OJL 報告書

トレースログ可視化ツール（TLV）に対するイベント検索機能の追加

350902054 鵜飼　真充

名古屋大学大学院 情報科学研究科
情報システム学専攻

2011 年 2 月

概要

目次

1 **はじめに** 3

2 **トレースログ可視化ツール** (TLV) 5
- 2.1 TLV における汎用性と拡張性 . . . 5
- 2.2 標準形式トレースログ . . . 5
  - 2.2.1 トレースログの抽象化 . . . 5
  - 2.2.2 標準形式トレースログの定義 . . . 6
  - 2.2.3 標準形式トレースログの例 . . . 7
- 2.3 トレースログの可視化 . . . 8
  - 2.3.1 可視化プロセス . . . 8
  - 2.3.2 変換ルール . . . 9
  - 2.3.3 可視化ルール . . . 9
  - 2.3.4 具体的な可視化例 . . . 9
- 2.4 TraceLogVisualizer のその他の機能 . . . 11
- 2.5 本 OJL で追加した機能 . . . 13

3 **実績** 14
- 3.1 開発プロセス . . . 14
- 3.2 開発スケジュール . . . 15
- 3.3 チケット終了率 . . . 19
- 3.4 発表実績 . . . 19
- 3.5 活用事例 . . . 20

4 **要求抽出** 21
- 4.1 ユーザからの要求抽出 . . . 21
- 4.2 要求された機能 . . . 21
  - 4.2.1 検索機能 . . . 21
  - 4.2.2 統計情報の表示 . . . 21
  - 4.2.3 標準形式ログの読込み機能 . . . 22
  - 4.2.4 凡例表示機能 . . . 22
- 4.3 本 OJL で対応する要求 . . . 23

5 **イベント検索機能** 24
- 5.1 実現可能性の調査 . . . 24
- 5.2 開発プロセス . . . 24

5.3 簡易イベント検索機能 25
5.3.1 機能仕様 25
5.3.2 ユースケース 26
5.3.3 設計 28
5.3.4 実装画面 30
5.4 詳細イベント検索機能 30
5.4.1 機能仕様 30
5.4.2 ユースケース 32
5.4.3 設計 34
5.4.4 実装画面 39
5.5 考察 39

6 **おわりに** 41
6.1 まとめ 41
6.2 所感 42

1 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ガ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 OJL 1 期生の後藤を中心に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 [1]．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．

TLV はリリースされた後も、OJL 2 期生の水野、柳澤が開発を引き継ぎ、新たな機能追加、保守が行われた [cite]. 主な追加機能としては、CPU 利用率といった、複数のログを解析しなければならない複雑な可視化を実現するスクリプト拡張機能、デバッグを効率化するために可視化図形の中に任意の文字列を挿入可能にしたアプリログ機能があげられる．また、TLV の処理の遅さを改善するためのリファクタリングも行われた．

本 OJL でも引き続き TLV への機能追加と保守を行った．ユーザから TLV への要求を抽出し，特に要望の強かったイベントの検索機能の実装を行った．

TLV ではタスクや周期ハンドラといったリソースの状態遷移や強制終了といったイベントを時間ごとに可視化するため，視覚的にリソースの挙動が分かるようになっている．しかし，それでもログが長くなると全範囲をチェックする負担は大きい．そのため，検索機能を用いることでデバッグの負担が軽くなる．

また，これまで TLV には変換ルール，可視化ルールの誤りを詳細に伝えられる検証機能が存在していなかった．そのため本 OJL において検証機能の導入を検討した．しかし，十分な能力を持ったライブラリを見つけることができず，一から開発するのはコストが高いと判断したため、導入を見送った．

最後に本報告書の構成を述べる．2 章では TLV の設計思想とその機能を述べる．3 章では本 OJL の開発プロセスやスケジュール管理といったプロジェクト管理について述べる．4 章ではユーザからの要求の詳細について述べる．5 章では検索機能について述べる．6 章ではルール検証機能について述べる

2 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可能になる．

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

TLV では任意のトレースログを扱うために，全てのログを TLV の定める標準形式トレースログに変換してから可視化処理を行う．全てのログに対応できる標準形式を定めるにあたり，トレースログの性質を調べ，抽象化を行う必要があった．次に示す 6 項目が TLV において抽象化されたトレースログである．

- **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，属性を持つ．
- **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

標準形式トレースログは，前小節で抽象化したトレースログを形式化したものである．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いる．正規表現はスラッシュ記号 (/) で挟む．

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．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．

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．

```
Time = /[0-9a-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．この定義により，時刻には，2 進数から 36 進数までを指定できる．

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，イベントは次のように定義されている．

```
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 通りの指定が可能である．

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．AttributeCondition は属性条件指定記述である．これは次のように定義される．

表 2.1　標準形式トレースログの例

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

```
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可能である．

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，次のように定義されている．

```
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されている．

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

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．

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 トレースログの可視化

TLV においてトレースログを可視化するまでには、様々な処理やデータが必要となる．本節では，まず TLV の可視化プロセスについて述べ，各ステップで使用されるデータについて言及する．最後に可視化処理の具体例を示す．

2.3.1 可視化プロセス

TLV においてトレースログを可視化するまでのプロセスを図 2.1 に示す．

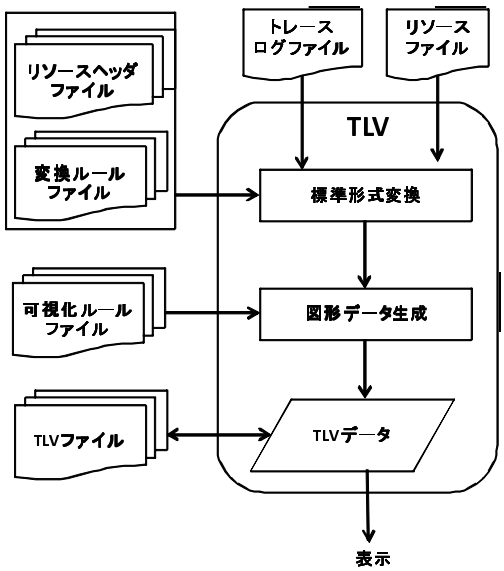

図 2.1　TLV における可視化プロセス

トレースログファイルとリソースファイルを TLV に入力すると，リソースヘッダファイルと変換ルールファイルを用いて標準形式トレースログへの変換が行われる．リソースファイルは，入力したトレースログの中の可視化したいリソースについて，名称やその他の属性を定義したものである．また，リソースヘッダファイルではリソースタイプが定義されている．

続いて，標準形式トレースログに対して可視化ルールファイルの定義に従い，表示する図形データを作成する．この図形データは標準形式トレースログとともに TLV データとして保存され，次回以降に同じログを可視化する際に使用することで可視化の高速化が実現される．

最後に，生成された図形データをもとに TLV 上で可視化表示がなされ，可視化処理のプロセスが終了する．

2.3.2 変換ルール

変換ルールは，トレースログを標準形式に変換する際の変換規則を示したものであり，各規則はJSON 記法で記述される．TLV の利用者は，自分が使用したい形式のログを標準形式に変換するための規則を記述することになる．なお，ASP，FMP，TEX の出力するトレースログ形式に対する変換ルールは，すでに TLV で作成済みである．図 2.2 に変換ルールの一例を示す．

1 行目の正規表現では，変換を適用するログのパターンを定義している．このパターンにマッチしたログがあると，TLV はその後に続く正規表現の形へ変換を行う．変換語のログは標準形式トレースログに則ったログである．

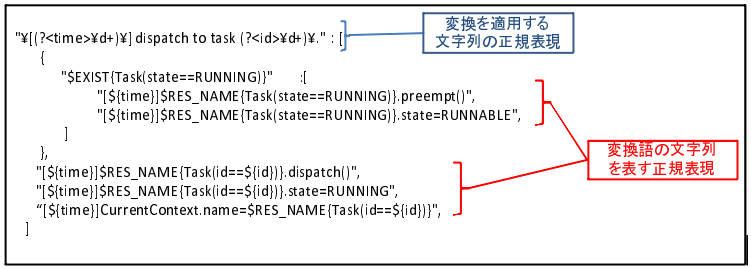

図 2.2　変換ルールの一部

2.3.3 可視化ルール

可視化ルールは，標準形式トレースログを図形データに落とし込むための規則を定義したものであり，変換ルールと同様に JSON 記法で記述される．TLV 利用者は自分が図形表示させたいイベントについて規則を記述する．ASP，FMP，TEX で表示できると便利であると予想される図形については既に TLV で定義済みである．図 2.3 に変換ルールの一例を示す．

TLV では，図形データを可視化ルールとイベントをキーにして管理している．1 行目の”taskStateChange” が可視化ルール名，5 行目の”stateChangeEvent” がイベント名である．そして次の 2 行でイベントの開始と終了を示すログパターンを定義した部分であり、マッチするログを発見すると，そのログのイベントが次行以降で定義されている図形と対応付けられる．ここで定義されている図形は “runningShapse” と”runnableShapes” であり，これらの形やサイズといった図形データも可視化ルールの中で定義する．

2.3.4 具体的な可視化例

ここまでに可視化プロセス，変換ルール，可視化ルールについて述べたが，実際にそれらがどのようにトレースログに適用されながら可視化が行われるかの具体例を図 2.4 に示す．

時刻 1000 と 1100 に発生したイベントを記録した任意形式のトレースログを TLV に入力するこ

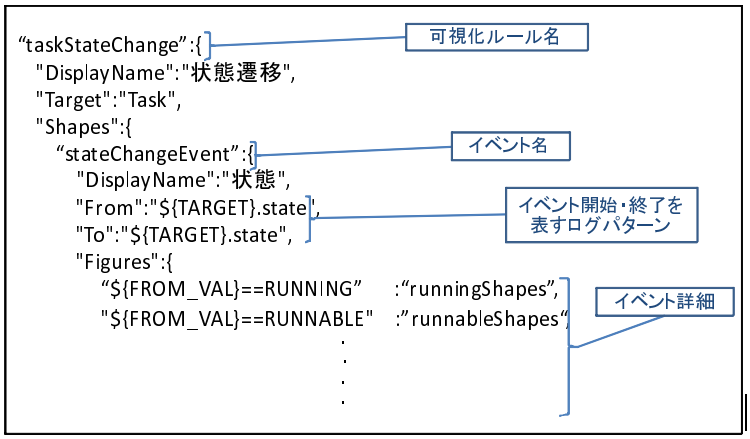

図 2.3　可視化ルールの一部

とを考える．TLV はこのログを標準形式に変換するためのルールを変換ルール群から探して適用する。時刻 1000 のログには ConverRule 内の最初の変換規則が適用される．同様に時刻 1100 のログには 2 番目の規則が適用され，標準形式トレースログへと変換される．次に，TLV は標準形式トレースログに対しては可視化ルールを適用し，図形データを生成する．今回は”taskStateChange” ルールが適用できるため，そのルールで示されている図形 “runningShape” の生成が行われ，最終的に図形としてタイムライン上で表示する．

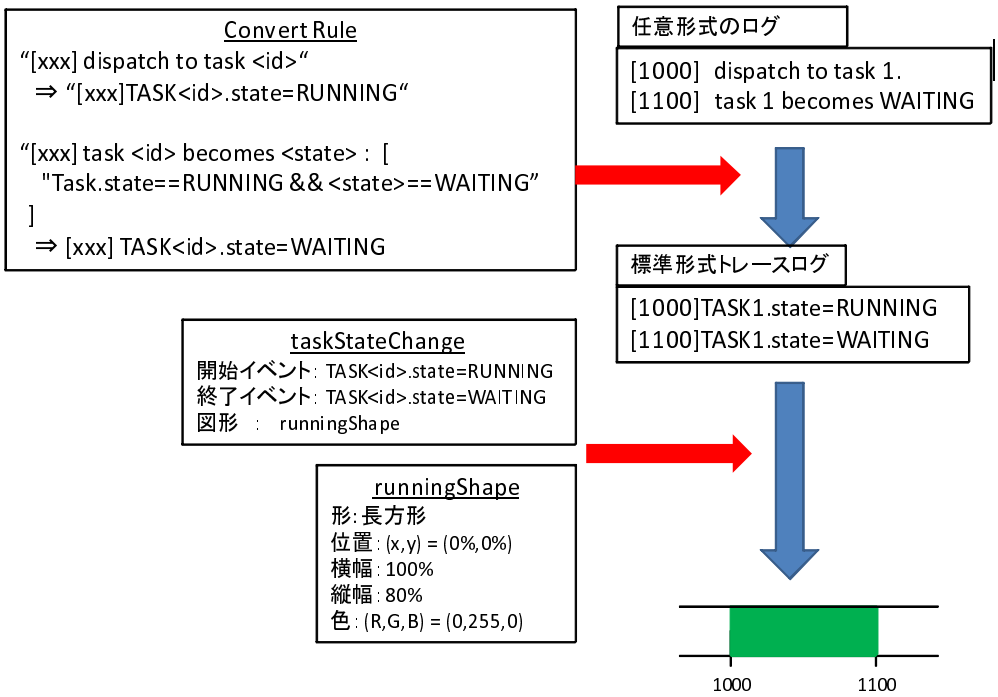

図 2.4　可視化例

以上のような可視化を入力された全てのログに対して適用することで，TLV は最終的に図 2.5 に示すように全区間の可視化を完了させる．

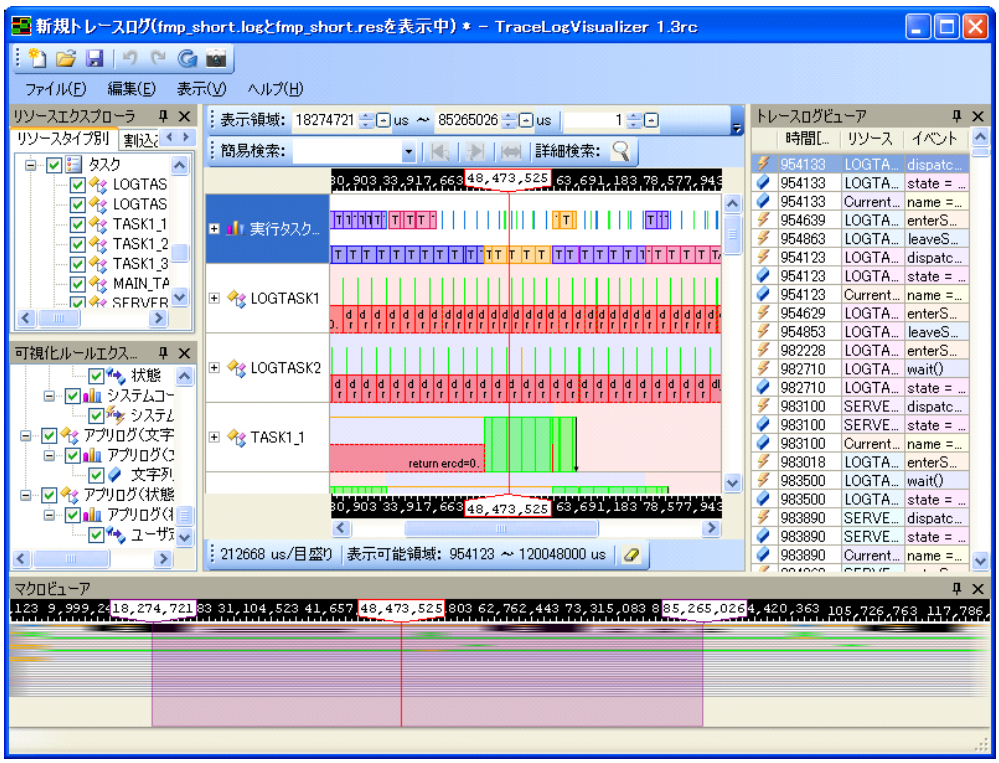

図 2.5　可視化終了後の TLV

2.4　TraceLogVisualizer のその他の機能

TLV に実装されている可視化処理以外の機能を以下に示す．

- **マーカー**：
  図 2.5 の中央部をタイムラインと呼ぶ．タイムライン上の任意の位置にマーカーと呼ぶ目印を配置することができ，注目すべきイベントにマーカーを置くことで解析の補助となる．
- **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が行える．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．
- **マクロ表示**：
  タイムライン上では，ある一定時間の範囲だけが可視化表示されている．マクロ表示機能は現在の表示範囲がトレースログ全体のうちのどの部分かを知るための機能である．マクロビューアにトレースログ全体が可視化されており，現在表示している範囲に薄いマスクをか

ぶせることで，一目で表示箇所が分かるようになっている．

- **トレースログのテキスト表示**：
  図形表示だけでは何が起きているか分かりづらいことがある．そのため TLV では標準形式に変換されたトレースログをトレースログウィンドウに表の形で表示させている．
- **可視化表示項目の表示/非表示の切り替え**：
  可視化された図形の表示/非表示を TLV 実行中に切り替える機能である．リソースエクスプローラでリソースを選択した場合は，タイムライン上の該当リソースの行が消える．可視化ルールウィンドウでルールを選択した場合は，そのルールによって可視化される該当リソースの図形が非表示になる．
- **アプリログ機能**：
  タイムライン上に任意の文字列を書き込む機能．書き出したい文字列を変換前のトレースログの中に所定の形式で埋め込むことで，指定したリソース行の指定した時刻に文字列が書き込まれる．
- **スクリプト拡張機能**：
  標準形式変換の際に変換ルールを用いず，利用者が独自のスクリプトで標準形式変換を可能にする機能．利用者が慣れ親しんだ言語で変換処理を記述できる．また，スクリプトに CPU 利用率などの統計情報を集計するコードを記述することで，それをタイムライン上にグラフとして表示するという複雑な可視化も可能である．

2.5　本 OJL で追加した機能

本 OJL で TLV に追加した機能について述べる。

■イベント検索機能

TLV を用いて長いログを解析する際，特定のイベントの発生時刻を様々な条件を考慮しつつ検索できる機能が必要であるという要求が利用者から得られた．そこで本 OJL では，TLV にイベント検索機能を実装した．詳細は 5 章で述べる．

■スクリーンショット機能

利用者からの要求でタイムラインだけのスクリーンショットを撮ることができる機能が欲しいという要求が出たため、本 OJL で実装を行った．

■リソース状態の追加

既存の可視化ルールに対して，「ディパッチ禁止状態」，「CPU ロック状態」，「割り込み禁止状態」を可視化するためのルールを追加した．図 2.6 におけるタイムライン上の丸枠で囲まれた 3 つの図形が今回新たに追加した図形である．

図 2.6　追加図形

■JSON Validator（開発途中）

現状の TLV には，ルールファイルに JSON 記述違反がある場合にも詳しいエラー内容を利用者に通知する機能を持っていない．そのため，利用者から JSON 検証機能の要求があった．そこで本 OJL では，Json.NET[参照] が提供する JSON パースライブラリを用いてルールファイルの検証機能の開発に取り組んだ．しかし，ライブラリで使用できる JSON スキーマでは，ルールファイルの検証に対して能力不足であることが判明したため，現在は開発を一時中断している状況である．

3　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る．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，生産現場で行われている実際の開発に関わることにより，納期，予算といった制約を踏まえたソフトウェア開発を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 2 名と教員 2 名がプロジェクトマネージャを務め，筆者含む学生 5 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．

■開発フェーズ　TLV の開発は OJL 一期生から現在まで継続して行われている。半年を 1 つの開発フェーズと規定し、各フェーズにおいて TLV に必要な機能の実装を行うというスパイラルモデル型の開発形態をとってきた。図 3.1 に各フェーズの作業内容と、各フェーズの担当者を示す。

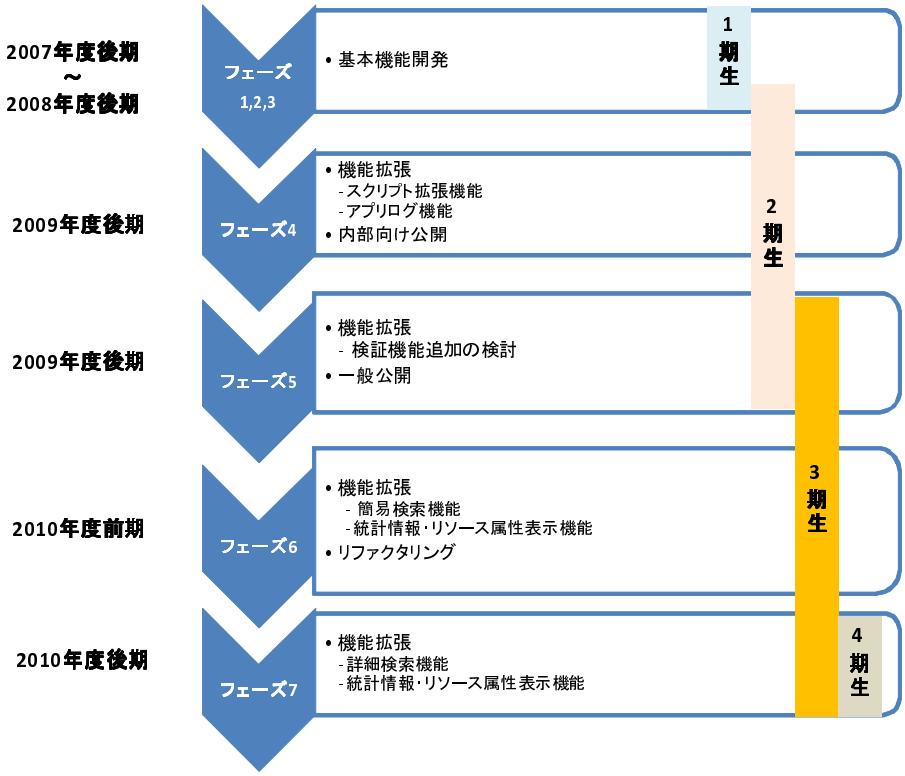

図 3.1　開発フェーズ

■本 OJL 以前に実施されたフェーズ

フェーズ 1 からフェーズ 3 にかけては、一期生、二期生より TLV のプロトタイプ開発が行われた [1]。

フェーズ 4 では、二期生によってスクリプト拡張機能 [2]、アプリログ機能 [3] が TLV に追加され、TLV の TOPPERS 内部向け公開が行われた。

■本 OJL で実施されたフェーズ

フェーズ 5 ではルール検証機能の実現可能性の調査と、TLV のリファクタリング案の検討を行った。また、TLV を使用したユーザから要求抽出を行い、TLV に追加すべき機能を検討した。

フェーズ 6 では、TLV のリファクタリングを行った。また、ユーザからの要望が高かったイベント検索機能、統計情報・リソース属性表示機能の開発に着手した。

フェーズ 7 では、フェーズ 6 に引き続きイベント検索機能の開発と、統計情報・リソース属性表示機能の開発を行った。

3.2 開発スケジュール

本 OJL で実施したフェーズ 5, フェーズ 6, フェーズ 7 の開発スケジュールについて述べる。

■フェーズ 5

期間： 2009 年度後期

実施内容：

- TLV のコードリーディングを行った
- ルール検証機能 JSON Validator の開発を通して、C ＃と TLV の学習を行った
- 標準形式変換の高速化を検討した
- フェーズ 4 までに累積していた不具合・要望への対処を行った
- ユーザからの要求抽出を行った

スケジュール：

フェーズ 5 のスケジュールを図 3.2 に示す。開発に参加してから約 1 月半にわたり、TLV の開発言語である C ＃をルール検証機能 JSON Validator の開発を通して学習した。それと並行する形で TLV のコードリーディングを行い TLV に関する知識を深めた。12 月からは、より深い部分のコードリーディングを行いつつ、可視化速度のボトルネックになっていた標準形式ログへの変換の高速化を検討し、リファクタリング案を作成した。また 12 月半ばから、累積していた不具合・要望への対処を行った。

成果：

JSON Validator は、開発に使用していた JSON パース用のライブラリに能力不足が発見されたため、スケジュールの都合もあって開発を中断することになった。しかし、C ＃の学習、検証機

能の開発に必要な機能の洗い出しという面で成果があった。

また、6 つのクラス、合計 1590 行を対象とする TLV のコードリーディングによって、可視化処理の根幹部分である標準形式変換、図形変換の処理について理解が深まった。さらに標準形式変換の処理にたいするリファクタリング案を作成した。

図 3.2 には載せていないが、2 月中旬に TLV ユーザから TLV に必要な機能の聞き取りを行った。その結果、イベントの検索機能、統計情報・リソース属性情報機能が必要であるという知見を得られた。

フェーズ 5 における不具合・要望への対処は、プロジェクト全体で 22 個、そのうち筆者自身の対処数は 3 個であった。

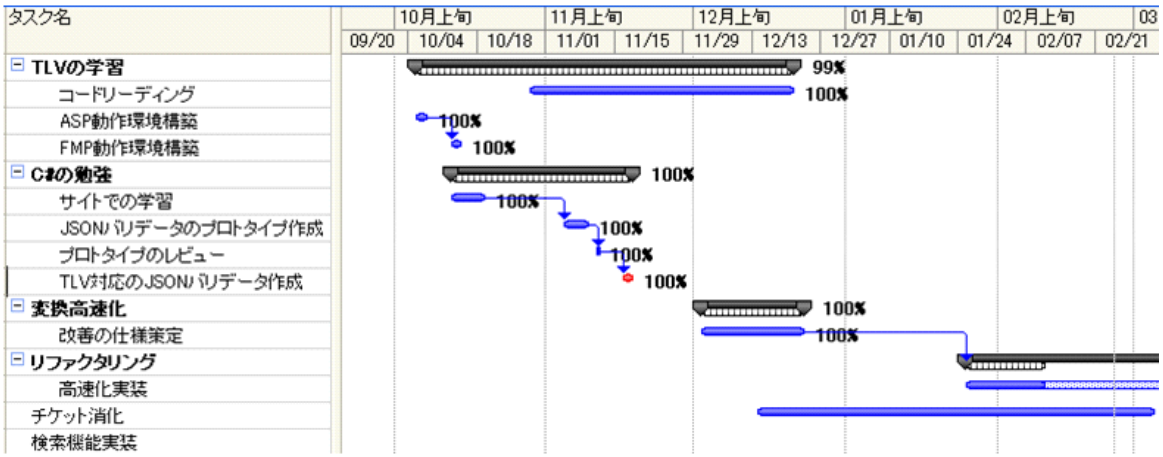

図 3.2　フェーズ 5 のスケジュール

■フェーズ 6

期間：　2009 年度後期

実施内容：

- TLV の GUI 部分に関するコードリーディングを行った
- イベント検索機能の一つである簡易検索機能を実装した
- リリース作業を担当した
- フェーズ 5 までに累積していた不具合、要望への対処を行った

スケジュール：

フェーズ 6 のスケジュールを図 3.3 に示す。フェーズ 6 ではイベント検索機能の開発を行うことになった。そこでまず TLV を構成する GUI 部品について理解を深めるために、GUI に関連する部分のコードリーディングを行った。それと並行してイベント検索機能の実現可能性を調査を行った。3 月終了時点で実現可能であること判断し、その後コードリーディングを継続しながら仕様定義、詳細設計、実装、テストを 6 月下旬までに行い、簡易検索機能を開発した。簡易検索機能に関

しては 5 章で詳細を述べる。また 7 月上旬から下旬にかけて、それまでに累積していた不具合・要望への対処を行った。

成果：

（対象クラス数、合計行数を調べておく）TLV の GUI 部分に関するコードリーディングにより、TLV の構成を理解した。また、イベント検索機能をどう実現するかの知見を得た。

TLV ユーザから要望のあった検索機能の実現を全て実現するには非常に多くの工数がかかり、フェーズ 6 内で終了できないと判断した。そのため、イベント検索機能を簡易検索、詳細検索という二つの機能に分割し、フェーズ 6 では簡易検索を実装した。

4 月にフェーズ 5 での成果をまとめた TLV.1.1.2 のリリースを行った。

フェーズ 6 における不具合・要望への対処は、プロジェクト全体で 10 個、そのうち筆者自身の対処数は 8 個であった。

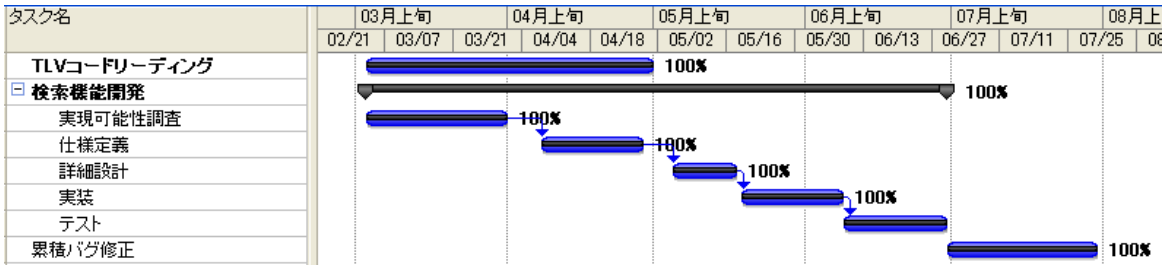

図 3.3　フェーズ 6 のスケジュール

■フェーズ 7

期間：　2009 年度後期

実施内容：

- 詳細検索機能を実装した
- リリース作業を担当した
- フェーズ 6 までに累積していた不具合、要望への対処を行った

スケジュール：

フェーズ 7 のスケジュールを図 3.4 に示す。フェーズ 7 では、フェーズ 6 で実装した簡易検索機能をもとに、より複雑な検索を行う詳細検索機能の開発を行った。8 月上旬から 10 月下旬にかけての 3 か月間で仕様定義、詳細設計、実装、テストまでを行った。詳細検索機能に関しては第 5 章で詳細を述べる。11 月からはそれまでに累積していた不具合・要望への対処を行った。

成果：

詳細検索の一機能として、イベント間のタイミングを検索条件に指定可能なタイミング検索を TLV に実装した。

8 月にフェーズ 6 での成果をまとめた TLV.1.2 のリリースを行った。また、12 月に詳細検索を搭載した TLV1.3rc を TOPPERS 内部向けにリリースした。

フェーズ 7 における不具合・要望への対処は、プロジェクト全体で 14 個、そのうち筆者自身の対処数は 12 個であった。

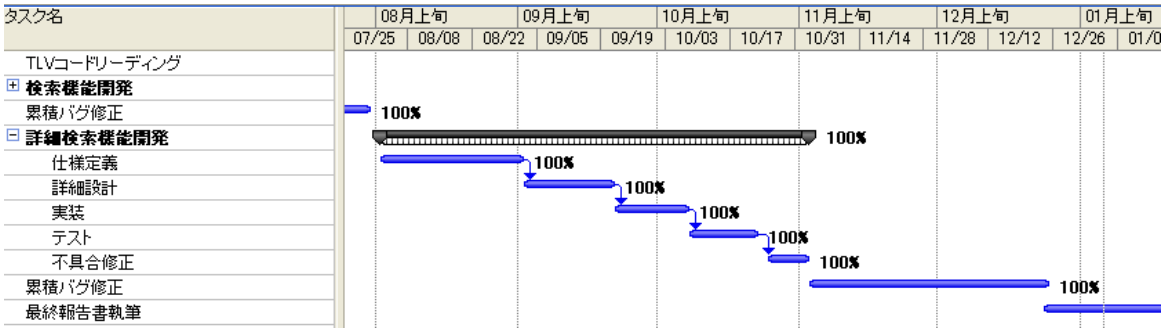

図 3.4　フェーズ 7 のスケジュール

3.3　チケット終了率

本 OJL におけるチケットの発生と消化についてまとめたグラフを図 3.5 に示す。グラフの横軸は期間を表す。横軸の始まりは本 OJL の開始直前である 2009 年 9 月、終わりは OJL 作業の実質的な終了月である 12 月までとなっている。グラフの縦軸はチケット数であり、上側の曲線がチケットの累積数数、下側の曲線がチケットの累積消化数を表わす。

本 OJL 開始から終わりまで、ほとんどの期間でチケットの発生数が消化数を若干とはいえ上回っている。曲線の概形からいうとプロジェクトは発展途上であり、これからもチケットは今のペースで増加することが予想される。そのため、不具合修正、要望への対応を行う時間を今よりも確保し、消化率を上げる必要がある。

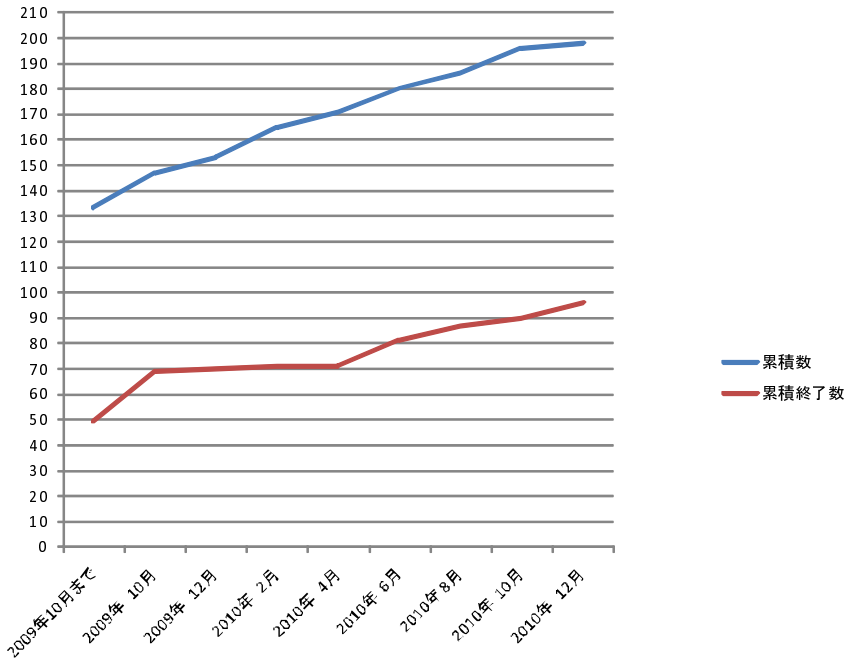

図 3.5　チケット累積件数

3.4　発表実績

■過去の発表実績

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 [4] 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

■本 OJL における発表実績

第 2 回名古屋大学組込みシステム研究センターシンポジウムで[2] ポスター発表を行った。

3.5 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)[3]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っ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/news/sympo2010.html
[3] http://www.nces.is.nagoya-u.ac.jp/

4　要求抽出

4.1　ユーザからの要求抽出

本プロジェクトでは、開発者・利用者用のメーリングリストが用意されており、不具合や要望があればメールですぐに伝えられるようになっている。TLV の開発では、メールで寄せられた意見に基づいて不具合修正、機能の追加に対応してきた。

フェーズ 5 終盤において、それまで着手してきた大型の案件が一通り終了し、新しい機能の追加を検討段階へと進んだ。それまでにユーザから寄せられたいくつかの要望の優先順位を決定すること、新たな機能についての要望を獲得することを目的として、2010 年 2 月 17 日にユーザミーティングを行った。

参加者は、開発者と利用者合わせて約１０人であった。まず TLV の機能概要について利用者に説明し、その後累積している課題を報告した。最後に利用者から追加機能について意見を求めた。

4.2　要求された機能

ユーザから要望があった追加機能について以下で述べる。

4.2.1　検索機能

TLV では各リソースの挙動を様々な形と色の図形で表示する。例えば図 4.1 に示した図形では、緑の四角形がリソースが実行状態であること、赤い直線がタスクが待ち状態であること、黄色い直線がタスクが実行可能状態にあることを表す。図 4.1 は全体として、タスクが実行状態から、待ち状態、実行可能状態を経て、再度実行状態になったことを示している。

図 4.1　図形パターン

TLV はトレースログを可視化することで、デバッグのコストを減らすことを目的としたツールである。しかしログが長くなれば、可視化されていたとしても解析にコストがかかってしまう。そこで、注目するリソースについて図形パターンを指定、もしくはどのような状態になったのかという条件を指定し、画面上の該当する時刻へと即座に移動、もしくは該当する時刻を列挙するという検索機能が欲しいという要望があった。

4.2.2　統計情報の表示

TLV の開発思想は、トレースログを可視化し、ログ 1 行 1 行を読まなくても済むようにすることでデバッグコストを下げるというものであり、ログを図形化するだけの機能しか持たせていな

い。しかし、どの時刻に何が起きているかを表示するだけでなく、どの時刻で、もしくはどの範囲でリソースがどれだけ動いているか、どれだけプリエンプとされたかという統計的な情報を表示できれば、それもデバッグの大きな支援になる。そのため、ユーザの指定した区間における統計情報を表示する機能が欲しいという要望があった。

統計情報として得たいという要望のあったものを以下に示す

- CPU 利用率
- タスクの実行回数
- イベント数
- 起動回数
- アイドル時間
- プリエンプト回数
- ディスパッチ回数
- 起動までの時間
- デッドラインミス回数
- デッドラインまでの時間
- タスク割り当ての不均衡具合

4.2.3 標準形式ログの読込み機能

TLV の処理速度が遅く、可視化に非常に多くの時間がかかってしまうため、高速化してほしいという要望が本ミーティング以前からあった。本 OJL では TLV に対してプロファイリングを行い、処理のボトルネックとなっている部分が標準トレースログへの変換処理であることまで突き止めており、解決のアプローチとして変換処理コードのリファクタリングを行っている。

一方ミーティングにおいて、最初から OS に標準トレースログを出力させるようにし、それを TLV に読込ませるというアプローチが提案された。これならば、そもそも変換処理が発生しないため、可視化までの速度は大幅に向上する。そこで標準形式ログを入力として使用できるようにしてほしいという要望があった。これを実現する場合、TLV の既存プロセスは図 4.2 のようにする必要がある。

4.2.4 凡例表示機能

TLV では、表示される図形が何を意味しているかを伝える機能が存在していない。そのため、TLV に習熟していないユーザにとっては 図 4.1 のような簡単な図形パターンでさえ、そのパターンがどんな意味を持っているかを理解するのが難しい。そこで、ユーザに分かりやすく図形の情報を伝える機能が欲しいという要望があった。

実現方式としては次の 2 つが考えられる。

- 変換ルールファイルと可視化ルールファイルから動的に図形情報を取得し、図形にマウスポ

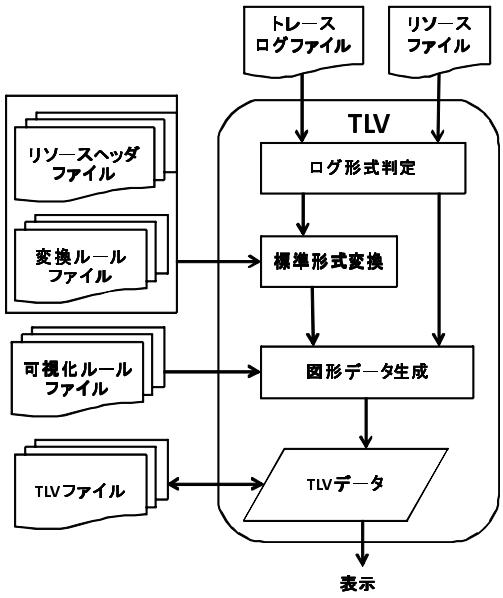

図 4.2　標準形式ログ入力機能を導入した場合のプロセス

インタが当った際に表示する、もしくはサブウィンドウに図形とともに表示する。

- 図形情報ファイルを用意しておき、マウスポインタが当る、もしくはサブウィンドウに図形とともに表示する。

4.3　本 OJL で対応する要求

4.2 節で述べた機能は TLV に全て実装する予定である。しかし OJL には期限が定められているため、優先順位を決定する必要がある。

４つの機能のうち検索機能と統計情報表示機能は、ユーザからの要望が大きかったものである。また、これらは開発工数が大きいと予想される。そのため、これらの機能から開発を始めることが望ましい。

今回の要求抽出はフェーズ 5 終了直前に行われたものである。続くフェーズ 6 では筆者を含めて開発人員は２人であるため、２つの機能をそれぞれが担当する形で開発を行うこととした。筆者は検索機能を担当した。検索機能の詳細については続く 5 章で述べる。

5　イベント検索機能

5.1　実現可能性の調査

要求された検索機能は、リソースの図形パターン、もしくは、リソースの状態を指定してマッチする時刻を探し出すというものであった。そこで、検索機能の実現可能性を調査した。

まずタイムライン上に表示された図形が、プログラム内部でどのような情報とともに保存されているかを調べたところ、次の情報を持っていることが分かった。

1. 図形の開始時刻
2. 図形の終了時刻
3. 図形の形（線、長方形 etc）、高さ、幅、色
4. 図形が表現するイベントの名前（状態遷移、強制終了 etc）
5. そのイベントが属するルールの名前

パターン検索を単純に実装する場合、検索条件に図形の名前を指定できるようにし、該当する図形の描画が始まる時刻をヒットさせるという方法が考えられる。しかし、図形が名前を持っていないため、この方法は不可能である。次に図形の色や形を指定して検索する方法も考えたが、目的のパターンを検索するにはやはり図形の意味を検索条件として指定できた方が良いため、この方法も見送った。パターン検索として一番良いものは、図形をグラフィカルに作成し、そのパターンにマッチする箇所を探すという方法であるが、作成した図形パターンと上記図形情報とのマッチング処理と図形作成機能の実装は非常にコストが高いと予想され、本 OJL では実現が難しいと考えられる。以上の考察から、図形パターン検索に関しては本 OJL では実現不可能と判断した。

一方、リソースの状態を指定するという検索方法については、4 番のイベント名利用すれば実現できそうである。イベントは可視化ルールの中で定義されており、例えば実行中のタスクが変化したことを示す”runningTaskChangeEvent” や、タスクの状態遷移が起きたことを示す”stateChangeEvent” といったものがある。これを検索条件として指定できれば、検索者も直観的に検索条件を作成できると考えられる。そこで本 OJL ではイベント検索機能の実装を進めることにした。

5.2　開発プロセス

検索機能に対するユーザからの要求では、複数のイベントを組合わせた複雑な条件で検索を行いたいというものがあった。例えば「タスク A において起動イベントが発生した時刻のうち、その時刻の 1 秒以内に終了イベントが発生した時刻を検索する」というような、イベント間に時間制約を課した検索が挙げられる。このような複雑な検索を実装するためには、まず 1 つのイベントを検索する機能を実装し、それを複数イベントに拡張するという手順を踏む必要がある。

実現可能性の調査が終了した段階では TLV のコードに習熟できておらず、1 つのイベントの検

索のコード作成はおろか、検索条件入力用のインタフェース、、検索結果の表示といった GUI 関連のコード作成にも多くの時間が必要になると予想された。そのため、検索機能を単独のイベントを検索する「簡易イベント検索」と、複数のイベントを組合わせて検索を行う「詳細イベント検索」とに分割した。フェーズ 6 ではコード学習を行いながら簡易イベント検索を実装し、それをもとにしてフェーズ 7 で詳細検索を実装するという開発プロセスをとることにした。これは小さな機能を徐々に開発しながら製品を組み立てていくスパイラルモデル型の開発プロセスである。

5.3 簡易イベント検索機能

5.3.1 機能仕様

簡易イベント検索の仕様を以下に記述する。

■検索条件

図形データの中に保持されているルール名とイベント名は、可視化ルールファイルから得られた情報である。図 5.1 に可視化ルールファイルの一部を示す。1 行目の”taskStateChange” がルール名であり、4 行目の “stateChangeEvent” がイベント名である。

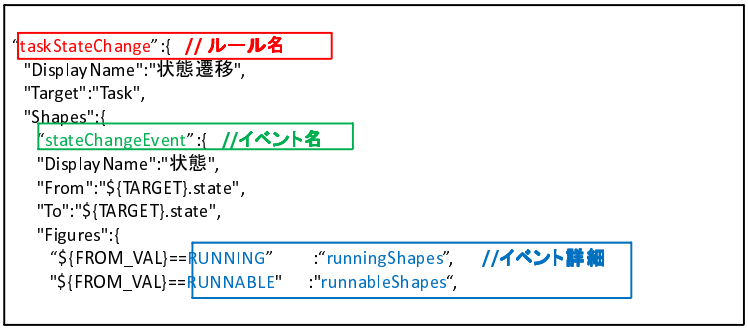

図 5.1 可視化ルールの一部

簡易イベント検索では、例えば「タスク 1 が実行状態になった時刻を探す」というような検索を行うため、検索条件には少なくともイベント名とイベント詳細が必要になる。また、TLV ではイベント名の重複が許されているため、イベント名から目的の図形一意に決定するためにはルール名も検索条件として必要になる。よって検索条件と指定する項目は以下の 4 条件となる。以降では、この 4 条件を**基本条件**と呼ぶ。

- リソース名
- ルール名
- イベント名
- イベント詳細

■検索操作のための GUI

検索にともなう操作としては、検索条件の指定、検索の実行、検索結果の表示があり、これらを行うための GUI が必要になる。簡易イベント検索では 4 つの検索条件指定ボックスと検索実行ボタンだけあればよいため、、別途フォームは用意せず、TraceLogDisplay 上でそれぞれ次のように実現する。

- 検索条件の指定：
  TraceLogDisplayPanel 上のタイムライン上部に条件指定用のコンボボックスを一列に配置する。リソース名だけを指定必須とし、各条件が指定されるごとに次の条件を指定するためのコンボボックスを順次出現させる。
- 検索の実行：
  検索条件指定ボックスの後方に、「後方検索」、「前方検索」、「全体検索」の順にボタンを配置する。検索開始の初期時刻は、ボタンが押された際のタイムライン上のカーソル位置が示す時刻とする。
- 検索結果の表示：
  検索結果はタイムライン上でカーソルを該当時刻へ動かす、もしくはマーカーを出現させることで表現する。後方検索、 前方検索の場合は検索時刻にカーソルを移動させ、同時に画面をカーソルの位置に移動させる。全体検索の場合は全ての検索時刻にマーカーを引き、画面移動はしない。

5.3.2 ユースケース

定義した仕様をもとに、ユーザが簡易イベント検索を行う際のユースケースを図 5.2 に示す。

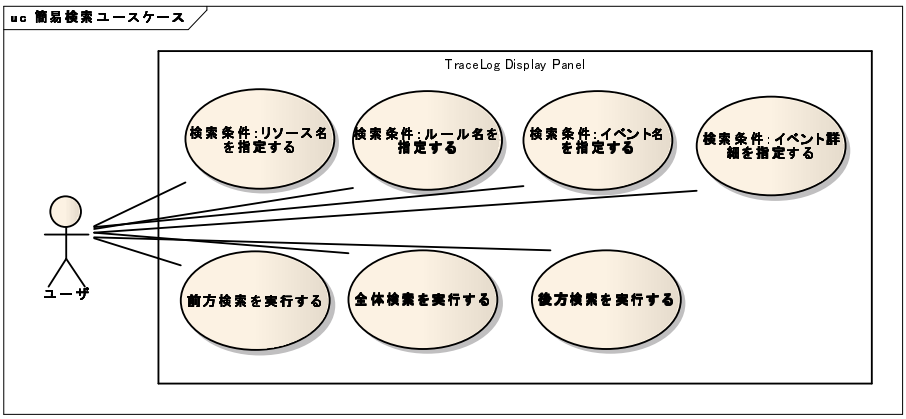

図 5.2　簡易イベント検索のユースケース

- **リソース名を指定する**：
  ユーザがドロップダウンボックスを操作してリソース名を指定する。指定がなされたらルール名用のドロップダウンボックスが出現する。
- **ルール名を指定する**：
  ユーザがドロップダウンボックスを操作してルール名を指定する。指定がなされたらイベント名用のドロップダウンボックスが出現する。
- **イベント名を指定する**：
  ユーザがドロップダウンボックスを操作してイベント名を指定する。指定されたイベントが詳細情報を指定できるものである場合、イベント詳細用のドロップダウンボックスが出現する。
- **イベント詳細を指定する**：
  ユーザがドロップダウンボックスを操作してイベントの詳細を指定する。
- **前方検索を実行する**：
  ユーザが前方検索を実行する。該当する時刻があれば、その時刻が画面の中心なるように画面が動く。また、その時刻にカーソルが出現する。該当時刻がない場合はその旨を伝えるポップアップウィンドウが出現する。
- **後方検索を指定する**：
  ユーザが後方検索を実行する。該当する時刻があれば、その時刻が画面の中心なるように画面が動く。また、その時刻にカーソルが出現する。該当時刻がない場合はその旨を伝えるポップアップウィンドウが出現する。
- **全体検索を実行する**：
  ユーザが後方検索を実行する。該当する時刻があれば、全ての該当にマーカーが出現する。該当時刻がない場合はその旨を伝えるポップアップウィンドウが出現する。

5.3.3　設計

■クラス設計

簡易イベント検索は、1 つのイベントを検索する機能であり、複数のイベントを組み合わせて検索する詳細イベント検索の基礎となる。そのため、簡易イベント検索のクラス設計で最も重要なことは、可能な限り詳細検索イベント機能を低コストで実装できるようにすることである。これを踏まえて図 5.3 のように簡易イベント検索を実現するクラスを設計した。

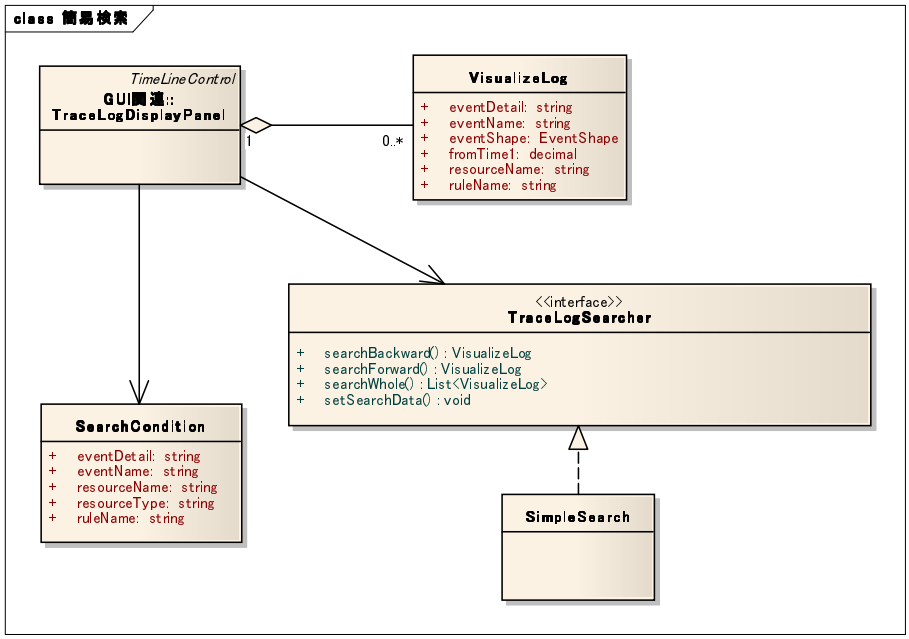

図 5.3　簡易イベント検索を実現するクラス

SearchCondition クラスは、基本条件を保持するクラスとして設計した。詳細イベント検索の実装の際に検索条件が増えた場合は、このクラスに条件を追加する形で拡張する。

詳細イベント検索では、様々な検索方法があると予想される。例えば、今回実装するイベント間のタイミングを考慮した検索方法がある。また、イベントの発生回数を考慮した検索方法であったり、イベントの継続時間を指定する検索などがあげられる。こうした検索方法を 1 つのクラスで実現しようとすると、クラスの責務が重くなりすぎるため、検索方法ごとに検索クラスを用意することが望ましい。さらに、ユーザに検索方法を指定させるなど、検索方法が動的に決定されるような仕組みにする場合は、検索処理と検索クラスの生成を分離できるとよい。そのため、インタフェース TraceLogSearcher を用意し、全ての検索処理クラスが実装するようにする。このようにすることで、生成された検索クラスを意識することなく検索処理が行える。

■シーケンス

前方検索が実行された際に、各クラスがどのように動作するかを図 5.4 に示す。なお、シーケンスの解説のために、各メッセージの名前は実際のメソッド名ではなく、そのメッセージによって起きる動作内容としてある。

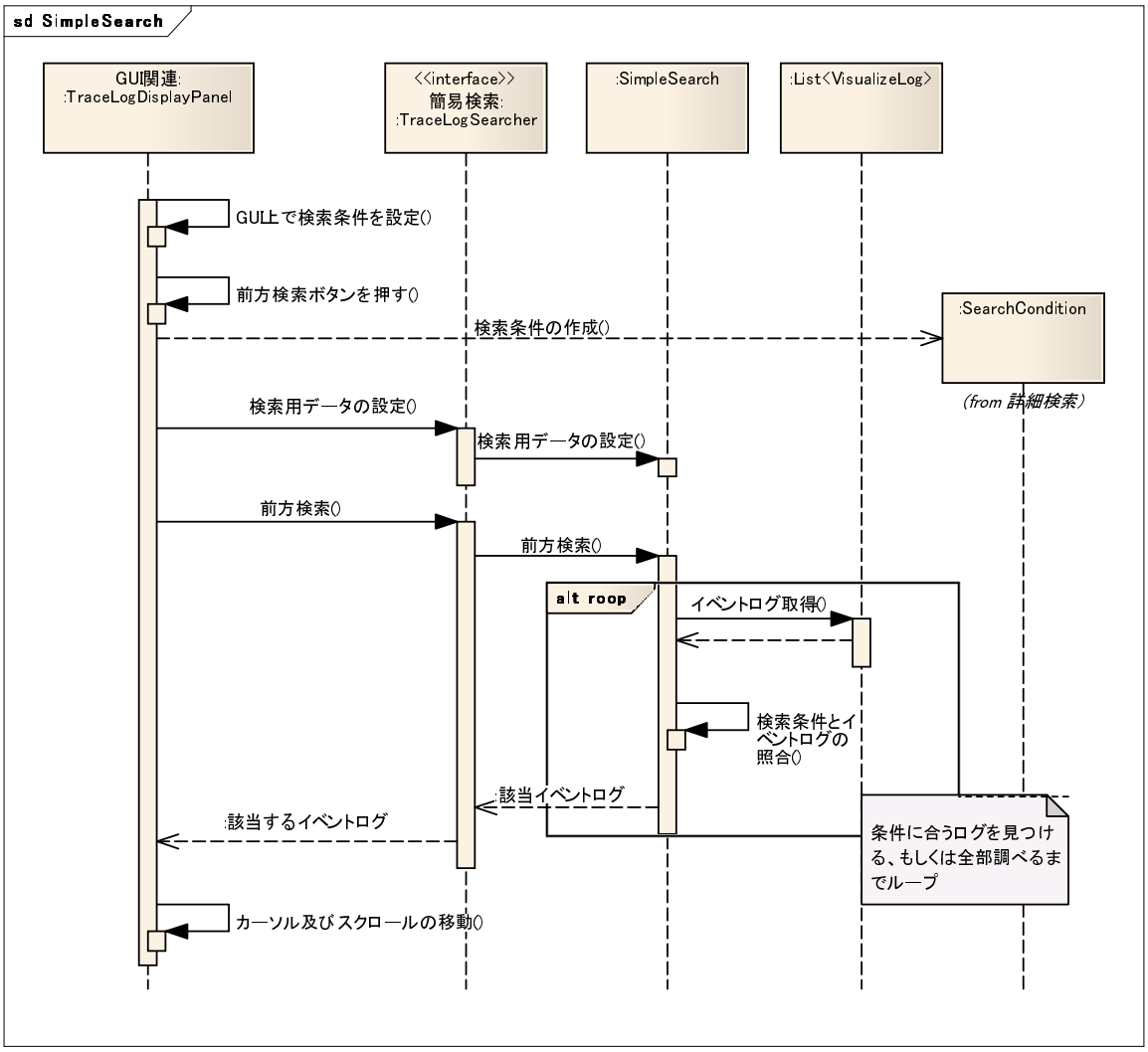

図 5.4　簡易イベント検索のシーケンス

TraceLogDisplayPanel 上で検索条件が入力され、前方検索ボタンが押されると検索条件を保持する SearchCondition クラスが作成される。TraceLogDisplayPanel では SimpleSearch クラスの検索メソッドを利用するため、List¡VisualizeLog¿と SearchCondition が検索用データとして渡される。その後 TraceLogDisplayPanel から SimpleSearch の前方検索メソッドが呼び出される。SimpleSearch では List¡VisualizeLog¿から順番にイベントログを取り出し、SearchCondition とのマッチングを行う。該当するログが見つかった時点でそのログを TraceLogDisplayPanel へ返

す。TraceLogDisplayPanel では返されたログの時刻へカーソルを移動させる、もしくはログが返ってこなかった場合は検索条件に該当するイベントが存在しない旨をポップアップウィンドウで表示する。

5.3.4 実装画面

簡易イベント検索機能の実装画面を図 5.5 に示す。画像上部の 2 行目のツールストライプが簡易イベント検索に関する操作を行う部分である。検索条件指定部で検索条件を入力し、検索実行部で 3 つのボタンのうちどれかを押すことで検索が実行される。後方検索、前方検索の場合はカーソルが該当時刻に移動し、全体検索の場合は該当時刻全てにマーカーが引かれる。

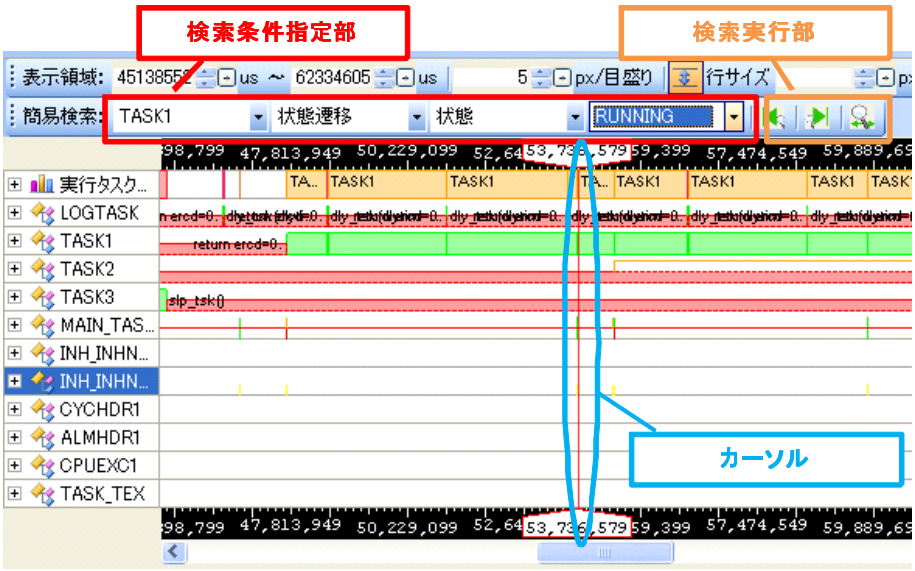

図 5.5 簡易イベント検索機能の実装画面

5.4 詳細イベント検索機能

5.4.1 機能仕様

詳細イベント検索の仕様を以下に記述する。

■検索条件

詳細イベント検索は、複数のイベントや、ある時点でのリソースの属性値など、複数の検索条件を組み合わせて目的のパターンを発見する機能である。検索条件の組み合わせが異なれば、処理の方式、検索条件の入力 GUI も大きく異なるため、組み合わせの種類（以後、検索方法と呼ぶ）ごとに検索処理クラス、GUI を作成することになる。

今回実装を行う検索方法は、イベントの発生タイミングを考慮した検索であり、**タイミング検索**と呼ぶ。例えば「タスク 1 に実行状態への状態遷移イベントが起きた時刻のうち、タスク 1 におい

てその時刻から 5mm 秒後以上後に強制終了イベントが発生している時刻」というような検索条件を指定できる。タイミング検索では、「実行状態への状態遷移イベント」が最終的に検索する時刻を指定する条件であり、「5mm 秒以降に強制終了イベントが発生」が検索対象イベントを絞込む条件となる。以後、前者を**基本条件**、後者を**絞込み条件**と呼ぶ。また、基本条件で検索される時刻を**基準時**と呼ぶ。

基本条件で指定できる条件は、簡易イベント検索で指定できた以下の４条件である。

- リソース名
- ルール名
- イベント名
- イベント詳細

絞込み条件で指定できる条件は、この 4 条件に次の 3 条件を加えたものである。

- タイミング
- 時間
- 条件の否定

タイミングとして指定できる条件は次の 4 つである。時間単位は秒以外にも ミリ秒、マイクロ秒など任意である（リソースファイルで指定された時間単位となる）。

- ○○秒以内（基準時以前）
- ○○秒以内（基準時以降）
- ○○秒以上前
- ○○秒以上後

基準時に対する上記 4 つのタイミングの関係を図図 5.6 に示す。

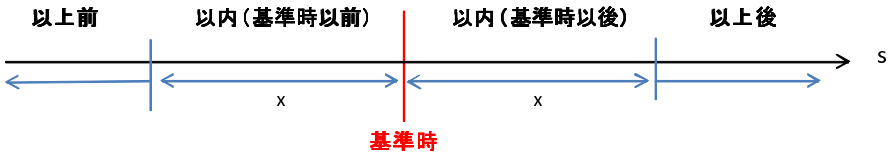

図 5.6　基準時とタイミングの関係

■検索操作のための GUI

詳細イベント検索では検索条件が複雑になるため、別途フォームを用意し、その中で検索条件の作成、検索の実行などの検索にともなう操作を行う。各操作の仕様を次に示す。

- 検索条件の追加：
  詳細イベント検索では複数の検索条件を指定可能であるため、それらを指定する領域を順次

画面上に追加していく必要がある。基本条件とそれに対する絞込み条件（複数可）をまとめて**条件セット**と呼ぶ。基本条件を設定する領域の下に、絞込み条件を設定する領域を順次追加していくようにする。なお、詳細イベント検索では複数の条件セットを指定可能とする。また、1 つの基本条件に対して複数の絞込み条件が存在する場合、基本条件と絞込み条件の関係はすべて AND 、もしくは OR のみとする。つまり、全て AND の場合は、基本条件*（絞込み条件 1 + 絞込み条件 2 ・・・）となり、全て OR の場合は、基本条件 * 絞込み条件 1 * 絞込み条件 2・・・となる。

- 検索条件の指定：
  リソース名やルール名といった条件の指定には間イベント検索と同じくコンボボックスを使用しドロップダウンリストからユーザに条件を指定させる。
- 検索の実行：
  簡易イベント検索と同じ 3 種類の検索ボタンを用意する。また作成した条件セット全てを用いた検索と、条件セットの一部だけを使用した検索両方が可能とする。
- 結果表示：
  簡易イベント検索と同様

5.4.2 ユースケース

定義した仕様をもとに、ユーザが詳細イベント検索を行う際のユースケースを図 5.7 に示す。Detail Search Panel は詳細検索用の画面であり、検索条件の作成を行う領域である Condition Setting Area と、それ以外の操作を行う領域である Operation Area で構成される。

■Search Operation Area におけるユースケース

- **カーソルを移動させる**：
  TraceLogDisplayPanel 上のカーソル（検索開始地点）を移動させる。
- **基本条件を追加する**：
  1 つの基本条件を設定するための領域を条件作成領域に追加する。すでに基本条件の設定領域が存在している場合は、その下に新たに基本条件設定用の領域が追加される。
- **全ての条件セットで検索を行う**：
  全ての条件セットを用いて前方検索、後方検索、全体検索を行う。前方検索、後方検索では、各条件セットで検索された時刻のうち、もっとも検索開始地点に近い時刻へカーソルが移動する。

■Search Condition Setting Area におけるユースケース

- **基本条件を設定する**：
  リソース名、ルール名、イベント名、イベント詳細を設定する。リソース名が指定されると、ルール名の指定ボックスが出現する、というように、順次指定ボックスが追加されていく。

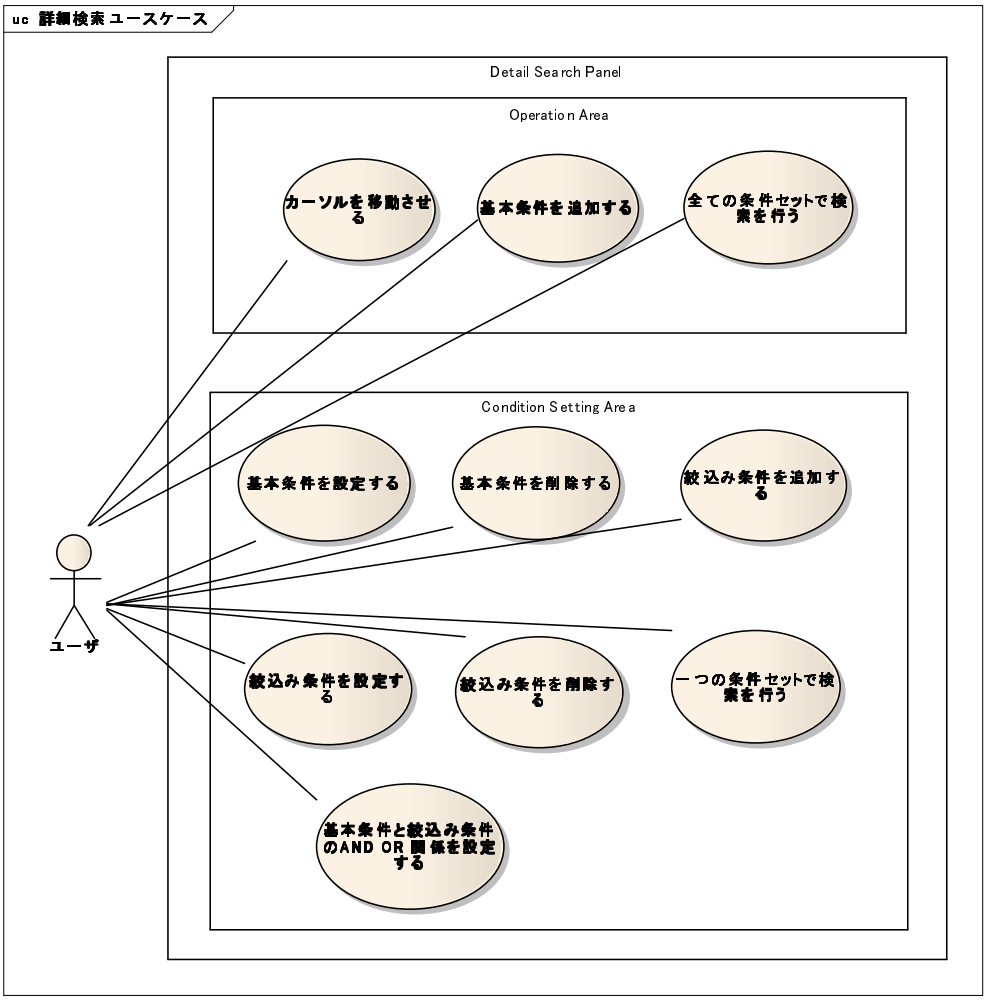

図 5.7　詳細イベント検索のユースケース

- **基本条件を削除する**：
  条件作成領域から基本条件とそれに対する絞込み条件を削除する
- **絞込み条件を追加する**：
  基本条件の設定領域の下に、絞込み条件を設定するための領域を追加する。
- **絞込み条件を設定する**：
  リソース名、ルール名、イベント名、イベント詳細、タイミング、タイミング値、条件の否定を指定する。
- **絞込み条件を削除する**：
  条件作成領域から対象の絞込み条件設定領域を削除する
- **基本条件と絞込み条件の AND、OR 関係を設定する**：
  基本条件に絞込み条件が複数存在している場合、AND のみか OR のみを選択する。

- **1 つの条件セットで検索を行う**：
  条件セット 1 だけを用いて、前方検索、後方検索、全体検索を行う。

5.4.3　設計

■クラス設計

詳細イベント検索を実現するためのクラス群を図 5.8 に示す。赤枠で囲った画面構成クラス群と、青枠で囲ったタイミング検索クラスが図 5.3 からの変更点である。

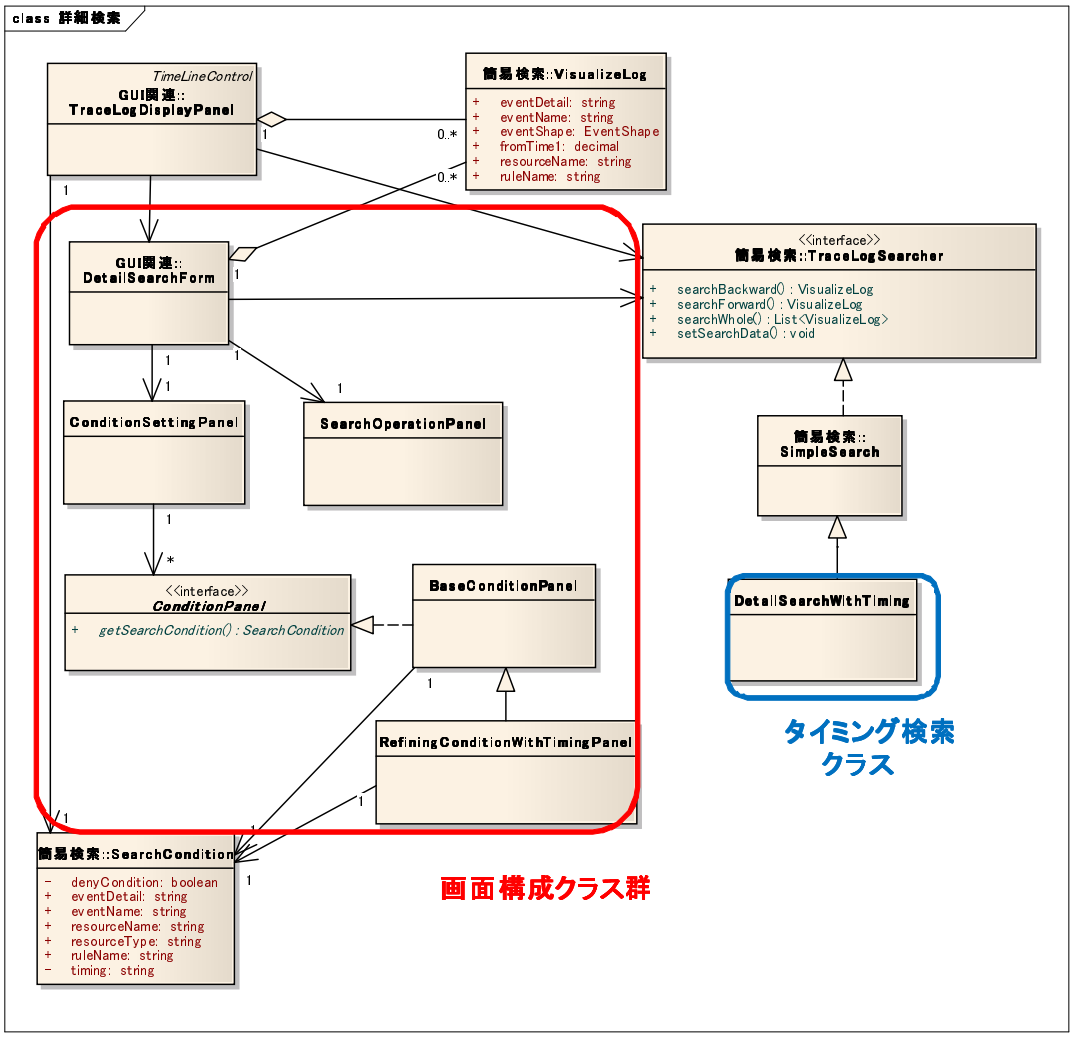

図 5.8　詳細イベント検索を実現するクラス

タイミング検索処理を行うクラスが DetailSearchWithTiming である。SimpleSearch で行われる検索処理に、タイミングによるフィルタをかける処理であるため、SimpleSearch を継承させた。

詳細イベント検索フォームの中核をなすのが DetailSearchPanel である。DetailSearchPanel の

詳細な構成を図 5.9 に示す。図中で名前に”*” がついているクラスは、DetailSearchPanel 上で複数存在できるクラスである。DetailSearchPanel は、「検索条件の設定 GUI の追加を簡単にする」というコンセプトのもとで設計した。そのため、1 つの条件セットを設定するパネルである ConditionSettingPanel を用意し、条件セットが追加されるたびに ConditionSettingArea に追加していく構造にした。さらに、ConditionSettingPanel には条件 1 つにつき 1 つの条件保持パネルを追加していく構造にし、こちらも簡単に条件を追加できるようにしてある。

条件保持パネルにはインタフェース ConditionPanel を実装させ、パネル内に保持されている条件を ConditionSettingPanel が受け取る際に、getSearchCondition メソッドを介するようにした。そのため COnditionSettingPanel は自分に張り付いている条件パネルの種類を考慮する必要がなくなる。これ利点は、この先様々な詳細検索が実装され、条件パネルの種類が増えた際にも、ConditionSettinPanel に変更を加える必要がなくなることである。

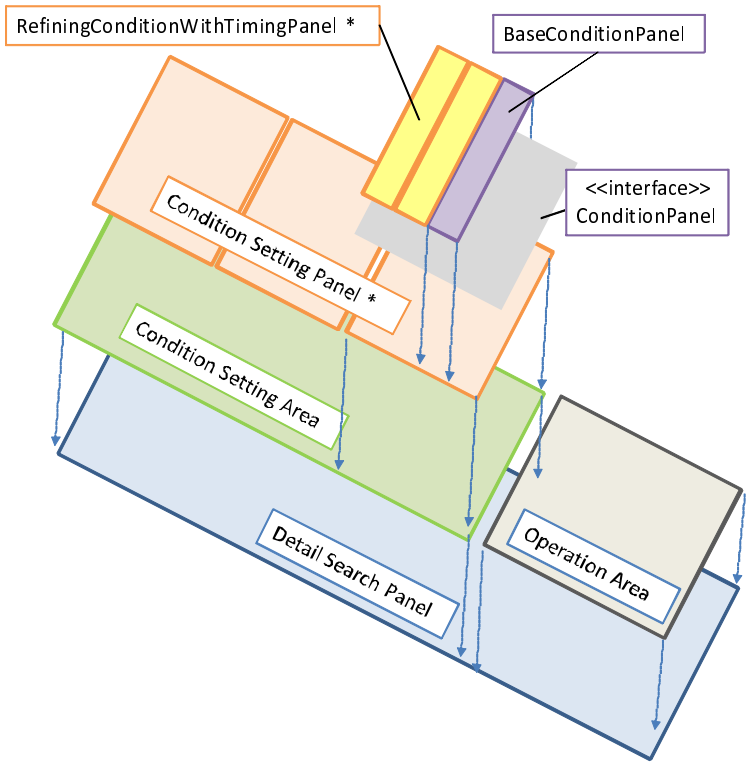

図 5.9　DetailSearchPanel の構成

基本条件は、この先様々な検索方法を実装することになった場合でも共通して使用される。つまり、各検索を行うためのフォームでも基本条件設定用の GUI が必要になる。そのため、基本条件を設定するための GUI がこのポーネント化されていることが望ましい。よって、基本条件設定用に BaseConditionPanel クラスを図 5.10 のような構造をもつクラスとして設計した。

今回実装するタイミング検索では、絞込み条件には基本条件とその他 3 つの条件を設定できる。そのため、絞込み条件設定用の RefiningConditionPanel クラスは BaseConditionPanel

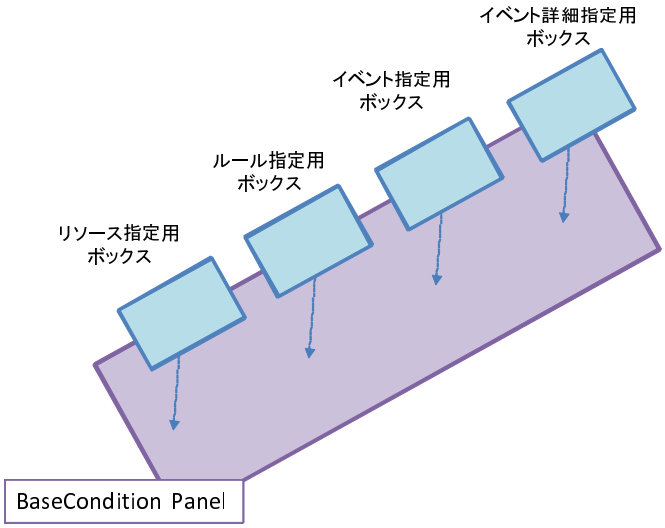

図 5.10　BaseConditionPanel の構成

を継承させる形で設計した。RefiningConditionPanel の構造を図 5.11 に示す。絞込み条件は DetailSearchPanel 上で頻繁に追加されることになるが、条件指定用の GUI がパネルにまとまっているため、絞込み条件が追加されても図 5.9 に示したように、ConditionSettingPanel に順次 RefiningConditionPanel を追加するだけでよく、設計コンセプトが活きているといえる。

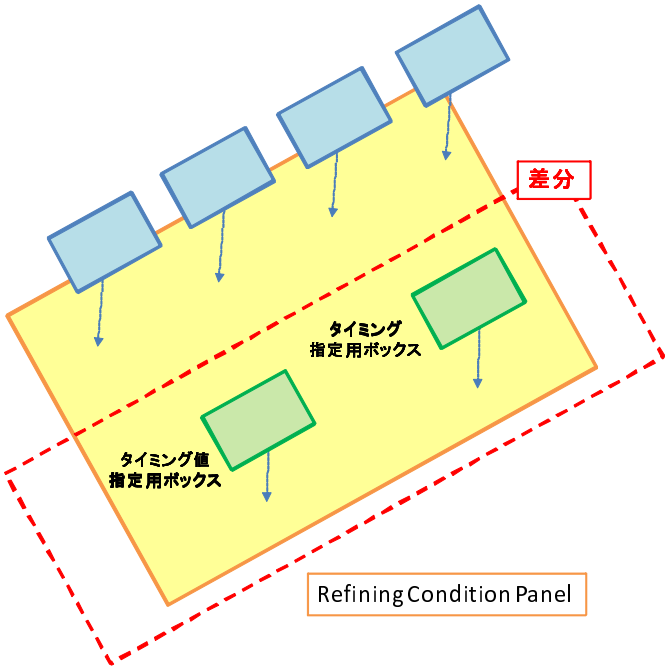

図 5.11　RefiningConditionPanel の構成

■シーケンス

前方検索が実行された際に、各クラスがどのように動作するかを図 5.12 に示す。

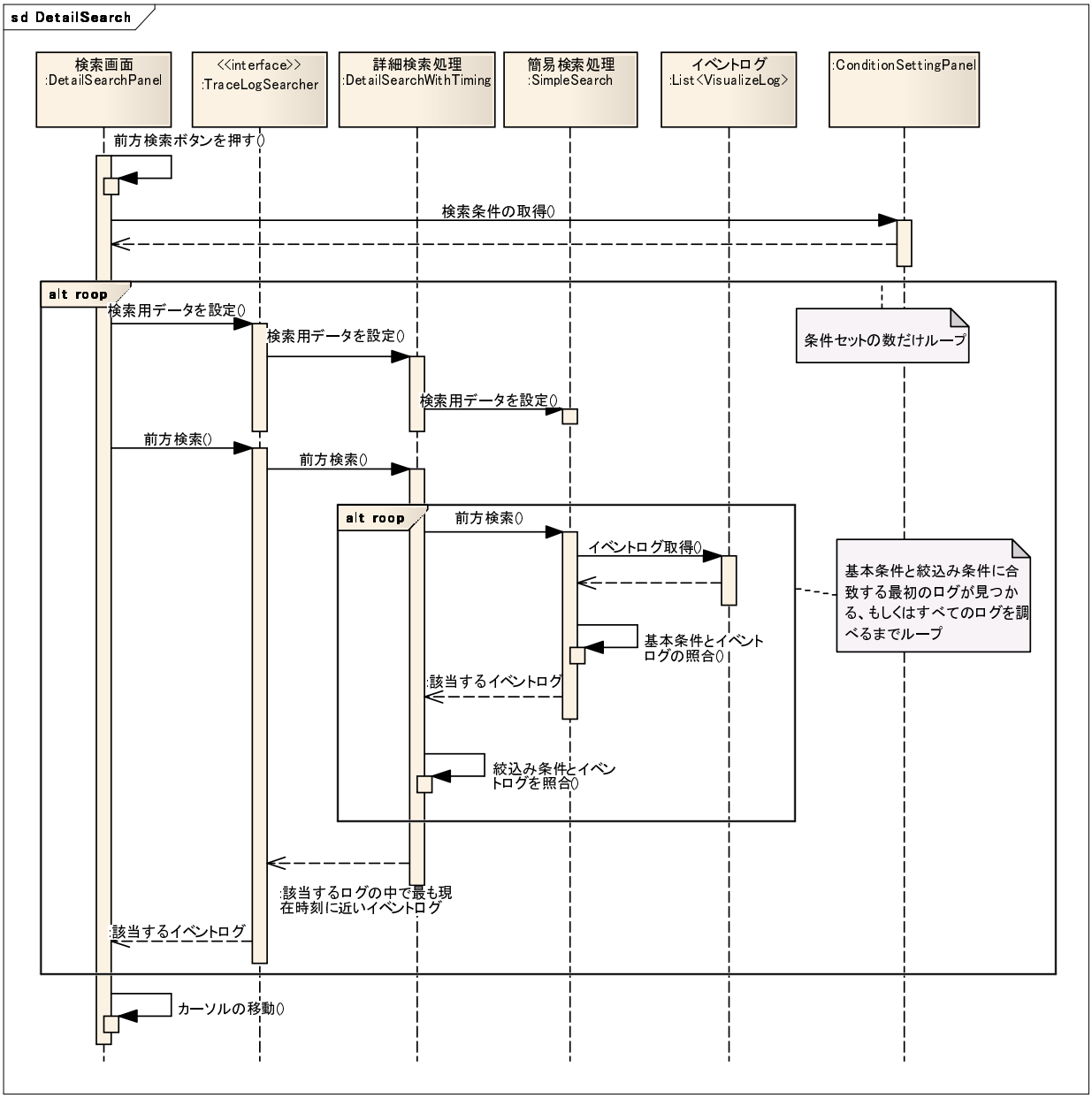

図 5.12　詳細イベント検索のシーケンス

DetailSearchPanel において前方検索の実行ボタンが押されると、まず ConditionSettingPanel から全ての条件セット（SearchCondition の組）を取得する。続いてタイミング検索を担当する DetailSearchWithTiming に 1 つの条件セットとイベントログが渡され、さらに DetailSearch-WithTiming が基本条件の検索を担当する SimpleSearch に基本条件とイベントログを渡す。ここ

までで検索条件の設定が終了する。

続いて DetailSearchPanel が検索実行命令を DetailSearchWithTiming に出し、基本条件に合致するイベントログの検索を開始する。この処理は DetailSearchWithTiming が SimpleSearch へ委譲する形で行われる。SimpleSearch は基本条件に合致するイベントログを見つけると、そのログを DetailSearchWithTiming に返す。見つからなければ DetailSearchPanel へその旨を通知する。なお、シーケンス図では省略しているが、条件の照合は SimpleFiler が担当している。そして DetailSearchWithTiming は、返されたログと絞込み条件の照合を行い、合致する場合には DetailSearchPanel へと渡す。合致しなければ、返されたログの時刻を基準時として再度前方検索に戻る。ここでの条件の照合は SmipleFilter と TimingFilter が行う（SimpleFilter を TimingFilter でデコレートする形で処理が行われる）。

DetailSearchPanel では、検索条件を満たすイベントログが返ってくると、TraceLogDisplay-Panel 上のカーソルをログの時刻に移動させる処理を行う。もしも条件セットが複数ある場合には、次の条件セットを用いて再度前方検索を行い、最終的に複数のログが検索にヒットすると、検索開始時刻に最も近い時刻へとカーソルを移動させる。

5.4.4 実装画面

詳細イベント検索フォームの実装画面を図 5.13 に示す。

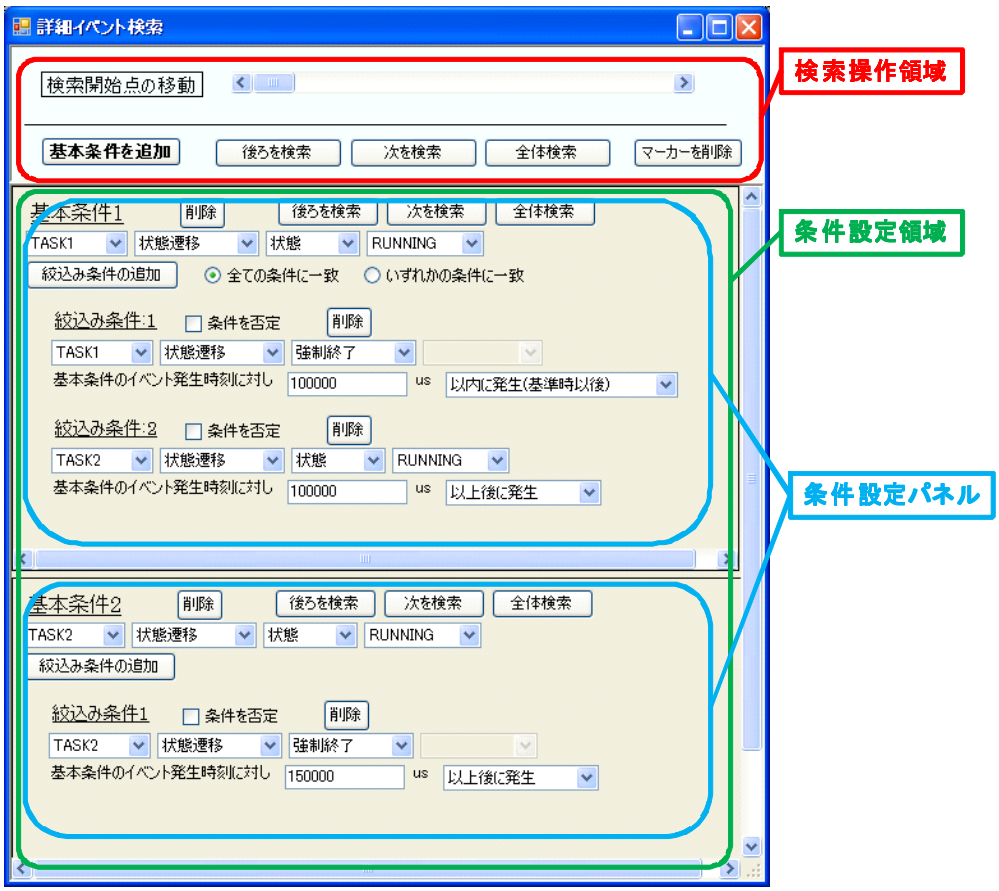

図 5.13　詳細イベント検索フォーム

上部の検索操作領域では、検索開始時刻の設定、基本条件の追加、検索の実行ができる。基本条件の追加ボタンが押されると、条件設定領域に条件設定パネルが追加されていく。各条件設定パネルでは、基本条件の作成と、絞込み条件の追加・編集ができ、さらにそのパネル内の条件だけを使用した検索も可能となっている。ここで実行された検索結果は、簡易イベント検索と同様に TraceLogDisplay パネルのタイムライン上に、カーソル、もしくはマーカーによって表示される。

5.5 考察

■タイミング検索の有用性

■設計についての考察

今回実装したタイミング検索は詳細イベント検索のうちの 1 つであり、将来的にもっとたくさんの検索方法を用意する予定である。例えばタスク A が 10 回目の実行状態になった時刻を探すという回数指定検索があげられる。

詳細イベント検索では、検索方法ごとに条件の指定が複雑になるため、それぞれに専用のフォーム

各フィルタをクラスとして用意し、自由に組み合わせられるようにすれば、新規に検索クラスを実装する場合には過去に実装したフィルタを取込むことができる。さらに、既存の検索クラスに指定できる検索条件を追加したくなった際にも、フィルタを追加するだけでよくなる。この仕組みを実現する方法として、Decorator パターンが利用できる。図 5.14 に Decorator パターンを適用した場合のクラス図を示す。

図 5.14 は、

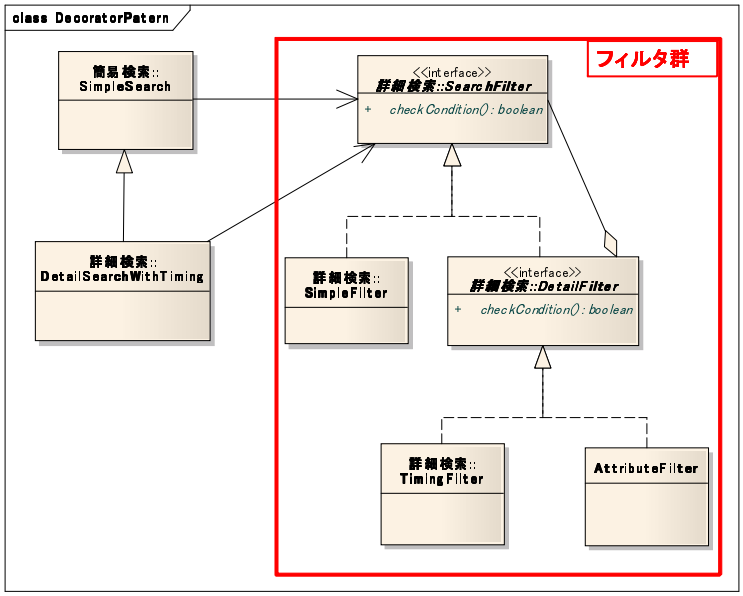

図 5.14　Decorator パターンの導入例

6 おわりに

6.1 まとめ

組込システムにおいてマルチコアプロセッサの利用が進むとともに、シングルコア環境で行われている従来のデバッグ方法では対応できなくなってきた。マルチコアプロセッサは各コアが並列に動作するため、プログラムの挙動が非決定的になるため再現性が保証されず、ブレークポイントを用いたステップ実行による解析といった、実行中のデバッグではバグを確実に発見できない。

そのためマルチコア環境でのデバッグでは、プログラムの実行ログであるトレースログを解析する手法が有効とされている。しかし、トレースログはリソースの挙動を逐一記録したログであり、サイズが大きくなりがちである。これを開発者が手動で解析するには大きなコストが必要になり、非効率的である。そこでトレースログを可視化するツールが様々な組織によって開発されている。しかしながら、既存の可視化ツールは、それぞれが特定の形式のトレースログしか扱えず汎用性に乏しい。また、可視化できる項目も限られている。

こうした背景をもとにして、様々な形式のトレースログを利用者が望む形で可視化することができるという汎用性を持ったトレースログ可視化ツール TLV が開発された [1][4]。TLV では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しており、任意の形式のトレースログを標準形式ログへと変換する仕組みを変換ルールとして形式化した。また、トレースログの可視化表示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。この 2 つの仕組みによって TLV では汎用性、拡張性を実現している。

本 OJL は、OJL1 期生と 2 期生が開発してきた TLV に対して機能追加を行う目的で実施した。まず利用者から要求抽出を行い、その中で要望の強かったイベント検索機能の実装を行った。要求された検索方法は、特定のイベントをその他のイベントの発生タイミングを考慮しながら検索するというものであり、処理が非常に複雑になると予想された。そのため、1 つのイベントを検索する簡易イベント検索と、利用者からの要求にあったタイミング検索を実現する詳細イベント検索に分割し、まず簡易イベント検索を実装し、それを拡張して詳細イベント検索を実装した。イベント検索機能によって、膨大な可視化領域から目的のパターンを即座に発見できるようになり、デバッグコストを低下させることができた。

本 OJL では詳細検索としてタイミング検索のみを実装したが、利用者から要望があるたびに新たな検索方法を実装していく。そのためコードの再利用性、拡張性が重要になる。検索を行うにはその操作を行うための GUI が必要になるが、その開発コストは高いと予想された。そのため、設計の際には GUI の再利用性を考慮し、検索条件を指定する GUI のコンポーネント化し、低コストで新規検索手法の導入が可能になるようにした。

今後の課題としては、利用者からの要求抽出によって新たな検索手法を獲得すること、以前から要望の強かった TLV の高速化があげられる。

6.2 所感

参考文献

[1] 後藤隼弐. Ojl によるトレースログ可視化ツールの開発, 2009.

[2] 水野洋樹. 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, 2010.

[3] 柳澤大祐. トレースログ可視化ツール (tlv) に対するアプリログ機能追加とプロファイリング, 2010.

[4] 後藤隼弐, 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, 2009.