組込み RTOS 向けアプリケーション開発支援ツール
TLV（トレース ログ ヴィジュアライザー）
フェーズ 5 リファクタリング仕様書

2010 年 5 月 11 日

改訂履歴

版番	日付	更新内容	更新者
1.0	10/4/23	新規作成	市原大輔
1.1	10/5/11	リファクタリング後の例変更。説明の具体化・明確化。 クラス図の変更・追加。リファクタリング報告書追加。	市原大輔

目次

1 はじめに

1.1 本書の目的

本書の目的は、文部科学省先導的 IT スペシャリスト育成推進プログラム「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」プロジェクトにおける、OJL 科目ソフトウェア工学実践研究の研究テーマである「組込み RTOS 向けアプリケーション開発支援ツールの開発」に対して、その開発するソフトウェアに対する設計を記述することである。

本書は特に、フェーズ 5 におけるリファクタリング作業に関する記述を行う。

1.2 本書の適用範囲

本書は、組込み MPRTOS 向けアプリケーション開発支援ツールの開発プロジェクト（以下本プロジェクト）のフェーズ 5 におけるリファクタリング作業に関する記述を行う。

1.3 用語の定義/略語の説明

表 1　用語定義

用語・略語	定義・説明
TLV	Trace Log Visualizer
MPRTOS	マルチプロセッサ対応リアルタイムオペレーティングシステム
トレースログファイル	RTOS のトレースログ機能を用いて出力したトレースログや、シミュレータなどが出力するトレースログをファイルにしたもの
標準形式トレースログファイル	本ソフトウェアが扱うことの出来る形式をもつトレースログファイル。各種トレースログファイルは、この共通形式トレースログファイルに変換することにより本ソフトウェアで扱うことが出来るようになる。
変換ルール	トレースログファイルを標準形式トレースログファイルに変換する際に用いられるルール。
可視化ルール	標準形式トレースログファイルを可視化する際に用いられるルール。
TLV ファイル	本ソフトウェアが中間形式として用いるファイル。前述の標準形式トレースログファイルは、この TLV ファイルの一部である。
EBNF	Extended Backus?Naur Form の略。

1.4 概要

本書では、組込み MPRTOS 向けアプリケーション開発支援ツールのソフトウェアの仕様を記述する。本書は特に、フェーズ 5 におけるリファクタリング作業に関する記述を行う。

第 I 部
リファクタリング仕様書

2 概要説明

2.1 リファクタリングを実施する理由

現状の TraceLog クラス (クラス概要は 2.2 参照) では、標準形式トレースログのパースにおいて、Listing1 のような複雑な正規表現を計 8 回適用している。これは、Object などの要素 1 つにつき 1 行の正規表現で取得するためである。こうすることで、Time や Value 等の有無に関わらず[*1]パースできるようにしている。しかし、このような複雑な正規表現は、可読性が低く保守が難しいため、リファクタリングを実施する。

リファクタリング後の TLV は、標準形式トレースログのパースを専用のパーサに任せる。パーサは、TLV 変換ルール・可視化ルールマニュアル (rule-maual.pdf) の「2.1.2 標準形式トレースログの定義」にある EBNF のようにコードを記述することで、構文を直感的に理解できるようになる。コード例を Listing2 に示す。

また、重複したコードおよび生成規則変更時の変更箇所も減らすことができるため、保守性も向上する。例えば、Time を囲む記号"["，"]"を"<"，">"にするとき、従来の正規表現を用いた方法では、8 行すべての正規表現を変更する必要があるが、リファクタリング後は 1 箇所で済むようになる。生成規則を追加する場合でも、従来の方法では 8 行の正規表現の中から変更箇所を探さねばならないが、リファクタリング後は、EBNF のように記述されているため、変更箇所が明確であるので見落としを減らせる。しかしながら、このような構文の変更は稀であるため、一番のメリットは構文を直感的に理解できるようになることである。

Listing 1　リファクタリング前のコード例

```
1  m = Regex.Match(_log, @"^(\[[^\]]+\])?(?<objectType>[^\[\]\(\)\.]+)\([^\)]+\)(\.[^\s]+)?$");
2          if (m.Success)
3                  ObjectType = m.Groups["objectType"].Value;
4          HasObjectType = m.Success;
```

Listing 2　リファクタリング後に表現したいコード例

```
1  public abstract class MiniMLParsers<TInput> : CharParsers<TInput>{
2      public MiniMLParsers() {
3          Whitespace = Rep(Char(' ').OR(Char('\t').OR(Char('\n')).OR(Char('\r'))));
4          WsChr =  chr => Whitespace.AND(Char(chr));
5          Id =     from w in Whitespace
6                   from c in Char(char.IsLetter)
7                   from cs in Rep(Char(char.IsLetterOrDigit))
8                   select cs.Aggregate(c.ToString(),(acc,ch) => acc+ch);
9          Ident =  from s in Id where s != "let" && s != "in" select s;
10         LetId =  from s in Id where s == "let" select s;
11         InId =   from s in Id where s == "in" select s;
12         Term1 = (from x in Ident
13                  select (Term)new VarTerm(x))
14                 .OR(
15                 (from u1 in WsChr('(')
```

[*1] 可視化ルールファイルには、Time がない、Value がない、AttributeName がない等といった不完全な標準形式トレースログが記述されている。

```
16                      from t in Term
17                      from u2 in WsChr(')')
18                      select t));
19          Term =  (from u1 in WsChr('\\')
20                      from x in Ident
21                      from u2 in WsChr('.')
22                      from t in Term
23                      select (Term)new LambdaTerm(x,t))
24                     .OR(
25                     (from letid in LetId
26                      from x in Ident
27                      from u1 in WsChr('=')
28                      from t in Term
29                      from inid in InId
30                      from c in Term
31                      select (Term)new LetTerm(x,t,c)))
32                     .OR(
33                     (from t in Term1
34                      from ts in Rep(Term1)
35                      select (Term)new AppTerm(t,ts)));
36          All =     from t in Term from u in WsChr(';') select t;
37      }
38
39      public Parser<TInput, char[]> Whitespace;
40      public Func<char, Parser<TInput, char>> WsChr;
41      public Parser<TInput, string> Id;
42      public Parser<TInput, string> Ident;
43      public Parser<TInput, string> LetId;
44      public Parser<TInput, string> InId;
45      public Parser<TInput, Term> Term;
46      public Parser<TInput, Term> Term1;
47      public Parser<TInput, Term> All;
48  }
```

2.2 リファクタリングを実施する対象

リファクタリング対象は、TraceLog クラス、特にそのコンストラクタである。この TraceLog クラスは、標準形式トレースログを構成するトークンを保持し、標準形式トレースログを用いた処理 (可視化表示など) を容易にする。

現在、標準形式トレースログのパースに正規表現を用いている箇所をパーサを用いるように、TraceLog コンストラクタを変更する。

3　変更内容

3.1　クラス構成

正規表現によるパース (Regex クラスおよび Match クラス) を廃止し、標準形式トレースログ用のパーサを作成して、TraceLog クラスはそのパーサへパース処理を委譲する。これにより、現在はパースの実装が TraceLog クラスに直書きされているためパースの実装方法を変更するのが困難であるのが、TraceLog クラスとパースの実装を分離することでパーサの交換が容易になり、パーサの実装方法変更が容易になる。

今回の変更に伴う影響範囲は、TraceLog クラス以外に存在しない。リファクタリング前後のクラス図を図 1,2 に示す。

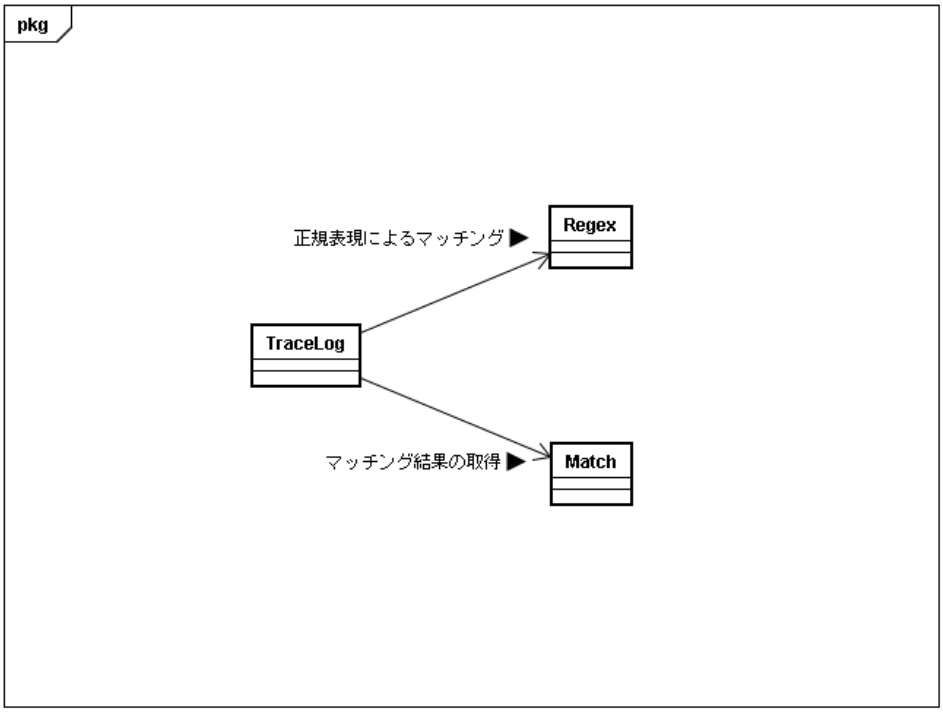

図 1　リファクタリング前のクラス構成

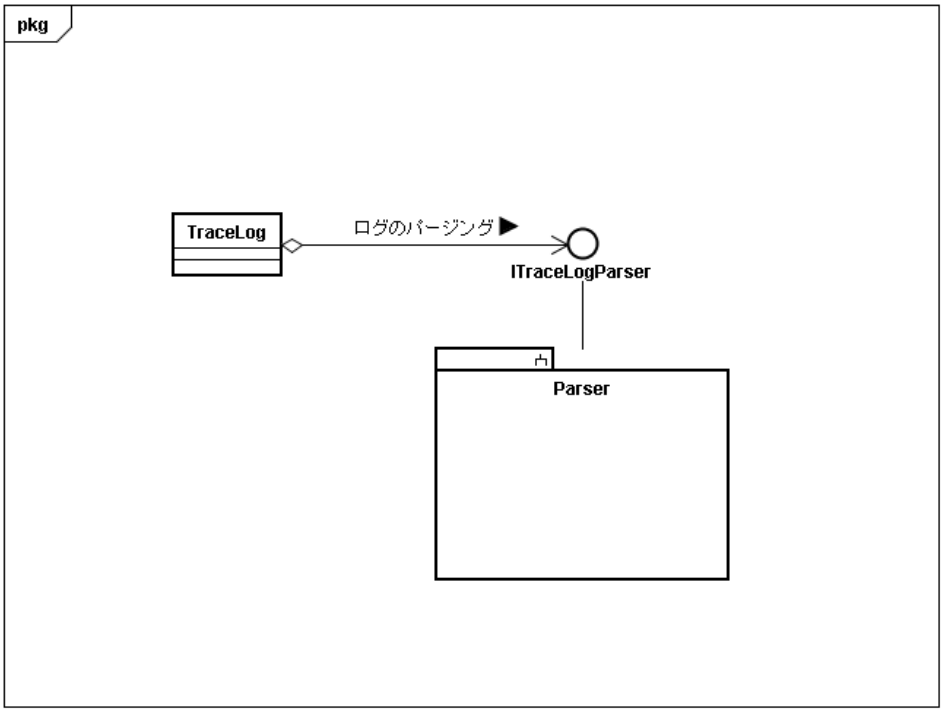

図 2　リファクタリング後のクラス構成

3.2　パーサの構成

パーサの実装は、コードで EBNF を表現し、直感的に理解できるようにする。

例として、Haskell のパーサコンビネータライブラリ Parsec を参考にした次の URL にあるものが挙げられる (コードの一部は Listing2 参照)。あくまで例であり、このコードを使用するか、自作など他の実装方法を用いるかは、デバッグの容易さ、実装のしやすさ、可読性、実行速度などの要素により決定する。

LukeH's WebLog : Monadic Parser Combinators using C# 3.0
http://blogs.msdn.com/lukeh/archive/2007/08/19/monadic-parser-combinators-using-c-3-0.aspx

第 II 部
リファクタリング報告書

4　概要

本書は、Phase5 で行ったリファクタリングにより変更・追加された箇所、および、リファクタリング後のシーケンスを説明し、今回採用しなかった手法との比較や今回の実装の採用理由を記述する。

5　コード設計

ここでは、リファクタリングの目標であった「EBNF を表現し、直感的に理解できる」コードの紹介と、小節にて実装に用いた手法・概念および主要メソッドの説明をする。

まず、「EBNF を表現」したコードの例を Listing3 に示す。

Listing 3　「コードで EBNF を表現」したコード例 (最終更新日現在)

```
var line = Char('[').Time().Char(']').Event();
...
var object_ =
        ObjectTypeName().Char('(').AttributeCondition().Char(')')
        .OR().
        ObjectName();
...
var typeName = Many1(() => AnyCharOtherThan('(', ')', '.'));
```

これは、次の EBNF をパースすることを示している (rule-manual.pdf の 2. 1.2 参照)。

Listing 4　Listing3 が表す実際の EBNF

```
TraceLogLine = "[", Time, "]", Event;

Resource = ResourceTypeName, "(", AttributeCondition, ")"
         | ResourceName;

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

このように、Listing3 は EBNF を表現しており、正規表現によるパージングより可読性が向上している。Listing3 は完全なものではなく、実際には Listing5 のような解析メソッド内に記述され、それを組み合わせることで Listing3 を実現している。

Listing 5　解析メソッド例

```
public ITraceLogParser ObjectTypeName()
{
        Begin();

        var typeName = Many1(() => AnyCharOtherThan('(', ')', '.'));

        typeName.ObjectTypeValue = Result();
        return (ITraceLogParser)typeName.End();
}
```

5.1 利用した概念・手法

5.1.1 再帰下降パーサ

今回、実装したパーサは、スタックを利用した再帰下降パーサであり、バックトラッキングを用いている。これは、EBNF のようにトップダウンに記述されているものを表現するのに適しているが、バックトラッキングによる処理効率低下が懸念される。

5.1.2 パーサ・コンビネータ

パーサ・コンビネータとは、パーサとパーサを組み合わせることのできるコンビネータを指す。具体的には、Many メソッド、Many1 メソッド、OR メソッドが挙げられる。これにより、EBNF の表現や LL(k) 文法のパージングを可能にしている。しかしながら、左再帰性の除去が必要である。

5.1.3 NullObject パターン

5.2 主要メソッドの説明

5.2.1 Begin メソッド

5.2.2 End メソッド

5.2.3 Result メソッド

5.2.4 OR メソッド

5.2.5 Many メソッド

5.2.6 Many1 メソッド

6 クラス設計

リファクタリング後のクラス図を図 3 に示す。

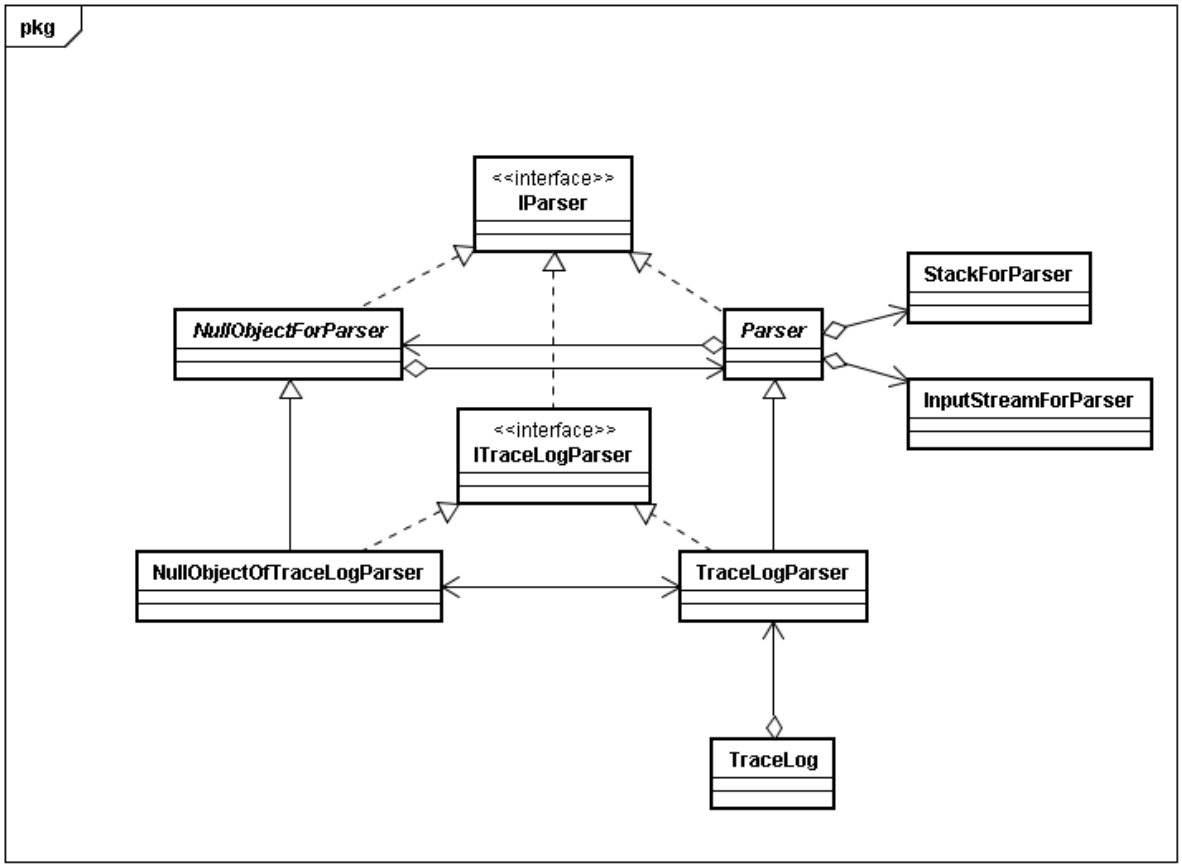

図３　リファクタリング後のクラス構成