社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－SE－151（13）
2006／3／24

Web 抽象プログラムを用いたリファクタリング

伊藤　崇 †　　金子 伸幸 ‡　　山本 晋一郎 §　　阿草 清滋 ††

† ‡ †† 名古屋大学大学院 情報科学研究科 情報システム学専攻
§ 愛知県立大学 情報科学部 情報システム学科

†t-ito@agusa.i.is.nagoya-u.ac.jp ‡nkaneko@agusa.i.is.nagoya-u.ac.jp
§yamamoto@ist.aichi-pu.ac.jp ††agusa@is.nagoya-u.ac.jp

概要

本稿では，Web アプリケーションを構成するコンポーネントの動作とコンポーネント間の関係を表現した Web 抽象プログラムとそれを用いたリファクタリング手法を提案する．Web 抽象プログラムは，コンポーネントの動作を描画命令と処理命令によって表現する．また，遷移や生成といったコンポーネント間の関係を表現する．Web 抽象プログラムの書き換えによって多種の言語で記述されるコンポーネントの再構成を行い，Web アプリケーションの内部構造の改善を図る．ソースプログラムから Web 抽象プログラムを生成するツールを開発し，サンプル Web アプリケーションに対して本手法を適用し，手法の有用性を示した．

キーワード:　Web アプリケーション　リファクタリング　抽象プログラム

A Refactoring Method Using Web Abstract Program

Takashi Ito†, Nobuyuki Kaneko‡, Shinichiro Yamamoto§ and Kiyoshi Agusa††

† ‡ ††Graduate School of Information Science, Nagoya University
§Faculty of Information Science and Technology, Aichi Prefectural University

†t-ito@agusa.i.is.nagoya-u.ac.jp ‡nkaneko@agusa.i.is.nagoya-u.ac.jp
§yamamoto@ist.aichi-pu.ac.jp ††agusa@is.nagoya-u.ac.jp

abstract

In this paper, we propose a refactoring method using an abstract program. Web abstract program represents behavior of components by using two abstract instructions called RENDERING and PROCESS. Web abstract program also represents relations between components such as transition and generation. We recompose various components by rewriting web abstract program. We implement a tool, which generates web abstract program from source program. We apply our method to an application and show useful of our method.

Keywords:　Web Application　Refactoring　Abstract Program

1　はじめに

近年 Web アプリケーションは，オンラインショピングシステムやオンライン座席予約システム，基幹業務システムに至るまで幅広い分野に普及している．Web アプリケーションは B2C(Business to Consumer) の要であり，ビジネスニーズのめまぐるしい変化を背景として，要求の追加や変更が頻繁に発生する [2]．要求の変更に伴なう度重なるソースプログラムの改変は，ソースプログラムの可読性や拡張性の低下を引き起こし，Web アプリケーションが保守の困難な状態となる．Web アプリケーションにおいて，設計が崩れていることに起因する保守性の低下は，リンク切れやページコンテンツの不整合など，いくつかの不具合を引き起こし，信頼性の確保を難しくする [2].

ソフトウェアに対して，外部から見た動作を保ったまま可読性や拡張性を改善する手法にリファクタリングがある．近年では，いくつかのリファクタリング操作の自動実行を行うツールが開発されており，広く利用されている．しかし，HTML や Java, JavaScript など多種の言語で記述された多様なコンポーネントの組み合わせによって構成される Web アプリケーションに対して，これらのリファクタリング手法をそのまま適用することはできない．

本稿では，Web アプリケーションの動作を Web 抽象プログラムとして表現する．抽象プログラムの書き換えによる Web アプリケーションのリファクタリング手法を提案する．

2　Web アプリケーションにおけるリファクタリング

広義にリファクタリングはソフトウェアの内部構造を改善させる作業である．一般に，比較的粒度の細かい変換規則を繰り返し適用しソースプログラムを書き換えることによって，内部構造であるクラス構成が改善される [3].

ソフトウェアとしての Web アプリケーションの内部構造は多種多様なコンポーネントの構成である．本研究では，Web アプリケーションにおけるリファクタリングを以下のように定義する．

Web アプリケーションリファクタリング
コンポーネントの相互作用で決定される外部から見た動作を保ちつつ，コンポーネントの再構成を行って Web アプリケーションのアーキテクチャを改善すること．

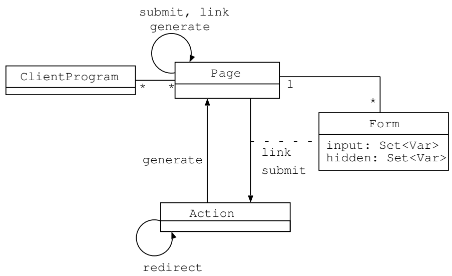

図 1: Web 抽象プログラムの構成要素

自動化を視野に入れた Web アプリケーションリファクタリングの実現には，以下の 3 つの要件を満たすモデルが必要である．

- ソースプログラムから変換可能である
- 内部構造を表現する
- 変換規則を与えられる

フォワードエンジニアリング的なアプローチにおける Web アプリケーションのオブジェクト指向モデルがいくつか提案されているが [4, 9]，既存のソースプログラムからこれらのモデルに変換することは難しい．このことは，コンポーネントの多様性に起因する．Web コンテンツを記述するレンダリング言語などはオブジェクト指向から乖離しているため，これらも含めてオブジェクト指向モデルに変換するには，構文的な抽象化ではなく意味的な抽象化が必要である．

3　Web 抽象プログラム

2 章で述べた 3 つの要件を満たすモデルとして Web 抽象プログラムを提案する．Web 抽象プログラムの構成要素を図 1 に，構文を付録 A に示す．

Web 抽象プログラムは，`Page` と `Action`，`Form`，`ClientProgram` のコンポーネント宣言によって Web アプリケーションを表現する．コンポーネントの動作は，`RENDERING` と `PROCESS` という式によって表現する．`RENDERING` は描画を表わす抽象命令であり，`PROCESS` は描画以外の処理を表す抽象命令である．コンポーネント間の関連は `page_generation` 構文と `action_invocation` 構文に従う式によって表現する．制御やデータの流れは，条件式を表わす抽象命令 `CONDITION` やフォーム入力を表わす `form_input` 構文に従う式を解析することで得られる．Web 抽象プログラムの記述例は図 7 を参照されたい．

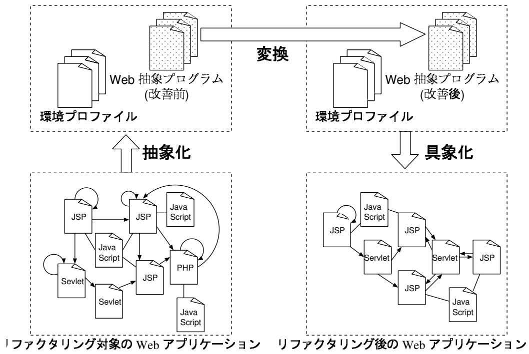

図 2: Web 抽象プログラムを用いたリファクタリング

4 Web 抽象プログラムを用いたリファクタリング

4.1 概要

Web 抽象プログラムを用いたリファクタリングの概要を図 2 に示す．リファクタリングプロセスは抽象化，変換，具象化に分けられる．

抽象化　抽象化は，リファクタリング対象 Web アプリケーションのソースプログラムを Web 抽象プログラムに変換する．また，具象化のために Web 抽象プログラムとソースプログラムとの対応関係を環境プロファイルとして保持する．

変換　変換は，Web 抽象プログラムに対してカタログ化された変換規則を繰り返し適用することで Web 抽象プログラムを書き換える．

具象化　具象化は，書き換え後の Web 抽象プログラムと環境プロファイルから Web アプリケーションのソースプログラムを復元する．

4.2 プレゼンテーションの分離

動機　ページデザインを決めるプレゼンテーションロジックとシステム内部で処理される業務ロジックが混在している場合，ページデザインの変更が難しくなる．そこで，プレゼンテーションロジックと業務ロジックを分離する．この変換を**プレゼンテーションの分離**と呼ぶ．

Bad Smell　`Page` コンポーネント内に以下のいずれかを満たす抽象命令 `PROCESS` が存在する．

- 制御構文 (`if_statement` など) 内に存在しない．
- 制御構文内に存在して，同じ制御構文内に `RENDERING` 命令が存在しない．
- 制御構文内に存在して，同じ制御構文内に `RENDERING` 命令が存在するが，`RENDERING` 命令によって参照される変数を変更していない．

変換手順　変換手順を以下に説明する．

1. `Page` においてどの `RENDERING` 命令もデータ依存していない `PROCESS` 命令を 1 つの `Action` として抽出する．
2. 抽出元の `Page` のインスタンスを生成しているページ生成文が記述されている箇所を 1. で抽出した `Action` の呼び出し式に変更する．この際，`Page` に渡されるパラメータを `Action` 呼び出し式のパラメータとして記述する．
3. `Action` に 2. で作成した `Page` の生成文を記述する．2. で作成した `Page` 中の `RENDERING` 命令が参照している変数のうち，1. で作成した `Action` 中の `PROCESS` 命令によって値を更新している変数をパラメータとして記述する．

適用例　図 3 に適用例を示す．変換前の `Page` search において図中の破線で示した `PROCESS` 命令以外の `PROCESS` 命令はどの `RENDERING` 命令もデータ依存関係にないため，これを searchAction として抽出する．次に `Page` index で search を生成しているページ生成文を searchAction の呼び出し式に変更する．さらに searchAction の最後に search を生成するページ生成文を追加する．これらの変換により変換後の `Page` search から `PROCESS` 命令が取り除かれている．図中の破線で示した `PROCESS` 命令のように抽出されない `PROCESS` 命令は抽象プログラム内においてプレゼンテーションロジックであることを意味する．

4.3 コントローラの分離

動機　ページやサービスの制御を行うプログラムをコントローラとして抽出することで Web アプリケーションの保守性が向上する．そこで，`Action` をコントローラとそれ以外の `Action` に分離する．この変換を**コントローラの分離**と呼ぶ．

Bad Smell　`Page` から呼び出される `Action` に多くの `PROCESS` 命令が含まれている．

```
Page index {
    ..................................

    submit search() {
        input(entry);
        new search(entry);
    }
}

Page search {

    search(Form entry) {
        RENDERING_search_1();
        {$where, $conn, $dv, $sv, $libList} =
                          PROCESS_search_1(entry);
        .................
        PROCESS_search_2();
        if (CONDITION_search_2($conn)) {
            RENDERING_search_4($db, $sv);
        } else if (CONDITION_search_3($libNum)) {
            RENDERING_search_5();
        } else {
            RENDERING_search_6($libNum);
            while(CONDITION_search_4($libList)) {
                {$libList, $iAuthor, $iTitle, $iPub} =
                             PROCESS_search_5($libList);
                RENDERING_search_7($iAuthor, $iTitle, $iPub);
            }
            RENDERING_search_8();
        }
        PROCESS_search_3();
        RENDERING_search_9();
    }
    .................
}
```

```
Page index {
    ...........................
    submit search() {
        input(entry);
        searchAction(entry);
    }
}

Action searchAction(Form entry) {
    {$where, $conn, $dv, $sv, $libList} = PROCESS_search_1(entry);
    PROCESS_search_2();
    PROCESS_search_3();
    new search($where, $conn, $dv, $sv);
}

Page search {
    search($where, $conn, $dv, $sv) {
        RENDERING_search_1();
        .................
        if (CONDITION_search_2($conn)) {
            RENDERING_search_4($db, $sv);
        } else if (CONDITION_search_3($libNum)) {
            RENDERING_search_5();
        } else {
            RENDERING_search_6($libNum);
            while(CONDITION_search_4($libList)) {
                {$libList, $iAuthor, $iTitle, $iPub} = PROCESS_search_5($libList);
                RENDERING_search_7($iAuthor, $iTitle, $iPub);
            }
            RENDERING_search_8();
        }
        RENDERING_search_9();
    }
    ......................
}
```

図 3: プレゼンテーションの分離 (左：変換前，右：変換後)

変換手順　変換手順を以下に説明する．

1. 同一制御構造内の連続した `PROCESS` 命令を 1 つの `Action` として抽出する．

2. 抽出元の `Action` に 1. で抽出した `Action` 呼び出し式を記述する．この際，1. で抽出した `Action` が参照する変数をパラメータとして渡す．

適用例　図 4 に適用例を示す．変換前の `Action` updateAction の `if-else` 文による制御構造で分けられた 2 つの `PROCESS` 命令列をそれぞれ `Action` getQuestionAction と getResultQuestion として抽出する．これらの変換により，変換後の `Action` updateAction が getQuestionAction と getResultQuestion を呼び出すコントローラとなる．

4.4　ページ生成プログラムの分割

動機　度重なる改変によってページ生成プログラムが多種のページを生成してしまうことがある．このようなページ生成プログラムを分割することで可読性や拡張性の向上が見込める．この変換を**ページ生成プログラムの分割**と呼ぶ．

Bad Smell　連続する多くの抽象命令を同一制御構文内に持つ分岐が存在する．

変換手順　変換手順を以下に説明する．

1. `Page` コンポーネント内の制御構造に着目して抽象命令を抽出して，新しく `Page` コンポーネントを作成する．

2. 抽出元の `Page` コンポーネントの生成文が記述されている箇所を，1. で作成した `Page` コンポーネントの生成文と，抽出元の `Page` コンポーネントの生成文を条件分岐させる．条件式は，変換前の `Page` コンポーネントから抽出する．

適用例　図 5 に適用例を示す．適用前の `Page search` の `if-else` 文による制御構造で分けられた 2 つの命令列をそれぞれ `Page search` と `detail` として抽出する．さらに，`Action searchAction` に変換前の `Page search` にあった `if-else` 文にある条件式 `CONDITION_search0` を利用したページ遷移制御文を追加する．これらの変換により，`Page search` を `search` と `detail` に分割できる．その結果，機能の異なるページ生成プログラムを分離し，`Action searchAction` により遷移を制御する構成となる．

5　Web 抽象プログラム生成ツール

`JSP` ファイルから `Web` 抽象プログラムと環境プロファイルを出力するツールを作成した．`JSP` の構文解析には `Sapid[1]` の `JSP-Model` を利用した．本章ではツールの主な機能について述べ，各

```
Page index {
  .........
  submit update() {
    input(entry);
    updateAction(entry);
  }
}

Action updateAction(entry) {
  PROCESS_updateAction0($entry.answer);
  {$question} = PROCESS_updateAction1();
  if (CONDITION_updateAction1($question)) {
    {$result} = PROCESS_updateAction2($question);
    {$result_question} = PROCESS_updateAction3($result);
    new simple($result_question);
  } else {
    new detail($question);
  }
}
```

```
Page index {
  .........
  submit update() {
    input(entry);
    updateAction(entry);
  }
}

Action getQuestionAction(entry) {
  PROCESS_updateAction0($entry.answer);
  {$question} = PROCESS_updateAction1();
}

Action updateAction(entry) {
  getQuestionAction(entry);
  if (CONDITION_updateAction1($question)) {
    {$result} = getResultQuestion($question);
    new simple($result_question);
  } else {
    new detail($question);
  }
}

Action getResultQuestion($question) {
  ........
```

図 4: コントローラの分離 (左：変換前，右：変換後)

```
Action searchAction(search) {
    {$book} =
        PROCESS_search0(search);
    new search($book);
}

Page search {
    search($book) {
        if (CONDITION_search0($book)) {
            RENDRING_search0();
              ........
        } else {
            RENDERING_search10();
             .........
        }
    }
}
```

```
Action searchAction(search) {
    {$book} =
        PROCESS_search0(search);
    if (CONDITION_search0($book)) {
        new search($book);
    } else {
        new detail($book);
    }
}

Page search {
    search($book) {
        RENDRING_search0();
              ........
    }
}

Page detail {
    detail($book) {
        RENDERING_search10();
         ...........
    }
}
```

図 5: ページ生成プログラムの分割 (左：変換前，右：変換後)

機能の JSP 以外の言語への対応について考察を加える．

5.1 コンポーネント抽出

1 つの JSP ファイルは 1 つの Page コンポーネントであるとみなす．ただし JSP ファイルに一切の HTML 記述が存在しない場合は Action コンポーネントとみなす．JSP に form タグが存在した場合，フォームの内容を Form コンポーネントとして抽出する．script タグが存在した場合，スクリプトの内容を ClientProgram コンポーネントとして抽出する．また，JSP の構文解析によって HTML タグから表 1 に示すコンポーネントの関係を抽出する．

本節で述べたコンポーネントの抽出法は JSP の構文に依存しないため，他の言語に対しても共通の抽出法が適用可能である．ただし，ファイルに HTML 記述が存在するか否かの判断は対象が JSP や PHP などの埋め込み型のスクリプトレット言語か Java Servlet などのプログラム言語かによって異なる．

5.2 抽象命令列の生成

ツールの入力となる JSP ファイルに対して以下の 2 つの前処理が行われていることを前提に図 6 のアルゴリズムによって各コンポーネントの抽象命令列を生成する．

1. JSP のスクリプトレット内における HTML の描画は全て ⟨%= %⟩ を利用して行われるように書き換える．
2. 制御構文に記述される条件式を変数で置き換える．こうすることで，全ての条件式において評価時に処理を伴わないようになる．

生成アルゴリズムを図 6 に示し，図 7 に生成例を示す．アルゴリズムにおいて‘‘描画文が参照している変数’’は，スクリプトレット ⟨%= %⟩ で囲まれる部分を解析し抽出する．‘‘逐次文が更新する可能性のある変数’’は代入文の左辺のほかに，メソッドの引数として参照を渡している変数も含む．図 6 のアルゴリズムに対して言語毎に解析手法を与えれば PHP や Java Servlet といった他の言語からも Web 抽象命令列を生成できる．

6 適用例

図 8 のコンポーネント構成を持つアンケートシステムに対して Web 抽象プログラムを用いたリファクタリングを適用する．このアプリケーションは簡単なアンケートシステムで，データベースで管理されるアンケートに回答し，集計結果を表示する．

前章で述べたツールを利用して Web 抽象プログラムを生成し，提案手法によるリファクタリング

表 1: JSP 内の HTML タグとコンポーネント間の関係

HTML タグ種別	コンポーネント間の関係
a	`link`
form	`submit`，`member_declaration(Form)`, `page_generation`, `action_invocation`
script	`member_declaration(ClientProgram)`

```
begin
    for s ∈ {JSP 中の制御構文 } do
        begin
            (s の条件式を CONDITION 命令に変換して抽出);
            (s と抽象命令の対応を環境プロファイルに出力);
        end;
    for s ∈ {JSP 中のステートメント式 } do
        case s of
            (描画文) : begin
                           refs := (s が参照している全ての変数);
                           (refs を引数とする RENDRING 命令に変換);
                           (s と抽象命令の対応を環境プロファイルに出力);
                       end
            (逐次文) : begin
                           refs := (s が参照している全ての変数);
                           updatevars := (s が更新する可能性のある全ての変数);
                           if updatevars = ø then
                               begin;
                                   (refs を引数とする PROCESS 命令に変換);
                                   (s と抽象命令の対応を環境プロファイルに出力);
                               end
                           else
                               begin
                                   (updatevar を左辺、refs を引数とする PROCESS 命令を右辺とする代入文に変換);
                                   (s と抽象命令の対応を環境プロファイルに出力);
                               end
                       end
end;
```

図 6: プログラムから抽象命令列の生成アルゴリズム

を行なった．enquete.jsp から生成された Page に対してプレゼンテーションの分離を適用することで EnqueteAction が抽出される．同様に result.jsp から AnswerAction が生成される．また，抽出された 2 つの Action に対してコントローラの分離を適用することで，他の Action が抽出された．以上よりコンポーネント構成は図 9 のように変化した．

リファクタリング前のアンケートシステムは，プレゼンテーション層にてデータの更新やページ遷移が行われる保守性の低いアプリケーションであった．Web 抽象プログラムを用いたリファクタリングにより MVC アーキテクチャに基づくコンポーネント構成となった．このことは，保守性の向上に繋がる．

7　関連研究

Ricca らはサーバサイドで動的にコンテンツを生成する Web アプリケーションの再構築に関する研究を行っているが [8, 7]，Web アプリケーションの構造の視覚化やテスト技術を目的としたモデルを提案しておりリファクタリング手法に関しては述べられていない．

既存 Web アプリケーションを MVC アーキテクチャに適用させる手法が提案されている [5, 6]．本手法では，ページ生成プログラムの分離など MVC アーキテクチャに適用させる以外のリファクタリングも可能である．

8　おわりに

本稿では，Web アプリケーションの動作を表現する Web 抽象プログラムを提案した．Web 抽象プログラムを用いることで具体的な処理内容に依存せず多種の言語で記述されるコンポーネントの再構成を行うことができる．また，Web 抽象プログラム生成ツールを開発し，アンケートアプリケーションに対して本稿で提案した変換規則を適用した．これにより保守性の高いコンポーネント構成

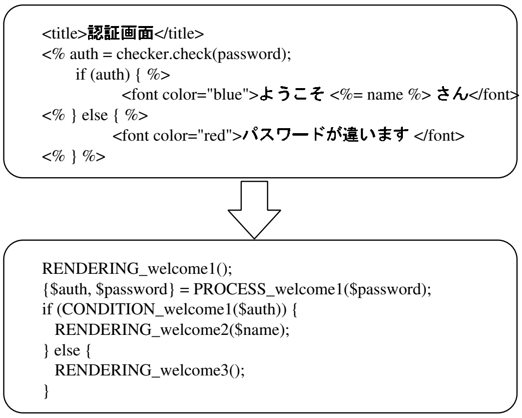

図 7: JSP プログラムから抽象命令列の生成例

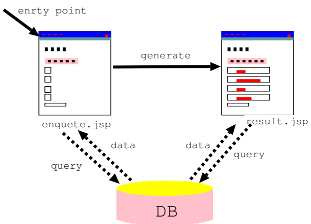

図 8: リファクタリング前のコンポーネント構成

を改善できることを確認した．今後の課題を以下に示す．

- Web アプリケーションリファクタリング支援系の作成
- Web 抽象プログラムの変換規則の充実

謝辞

本研究の一部は文部科学省科学技術研究費基盤研究（B）課題番号 17300006 および文部科学省リーディングプロジェクト基盤ソフトウェアの統合開発 e-Society 高信頼 WebWare の生成技術，文部科学省科学技術研究費若手研究（B）課題番号 17700030 の助成による．

参考文献

[1] Sapid Home Page. http://www.sapid.org/.

[2] C. Boldyreff and R. Kewish. Reverse Engineering to Achieve Maintainable WWW Sites. In *WCRE'01*, pp. 249--257. IEEE CS, 2001.

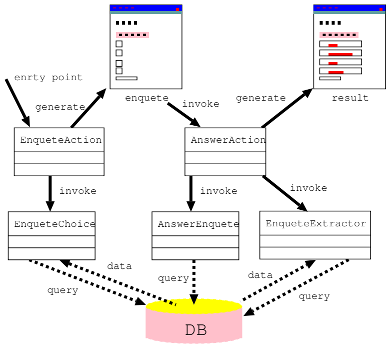

図 9: リファクタリング後のコンポーネント構成

[3] M. Fowler. *Refactoring : Improving The Design of Existing Code*. Addison Wesley, 1999.

[4] N. Koch, H. Baumeister, R. Hennicker1, and L. Mandel. Extending UML to Model Navigation and Presentation in Web Applications. In *Modeling Web Applications Workshop of the UML 2000*, 2000.

[5] Y. Ping and K. Kontogiannis. Refactoring Web sites to the Controller-Centric Architecture. In *CSMR '04*, pp. 204--213, 2004.

[6] Y. Ping, K. Kontogiannis, and T. C. Lau. Transforming Legacy Web Applications to the MVC Architecture. In *STEP '03*, 2003.

[7] F. Ricca and P. Tonella. Analysis and Testing of Web Applications. In *ICSE'01*, pp. 25--34. IEEE CS, 2001.

[8] F. Ricca and P. Tonella. Using Clustering to Support the Migration from Static to Dynamic Web Pages. In *IWPC'03*, pp. 207--216. IEEE CS, 2003.

[9] 崔 銀惠，渡邊 宏． Web アプリケーションのクラス設計仕様に対するモデル化と検証．ソフトウェアテストシンポジウム 2005 予稿集，pp. 90--97，2005.

A　Web 抽象プログラム構文

```
<web_application> ::= { <form_declaration} | <page_declaration> | <action_declaration> }
<form_declaration> ::= "Form" <identifier> "{"
                          "input" "{" [ <variable_list> ";" ] "}"
                          "hidden" "{" [ <variable_list> ";" ] "}" "}"
<page_declaration> ::= "Page" <identifier> "{"
                     { <member_declaration> }
                     <page_generator_declaration>
                     { <transition> } "}"
<member_declaration> ::= <type> <identifier> ["," <identifier>] ";"
<page_generator_declaration> ::= <identifier> "(" ( <variable_list> |
                     "Form" <identifier> ) ")" <statement_block>
<transition> ::= <transition_type> <identifier> "(" <variable_list> ")" <statement_block>
<transition_type> ::= "submit" | "link"
<action_declaration>  ::= ("Action" | "ClientProgram") <identifier> "(" ( <variable_list>
                     | "Form" <identifier> ) ")" <statement_block>
<type> ::= "Page" | "Action" | "ClientProgram" | "Form"
<identifier> ::= "a..z, _, $" { "a..z, _, 0..9" }
<variable_list> ::= ( <identifier> | <form_item> ) { "," ( <identifier> | <form_item> }
<statement_block> ::= "{" { <statement> } "}"
<statement> ::= <expression> ";"
                     | <if_statement>
                     | <do_statement>
                     | <while_statement>
                     | <for_statement>
                     | "break;"
                     | "continue;"
                     | ";"
<expression> ::= <identifier>
                     | <form_input>
                     | <form_item>
                     | <page_generation>
                     | <action_invocation>
                     | "RENDERING_" <identifier> "(" <identifier> ")"
                     | [ "{" <variable_list> "}" "=" ] "PROCESS_" <identifier> "(" <identifier> ")"
<form_input> ::= "input" "(" <identifier> ")"
<form_item> ::= <identifier> "." <identifier>
<page_generation> ::= "new" <identifier> "("<variable_list> ")"
<action_invocation> ::= <identifier> "(" [ <variable_list> ] ")"
<if_statement>
    ::= "if" "(" <expression>  ")" <statement_block> [ "else" <statement_block> ]
<do_statement> ::= "do" <statement_block> "while" "(" <condition_expression> ")" ";"
<while_statement> ::= "while" "(" <condition_expression> ")" <statement_block>
<for_statement> ::= "for" "(" [ <expression> ]";" <condition_expression> ";"
        [ <expression> ] ")"  "{" <statement_block> "}"
<condition_expression> ::= "CONDITION_" <identifier> "(" <variable_list> ")"

------------------------------ meta-symbols --------------------------------
::=    is defined as
|      or
<>     category name
[]     optional items
{}     repetitive items(zero or more times)
```