関するソースコードが複雑化している点が障害となった。そこで、関連するソースコードの調査した。調査の結果、クラス設計に大きな問題はなかったが、多くのクラスにおいて、複雑な処理を行なうメソッドが存在しており、これがソースコードを複雑化している原因であった。そこで、複雑化したメソッドの整理をリファクタリングの目的とした。

O J L 報 告 書

トレースログ可視化ツール (TLV) に対する 機能追加とリファクタリング

350802246 水野 洋樹

名古屋大学 大学院情報科学研究科

情報システム学専攻

2010 年 1 月

目次

第 1 章	はじめに	1
第 2 章	トレースログ可視化ツール (TLV)	3
2.1	TLV における汎用性と拡張性	3
2.2	標準形式トレースログ	3
2.2.1	トレースログの抽象化	3
2.2.2	標準形式トレースログの定義	5
2.2.3	標準形式トレースログの例	7
2.3	図形データ	7
2.3.1	座標系	7
2.3.2	基本図形と図形、図形群	8
2.4	図形データとイベントの対応	9
2.4.1	開始イベント、終了イベント、イベント期間	9
2.4.2	可視化ルール	10
2.5	TraceLogVisualizer の設計	11
2.5.1	標準形式トレースログへの変換	12
2.5.2	図形データの生成	13
2.6	TraceLogVisualizer のその他の機能	14
2.6.1	マーカー	14
2.6.2	可視化表示部の制御	14
2.6.3	マクロ表示	15
2.6.4	トレースログのテキスト表示	15
2.6.5	可視化表示項目の表示非表示切り替え	15
2.7	本 OJL で追加した機能	16
2.7.1	スクリプト拡張機能	16

2.7.2	タイトルバーに表示中のログファイル名・リソースファイル名を表示	16
2.7.3	リロード機能	16
2.7.4	ドラッグによる表示範囲移動	16
2.7.5	時刻表示のカンマ区切り	17
2.7.6	マクロビューの拡大表示	17
第3章	実績	18
3.1	開発プロセス	18
3.1.1	フェーズ分割	18
3.2	チケット終了率	22
3.3	発表実績	22
3.4	活用事例	23
第4章	スクリプト拡張機能	24
4.1	目的	24
4.2	設計	24
4.2.1	変換・可視化の実現	24
4.2.2	変換ルール・可視化ルールの拡張	25
4.3	実装	25
4.3.1	TLV ファイルの拡張	25
4.3.2	変換ルールの拡張	26
4.3.3	可視化ルールの拡張	28
4.3.4	外部プロセスによる変換・図形データ生成の実現	29
4.4	例: CPU 利用率可視化表示	31
第5章	リファクタリング	32
5.1	目的	32
5.2	対象	32
5.3	発見した問題点と改善方法	32
5.3.1	標準形式トレースログのパーサングにおける正規表現の多用	33
5.3.2	可視化ルールにおける暗黙的な木構造	33
第6章	おわりに	37
6.1	関連研究	37

6.2	まとめ	42
6.3	今後の課題	43
参考文献		45

第1章

はじめに

近年，PC，サーバ，組込みシステム等，用途を問わずマルチプロセッサの利用が進んでいる。その背景として，シングルプロセッサの高クロック化による性能向上の限界や，消費電力・発熱の増大があげられる。マルチプロセッサシステムでは処理の並列性を高めることにより性能向上を実現するため，消費電力の増加を抑えられる。組込みシステムにおいては，機械制御とGUIなど要件の異なるサブシステム毎にプロセッサを使用する例があるなど，従来から複数のプロセッサを用いるマルチプロセッサシステムが存在していたが，部品点数の増加によるコスト増を招くため避けられていた。しかし，近年は，1つのプロセッサ上に複数の実行コアを搭載したマルチコアプロセッサの登場により低コストで利用することが可能になり，低消費電力要件の強い，組込みシステムでの利用が増加している。

マルチプロセッサ環境では，処理の並列性からプログラムの挙動が非決定的になり，タイミングによってプログラムの挙動が異なる。そのため，ブレークポイントやステップ実行を用いたシングルプロセッサ環境で用いられているデバッグ手法を用いることができない。

そのため，マルチプロセッサ環境では，プログラム実行履歴であるトレースログの解析によるデバッグ手法が主に用いられる。トレースログを解析することで，各プロセスが，いつ，どのプロセッサで，どのような動作したかというプログラムのデバッグに必要な情報が全て記録される。しかし，膨大な量となるトレースログから特定の情報を探し出すのが困難である。さらに，各プロセッサのログが時系列に分散して記録されるため，逐次的にトレースログを解析することも困難である。そのため，開発者が直接トレースログを解析するのは効率が悪い。

トレースログの解析を支援するために，多くのトレースログ可視化ツールが開発されて

いる。組込みシステム向けデバッグソフトウェアや統合開発環境の一部、Unix 系 OS のトレースログプロファイラなどが存在する。しかし、これら既存のツールが扱うトレースログは、OS やデバッグハードウェアごとに異なるため、可視化対象が限定されおり、汎用性に乏しい。さらに、可視化表示項目が提供されているものに限られ、追加や変更が容易ではなく、拡張性に乏しい。

そこで後藤ら [1, 2] によって汎用性と拡張性を備えたトレースログ可視化ツール **TraceLogVisualizer (TLV)** が開発された。TLV 内部でトレースログを抽象的に扱えるよう、トレースログを一般化した標準形式トレースログを定め、任意の形式のトレースログを標準形式トレースログに変換する仕組みを変換ルールとして形式化した。標準形式トレースログから図形データを生成する仕組みを抽象化し、可視化ルールとして形式化した。TLV では、変換ルールと可視化ルールを外部ファイルとして与えることで、汎用性と拡張性を実現している。

本 OJL では、TLV の開発を継続して行なった。後藤らによって開発された TLV のリリースを行ない、要求の収集を行なった。収集した要求のうち “CPU 利用率表示などの複雑な可視化の実現” と “TLV の高速化” に対する要求が強かつたため、これらの実現を行なった。

変換ルール、可視化ルールの機能は限定的であるため、CPU 利用率などの複数のログをもとに図形を決定する必要のある複雑な可視化を行なえなかつた。そこで、標準形式トレースログへの変換、及び図形データの生成を外部プロセスで行なえるようにする。外部プロセスにすることで、任意の言語で変換ルールと可視化ルールを記述できるため、複雑な可視化を実現できる。

長時間動作するプログラムのトレースログは膨大な量となるため、トレースログ解析を高速化する必要がある。しかし、標準形式トレースログへの変換及び図形データの生成に関するソースコードが複雑化しているため、高速化のための変更を加えることが難しい。そこで、この問題を改善するために、関連するソースコードの調査し、複雑化している原因を特定し、それを改善するためのリファクタリング方針の決定を行なう。

最後に、本報告書の構成を述べる。2 章では、TLV の設計について述べる。3 章では、TLV の開発プロセスや発表実績などについて述べる。4 章では、複雑な可視化を行なうための機能追加について述べる。5 章では、高速化のためのリファクタリングについて述べる。最後に 6 章で本論文のまとめと今後の展望と課題について述べる。

第2章

トレースログ可視化ツール (TLV)

2.1 TLVにおける汎用性と拡張性

TLVは、汎用性と拡張性を実現することを目標としている。

汎用性とは、可視化表示したいトレースログの形式を制限しないことであり、可視化表示の仕組みをトレースログの形式に依存させないことによって実現する。具体的には、まず、トレースログを抽象的に扱えるように、トレースログを一般化した標準形式トレースログを定義する。そして、任意の形式のトレースログを標準形式トレースログに変換する仕組みを、変換ルールとして形式化する。変換ルールの記述で任意のトレースログが標準形式トレースログに変換することができるため、あらゆるトレースログの可視化に対応することが可能となる。

拡張性とは、トレースログに対応する可視化表現をユーザレベルで拡張できることを表し、トレースログから可視化表示を行う仕組みを抽象化し、それを可視化ルールとして形式化して定義することで実現する。可視化ルールを記述することにより、トレースログ内の任意の情報を自由な表現方法で可視化することができる。

2.2 標準形式トレースログ

2.2.1 トレースログの抽象化

標準形式トレースログを提案するにあたり、トレースログの抽象化を行った。

トレースログを、時系列にイベントを記録したものである。イベントとはイベント発生源の属性の変化、イベント発生源の振る舞いと考えた。ここで、イベント発生源をリソースと呼称し、固有の識別子をもつものとする。つまり、リソースとは、イベントの発生源

であり、名前を持ち、固有の属性をもつものと考えることができる。

リソースは型により属性、振る舞いを特徴付けられる。ここでリソースの型をリソースタイプと呼称する。

属性は、リソースが固有にもつ文字列、数値、真偽値で表されるスカラーデータとし、振る舞いはリソースの行為であるとする。

リソースタイプとリソースの関係は、オブジェクト指向におけるクラスとオブジェクトの関係に類似しており、属性と振る舞いはメンバ変数とメソッドに類似している。ただし、振る舞いはリソースのなんらかの行為を表現しており、メソッドの、メンバ変数を操作するための関数や手続きを表す概念とは異なる。

主に、振る舞いは、属性の変化を伴わないイベントを表現するために用いる。振る舞いは任意の数のスカラーデータを引数として受け取ることができ、これは、図形描画の際の条件、あるいは描画材料として用いられることを想定している。

トレースログの抽象化を以下にまとめる。

トレースログ

時系列にイベントを記録したもの。

イベント

リソースの属性の値の変化、リソースの振る舞い。

リソース

イベントの発生源、固有の名前、属性をもつ。

リソースタイプ

リソースの型、リソースの属性、振る舞いを特徴付ける。

属性

リソースが固有にもつ情報、文字列、数値、真偽値のいずれかで表現されるスカラーデータで表される。

振る舞い

リソースの行為。主に属性の値の変化を伴わない行為をイベントとして記録するために用いることを想定している。振る舞いは任意の数のスカラーデータを引数として受け取ることができる。

2.2.2 標準形式トレースログの定義

本小節では、前小節で抽象化したトレースログを、標準形式トレースログとして形式化する。標準形式トレースログの定義は、EBNF(Extended Backus Naur Form) および終端記号として正規表現を用いて行う。正規表現はスラッシュ記号 (/) で挟むものとする。

前小節によれば、トレースログは、時系列にイベントを記録したものであるので、1つのログには時刻とイベントが含まれるべきである。トレースログが記録されたファイルのデータを TraceLog、TraceLog を改行記号で区切った1行を TraceLogLine とすると、これらは次のEBNFで表現される。

```
TraceLog = { TraceLogLine, "\n" };
TraceLogLine = "[", Time, "] ", Event;
```

TraceLogLine は”[,”]”で時刻を囲み、その後ろにイベントを記述するものとする。

時刻は Time として定義され、次に示すように数値とアルファベットで構成するものとする。

```
Time = /[0-9a-zA-Z]+/;
```

アルファベットが含まれるのは、10進数以外の時刻を表現できるようにするためにある。これは、時刻の単位として「秒」以外のもの、たとえば「実行命令数」などを表現できるように考慮したためである。この定義から、時刻には、2進数から36進数までを指定できることがわかる。

前小節にて、イベントを、リソースの属性の値の変化、リソースの振る舞いと抽象化した。そのため、イベントを次のように定義した。

```
Event = Resource, ". ", (AttributeChange | BehaviorHappen);
```

Resource はリソースを表し、AttributeChange は属性の値の変化イベント、BehaviorHappen は振る舞いイベントを表す。リソースはリソース名による直接指定、あるいはリソースタイプ名と属性条件による条件指定の2通りの指定方法を用意した。

リソースの定義を次に示す。

```
Resource = resourceName
| ResourceTypeName, "(", AttributeCondition, ")";
```

```

ResourceName = Name;
ResourceTypeName = Name;
Name = /[0-9a-Z_]+/;

```

リソースとリソースタイプの名前は数値とアルファベット、アンダーバーで構成されるとした。AttributeCondition は属性条件指定記述である。これは次のように定義する。

```

AttributeCondition = BooleanExpression;
BooleanExpression = Boolean
| ComparisonExpression
| BooleanExpression,[{LogicalOpe,BooleanExpression}]
| "(" , BooleanExpression, ")" ;
ComparisonExpression = AttributeName,ComparisonOpe,Value;
Boolean = "true"|"false";
LogicalOpe = "&&"|"||";
ComparisonOpe = "=="|"!="|"<"|">"|"<="|">=";

```

属性条件指定は、論理式で表され、命題として属性の値の条件式を、等価演算子や比較演算子を用いて記述できるとした。

AttributeName はリソースの名前であり、リソース名やリソースタイプ名と同様に、次のように定義する。

```

AttributeName = Name;

```

イベントの定義にて、AttributeChange は属性の値の変化を、BehaviorHappen は振る舞いを表現しているとした。これらは、リソースとドット”.”でつなげることでそのリソース固有のものであることを示す。リソースの属性の値の変化と振る舞いは次のように定義した。

```

AttributeChange = AttributeName,"=",Value;
Value = /[^"\\"]+/;
BehaviorHappen = BehaviorName,"(",Arguments,")";
BehaviorName = Name;
Arguments = [{Argument,[",","]}];
Argument = /[^"\\"]*/;

```

表 2.1 標準形式トレースログの例

1	[2403010]MAIN_TASK.leaveSVC(ena_tx,ercd=0)
2	[4496099]MAIN_TASK.state=READY
3	[4496802]TASK(state==RUNNING).state=READY

属性の変化イベントは、属性名と変化後の値を代入演算子でつなぐことで記述し、振る舞いイベントは、振る舞い名に続けてカンマで区切った引数を括弧”()”で囲み記述するとした。

2.2.3 標準形式トレースログの例

前小節の定義を元に記述した、標準形式トレースログの例を表 2.1 に示す。

1 行目がリソースの振る舞いイベントであり、2 行目、3 行目が属性の値の変化イベントである。1 行目の振る舞いイベントには引数が指定されている。

1 行目、2 行目はリソースを名前で直接指定しているが、3 行目はリソースタイプと属性の条件によってリソースを特定している。

2.3 図形データ

2.3.1 座標系

図形を定義する座標系と、表示する座標系は分離して考えるとする。これにより、図形を表示環境から独立して定義することが可能になる。図形を定義する座標系をローカル座標系、表示する座標系をデバイス座標系と呼称する。

また、TLV では、高さと時間を次元に持つ、ワールド座標系という座標系を導入した。ローカル座標系で定義された図形は、はじめに、ワールド座標系における、図形を表示すべき時間の領域にマッピングされ、これを表示環境に依存するデバイス座標系にマッピングすることで表示する。これにより、図形の表示領域を、抽象度の高い時刻で指定することが可能になる。ここで、ローカル座標系からワールド座標系へのマッピングをワールド変換と呼称する。また、表示する時間の領域を表示期間と呼称する。表示期間は開始時刻と終了時刻で表される時刻のペアである。

ローカル座標系において、図形の大きさと位置を定義する際は、pixel 単位による絶対指定か、ワールド座標系へのマッピング領域に対する割合を % で指定する相対指定かの

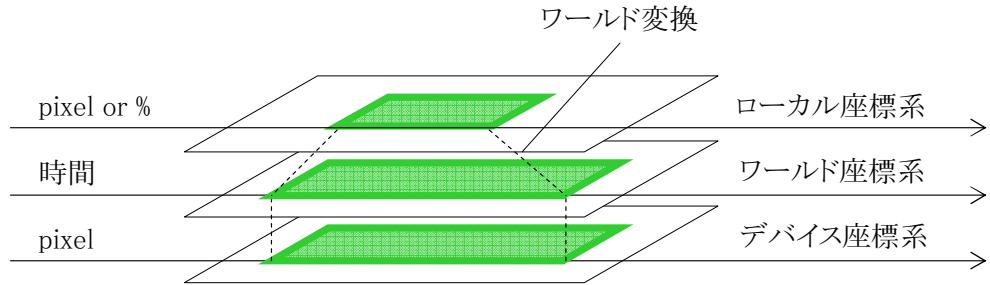


図 2.1 座標系

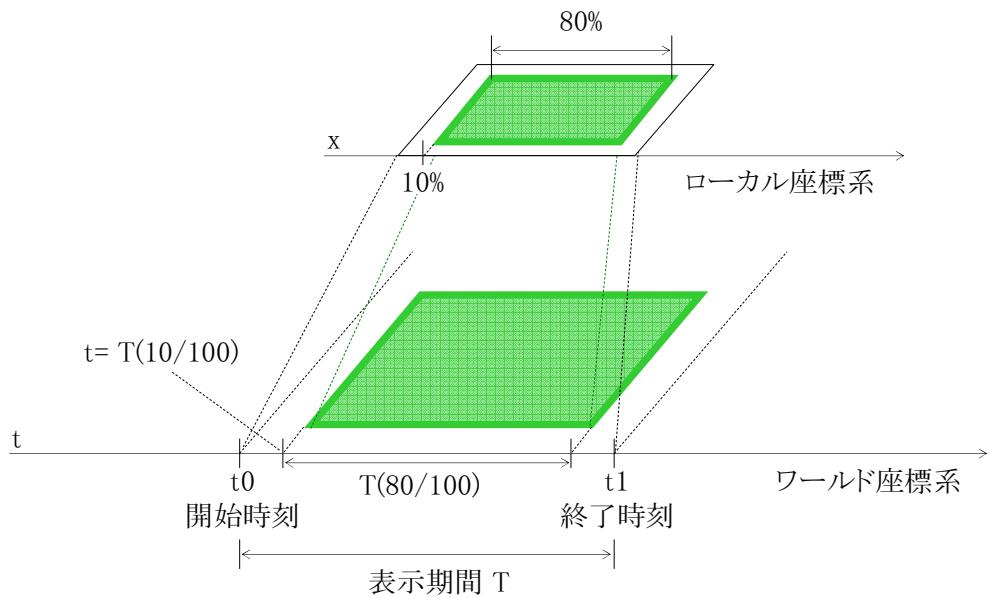


図 2.2 ワールド変換

いずれかを用いる。

図 2.1 に座標系の例を、図 2.2 にワールド変換の例を示す。

2.3.2 基本図形と図形、図形群

可視化表現は、複数の図形を組み合わせることで実現する。この際、基本となる図形の単位を基本図形と呼称する。

基本図形として扱える形状は楕円、多角形、四角形、線分、矢印、扇形、文字列の 7 種類とする。基本図形は、形状や大きさ、位置、塗りつぶしの色、線の色、線種、透明度などの属性を指定して定義する。

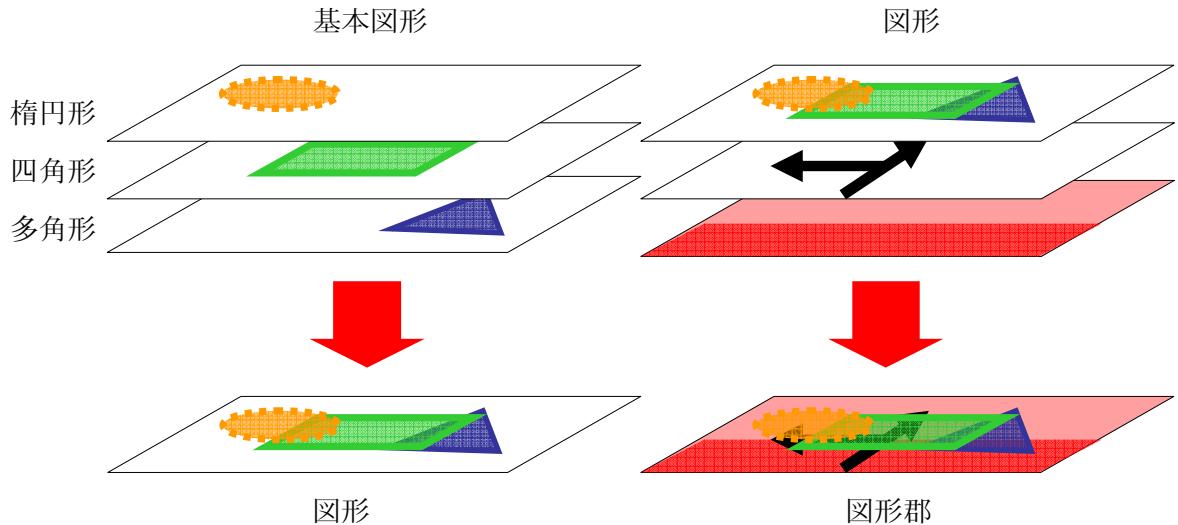


図 2.3 図形と図形群

複数の基本図形を仮想的に z 軸方向に階層的に重ねたものを、単に図形と呼称し、可視化表現の最小単位とする。図形は、構成する基本図形を順序付きで指定し、名前をつけて定義する。図形は名前を用いて参照することができ、その際に引数を与えることができる。この際、引数は、図形を構成する基本図形の、任意の属性に割り当てる想定している。

複数の図形を仮想的に z 軸方向に階層的に重ねたものを図形群と呼称する。図 2.3 に図形と図形群の例を示す。

2.4 図形データとイベントの対応

本小節では、前小節で述べた可視化表現とトレースログのイベントをどのように対応付けるのかを述べる。

2.4.1 開始イベント、終了イベント、イベント期間

前小節において、可視化表現は、図形をワールド変換を経て表示期間にマッピングすることであることを説明した。ここで、表示期間の開始時刻、終了時刻を、イベントを用いて指定するとする。つまり、指定されたイベントが発生する時刻をトレースログより抽出することにより表示期間を決定する。このようにして、トレースログのイベントと可視化表現を対応付ける。ここで、開始時刻に対応するイベントを開始イベント、終了時刻に対

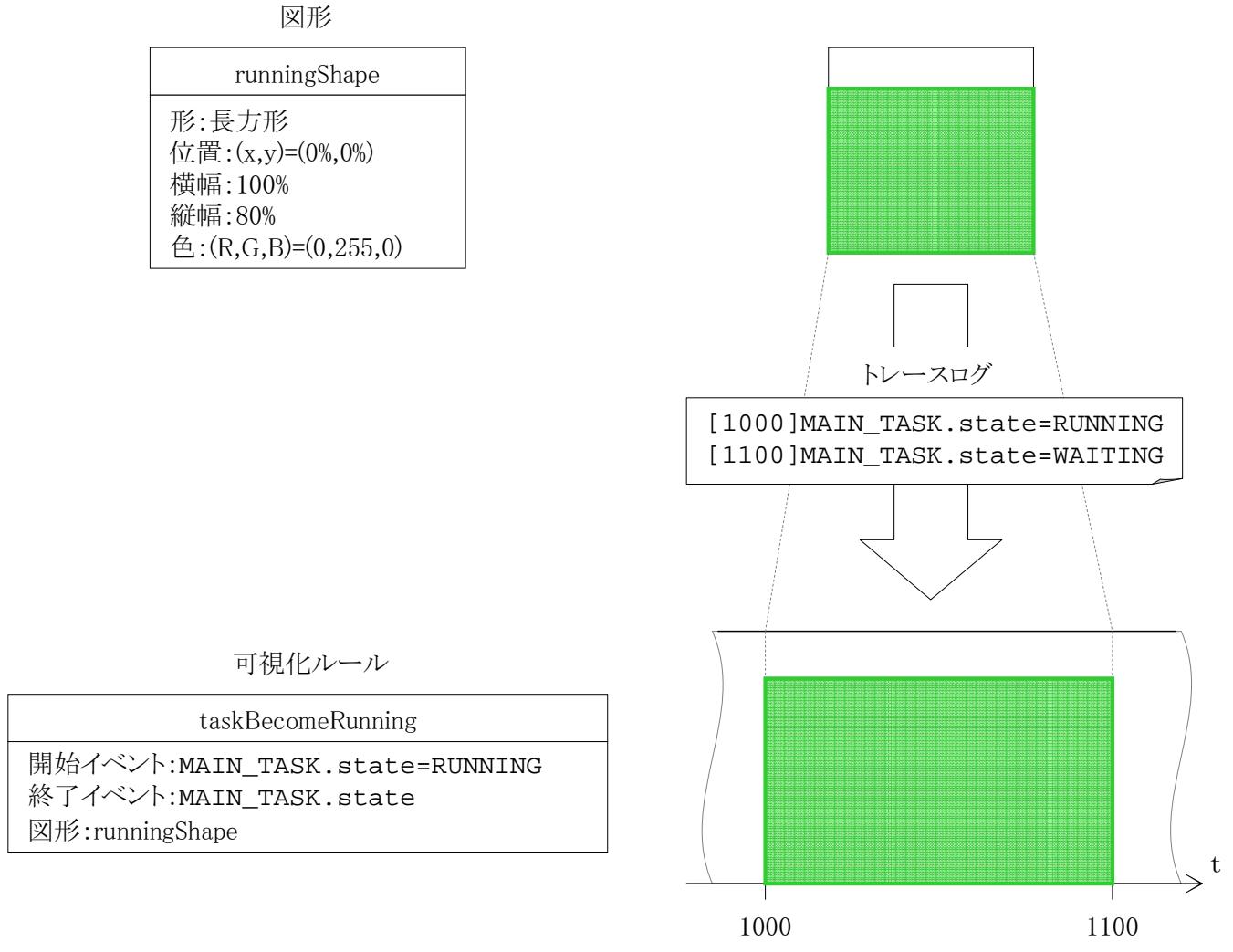


図 2.4 可視化ルール

応するイベントを終了イベントと呼称し、表示期間をイベントで表現したものをイベント期間と呼称する。

2.4.2 可視化ルール

図形群と、そのマッピング対象であるイベント期間を構成要素としてもつ構造体を、可視化ルールと呼称する。図 2.4 に、標準形式トレースログを用いてイベント期間を定義した可視化ルールの例を示す。図 2.4において、runningShape を、位置がローカル座標の原点、大きさがワールド座標系のマッピング領域に対して横幅 100%，縦幅 80% の長方形で色が緑色の図形とする。この図形を、開始イベント MAIN_TASK.state=RUNNING,

表 2.2 イベント期間を抽出するトレースログ

1	[1000]MAIN_TASK.state=RUNNING
2	[1100]MAIN_TASK.state=WAITING

終了イベント MAIN_TASK.state となるイベント期間で表示するよう定義したものが可視化ルール taskBecomeRunning である。開始イベント MAIN_TASK.state=RUNNING は、リソース MAIN_TASK の属性 state の値が RUNNING になったことを表し、終了イベント MAIN_TASK.state は、リソース MAIN_TASK の属性 state の値が単に変わったことを表している。

taskBecomeRunning を、表 2.2 に示すトレースログからイベントを抽出して表示期間の時刻を決定し、図形のワールド変換を行った結果が図 2.4 の右下に示すものである。

2.5 TraceLogVisualizer の設計

TLV の主機能は、2 つの主たるプロセスと 6 種の外部ファイルによって実現される。図 2.5 に TLV の全体像を示す。

2 つの主たるプロセスとは、標準形式への変換と、図形データの生成である。標準形式への変換は、任意の形式をもつトレースログを標準形式トレースログに変換する処理である。この処理には、外部ファイルとして変換元のトレースログファイル、リソースを定義したリソースファイル、リソースタイプを定義したリソースヘッダファイル、標準形式トレースログへの変換ルールを定義した変換ルールファイルが読み込まれる。

また、図形データの生成は、変換した標準形式トレースログに対して可視化ルールを適用し図形データを生成する処理である。この処理には外部ファイルとして可視化ルールファイルが読み込まれる。可視化ルールファイルとは図形と可視化ルールの定義を記述したファイルである。

TLV は、トレースログとリソースファイルを読み込み、トレースログの対象に対応したリソースヘッダファイル、変換ルールファイルの定義に従い、標準形式トレースログを生成する。生成された標準形式トレースログに可視化ルールファイルで定義される可視化ルールを適用し図形データを生成した後、画面に表示する。

生成された標準形式トレースログと図形データは、TLV データとしてまとめられ可視化表示の元データとして用いられる。TLV データは TLV ファイルとして外部ファイルに保存することが可能であり、TLV ファイルを読み込むことで、標準形式変換と図形デー

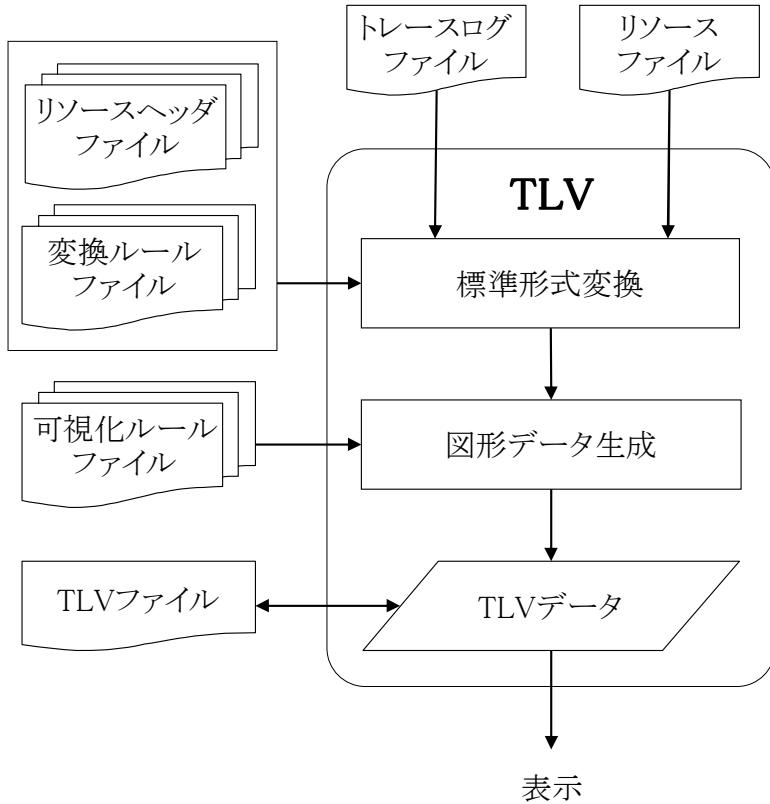


図 2.5 TLV の全体像

タ生成の処理を行わなくとも可視化表示できるようになる。

図 2.6 に、TLV のスクリーンショットを示す。

2.5.1 標準形式トレースログへの変換

標準形式トレースログへの変換は、トレースログファイルを先頭から行単位で読み込み、変換ルールファイルで定義される置換ルールに従い標準形式トレースログに置換していくことで行われる。

リソース属性の遷移に伴うイベントや、元のトレースログに欠落している情報を補うイベントなど、元のトレースログの情報だけでは判断できないイベントを出力するには、特定時刻における特定リソースの有無やその数、特定リソースの属性の値などの条件で出力を制御できる必要がある。そのため、TLV の変換ルールでは、置換する条件の指定と、条件指定の際に用いる情報を置換マクロを用いて取得できる仕組みを提供している。

標準形式トレースログに含まれるリソースは、リソースファイルで定義されていなければ



図 2.6 TLV のスクリーンショット

ばならない。リソースファイルには、各リソースについて、その名前とリソースタイプ、必要であれば各属性の初期値を定義する。また、その際に使用されるリソースタイプはリソースヘッダファイルで定義されていなければならない。リソースヘッダファイルには各リソースタイプについて、その名前と属性、振る舞いの定義を記述する。

リソースヘッダ、変換ルール、可視化ルールは可視化するターゲット毎に用意する。その際のターゲットはリソースファイルに記述する。

2.5.2 図形データの生成

標準形式変換プロセスを経て得られた標準形式トレースログは、可視化ルールを適用され図形データを生成する。ここで、図形データとは、ワールド変換が行われた全図形のデータを指す。可視化ルールは可視化ルールファイルとして与えられ、適用する可視化ルールはリソースファイルに記述する。

図形データの生成方法は、標準形式トレースログを一行ずつ可視化ルールのイベント期

間と一致するか判断し、一致した場合にその可視化ルールの表示期間をワールド変換先の領域として採用しワールド変換することで行われる。

2.6 TraceLogVisualizer のその他の機能

本節では、標準形式変換と可視化データ生成の他に TLV が備える機能について詳述する。

2.6.1 マーカー

TLV では、可視化表示部にマーカーと呼ぶ印を指定の時刻に配置することができる注目すべきイベントの発生時刻にマーカーを配置することで、可視化表示内容の理解を補助することができる。

マーカーとマーカーの間には、その間の時間が表示されるので、ソフトウェアの計測を行うことができる。マーカーには名前を付けることができ、色の指定が可能である。また、マーカーはマウス操作で選択することができ、拡大縮小などの各種操作に利用される。マーカーは階層構造で管理し、階層ごとに表示の切り替え、マーカー間時間の表示などをを行うことができる。

2.6.2 可視化表示部の制御

可視化表示ツールでは、可視化表示部を制御する操作性が使い勝手に大きく影響するため、TLV では目的や好みに合わせて様々な操作で制御を行えるようにした。TLV では、可視化表示部の制御として、表示領域の拡大縮小、移動を行うことができる。これらの操作方法として、キーボードによる操作、マウスによる操作、値の入力による操作の 3 つの方法を提供した。

マウスによる操作は、クリックによる操作、ホイールによる操作、選択による操作がある。クリックによる操作はカーソルを虫眼鏡カーソルに変更してから行う。左クリックでカーソル位置を中心に拡大、右クリックで縮小を行う。また、左ダブルクリックを行うことでクリック箇所が中心になるように移動する。ホイールによる操作は、コントロールキーを押しながらホイールすることで移動を行い、シフトキーを押しながら上へホイールすることでカーソル位置を中心に拡大、下へスクロールすることで縮小する。選択による操作では、マウス操作により領域を選択し、その領域が表示領域になるように拡大する。

キーボードによる操作は、可視化表示部において方向キーを押すことで行い、微調整す

るのに適している。左キーで表示領域を左に移動し、右キーで右に移動する。上キーでカーソル位置、または選択されたマーカーを中心に表示領域を縮小し、下キーで拡大する。

値の入力による操作では、より詳細な制御を行うことができる。可視化表示部の上部にはツールバーが用意されており、そこで表示領域の開始時刻と終了時刻を直接入力することができます。

2.6.3 マクロ表示

TLV の要求分析を行った際、可視化表示部で拡大した場合に全体の内どの領域を表示しているのかを知りたいという要求があった。そのため、TLV では、マクロビューアというウィンドウを実装した。

マクロビューアでは、トレースログに含まれるイベントの最小時刻から最大時刻までを常に可視化表示しているウィンドウで、可視化表示部で表示している時間を半透明の色で塗りつぶして表示する。塗りつぶし領域のサイズをマウスで変更することができ、それに応じて可視化表示部の表示領域を変更することができる。

マクロビューアでは可視化表示部と同じように、キーボード、マウスにより拡大縮小、移動の制御を行うことができる。

2.6.4 トレースログのテキスト表示

TLV では、標準形式トレースログをテキストで表示するウィンドウを実装した。ここでは、トレースログの内容を確認することができる。

可視化表示部とテキスト表示ウィンドウは連携しており、テキスト表示ウィンドウでマウスを移動すると、カーソル位置にあるトレースログの時刻にあわせて可視化表示部のカーソルが表示されたり、ダブルクリックすることで対応する時刻に可視化表示部を移動することができる。また、可視化表示部でダブルクリックすることで、ダブルクリック位置にある図形に対応したトレースログが、テキスト表示ウィンドウの先頭に表示されるようになっている。

2.6.5 可視化表示項目の表示非表示切り替え

TLV では、可視化表示する項目を可視化ルールにより変更、追加することができるが、それらの表示を可視化ルールやリソースを単位で切り替えることができる。これらの操作は、リソースウィンドウと可視化ルールウィンドウで行う。リソースウィンドウではリ

ソースファイルで定義されたリソースを、リソースタイプやグループ化可能な属性毎にツリービュー形式で表示しており、チェックの有無でリソース毎に表示の切り替えを行える。同じように、可視化ルールウィンドウでは、可視化ルールごとに表示の切り替えを行える。

2.7 本 OJL で追加した機能

本 OJL で追加したその他の機能について述べる。

2.7.1 スクリプト拡張機能

2.5.1 節、2.5.2 節で述べた変換・可視化ルールでは記述できない複雑な可視化を実現するためのスクリプト拡張機能を追加した。

詳しくは、4 章で述べる。

2.7.2 タイトルバーに表示中のログファイル名・リソースファイル名を表示

トレースログを開いた直後はタイトルバーに「新規トレースログ」と表示していた。しかし、どのトレースログを可視化したのが分かりにくいとよう要望があった。

そこで、タイトルバーに、開いているログファイル名とリソースファイル名を表示するように変更した。

2.7.3 リロード機能

変換ルール・可視化ルールを記述している際に、記述を変更するたびにトレースログを開き直す必要があり不便だ、という要望があった。

そこで、現在開いているトレースログを再度、変換・可視化を行なうリロードボタンを追加した。

2.7.4 ドラッグによる表示範囲移動

表示範囲の移動は、シフト + マウスホイールかスクロールバーのクリックによって行なっていた。しかし、マウスのドラッグによっても表示範囲を移動させたいという要望が

あた。

そこで、ドラッグによっても表示範囲を移動させるように変更した。

2.7.5 時刻表示のカンマ区切り

可視化ウインドウの時刻表示が、カンマ区切りになっておらず、読みづらという要望があつた。

そこで、時刻の基数が 10 の場合は、3 桁ごとにカンマで区切るように変更した。

2.7.6 マクロビューの拡大表示

長いトレースログの場合、マクロビューの表示が細かくなりすぎて、可視化範囲の選択が難しいので、改善して欲しいという要望があつた。

そこで、マクロビューでシフト + マウスホイールを行なった場合、マクロビューの表示を拡大・縮小するように変更した。

第3章

実績

3.1 開発プロセス

TLV は、OJL(On the Job Learning) の開発テーマとして開発された。OJL とは、企業で行われているソフトウェア開発プロジェクトを教材とする実践教育であり、製品レベルの実システムの開発を通じて創造的な思考力を身につけるとともに、単なる例題にとどまらない現実の開発作業を担うことにより、納期、予算といった実社会の制約を踏まえたソフトウェア開発の実際について学ぶことを目的としている。

TLV はプロジェクトベースで開発が行われ、本年度は企業出身者 1 名と教員 1 名がプロジェクトマネージャを務め、筆者含む学生 4 名が開発実務を行った。進捗の報告は、週に 1 度のミーティングと週報の提出により行った。

3.1.1 フェーズ分割

TLV の開発は複数フェーズに分割して行われている。フェーズ 1,2,3 は主に後藤ら [1] によって実施された。本報告書ではフェーズ 3 以降について述べる。

各フェーズの内容は次に述べる通りである。

フェーズ 1,2,3

2007 年 5 月～2008 年 3 月までに、一期生によって実施され、TLV の主要機能の実装が行われた [1]。

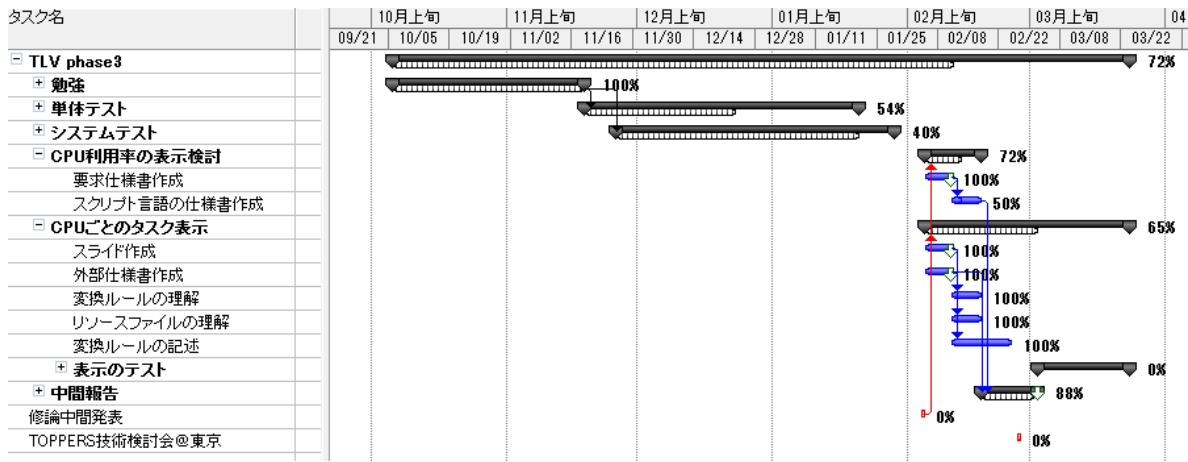


図 3.1 フェーズ 3 のスケジュール

フェーズ 3

■期間 2008 年度後期

■実施内容

- 新規に参加したため、前提となる C#,TOPPERS の学習を行なった。
- TLV の各クラスに対する単体テストと、TLV 全体に対するシステムテストとして、入力ファイルの一部を書き換え不正なファイルを大量に作成し、TLV に与えるファジングテスト [3] を行なった。
- 次フェーズで行なう予定であった複数のイベントに基づく可視化の要件を検討した。

■スケジュール 図 3.1 のように、10 月始めから 11 月中旬にかけて前提となる知識の学習を行ない、11 月中旬から 1 月末にかけて TLV に対するテストを行ない、1 月末から 2 月中旬にかけて要件の検討を行なった。

図中の“CPU ごとのタスク表示”は、筆者以外の学生が行なった内容である。

■成果 単体テストは 328 個のメソッドのうち 162 個のメソッドに対して単体テストを作成した。システムテストでは不具合を発見できなかった。機能追加と並行してテストを実施したため、仕様が安定せず効率が悪かった。

また、3 月 25 日に TOPPERS 会員に向けに 1.0rc をリリースし、要求の収集を行なつ

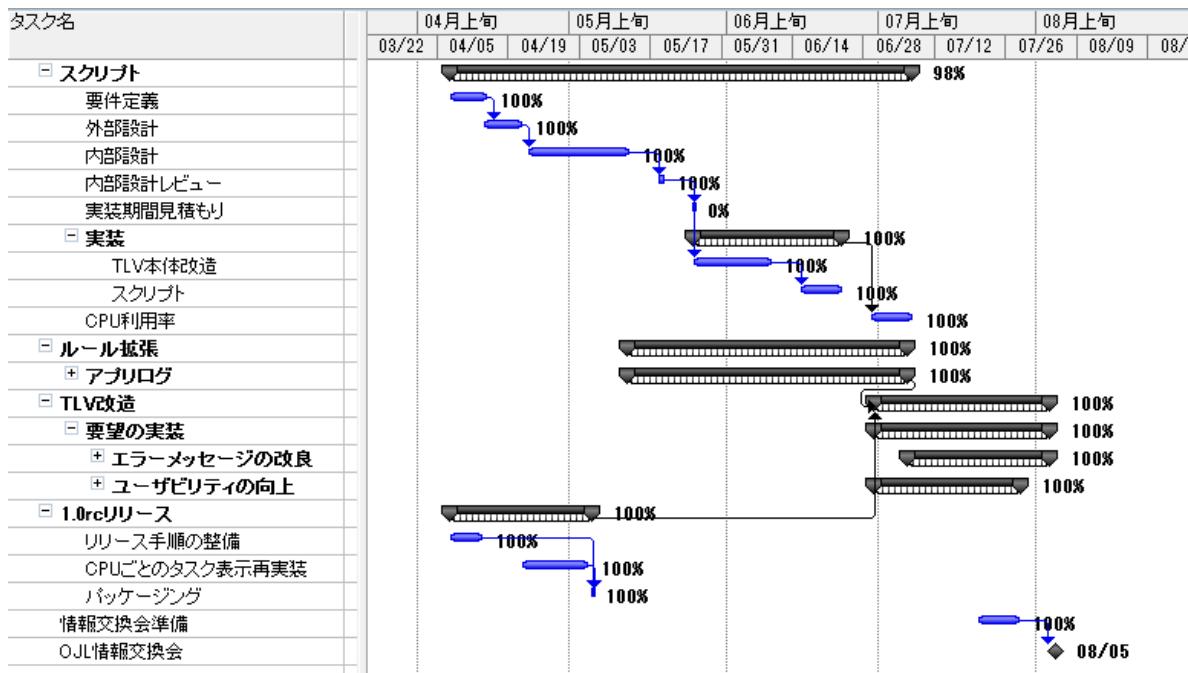


図 3.2 フェーズ 4 のスケジュール

た。1.0rc のリリースによってユーザインターフェースの改良案、追加の機能要求、可視化表現項目について要求が得られた。

フェーズ 4

■期間 2009 年度前期

■実施内容

- 4 章で述べる複雑な可視化を行なうための機能追加を行なった。
- 前フェーズで収集した要求のうち、TLV の使いやすさを向上させる要求を実装した。

■スケジュール 図 3.2 のように、4 月始めから 7 月始めにかけて、複数のイベントに基づく可視化に対応するためのスクリプト拡張を行なった。7 月始めから 8 月上旬にかけて、TLV の使いやすさを向上させる要求を実装した。

図中の“アプリログ”は、筆者以外の学生が行なった内容である。

■成果 4 章で述べる機能を追加した。

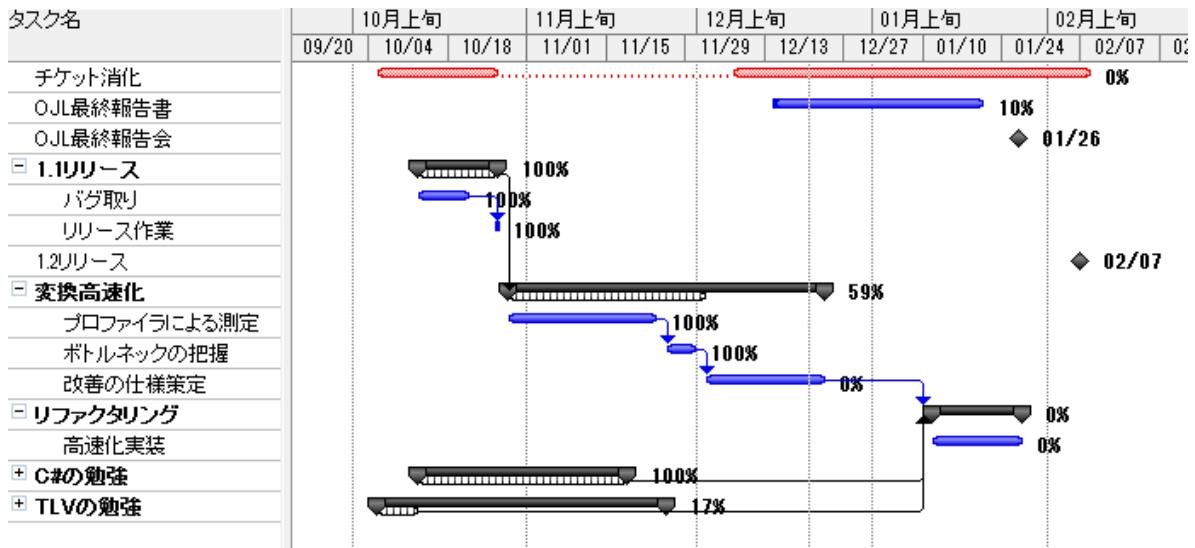


図 3.3 フェーズ 5 のスケジュール

5月13日に、TOPPERS会員に対して、フェーズ3までの内容をまとめたTLV 1.0をリリースした。

フェーズ5

■期間 2009年度後期

■実施内容

- 5で述べるように、標準形式トレースログへの変換・図形データの生成に関するソースコードの調査、およびリファクタリング方針の決定を行なった。
- 新たに三期生が参加したため、課題を与えるなどの指導を行なった。

■スケジュール 図3.3のように、10月始めから11月末にかけて、TLVの解析を行なった。12月始めから12月末にかけてリファクタリング案の策定を行なった。

■成果物 5章で述べるリファクタリング方針を決定した。また、TOPPERS会員に対してRC版をリリースし、その後一般向けにTLV1.1をリリースした。

- 10月1日にTOPPERS会員に対してTLV 1.1rc1を公開。
- rc1の不具合を修正し、10月26日にTOPPERS会員に対してTLV 1.1rc2を公開。

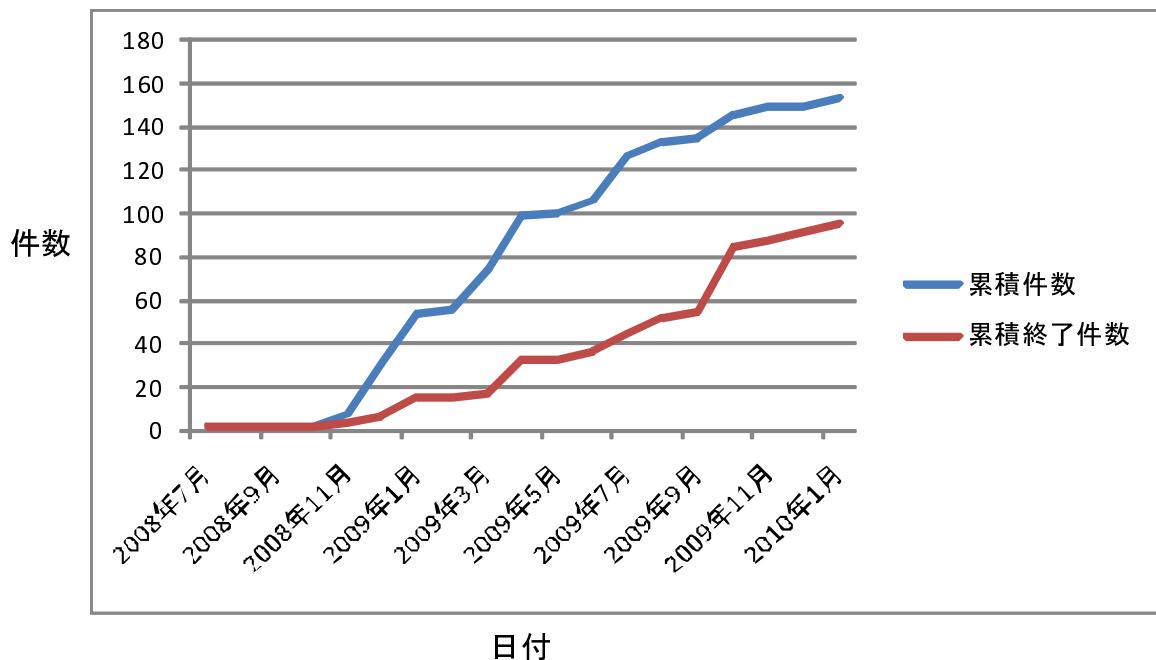


図 3.4 チケット累積件数

- 11月9日に一般向けに TLV 1.1 を公開.

3.2 チケット終了率

今期の OJL で収集した要求・不具合はすべて Trac のチケットとして管理した。フェーズ 3 で新たに追加されたチケット件数は 20 件、終了したチケットは 2 件、フェーズ 4 で新たに追加されたチケット件数は 52 件、終了したチケットは 27 件、フェーズ 5 で新たに追加されたチケット件数は 27 件、終了したチケットは 51 件であった。

チケットの累積件数・累積終了件数を図 3.4 に示す。図より、一般的な要求曲線を描いていることがわかる。

3.3 発表実績

平成 21 年度 情報処理学会 第 139 回 システム LSI 設計技術 (SLDM) 研究会において発表 [2] を行ない、情報処理学会山下記念研究賞/学生奨励賞を受賞した。

Embedded Technology 2009(ET2009) ^{*1}において、ブース出展及び TLV の一般公開についてプレス発表を行なった。

3.4 活用事例

名古屋大学大学院情報科学研究科付属組込みシステム研究センター (NCES) ^{*2}内の 7 件のプロジェクトのうち、2 件のプロジェクトによって利用されている。また、同 NCES のコンソーシアム型共同研究によても利用されている。

^{*1} <http://www.jasa.or.jp/et/>

^{*2} <http://www.nces.is.nagoya-u.ac.jp/>

第 4 章

スクリプト拡張機能

4.1 目的

変換ルール、可視化ルールの機能は限定的であるため、複数のイベントに基づく可視化を行なえなかった。例えば、ある時間に起動しているタスクの数を数える必要のあるCPU 使用率の可視化を行なえない。

しかし、先行リリースなどによって収集した要求のなかで、こういった複数のイベントに基づく可視化を行ないたいという要望が強かった。そのため、複数のイベントに基づく可視化の実現するための機能追加を行なう。

4.2 設計

4.2.1 変換・可視化の実現

複数のイベントに基づく可視化の実現する方法について、次の 3 つの方針を検討、比較した。

1 つ目は、変換ルールと可視化ルールを拡張し、複数のトレースログに基づく可視化を記述できるようにする方針である。既存のルールを拡張するので、すでにルールの記述法を習得している利用者の学習コストが低い。しかし、可視化ルールが複雑化しているため、可読性と後方互換性を保ちつつ、拡張を加えるのは困難である。またルールの処理に関するソースコードが複雑化しているため、変更を加えるのが難しい。

2 つ目は、標準トレースログ間の変換を記述するための言語を実装し、変換処理と可視化処理の間に標準形式トレースログを変形する処理を追加する方針である。ルールの複雑化や互換性の維持のためのコストを避けることができる。また、既存の処理と独立して実

装できるため、ソースコードの複雑さを避けられる。しかし、利用者は新たな言語の学習が必要となる。また、新たな言語を設計する必要があるため、実装コストも高い。

3つ目は、外部のプロセスによって標準形式トレースログへの変換と図形データの生成を行なえるように TLV を拡張する方針である。変換処理は外部プロセスで行なうため、任意のプログラミング言語で記述できる。利用者が自分の使えるプログラミング言語で変換を記述できるため、学習コストもそれほど高くない。また、既存のプログラミング言語を利用できるため、言語設計のコストを避けられる。既存の処理と独立して実装できるため、ソースコードの複雑さを避けられる。

上記の点を考慮し、利用者の学習コストと実装コストを低くするために、3つめの外部プロセスによる変換の実現を選択した。

4.2.2 変換ルール・可視化ルールの拡張

変換・可視化処理を行なう外部プロセスを指定する方法について、次の2つの方針を検討、比較した。

1つ目は、外部プロセスを指定するためのルールファイルを追加する方針である。既存の変換ルール・可視化ルールとの互換性などを考慮する必要がないが、利用者は新たに学習する必要がある。また、新たなルールファイルを追加するため実装・保守コストが高い。さらに、旧ルールファイルと新ルールファイルが衝突した際の処理も必要となる。

2つ目は、既存の変換ルール・可視化ルールの拡張し、変換を行なう外部プロセスを指定できるようにする方針である。すでにルールの記述法を習得している利用者の学習コストが低いが、既存のルールとの互換性などを考慮する必要がある。ただし、既存のルールはそれほど複雑ではないため、互換性を保つためのコストはそれほど高くない。また、既存の処理を拡張することで実装できるため、実装コストが低い。

上記の点を考慮し、2つめの既存のルールの拡張を選択した。

4.3 実装

4.3.1 TLV ファイルの拡張

フェーズ3直後の TLV は、図 4.1 のように、中間形式である TLV ファイルは、図形データを保持していない。TLV ファイルを開く際に、TLV ファイル内の標準形式トレースログに可視化ルールを適用し、図形データを生成した後、描画を行なっていた。

この TLV ファイルのまま、図形データの生成を外部プロセスで行なうようにすると、

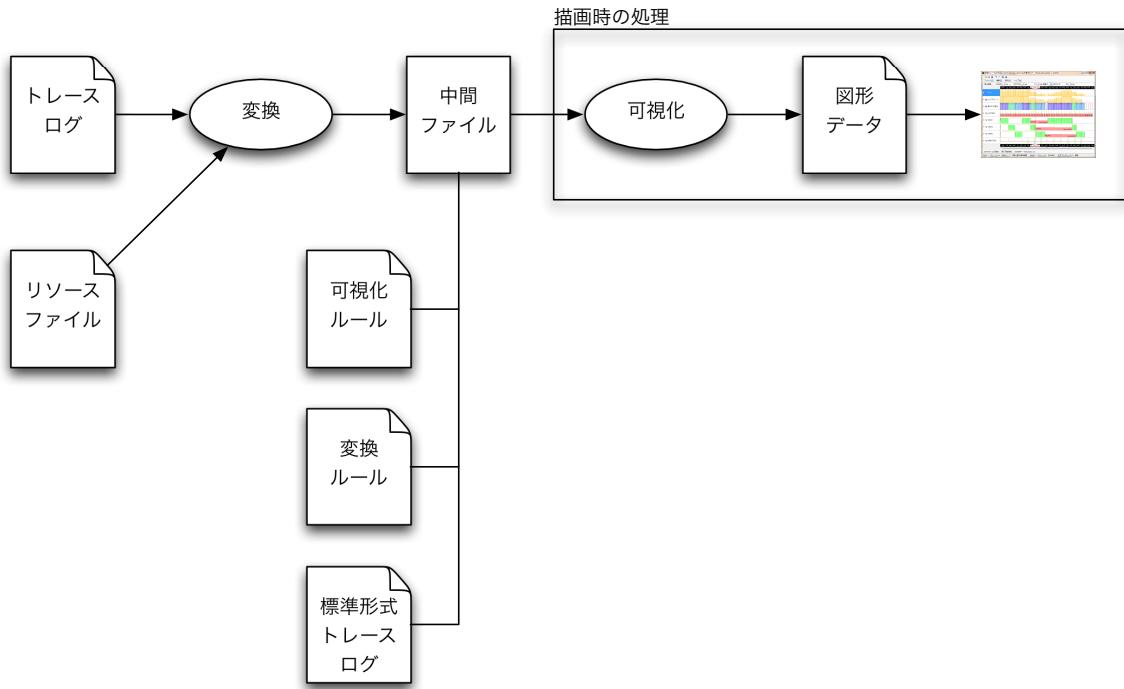


図 4.1 変換機能拡張前の TLV の変換処理

TLV ファイルを開く際に外部プロセスを実行する必要がある。しかし、外部プロセスを実行する環境は、全ての計算機上には用意されていないため、TLV ファイルの可搬性が失なわれる。また図形データ生成の結果は常に同一であるにもかかわらず、毎回ファイルを開く際に図形データ生成が行なわれるため効率が悪い。

そこで、TLV を図 4.2 のように変更し、TLV ファイルに図形データも保持し、ファイルを開く際はその図形データを描画するようにする。

4.3.2 変換ルールの拡張

変換ルールファイルを変換に用いる外部スクリプトを指定できるように拡張する。

変換に用いる外部スクリプトを指定するために、変換ルールファイルには表 4.1 の要素を用いる。図 4.3 のように、`arguments` を用いて外部スクリプトを指定する。あるいは、図 4.4 のように `script` を用いて変換ルールファイル内にスクリプトを直接記述すること可能である。

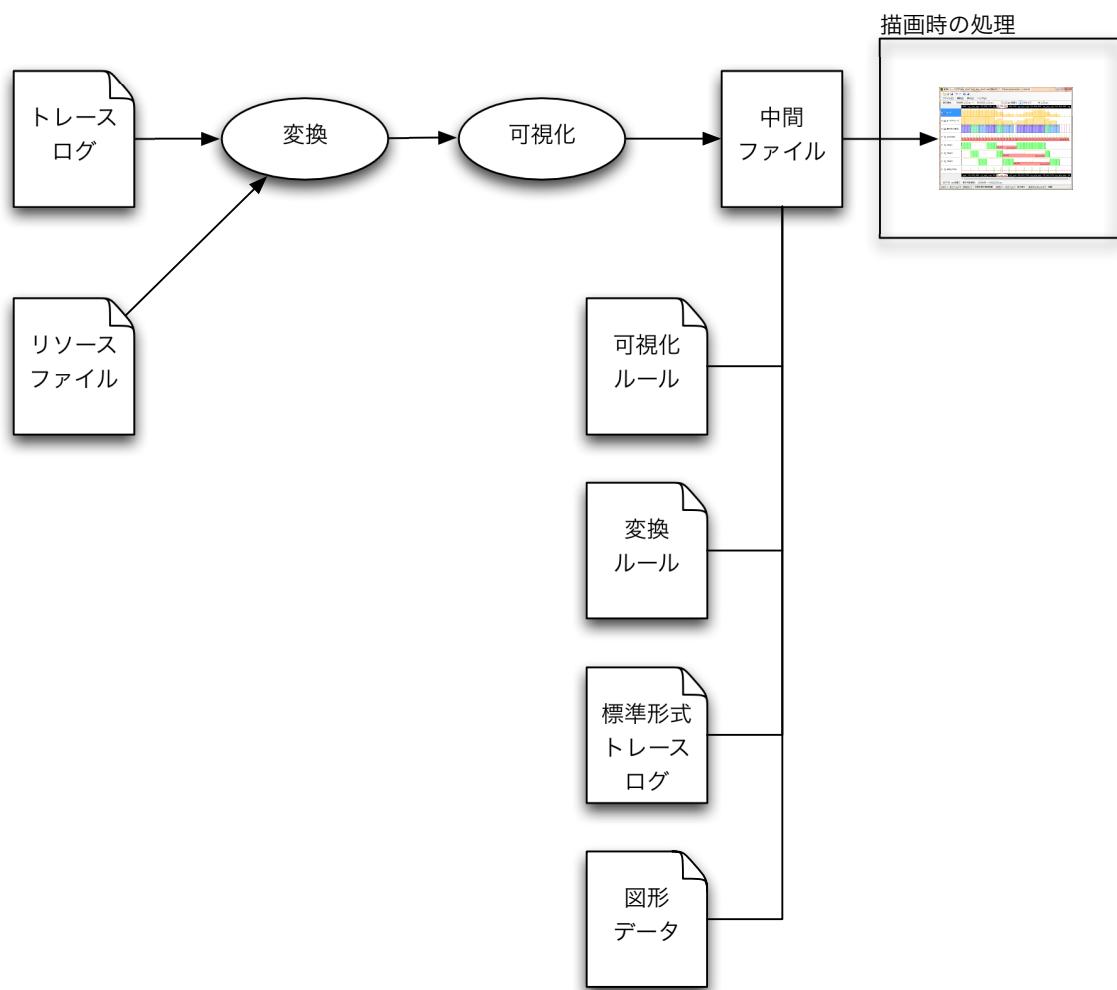


図 4.2 変換機能拡張後の TLV の変換処理

```

1  {
2      "asp2": {
3          "$STYLE": "script",
4          "fileName": "c:/cygwin/bin/ruby",
5          "arguments": "conv.rb",
6      }
7  }

```

図 4.3 外部スクリプトを指定する変換ルールの例

```

1  {
2      "asp2": {
3          "$STYLE": "script",
4          "fileName": "c:/cygwin/bin/ruby",
5          "arguments": "{0}",
6          "script" : "puts '[1]TASK1.state=RUNNING'"
7      }
8  }

```

図 4.4 直接記述する変換ルールの例

4.3.3 可視化ルールの拡張

可視化に用いる外部スクリプトを指定するために、可視化ルールファイルに図 4.2 の要素を用いる。図 4.6 のように、`arguments` を用いて外部スクリプトを指定する。あるいは、図 4.7 のように `script` を用いて可視化ルールファイル内にスクリプトを直接記述することも可能である。

表 4.1 変換ルールに追加された要素

要素	内容
\$STYLE	旧ルールと区別するための要素。常に <code>script</code> と記述する
fileName	スクリプトを実行する処理系
arguments	実行時に渡される引数。 <code>{0}</code> は一時ファイル名に置き換えられる。
script	変換スクリプトの内容

表 4.2 可視化ルールに追加された要素

要素	内容
Style	旧ルールと区別するための要素。常に <code>script</code> と記述する
FileName	スクリプトを実行する処理系
Arguments	実行時に渡される引数。 <code>{0}</code> は一時ファイル名に置き換えられる。
Script	一時ファイルの内容

```

1 [
2 {
3   "Type": "Rectangle",
4   "Size": "100,80",
5   "Pen": {"Color": "ff00ff00", "Width": 1},
6   "Fill": "6600ff00"
7 },
8 {
9   "Type": "Rectangle",
10  "Size": "100,80",
11  "Pen": {"Color": "ff00ff00", "Width": 1},
12  "Fill": "6600ff00"
13 }
14 ]

```

図 4.5 図形データの例

```

1 {
2   "asp2": {
3     "VisualizeRules": {
4       "taskStateChange": {
5         "Style": "script",
6         "DisplayName": "状態遷移",
7         "Target": "Task",
8         "FileName": "c:/cygwin/bin/ruby",
9         "Arguments": "viz.rb",
10        }
11      }
12    }
13 }

```

図 4.6 外部ファイルを指定する可視化ルールの例

4.3.4 外部プロセスによる変換・図形データ生成の実現

変換・可視化処理を図 4.8 のように拡張し、外部プロセスによって変換・可視化を行うようとする。

変換ルールで外部プロセスが指定されている場合、指定された外部プロセスを生成する。外部プロセスの標準入力に対して、リソースファイルとトレースログを出力する。外

```

1  {
2      "asp2":{
3          "VisualizeRules":{
4              "taskStateChange":{
5                  "Style": "script",
6                  "DisplayName 状態遷移":"",
7                  "Target":"Task",
8                  "FileName": "c:/cygwin/bin/ruby",
9                  "Arguments": "{0}",
10                 "Script" : "puts '{ \"Type\":\"Rectangle\", ... }'"
11             }
12         }
13     }
14 }
```

図 4.7 直接記述する可視化ルールの例

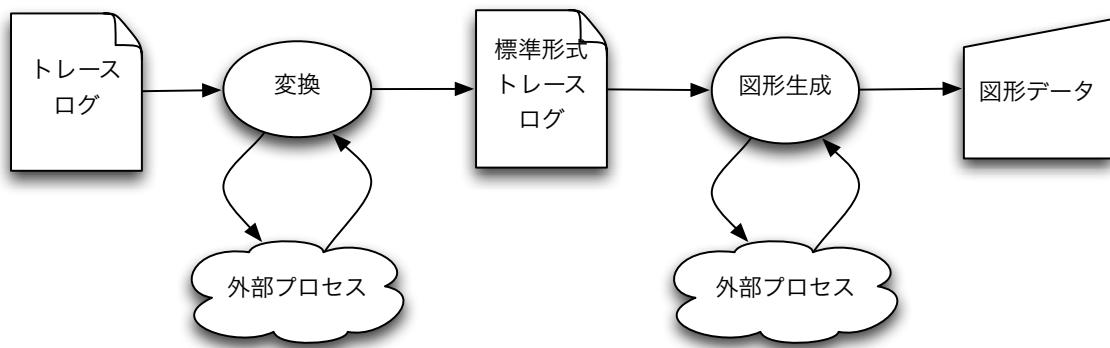


図 4.8 外部プロセスによる変換・図形データの生成

部プロセスの標準出力から標準形式トレースログを受け取り、変換処理を完了する。

可視化ルールで外部プロセスが指定されている場合、指定された外部プロセスを生成する。外部プロセスの標準入力に対して、リソースファイルと標準形式トレースログを出力する。外部プロセスの標準出力から図形データを受け取り、図形データの生成を完了する。

外部ルールが指定されていない場合は、従来のルールと同様の処理を行なう。

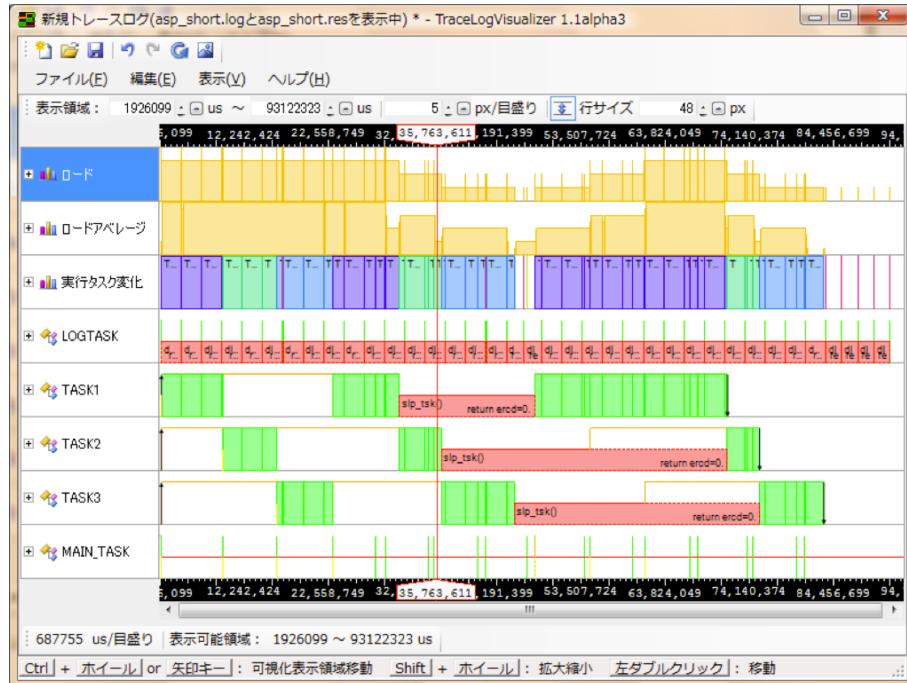


図 4.9 CPU ロード・ロードアベレージの可視化表示の例

4.4 例: CPU 利用率可視化表示

スクリプト拡張によって記述した可視化の例として、CPU ロード・ロードアベレージの可視化を図 4.9 に示す。ある時間において実行可能・実行中状態のタスクの数を CPU ロード、CPU ロードの平均をロードアベレージとし可視化を行なった。外部スクリプトとしては Ruby を利用し 269 行であった。

スクリプト拡張による可視化と従来のルールファイルによる可視化は混在可能なので、実行タスク変化などの可視化は従来の可視化ルールによって行なっている。

第5章

リファクタリング

5.1 目的

長時間動作するプログラムのトレースログは膨大な量となるため、トレースログ解析を高速化する必要がある。しかし、標準形式トレースログへの変換及び図形データの生成に関するソースコードが複雑化しているため、高速化のための変更を加えることが難しい。

そこで、この問題を改善するために、関連するソースコードを対象に、複雑化させている原因を調査する。その後、その原因を対象するためのリファクタリング方針の決定を行なう。

5.2 対象

調査対象となったクラスは6個であり、合計行数は1590行であった。

5.3 発見した問題点と改善方法

クラスを調査した結果、各クラスの単一の責務を持ち、過度に結合しているクラスも存在しなかった。そのため、クラス設計について大きな問題はない。

しかし、多くのクラスにおいて、複雑な処理を行なうメソッドが存在しており、これがソースコードを複雑化させている原因である。

そこで、メソッドの整理するための共通機能のくくりだしやクラス分割をリファクタリングの目的とした。

5.3.1 標準形式トレースログのパーサングにおける正規表現の多用

標準形式トレースログのパーサングにおいて、図 5.1 のように複雑な正規表現を多用されていた。複雑な正規表現が用いられているため、可読性が低く、保守も難しい。

これを改善するためは、正規表現ではなく文脈自由文法を扱えるパーサを採用する必要がある。パーサの実現方法について、次の 2 つの方針を比較・検討した。

1 つ目は、Yacc などのコンパイラ・コンパイラを用いて、パーサを生成する方針である。高速なパーサを生成でき、またコンパイラ・コンパイラの種類によっては LALR 文法という強力な文法を利用可能である。構文定義を更新するたびにパーサを再生成させる必要がある。また、デバッグには、そのコンパイラ・コンパイラ固有の知識が要求される。

2 つ目は、Parsec[18] などに代表されるパーサ・コンビネータを実装し、その上でパーサを記述する方針である。構文定義を更新するたびにパーサを再生成する必要がなく、デバッグも通常のプログラムとは同様に行なえる。しかし、利用可能な文法が LL(k) 文法なので左再帰の除去などが必要となる。また、パーサ・コンビネータを実装する必要があるため実装コストが高い。

両手法で利用可能な文法のクラスは包含関係にないため、表現能力を単純に比較することはできない。トレースログのパーサングは比較的単純であるため、どちらを用いても問題ない。

パーサを再生成する必要がない点と、デバッグの容易さなどから、2 つめのパーサ・コンビネータを用いる方針を採用した。

5.3.2 可視化ルールにおける暗黙的な木構造

可視化ルールは図 5.2 のように、複数の `Figures` の集約によって表現される。`Figures` には `Shape` と `Figures` の二種類が存在し、`Figures` は複数の `Shape` によって構成される。また `Figures` の `Condition` フィールドに、その図形を用いる条件式が格納されている。条件式には、等号、不等号、論理和、論理積などがある。

`Figures` クラスと `Condition` フィールドは木構造を持つデータであるが、木構造によって表現されていない。そのため、木構造をトラバースするためのコードが多数の場所に存在し、保守性と可読性が低下している。

木構造によって表現するために Composite パターン [19] を適用し、図 5.3 のようにする。`Figures` クラスを `Figure` インタフェース、枝である `Figures` クラス、葉である

```

1 ...
2 m = Regex.Match(_log, @"^(?<time>[0-9a-zA-Z]+(\.[0-9a-zA-Z]*?)?)$");
3 if (m.Success)
4     Time = m.Groups["time"].Value;
5 HasTime = m.Success;
6
7 m = Regex.Match(_log, @"^([^\n]+)?(<object>[^\\n](\\n).+(([^\\n]+\\n))?)($|(.[^\\s]+))");
8 if (m.Success)
9     Object = m.Groups["object"].Value;
10
11 m = Regex.Match(_log, @"^([^\n]+)?(<objectName>[^\\n](\\n).+(([^\\n]+\\n))?)($|(.[^\\s]+))");
12 if (m.Success)
13     ObjectName = m.Groups["objectName"].Value;
14 HasObjectName = m.Success;
15
16 m = Regex.Match(_log, @"^([^\n]+)?(<objectType>[^\\n](\\n).+(([^\\n]+\\n))?)($|(.[^\\s]+))");
17 if (m.Success)
18     ObjectType = m.Groups["objectType"].Value;
19 HasObjectType = m.Success;
20 ...

```

図 5.1 正規表現を多用した例

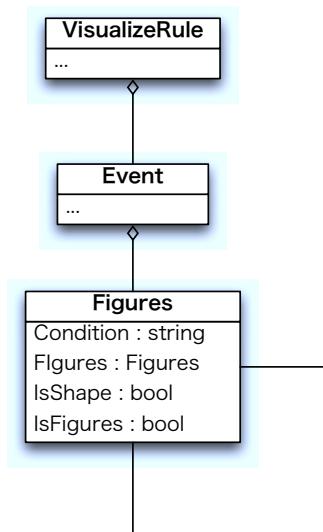


図 5.2 現状の可視化ルール

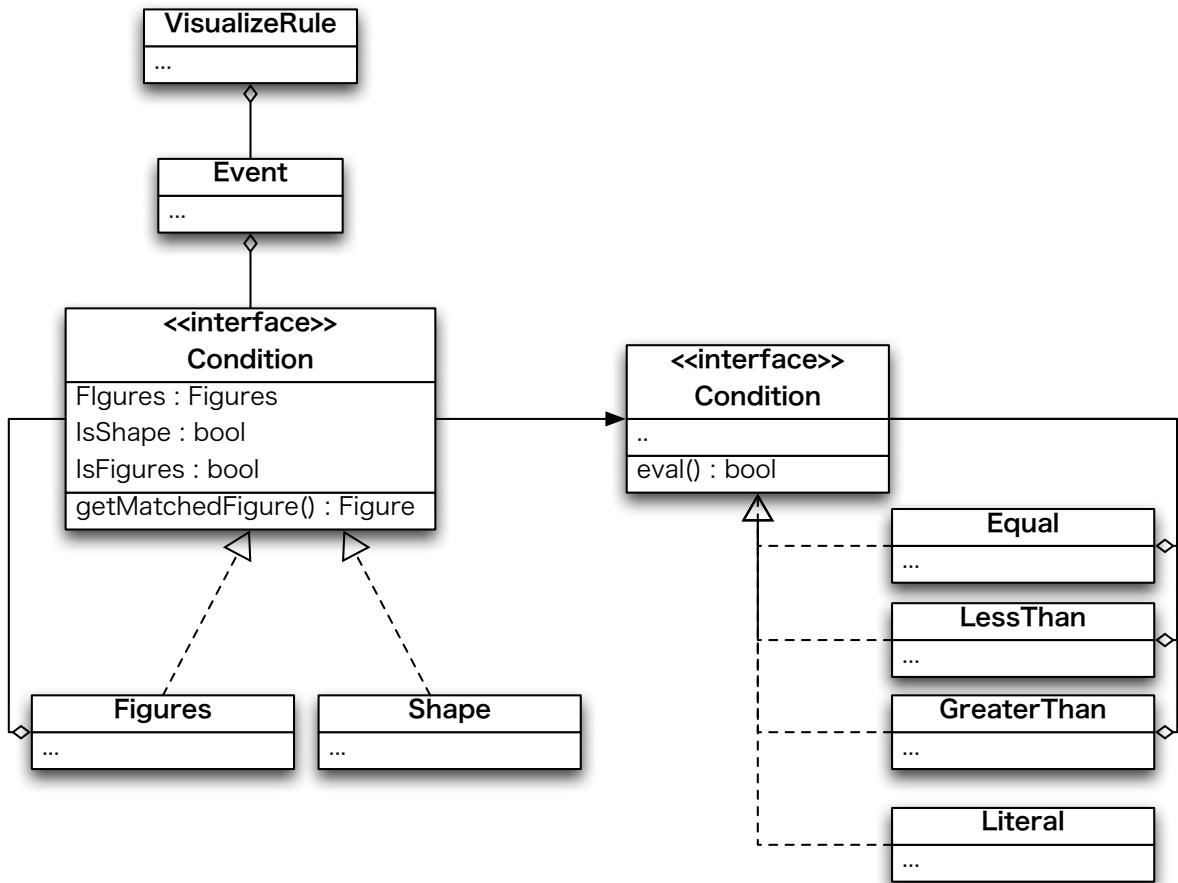


図 5.3 Composite パターンの適用

`Shape` クラスに分解する。また、`Figures` クラスに関連した処理を `Figures` クラスと `Shape` クラスに移動させる。同様に、`Condition` フィールドを `Condition` インタフェースと派生クラスに分解する。

しかし、Composite パターンを用いると、処理を追加する際の変更範囲が大きくなってしまう。例えば、`Condition` インタフェースに関する処理を追加する場合は、`Equal` クラス、`LessThan` クラス、`GreaterThan` クラス、`Literal` クラスを全て変更する必要がある。

そこで、図 5.4 のように Visitor パターンを適用し、処理を外部のクラスへと分離する。これにより、処理の追加が容易になる。

Visitor パターンを用いた場合、構造を変更する際の修正範囲が広くなる。しかし、可視化ルールの構造はこれまで変更が行なわれておらず、また今後も予定はない。そのため問題ないと判断した。

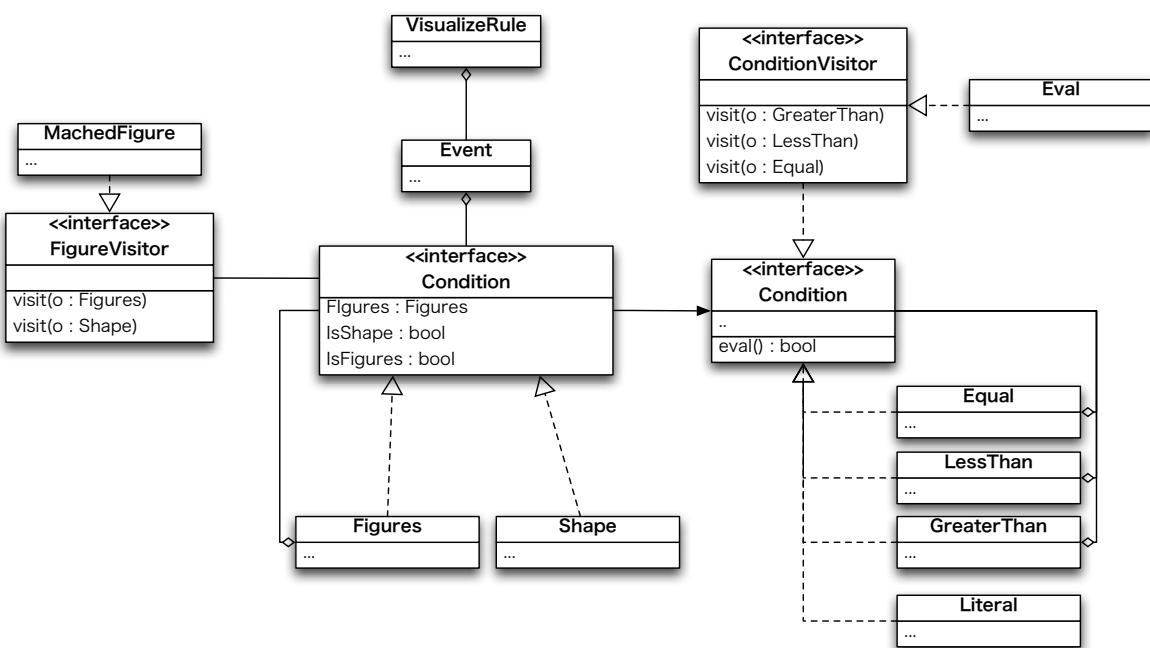


図 5.4 Visitor パターンの適用

第 6 章

おわりに

6.1 関連研究

組込みシステム向けデバッガソフトウェア

組込みシステム向けデバッガソフトウェアには、機能の1つとしてトレースログを可視化する機能が含まれている場合がある。組込みシステム向けデバッガソフトウェアとは、ICE (In-Circuit Emulator) や JTAG エミュレータなどの、組込みシステム向けデバッガに付属するデバッグ用のソフトウェアである。組込みシステム向けデバッガとは、ターゲットシステム上で動くプログラムをホストシステム上でデバッグを行えるようにするために、ターゲット CPU にアクセスする手段を提供する装置を指す。

組込みシステム向けデバッガソフトウェアとしては、京都マイクロコンピュータ株式会社の PARTNER[4] や、株式会社ソフィアシステムズの WatchPoint[5] などがあり、それぞれ、イベントトラッカー、OS アナライザというトレースログを可視化する機能を提供している。図 6.1 に、イベントトラッカーのスクリーンショットを、図 6.2 に OS アナライザのスクリーンショットを示す。

これら組込みシステム向けデバッガソフトウェアの一機能としての可視化ツールは、その性質からターゲットとなる OS やプロセッサが限定される。これは、組込みシステム向けデバッガは、通常、ターゲットとなるプロセッサが限られており、デバッガソフトウェアが対応する OS も限られているからである。また、可視化表示したい情報も提供されているものに限られるなど、可視化ツールとしては汎用性、拡張性に乏しい。

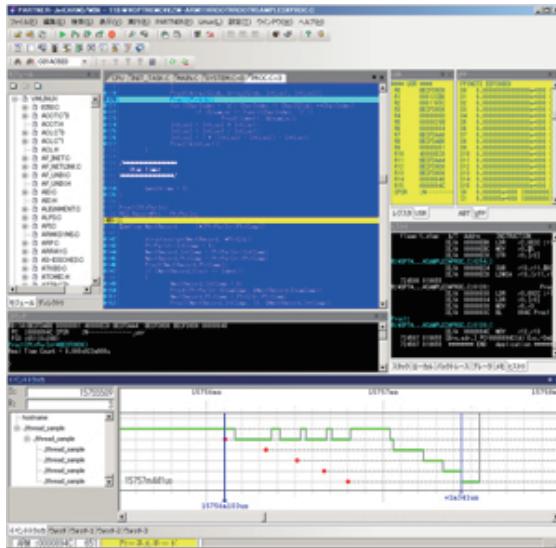


図 6.1 PARTNER イベントトラッカー

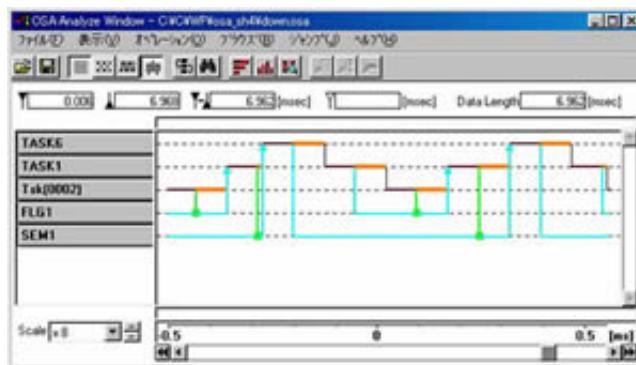


図 6.2 WatchPoint OS アナライザ

組込み OS 向けの統合開発環境

QNX Software Systems 社は、自社の組込みリアルタイムオペレーティングシステム QNX の統合開発環境として QNX Momentics を販売している。QNX Momentics にはシステムプロファイラとして、システムコールや割込み、スレッド状態やメッセージなどを可視化する QNX System Profiler[6] という機能を提供している。図 6.3 に QNX System Profiler のスクリーンショットを示す。

また、イーソル株式会社は T-Kernel/μITRON ベースシステムの統合開発環境として

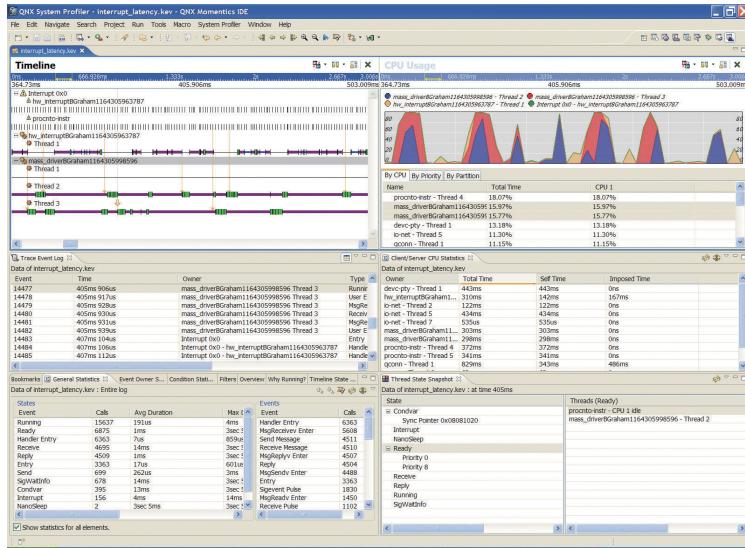


図 6.3 QNXSystemProfiler

eBinder[7] を販売している。eBinder にはイベントログ取得・解析ツールとして EvenTrek が付属しており、システムコール、割込み、タスクスイッチ、タスク状態遷移などを可視化することができる。図 6.4 に EvenTrek のスクリーンショットを示す。

このように、商用の組込み OS 向けの統合開発環境には OS の実行履歴を可視化表示する機能が搭載されている場合がある。しかしながら、これらは、各ベンダーが自社 OS の競争力を高めるために提供しているものであり、当然ながら可視化表示に対応する OS は自社提供のものに限られている。また、可視化表示する情報も提供するものに限られており、表示のカスタマイズ機能もそれほど自由度は高くはない。

Unix 系 OS のトレースログプロファイラ

Unix 系 OS では、これまでに、パフォーマンスチューニングや障害解析を目的として、カーネルの実行トレースを取得するソフトウェアがいくつか開発されている。ここでは、単にこれをトレースツールと呼称する。

Linux 用のトレースツールとしては LKST[8], SystemTap[9], LTTng[10] などがあり、Solaris 用には Dtrace[11] がある。これらトレースツールが提供する主な機能は、カーネル内にフックを仕込みカーネル内部状態をユーザー空間に通知する機能と通知をログとして記録する機能である。

これらのトレースツールは、ログを分析、提示する、専用のプロファイラツールを提供



図 6.4 eBinder EvenTrek

している場合が多い。たとえば、LTTng には LTTV[12] が、DTrace には Chime[13] が、プロファイラツールとして提供されている。図 6.5 に LTTV のスクリーンショットを、図 6.6 に Chime のスクリーンショットを示す。

これら、プロファイラツールは、主に、カーネルの内部状態を統計情報として出力することにより、ボトルネックを探したり、障害の要因を探る目的で使用されるが、ソフトウェアのデバッグを目的に使用することもできる。DTrace などはログ出力のためのカーネルフックポイントを独自のスクリプト言語を用いて制御できるなど、任意の情報をソフトウェアの実行から取得することができる。しかしながら、取得したログを任意の図形で可視化する手段は提供されておらず、テキスト形式での確認となる。また、可視化表示する際のログの形式は、その OS のトレースツールが出力する形式に依存するため、他の OS のトレースを可視化することはできない。

波形表示ツールの流用

任意の OS、アプリケーションのトレースログを可視化表示する手段として、波形表示ツールを流用する方法がある。波形表示ツールとは、Verilog 等のデジタル回路設計用論理シミュレータの実行ログを波形で表示するソフトウェアのことを指す。

デジタル回路設計用論理シミュレータの実行ログには、VCD(Value Change Dump) 形式というオープンなファイルフォーマットが存在する。そのため、任意のログを VCD

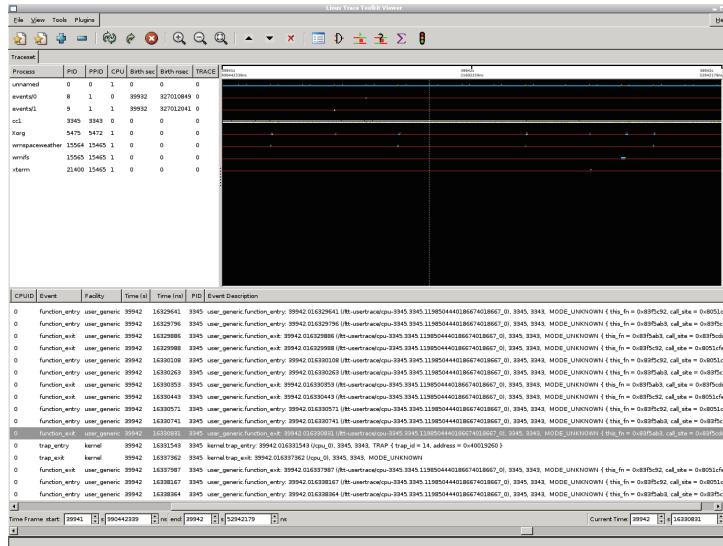


図 6.5 LTTV

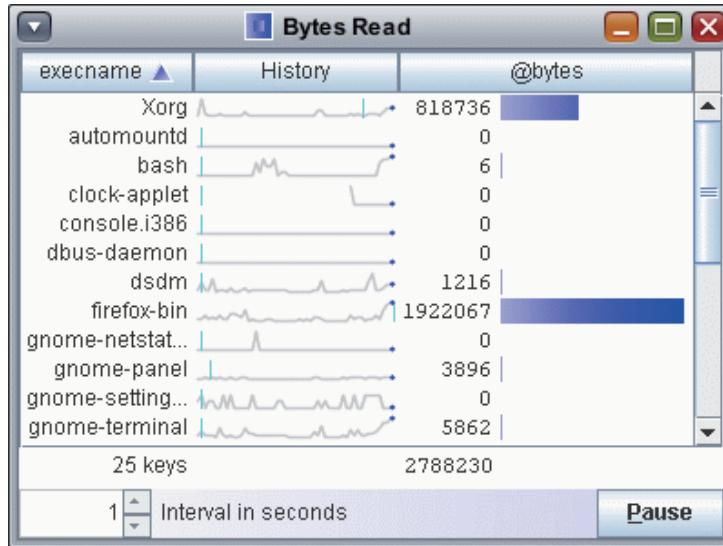


図 6.6 Chime

形式として出力することにより、これらのツールで可視化表示することが可能になる。図 6.7 に、VCD 形式のログの可視化に対応した波形表示ツール GTKWave のスクリーンショットを示す。

波形表示ツールを流用する方法では、任意のログをオープンフォーマットなファイル形式に変換することによりログの形式に依存せずに利用できる反面、表示能力に乏しく、複

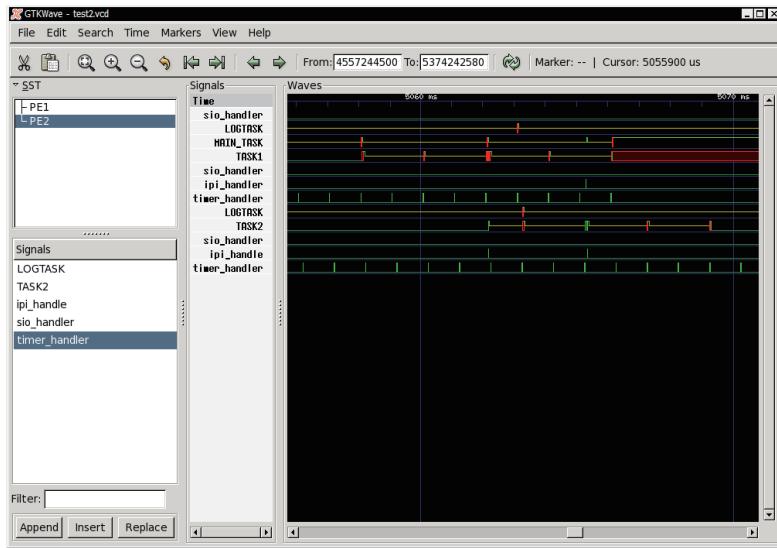


図 6.7 GTKWave

雑な可視化表現は難しいという問題がある。

6.2 まとめ

本OJLでは、トレースログ可視化ツールであるTLVの開発を継続して行ない、TLVに対して機能追加とリファクタリングを行なった。

TLVを開発された背景として、組込みシステムにおいてもマルチコアプロセッサの利用が進んでおり、それに伴い従来のデバッグ方法が有効でなくなってきたことを述べた。これは、マルチコアプロセッサが各コアで並列処理を行うため、プログラムの挙動が非決定的になり、バグの再現が保証されず、従来のブレークポイントによるステップ実行ではバグを確実に捕らえることができないからである。

一方、マルチコアプロセッサ環境におけるデバッグで有効な方法として、実行中にデバッグを行うのではなく、実行後にトレースログを解析する手法がある。そして、開発者が直接トレースログを扱うのは効率が悪く、トレースログの解析を支援するツールが要求されており、その1つとして可視化表示ツールがある。

既存のトレースログ可視化ツールは、標準化されたトレースログを扱っていないため、利用できるトレースログの形式が限られており、汎用性に乏しい。また、可視化表示項目が提供されているものに限られ、変更や追加を行う仕組みも提供されておらず、拡張性に乏しい。

そこで後藤ら [1, 2] によって、汎用性と拡張性を備えたトレースログ可視化ツール TLV が開発された。TLV 内部でトレースログを抽象的に扱えるよう、トレースログを一般化した標準形式トレースログを定め、任意の形式のトレースログを標準形式トレースログに変換する仕組みを変換ルールとして形式化した。トレースログの可視化表現を指示する仕組みを抽象化し、可視化ルールとして形式化した。TLV では、変換ルールと可視化ルールを外部ファイルとして与えることで、汎用性と拡張性を実現している。

本 OJL では、TLV のリリースを複数回行ない、要求の収集を行なった。収集した要求のうち “CPU 利用率表示などの複雑な可視化の実現” と “TLV の高速化” に対する要求が強かつたため、これらの実現を行なった。

複雑な可視化を行なうための機能追加を行なった。変換・可視化を外部プロセスで行なえるようすることで、任意の言語で変換ルール・可視化ルールを記述できるようにした。変換ルール・可視化ルールの互換性を保つように機能追加を行なったため、従来のルールと混在させて利用することも可能である。実際に CPU 利用率の可視化を行ない、有用性を確認した。

TLV の高速化を行なうためには、標準形式トレースログへの変換と図形データの生成を高速化する必要がある。その際、変換処理と図形データに関するソースコードが複雑化している点が障害となる。そこで、複雑化している原因の調査を行ない、その原因を改善するリファクタリング方針の決定を行なった。

6.3 今後の課題

変換・可視化の高速化

TLV の変換・可視化の高速化を行なう必要がある。

5 章で述べたリファクタリング方針に従い、リファクタリングの実施を行なう必要がある。その後、プロファイラ等を用いて、ボトルネックの特定を行ない、ボトルネックを改善し、変換・可視化の高速化を行なう必要がある。

TLV の応答性の向上

TLV の応答性を向上させる必要がある。

現在、大量のトレースログを可視化した際、スクロールバーの応答性が鈍いため、操作性が悪い。スクロールバー以外の可視化範囲の変更方法を提供することで、改善する必要がある。

謝辞

TLVを開発するにあたり、ご指導を頂きました名古屋大学大学院情報科学研究科情報システム学専攻組込みリアルタイムシステム研究室の高田広章教授に深く感謝致します。また、開発プロジェクトマネージャとして日頃より多くのご助言を頂きました愛知県立大学情報科学部情報システム学科の山本晋一郎准教授、名古屋大学大学院情報科学研究科情報システム学専攻組込みリアルタイムシステム研究室の本田晋也助教、企業出身者としての立場から実践的なご意見を頂きました同研究科付属組込みリアルタイム研究センターの長尾卓哉研究員に深く感謝致します。

参考文献

- [1] 後藤隼式『OJLによるトレースログ可視化ツールの開発』修士論文,名古屋大学,2009
- [2] 後藤隼式, 本田晋也, 長尾卓哉, 高田広章『トレースログ可視化ツールの開発』情報処理学会 第 139 回 システム LSI 設計技術 (SLDM) 研究会, 2009 年 2 月 26 日
- [3] Michael Sutton, Adam Greene, Pedram Amini, 園田道夫, 伊藤裕之『ファジング: ブルートフォースによる脆弱性発見手法』毎日コミュニケーションズ, 2008 年 5 月 28 日
- [4] JTAG ICE PARTNER-Jet, <http://www.kmckk.co.jp/jet/>, 最終アクセス 2009 年 1 月 14 日
- [5] WatchPoint デバッガ, <https://www.sophia-systems.co.jp/ice/products/watchpoint>, 最終アクセス 2009 年 1 月 14 日
- [6] QNX Momentics Tool Suite, <http://www.qnx.co.jp/products/tools/>, 最終アクセス 2009 年 1 月 14 日
- [7] eBinder, <http://www.esol.co.jp/embedded/ebinder.html>, 最終アクセス 2009 年 1 月 14 日
- [8] LKST (Linux Kernel State Tracer) - A tool that records traces of kernel state transition as events, <http://oss.hitachi.co.jp/sdl/english/lkst.html>, 最終アクセス 2009 年 1 月 14 日
- [9] Prasad, V., Cohen, W., Eigler, F. C., Hunt, M., Keniston, J. and Chen, B.: Locating system problems using dynamic instrumentation. Proc. of the Linux Symposium, Vol.2, pp.49—64, 2005.
- [10] Mathieu Desnoyers and Michel Dagenais.: The lttng tracer : A low impact performance and behavior monitor for gnu/linux. In OLS (Ottawa Linux Symposium) 2006, pp.209—224, 2006.
- [11] R. McDougall, J. Mauro, and B. Gregg.: Solaris(TM) Performance and Tools:

DTrace and MDB Techniques for Solaris 10 and OpenSolaris. Pearson Professional, 2006.

- [12] Mathieu Desnoyers and Michel Dagenais.: OS Tracing for Hardware, Driver and Binary Reverse Engineering in Linux. CodeBreakers Journal Article, vol.4, no.1, 2007.
- [13] OpenSolaris Project: Chime Visualization Tool for DTrace, <http://opensolaris.org/os/project/dtrace-chime/>, 最終アクセス 2009 年 1 月 14 日
- [14] RFC3164 The BSD syslog Protocol, <http://www.ietf.org/rfc/rfc3164.txt>, 最終アクセス 2009 年 1 月 14 日
- [15] RFC4627 The application/json Media Type for JavaScript Object Notation (JSON), <http://tools.ietf.org/html/rfc4627>, 最終アクセス 2009 年 1 月 14 日
- [16] TOPPERS Project, <http://www.toppers.jp/>, 最終アクセス 2009 年 1 月 14 日
- [17] Takuya Azumi and Masanari Yamamoto and Yasuo Kominami and Nobuhisa Takagi and Hiroshi Oyama and Hiroaki Takada.:A New Specification of Software Components for Embedded Systems. Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2007), pp.45-50, 2007
- [18] Daan Leijen and Erik Meijer, Parsec: Direct Style Monadic Parser Combinators for the Real World, Department of Computer Science, Universiteit Utrecht, UU-CS-2001-27, 2001
- [19] Erich Gamma, Ralph Johnson, Richard Helm, John Vlissides『オブジェクト指向における再利用のためのデザインパターン』, ソフトバンクパブリッシング, 1995