トレースログ可視化ツール TraceLogVisualizer(TLV)

後藤 隼弐　本田 晋也　長尾 卓哉　高田 広章

マルチプロセッサ環境で動作するソフトウェアのデバッグには，トレースログの解析が有効であるが，開発者がトレースログを直接扱うのは非効率的である．そのため，トレースログを可視化することにより解析を支援するツールが存在するが，これらは汎用性や拡張性に乏しいという問題がある．そこで我々は，これらの問題を解決したトレースログ可視化ツール (TLV) を開発した．TLV は，変換ルールによりトレースログを標準形式に変換することで汎用性を実現し，可視化ルールにより可視化表示項目の追加・変更を可能にすることで拡張性を実現している．TLV を用いて形式の異なる 3 種類のトレースログを可視化し，汎用性と拡張性を確認した．

It is effective to debug software executed in multi processors by analyzing a trace log, but it is inefficient that developers analyze the trace log directly. Therefore, some visualization tools for the trace log had been developed before now, but there are two problems, lacks of general versatility and expandability. To that end, we developed TraceLogVisualizer(TLV), a visualization tool for the trace log that solved those problems. In this paper, we will discuss requirements to achieve general versatility and expandability, and how we achieved them.

1　はじめに

近年，組込みシステムにおいても，マルチプロセッサ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限界や，消費電力の増大がある．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ることができる．しかし，マルチプロセッサ環境で動作するソフトウェアのデバッグを行うのは，シングルプロセッサ環境に比べ困難であるという問題がある．これは，各プロセッサが非同期的に並列動作するため，タイミングに依存した再現性の低いバグが発生する可能性があることや，ハードウェアの制約からプロセッサ間の同期精度が低い場合があり，ブレークポイントやステップ実行を用いた従来のデバッグ手法が有効ではない場合があるからである．

一方，マルチプロセッサ環境で有効なデバッグ手法として，プログラム実行履歴であるトレースログを解析する手法がある．この手法が有効である理由は，並列プログラムのデバッグを行う際に必要な情報である，プロセスが，いつ，どのプロセッサで，どのように動作していたかを知ることができるからである．しかしながら，開発者がトレースログを直接解析するのは効率が悪い．図 1 に，マルチプロセッサ対応の RTOS である，TOPPERS/FMP カーネル (以下，FMP カーネル) [2] が 2 プロセッサで動作した場合のトレースログの例を示す．図 1 の例では，約 430μs の間に 5 行のトレースログが出力されており，トレースログを収集する時間が長くなれば膨大な量になることを示唆している．取得する情報の粒度を細かく

Visualization Tool for Trace Log. TraceLogVisualizer (TLV)

Junji Goto, Hiroaki Takada, 名古屋大学 大学院情報科学研究科, Graduate School of Information Science, Nagoya University.

Shinya Honda, Takuya Nagao, 名古屋大学 大学院情報科学研究科 附属組込みシステム研究センター, Center for Embedded Computing Systems, Graduate School of Information Science, Nagoya University

[ソフトウェア論文] xx 年 x 月 xx 日受付.

[*time(μs)*]:[*core_id*]: *information*

```
[60692484]:[1]: task 4 becomes WAITING.
[60692586]:[2]: leave to sns_ctx state=0.
[60692708]:[1]: dispatch from task 4.
[60692798]:[2]: enter to get_pid p_prcid=304.
[60692914]:[1]: dispatch to task 1.
```

図 1　2 プロセッサ上で動作する TOPPERS/FMP カーネルが出力するトレースログの例

したり，処理の複雑さが増すと，出力されるトレースログの量はさらに増大するため，デバッグを行う際に必要な情報を探し出すのが困難となる．また，各プロセッサのトレースログは時系列に分散して記録されているため，プロセッサ毎の動作を解析するのが困難である．

トレースログの解析を支援する方法として，ツールによるトレースログの可視化があり，これまでに幾つかのツールが開発されている．具体的には，組込みシステム向けデバッガソフトウェアや統合開発環境の一部 [3] [4] [5] [6] や Unix 系 OS のトレースログプロファイラ [8] [10] などが存在する．しかしながら，これら既存のツールは，特定の環境 (OS やデバッグハードウェア) を対象としているため，可視化対象となるトレースログの形式が固定されている．そのため，他の環境が出力するトレースログを可視化することができず，汎用性に乏しいという問題がある．また，可視化表示する項目やその表現方法が固定である場合が多く，拡張性に乏しいという問題もある．

そこで我々は，これらの問題を解決し，汎用性と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．TLV は,「OJL による最先端技術適応能力を持つ IT 人材育成拠点の形成」の OJL(On the Job Learning) の開発テーマとして開発し，オープンソースソフトウェアとして公開している [1].

開発にあたり，我々は，まず，TLV 内部でトレースログを一般的に扱えるよう，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た．さらに，任意の形式のトレースログを標準形式に変換するためのルールを変換ルールとして形式化し，ユーザがツールの外部から指定できる仕組みを提供した．次に，トレースログの可視化表示項目やその表現方法を指示する仕組みの抽象化を行い，可視化ルールとして形式化した．また，可視化ルールをユーザが外部から指定できる仕組みを提供した．これにより，TLV では，トレースログの形式毎に変換ルールと可視化ルールを外部ファイルとして用意することで可視化が可能であり，汎用性と拡張性を実現している．

本論文は次のように構成される．まず，2 章で既存のトレースログ可視化ツールとその問題点について述べる．3 章では，汎用性と拡張性を実現するためのトレースログ可視化ツールの要件と，TLV での実現方針について述べる．4 章では，要件と実現方針に基づく TLV の設計について述べ，5 章では，その実装について述べる．7 章では TLV のユーザーインタフェースについて述べ，7 章では，複数のトレースログの可視化表示や，可視化表示項目の変更，追加を行い，TLV の汎用性と拡張性を示す．最後に，8 章でまとめと今後の課題，展望を述べる．

2　既存のトレースログ可視化ツールと問題点

2.1　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，組込みシステム向けデバッガソフトウェアや統合開発環境の一部，Unix 系 OS の可視化ツール，波形表示用ツールなどがある．

組込みシステム向けデバッガソフトウェアには，トレースログを可視化する機能を持つものがある．PARTNER [3] はイベントトラッカー，WatchPoint [4] は，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．また，組込み OS 向けの統合開発環境にも OS のトレースログを可視化するツールが含まれている．QNX 用の統合開発環境である QNX Momentics では，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 [5] を提供している．また，T-Kernel 用の統合開発環境である eBinder [6] 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．

Unix 系 OS では，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

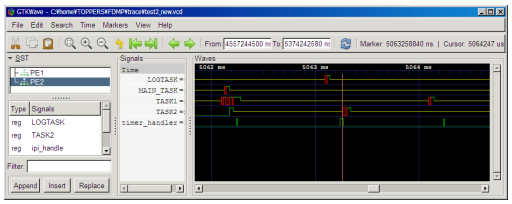

図 2 GTKWave による RTOS のトレースログの可視化

得しそれを可視化するソフトウェアがいくつか開発されている．Linux の実行トレースを取得する LTTng [7] には，LTTV [8] が，Solaris の実行トレースを取得する Dtrace [9] には，Chime [10] が可視化ツールとして提供されている．

また，デジタル回路設計用論理シミュレータ用に，VCD(Value Change Dump) 形式というオープンなトレースログの形式が存在する．VCD 形式を可視化するツールとして，GTKWave が存在する．

2.2 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の多くは，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や開発環境の一部として提供されている．そのため，扱えるトレースログの形式が限定されている．

任意の形式のトレースログを VCD 形式に変換することにより，GTKWave 等を利用できる．図 2 に，RTOS のトレースログを VCD 形式として，GTKWave で可視化したスクリーンショットを示す．VCD 形式はデジタル回路設計用の形式であるため，表示能力に乏しく，複雑な可視化表現は不可能である．

扱うトレースログの形式を制限しないためには，VCD 形式のようにトレースログ可視化ツール用に標準化されたトレースログ形式を定める必要があると考えられる．しかしながら，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．

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．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．

3 要件と実現方針

本章では，トレースログ可視化ツールが汎用性と拡張性を備えるための要件と，TLV での実現方針についてそれぞれ説明する．

3.1 要件

まず，トレースログ可視化ツールの定義，ならびにその汎用性と拡張性の要件を明確にする．

本研究では，トレースログ可視化ツールを，時系列に記録されたプログラムの実行履歴をテキスト形式でファイル化したトレースログファイルを読み込み，その内容を GUI を通じて時系列に可視化表示するツールとする．

可視化表示項目とは，トレースログの内容や出力元の環境，解析の目的などに基づき区分される可視化表示する情報の単位である．具体的には，RTOS のトレースログの場合，“タスクの状態遷移” や “システムコール呼び出し” などが該当する．

汎用性は，ユーザがツールのバイナリを変更することなく，様々な形式のトレースログを可視化表示できることとする．すなわち，入力するトレースログの形式を制限しない．

拡張性は，ユーザがツールのバイナリを変更することなく，容易に可視化表示項目の追加・変更・削除ができることとする．

3.2 汎用性の実現方針

汎用性を実現するために，トレースログの標準形式を定め，可視化表示の仕組みが標準形式にのみ依存するようにする．そして，任意の形式のトレースログを，標準形式に変換するためのルールを変換ルールとして形式化し，ユーザが外部から指定できる仕組みを提供する．

トレースログ可視化ツールが汎用性を実現するためには，可視化表示の仕組みが特定のトレースログの形式に依存しないようにしなければならない．そのため，様々なトレースログ形式を一般化したトレースロ

グ形式を定め，トレースログ可視化ツールはこの形式を扱うこととする．一般化したトレースログの形式を標準形式と呼ぶ．

標準形式への変換は，逐次的なテキストの置き換えといった，単純なテキスト処理では要件を満たせない場合がある．

例として，図 3 に示す，ITRON 仕様 [14] の RTOS である TOPPERS/ASP カーネル (以下，ASP カーネル) [13] のトレースログを標準形式に変換する場合を考える．まず，1 行目で実行状態 (RUUNABLE) のタスクが act_tsk() というシステムコールで，タスク ID(tskid) が 1 番のタスクを起動する．2 行目で起動されたタスク ID が 1 番のタスクの状態が実行可能状態 (READY) となる．次に，タスク ID が 1 番のタスクは，実行状態のタスクより優先度が高いため，3 行目でディスパッチが発生する．

ここで，3 行目には，実行状態 (RUUNABLE) のタスクの状態が実行可能状態 (READY) となり，代わりに，タスク ID が 1 番のタスクの状態が実行状態 (RUUNABLE) となったという情報が暗に含まれている．ASP カーネルがこのログを出力しないのは，トレースログを出力するコストを減らすためである．一方，標準形式のトレースログでは，可視化のために，タスク状態の変更箇所を明確に出力する必要がある．そのため，変換ルールでは，3 行目のディスパッチのログを変換する際に，この時点で実行状態であるタスクが存在するかどうか†1，また，存在する場合はどのタスクであるのか，という情報が必要となる．そのため，これらの情報を，変換元のトレースログファイルの内容から判断する必要がある．

このように，標準形式への変換処理においては，変換元のトレースログが暗に含む情報を出力するため，それを出力する条件の制御，また条件を制御するために必要な情報を取得する手段が必要である．

†1 実行中のタスクが存在しない状態で，割込みハンドラからタスクが起動された場合は，実行状態のタスクは存在しない．

```
1 [1000]enter to act_tsk tskid=1
2 [1005]task 1 becomes READY
3 [1010]dispatch to task 1.
```

図 3 TOPPERS/ASP カーネルのトレースログの例

3.3 拡張性の実現方針

拡張性を実現するためには，汎用的な可視化表示の仕組みを提供し，ユーザが可視化表示項目に合わせてそれを制御できればよい．つまり，可視化表示項目毎に可視化表示の仕組みを用意するのではなく，汎用的な可視化表示の仕組みに対して，可視化表示項目を実現するパラメータを指定することで可視化表示を行う．そして，ユーザがツールの外部からパラメータを指定する手段を実現すればよい．その際，汎用性の実現のため，可視化表示の仕組みは，標準形式のトレースログにのみ依存するようにしなければならない．また，指定するパラメータは，描画処理に必要な GUI フレームワークや表示デバイスに依存しないことが要求される．これは，プログラムの可搬性を損なわないためである．

本研究では，汎用的な可視化表示の仕組みを実現するために，まず，トレースログの内容から可視化表現を決定して表示するまでの流れを抽象化し，本質的な処理を洗い出す．次に，ユーザが，どの処理に，どのようなパラメータを指定できればよいかを判断し，可視化ルールとして形式化する．そして，ユーザがツールの外部から可視化ルールを指定できる仕組みを提供する．

4 設計

本章では，前章で汎用性を実現するための手段として挙げた標準形式と変換ルールと，拡張性を実現するための手段である可視化表示の仕組みの設計について説明する．

4.1 標準形式

トレースログを一般化するため，RTOS や Unix 系 OS，ISS(Instruction Set Simulator) などのトレースログの形式の調査を行った．その結果から，次のようにトレースログを一般化した．

トレースログは，時系列にイベントを記録したものとした．イベントとはイベント発生源の事象であり，イベント発生源の “属性の変化”，または “振る舞い” とした．ここで，イベント発生源をリソースと呼称し，固有の識別子として名前を持つとした．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，固有の属性を持つ．固有の属性としては，“属性” と “振る舞い” があり，型により定義される．型は，リソースタイプと呼称して固有の識別子として名前をもつとした．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．それぞれは固有の識別子として名前をもつ．

例として，RTOS が出力する,「TASK1 という名前のタスクの状態が，実行状態に遷移した」という情報のトレースログを，一般化した方法で考えてみる．この場合，TASK1 がリソースであり，タスクがリソースタイプである．また，タスクの状態は属性にあたり，その属性が実行状態という値に変化したというイベントであると言える．また,「TASK1 という名前のタスクがセマフォを取得するシステムコールを呼び出した」という情報のトレースログでは，“システムコール呼び出し” が振る舞いとなる．

図 4 に，一般化の結果を形式化したトレースログの標準形式の定義を示す．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ている．正規表現はスラッシュ記号 (/) で挟み記述している．

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．TraceLogLine は “[”,“]” で時刻を囲み，その後ろにイベント (Event) 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．

イベント (Event) は，リソース (Res) と，属性の値の変化イベント (AttributeChange) もしくは，振る舞いイベント (BehaviorHappen) をドッド “.” で繋げることで，どのリソースの属性の値もしくは振る舞いのイベントが発生したかを表す．

```
1  TraceLog = { TraceLogLine,"\n" };
2  TraceLogLine = "[",Time,"]",Event;
3  Time = /[0-9a-zA-Z]+/;
4  Event = Res,".",(AttrChange|BhvrHappen);
5  Res = ResName|ResTypeName,"(",AttrCond,")";
6  ResName = Name;
7  ResTypeName = Name;
8  AttrCond = BoolExp;
9  BoolExp = Bool|CompExp
10    |BoolExp,[{LogclOpe,BoolExp}]
11    |"(",BoolExp,")";
12 CompExp = AttrName,CompOpe,Value;
13 Bool = "true"|"false";
14 LogclOpe = "&&"|"||";
15 CompOpe = "=="|"!="|"<"|">"|"<="|">=";
16 AttrName = Name;
17 AttrChange = AttrName,"=",Value;
18 Value = /[^"\\]+/;
19 BhvrHappen =  BhvrName,"(",Args,")";
20 BhvrName = Name;
21 Args = [{Arg,[","]}];
22 Arg = /[^"\\]*/;
23 Name = /[0-9a-zA-Z_]+/;
```

図 4 標準形式の定義

リソースの記述方法 (5 行目) は，リソースの名前 (`ResName`) による直接指定か，リソースタイプの名前と属性の条件 (`ResTypeName(AttrCond)`) による条件指定の 2 通りの方法で指定できるとした．条件指定では，リソースタイプとリソースの属性の値から，特定のリソース，または複数のリソースを表現することができる（8〜15 行目）．条件指定は，3. 2 節で述べた，変換元のトレースログが暗に含む情報の表現を可能にするため導入した．

属性の値の変化イベント (17 行目) と振る舞いのイベント (19 行目) の記述方法は，Java や C++などのオブジェクト指向言語における，メンバ変数への代入と，メソッドの呼び出しの記法と同様である．

4.1.1 標準形式の例

図 3 の ASP カーネルトレースログを標準形式とした例を図 5 に示す．タスク ID が 1 のタスクは TASK1 という名前のリソースとする[†2]．

1 行目が振る舞いのイベントであり，2〜4 行目が

†2 図 5 標準形式は変換ルールによる変換結果ではなく，標準形式の説明のための人手による変換結果である．

```
1 [1000]TASK(state==RUNNING).enterSVC(act_tsk, tskid=1)
2 [1005]TASK1.state=READY
3 [1010]TASK(state==RUNNING).state=READY
4 [1010]TASK1.state=RUNNING
```

図 5　図 3 の ASP カーネルのトレースログの標準形式による表現

属性の値の変化イベントである．1 行目は，実行状態 (RUNNING) のリソースが `leaveSVC` という振る舞いを，`act_tsk` と `tskid=1` を引数として発生したことを表現している．2〜3 行目は各リソースの属性であるタスク状態 (`state`) の値が変化したことを表現している．2 行目と 4 行目はリソースを名前で直接指定しているが，1 行目と 3 行目はリソースタイプの名前 (TASK) と属性の条件 (state==RUNNING) によってリソースを指定している．それぞれ，その時点でリソースタイプの名前が `TASK` のリソースのうち，属性のタスク状態 (`state`) が実行状態 (RUNNING) であるリソースを指定している．

4.2　変換ルール

変換ルールは，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ためのルールであり，これらの対応関係を定義したものである．具体的には，変換対象となる変換元のトレースログを正規表現として記述し，その正規表現にマッチした場合に標準形式のトレースログに出力する内容を指定した，ルールの集合である．標準形式への変換は，変換ルールに基づき，テキストの変換処理を行う処理である．

標準形式への変換処理に，awk などの既存のテキスト処理用スクリプト言語を使用する方法がある．しかしながら，3.2 節で述べたように，変換元のトレースログが暗に含む情報を出力するために，リソースの属性の初期値や型などの情報を参照したり，リソース毎に属性の値の遷移を監視する必要があるなど，多くの複雑な記述をする必要があり，実用的ではない．

そこで，TLV では，出力する条件の制御を，指定された時刻での特定リソースの有無や数や属性の値，属性値変更イベントであれば変更後の値を，振る舞いイベントであればその引数などを用いて論理式を記述することで行う．論理式の記述は，それらの値を取得できる置換マクロを用いて行い，その際の記法には標準形式を用いる．

変換ルールを 3.2 節で述べた，図 3 の 3 行目の「あるタスクにディスパッチした」というトレースログを標準形式トレースに変換する例で説明する．出力したい標準形式のトレースログは図 5 の 3〜4 行目である．4 行目はディスパッチ先のタスクを実行状態 (RUNNING) とすればよいため，単純な置換で実現可能である．3 行目は，置換マクロを用いて，実行状態 (RUNNING) のタスクが存在するかチェックして，存在すれば，そのタスクの状態が実行可能状態 (READY) となったことを標準形式トレースログとして生成する．

4.3　可視化表示の仕組み

3.3 節の方針に従い，トレースログの内容から可視化表現を決定して表示するまでの流れを抽象化し，可視化表示の仕組みとして本質的な処理を洗い出す．そして，ユーザが，どの処理に，どのようなパラメータを指定すればよいかを判断する．

4.3.1　可視化表示の抽象化

トレースログ可視化ツールにおいて，可視化表示とは，可視化表示項目毎に要求される表現 (以下，可視化表現) を，トレースログの内容に従い時系列の図として画面に描画する処理であると考えられる．ここで，画面への描画は，GUI フレームワークに依存するため，可視化表示の仕組みとしては本質的ではない．そのため，可視化表示の仕組みは，可視化表現を，時間と高さを次元にもつ仮想的な座標に割り当てる処理であると抽象化できる．ここで，可視化表現は，複数の図形で構成されるものとする．図 6 に，抽象的な可視化表示の仕組みを示す．

図形は，楕円や四角などの複数の基本図形で構成されていると考えることができる．このとき，図形を定義する座標系をローカル座標系と呼称する．次に，図形をトレースログの内容に従い配置する座標である，時間と高さを次元にもつ仮想的な座標をワールド座標系と呼称する．ワールド座標系へ割り当てられた図形は，表示開始時刻と単位時間あたりのピクセル数により表示デバイスに表示される．このとき，表示先

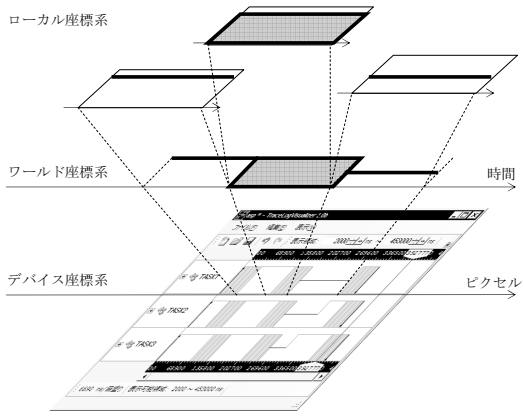

図 6　抽象的な可視化表示の仕組み

の座標系をデバイス座標系と呼称する．単位時間あたりに，何ピクセル数でワールド座標系をデバイス座標系に割り当てるかという処理が，表示上の拡大，縮小の処理となる．

以上の可視化表示の仕組みの抽象化から，ユーザが任意の可視化表示項目を実現するために，汎用的な可視化表示の仕組みに対して与える指示としては，どのような図形を，どの時間領域に割り当てるかということであると考えられる．

時間領域の開始時刻と終了時刻は，イベントを用いて指定することとした．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．

4.3.2　図形の定義と可視化ルール

図形は楕円，多角形，四角形，線分，矢印，扇形，文字列の 7 種類を用意する．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示することができる．

どの図形をどの時間領域に割り当てるか，つまり，図形をワールド座標系のどの領域に割り当てるかの指示は，割り当て先の領域に対して一般的でなければならない．つまり，割り当て先の個々の領域をすべて指定するのではなく，ルールに基づいて割り当てが行われるように指定されるべきである．このルールを可視化ルールと呼称する．

すなわち，可視化は，標準形式トレースログに対して，可視化ルールを適用して，指定された開始イベントと終了イベントに一致するイベントを標準形式トレースログから探し，そのイベントの時刻を割り当て先の時間領域として採用することで，ワールド座標系への割り当てを行う．

4.3.3　図形の定義と可視化ルールの例

図 7 に図形の定義と可視化ルールの例を示す．

図形は，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を runningShape として定義している．

可視化ルール (taskBecomeRunning) では，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指定している．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．

この可視ルールを図 7 右側中央のトレースログに適応すると，トレースログからイベントを抽出して時間領域を決定して，図形をワールド座標にマッピングする．

5　実装

本章では，前章で行った設計を基にした TLV の実装について述べる．まず全体の構成について説明する．次に変換ルールや可視化ルールをどのように形式化したか，また，それらをどのように用いて可視化を行うのかを説明する．

TLV の実装言語は C#3.0 であり，統合開発環境として Microsoft 社の Visural Studio 2008 Professional Edition を用いた．

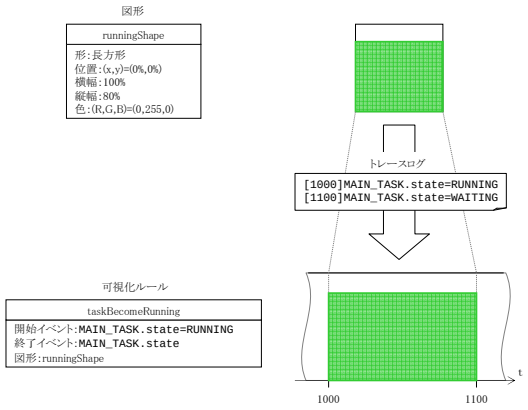

図 7　図形の定義と可視化ルールの例

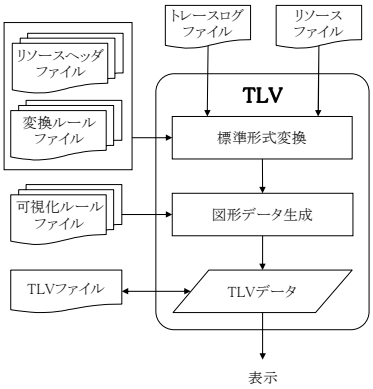

図 8　TLV の構成

5.1　構成

図 8 に TLV の構成を示す．TLV は，6 種の入出力ファイルと 2 つの主たる処理によって構成される．

トレースログを TLV により可視化表示する際，ユーザは，トレースログファイルとリソースファイルを用意する．また，トレースログファイルの形式毎に変換ルールファイルとリソースヘッダファイル及び可視化ルールファイルを用意する．

標準形式変換は，トレースログとリソースファイルを読み込み，トレースログにで指定されたリソースヘッダファイルと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．

図形データ生成は，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．

それぞれの処理で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，2 つの処理を省略して可視化表示することができる．

TLV が扱う 6 種のファイルのうち，トレースログファイル以外は JSON(JavaScript Object Notation) [12] と呼ばれるデータ記述言語を用いて記述する．JSON は，主にウェブブラウザなどで使用される ECMA-262，revision 3 準拠の JavaScript(ECMAScript) と呼ばれるスクリプト言語のオブジェクト表記法をベースとしており，RFC 4627 として仕様が規定されている．JSON の特徴は，構文が単純であることである．そのため，人間にとっても読み書きが容易であり，コンピュータにとっても解析が容易であるため，ユーザの記述コスト，習得コスト，またファイルの解析を行う実装コストを低減することができる．

5.2　トレースログファイル

トレースログファイルは，可視化対象となる環境が出力した，任意の形式のトレースログをファイル化したものである．

標準形式トレースログに変換する元となるトレースログは，トレースログファイルとして読み込む．トレースログファイルはテキストファイルであり，行単位でトレースログが記述されていなければならない．これ以外のシンタックス，セマンティクスに関する制限はない．

5.3　リソースヘッダファイル

リソースヘッダファイルにはリソースタイプの定義を記述する．リソースタイプの定義には，リソースタイプの名前，表示名，リソースタイプの持性，振る舞いを記述する．

図 9 に，ASP カーネルのリソースタイプ `Task` を定義したリソースヘッダファイルの例を示す．属性 (Attribute) としては，ID 番号 (`id`) と状態 (`state`) が定義されている．属性には，型 (VariableType),

```
"Task":{                        /* リソースタイプの名前 */
   "DisplayName":"タスク",       /* 表示名 */
   "Attributes":{                /* 属性の定義始まり */
     "id":{                      /* 属性名 */
       "VariableType"  :"Number",/* 属性の型 */
       "DisplayName"   :"ID",    /* 属性の表示名 */
       "AllocationType":"Static",/* 値は動的か静的か */
       "CanGrouping"   :false /* 同じ属性値同士でグループ */
     },                          /* を構成できるか (GUI 用) */
     "state":{
       "VariableType"   :"String",
       "DisplayName"    :"状態",
       "AllocationType" :"Dynamic",
       "CanGrouping"    :false,
       "Default"        :"DORMANT"
     }
     /* ... 省略 ... */
   },                              /* 属性の定義終わり */
   "Behaviors":{                   /* 振る舞いの定義開始 */
     /* ... 省略 ... */
     "enterSVC":{           /* システムコールの呼び出し */
       "DisplayName":"サービスコールに入る",
       "Arguments":{"name":"String","args":"String"}
     },
     /* ... 省略 ... */
   }
}
```

図 9 ASP カーネルのリソースヘッダファイルの一部

表示名 (DisplayName)，値が動的か静的か (AllocationType)，グループか可能か (CanGrouping) を指定可能である．ID 番号は実行中に変化しないため，AllocationType は “Static” に，状態は実行中に変化するため，“Dynamic” が指定されている．

振る舞い (Behaviors) としては，`enterSVC` が定義されている．振る舞いは，表示名 (DisplayName)，引数 (Arguments) を指定可能である．

5.4 リソースファイル

リソースファイルには，主にトレースログに登場するリソースの定義を記述する．他にも，時間の単位や時間の基数，適用する変換ルール，リソースヘッダ，可視化ルールを定義する．リソースの定義には，名前とリソースタイプ，必要があれば属性の初期値を記述する．

ASP カーネルのリソースファイルの例を図 10 に示す．リソースヘッダとしては，図 9 のリソースヘッダファイルを指定している．リソースとしては，リソースタイプ `Task` の `TASK1` を定義して，属性の初期値を指定している．`TASK1` は，id 番号が 1 であり，初期状態が休止状態 (DORMANT) である．

```
{ "TimeScale" :"us", "TimeRadix" :10,  /* 時刻の単位と基数 */
  "ConvertRules"   :["asp"],              /* 変換ルール指定 */
  "VisualizeRules" :["toppers","asp"], /* 可視化ルール指定 */
  "ResourceHeaders":["asp"],          /* リソースヘッダ指定 */
  "Resources":{                     /* リソースの定義始まり */
    "TASK1":{           /* TASK1 という名前のリソースを定義 */
      "Type":"Task",                      /* リソースタイプ */
      "Color":"ff0000",         /* リソース固有の色 (GUI 用) */
      "Attributes":{                  /* 属性の初期値を指定 */
        "id"    :1,
        "state" :"DORMANT",
        /* ... 省略 ... */
      }
    },
    /* ... 省略 ... */
```

図 10 ASP カーネルのリソースファイルの一部

5.5 変換ルールファイルと標準形式変換

変換ルールファイルには，ターゲットとなるトレースログを標準形式トレースログに変換するためのルールが記述される．

図 11 に，4.2 節で述べた変換ルールを形式化したものを示す．1 行目は，元のトレースログファイルから「あるタスクにディスパッチされた (図 3 の 3 行目)」トレースログを検索するための正規表現であり，マッチした場合に 2〜9 行目で指定される標準形式のトレースログを出力する．3 行目は，出力する条件の制御であり，条件中に`$EXIST{`*resource*`}`という置換マクロを用いてリソース *resource* の有無を判定している．ここでは，実行状態 (RUNNING) のタスクが存在するかチェックして，存在すれば，そのタスクが実行可能状態 (READY) となる標準形式のトレースログ (4 行目) を出力する．一方，7 行目は，1 行目のマッチにより無条件に出力され，ディスパッチ先のタスクの状態を実行状態 (RUNNING) とする標準形式トレースログを出力する．9〜10 行目は，システムコール呼び出しを変換するための変換ルールである．

置換マクロは 6 種類用意されており，特定リソースの有無や数，属性の値を得ることができる．また，置換マクロは，条件判定のときだけでなく，出力する標準形式トレースログの記述にも用いることができる．

- `$EXIST{`*resource*`}`
  指定されたリソース *resource* が存在すれば `true`，存在しなければ `false` に置換する．
- `$COUNT{`*resource*`}`
  指定されたリソース *resource* の数に置換する．

```
"\[(?<t>\d+)\] dispatch to task (?<id>\d+)\.":[
  {
    "$EXIST{Task(state==RUNNING)}":[
      "[${t}]$RES_NAME{Task(state==RUNNING)}.state=READY",
    ]
  },
  "[${t}]$RES_NAME{Task(id==${id})}.state=RUNNING",
],
"\[(?<time>\d+)\] enter to (?<name>(i\w+[_]\w+)) \
                        ( (?<args>.+))?\.?" :[
    "[${time}]$ATTR{CurrentContext.name}.enterSVC(${name}, \
                                                  ${args})"
],
```

図 11　ASP カーネルの変換ルールファイルの一部

```
[1000]TASK(state==RUNNING).enterSVC(act_tsk, tskid=1)
[1005]TASK(id==1).state=READY
[1010]TASK(state==RUNNING).state=READY
[1010]TASK(id==1).state=RUNNING
```

図 12　図 3 のトレースログの図 11 の変換ルールによる標準形式への変換結果

- `$ATTR{`*attribute*`}`
  指定された属性 *attribute* の値に置換する．
- `$RES_NAME{`*resource*`}`
  指定されたリソース *resource* の名前に置換する
- `$RES_DISPLAYNAME{`*resource*`}`
  指定されたリソース *resource* の表示名に置換する．
- `$RES_COLOR{`*resource*`}`
  指定されたリソース *resource* の色に置換する．

標準形式変換は，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することで行われる．図 12 に図 3 とトレースログを図 11 の変換ルールによりを標準形式トレースログに変換した例を示す．

5.6　可視化ルールと図形データ作成

図形と可視化ルールの定義は可視化ルールファイルに記述する．図 13 に ASP カーネルの可視化ルールファイルの一部を示す．2 行目から 10 行目までが図形の定義であり，それ以降が可視化ルールの定義である．

図形としては，長方形の `runningShapes` が定義されている．可視化ルールは，RTOS のタスクの状態表

```
{ "toppers":{
    "Shapes":{                          /* 図形の定義始まり */
      "runningShapes":[{
          "Type":"Rectangle",           /* 形状の指定（長方形）*/
          "Size":"100%,80%",            /* 大きさの指定 */
          "Pen":{"Color":"ff00ff00","Width":1},/* 線種の指定 */
          "Fill":"6600ff00"             /* 塗りつぶしの指定 */
        }],
      /* ... 省略 ... */
    },                                  /* 図形の定義終わり */
    "VisualizeRules":{        /* 可視化ルールの定義始まり */
      "taskStateChange":{
        "DisplayName":"状態遷移", /* 可視化ルールの表示名 */
        "Target":"Task",         /* 適用するリソースタイプ */
        "Shapes":{                /* 表示項目の指定 */
          "stateChangeEvent":{
            "DisplayName":"状態", /* 表示項目の表示名 */
            "From":"${TARGET}.state", /* 開始イベント */
            "To"  :"${TARGET}.state", /* 終了イベント */
            "Figures":{               /* 表示する図形 */
              "${FROM_VAL}==RUNNING" :"runningShapes",
              "${FROM_VAL}==READY":"readyShapes"
              /* ... 省略 ... */
            }
          },
          /* ... 省略 ... */
        }
      },                        /* 可視化ルールの定義終わり */
      /* ... 省略 ... */
```

図 13　ASP カーネルの可視化ルールファイルの一部

示のための `taskStateChange` が定義されている．このルールは，開始イベントと終了イベントとして，リソースタイプであるタスク（`Task`）の属性の値変更イベントを用いることを `Target` で指定している．そして，ワールド座標系に割り当てる図形を，変更後の属性の値（タスクの状態）によって場合分けしており，実行状態 (RUNNING) の場合は，`runningShapes` を用いる．

図形データ生成は，4.3 節で述べたとおり，標準形式トレースログの内容と図形と可視化ルールの定義に従い，ワールド座標系に図形を割り当てる．

6　TLV のユーザーインタフェース

TLV のユーザーインタフェースについて解説する．図 14 に，ASP カーネルが出力したトレースログを，TLV を用いて可視化表示した結果を示す．

6.1　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（図中の中央上側のペイン）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

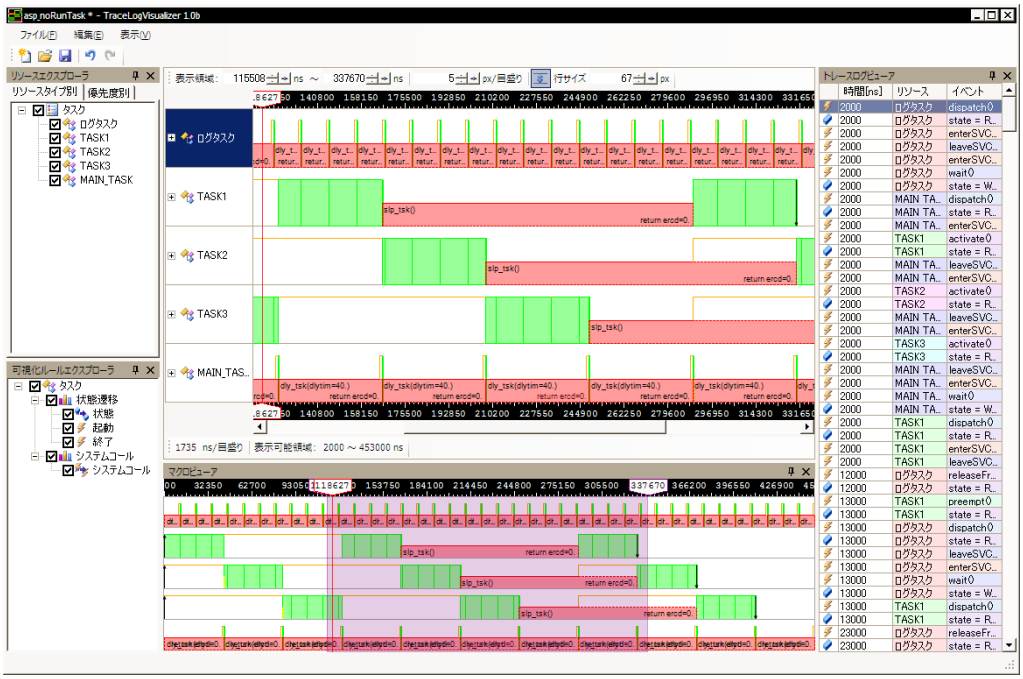

図 14　TOPPERS/ASP カーネルのトレースログの可視化表示

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る．

6.2　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（図中の一番下のペイン）．

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，それに対応し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．

6.2.1 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（図中の右側のペイン）．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．

6.2.2 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（図中の左側のペイン）．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．

7 評価

汎用性の評価として，トレースログの形式が異なる RTOS(ASP カーネル) 及びソフトウェアコンポーネント (TECS) のトレースログの可視化がそれぞれ可能であるか評価する．次に，拡張性の評価として，ASP カーネルをマルチプロセッサ拡張した FMP カーネルの可視化を ASP カーネル用のファイルをベースに拡張して可視化表示が可能か評価する．

7.1 ASP カーネルの可視化表示

図 14 に，ASP カーネルが出力するトレースログを，TLV を用いて可視化表示した結果を示す．タスクの状態遷移やシステムコールの出入り，その際の引数，返値を可視化表示できていることがわかる．

変換ルールファイルは約 136 行であり，可視化ルールファイルは可視化ルールが 346 行，図形の定義が 198 行であった．また，リソースヘッダファイルは 412 行であった．標準形式への変換処理に要した時間は，トレースログファイルが 353 行の場合で約 2 秒であった．

7.2 ソフトウェアコンポーネント (TECS) の可視化表示

組込みコンポーネントシステム TECS(TOPPERS Embedded Component System) [15] のトレースログを用いた．図 15 に，組み込みコンポーネントシステムである TECS のトレースログを TLV 用いて可視化表示した結果を示す．図 15 より，コンポーネントの呼び出し関係が可視化表示できていることがわかる．このとき，変換ルールは 15 行であり，可視化ルールは 88 行であった．また，リソースヘッダファイルは 47 行であった．

7.3 FMP カーネルの可視化表示

次に，ASP カーネルをマルチプロセッサ用に拡張した FMP カーネルが出力するトレースログの可視化表示を行った．FMP カーネルが出力するトレースログの形式は，図 1 に示すように，ASP カーネルが出力するトレースログの形式に，プロセッサの番号を付加した形式になっている．また，FMP カーネルの可視化表示項目は，ASP カーネルの可視化表示項目に加え，そのタスクを実行しているプロセッサを背景色で区別するように拡張したものが要求される．そのため，ASP カーネル用の変換ルールと可視化ルールをベースに拡張を行った．

その結果，変換ルールファイルは 160 行，可視化ルールファイルは可視化ルールが 250 行，図形定義が 30 行，リソースヘッダファイルは 439 行となった．可視化ルールファイルは ASP カーネル用のファイル

図 15　ソフトウェアコンポーネント（TECS）のトレースログの可視化表示

に対する追加ルールとなっている．

図 16 に，FMP カーネルが出力するトレースログを，TLV 用いて可視化表示した結果を示す．タスクの状態遷移を表示している箇所に，背景色で実行コアを区別し表現できていることがわかる．これにより，拡張性が確認できた．

8　おわりに

本論文では，トレースログ可視化ツールである TLV について，背景と既存ツールの問題点，それを解決するための要件，要件を満たすための方法とその実装，評価について述べた．

今後の課題としては，現状の変換ルールや可視化ルールの記述では実現できない，計算の必要な可視化表示項目について検討することが挙げられる．例えば，RTOS における CPU 使用率やタスクのプロセッサ占有率の可視化表示等である．これらの可視化項目を扱うには，可視化ルールを現状の単純なイベントの指定ではなく，イベントの状態遷移で指定できるようにする方法や，変換ルールをスクリプト言語化する方法などが考えられる．

参 考 文 献

[1] TraceLogVisualizer(TLV) http://www.toppers.jp/tlv.html.
[2] TOPPERS/FMP kernel, http://www.toppers.jp/fmp-kernel.html.
[3] JTAG ICE PARTNER-Jet, http://www.kmckk.co.jp/jet/.
[4] WatchPoint デバッガ, https://www.sophia-systems.co.jp/.
[5] QNX Momentics Tool Suite, http://www.qnx.co.jp/products/tools/.
[6] eBinder, http://www.esol.co.jp/embedded/ebinder.html.
[7] 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.
[8] Mathieu Desnoyers and Michel Dagenais, "OS Tracing for Hardware, Driver and Binary Reverse Engineering in Linux," CodeBreakers Journal Article, vol.4, no.1, 2007.
[9] R. McDougall, J. Mauro, and B. Gregg.: Solaris(TM) Performance and Tools: DTrace and MDB Techniques for Solaris 10 and OpenSolaris. Pearson Professional, 2006.
[10] OpenSolaris Project: Chime Visualization Tool for DTrace, http://opensolaris.org/os/project/dtrace-chime/.
[11] RFC3164 The BSD syslog Protocol, http://www.ietf.org/rfc/rfc3164.txt.

]

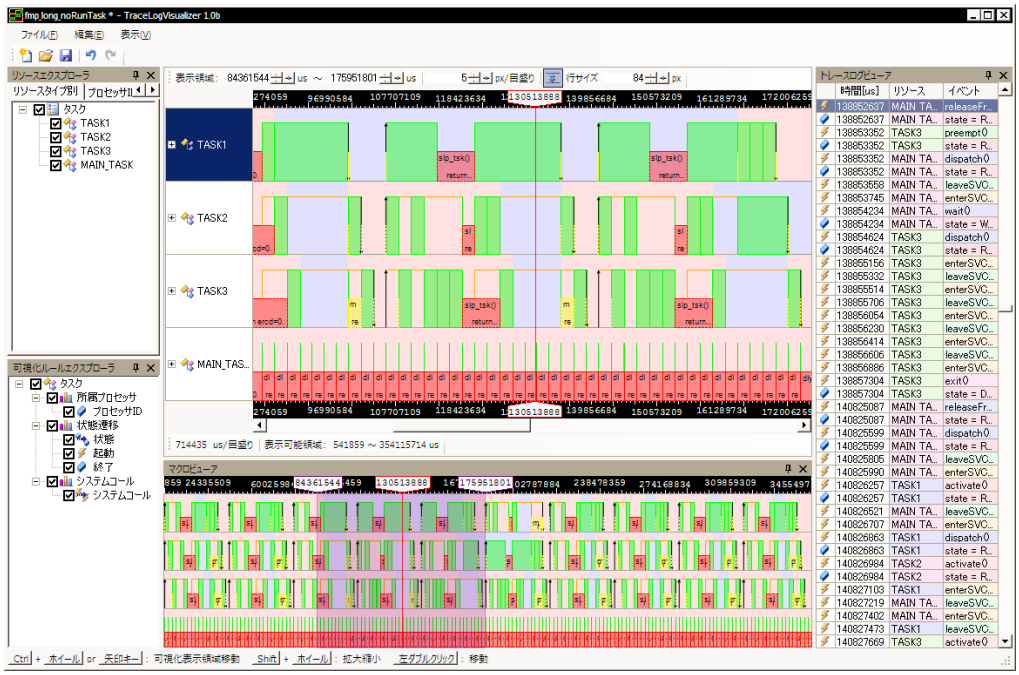

図 16　マルチプロセッサ対応 RTOS(TOPPERS/FMP カーネル) のトレースログの可視化表示

[12] RFC4627 The application/json Media Type for JavaScript Object Notation (JSON), http://tools.ietf.org/html/rfc4627.

[13] TOPPERS ASP Kernel, http://www.toppers.jp/asp-kernel.html.

[14] 坂村健監修, 高田広章編, “μITRON4.0 仕様 Ver.4.02.00,” トロン協会, 2004.

[15] Takuya Azumi, Masanari Yamamoto, Yasuo Kominami, Nobuhisa Takagi, 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.