OJL によるトレースログ可視化ツールの開発

350702101　**後藤 隼弐**

要旨

近年，PC，サーバ，組み込みシステム等，用途を問わずマルチプロセッシングシステムの利用が進んでいる．その背景には，シングルプロセッサの高クロック化による性能向上効果の停滞や，それに伴う消費電力・発熱の増大がある．マルチプロセッシング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ることが出来る．マルチコアプロセッサ環境でソフトウェアを開発する際に問題になる点として，デバッグの困難さが挙げられる．これは，マルチコアプロセッサ環境では，従来のブレークポイントやステップ実行を用いたデバッグの挙動が，シングルプロセッサ環境における場合と異なるからである．

一方，マルチコアプロセッサ環境で有効なデバッグ手法としてトレースログの解析によるデバッグがある．これは，プログラムの実行中に，デバッグの判断材料となる情報をログとして出力することによりプログラムの動作を確認する手法である．しかしながら，トレースログを開発者が直接扱うのは困難である場合が多い．これは，膨大なログから所望の情報を探し出すのが困難であることや，各コアのログが時系列に分散して記録されるため，逐次的にログを解析することが困難であるからである．そのため，トレースログの解析を支援するツールが要求されている．

既存のトレースログ可視化表示ツールとしては，ICE 付属のデバッガソフトウェアや，組み込みシステム向け統合開発環境の一部，カーネルトレースツールの GUI，波形出力用ツールの流用などがある．しかし，これら既存のトレースログ可視化ツールは，ターゲット OS やターゲット CPU が制限されていたり，可視化表示項目が固定されていたりなど，汎用性や拡張性に乏しい．そこで我々は，これらの問題点を解決し，汎用性，拡張性を備えたトレースログ可視化ツールを開発することを目的とし，TraceLogVisualizer (TLV) を開発した．また，その過程でトレースログの標準形式を提案した．TLV の特徴は，既存のトレースログを標準形式トレースログに変換することによるログ形式の非依存化と可視化表示項目のプラグイン化である．

開発した TLV は，シングルコアプロセッサ用 RTOS やマルチコアプロセッサ用 RTOS，自動車制御用 RTOS，組込みコンポーネントシステムなどの複数のトレースログの可視化表示を行うことで汎用性の確認を行い，可視化項目を目的に合わせてカスタマイズしたりプラグインとして追加出来ることを示し，拡張性があることを確認した．また，TLV を複数の組み込みソフトウェア開発者に評価してもらい，マルチコアプロセッサ環境でのデバッグに有効であることを確認した．

TLV の開発は OJL(On the Job Learning) 形式で行い，ユースケース駆動アジャイル開発というソフトウェア開発プロセスを用いて実施された．

Development of Visualization Tool for Trace Log by OJL

350702101 Junji Goto

Abstract

Abstract

修 士 論 文

OJLによるトレースログ可視化ツールの開発

350702101　後藤 隼弐

名古屋大学 大学院情報科学研究科

情報システム学専攻

2009年1月

目 次

第1章　はじめに

1.1　開発背景

近年，PC，サーバ，組み込みシステム等，用途を問わずマルチプロセッシングシステムの利用が進んでいる．その背景には，シングルプロセッサの高クロック化による性能向上効果の停滞や，それに伴う消費電力・発熱の増大がある．マルチプロセッシング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ることが出来る．組み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る方向にあった．しかしながら，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み込みシステムでの利用が増加している．

マルチコアプロセッサ環境でソフトウェアを開発する際に問題になる点として，デバッグの困難さが挙げられる．これは，マルチコアプロセッサ環境では，従来のブレークポイントやステップ実行を用いたデバッグの挙動が，シングルプロセッサ環境における場合と異なるからである．たとえば，複数のコアで実行される可能性のあるコードにブレークポイントを置きデバッグを行う場合，ブレーク後にステップ実行を行い処理を追う最中に，他のコアのブレークにより中断される可能性があるため，ブレーク後にブレークポイントを一時的に削除する必要があり，頻繁にブレークポイントの設置，削除を行わなければならなくなる．また，マルチコアプロセッサ環境では，特殊な状況下でのみ起こるバグが発生する可能性がある．その場合，原因を特定するのは通常困難である．なぜならば，マルチコアプロセッサ環境では，特別な制御を行わない限り各コアが非同期的に並列動作するため，バグの発生を再現することが難しいからである．また，再現が可能であったとしても，原因を特定するためには，バグが発生するまでの過程をすべてのコアの実行状況を監視しながら行う必要があり，シングルプロセッサ環境の場合に比べ非常に煩雑になる．

一方，マルチコアプロセッサ環境で有効なデバッグ手法としてトレースログの解析によるデバッグがある．これは，プログラムの実行中に，デバッグの判断材料となる情報をログとして出力することによりプログラムの動作を確認する手法である．ログ

の出力元としては，OS や ICE，シミュレータなどがあり，情報の粒度もアプリケーションレベル，タスクレベル，カーネルレベル，ハードウェアレベルなど様々である．

しかしながら，トレースログを開発者が直接扱うのは困難である場合が多い．これは，ログの情報の粒度が細かくなるほど単位時間あたりのログの量が増える傾向にあり，膨大なログから所望の情報を探し出すのが困難であることや，各コアのログが時系列に分散して記録されるため，逐次的にログを解析することが困難であるからである．そのため，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，ログを解析し統計情報として出力したり，可視化表示することで開発者の負担を下げる試みが行われている．

1.2 既存のトレースログ可視化ツール

既存のトレースログ可視化表示ツールとしては，ICE 付属のデバッガソフトウェアや，組み込みシステム向け統合開発環境の一部，カーネルトレースツールの GUI，波形表示用ツールの流用などがある．

本節では，これら既存のトレースログ可視化ツールについて説明した後，これらの問題点について指摘する．

1.2.1 JTAG エミュレータ付属デバッガソフトウェア

JTAG エミュレータとは，オンチップ・エミュレータの一種であり，基板上にプロセッサを実装した状態でプログラムのデバッグを行うことのできる装置である．主に組み込みシステムでのソフトウェア開発に使用され，ターゲットコンピュータと JTAG エミュレータをデバッグ用のインターフェイスで接続し，JTAG エミュレータとホストコンピュータを Ethernet やシリアルポートで接続することで，ターゲットコンピュータ上でプログラムを実行しながらホストコンピュータでデバッグを行うことができる．ホストコンピュータでは，JTAG エミュレータ付属のデバッガソフトウェアを使用してデバッグを行うが，そのソフトウェアに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．図 1.1 に，京都マイクロコンピュータ株式会社の JTAG エミューレータ PARTNER-Jet[1] に付属するデバッガソフトウェアの可視化機能であるイベントトラッカーのスクリーンショットを示す．

JTAG エミューレータ付属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てしまう．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．

1.2.2 組み込み OS 向けの統合開発環境

QNX Software Systems 社は，自社の組み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[2] という機能を提供している．図 1.2 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-Kenel/μITRON ベースシステムの統合開発環境として eBinder[3] 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出来る．図 1.3 に QNX System Profiler のスクリーンショットを示す.

このように，商用の組み込み 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されているものであり，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.

1.2.3 カーネルトレースツールの GUI

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，オペレーティングシステム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．Linux 用のトレースツールとしては LKST[4]，SystemTap[5]，LTTng[6] などがあり，Solaris 用には Dtrace[7] がある．これらトレースツールは，主に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，通知をログとして記録する機能を提供している．また，ログを分析・提示する GUI ソフトウェアとして専用の可視化ツールを提供している場合が多い．たとえば，LTTng には LTTV[8] が，DTrace には Chime[9] が専用の可視化ツールとして提供されている．図 1.4 に LTTV のスクリーンショットを，図 1.5 に Chime のスクリーンショットを示す.

これら，カーネルトレースツールの GUI ソフトウェアは，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

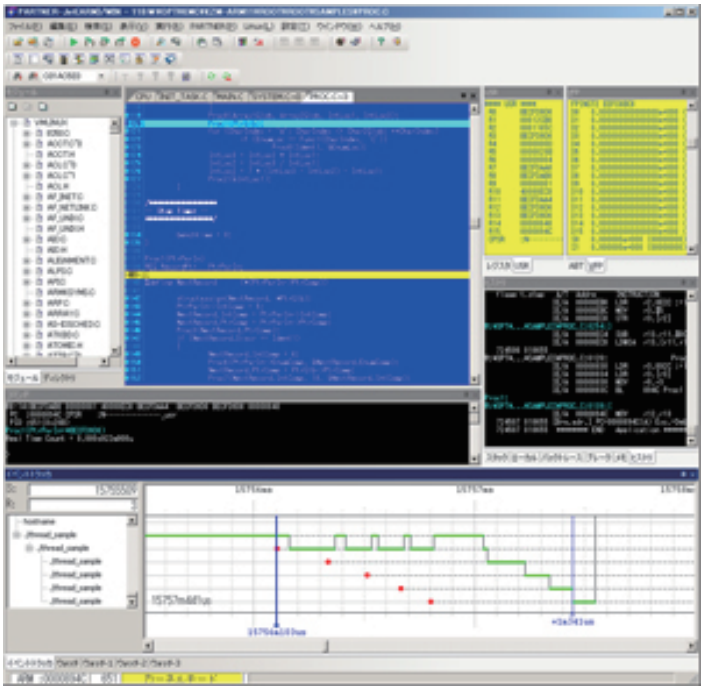


図 1.1: PARTNER-JET イベントトラッカー

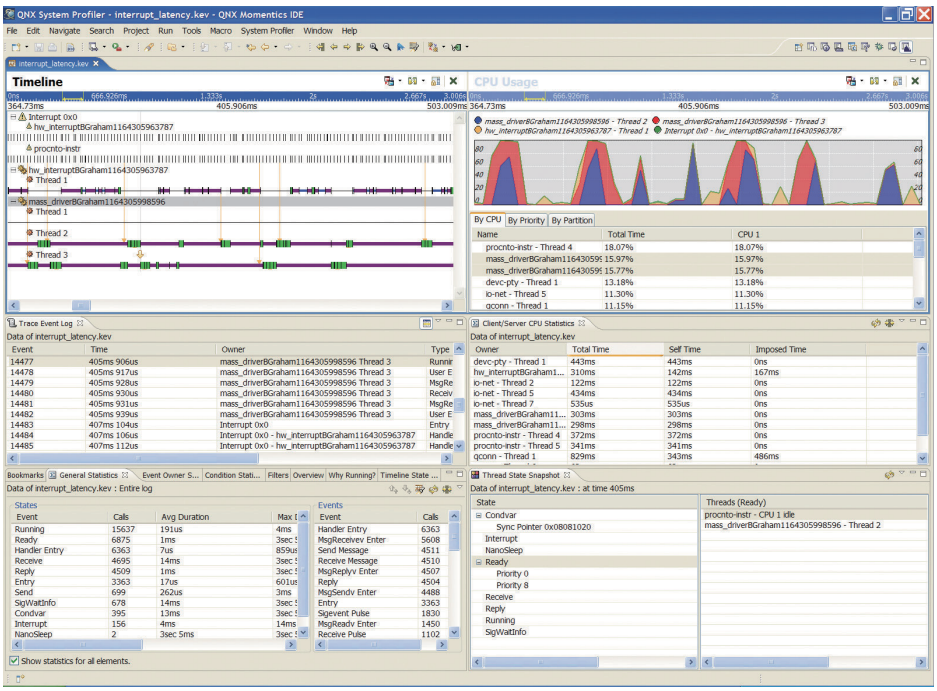


図 1.2: QNXSystemProfiler

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．

1.2.4 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提供されている．そのため，任意のログを VCD 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 1.6 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，表示能力に限界があり，複雑な可視化表現は難しい．

1.2.5 既存のトレースログ可視化ツールの問題点

既存のトレースログ可視化ツールは，可視化ツールとして単体で存在しているわけではなく，トレースログを出力するソフトウェアの一部として提供されている．そのため，読み込めるログの形式が出力ソフトウェアに依存し，ターゲット OS，ターゲット CPU が制限されてしまっている．読み込むログの形式を制限しないためには，波形表示ツールのように可視化ツール用のオープンなログ形式を定める必要があると考えられるが，現状，公表されているものではそのようなものは確認できていない．ログ形式の標準化としては，syslog[10] と呼ばれる，ログメッセージを IP ネットワーク上で転送するための標準規格が RFC3164 として策定されており，その一部にログ形式について規定されている．しかしながら，syslog におけるログ形式は，時刻の最小単位が秒であり，デバッグを目的に利用するには粒度が荒く，また，メッセージ内容がフリーフォーマットであるなど，自由度が高すぎるため，可視化ツールが読み込むログの標準形式としては不適切である．

既存のトレースログ可視化ツールのもうひとつの問題点としては，可視化表示の項目や形式が，提供されているものに限られていることが挙げられる．これは，ログのターゲットや可視化の目的に合わせて可視化表現が最適化されているためである．また，既存のトレースログ可視化ツールで可視化表示の項目を追加，変更したり，可視化表現をカスタマイズする機能を搭載するものは確認できていない．

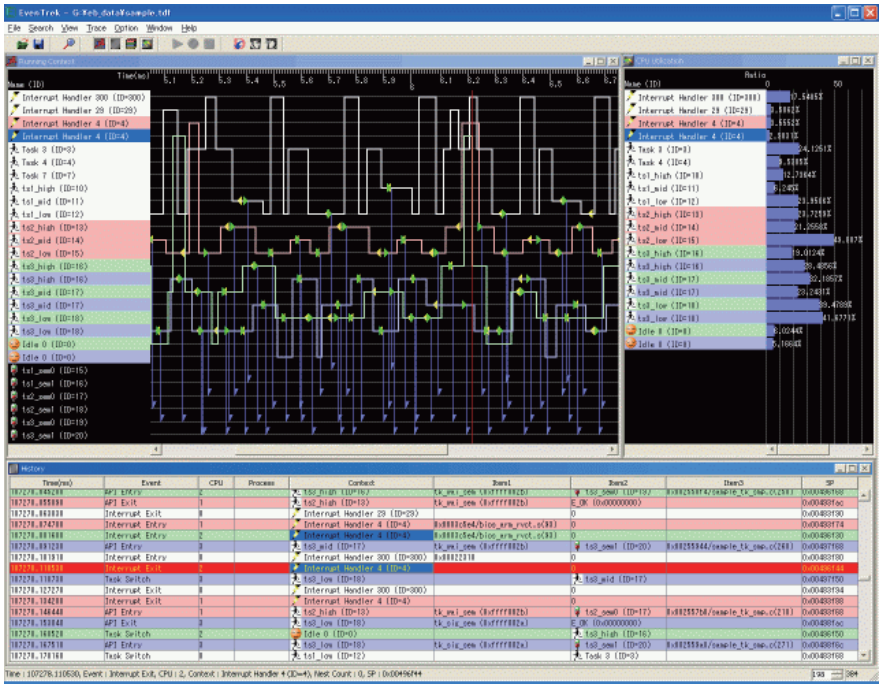


図 1.3: eBinder EvenTrek

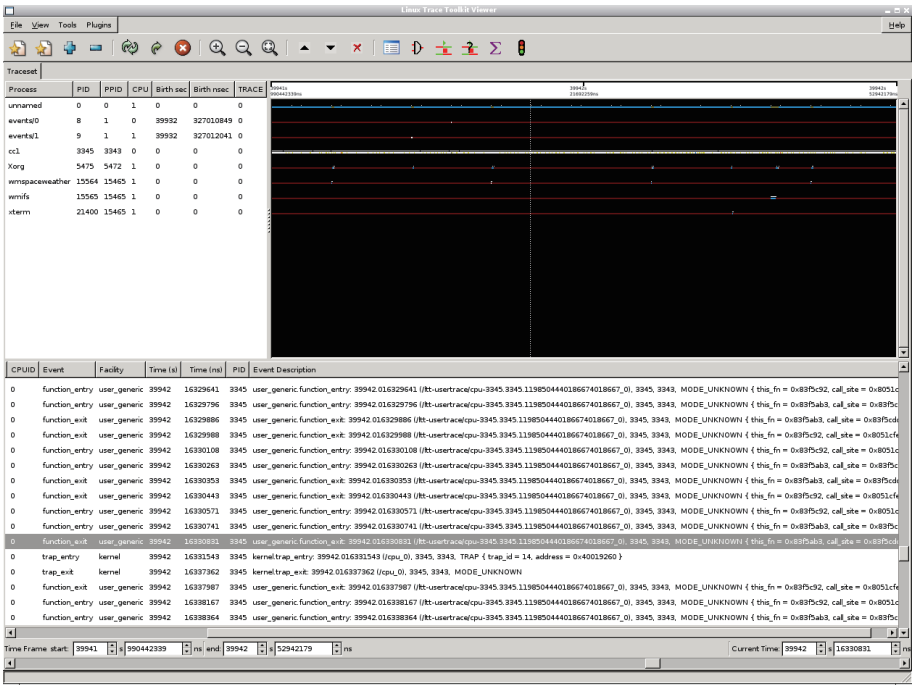


図 1.4: LTTV

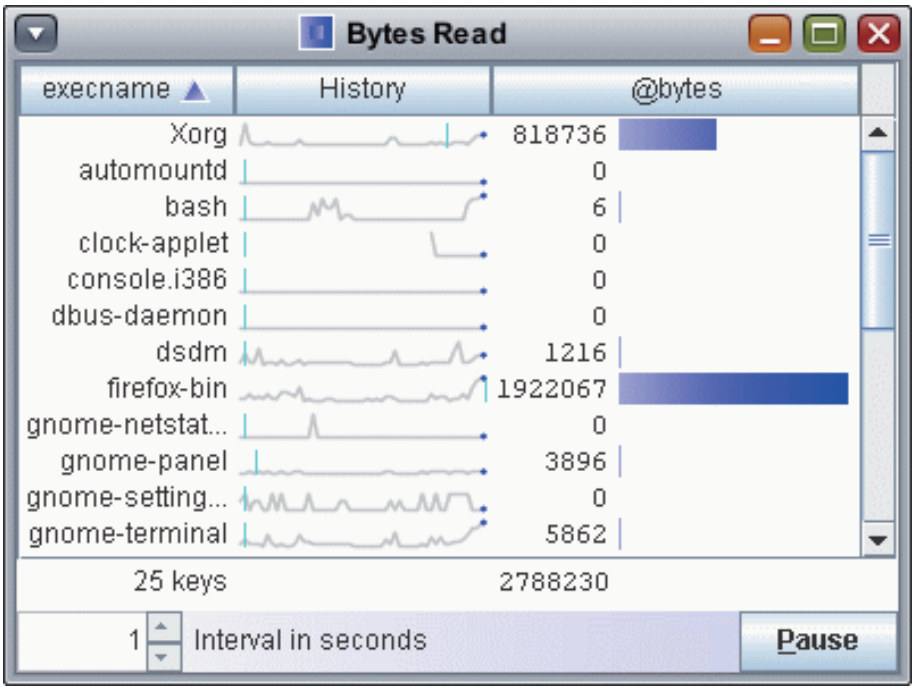


図 1.5: Chime

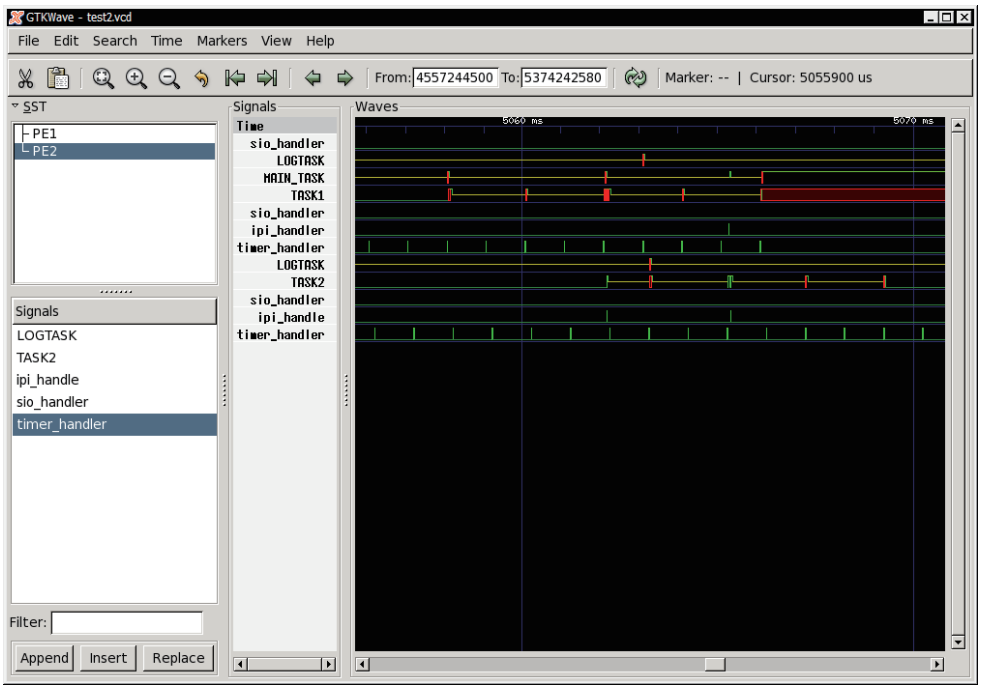


図 1.6: GTKWave

1.3 開発目的と内容

前節で説明したとおり，既存のトレースログ可視化ツールには，オープンなログ形式がないことによりターゲットが限定されている点，可視化表示項目がターゲットにより固定されている点の２つの問題点があることを指摘した．そこで我々は，これらの問題点を解決し，汎用性，拡張性を備えたトレースログ可視化ツールを開発することを目的とし，成果物として TraceLogVisualizer (TLV) を開発した．また，その過程でトレースログの標準形式を提案した．

問題解決のために TLV が実装した機能は，トレースログの標準形式への変換と可視化表示項目のプラグイン化である．トレースログを標準形式トレースログに変換することにより，あらゆるトレースログが TLV で可視化表示可能となる．

1.4 論文の構成

本節では本論文の構成を述べる．

2 章では，TLV の設計について述べる．ここでは，問題解決のために開発方針をどのように設定したかを述べ，具体的な解決策をどのように設計したかを述べる．3 章では，TLV の実装について述べる．2 章で述べた設計をメカニズムとしてどのように実現しているかを，プロセスとインターフェイスに分けて説明する．4 章では，TLV を利用した例を示し，その有効性について言及している．5 章では TLV を開発するにあたり，どのような開発プロセスを用いたのかを述べている．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 TraceLogVisualizer の設計

2.1　開発方針

TLV の開発目標は，汎用性と拡張性を備えることである．

ここで，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，可視化表示メカニズム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，標準となるトレースログの形式を定義し，可視化表示メカニズムがこれに依存するようにする．本論文では以後，この形式を標準形式トレースログと呼称する．ターゲット依存のトレースログを標準形式トレースログに変換することにより，トレースログの形式に依存せずに可視化表示が可能となる．

次に，拡張性とは，トレースログに対する可視化表現をユーザレベルで拡張出来ることを表し，可視化表現，また可視化表現とトレースログの対応を抽象化し，独立して定義できるようにすることで実現する．具体的には，表示する図形，トレースログと表示する図形の対応をテキスト形式でユーザが定義出来るようにし，それらをプラグインとして動的に組み込ませることを指す．

2.2　標準形式トレースログ

TLV の汎用性は，標準形式トレースログを定義し，これにのみ依存させることで実現することを前節で述べた．

本節では，標準形式トレースログを定義するために行ったトレースログの抽象化と，標準形式トレースログの定義について述べる．

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．

はじめに，トレースログを時系列にイベントを記録したものと考えた．次に，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出来る．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．属性と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，振る舞いとはリソースの行為であるとする．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．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，オブジェクト指向におけるメソッドの，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ものである．振る舞いの引数は，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．

図 2.1 と図 2.2 に，リソースタイプとリソースを図で表現した例を示す．さらに，図 2.3 に，RTOS(Real-time operating system) におけるタスクの概念をリソースタイプとして表現した例を，図 2.4 にリソースタイプ Task のリソースの例として MainTask を示す．

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．

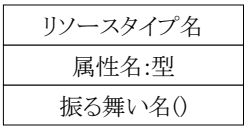


図 2.1: リソースタイプ

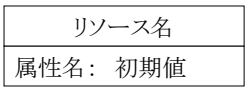


図 2.2: リソース

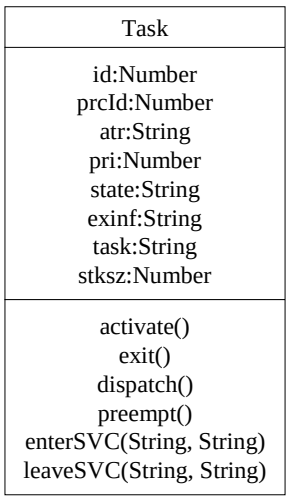


図 2.3: タスクをリソースタイプ Task として表現した例

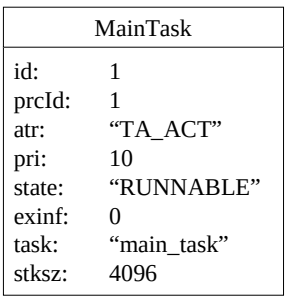


図 2.4: リソースタイプ Task のリソース MainTask の例

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定義する．標準形式トレースログの定義にはEBNF(Extended Backus—Naur Form) および終端記号として正規表現を用いる．正規表現はスラッシュ記号 (/) で挟むとする．

前小節によればトレースログと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．

```
TraceLog = { TraceLogLine };
TraceLogLine = "[",Time,"]",Event,"\n";
Time = /^[0-9a-Z]+$/;
```

トレースログが記録されたファイルのデータをTraceLogとした．また，TraceLogを改行記号で区切った1行をTraceLogLineとし，トレースログの単位とした．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，数値とアルファベットで構成される．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である．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」などを表現出来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，Eventを次のように定義した．

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

リソースはリソース名による直接指定、あるいは型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．

```
Resource = ResourceName
         | ResourceTypeName,"(",AttributeCondition,")";
ResourceName = Name;
ResourceTypeName = Name;
Name = /^[0-9a-Z_]+$/;
```

AttributeChange は属性の値の変化を，BehaviorHappen は振る舞いを表現している．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以下のように定義した．

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

リソースの条件指定の際の AttributeCondition は以下のように定義する．

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

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以下に示す．例中のリソースは，図 2.3 と図 2.4 に示したリソースタイプ Task のリソースとする．

```
[2403010]MAIN_TASK.leaveSVC(ena_tex,ercd=0)
[4496099]MAIN_TASK.state=RUNNABLE
[4496802]TASK(state==RUNNING).preempt()
[4496802]TASK(state==RUNNING).state=RUNNABLE
[4496802]TASK(id==2).dispatch()
```

2.3　可視化表示メカニズムの抽象化

前節では，トレースログを抽象化し，標準形式トレースログとして定義した．TLV の可視化表示メカニズムは，この抽象化にのみ依存するように設計されなければならない．本節では，可視化表現と可視化表現とトレースログの対応を抽象化する．

2.3.1　可視化表現

TLV における可視化表現は，2 次元直交座標系における図形の描画とする．x 軸は水平方向に右の方向を正の向きとし，y 軸は垂直方向に上の方向を正の向きとする．

座標系

図形を定義する座標系をローカル座標系，図形を時系列にマッピングする際の座標系をワールド座標系，表示デバイスの座標系をデバイス座標系と呼称する．

ローカル座標系において図形を定義する際は，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いずれかを用いる．

ワールド座標系の x 軸は時間軸となる．ローカル座標系で定義された図形は，表示する時間の領域にマッピングされる．これ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．

図 2.5 に座標系の例を，図 2.7 にワールド変換の例を示す．

基本図形と図形

図形の基本単位は楕円形，多角形，四角形，線分，矢印，扇形，文字列とし，これらを基本図形と呼称する．

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．また，同じように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 2.7 に図形と図形群の例を示す．

2.3.2　図形とイベントの対応

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．

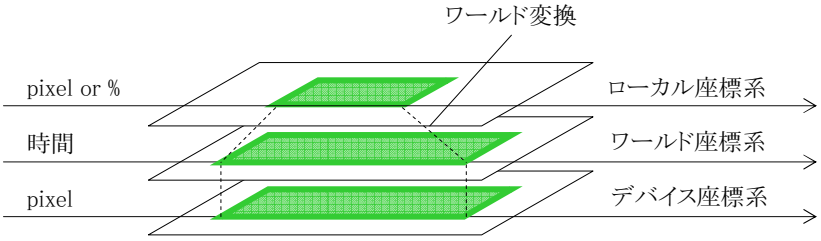


図 2.5: 座標系

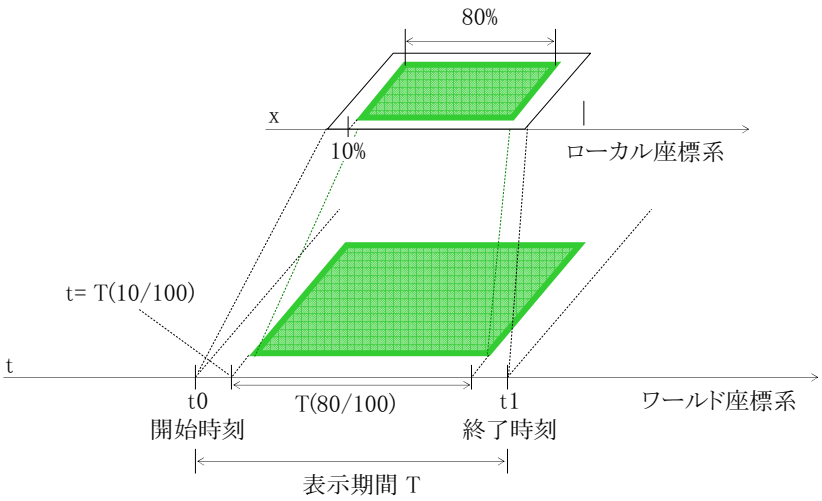


図 2.6: ワールド変換

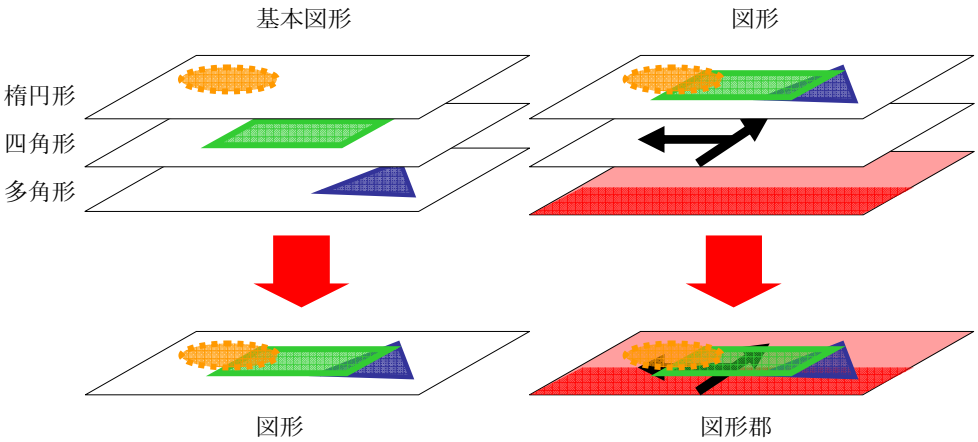


図 2.7: 図形と図形群

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，図形はワールド変換を経て表示期間にマッピングされることを説明した．ここで表示期間の開始時刻，終了時刻はイベントを用いて指定する．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．ここで，開始時刻と終了時刻に対応するイベントを開始イベント，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．

期間図形

図形群あるいは図形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期間図形と呼称する．

図 2.8 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期間図形の例を示す．ここで，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，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期間図形 runningTimeShape 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．

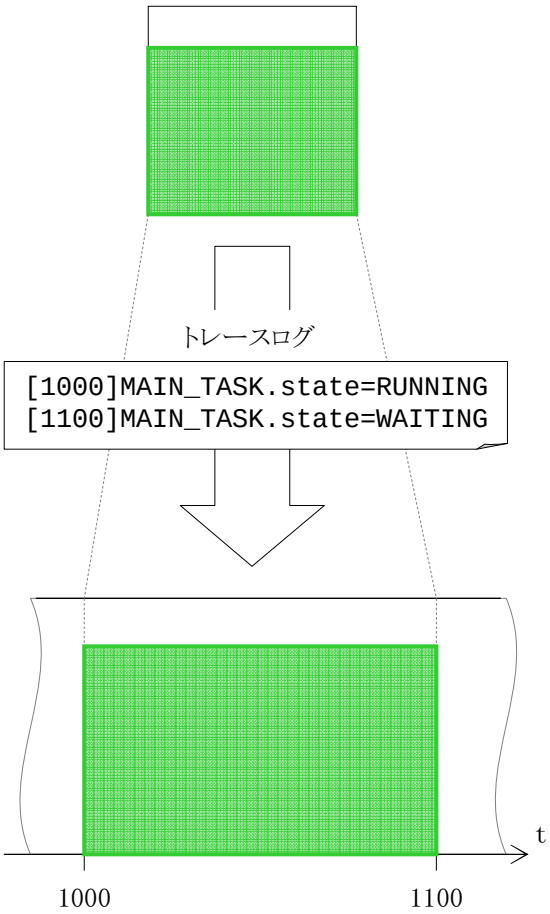
runningTimeShape を以下のトレースログからイベントを抽出して表示期間の時刻を決定しワールド変換を行った結果が図 2.8 の右下に示すものである．

```
[1000]MAIN_TASK.state=RUNNING
[1100]MAIN_TASK.state=WAITING
```

開始イベントと終了イベントから表示期間が決定されワールド変換が行われている．

図形

runningShape
形:長方形 位置:(x,y)=(0%,0%) 横幅:100% 縦幅:80% 色:(R,G,B)=(0,255,0)



期間図形

runningTimeShape
開始イベント:MAIN_TASK.state=RUNNING 終了イベント:MAIN_TASK.state 図形:runningShape

図 2.8: 期間図形

第3章　トレースログ可視化ツール TraceLogVisualizer の実装

3.1　TraceLogVisualizer のプロセス

3.1.1　標準形式への変換

3.1.2　図形データの生成

3.2　TraceLogVisualizer のインターフェイス

3.2.1　Json 形式

3.2.2　トレースログファイル

3.2.3　リソースファイル

3.2.4　リソースヘッダファイル

3.2.5　変換ルールファイル

3.2.6　可視化ルールファイル

3.2.7　TLV ファイル

第4章　トレースログ可視化ツール TraceLogVisualizer の利用

4.1　組み込みRTOSのトレースログの可視化

4.1.1　シングルコアプロセッサ用RTOSのトレースログの可視化

4.1.2　マルチコアプロセッサ用RTOS対応への拡張

4.1.3　その他のシステムのトレースログの可視化

4.2　可視化表示項目の追加・カスタマイズ

第5章　開発プロセス

5.1　OJL

5.1.1　フェーズ分割

5.2　ユースケース駆動アジャイル開発

5.2.1　プロジェクト管理

5.2.2　設計

5.2.3　テスト

5.2.4　実装

5.3　開発成果物

第6章　おわりに

6.1　まとめ

6.2　今後の展望と課題

謝辞

参考文献

[1] JTAG ICE PARTNER-Jet，http://www.kmckk.co.jp/jet/，最終アクセス 2009 年 1 月 14 日

[2] QNX Momentics Tool Suite，http://www.qnx.co.jp/products/tools/，最終アクセス 2009 年 1 月 14 日

[3] eBinder，http://www.esol.co.jp/embedded/ebinder.html，最終アクセス 2009 年 1 月 14 日

[4] 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

[5] 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.

[6] 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.

[7] R. McDougall, J. Mauro, and B. Gregg.: Solaris(TM) Performance and Tools: DTrace and MDB Techniques for Solaris 10 and OpenSolaris. Pearson Professional, 2006.

[8] Mathieu Desnoyers and Michel R. Dagenais.:Tracing for Hardware, Driver, and Binary Reverse Engineering in Linux. Recon 2006

[9] 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

[10] RFC3164 The BSD syslog Protocol, http://www.ietf.org/rfc/rfc3164.txt，最終アクセス 2009 年 1 月 14 日

OJLによるトレースログ可視化ツールの開発

350702101　後藤 隼弐