論文 / 著書情報
Article / Book Information

題目(和文)	クライアント協調型Webアプリケーションの生成法に関する研究
Title(English)	
著者(和文)	田口満久
Author(English)	MITSUHISA TAGUCHI
出典(和文)	学位:博士（工学）, 学位授与機関:東京工業大学, 報告番号:甲第6565号, 授与年月日:2006年3月26日, 学位の種別:課程博士, 審査員:徳田雄洋
Citation(English)	Degree:Doctor of Engineering, Conferring organization: Tokyo Institute of Technology, Report number:甲第6565号, Conferred date:2006/3/26, Degree Type:Course doctor, Examiner:
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

クライアント協調型Webアプリケーションの生成法に関する研究

平成17年度 学位論文

指導教官　徳田 雄洋
提出者　　東京工業大学 大学院
情報理工学研究科 計算工学専攻
田口 満久

目 次

第1章

はじめに

1.1　背景

今日の情報化社会では，情報処理に関する技術が急速に発達しており，とくにハードウェア技術として高性能の CPU や大容量の記憶メディア，高速な通信ネットワークなどの開発が行われている．これに呼応するように，豊富なハードウェア資源を用いてより高度な情報処理を行いたいという要求が高まっている．しかし，高度な情報処理の実現には複雑で大規模なソフトウェアを必要とする場合が多く，その開発・管理には多大な労力と高い技術力が必要とされる．そこで，高品質なソフトウェアの開発・管理を支援するための計算機環境が一層求められている．

一方，近年のネットワーク技術の発達により，世界規模の計算機ネットワークであるインターネットの普及が急速に進んでいる．また，相互運用性の観点から，一組織内や特定の複数組織間をつなぐネットワークにおいても，インターネット関連技術を用いてイントラネットやエクストラネットとよばれるネットワークの構築が進んでいる．このような計算機ネットワーク上では，電子メール，ニュースの配信，World Wide Web（以下，Web という），電子商取引，映像・音声の配信などの多様なシステムが構築されている．

とくに近年，Web を利用して送受信される情報量が増大し扱うべき情報の種類も多様化するにともない，Web 上で効率的に情報を選択・加工・保存・提示するための応用ソフトウェア（以下，Web アプリケーションという）が不可欠となっている．ここで，Web アプリケーションを構成するコンポーネントとしては，大きく 2 つの構成要素に分類することができる．1 つは，Web クライアント側で実行されるプログラムであり，主に利用者に対する情報の提示方法を制御するために用いられる．もう 1 つは，Web クライアントからの要求に応じて Web サーバ側で実行されるプログラムであり，主に利用者が求める情報の適切な選択や利用者から送信された情報の保存を行うために用いられる．さらに，Web サーバ側の情報処理を実現するための実装アーキテクチャとして，大きく分けて 3 種類のアーキテクチャが存在する．1 つ目はプログラム中で Web ページ文書の断片を順次出力するタイプ（以下，サーバプログラム型という）であり [21, 32]，2 つ目は Web ページ文書中にプログラムのコードの断片を埋め込むタイプ（以下，サーバページ型という）であり [5, 17, 20, 28]，3 つ目はサーバプログラムとサーバページを両方用いるタイプ（以下，ページ・プログラム分離型という）である [5]．これらアーキテクチャは，サーバ側プログラムの実装を支援するためのものであり，開発者は対象アプリケーションの性質に応じて適切なアーキテクチャを選択することができる．

しかし，Web アプリケーションの開発・管理には，一般的な計算機ソフトウェアの開発・管理の問題に加え，Web アプリケーション特有の問題が依然として存在する．Web アプリケーションは，Web ページテンプレート，サーバ側とクライアント側の実行プログラムラム，データベースに格納されたデータ，映像・音声ファイルなど，多様な構成要素を含むため，一般的なソフトウェアに比べ構成要素間の一貫性維持が大きな問題となる．また，ネットワーク上で広く公開され使用される Web アプリケーションの場合，利用者から開発者が意図していない要求を受信する場合も多く，システムが本来意図しない処理を実行しないよう注意深く設計・実装を行わなければならない．さらに，近年は Web クライアンとして PDA や携帯電話など多様なデバイスが使用される傾向にあり，Web クライアントの多様性を意識した設計・実装が求められる．

このため，一貫性と安全性を維持するための定型的処理を隠蔽するとともに，多様な Web クライアントに対応するための重複開発を軽減するような，開発支援手法および開発支援システムが求められている．

1.2 研究の目的

一貫性および安全性を備えた Web アプリケーションの開発を支援するための手法として，従来よりアノテーション方式による Web アプリケーション生成系 [2, 3] やダイアグラム方式による Web アプリケーション生成系 [23, 24, 25, 26, 29] が提案されている．アノテーション方式による Web アプリケーション生成系とは，HTML や XHTML[13] により記述された静的な Web ページテンプレートに，簡便な宣言的記述を付加することにより，Web アプリケーションを自動生成する手法である．一方，ダイアグラム方式による Web アプリケーション生成系とは，Web アプリケーションの全体的振る舞いを記述したダイアグラムと各ノードに関する付加的記述から，Web アプリケーションを自動生成する手法である．生成システムを利用することにより，一貫性と標準的な安全性を備えた Web アプリケーションを，Web アプリケーションに関する深い技術的知識を要することなく容易に作成することができる．本研究では，全体的振る舞いを把握しながら迅速に Web アプリケーションを作成することができる，ダイアグラム方式による生成系に注目する．

ここで，従来のダイアグラム方式による Web アプリケーション生成系は，実装アーキテクチャとして主にサーバプログラム型アーキテクチャを対象としてきた．それは，Web ページを動的に作成するための処理プログラムが一箇所にまとまっているというアーキテクチャの性質上，処理プログラムの自動生成が比較的容易だからである．一方で，近年はサーバページ型やページ・プログラム分離型の Web アプリケーションが利用される場合も多く，利用者は対象アプリケーションの性質に応じて適切な実装アーキテクチャを選択して生成できることが望ましい．しかし，従来の生成手法および生成システムは実装アーキテクチャの変更に対する柔軟性に乏しく，これらアーキテクチャへの適用が困難であった．さらに，近年は高性能化した Web クライアントの処理能力を活用する Web アプリケーションが注目されているが，従来手法ではクライアント側プログラムの自動生成は意識されていなかった．

そこで本研究では，まず，従来のサーバプログラム型 Web アプリケーションの生成手法を応用し，サーバページ型 Web アプリケーションの生成手法を提案する．具体的には，従来より用いられている Web アプリケーションの全体的振る舞いを記述するためのダイアグラムを拡張し，サーバページ型 Web アプリケーションの振る舞いを記述するためのダイア

グラムを提案する．そして，上記ダイアグラムからサーバページ型 Web アプリケーションを自動生成するための生成システムを提案する．次に，サーバプログラム型とサーバページ型の生成手法を一般化・上位概念化し，サーバページ型，サーバプログラム型，ページ・プログラム分離型の Web アプリケーションに対応した生成手法を提案する．具体的には，上記ダイアグラムを上位概念化するとともに，生成規則の記述を入れ替えることで同一ダイアグラムから各アーキテクチャ向けのコンポーネントを自動生成できる生成手法を提案する．そして，上記生成手法を拡張し，クライアント側プログラムとサーバ側プログラムが協調して処理を実行するクライアント協調型 Web アプリケーションの生成手法を提案する．具体的には，クライアント側プログラムも生成する生成規則を追加し，同一のダイアグラムからクライアント側処理を含む場合と含まない場合の両方の Web アプリケーションを生成できるようにする．サーバ集中型生成システムの生成対象はサーバページ型生成システムの生成対象を包含し，クライアント協調型生成システムの生成対象はサーバ集中型生成システムの生成対象を包含する．このように，本研究は，とくにクライアント協調型アーキテクチャを意識し，クライアント側プログラムを含むさまざまな Web アプリケーションのアーキテクチャに対応する統一的な Web アプリケーション生成手法の提案を目的とする．

1.3 本論文の構成

本論文の構成は以下の通りである．まず，第 2 章で Web アプリケーションの基本動作や各アーキテクチャの概要および従来の Web アプリケーション開発の問題点について述べる．第 3 章で，提案手法で使用するモデルである，Web アプリケーションの全体的振る舞いを記述するためのダイアグラム（以下，Web 遷移図という）について述べる．そして第 4 章で，Web 遷移図からサーバページ型 Web アプリケーションを自動生成する生成システム（以下，T-Web システムという）について述べる．第 5 章では，T-Web システムをサーバ側で集中的に処理を行う Web アプリケーションの自動生成へと一般化した生成システムについて述べる．第 6 章では，T-Web システムをクライアント協調側 Web アプリケーションの自動生成へと拡張した生成システムについて述べる．第 7 章で Web 遷移図と T-Web システムについて評価を行い，他の開発支援技術との比較を行う．最後に，第 8 章で結論および今後の課題について述べる．

第2章

Webアプリケーション

本章では，Web アプリケーションの動作について説明する．Web アプリケーションのアーキテクチャとして，3 種類のサーバ側実装アーキテクチャとクライアント協調型アーキテクチャを紹介し，各アーキテクチャでの処理の流れおよびプログラムの記述方法を簡潔に述べる．さらに，既存技術を利用した開発手法の問題点を明らかにする．

2.1　Web アプリケーションの基本動作

Web アプリケーションとは，Web クライアントからの要求に応じて Web サーバ側のプログラムが呼び出され，情報の選択・加工・保存・提示などの処理が実行されるネットワークアプリケーションの総称であり，クライアント/サーバ型のソフトウェアアーキテクチャ[6]に基づく．

Web では，Web クライアントと Web サーバ間の通信は W3C（World Wide Web Consortium）によって規格化されているため [14]，Web クライアントと Web サーバが規格を満たしている限り，内部の実装方法に依存せず相互に通信が可能となる．Web クライアントは Web サーバに対し HTTP リクエストを送信し，Web サーバは HTTP リクエストの内容に基づいて適切な Web ページ文書とこれに付随する Web コンテンツを HTTP レスポンスとして Web クライアントに返信する．ここで，大量の Web ページ，Web コンテンツを効率的に管理し，HTTP リクエストとして受信した情報に基づいて Web コンテンツを更新するために，Web アプリケーションの作成が必要となる．

Web クライアントでのイベント発生から利用者に情報が提示されるまでのデータフローを図 2.1 に示す．Web アプリケーションの処理順序は以下の通りである．

1. 利用者の操作により Web クライアント上でイベントが発生する．Web クライアント側でイベントに反応して起動されるプログラムが設定されている場合，まずそのプログラムが実行される．クライアント側プログラムは，Web フォームの入力値や一時記憶領域であるクッキーに保存された値に基づいて計算を行う．その後，Web サーバに対して Web フォームの入力値などを含む HTTP リクエストが送信される．

2. HTTP リクエストを受信した Web サーバは，指定された Web コンテンツの種類に応じて適切な実行プログラムを選択し受信内容を渡す．実行制御が渡されたプログラムは，必要に応じて以下の処理を実行する．

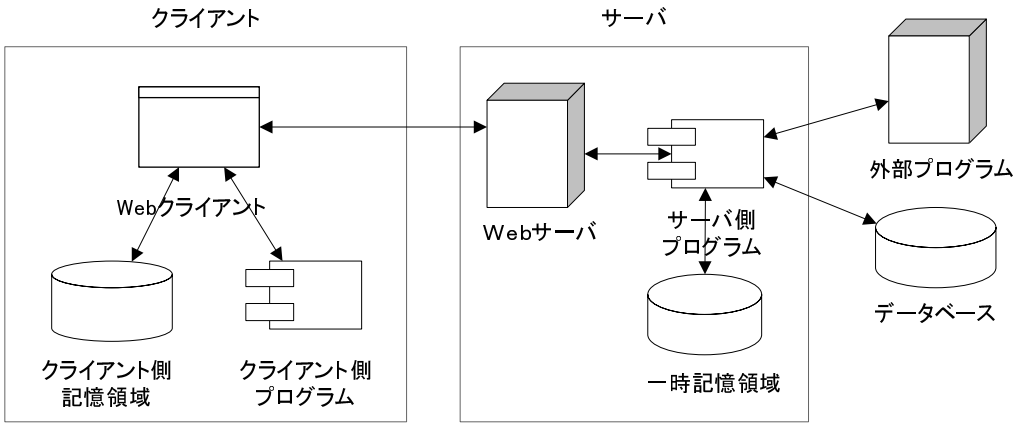

図 2.1 Web アプリケーションの基本動作

- さらに他のプログラムを読み込み実行するか，Web サービスなどの外部プログラムを呼び出し処理結果を取得する
- データベースやファイルシステム上のファイル，メモリ上の一時記憶領域からデータを読み込む
- データベースやファイルシステム上のファイル，メモリ上の一時記憶領域にあるデータを更新する

Web サーバは実行プログラムから処理結果として Web ページ文書を受け取り，Web クライアントに対して HTTP レスポンスとして Web ページ文書を返信する．

3. Web クライアントは HTTP レスポンスを受信する．HTTP レスポンスにクッキーの値が設定されている場合，クッキーの値が更新される．また，Web ページの提示前に起動されるプログラムが設定されている場合，そのプログラムが実行される．その後，Web ページ文書に基づいて Web ページが利用者に提示される．

ただし本論文では，Web ページを視覚的に表示する Web クライアント（以下，Web ブラウザという）用の Web アプリケーションを対象とする [22, 30]．現在の Web で最も多く利用されている形態だからである．また，Web アプリケーションを構成するプログラムとして，Web ブラウザ上で実行されるクライアント側プログラムと Web サーバから実行制御が渡されるサーバ側プログラムを対象とする．これらプログラムの開発は Web アプリケーション特有の問題を多く含む一方，その他の外部プログラムの開発には一般的なソフトウェア開発の手法を有効に適用することができるからである．

以下，クライアント側プログラムとサーバ側プログラムを実装するための代表的技術について概説する．

クライアント側プログラム クライアント側で実行されるプログラムに関する技術は，以下の 2 種類に分類することができる．

クライアント側スクリプト Web ブラウザが Web ページ文書を解釈する際に，その中に埋め込まれたコードを実行する方式である．実装技術としては，Netscape 社が提案した JavaScript や Microsoft 社が提案した VBScript 等が存在する [18, 27]．

ダウンロード型コンポーネント Web ページを制御するためのコンポーネントを Web ページ文書とは別にダウンロードする方式である．実装技術としては，Microsoft 社が提案した ActiveX コントロールや SunMicrosystems 社が提案した JavaApplet，Macromedia 社が提案した Flash 等が存在する [1, 9, 8]．

サーバ側プログラム サーバ側で実行されるプログラムに関する技術は，さらに以下の 2 種類に分類することが可能である．

サーバページ Web ブラウザが Web サーバに配置されたプログラムを含む Web ページ文書を指定して HTTP リクエストを送信する方式である．Web ページ文書中にはコードの断片が埋め込まれており，実行環境によって Web ページ文書全体がプログラムとして解釈される．実装技術として，PHP や Microsoft 社が提案した ASP，ASP.NET，SunMicrosystems 社が提案した JSP 等が存在する [5, 17, 20, 28]．

サーバプログラム Web ブラウザが Web サーバに配置されたプログラムモジュールを指定して HTTP リクエストを送信する方式である．プログラムモジュール中には Web ページ文書の断片を出力するコードが記述されており，処理結果として 1 つの Web ページ文書を出力する．実装技術として，CGI（Common Gateway Interface）や SunMicrosystems 社が提案した JavaServlet 等が存在する [21, 32]．

これら実装技術を組み合わせることで，次節で述べる実装アーキテクチャを実現することができる．なお，クライアント側プログラムとサーバ側プログラムの実装技術は，独立に選択することができる．

2.2 Web アプリケーションのアーキテクチャ

Web アプリケーションのアーキテクチャとして，サーバページ型，サーバクライアント型，ページ・プログラム分離型，クライアント協調型について述べる．

2.2.1 サーバページ型

サーバページ型 Web アプリケーションは，サーバ側プログラムとしてサーバページのみを用い，クライアント側プログラムとの協調を行わない Web アプリケーションである．サーバページ型 Web アプリケーションの実行について，Web サーバが Web ブラウザから HTTP リクエストを受信し Web ブラウザに HTTP レスポンスを返信するまでの流れを図 2.2 に示す．

Web サーバは Web ブラウザから HTTP リクエストを受信すると，リクエスト中の URL（Unified Resource Locator）によって指定された Web ページ文書が，HTML や XHTML のみで記述されたプログラム処理を必要としない文書（以下，静的 Web ページ文書という）か，ASP 文書や JSP 文書等のプログラム処理を要する文書（以下，動的 Web ページ文書という）かを判断する．静的 Web ページ文書である場合，Web サーバは Web ページ文書をそのまま Web ブラウザに対して送信する．動的 Web ページ文書である場合，その Web ページ文書を処理可能な実行環境に渡し，実行環境から返される処理結果としての Web ページ文書を Web ブラウザに対して送信する．したがって，Web サーバは Web ページ文書の種類とそれを処理することができる実行環境の対応関係を保持している必要がある．

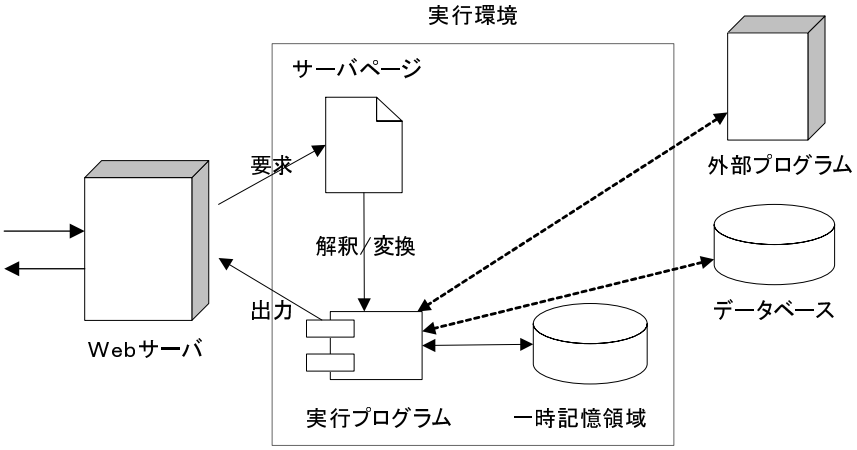

図 2.2 サーバページ型 Web アプリケーションの動作

サーバページの実行環境は，Web ページ文書中に散在する HTML や XHTML で記述された部分（以下，静的記述部分という）とプログラムの部分（以下，プログラム部分という）を区別する．実行環境は，静的記述部分は計算を行わずにそのまま文字列として Web サーバに返し，プログラム部分は計算を行い計算結果に応じた文字列を Web サーバに返す．Web サーバに返される文字列は，全体として 1 つの Web ページ文書を構成する．ここで，プログラムの記述に複数のプログラミング言語の使用が許容されている場合，実行環境は使用されているプログラミング言語に応じて適切な実行モジュールを選択する．

実行環境内の具体的な処理順序は，サーバページの実装アーキテクチャに依存する．例えば，ASP アーキテクチャの場合，ASP のスクリプトエンジンとよばれる実行環境が，ASP 文書全体を 1 つのスクリプト記述とみなして文書の先頭から逐次解釈・計算を行う．ASP 文書では，”<%”と”%>”の間に記述されたコードがプログラムとみなされる．一方，JSP アーキテクチャの場合，JSP コンテナとよばれる実行環境が，JSP 文書全体を 1 つの Java クラスに変換した後その Java クラスのメソッドを呼び出して計算を行う．JSP 文書では，ASP 文書と同様，”<%”と”%>”の間に記述されたコードがプログラムとみなされる．

JSP アーキテクチャに基づいて記述したサーバページの例を図 2.3 に示す．このサーバページは，Web ブラウザから Web フォームの入力を受信し，入力値に基づいて計算を行い，動的に値を埋め込んだ Web ページ文書を出力する．

2.2.2 サーバプログラム型

サーバプログラム型 Web アプリケーションは，サーバ側プログラムとしてサーバプログラムのみを用い，クライアント側プログラムとの協調を行わない Web アプリケーションである．サーバプログラム型 Web アプリケーションの実行について，Web サーバが Web ブラウザから HTTP リクエストを受信し Web ブラウザに HTTP レスポンスを返信するまでの流れを図 2.4 に示す．

Web サーバは Web ブラウザから HTTP リクエストを受信すると，リクエスト中の URL によって指定されたファイルが，静的 Web ページ文書か，Perl や Java 等で記述されたプログラムモジュールかを判断する．静的 Web ページ文書である場合，サーバページ型と同様，

```
<%@ page language="java" contentType="text/html"%>
<%
    // Web フォームからの入力を受け取る
    String email = request.getParameter("email");
    // 入力が空のとき既定値を設定する
    if(email == null || email.equals(""))
        email = new String("nothing");
%>
<html>
<head><title>sample</title></head>
<body>
    Hello, is your E-mail address <%= email %> ?<br/>
    <form action="sample.jsp" method="post">
        Your address: <input type="text" name="email"/>
        <input type="submit"/>
    </form>
</body>
</html>
```

図 2.3 サーバページ型 Web アプリケーションの記述例

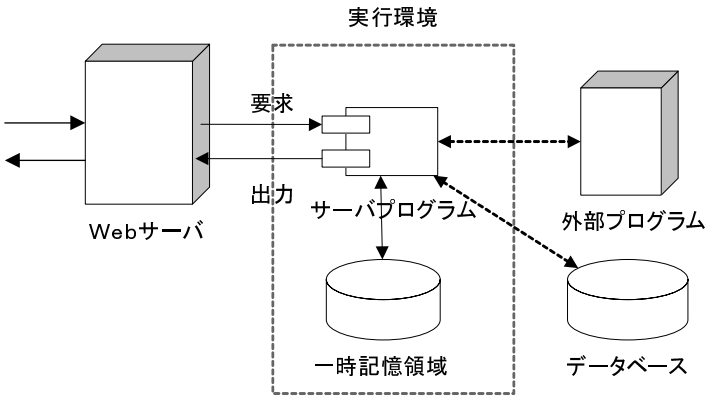

図 2.4 サーバプログラム型 Web アプリケーションの動作

Web サーバは Web ページ文書をそのまま Web ブラウザに対して送信する．プログラムモジュールである場合，そのプログラムモジュールを実行可能な実行環境に渡し，実行環境から返される処理結果としての Web ページ文書を Web ブラウザに対して送信する．したがって，Web サーバはプログラムモジュールの種類とそれを処理することができる実行環境の対応関係を保持している必要がある．

サーバプログラムの実行環境は，プログラムモジュールに HTTP リクエストを入力として与え，プログラムを実行させて計算結果に応じた文字列を出力として得る．Web サーバに返される文字列は，全体として 1 つの Web ページ文書を構成する．ここで，プログラムの記述に複数のプログラミング言語の使用が許容されている場合，実行環境は使用されているプログラミング言語に応じて適切な実行モジュールを選択する．

実行環境内の具体的な処理順序は，サーバプログラムの実装アーキテクチャに依存する．例えば，CGI アーキテクチャの場合，HTTP リクエストの度に Web サーバによって Perl 言語や C 言語などのプログラムを実行する実行プロセスが生成される．一方，servlet アーキテクチャの場合，servlet コンテナとよばれる実行環境が Java クラスを仮想計算機上に読み込み，所定のメソッドを呼び出してスレッドとして計算を行う．servlet プログラムでは，servlet コンテナから呼び出される”doGet()”や”doPost()”などのメソッドを実装する．

servlet アーキテクチャに基づいて記述したサーバプログラムの例を図 2.5 に示す．このサーバプログラムは，図 2.3 に示したサーバページと同様の振る舞いを実現するものである．

2.2.3 ページ・プログラム分離型

ページ・プログラム分離型 Web アプリケーションは，サーバ側プログラムとしてサーバページとサーバプログラムを両方用い，クライアント側プログラムとの協調を行わない Web アプリケーションである．サーバページ型およびサーバプログラム型アーキテクチャでは，HTML や XHTML で記述した Web ページ表示部分とプログラミング言語で記述した処理部分が 1 ファイル中に混在している．このため，大規模な Web アプリケーションの場合，Web ページの表示方法を変更する作業が煩雑となり，複数 Web ページ間で類似する処理プログラムを再利用することも困難となる．そこで，サーバ側プログラムの実装に Model-View-Controller アーキテクチャを適用したものである [6]．ページ・プログラム分離型 Web アプリケーションの実行について，Web サーバが Web ブラウザから HTTP リクエストを受信し Web ブラウザに HTTP レスポンスを返信するまでの流れを図 2.6 に示す．

Web ブラウザは，Web サーバ上に配置されたプログラムモジュールに対して URL を指定して HTTP リクエストを送信する．プログラムモジュールの実行が開始されるまでの手順は，サーバプログラム型アーキテクチャと同様である．ここで，プログラムモジュールは入力値に基づいて計算を行うが，Web ページ文書の出力は行わない．計算を終えたプログラムモジュールは，動的 Web ページ文書に計算結果を入力として与えて実行制御を渡す．動的 Web ページ文書が実行環境内で処理され Web サーバに Web ページ文書を出力するまでの手順は，サーバページ型アーキテクチャと同様である．このようなアーキテクチャによれば，Web ページ表示部分と処理部分を分離することができ，それぞれの部分の変更や再利用が容易となる．

ページ・プログラム分離型の実行環境内の具体的な処理順序は，サーバページとサーバプログラムの実装アーキテクチャに依存する．例えば，JSP/servlet を使用した Model2 アーキテクチャの場合，servlet コンテナがプログラムモジュールである Java クラスを仮想計算

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class sample extends HttpServlet {
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response) throws Exception {
        // Web フォームからの入力を受け取る
        String email = request.getParameter("email");
        // 入力が空のとき既定値を設定する
        if(email == null || email.equals(""))
            email = new String("nothing");
        // HTML を出力する
        PrintWriter writer = response.getWriter();
        writer.println("<html><head><title>sample</title></head><body>");
        writer.println("Hello, is your Email address" + email + "<br/>");
        writer.println("<form action='/servlet/sample' method='post'>");
        writer.println("Your address: <input type='text' name='email'/>");
        writer.println("<input type='submit'/>");
        writer.println("</form>");
        writer.println("</body></html>");
    }
}
```

図 2.5 サーバプログラム型 Web アプリケーションの記述例

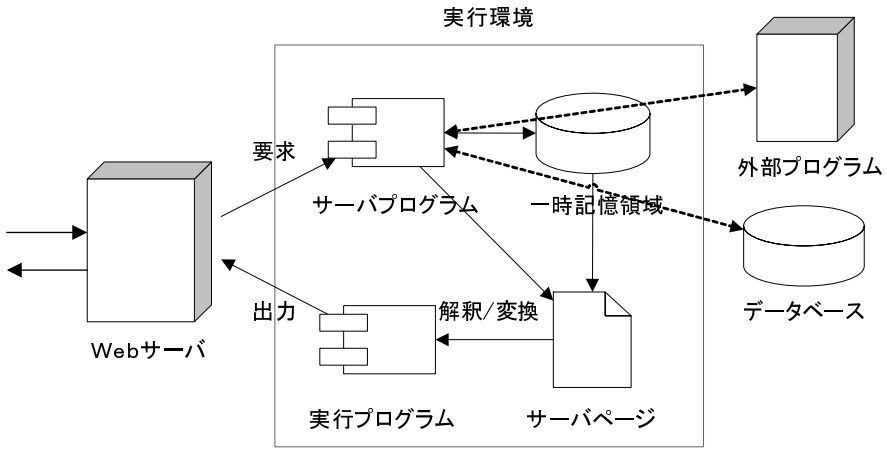

図 2.6 ページ・プログラム分離型 Web アプリケーションの動作

機上に読み込み，JSP コンテナが JSP 文書の処理を行う．servlet プログラムが出力する計算結果は，コンテキスト情報として保持され JSP コンテナへ引渡される．

JSP/servlet の Model2 アーキテクチャに基づいて記述したサーバページとサーバプログラムの例を図 2.7 に示す．このサーバページとサーバプログラムは，図 2.3 に示したサーバページと同様の振る舞いを実現するものである．

2.2.4 クライアント協調型

クライアント協調型 Web アプリケーションは，クライアント側プログラムとサーバ側プログラムとが処理を分担する Web アプリケーションである．上記 3 種類の実装アーキテクチャは，いずれも情報の選択・加工などの処理をすべて Web サーバ側で行っている．しかし，このようなアーキテクチャではサーバ側の計算機やネットワークに負荷が集中するという問題がある．また，情報の表示方法が Web ブラウザがもつ標準的機能に限定されるという問題もある．一方，近年の Web クライアントの高性能化にともない，Web クライアントがもつ処理能力を有効に活用したいという要求が高まっている．そこで，クライアント側プログラムとサーバ側プログラムとが処理の分担を行うクライアント協調型 Web アプリケーションが注目されつつある．なお，サーバ側プログラムの実装アーキテクチャとしては，サーバページ型，サーバプログラム型，ページ・プログラム分離型のいずれのアーキテクチャも採用することができる．クライアント協調型 Web アプリケーションの実行について，Web ブラウザでイベントが発生してから Web ブラウザに新たな Web ページが表示されるまでの流れを図 2.8 に示す．

Web ブラウザでイベントが発生すると，イベントに反応して起動されるスクリプトやコンポーネントなどのクライアント側プログラムが設定されている場合，Web ブラウザはそのプログラムを実行可能な実行環境に渡す．実行環境は，Web フォームの入力値やクッキーの値などを入力として与えてプログラムを実行し，計算結果に応じて Web サーバに対して HTTP リクエストを送信するか否かを判断する．HTTP リクエストを送信する場合，Web サーバから HTTP レスポンスとして新たな Web ページ文書を取得する．その後，Web ページの表示前に起動されるクライアント側プログラムが設定されている場合，Web ブラウザは同様にそのプログラムを実行し，Web ページを表示する．ここで，クライアント側プログラムを実行するためには，あらかじめ Web ブラウザにそのプログラムを実行するための実行環境が組み込まれている必要がある．適切な実行環境が存在しない場合，Web ブラウザは通常そのプログラムを無視する．

クライアント側プログラムの実行環境内の具体的な処理順序は，クライアント側プログラムの実装技術に依存する．例えば，JavaScript の場合，Web ブラウザに組み込まれたスクリプトエンジンがコードの先頭から逐次解釈・計算を行う．また，JavaApplet の場合，Web ブラウザが Java クラスを仮想計算機上に読み込み，所定のメソッドを呼び出してスレッドとして計算を行う．

サーバ側プログラムとして JSP アーキテクチャを用い，クライアント側プログラムとして JavaScript を用いて記述したサーバページの例を図 2.9 に示す．このサーバページは，図 2.3 に示したサーバページの機能に加え，クライアント側で入力値の検査を行うものである．

(a) サーバプログラム

```
import javax.servlet.*;
import javax.servlet.http.*;

public class sample extends HttpServlet {
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response) throws Exception {
        // Web フォームからの入力を受け取る
        String email = request.getParameter("email");
        // 入力が空のとき既定値を設定する
        if(email == null || email.equals(""))
            email = new String("nothing");
        // 値を保存する
        request.setAttribute("email", email);
        // サーバページへ遷移する
        RequestDispatcher dispatcher = getServletContext()
                                       .getRequestDispatcher("/sample.jsp");
        dispatcher.forward(request,response);
    }
}
```

(b) サーバページ

```
    <%@ page language="java" contentType="text/html"%>
    <%
        // 値を読み込む
        String email = (String)request.getAttribute(errorMsg);
    %>
    <html>
    <head><title>sample</title></head>
    <body>
        Hello, is your E-mail address <%= email %> ?<br/>
        <form action="sample.jsp" method="post">
            Your address: <input type="text" name="email"/>
            <input type="submit"/>
        </form>
    </body>
    </html>
```

図 2.7 ページ・プログラム分離型 Web アプリケーションの記述例

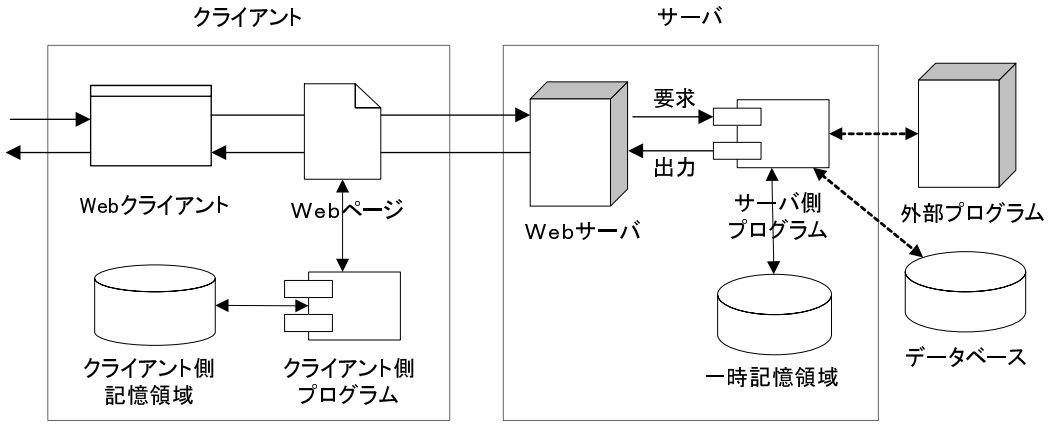

図 2.8 クライアント協調型 Web アプリケーションの動作

```
<%@ page language="java" contentType="text/html"%>
<%
    // Web フォームからの入力を受け取る
    String email = request.getParameter("email");
%>
<html>
<head>
    <title>sample</title>
    <script type="text/JavaScript">
    function check() {
        return document.inform.email.value
                                .match("^[^@]+@[^.]+\\..+$");
    }
    </script>
</head>
<body>
    Hello, is your E-mail address <%= email %> ?<br/>
    <form action="sample.jsp" method="post" name="inform"
                                onSubmit="return check()">
        Your address: <input type="text" name="email"/>
        <input type="submit"/>
    </form>
</body>
</html>
```

図 2.9 クライアント協調型 Web アプリケーションの記述例

以上，Web アプリケーションを実現するアーキテクチャとして，多様な実装アーキテクチャが存在することを述べた．ここで，Web アプリケーションの利用者は，サーバ側の実装アーキテクチャの違いを全く意識する必要がない．このため，既存システムとの相互運用性等のシステム上の制約がある場合を除き，一般に開発者はいずれの実装アーキテクチャも選択することができる．しかし，Web アプリケーション開発後の維持管理の容易性について，各実装アーキテクチャは一般に以下の特徴をもつ．

- サーバページ型 Web アプリケーションは，Web ページ内のデータ構造があまり変化しない場合や動的に値が変化する部分が比較的小さい場合に有効である．このような場合，サーバプログラム型では単に固定された文字列を出力するコードが多くなり，コードの可読性が低下するからである．また，サーバページは，視覚的エディタを使用して Web ページのレイアウトの変更を行うことができる．

- サーバプログラム型 Web アプリケーションは，Web ページ内の動的に値が変化する部分が比較的大きい場合に有効である．このような場合，サーバページ型では条件分岐等の制御構造が複雑になり，Web ページ文書の可読性が低下するからである．

- ページ・プログラム分離型は，Web アプリケーションの規模が大きい場合に有効である．Web アプリケーション内に類似した処理や Web ページ記述が複数ある場合に，それらを一部共通化することができるからである．また，Web ページのレイアウト等を頻繁に変更する場合にも有効である．なお，Web アプリケーションの規模が小さい場合にページ・プログラム分離型を採用すると，Web アプリケーションを構成するファイルの数が増える一方，ページ・プログラム分離型の利点を十分に生かすことができない．このため，小規模な Web アプリケーションでは，サーバページ型やサーバプログラム型を採用した方がよい．

このように，開発後の維持管理を考慮する場合，開発対象の Web アプリケーションの性質に応じて適切な実装アーキテクチャを選択する必要がある．

2.3 Web アプリケーション開発の問題点

従来の Web アプリケーション開発における Web アプリケーション特有の問題について検討する．

2.3.1 一貫性の問題

上述の通り，Web アプリケーションは構成要素として多様なコンポーネントを含む．例えば，Web サーバには JSP 文書と servlet プログラムを配置し，servlet プログラムは関係データベースを参照し，動的に出力される Web ページ文書は JavaScript ファイルと JavaApplet プログラムを参照し，さらに他のサーバプログラムへのハイパーリンクを保持するような Web アプリケーションが考えられる．とくに Web アプリケーションが大規模になると，アプリケーション開発者，データベース技術者，Web ページデザイナーなど，複数の技術者が関与する場合も多い．このような Web アプリケーションでは，開発・管理の過程で以下の一貫性の欠如が生じやすい．

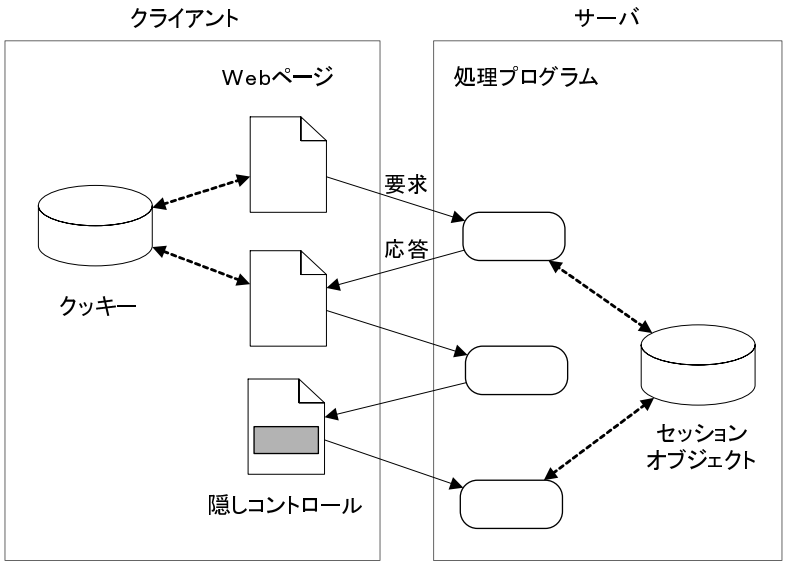

図 2.10 セッション管理の方法

リンク先の不存在 Web ページが参照しているファイルが存在しないか，存在しても設計通りの先を指していない．Web ページ文書の追加・削除や配置位置の変更によって一貫性欠如が生じやすい．その結果，運用によるシステム上の不具合は生じないが，意図した Web ページへ利用者を誘導することができず，Web サイトのユーザビリティ低下を招く．

渡される引数の不一致 サーバページやサーバプログラム間で，渡される引数についての前提条件が崩れている．Web ページ文書中の Web フォームの変更やサーバ側プログラムの変更，データベースのスキーマ変更によって一貫性欠如が生じやすい．その結果，形式的不一致の場合は引数の値の読み取り失敗，意味的不一致の場合はアプリケーションの設計に反した計算が行われる可能性がある．

上記の一貫性欠如のほか，セッション管理に関して意味的な一貫性欠如の問題がある．セッションとは，同一の Web クライアントによって行われる同一の Web サーバに対する単一または一連の HTTP リクエストをいう．Web アプリケーションにおいては，利用者が複数 Web ページにわたる一連の操作を行う場合，操作が完了するまでの間入力値などを保持する必要がある．例えば，図 2.3 に示した Web アプリケーションで，入力内容の確認のための Web ページを追加する場合，確認ページとその後に表示される Web ページの間で入力値を引継ぐ必要がある．しかし，HTTP プロトコルを用いた通信では，1 組の HTTP リクエストとそれに対する HTTP レスポンスをもって接続が切断されるため，入力値などを引継ぐための方法（以下，セッション管理という）が別途必要となる．そこで，Web アプリケーションの実装技術では，以下のセッション管理の方法が用意されている（図 2.10）．

- **隠しコントロール** Web フォームの 1 要素だが Web ブラウザには表示されない項目（以下，隠しコントロールという）を Web ページ文書中に含める方法である．つまり，値の保持はクライアント側で実現される．

- **クッキー** Web ブラウザから参照可能なクッキーに値を一時的に保存する方法である．つまり，値の保持はクライアント側で実現される．
- **セッションオブジェクト** Web サーバのメモリ上のセッション毎に割り当てられる一時的記憶領域（以下，セッションオブジェクトという）に値と保持する方法である．つまり，値の保持はサーバ側で実現される．

しかし，上記方法は値の引継ぎに関して意味的一貫性を保証するものではなく，セッション管理に関して以下の問題が知られている [33]．

開始問題 URL の直接指定や Web ブラウザのブックマーク機能等を用いることで，利用者がアプリケーションの意図しない Web ページからセッションを開始する問題である．開発者が意図したページ遷移を行った場合に存在するクッキーやセッションオブジェクト等が存在しないために，プログラムの実行時エラーが発生する．

続行問題 利用者がセッションの正常終了を待たずに，Web ブラウザを終了したり別 Web サーバへの接続を行う問題である．セッションオブジェクトが存在する場合，Web サーバ上のメモリが開放されず資源の浪費となる．

戻り問題 利用者が Web ブラウザの戻りボタンを使用することで，Web ブラウザ上に表示されている Web ページと Web サーバの状態が相違する問題である．利用者の操作により意図しない副作用が Web サーバ側で発生する可能性がある．

2 度書き問題 利用者が Web フォームのボタンを複数回連続して押すことで，同一内容の HTTP リクエストに基づいて複数回副作用が発生する問題である．一般に，連続した副作用の発生は利用者の意図を反映していない場合が多い．

2 度押し問題 利用者が Web フォームのボタンを複数回連続して押すことで，サーバ側では 1 度目の HTTP リクエストに基づいて計算が正常終了しているために 2 度目以降の計算でエラーが発生し，Web ブラウザに対して最終的にエラーを表す Web ページ文書が返信される問題である．エラーページを見た利用者は，計算が正常終了していないとの錯覚をもつ．

偽造問題 利用者がクッキーの値や Web フォームの入力値等を故意に偽造した上で Web サーバに HTTP リクエストを送信する問題である．利用者のなりすましやプログラムの実行時エラーが引き起こされる可能性がある．

2.3.2 セキュリティの問題

ネットワーク上で広く公開のもと使用される Web アプリケーションでは，開発者が想定した使用方法を遵守するよう利用者に強いることは困難である．このため，アプリケーションが本来意図しない内容の HTTP リクエストを Web サーバが受信する場合がある．Web アプリケーションに対するセキュリティ攻撃として，以下の方法が知られている [34]．

バッファオーバフロー Web アプリケーションに対し許容量を超える入力値を与えることで，プログラムを暴走させたり攻撃者の意図したプログラムを Web サーバ上で実行させる．

クロスサイトスクリプト Web フォームにクライアント側スクリプト含む文字列を入力した場合に，次に表示される Web ページにクライアント側スクリプトが埋め込まれてしまうという脆弱な Web アプリケーションを利用して，Web フォームやクッキーの値を取得する．

パラメータ改ざん Web フォームの入力値（とくに隠しコントロールの値）を改ざんして HTTP リクエストを送信することで，攻撃者の意図した通りに電子メールを送信させたりデータベースを操作したりする．

バックドアとでバックオプション 開発者がデバッグ用に用意した機能で運用段階への移行時に消去し忘れたものを利用する．

強制ブラウズ URL を推測して HTTP リクエストを送信することで，ハイパーリンクが設定されていない非公開のはずのデータやアクセス制限のあるデータを取得する．

セッションハイジャック/リプレイ 他人のセッション ID を推測したり盗むことで，他人になりすまし Web アプリケーションを利用する．

パスの乗り越え 入力値に応じて読み込むべきファイルを指定する Web アプリケーションにおいて，恣意的な入力値を与えることで，攻撃者の意図したファイルの内容を Web ブラウザに表示させる．

SQL の挿入 利用者の入力値に応じて動的に SQL 文を作成する Web アプリケーションにおいて，恣意的な入力値を与えることで，攻撃者の意図した SQL 文を実行させデータベースを操作する．

OS コマンドの挿入 入力値に応じて外部コマンドを実行する Web アプリケーションにおいて，恣意的なな入力値を与えることで，攻撃者の意図したプロセスを Web サーバ上で実行させる．

クライアント側コメント Web 文書中に記述されたコメントなどの Web ブラウザに表示されない情報を手がかりに，Web アプリケーションの内部構造を推測する．

エラーコード エラー発生時の処理が不十分な Web アプリケーションにおいて，恣意的にエラーが発生する入力を与えエラーメッセージを取得することで，攻撃可能な入力値を推測する．

上記攻撃に対しては，専用のハードウェアを利用したファイアウォールや不正侵入検知システムでは検知・防御が困難であることが知られており，ソフトウェアによる防御が必要不可欠である． したがって，開発者はセキュリティに関し注意深く実装を行うとともに，新たに発見される攻撃手法に対しても迅速に防御方法を組み込めるよう配慮した設計を行う必要がある． しかし，このようなセキュリティを十分考慮した Web アプリケーションの開発には，多大な労力と高い技術力を必要とする．

2.3.3 クライアントの多様性の問題

近年のハードウェアの小型化により，携帯電話や PDA などの携帯端末も Web クライアントとしての機能を有するようになり，携帯端末からの Web サーバへのリクエストが増加

している．また，Web クライアントとして汎用的計算機が用いられる場合でも，組み込まれているプラグインによって実行可能なクライアント側プログラムの種類が異なる．そのため，Web アプリケーションは Web ブラウザの種類やプラグインの存在に応じて振る舞いを変える必要がある．

具体的には，Web アプリケーションは Web ブラウザが解釈可能な Web ページ文書を出力する必要がある．汎用的計算機の Web ブラウザは通常 HTML，XHTML，XML などで記述された Web ページ文書を解釈することができるが，携帯端末の Web ブラウザは CompactHTML や WML などで記述された Web ページ文書を解釈することができる．そのため，Web アプリケーションは HTTP リクエストから Web ブラウザの種類を判別し，出力する Web ページ文書の記述言語を変える必要がある．また，クライアント側プログラムを使用する場合，Web ブラウザに必要なプラグインが組み込まれていないとプログラムが無視されることから，クライアント側プログラムを使用しない Web ページも同時に用意しておく必要がある．さらに，クライアント側スクリプトの解釈が Web ブラウザの種類によって異なる場合があるため，あらかじめ代表的な Web ブラウザで振る舞いの確認を行う必要がある．

2.4 従来の Web アプリケーション生成系

上記の一貫性と安全性の問題を解決するため，これまで幾つかの Web アプリケーション生成系が提案されている．ここでは，ダイアグラム方式による Web アプリケーション生成系として，PF-Web システムと T-Web システムについて述べる [29, 25]．

PF-Web システムは，Web 遷移図とよばれるダイアグラムから Web アプリケーションを自動生成する生成システムである．この生成手法では，Web アプリケーションの実行をパイプ/フィルタアーキテクチャに基づくアプリケーションの実行とみなす．フィルタに相当する処理プログラムの計算を関数として宣言的に記述することにより，手続き的なプログラムの記述を必要とすることなく Web アプリケーションの生成を行う．PF-Web システムが生成した Web アプリケーションは，副作用の二重発生防止などのセッション管理機能をもつ．一方，副作用発生後に利用者が Web ページの履歴を参照することを許容しないなど，Web ブラウザ振る舞いの意味を固定的に解釈する．また，フィルタとして用意しているプログラムモジュールの粒度が比較的小さいという特徴をもつ．PF-Web システムの現在の実装は，CGI アプリケーションのみを生成対象としている．

T-Web システムは，PF-Web システムと同様，Web 遷移図から Web アプリケーションを自動生成する生成システムである．この生成手法では，処理プログラムの計算としてあらかじめ用意された汎用的なプログラムテンプレートから選択し適用することで，手続き的なプログラムの記述を必要とすることなく Web アプリケーションの生成を行う．T-Web システムが生成した Web アプリケーションは，入力値検査など標準的なセキュリティ維持のための機能をもつ．一方，プログラムテンプレートの粒度が大きく，生成対象が特定の実装アーキテクチャに特化しているという特徴をもつ．T-Web システムの現在の実装は，CGI アプリケーションやと JSP/servlet アプリケーションのみを生成対象としている．

第3章

Web遷移図

本章では，Web 遷移図の概念，構成要素および構成方法について述べる．Web 遷移図は，従来の Web アプリケーション生成系の一部でも用いられているが [24, 23, 25, 26, 29]，本研究ではこれを再定義し一般化および拡張を行う．

3.1　Web 遷移図の概念

本研究で用いる Web 遷移図とは，Web アプリケーションの全体的な振る舞いを記述するためのダイアグラムであり，Web アプリケーションの構成要素をノードとし構成要素間のデータフローなどをエッジとして表す有向グラフである．Web アプリケーションは，Web ページ間の静的関係とプログラムによるデータフロー制御という 2 つの側面を有する．しかし，Web ページから処理プログラムが呼び出され次の Web ページが生成されるという実行列もつ Web アプリケーションでは，ハイパーリンクによる Web ページ間の遷移とデータフローを分離してモデル化することは直感的理解を妨げる．そこで Web 遷移図は，この 2 つの側面を 1 つのダイアグラムとして同時に表現することで，Web アプリケーションの全体的な振る舞いの直感的理解を容易にしている．

Web 遷移図では，Web アプリケーションの動作列を一連のパイプ/フィルタアーキテクチャに基づくアプリケーションの動作列とみなす [6]．Web アプリケーションの実行は，Web ページの表示とその Web ページから発生する HTTP リクエストに基づく処理プログラムの実行とを交互に繰り返すことにより進行する．この点で Web アプリケーションのアーキテクチャは，パイプとフィルタが交互に結合することにより計算が進行するアーキテクチャであるパイプ/フィルタアーキテクテャと共通点を有する．さらに，Web アプリケーションの処理プログラムは他の処理プログラムと変数を共有しない点でも，パイプ/フィルタアーキテクテャにおけるフィルタと共通する．

そこで，Web 遷移図では，パイプ/フィルタアーキテクテャにおけるパイプとフィルタを，Web ページと処理プログラムにそれぞれ対応付けて意味を理解する（図 3.1）．Web ページは単にデータを中継するか利用者からの入力を受理するのみで計算を行わないためパイプに相当し，入力値から出力値を計算する処理プログラムはフィルタに相当する．また，データベース等の記憶領域との入出力は，記憶領域が保持するデータ全体がフィルタに対し入力または出力されると考える．例えば，データベーステーブルに対してレコードの検索や追加，更新，削除を行うフィルタは，そのテーブル内の全レコードを入力として受け取り計算に用

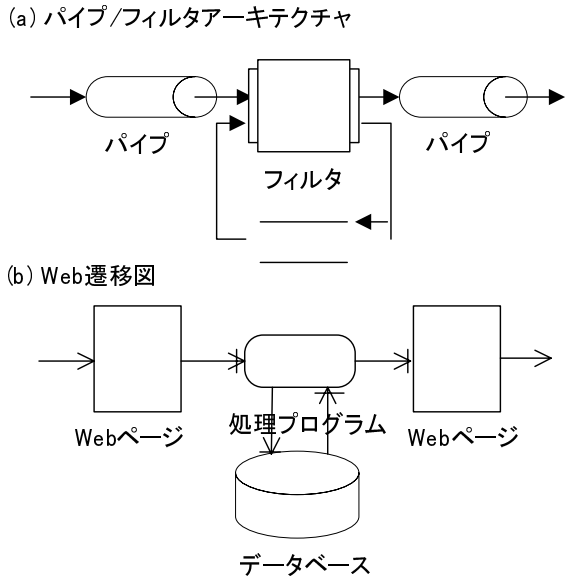

図 3.1 Web 遷移図とパイプフィルタアーキテクチャ

いる．フィルタが出力するレコードの集合はそのままデータベーステーブルに上書きされると考える．

Web 遷移図の基本形は以下の式で定義される．

$$
\begin{aligned}
WTD &= (P, E, D, \sigma, s, F) \\
P &:= \{p_1, p_2, ..., p_{|P|}\} \\
E &:= \{e_1, e_2, ..., e_{|E|}\} \\
D &:= \{d_1, d_2, ..., d_{|D|}\} \\
\sigma &: P \times (E \cup \{\epsilon\}) \rightarrow P \times 2^{D \cup \overline{D}} \quad \overline{D} := \{\overline{d_i} \mid d_i \in D\} \\
s &\in P \\
F &\subseteq P
\end{aligned}
$$

Web 遷移図 WTD は，Web ページの集合 P，処理プログラムの集合 E，記憶領域の集合 D，状態遷移関数 σ，開始状態 s，終了状態の集合 F で構成される．上記定義は，ϵ 遷移のある非決定性有限順序機械を拡張したものであり，Web ページが状態に，処理プログラムが入力に，記憶領域が出力にそれぞれ対応する．また，ハイパーリンクによる Web ページ間の遷移は，データフローをともなわない遷移と解釈し，ϵ 遷移として定義する．状態遷移関数 σ では，処理プログラム p_k が記憶領域 d_k を更新する場合，遷移後の出力を $(p_k, \{d_k\})$ として定義し，p_k が d_k からデータを取得する場合，$(p_k, \{\overline{d_k}\})$ として定義する．すなわち，$\overline{D}$ は D の要素うち，処理プログラムによってデータを読み取り可能な記憶領域の集合である．なお，順序機械では一般に終了状態を定義しないが，Web アプリケーションにおけるセッションの正常終了を判断するために終了状態を定義している．

上記定義によれば，Web 遷移図はセッションの開始ページとして 1 個の Web ページをもち，セッションの正常終了ページとして 0 個以上の Web ページをもつ．また，Web ページ間の遷移として，処理プログラムを介して発生する遷移とハイパーリンクによる遷移をも

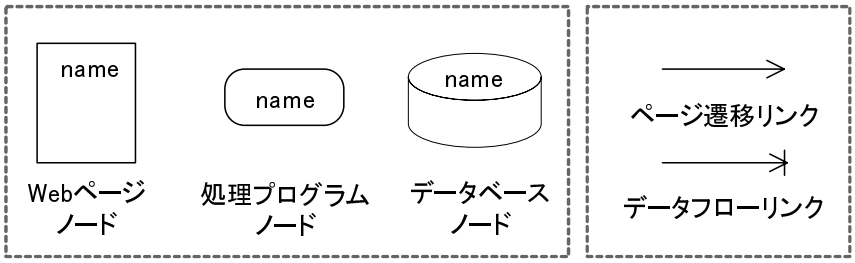

図 3.2 Web 遷移図の基本形の表記

ち，遷移にともない 0 個以上の記憶領域とのデータ入出力が行われる．記憶領域としては，例えばデータベーステーブル，サーバメモリ上の一時記憶領域，Web ブラウザが保持するクッキー，データ入出力を行う Web サービスなどの外部プログラム等が該当する．

Web 遷移図の基本形の視覚的表記を図 3.2 に示す．このダイアグラムは，3 種類のノードと 2 種類のエッジから構成される．

Web ページノード 形式的定義の Web ページに対応する．長方形の内側に識別子としての文字列を付したノードを用いて 1 つの Web ページテンプレートを表現する．Web ページノードが開始状態である場合には左上に入力矢印を付加し，終了状態である場合には長方形の線を二重線にして表記する．

処理プログラムノード 形式的定義の処理プログラムに対応する．楕円の内側に識別子としての文字列を付したノードを用いて 1 つの処理プログラムを表現する．

データベースノード 形式的定義の記憶領域に対応する．円柱形の内側に識別子としての文字列を付したノードを用いて 1 つの記憶領域を表現する．

ページ遷移リンク 形式的定義の ϵ 遷移に対応する．Web ページノードから Web ページノードへの矢印線を用いて 1 つのハイパーリンクを表現する．

データフローリンク 形式的定義の ϵ 遷移以外の遷移に対応する．Web ページノードから処理プログラムノードへのリンクを用いて HTTP リクエストを表現し，処理プログラムノードから Web ページノードへのリンクを用いて HTTP レスポンスを表現する．また，処理プログラムノードからデータベースノードへのリンクを用いてデータ出力を表現し，データベースノードから処理プログラムノードへのリンクを用いてデータ入力を表現する．なお，データフローリンクはページ遷移リンクと区別するため，横線付き矢印線として表記する．

Web 遷移図の基本形の例を図 3.3 に示す．このダイアグラムは，簡単な掲示板アプリケーションの全体的振る舞いを記述したものである．この Web アプリケーションのセッションは Web ページ”bbs”から開始される．利用者は Web ページ”bbs”で掲示板に書き込むデータを入力する．入力データが所定の書式に合致すれば，処理プログラム”write”によって記憶領域”book”の保持するデータが更新され Web ページ”success”に遷移する．一方，入力データが所定の書式に合致しなければ Web ページ”error”に遷移する．Web ページ”error”からは Web ページ”bbs”にハイパーリンクにより遷移することができる．さらに，Web ペー

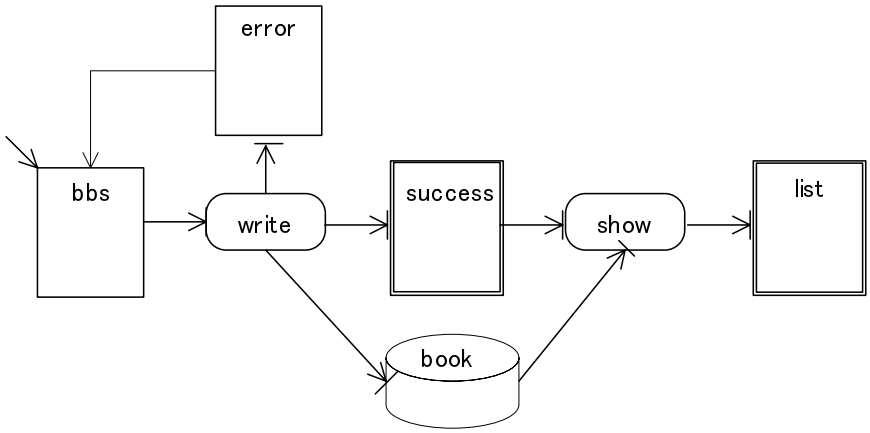

図 3.3 Web 遷移図の基本形の例

ジ”success”からは，記憶領域”book”の保持するデータを取得する処理プログラム”show”を介して，過去の記事一覧を表示する Web ページ”list”に遷移することができる．Web ページ”success”と”list”は，セッションの正常終了ページに指定されている．このように，Web 遷移図を用いることで，Web アプリケーションにおける Web ページ間の遷移と構成要素間のデータフローを 1 つのダイアグラムとして直感的に記述することができる．

この Web 遷移図の形式的定義は以下の通りである．

$$
\begin{aligned}
P &= \{bbs, success, error, list\} \\
E &= \{write, show\} \\
D &= \{book\} \\
\sigma &= \{(bbs, write) \mapsto (success, \{book\}),\ (bbs, write) \mapsto (error, \phi), \\
&\quad\ (error, \epsilon) \mapsto (bbs, \phi),\ (success, show) \mapsto (list, \{\overline{book}\})\} \\
s &= bbs \\
F &= \{success, list\}
\end{aligned}
$$

3.2 Web 遷移図の拡張

3.2.1 サーバページ型への拡張

上記 Web 遷移図の基本形は，Web アプリケーションのアーキテクチャに非依存であるが，抽象的な記述であるため実行可能な Web アプリケーションを自動生成するのに十分な情報を保持していない．そこで，従来のサーバプログラム型生成システムにおいてもダイアグラムの記述力を高めるべく，サーバプログラム型 Web アプリケーションの記述のために拡張した Web 遷移図を用いている [25, 26]．本節では，上記 Web 遷移図の基本形をサーバページ型 Web アプリケーションの記述のために拡張した Web 遷移図を定義する．

サーバページ型アーキテクチャのために拡張した Web 遷移図の視覚的表記を図 3.4 に示す．このダイアグラムは，8 種類のノードと 2 種類のエッジから構成される．Web 遷移図の基本形における Web ページノードは固定 Web ページノード，出力 Web ページノードに下

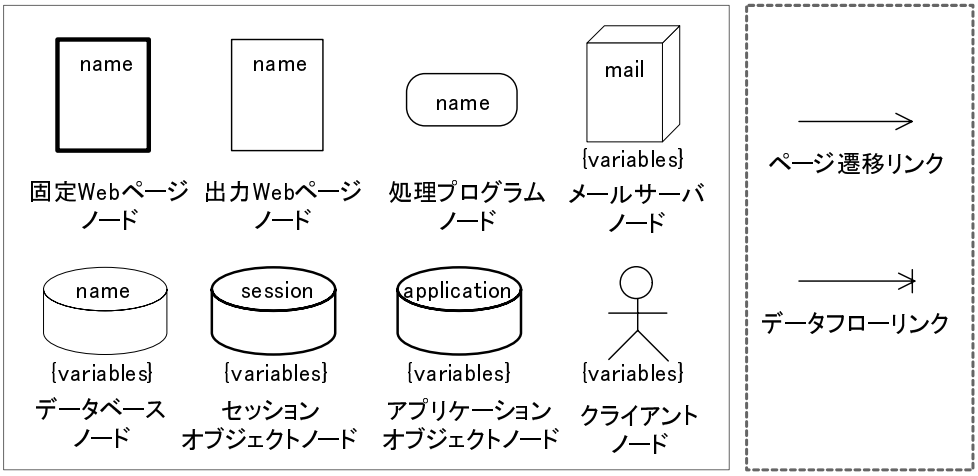

図 3.4 サーバページ型 Web 遷移図の表記

位概念化され，データベースノードはデータベースノード，セッションオブジェクトノード，アプリケーションオブジェクトノード，クライアントノード，メールサーバノードに下位概念化される．

固定 Web ページノード 静的 Web ページ文書で表現される 1 つの Web ページテンプレートを表し，太線枠の長方形を用いて表記する．

出力 Web ページノード 動的 Web ページ文書で表現される 1 つの Web ページテンプレートを表し，細線枠の長方形を用いて表記する．処理プログラムの出力する値が Web ページテンプレート中に動的に埋め込まれる．

処理プログラムノード 基本形と同様の意味をもち，同様の表記法を用いる．

データベースノード 1 つのデータベーステーブルを表し，細線枠の円柱形を用いて表記する．ノードの下部に付した”{”と”}”の間にテーブルカラム名を列挙する．

セッションオブジェクトノード 同一セッションからのみ認識可能なサーバメモリ上の一時記憶領域（以下，セッションオブジェクトという）を表し，識別子を session とする太線枠の円柱形を用いて表記する．ノードの下部に付した”{”と”}”の間に Web 遷移図が表すセッション内で保持されるべきセッションオブジェクト名を列挙する．

アプリケーションオブジェクトノード すべてのセッションから共通に認識可能なサーバメモリ上の一時記憶領域（以下，アプリケーションオブジェクトという）を表し，識別子を application とする太線枠の円柱形を用いて表記する．ノードの下部に付した”{”と”}”の間にそのアプリケーションの範囲内で保持されるべきアプリケーションオブジェクト名を列挙する．

クライアントノード Web ブラウザが保持するクッキーを表し，人間の形を用いて表記する．ノードの下部に付した”{”と”}”の間にクッキー名を列挙する．

メールサーバノード SMTP サーバおよび POP サーバを表し，識別子を mail とする立方体を用いて表記する．

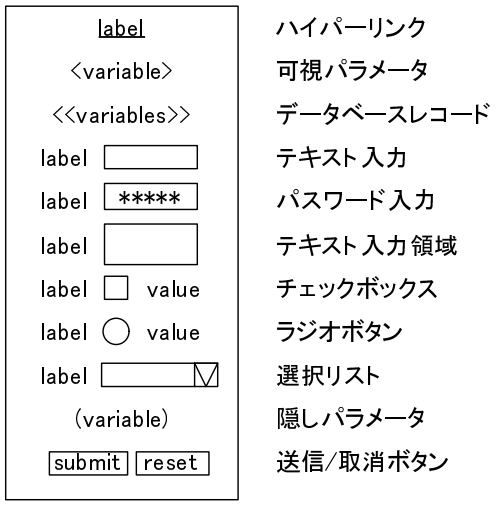

図 3.5 サーバページ型 Web 遷移図の Web ページ要素

ページ遷移リンク 固定 Web ページノードまたは出力 Web ページノードから固定 Web ページノードへの矢印線を用いて 1 つのハイパーリンクを表現する．

データフローリンク 固定 Web ページノードまたは出力 Web ページノードから処理プログラムノードへのリンクを用いて HTTP リクエストを表し，処理プログラムノードから出力 Web ページノードへのリンクを用いて HTTP レスポンスを表す．処理プログラムノードとデータベースノードとの間のリンクを用いてデータベーステーブルのレコード更新およびレコード取得を表す．処理プログラムノードとセッションオブジェクトノードとの間のリンクを用いてセッションオブジェクトが保持するデータの更新および取得を表す．処理プログラムノードとアプリケーションオブジェクトノードとの間のリンクを用いてアプリケーションオブジェクトが保持するデータの更新および取得を表す．処理プログラムノードとクライアントノードとの間のリンクを用いてクッキーが保持するデータの更新および取得を表す．また，処理プログラムノードからメールサーバノードへのリンクを用いて電子メールの送信を表し，メールサーバノードから処理プログラムノードへのリンクを用いて電子メールの受信を表す．表記法は基本形と同様である．なお，1 つの処理プログラムノードから複数の出力 Web ページノードへのリンクがある場合，カード（[...]）を用いて各リンクの遷移条件を記述することができる．

また，各固定 Web ページノードおよび出力 Web ページノードは 0 個以上の Web ページ要素を保持する．拡張した Web 遷移図で使用する Web ページ要素を図 3.5 に示す．固定 Web ページノードは可視パラメータ要素とデータベースレコード要素以外の Web ページ要素を保持することができ，出力 Web ページノードはすべての Web ページ要素を保持することができる．

ハイパーリンク要素 他の Web ページへのハイパーリンクを表し，識別子である文字列に下線を付して表記する．

可視パラメータ要素 処理プログラムにより値が埋め込まれる位置を表し，”<”と”>”の間に識別子である変数名を記載して表記する．

データベースレコード要素 処理プログラムにより値の配列が一覧として埋め込まれる位置を表し，”<<”と”>>”の間に識別子である変数名を列挙して表記する．

テキスト入力要素 Web フォームにおける 1 行の文字列を入力するための入力領域を表す．

パスワード入力要素 Web フォームにおけるパスワードを入力するための入力領域を表す．

テキスト入力領域要素 Web フォームにおける複数行の文字列を入力するための入力領域を表す．

チェックボックス要素 Web フォームにおける一覧から複数の値を選択するための選択領域を表す．

ラジオボタン要素 Web フォームにおける一覧から高々1 個の値を選択するための選択領域を表す．

選択リスト要素 Web フォームにおける一覧から高々1 個の値を選択するためのリスト型選択領域を表す．

隠しパラメータ要素 Web フォームにおける隠しコントロールが埋め込まれる位置を表す．

送信ボタン要素 Web フォームにおける入力値を送信するためのボタンを表す．

取消ボタン要素 Web フォームにおける入力値を初期化するためのボタンを表す．

拡張した Web 遷移図を用いて記述したダイアグラムの例を図 3.6 に示す．このダイアグラムは，図 3.3 に示した掲示板アプリケーションの記述を拡張したものである．データベーステーブル”book”はデータとして名前，メールアドレス，日付，記事を保持する．利用者は Web ページ”bbs”で名前，メールアドレス，記事を入力することができ，日付は処理プログラム”write”によって自動的に付与される．Web ページ”list”では，名前，日付，記事の一覧が表示される．このように，拡張した Web 遷移図は，Web 遷移図のもつ理解容易性を維持し，典型的なサーバページ型 Web アプリケーションを自動生成するのに必要な記述力を有している．

3.2.2 サーバ集中型への一般化

従来のサーバプログラム型 Web アプリケーションのための Web 遷移図や上記サーバページ型 Web アプリケーションのための Web 遷移図は，Web アプリケーションの各アーキテクチャに依存したものであり，対象アプリケーションの自動生成には十分な高い記述力を有している．しかし，アーキテクチャ依存のダイアグラムを用いたのでは，事後的に実装アーキテクチャを変更する必要が生じた場合や複数の実装アーキテクチャ用の Web アプリケーションを作成する必要がある場合に，Web 遷移図の記述を修正する作業が煩雑となる．そこで本節では，一般化ないし抽象化を行い，実装アーキテクチャに非依存な Web 遷移図を定義する．一般化した Web 遷移図は，サーバ側プログラムがクライアント側プログラムとは協調を行わないアーキテクチャである，サーバページ型アーキテクチャ，サーバプログラム

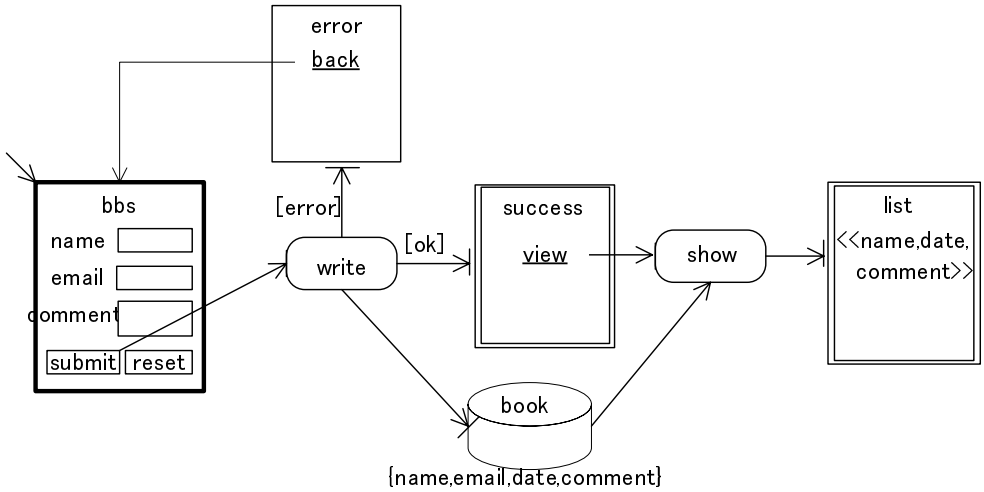

図 3.6 サーバページ型 Web 遷移図の例

型アーキテクチャ，ページ・プログラム分離型アーキテクチャのいずれかのアーキテクチャ（以下，サーバ集中型アーキテクチャという）に基づく Web アプリケーションを記述するものである．

ここで，Web 遷移図の記述をデータベース等のデータ保持手段や Web ページ文書の言語仕様から独立とするために，データ型の概念を導入する．記憶領域が保持するデータは複合データ型により規定され，複合データ型は 1 つ以上の基本データ型の集合として規定される．各基本データ型に対し，受理可能なデータ形式と Web フォームにおける入力手段を指定する．基本データ型のデータ形式は，正規表現による文字列パターン，最大文字列長，最小文字列長で構成される．これらは，以下のように XML Schema のデータ型定義の文法に従って記述する [16]．

```
<xsd:simpleType name="ZipCode" base="xsd:string">
  <xsd:pattern value="^[0-9]{3}-[0-9]{4}$"/>
  <xsd:maxlength value="8"/>
  <xsd:minlength value="8"/>
</xsd:simpleType>
```

上記記述によれば，日本の郵便番号を表す ZipCode は，文字列パターンが 3 桁の数字+ハイフン+4 桁の数字，最大文字列長と最小文字列長がともに 8 文字と定義される．また，基本データ型の入力手段は，以下の XForms で規定する入力手段のいずれか 1 つを指定する [15]．

- **input** 1 行の文字列を入力するための入力領域.
- **secret** 入力した文字列を表示しない入力領域.
- **textarea** 複数行の文字列を入力するための入力領域.
- **select** 項目一覧から複数の値を選択するための選択領域.
- **select1** 項目一覧から 1 つの値を選択するための選択領域.

開発者は上記方法により任意の基本データ型を定義することができるが，Web アプリケーションで頻繁に用いられる以下の既定の基本データ型に名前を付与して使用することもできる．

Boolean True または False の真偽値を表す文字列．入力手段は select1．

Date ”YYYY-mm-DD”形式の日付を表す文字列．入力手段は input．

DateTime ”YYYY-mm-DD HH:MM:SS”形式の日時を表す文字列．入力手段は input．

Email 電子メールアドレスを表す文字列．入力手段は input．

Integer 整数を表す文字列．入力手段は input．

Number 小数を含む数値を表す文字列．入力手段は input．

Password パスワードを表す半角英数字からなる文字列．入力手段は secret．

TelNumber 電話番号を表す文字列．入力手段は input．

Text 改行を含む任意の文字列．入力手段は textarea．

Time ”HH:MM:SS”形式の日時を表す文字列．入力手段は input．

Word 改行含まない文字列．入力手段は input．

ZipCode 郵便番号を表す文字列．入力手段は input．

定義した複合データ型に基づいてサーバ集中型 Web 遷移図を構成する．ダイアグラムで用いるノードとエッジの種類は Web 遷移図の基本形と同様であり，表記法も図 3.2 に示したものを用いる．

Web ページノード 1 つの Web ページテンプレートを表す．静的 Web ページと動的 Web ページの違いは後述の Web ページ要素に基づいて判断されるため，開発者が意識的に区別する必要はない．

処理プログラムノード 1 つの処理プログラムを表す．

データベースノード 1 つの記憶領域を表す．関係データベース，XML データベース，サーバメモリ上の一時記憶領域，Web ブラウザがもつ一時記憶領域，Web サービスなど，データの保持・更新の機能を有するものを指す．具体的な実装技術は，パラメータ値として指定する．

ページ遷移リンク 1 つのハイパーリンクを表す．

データフローリンク Web ページと処理プログラムとの間のデータ送受信または処理プログラムと記憶領域との間のデータ入出力を表す．

各 Web ページノードは 0 個以上の Web ページ要素を保持する．サーバ集中型 Web 遷移図で使用する Web ページ要素を図 3.7 に示す．

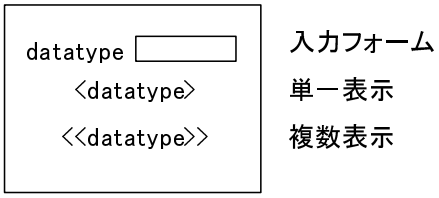

図 3.7 サーバ集中型 Web 遷移図の Web ページ要素

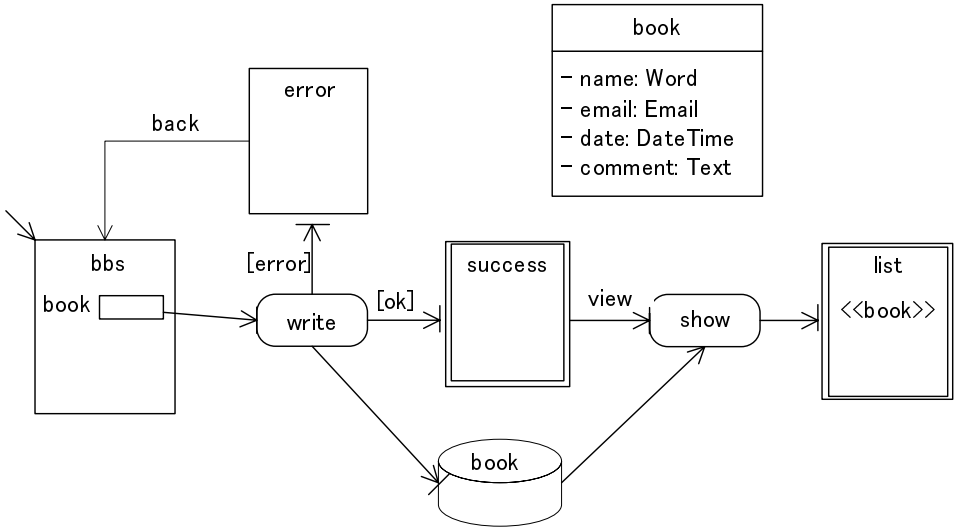

図 3.8 サーバ集中型 Web 遷移図の例

入力フォーム要素 Web フォームによる入力手段を表し，複合データ型名を表記する．複合データ型に含まれる各基本データ型のデータを入力する入力手段を意味する．具体的な入力手段は，複合データ型を構成する各基本データ型で指定した入力手段に従う．

単一表示要素 処理プログラムにより値が埋め込まれる位置を表し，”<”と”>”の間に複合データ型名を表記する．

複数表示要素 処理プログラムにより値の配列が一覧として埋め込まれる位置を表し，”<<”と”>>”の間に複合データ型名を表記する．

一般化した Web 遷移図を用いて記述したダイアグラムの例を図 3.8 に示す．このダイアグラムは，図 3.6 に示した掲示板アプリケーションと同一の Web アプリケーションを記述したものである．複合データ型”book”は，構成要素として Word 型の”name”，Email 型の”email”，DateTime 型の”date”，Text 型の”comment”を基本データ型としてもつ．Web ページノード”bbs”は入力フォーム要素をもち，Web ページ”list”は複数表示要素をもつ．このように，一般化した Web 遷移図は，Web 遷移図のもつ理解容易性を維持し，Web アプリケーションの実装アーキテクチャや記憶領域の実装技術に非依存となっている．

3.2.3 クライアント協調型への拡張

本節では，サーバ集中型 Web アプリケーションのための Web 遷移図を拡張し，クライアント協調型 Web アプリケーションを記述するための Web 遷移図を定義する．拡張した Web 遷移図では，クライアント側プログラムとサーバ側プログラムの振る舞いは分離して記述せず，プログラムによるデータ処理の一部がクライアント側とサーバ側のどちらで実行されるかを隠蔽する．サーバ集中型 Web 遷移図とクライアント協調型 Web 遷移図との差異は，複合データ型にメタデータを付与する点と，処理プログラムノードにステレオタイプを付与する点である．これは，Web 遷移図を分析することで，クライアント側で実行可能なデータ処理を判断できるようにするためである．

各複合データ型に対し以下のメタデータを付与する．それぞれの項目は，制限を受けない（+），制限を受ける（-）のいずれかの値をとる．

- **R（Read）** データの取得にユーザ認証が必要か否かを規定する．
- **W（Write）** データの追加にユーザ認証が必要か否かを規定する．
- **U（Update）** データの更新にそのデータを追加したユーザであることの確認が必要か否かを規定する．
- **S（Save）** Web ブラウザでのデータの複製が制限されるか否かを規定する．

既定では R+W+U-S+が付与される．これは，データの取得・追加にユーザ認証は不要でありクライアント側でのデータ処理は制限されないが，既存データの更新はそのデータを追加したユーザのみ認められることを意味する．

また，各処理プログラムノードに対し以下のステレオタイプのうちいずれか 1 つを付与することができる．ステレオタイプを付与しない処理プログラムノードは，プログラムによる処理内容が未定義であると解釈する．

Get データを記憶領域から取得するプログラムであることを明示する．取得するデータの条件を入力とし，取得したデータを識別する識別子を出力する．

Create データを記憶領域に追加するプログラムであることを明示する．追加するデータの値を入力とし，追加したデータを識別する識別子を出力する．

Modify 記憶領域が保持するデータを更新するプログラムであることを明示する．更新するデータの条件とデータの値を入力とし，更新したデータを識別する識別子を出力する．

Delete 記憶領域からデータを削除するプログラムであることを明示する．削除するデータの条件を入力とし，出力は返さない．

ステレオタイプを付与した処理プログラムノードと Web ページ要素との典型的な関係を図 3.9 に示す．

- 入力フォーム要素からノード <<Get>> へのリンクは，Web フォームの入力値を検索条件とするデータ検索を意味する．
- 入力フォーム要素からノード <<Create>> へのリンクは，Web フォームの入力値に基づくデータの追加を意味する．

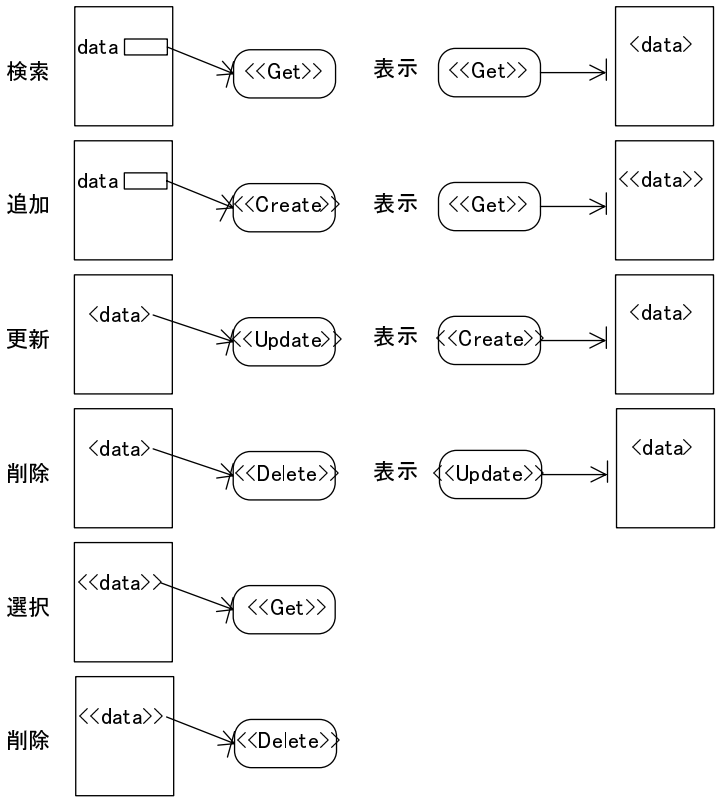

図 3.9 クライアント協調型 Web 遷移図のデータフロー例

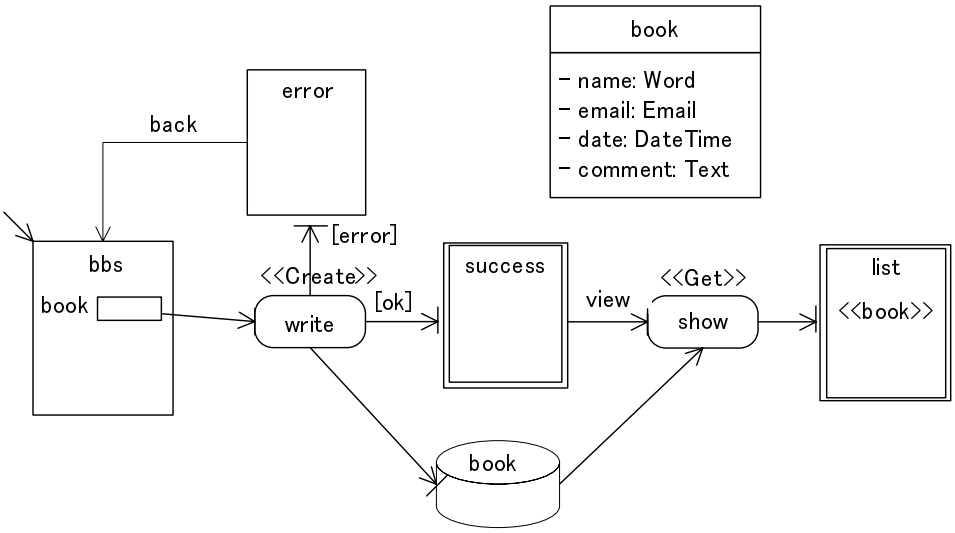

図 3.10 クライアント協調型 Web 遷移図の例

- 単一表示要素からノード <<Update>> へのリンクは，そのデータの与えられた値への更新を意味する．
- 単一表示要素からノード <<Delete>> へのリンクは，そのデータの削除を意味する．
- 複数表示要素からノード <<Get>> へのリンクは，指定したデータの詳細の取得を意味する．
- 複数表示要素からノード <<Delete>> へのリンクは，指定したデータの削除を意味する．
- ノード <<Get>> から単一表示要素の出力は，検索結果に応じたデータの表示を意味する．
- ノード <<Get>> から複数表示要素の出力は，検索結果に応じたデータ一覧の表示を意味する．
- ノード <<Create>> から単一表示要素の出力は，追加したデータの表示を意味する．
- ノード <<Update>> から単一表示要素の出力は，更新したデータの表示を意味する．

拡張した Web 遷移図を用いて記述したダイアグラムの例を図 3.10 に示す．このダイアグラムは，図 3.8 に示した掲示板アプリケーションの Web 遷移図を拡張したものである．複合データ型”book”のメタデータは R+W+U-S+である．処理プログラムノード”write”にステレオタイプ <<Create>> を，”show”にステレオタイプ <<Get>> を付与している．

3.3 Web 遷移図の構成方法

Web 遷移図は特定の開発プロセスを前提とするものではなく，ウォーターフォール型やスパイラル型などの任意の開発プロセスで使用することが可能である．しかし，データ処理

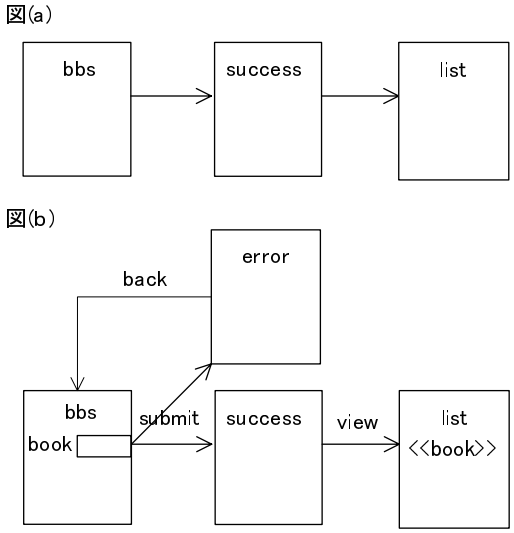

図 3.11　Web 遷移図の記述過程例

中心の Web アプリケーションを新規に作成する場合は，一般に以下のような手順で Web 遷移図を構成する．

1. Web アプリケーションの処理対象となるデータを定義する．典型的な Web アプリケーションで頻繁に使用されるデータ型定義は再利用することができる．仕様からの必要なデータ型の抽出には，オブジェクト指向分析におけるクラス抽出など一般的なソフトウェア分析・設計技術を利用することが可能である．

2. Web ページ間の遷移を設計する．仕様からの必要な Web ページの抽出は，ロバストネス分析による境界オブジェクト抽出など一般的なソフトウェア分析・設計技術を利用することが可能である．

3. Web ページにデータの入力手段，表示手段を割り当てる．

4. Web ページ間の遷移の途中でデータ処理を行う必要がある場合，その箇所に処理プログラムを割り当てる．

図 3.8 に示した Web 遷移図を記述する途中過程を図 3.11 に示す．図 3.11(a) は上記手順 2 において Web ページの遷移を設計した段階である．図 3.11(b) は上記手順 3 においてデータ入力手段と表示手段を割り当てた段階である．手順 3 でデータ入力手段を割り当てたことから，入力エラーを表示するための Web ページが必要であると判断し，Web ページノードとページ遷移リンクを追加している．図 3.11(b) に処理プログラムを挿入することで，図 3.8 が構成される．

3.4　セッション図

Web 遷移図ではセッションの開始ページと正常終了ページが明記されるため，意図した Web ページ以外からセッションが開始されることを防止するなど，生成システムが一定の

セッション管理を行うプログラムを自動生成することができる．しかし，利用者が Web ブラウザの保持する履歴情報を用いて任意の HTTP リクエストの列を送信した場合，Web アプリケーションが実現すべき振る舞いを Web 遷移図から一意に決定することは困難である．例えば，掲示板アプリケーションにおいて 1 つ前の履歴ページの Web フォームを使用した 2 度目の HTTP リクエストはデータの追加を意味すると考えられるが，アンケートフォームでは 2 度目の HTTP リクエストはデータの上書き（操作のやり直し）を意味すると考えられる．このように，Web ブラウザの履歴情報を用いた操作はアプリケーションによってその意味が異なるため，つねに履歴情報の使用を禁止するなど一律に振る舞いを決定する方法では適切なセッション管理を行えない．そこで，Web 遷移図の記述を補完し生成システムによるセッション管理機能の強化を図るべく，セッション内の振る舞いの意味を規定するセッション図を使用する．

ここで，セッション図を定義する前に，Web 遷移図におけるセッションの範囲を厳密に定義する．セッションは，Web ブラウザが送信する Web サーバに対する一連の HTTP リクエストで構成されるが，セッションの開始時点と終了時点の認識は Web ブラウザと Web サーバとで異なる．Web ブラウザが認識するセッションの開始は，セッションの開始ページに相当する Web ページ文書を Web サーバから取得した時点であり，セッションの終了は，セッションの終了ページに相当する Web ページ文書を Web サーバから取得した時点である．一方，Web サーバが認識するセッションの開始は，サーバ側プログラムがセッションの開始を許可した時点であり，セッションの終了は，サーバ側プログラムがセッションの終了を許可した時点である．そこで，Web 遷移図ではセッションの開始時点と終了時点を，処理プログラムノードから正常終了時に出力される Web ページノードへのデータフローリンクの遷移が発生した時点として表現することとする．

セッション図は，処理プログラムの可能な実行列を明示するためのダイアグラムであり，Web サーバの状態の遷移を規定した決定性有限状態機械として定義される．ここで，Web サーバの状態を更新可能な記憶領域がもつ状態集合の直積の部分集合と定義する．セッション図の定義は以下の通りである．

$$
\begin{aligned}
SD &:= (Q, T, \sigma, s, F) \\
Q &\subseteq D_1 \times D_2 \times ... \times D_n \\
T &:= \{t_1, t_2, ..., t_m\} \\
\sigma &: Q \times T \to Q \\
s &\in Q \\
F &\subseteq Q
\end{aligned}
$$

セッション図 SD は，Web サーバの状態の集合 Q，処理プログラムの集合 T，状態遷移関数 σ，開始状態 s，終了状態 F で構成される．集合 Q は，各記憶領域がもつ状態集合 D_k の直積の部分集合である．遷移関数 σ は，ある Web サーバの状態から処理プログラムの実行後どの状態に遷移するかを決定する関数である．セッション図を使用することにより，セッション内で許容される処理プログラムの実行列を明示的に定義することができる．

セッション図における状態遷移には，記憶領域が保持するデータの更新に関して 4 つのパターンが存在する（図 3.12）．データの更新では，セッション内で 1 回のみ更新を許可する振る舞い，1 回以上任意の回数の更新を許可する振る舞い，高々1 回の更新を許可する振る舞い，処理プログラム実行の成功・失敗のような条件分岐をともなう振る舞いのパターンが

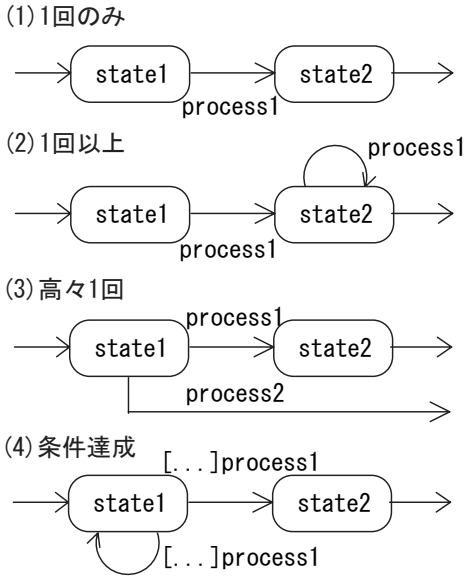

図 3.12 セッション図における状態遷移のパターン

ある．これらパターンを組み合わせることでセッション図を構成し，Web アプリケーションの振る舞いを規定する．

セッション図は，先に記述した Web 遷移図を参照して以下の手順で記述する．

1. Web 遷移図に含まれるセッションを抽出し，各セッションの範囲を明確にする．

2. セッション毎に記憶領域を，永続記憶領域と一時記憶領域に分類して抽出する．ここで，永続記憶領域とはセッションが終了しても保持するデータが消去されない記憶領域をいい，一時記憶領域とはセッションの終了によって保持するデータが消去される記憶領域をいう．また，セッション毎に処理プログラムを，記憶領域のデータを更新する処理プログラム，一時記憶領域からデータを読み込む処理プログラム，その他の処理プログラムに分類して抽出する．

3. 各記憶領域が取りうる状態の集合を定義する．セッション図における状態は，各セッションにおいてデータの更新が行われる記憶領域の状態集合の直積の部分集合として定義される．ここで部分集合としたのは，セッション中に存在し得ない状態を集合から除外する意味である．そして，セッション毎に初期状態および終了状態を定義する．

4. 記憶領域のデータを更新する処理プログラムに対し，その実行の意味を状態遷移のパターンから選択し，遷移関数を決定する．次に，一時記憶領域からデータを読み込む処理プログラムの実行に関する遷移関数を決定する．最後に，その他の処理プログラムの実行に関する遷移関数を決定する．

簡単なショッピングカートアプリケーションを記述した Web 遷移図とこれに基づくセッション図の例を図 3.13 に示す．この Web アプリケーションでは，Web ページ”top”からセッションが開始される．処理プログラム”show”がデータベーステーブル”goods”から商品データを取得し，処理プログラム”select”が買い物かごへの商品追加を行い，処理プログラム”pay”

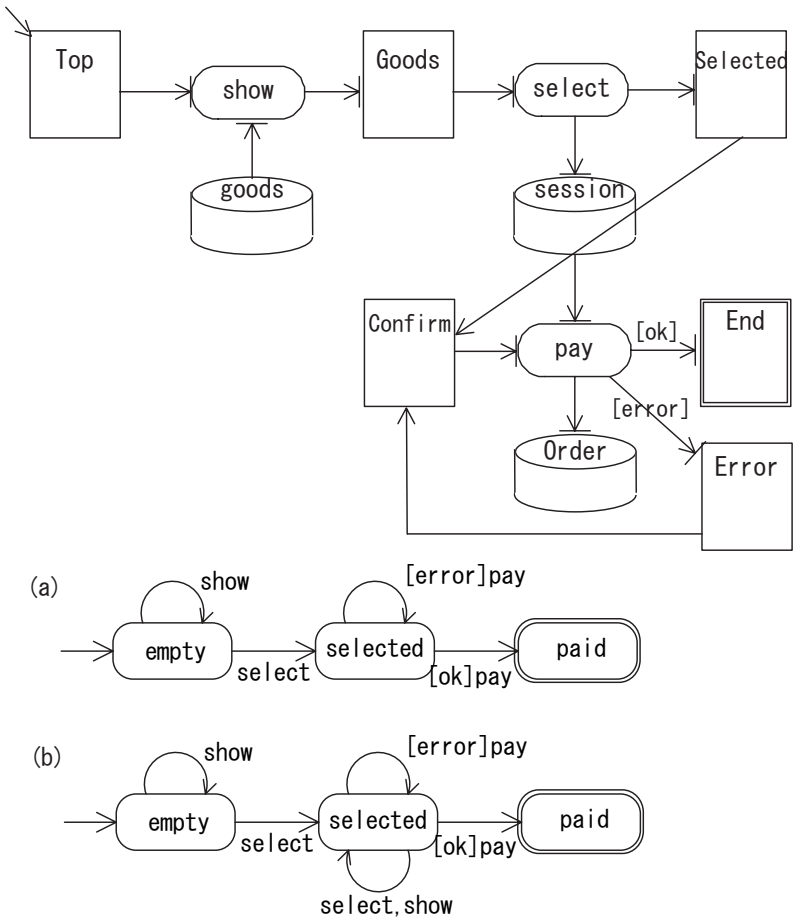

図 3.13 セッション図の記述例

が注文の確定を行う．Web ページ”End”が出力された時点でセッションが正常終了する．この Web 遷移図に含まれるセッションは 1 つであり，記憶領域”session”と”order”がセッション内で更新される記憶領域である．そこで，セッション図における状態の集合 Q は以下のように定義される．

$$
\begin{aligned}
D_{session} &= \{not\_selected, selected\} \\
D_{order} &= \{not\_ordered, ordered\} \\
Q &= \{empty, selected, paid\} \\
& \quad empty = (not\_selected, not\_ordered) \\
& \quad selected = (selected, not\_ordered) \\
& \quad paid = (selected, ordered)
\end{aligned}
$$

記憶領域”session”が取りうる状態は商品の”未選択”，”選択済”のいずれかであり，記憶領域”order”が取りうる状態は注文の”未注文”，”注文済”のいずれかである．状態集合 Q は状態集合 $D_{session}$ と D_{order} との直積となる．ただし，(”未選択”,”注文済”) という状態はセッション中に存在しえないため除去してある．この状態集合に基づいてさらに遷移関数を付与したものがセッション図となる．図 3.13 に示した 2 つのセッション図は，いずれも上記 Web 遷移図に基づいて記述したものである．セッション図 3.13(a) は商品の選択が 1 回のみ可能であるという振る舞いを規定している．つまり，1 つの商品ごとに注文の確定が必要な Web アプリケーションの仕様を表している．一方，セッション図 3.13(b) は商品の選択が 1 回以上可能であるという振る舞いを規定している．つまり，1 度で複数商品の注文が可能な Web アプリケーションの仕様を表している．

このように，同一の Web 遷移図に基づいて複数のセッション図を構成することができる．セッション図は，Web 遷移図で明示した遷移以外の処理プログラム実行列を許可する場合にセッションの振る舞いを定義するものであり，Web 遷移図の記述を補完し仕様を追加する役割を果たす．したがって，セッション図を使用することにより，より高度なセッション管理機能をもつ Web アプリケーションを自動生成することが可能となる．

第4章

サーバページ型Webアプリケーションの生成

本章では，Web 遷移図からサーバページ型 Web アプリケーションを自動生成する手法および生成システムについて述べる．

4.1　生成システムの概要

本研究で提案するサーバページ型 Web アプリケーションのための生成システム（以下，本章で T-Web システムという）は，第 3 章で述べたサーバページ型 Web アプリケーションを記述するための Web 遷移図と各ノードに関する付加的記述から，サーバページ型 Web アプリケーションを自動生成するシステムである．

Web 遷移図は，Web アプリケーションにおけるデータフローの本質的要素を簡潔に記述したものであり，Web アプリケーションの処理プログラムが行う計算の手順等を詳細に記述したものではない．そのため，Web 遷移図のみから，Web アプリケーションの雛形を自動生成するすることはできても，実行可能な Web アプリケーションを自動生成することはできない．一方，Web 遷移図に基づく自動生成システムを使用することにより，手作業でプログラムを記述する場合に問題となる構成要素間の一貫性欠如を防止することができる．そこで T-Web システムでは，Web 遷移図の各処理プログラムノードにあらかじめ用意された汎用的なプログラムモジュール（以下，処理プログラムテンプレートという）を対応付けることにより，実行可能な Web アプリケーションの自動生成を実現する．なお，従来のサーバプログラム型 Web アプリケーションのための T-Web システムも処理プログラムテンプレートを用いた自動生成を行うが，アーキテクテャの相違から処理プログラムテンプレートの内容および適用の方法が提案システムとは異なる．

T-Web システムの全体構造を図 4.1 に示す．T-Web システムは，Web 遷移図エディタと Web アプリケーションジェネレータの 2 つのサブシステムから構成される．一貫性を保持した Web アプリケーションを生成するためには，一貫性のある Web 遷移図の記述が入力として与えられる必要がある．そこで T-Web システムは，Web 遷移図の記述を支援するための Web 遷移図エディタをサブシステムとしてもつ．開発者は，Web 遷移図エディタの専用 GUI エディタを使用して Web 遷移図の記述，処理プログラムテンプレートの関連付けおよび各ノードの付加的記述を視覚的操作により行う．GUI エディタを用いて記述された Web

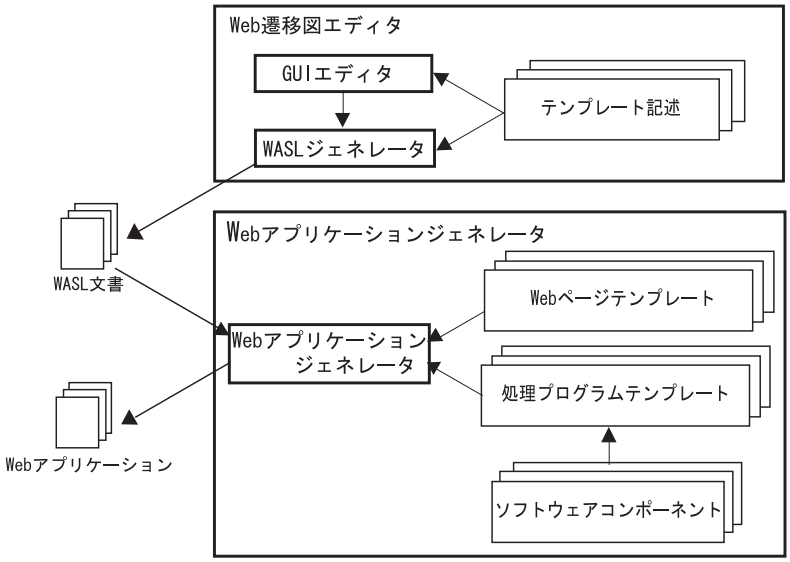

図 4.1 サーバページ型 T-Web システムの構造

遷移図および付加的記述は，Web 遷移図エディタがもつ WASL ジェネレータにより WASL（Web Application Specification Markup Language）文書に変換される．WASL とは，サーバページ型 Web アプリケーションの仕様を記述するために T-Web システムが使用する独自の中間言語である．サブシステム Web アプリケーションジェネレータは，WASL 文書を入力として実行可能なサーバページ型 Web アプリケーションを自動生成する．生成の過程で，Web アプリケーションジェネレータはシステム内にもつ処理プログラムテンプレートとフォーム要素等の Web ページテンプレートを適用する．

T-Web システムを使用することにより，開発者はプログラムや Web ページ文書を手作業で記述することなく Web アプリケーションを生成することができる．なお，Web アプリケーションジェネレータは WASL の仕様のみに依存することから，Web 遷移図エディタと Web アプリケーションジェネレータの実装は独立に拡張が可能である．

4.2 テンプレート

T-Web システムで採用するテンプレート方式による Web アプリケーションの生成とは，プログラムと Web ページに関する記述中の可変的要素をパラメータ化した汎用的なテンプレートを使用して，Web アプリケーションを自動生成する方法である．汎用的テンプレートの各パラメータに具体的な値を適用することにより，具体的計算手順を記述したコードおよび具体的な Web ページに関する記述を得る．T-Web システムは，Web アプリケーションジェネレータの一部として処理プログラムテンプレートと Web ページテンプレートをもつ．

4.2.1 処理プログラムテンプレート

T-Web システムがもつ処理プログラムテンプレートは，データベース内のデータの検索・更新，電子メールの送信等，特定のアプリケーション領域に依存しない汎用的処理を記述し

たプログラムである．そのため，開発者は典型的な Web アプリケーションの多くを，T-Web システムにあらかじめ組み込まれた処理プログラムテンプレートを使用して開発することができる．なお，データの比較・集計等のアプリケーション領域に依存した処理を必要とする場合，開発者は処理プログラムテンプレートを作成して T-Web システムに追加することも可能である．T-Web システムの実装がもつ処理プログラムテンプレートは以下の 13 種類である．

ADD データベーステーブルに新規レコードを追加するテンプレート．ただし各カラムについて既存レコードと重複する値がある場合，レコードの追加を認めない．

ADD データベーステーブルに新規レコードを追加するテンプレート．ただしレコード全体として既存レコードと重複がある場合，レコードの追加を認めない．

ADD データベーステーブルに新規レコードを追加するテンプレート．ただし指定カラムの値が重複した既存レコードが存在する場合，既存レコードに新規レコードを上書きする．

ADD データベーステーブルに新規レコードを追加するテンプレート．指定カラムの値が重複した既存レコードが存在する場合，既存レコードに新規レコードを上書きし，かつセッションオブジェクトにデータを追加する．

SHOW データベーステーブルからすべてのレコードを読み込み表示させるテンプレート．

SHOW データベーステーブルから条件に合うレコードを読み込み表示させるテンプレート．

SHOW セッションオブジェクトからすべてのデータを読み込み表示させるテンプレート．

MATCH データベーステーブル中に条件に合うレコードが存在するか調べるテンプレート．

MATCH データベーステーブル中に条件に合うレコードが存在するか調べるテンプレート．存在する場合は他のデータベーステーブルから条件に合うレコードを読み込み表示させる．

UPDATE データベーステーブル中の条件に合うレコードの指定カラム値を更新するテンプレート．

CANCEL データベーステーブル中の条件に合うレコードの指定カラムを空にするテンプレート．

DELETE データベーステーブル中の条件に合うレコードを削除するテンプレート．

SENDMAIL 電子メールを送信するテンプレート．

すべての処理プログラムテンプレートは，入力値検査，利用者認証およびデータベース更新時の整合性検査のコードをもつ．データベースを更新する処理プログラムテンプレートが規定する計算の流れを図 4.2 に示す．

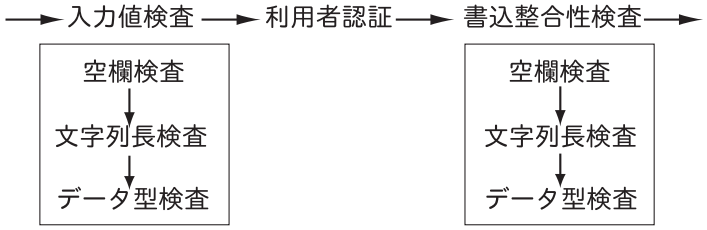

図 4.2 処理プログラムテンプレートが規定する計算の流れ

1. 処理プログラムはまず，Web フォームの入力値に対して空欄検査，文字列長検査およびデータ型検査を行う．入力値の空欄検査では，空文字列の入力が許容されていない入力欄に空文字列が値として与えられた場合にエラー出力を行う．入力値の文字列長検査では，指定された最大文字列長以上の文字列が入力欄に値として与えられた場合にエラー出力を行う．入力値のデータ型検査では，指定された型に変換不可能な文字列が入力欄に値として与えられた場合にエラー出力を行う．データ型検査では正規表現を用いてデータ型の判断を行う．

2. 入力値検査に成功した場合，次に利用者認証を行う．利用者認証とは，利用者が開発者の意図した順序に従って Web ページを遷移しているか検査することである．利用者が URL を直接指定してセッション開始ページを経由せずに Web ページを閲覧しようとする場合など，開発者の意図しない HTTP リクエストに対してエラー出力を行う．利用者認証は Web ブラウザが保持するクッキーの値を用いて実現する．

3. 上記検査に成功した場合のみ，処理プログラムは本来の計算を実行する．処理プログラムがデータベースの更新を行う場合，更新前に整合性検査を行う．データベース更新時の整合性検査は，空欄検査，文字列長検査およびデータ型検査から構成される．データベースの空欄検査では，データベーステーブルの定義で空文字列が許容されていないカラムに対して空文字列を書き込もうとする場合にエラー出力を行う．データベースの文字列長検査では，データベーステーブルの定義で規定された最大文字列長以上の値を書き込もうとする場合にエラー出力を行う．データベースのデータ型検査では，データベーステーブルで定義されたデータ型と異なる型の値を書き込もうとする場合にエラー出力を行う．

上記のすべての検査に成功した場合のみ処理プログラムの実行が正常終了し，少なくとも 1 つのエラーが出力された場合は処理プログラム全体として実行エラーとなる．上記検査は Web アプリケーションの安全性やデータの一貫性を維持する上で必要不可欠であるが，プログラムを記述するためのコードが煩雑で膨大になりやすく手作業による開発では記述漏れが生じやすい．一方，T-Web システムの処理プログラムテンプレートは上記コードを標準的にもつ．

T-Web システムがもつ処理プログラムテンプレートの記述例を図 4.3 に示す．処理プログラムテンプレートでは，各パラメータ名を”/*” と”*/” の間に記述する．また，1 パラメータが配列として複数の値をもつ場合，”/**” と”**/” の間に各パラメータ名を記述をする．

```
<%@ Language=VBScript %>
<% Response.ContentType="/*CONTENTTYPE*/" %>
<%
    ’入力パラメータの処理
    /**INPUTPARAM
    Dim input_/*NAME*/
    input_/*NAME*/ = Request.Form("/*NAME*/")
    **/
    ’データベースへの接続
    Set _conn = Server.CreateObject("ADODB.Connection")
    Set _rs = Server.CreateObject("ADODB.Recordset")
    _conn.open "/*DBNAME*/", "/*DBUSR*/", "/*DBPASSWD*/"
    ------- 省略 -------
    Set _rs = conn.Execute(query)
%>
/***PAGETEMPLATES***/
<%
    ’データベース接続の切断
    _rs.Close
    _conn.Close
%>
```

図 4.3 処理プログラムテンプレートの例

4.2.2 Web ページテンプレート

T-Web システムがもつ Web ページテンプレートは，第 3 章で示した Web 遷移図の各 Web ページ要素に対応した，Web ページ文書を構成する Web ページ記述の断片である．T-Web システムの実装がもつ Web ページテンプレートは以下の 13 種類である．

ROOTDOC Web ページ記述のルート要素を記述したテンプレート．Web ページ名および HTML や XHTML などの文書の記述言語を定義する．

DBRECORD データベースレコードの一覧を表示するためのテンプレート．

SESSIONOBJ セッションオブジェクトが保持するデータの一覧を表示するためのテンプレート．

VISIBLEPARAM 可視パラメータを表示するためのテンプレート．

HYPERLINK ハイパーリンクを表示するためのテンプレート．0 個以上の引数をもつことができる．

FORMDOC Web フォームのルート要素を記述したテンプレート．Web フォームの送信先を定義する．

TEXTINPUT Web フォームのテキスト入力欄を表示するテンプレート．

PASSWDINPUT Web フォームのパスワード入力欄を表示するテンプレート．

TEXTAREA Web フォームのテキスト入力領域を表示するテンプレート．

CHECKBOX Web フォームのチェックボックスを表示するテンプレート．

RADIOBUTTON Web フォームのラジオボタンを表示するテンプレート．

SELECTLIST Web フォームの選択リストを表示するテンプレート．

HIDDENPARAM Web フォームの隠しコントロールを定義するテンプレート．

T-Web システムがもつ Web ページテンプレートの記述例を図 4.4 に示す．Web ページテンプレートでは，処理プログラムテンプレートと同様に，各パラメータ名を”/*” と”*/” の間に記述し，1 パラメータが配列として複数の値をもつ場合は”/**” と”**/” の間に各パラメータ名を記述する．

4.3 Web 遷移図エディタ

Web 遷移図エディタは T-Web システムのサブシステムであり，GUI エディタおよび WASL ジェネレータから構成される．

```
<table>
<% Do While not twb_rs.EOF %>
    <tr>
        /**DBRECORD
        <td><%= _rs.Fields("/*COLUMN*/").Value %></td>
        **/
    </tr>
<%
    twb_rs.MoveNext
    Loop
%>
</table>
```

図 4.4 Web ページテンプレートの例

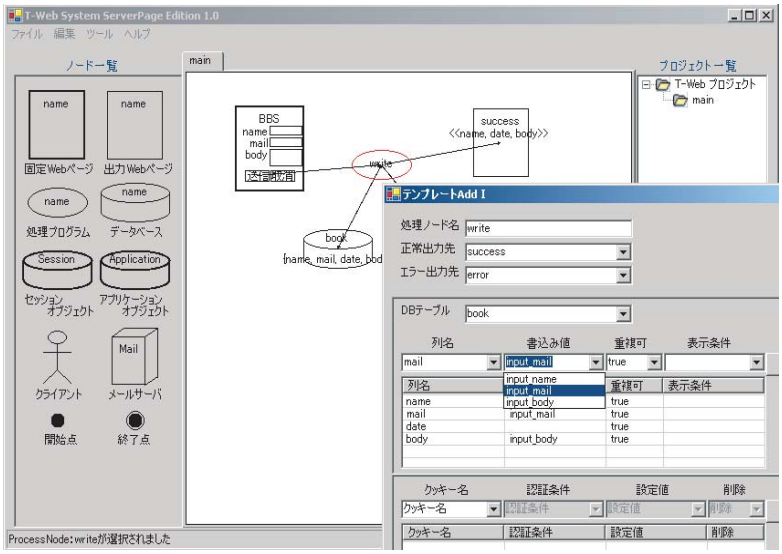

図 4.5 サーバページ型 T-Web システムの GUI エディタ

4.3.1 GUI エディタ

GUI エディタは一貫性を有する Web 遷移図の記述を支援するための専用エディタであり，すべての操作を視覚的に行えるように設計されている．GUI エディタのスナップショットを図 4.5 に示す．GUI エディタを利用して，一般に以下の手順で Web 遷移図を記述する．

1. Web ページノードや処理プログラムノード等，Web アプリケーションの振る舞いを記述するために必要なノードを描画領域上に配置する．開発者はマウスポインタ等を用いて，ノード一覧から適切なノードを選択し配置する．

2. 次に，記憶領域に関する各ノードのパラメータ値を設定する．パラメータ値は，ノードの種類に応じて用意された専用のウインドウ（以下，プロパティウィンドウという）を用いて入力する．プロパティウィンドウは，任意の値を入力するテキスト入力領域と一覧から値を選択する選択リストをもつ．選択リストから値を選択できることで，一貫性を保持した Web 遷移図の記述が容易となる．プロパティウィンドウを用いて，メールサーバノードを除くストレージに関するノードに対してデータ要素名やデータ型などを設定する．また，メールサーバノードに対して SMTP サーバ名などを設定する．

3. 各 Web ページノードのパラメータ値を設定する．プロパティウィンドウを用いて Web ページ名や Web ページ要素を設定する．

4. 各処理プログラムノードに対して処理プログラムテンプレートを割り当てる．開発者は T-Web システムがもつ処理プログラムテンプレートの一覧から適切なものを 1 つ選択する．そして，選択した処理プログラムテンプレートで設定すべきパラメータ値をプロパティウィンドウを用いて設定する．パラメータ値として，データベースの操作などに関する項目等がある．

5. Web ページ間のページ遷移リンクを記述する．Web ページノードのプロパティウィンドウを用いて，遷移可能なリンク先一覧から遷移先を選択する．

6. 最後に，Web 遷移図全体に関する大域パラメータ値を設定する．アプリケーション名，データベースとの接続に関する項目，意図しないエラーが発生した場合の遷移先 Web ページを設定する．

4.3.2 中間言語 WASL

WASL ジェネレータは，Web 遷移図と付加的記述を入力として WASL 文書を出力する．ここで，中間言語 WASL は，サーバページ型 Web アプリケーションの仕様を記述するための XML に基づく言語である．図 4.6 に WASL のスキーマ定義を表す DTD（Document Type Definition）記述の一部を示す．WASL はサーバページ型 Web アプリケーションの振る舞いの本質を記述するための言語であり，ASP や JSP 等の特定の実装アーキテクテャに依存しない．そのため，同一の WASL 文書から複数のサーバページ型アーキテクチャに基づく Web アプリケーションを生成することができる．WASL 文書は，サーバページ型 Web アプリケーションの構成要素に関する記述の集合として構成される．”page”タグは，Web アプリケーションの Web ページテンプレートを意味する．”process”タグは，Web アプリケー

ションの処理プログラムを意味する．”dbtable”タグは，Web アプリケーションで使用するデータベースのテーブル定義を意味する．”sessionobj”タグは，Web アプリケーションで使用するセッションオブジェクトを意味する．

4.4 Web アプリケーションジェネレータ

Web アプリケーションジェネレータは，WASL 文書を入力として実行可能なサーバページ型 Web アプリケーションの Web ページ文書を出力する．Web 遷移図と出力される Web ページ文書との対応関係は以下の通りである（図 4.7）．

静的 Web ページ文書 固定 Web ページノード毎に，固定 Web ページノードの付加的記述を Web ページテンプレートに適用して静的 Web ページ文書を出力する．Web ページテンプレートの適用は，”/*”と”*/”で囲まれた部分を対応するパラメータ値に置換し，”/**”と”**/”で囲まれた部分を対応するパラメータ値の個数だけ繰り返した文字列に置換することで行う．ここで，ハイパーリンクや Web フォームの遷移先は，後述の処理プログラムを有する動的 Web ページ文書に設定する．文字列の置換を階層的に行うことで，木構造をもつ Web ページ文書を出力する．

処理プログラムを有する動的 Web ページ文書 処理プログラムノード毎に，処理プログラムノードの付加的記述および参照しているデータベースノード等の付加的記述を処理プログラムテンプレートに適用し，さらに処理プログラムの正常遷移先として定義された Web ページノードの付加的記述を Web ページテンプレートに適用して動的 Web ページ文書として出力する．つまり，処理プログラムのコードと正常終了時に出力される Web ページ記述が 1 つの動的 Web ページ文書として出力される．ここで，処理プログラムの実行中にエラーが発生した場合，後述のエラー表示のための動的 Web ページ文書に遷移するようパラメータ値を設定する．処理プログラムテンプレートの適用は，Web ページテンプレートと同様の方法で行う．

エラー表示のための動的 Web ページ文書 処理プログラムのエラー発生時の遷移先として定義された出力 Web ページノード毎に，出力 Web ページノードの付加的記述を Web ページテンプレートに適用して動的 Web ページ文書を出力する．

T-Web システムでは，適用する処理プログラムテンプレートを切り替えることで，異なるサーバページ型アーキテクチャの Web アプリケーションを生成することが可能である．また，適用する Web ページテンプレートを切り替えることで，異なるマークアップ言語を用いた Web ページ文書を生成することが可能である．T-Web システムの現在の実装は，処理プログラムテンプレートとして，ASP アプリケーションと JSP アプリケーション用のテンプレートをもつ．また，Web ページテンプレートとして，HTML を用いた Web ページ文書と XML を用いた Web ページ文書用のテンプレートをもつ．なお，T-Web システムが生成する文書の種類は，生成対象とするアーキテクテャと使用するマークアップ言語により異なる（表 4.1）．

HTML 形式による生成 HTML 用テンプレートを用いた生成では，以下の 3 種類の文書が出力される．

```
<!ELEMENT web(application,page*,process*,dbtable*,sessionobj?)>
<!ELEMENT application(name,database,illegalpage,smtp?)>
<!ELEMENT database(name,user,passwd)>
<!ELEMENT illegalpage(name)>
<!ELEMENT smtp(server,sender)>

<!ELEMENT page(type,title,dbrecord?,visibleparam?,hyperlink*,formdoc*)>
<!ELEMENT dbrecord(column+)>
<!ELEMENT visibleparam(name+)>
<!ELEMENT hyperlink(name,target,param*)>
<!ELEMENT formdoc(targer,(checkbox|hiddenparam|passwdinput|radiobutton
                                    |selectlist|textarea|textinput)+)>
<!ELEMENT param(name,value)>
<!ELEMENT checkbox(name,value+)>
<!ELEMENT hiddenparam(name,value)>
<!ELEMENT passwdinput(name,size)>
<!ELEMENT radiobutton(name,value+)>
<!ELEMENT selectlist(name,value+)>
<!ELEMENT textarea(name)>
<!ELEMENT textinput(name,size)>

<!ELEMENT process(name,template,normalpage,errorpage,inputtype,
                  inputparam?,getcookie?,authcookie?,setcookie?,
                  delcookie?,dbcolumns?,subdbcolumns?,getrecordcond?,
                  writerecord?,cancelcond?,cancelcolumn?,deletecond?,
                  matchcond?,updatecond?,updaterecord?,writesession?,
                  sendmail?)>
------- 以下省略 -------
```

図 4.6 WASL のスキーマ定義

	HTML	XML
Web ページ文書	静的 Web ページ文書 (*.html)	静的 Web ページ文書 (*.xml)
	動的 Web ページ文書	動的 Web ページ文書
	(*.asp,*.jsp など)	(*.asp,*.jsp など)
スタイル定義文書	CSS 文書 (*.css)	CSS 文書 (*.css)
		XSL ファイル (*.xsl)
スキーマ定義文書		DTD 文書 (*.dtd)

表 4.1 T-Web システムが生成する文書の種類

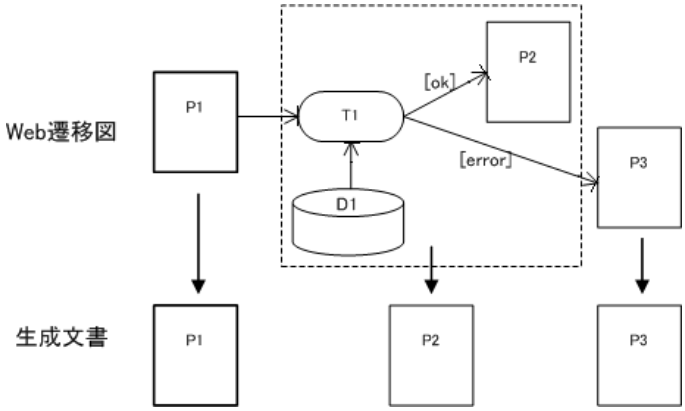

図 4.7 サーバページ型 Web アプリケーションの生成規則

- **静的 Web ページ文書** 拡張子が".html" の HTML 文書.
- **動的 Web ページ文書** 拡張子が".asp" や".jsp" などの動的 Web ページ文書で, HTML で記述されたもの.
- **スタイル定義文書** アプリケーション毎に 1 つの CSS（Cascading Style Sheet）文書．生成後に Web ページの色彩やフォント等の表示方法を変更するために使用する [11].

XML 形式による生成 XML 用テンプレートを用いた生成では，以下の 5 種類の文書が出力される.

- **静的 Web ページ文書** 拡張子が".xml" の XML 文書.
- **動的 Web ページ文書** 拡張子が".asp" や".jsp" などの動的 Web ページ文書で, XML で記述されたもの.
- **スタイル定義文書** アプリケーション毎に 1 つの CSS 文書．また，各 Web ページ文書ごとに 1 つの XSL（eXtensible Style Language）文書．生成後に Web ページのレイアウト等の表示方法を変更するために使用する [12].
- **スキーマ定義文書** アプリケーション毎に 1 つの DTD 文書．Web ページ文書で使用する XML のスキーマを定義するものである.

なお，T-Web システムが生成する Web ページ文書で用いる XML は，図 4.8 に示すスキーマ定義に従う.

表 4.2 は，第 3 章で示した Web 遷移図の記述例を T-Web システムの入力として与えた場合に出力される文書を示している．ここでは，HTML 形式と XML 形式により，それぞれ ASP アプリケーションと JSP アプリケーションを生成した場合について示している．また，生成された動的 Web ページ文書"success.asp" の記述を図 4.9 示す．生成した Web アプリケーションを実行したときの Web ページの遷移は図 4.10 に示す通りである.

```
<!ELEMENT page(type,title,message*,(dbrecord|sessionobj|visibleparam
                                          |hyperlink|formdoc)*)>
<!ELEMENT dbrecord(record*)>
<!ELEMENT record(value+)>
<!ELEMENT sessionobj(record*)>
<!ELEMENT visibleparam(name)>
<!ELEMENT hyperlink(name,target,param?)>
<!ELEMENT formdoc(target,(checkbox|hiddenparam|passwdinput|radiobutton
                                |selectilist|textarea|textinput)+)>
<!ELEMENT checkbox(name,value+)>
<!ELEMENT hiddenparam(name,value)>
<!ELEMENT passwdinput(name,size)>
<!ELEMENT radiobutton(name,value+)>
<!ELEMENT selectlist(name,value+)>
<!ELEMENT textarea(name)>
<!ELEMENT textinput(name,size)>

<!ELEMENT type(#PCDATA)>
<!ELEMENT title(#PCDATA)>
<!ELEMENT message(#PCDATA)>
<!ELEMENT name(#PCDATA)>
<!ELEMENT target(#PCDATA)>
<!ELEMENT param(#PCDATA)>
<!ELEMENT value(#PCDATA)>
<!ELEMENT size(#PCDATA)>
```

図 4.8 T-Web システムが生成する XML 文書のスキーマ定義

```
<%@ Language=VBScript %>
<%  Response.ContentType="text/html" %>
<%
    ’入力パラメータの処理
    Dim input_name
    input_name = Request.Form("name")
    Dim input_email
    input_email = Request.Form("email")
    Dim input_comment
    input_comment = Request.Form("comment")
    ’データベースへの接続
    Set _conn = Server.CreateObject("ADODB.Connection")
    Set _rs = Server.CreateObject("ADODB.Recordset")
    _conn.open "book", "", ""
    ------- 省略 -------
    Set _rs = conn.Execute(query)
%>
<html>
<head><title>success</title></head>
<body>
<a href="list.asp">view</a><br/>
</body>
</html>
<%
    ’データベース接続の切断
    twb_rs.Close
    conn.Close
%>
```

図 4.9 T-Web システムが生成する動的 Web ページ文書の例

	ASP 文書 (HTML 形式)	JSP 文書 (HTML 形式)
Web ページ文書	bbs.html, success.asp, error.asp, list.asp, SystemError.html	bbs.html, success.asp, error.jsp, list.jsp, SystemError.html
スタイル定義文書	bbs.css	bbs.css
	ASP 文書 (XML 形式)	**JSP 文書 (XML 形式)**
Web ページ文書	bbs.xml, success.asp, error.asp, list.asp, SystemError.xml	bbs.xml, success.asp, error.jsp, list.jsp, SystemError.xml
スタイル定義文書	bbs.css bbs.xsl, success.xsl, error.xsl, list.xsl, SystemError.xsl	bbs.css bbs.xsl, success.xsl, error.xsl, list.xsl, SystemError.xsl
スキーマ定義文書	bbs.dtd	bbs.dtd

表 4.2 T-Web システムが生成する文書例

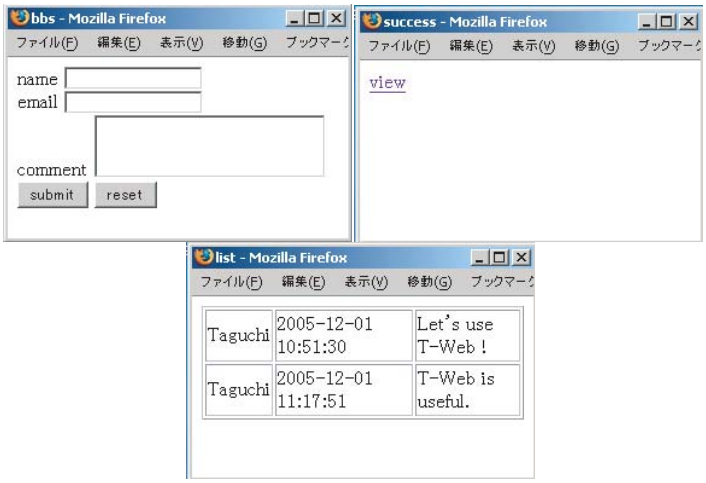

図 4.10 T-Web システムが生成する Web アプリケーションの実行例

第5章

サーバ集中型Webアプリケーションの生成

本章では，一般化した Web 遷移図からサーバ集中型の各アーキテクチャに基づく Web アプリケーションを自動生成する手法および生成システムについて述べる．

5.1　生成システムの概要

本研究で提案するサーバ集中型 Web アプリケーションのための生成システム（以下，本章で T-Web システムという）は，第 3 章で述べたサーバ集中型 Web アプリケーションを記述するための Web 遷移図（以下，本章で Web 遷移図という）と各ノードに関する付加的記述から，サーバ集中型の各アーキテクチャに基づく Web アプリケーションを自動生成するシステムである．

従来のサーバプログラム型 Web アプリケーションを生成する T-Web システムや第 4 章で述べたサーバページ型 Web アプリケーションを生成する T-Web システムは，それぞれのアーキテクチャに基づく Web アプリケーションの自動生成の柔軟性と容易性とを比較考量し，典型的な Web アプリケーションをより少ない労力で作成することを可能にする．しかし，上記生成システムではさまざまな実装アーキテクチャを考慮した体系的な生成手法が提供されていないため，他の実装アーキテクチャへの生成システムの応用や，複数の実装アーキテクチャに基づく Web アプリケーションの一括生成が困難であった．そこで，本章で述べる生成システムは，これまでのサーバプログラム型およびサーバページ型 Web アプリケーションの生成手法を踏まえて，実装アーキテクチャに依存しない体系的な生成手法を提供する．

サーバ集中型 T-Web システムでは，サーバページ型 T-Web システムと同様，あらかじめ用意された処理プログラムテンプレートを Web 遷移図の各処理プログラムノードに対応付けることで，実行可能な Web アプリケーションを自動生成する．生成される処理プログラムは，標準的なセッション管理機能とセキュリティ保持機能を備える．ただし，複数の実装アーキテクチャに基づく Web アプリケーションの生成で必要なテンプレートの一部を共通化するため，従来 1 つの処理プログラムテンプレートとして実現していたものを一定の機能ごとに分割してある．

T-Web システムの全体構造を図 5.1 に示す．T-Web システムは，Web 遷移図エディタと

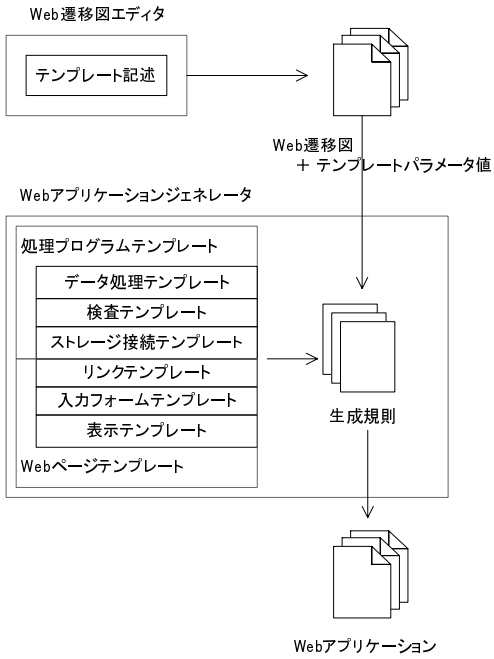

図 5.1　サーバ集中型 T-Web システムの構造

Web アプリケーションジェネレータの 2 つのサブシステムから構成される．Web 遷移図エディタは，一貫性を有する Web 遷移図の記述を支援するための専用 GUI エディタである．開発者は，Web 遷移図エディタを用いることで複合データ型の定義，ダイアグラムの記述，処理プログラムテンプレートの関連付け，各ノードの付加的記述を，すべて視覚的操作により行うことができる．図 5.2 は Web 遷移図エディタの実装である．サブシステム Web アプリケーションジェネレータは，上記記述と実装アーキテクチャを指定するパラメータ値を入力として，実行可能な Web アプリケーションを生成する．ここで，Web アプリケーションジェネレータは，指定された実装アーキテクチャに応じて生成規則を切り替え適切なテンプレートを適用する．

サーバ集中型 T-Web システムを使用することにより，サーバページ型 T-Web システムと同様，プログラムや Web ページ文書を手作業で記述することなく Web アプリケーションを生成することができる．また，生成規則が用意されている実装アーキテクチャについては，パラメータによる指定を変更することで，生成対象を切り替えることができる．生成規則が用意されていない実装アーキテクチャについては，生成規則を記述し必要に応じて一部テンプレートを作成することで，生成システムの拡張が可能である．

5.2　テンプレート

T-Web システムは，プログラムと Web ページに関する記述中の可変的要素をパラメータ化した汎用的なテンプレートを使用する．汎用的テンプレートの各パラメータに具体的な値を適用することにより，具体的計算手順を記述したコードおよび具体的な Web ページに関する記述を得る．T-Web システムは，Web アプリケーションジェネレータの一部として処理プログラムテンプレートおよび Web ページテンプレートをもつ．処理プログラムテンプ

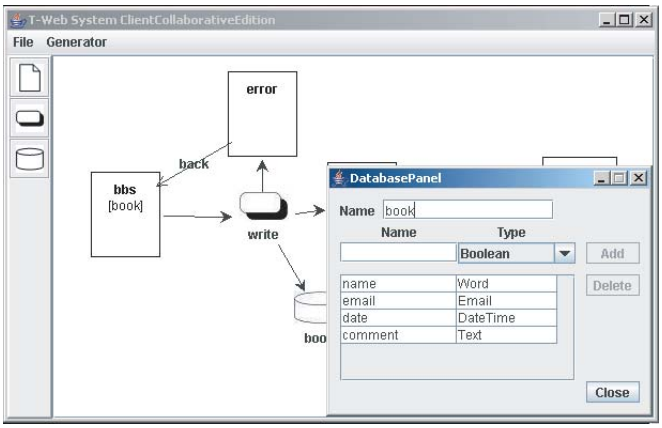

図 5.2 サーバ集中型 T-Web システムの Web 遷移図エディタ

レートはさらに，データ処理テンプレート，検査テンプレート，ストレージ接続テンプレートに分類される．Web ページテンプレートはさらに，リンクテンプレート，入力フォームテンプレート，表示テンプレートに分類される．

5.2.1 処理プログラムテンプレート

T-Web システムがもつ処理プログラムテンプレートの機能は，以下の通りである．

データ処理テンプレート 記憶領域が保持するデータの検索・更新，電子メールの送信等，特定のアプリケーション領域に依存しない汎用的なデータ処理を記述したプログラムモジュールである．開発者は，T-Web システムがもつデータ処理テンプレートを使用することで，典型的な Web アプリケーションのデータ処理を実現することができる．なお，データ処理テンプレートは記憶領域が保持するデータの操作に関するロジックのみを記述し，入力値検査やデータベース接続等の記述は後述のテンプレートに分離されている．

検査テンプレート 入力値検査およびセッション管理の処理を記述したプログラムモジュールである．入力値検査として，正規表現による文字列パターン検査，制御文字検査，文字列最大長検査，空文字列検査を行うテンプレートがある．セッション管理として，セッションの開始・終了処理，セッションの有効性判断，ユーザ認証の有効性判断を行うテンプレートがある．また，HTML のエスケープ文字や SQL の特殊文字などシステムにおいて特別な意味を有する文字列を無害化するためのテンプレートも含まれる．

ストレージ接続テンプレート 記憶領域の種類ごとに，記憶領域との接続処理を記述したプログラムモジュールである．記憶領域として，関係データベース，XML データベース，セッションオブジェクト，クッキー，Web サービスなどが考えられる．データベースの場合は接続ドライバ，セッションオブジェクトの場合はオブジェクト参照，クッキーの場合は HTTP リクエスト/レスポンス，Web サービスの場合はクライアントスタブを通じて，データの取得・更新を可能とする．

開発者は，必要に応じて処理プログラムテンプレートを作成し T-Web システムに追加することが可能である．例えば，特定のアプリケーション領域で用いられる統計処理のために，データ処理テンプレートを追加することができる．Web アプリケーションに対する新たな攻撃手法が発見されたときは，それを防御する検査テンプレートを追加することができる．記憶領域の新たな実装技術に対しては，処理プログラムからその記憶領域を使用するためのストレージ接続テンプレートを追加することができる．

文字列パターン検査のテンプレート記述例を以下に示す．このプログラムモジュールは，基本データ型で指定された正規表現に合致しない文字列が入力された場合にエラー出力を行う．

```
_typecheck_result =
    _typecheck_result & /*VALUE*/.matches("/*TYPE*/");
if(!_typecheck_result) {
    /*RESPONSE*/; return;
}
```

制御文字検査のテンプレート記述例を以下に示す．このプログラムモジュールは，改行文字（LF=x0A および CR=x0D）以外の制御文字を含む文字列が入力された場合にエラー出力を行う．

```
_controlcheck_result =
    _controlcheck_result & /*VALUE*/
        .matches("^[^\\x00-\\x09\\x0B\\x0C\\x0E-\\x1F\\x7F]*$");
if(!_controlcheck_result) {
    /*RESPONSE*/; return;
}
```

文字列最大長検査のテンプレート記述例を以下に示す．このプログラムモジュールは，基本データ型で指定された最大文字列長を超える文字列が入力された場合にエラー出力を行う．

```
sizecheck_result =
    _sizecheck_result & /*VALUE*/.length() <= /*MAXLENGTH*/;
if(!_sizecheck_result) {
  /*RESPONSE*/; return;
}
```

空文字検査のテンプレート記述例を以下に示す．このプログラムモジュールは，空文字列が入力された場合にエラー出力を行う．

```
_blankcheck_result =
    _blankcheck_result & /*VALUE*/.length() != 0;
if(!_blankcheck_result) {
    /*RESPONSE*/; return;
}
```

文字列無害化のテンプレート記述例を以下に示す．このプログラムモジュールは，エスケープ文字を同等の意味をもつ別の文字列に置換する．具体的には，&を& に，”を" に，’を' に，<を< に，>を> に，\ を \\ にそれぞれ置換する．

```
/*VALUE*/ = /*VALUE*/.replaceAll("&", "&amp;")
    .replaceAll("\"", "&quot;")
    .replaceAll("'", "&#39;")
    .replaceAll("<", "&lt;")
    .replaceAll(">", "&gt;")
    .replaceAll("\\", "\\\\");
```

関係データベース接続のテンプレート記述例を以下に示す．このプログラムモジュールは，指定されたデータベースに指定されたユーザ ID とパスワードで接続を試み，接続に失敗した場合にエラー出力を行う．

```
java.sql.Connection _db_connection = null;
java.sql.Statement _db_statement = null;
java.sql.ResultSet _db_result = null;
try {
    Class.forName("/*DRIVER*/");
    _db_connection = java.sql.DriverManager.getConnection(
                "jdbc:odbc:/*DATABASE*/","/*USER*/","/*PASSWORD*/");
    _db_statement = _db_connection.createStatement(
                java.sql.ResultSet.TYPE_SCROLL_SENSITIVE,
                java.sql.ResultSet.CONCUR_UPDATABLE);
} catch(Exception e) {
    /*RESPONSE*/; return;
}
```

5.2.2 Web ページテンプレート

T-Web システムがもつ Web ページテンプレートの機能は，以下の通りである．

リンクテンプレート Web ページのハイパーリンクを表現するための Web ページ記述である．他の Web ページ文書への静的参照のほか，パラメータ値を有するサーバ側プログラムへの参照も含む．

入力フォームテンプレート Web フォームの入力手段を表現するための Web ページ記述である．Web フォームに関する HTML タグに対応するテンプレートが用意されている．具体的には以下のテンプレートがある．

- **FormDocument** Web フォームのルート要素を記述したテンプレート．
- **TextInput** テキスト入力欄を表示するテンプレート．
- **PasswordInput** パスワード入力欄を表示するテンプレート．
- **Textarea** テキスト入力領域を表示するテンプレート．
- **CheckBox** チェックボックスを表示するテンプレート．
- **Radiobutton** ラジオボタンを表示するテンプレート．
- **SelectList** 選択リストを表示するテンプレート．

- **HiddenParameter** 隠しコントロール値を埋め込むテンプレート.
- **SubmitButton** 送信ボタンを表示するテンプレート.
- **ResetButton** リセットボタンを表示するテンプレート.

表示テンプレート Web ページの動的に値が埋め込まれる領域を表現するための Web ページ記述である．テーブル構造を表現するテンプレートとして，TableDocument，TableRow，TableHeader，TableData がある．表示するデータを設定するテンプレートとして，SingleDataView と MultiDataView がある．

開発者は，必要に応じて Web ページテンプレートを作成し T-Web システムに追加することが可能である．例えば，新たな Web ページ記述言語を用いた Web ページ文書を出力する場合，リンクテンプレート，入力フォームテンプレート，表示テンプレートのテーブル構造の関するものを追加する．表示テンプレートのデータ設定に関するものは，Web ページ記述言語に依存しないが，実装アーキテクチャに依存する．

リンクテンプレートの記述例を以下に示す.

```
<a href="/*TARGET*/?/*PARAMETERS*/">/*LABEL*/</a>
```

入力フォームテンプレート TextInput の記述例を以下に示す．このテンプレートは，入力可能な最大文字列長を規定するものである．

```
<input type="text" name="/*NAME*/"
       maxlength="/*MAXLENGTH*/" value="/*VALUE*/"/>
```

表示フォームテンプレート TableDocument の記述例を以下に示す.

```
<table name="/*NAME*/">
    <caption>/*LABEL*/</caption>
    /*ROWS*/
</table>
```

表示フォームテンプレート MultiDataView の記述例を以下に示す．このテンプレートは，テーブル 1 行ごとにデータを取得し列値としてデータを埋め込むものである．

```
while(_db_result.next()) {
    /** /*VARIABLE*/ =  _db_result.getString("/*VALUE*/"); **/
    /*ROW*/
}
```

5.3 生成規則

Web アプリケーションジェネレータは，Web 遷移図等の記述を処理プログラムテンプレートと Web ページテンプレートに適用することで，実行可能な Web アプリケーションを生成する．具体的には，テンプレート中の”/*”と”*/”で囲まれた部分を対応するパラメータ値に置換し，”/**”と”**/”で囲まれた部分を対応するパラメータ値の個数だけ繰り返した文字列に置換する．

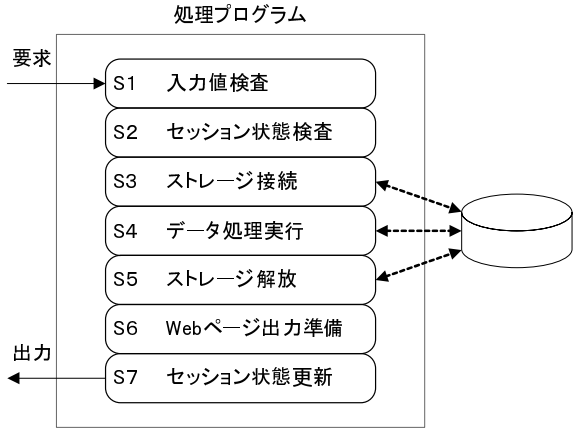

図 5.3 処理プログラムが実行する計算の流れ

生成されるすべての処理プログラムは，Web アプリケーションの実装アーキテクチャにかかわらず，一般に図 5.3 に示した計算実行手順に従う．これは，すべての処理プログラムが標準的なセキュリティ維持機能とセッション管理機能をもつ必要があるためである．

1. **入力値検査** 処理プログラムはまず，Web フォーム等からの入力値に対して文字列パターン検査，制御文字検査，最大文字列長検査，空文字検査を行う．具体的な計算手順は，入力値検査テンプレートの規定に従う．データ型定義に合致しない場合，改行以外の制御文字を含む場合，最大文字列長を超える場合，空文字列が許容されていない項目に対して空文字列が入力された場合の 1 つ以上に該当するときは，処理プログラムはエラー出力を行う．上記検査に成功した場合，文字列無害化テンプレートが規定する計算手順に従い，文字列置換を行う．

2. **セッション状態検査** 処理プログラムは，利用者が開発者の意図した順序に従って Web ページを遷移しているか，セッション状態の検査を行う．具体的な計算手順は，セッション管理テンプレートの規定に従う．セッション状態が不適切な HTTP リクエストに対して，処理プログラムはエラー出力を行う．

3. **ストレージ接続** 処理プログラムは，後述のステップ 4 で必要なデータを保持する記憶領域との接続を試みる．例えば，データベースとの接続オブジェクトを作成し接続を行う．具体的な計算手順は，記憶領域の種類に応じたストレージ接続テンプレートの規定に従う．記憶領域の不存在や認証エラー等の理由で接続に失敗した場合，処理プログラムはエラー出力を行う．

4. **データ処理実行** 処理プログラムは，入力データと記憶領域が保持するデータを用いて，データ処理を実行する．一般的には，データの検索，取得，更新の順に処理が実行される．具体的な計算手順は，選択されたデータ処理テンプレートの規定に従う．

5. **ストレージ開放** 処理プログラムは，ステップ 3 で確立した記憶領域との接続のうち，開放する必要があるものを開放し接続オブジェクトを破棄する．具体的な計算手順は，ストレージ接続テンプレートの規定に従う．

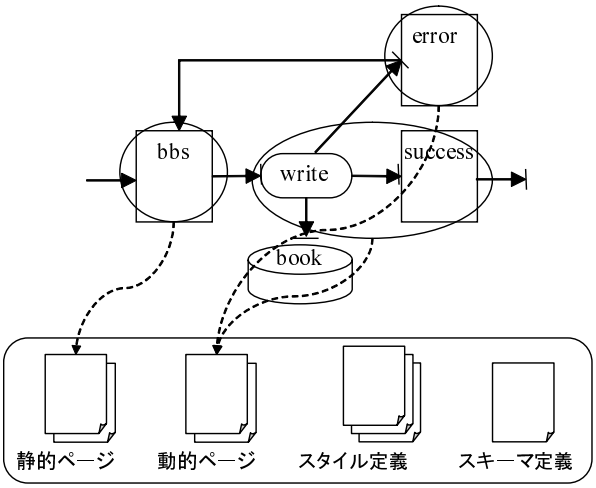

図 5.4 サーバページ型 Web アプリケーションの生成規則

6. **Web ページ出力準備** 処理プログラムは，ステップ 4 で取得したデータを用いて，Web ページ文書（またはその一部）の生成を行う．具体的な計算手順は，適用する表示テンプレートの規定に従う．

7. **セッション状態更新** 処理プログラムは，ステップ 4 の計算結果に従いセッション状態の更新を行う．例えば，クッキーやセッションオブジェクトが保持する値を更新する．具体的な計算手順は，セッション管理を行う検査テンプレートの規定に従う．

T-Web システムが生成する文書の種類は，選択した実装アーキテクチャによって異なる．しかし，サーバページ型，サーバプログラム型，ページ・プログラム分離型の各実装アーキテクチャ内では，それぞれ同様の Web 遷移図と生成文書間の対応関係をもつ場合が多い．本節では以下，各アーキテクチャの生成規則について述べる．

5.3.1 サーバページ型

サーバページ型 Web アプリケーションの生成規則の概念図を図 5.4 に示す．サーバページ型 Web アプリケーションは，静的 Web ページ文書と動的 Web ページ文書を必須の構成要素とする．Web 遷移図から生成文書への対応は以下の生成規則に従う．

生成規則 1 (サーバページ型)

1.1 (静的 Web ページ文書)

$foreach\ p \in P\ where\ p' \in P \wedge e \in E \wedge \Delta \in 2^{D \cup \overline{D}} \wedge (p, \Delta) \notin \delta(p', e)$
　$document.name = p.name;$
　$document \leftarrow Template('link', \{p' \mid p' \in P \wedge (p', \phi) \in \delta(p, \epsilon)\});$
　$foreach\ e' \in E\ where\ \delta(p, e') \neq \phi$
　　$document \leftarrow Template('inputform', p.InputForm \cap e'.In);$
　end
end

1.2 (動的 Web ページ文書 (プログラム))

$foreach\ e \in E$
$document.name = e.Transition[ok].name;$
$document \leftarrow Template('inputcheck', e.In);$
$document \leftarrow Template('session\_check', (P, \delta, s, F));$
$document \leftarrow Template('storage\_con', \{d, \overline{d} \mid d \in D \wedge p, p' \in P \wedge \Delta \in 2^{D \cup \overline{D}} \wedge (p', \Delta) \in \delta(p, e) \wedge (d \in \Delta \vee \overline{d} \in \Delta)\});$
$document \leftarrow Template('processing', e.template);$
$document \leftarrow Template('storage\_discon', \{d, \overline{d} \mid d \in D \wedge p, p' \in P \wedge \Delta \in 2^{D \cup \overline{D}} \wedge (p', \Delta) \in \delta(p, e) \wedge (d \in \Delta \vee \overline{d} \in \Delta)\});$
$foreach\ p \in P\ where\ p' \in P \wedge \Delta \in 2^{D \cup \overline{D}} \wedge (p, \Delta) \in \delta(p', e)$
$document \leftarrow Branch(e.Transition);$
$if\ p \neq e.Transition[ok]$
$document \leftarrow Template('session\_change', (P, \delta, s, F));$
$document \leftarrow Transit(p);$
$else$
$document \leftarrow Template(link', \{p' \mid p' \in P \wedge (p', \phi) \in \delta(p, \epsilon)\});$
$foreach\ e' \in E\ where\ \delta(p, e') \neq \phi$
$document \leftarrow Template('inputform', p.InputForm \cap e'.In);$
end
$document \leftarrow Template('singleview', p.SingleView);$
$document \leftarrow Template('multiview', p.MultiView);$
$end\ if$
end
$document \leftarrow Template('session\_change', (P, \delta, s, F));$
end

1.3 (動的 Web ページ文書 (その他))

$foreach\ p \in P\ where\ p' \in P \wedge e \in E \wedge \Delta \in 2^{D \cup \overline{D}} \wedge (p, \Delta) \in \delta(p', e) \wedge p \neq e.Transition[ok]$
$document.name = p.name;$
$document \leftarrow Template('link', \{p' \mid p' \in P \wedge (p', \phi) \in \delta(p, \epsilon)\});$
$foreach\ e' \in E\ where\ \delta(p, e') \neq \phi$
$document \leftarrow Template('inputform', p.InputForm \cap e'In);$
end
$document \leftarrow Template('singleview', p.SingleView);$
$document \leftarrow Template('multiview', p.MultiView);$
end

サーバページ型アーキテクチャでは，どの処理プログラムノードからも遷移先となっていない Web ページノードに対して，1 つの静的 Web ページ文書が生成される．静的 Web ページ文書は，ハイパーリンクと Web フォームの記述を含む．一方，処理プログラムノードから正常遷移先となっている Web ページノードに対して，1 つの動的 Web ページ文書が生成される．この動的 Web ページ文書は，図 5.3 に示した計算を実行する処理プログラム，ハ

	HTML	XML
Web ページ文書	静的 Web ページ文書 (*.html)	静的 Web ページ文書 (*.xml)
	動的 Web ページ文書	動的 Web ページ文書
	(*.asp,*.jsp など)	(*.asp,*.jsp など)
スタイル定義文書	CSS 文書 (*.css)	CSS 文書 (*.css)
		XSL ファイル (*.xsl)
スキーマ定義文書		DTD 文書 (*.dtd)

表 5.1 サーバページ型 Web アプリケーションを構成する文書

イパーリンク，Web フォーム，データ表示の記述を含む．さらに，処理プログラムノードからエラー遷移先となっている Web ページノードに対して，1 つの動的 Web ページ文書が生成される．この動的 Web ページ文書は，ハイパーリンク，Web フォーム，データ表示の記述を含む．

T-Web システムが生成する文書の種類は，使用する Web ページ記述言語によっても異なる．サーバページ型 Web アプリケーションを構成する文書の種類を表 5.1 に示す．また，生成される動的 Web ページ文書の記述例を図 5.5 示す．

5.3.2 サーバプログラム型

サーバプログラム型 Web アプリケーションの生成規則の概念図を図 5.6 に示す．サーバプログラム型 Web アプリケーションは，静的 Web ページ文書とサーバプログラム文書を必須の構成要素とする．Web 遷移図から生成文書への対応は以下の生成規則に従う．

生成規則 2 (サーバプログラム型)

2.1 (静的 Web ページ文書)

$foreach\ p \in P\ where\ p' \in P \wedge e \in E \wedge \Delta \in 2^{D \cup \overline{D}} \wedge (p, \Delta) \notin \delta(p', e)$
　$document.name = p.name;$
　$document \leftarrow Template('link', \{p' \mid p' \in P \wedge (p', \phi) \in \delta(p, \epsilon)\});$
　$foreach\ e' \in E\ where\ \delta(p, e') \neq \phi$
　　$document \leftarrow Template('inputform', p.InputForm \cap e'.In);$
　end
end

2.2 (サーバプログラム文書)

$foreach\ e \in E$
　$document.name = e.name;$
　$document \leftarrow Template('inputcheck', e.In);$
　$document \leftarrow Template('session\_check', (P, \delta, s, F));$
　$document \leftarrow Template('storage\_con', \{d, \overline{d} \mid d \in D \wedge p, p' \in P$
　　$\wedge \Delta \in 2^{D \cup \overline{D}} \wedge (p', \Delta) \in \delta(p, e) \wedge (d \in \Delta \vee \overline{d} \in \Delta)\});$
　$document \leftarrow Template('processing', e.template);$
　$document \leftarrow Template('storage\_discon', \{d, \overline{d} \mid d \in D \wedge p, p' \in P$

```
<%@ page language="java" contentType="text/html; charset=Shift_JIS" %>
<%@ page errorPage="error.jsp" %>
<%
    String name = request.getParameter("name");
    if(name == null) name = new String("");
    String email = request.getParameter("email");
    if(email == null) email = new String("");
    String comment = request.getParameter("comment");
    if(comment == null) comment = new String("");
    boolean _typecheck = true;
    _typecheck = _typecheck & name.matches("^.*$");
    _typecheck = _typecheck & email.matches("^[^@]+@[^.]+\\..+$|^$");
    _typecheck = _typecheck & comment.matches("^(.|\n|\r)*$");
    ------- 省略 -------
    _stmt.executeUpdate(_query);
    _stmt.close(); _conn.close();
%>
<html>
<head><title>success</title></head>
<body>
    <h1>success</h1>
    <a href="list.jsp">view</a><br/>
</body>
</html>
```

図 5.5 サーバページ型 Web アプリケーションの文書例

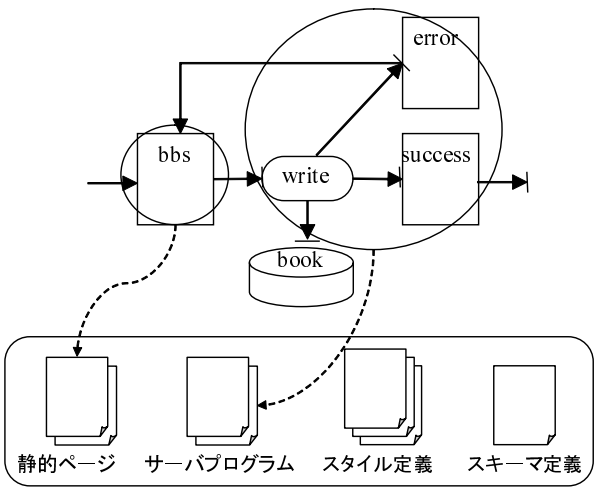

図 5.6 サーバプログラム型 Web アプリケーションの生成規則

	HTML	XML
Web ページ文書	静的 Web ページ文書 (*.html)	静的 Web ページ文書 (*.xml)
サーバプログラム文書	ソースコード (*.cgi,*.java など)	ソースコード (*.cgi,*.java など)
スタイル定義文書	CSS 文書 (*.css)	CSS 文書 (*.css) XSL ファイル (*.xsl)
スキーマ定義文書		DTD 文書 (*.dtd)

表 5.2 サーバプログラム型 Web アプリケーションを構成する文書

$$\wedge\, \Delta \in 2^{D \cup \overline{D}} \wedge (p', \Delta) \in \delta(p, e) \wedge (d \in \Delta \vee \overline{d} \in \Delta)\});$$

$foreach\ p \in P\ where\ p' \in P \wedge \Delta \in 2^{D \cup \overline{D}} \wedge (p, \Delta) \in \delta(p', e)$

$document \leftarrow Branch(e.Transition);$

$document \leftarrow Template('link', \{p' \mid p' \in P \wedge (p', \phi) \in \delta(p, \epsilon)\});$

$foreach\ e' \in E\ where\ \delta(p, e') \neq \phi$

$document \leftarrow Template('inputform', p.InputForm \cap e'.In);$

end

$document \leftarrow Template('singleview', p.SingleView);$

$document \leftarrow Template('multiview', p.MultiView);$

end

$document \leftarrow Template('session\_change', (P, \delta, s, F));$

end

サーバプログラム型アーキテクチャでは，どの処理プログラムノードからも遷移先となっていない Web ページノードに対して，1 つの静的 Web ページ文書が生成される．静的 Web ページ文書は，ハイパーリンクと Web フォームの記述を含む．また，処理プログラムノードに対して，1 つのサーバプログラム文書が生成される．サーバプログラム文書は，図 5.3 に示した計算を実行する処理プログラムのソースコード，ハイパーリンク，Web フォーム，データ表示の記述を含む．

サーバプログラム型 Web アプリケーションを構成する文書の種類を表 5.2 に示す．また，生成されるサーバプログラム文書の記述例を図 5.7 示す．

5.3.3 ページ・プログラム分離型

ページ・プログラム分離型 Web アプリケーションの生成規則の概念図を図 5.8 に示す．ページ・プログラム分離型 Web アプリケーションは，静的 Web ページ文書，サーバプログラム文書，動的 Web ページ文書を必須の構成要素とする．Web 遷移図から生成文書への対応は以下の生成規則に従う．

生成規則 3 (ページ・プログラム分離型)

3.1 (静的 Web ページ文書)

$foreach\ p \in P\ where\ p' \in P \wedge e \in E \wedge \Delta \in 2^{D \cup \overline{D}} \wedge (p, \Delta) \notin \delta(p', e)$

$document.name = p.name;$

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class sample extends HttpServlet {
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response) throws Exception {
        String name = request.getParameter("name");
        if(name == null) name = new String("");
        String email = request.getParameter("email");
        if(email == null) email = new String("");
        String comment = request.getParameter("comment");
        if(comment == null) comment = new String("");
        boolean _typecheck = true;
        _typecheck = _typecheck & name.matches("^.*$");
        _typecheck = _typecheck & email.matches("^[^@]+@[^.]+\\..+$|^$");
        _typecheck = _typecheck & comment.matches("^(.|\n|\r)*$");
        ------- 省略 -------
        _stmt.executeUpdate(_query);
        _stmt.close(); _conn.close();
        PrintWriter writer = response.getWriter();
        writer.println("<html>");
        writer.println("<head><title>success</title></head>");
        writer.println("<body>");
        writer.println("    <h1>success</h1>");
        writer.println("    <a href=\"/servlet/show\">view</a><br/>");
        writer.println("</body>");
        writer.println("</html>");
    }
}
```

図 5.7 サーバプログラム型 Web アプリケーションの文書例

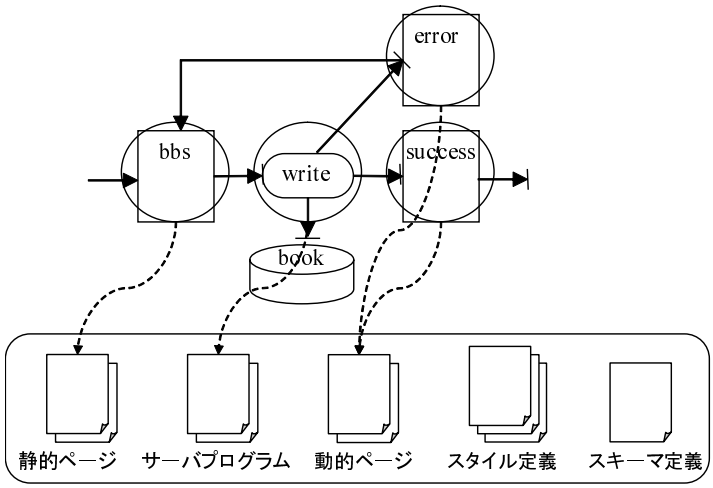

図 5.8 ページ・プログラム分離型 Web アプリケーションの生成規則

$document \leftarrow Template('link', \{p' \mid p' \in P \wedge (p', \phi) \in \delta(p, \epsilon)\});$
$foreach\ e' \in E\ where\ \delta(p, e') \neq \phi$
$document \leftarrow Template('inputform', p.InputForm \cap e'.In);$
end
end

*3.2 (*サーバプログラム文書*)*

$foreach\ e \in E$
$document.name = e.name;$
$document \leftarrow Template('inputcheck', e.In);$
$document \leftarrow Template('session_check', (P, \delta, s, F));$
$document \leftarrow Template('storage_con', \{d, \overline{d} \mid d \in D \wedge p, p' \in P \wedge \Delta \in 2^{D \cup \overline{D}} \wedge (p', \Delta) \in \delta(p, e) \wedge (d \in \Delta \vee \overline{d} \in \Delta)\});$
$document \leftarrow Template('processing', e.template);$
$document \leftarrow Template('storage_discon', \{d, \overline{d} \mid d \in D \wedge p, p' \in P \wedge \Delta \in 2^{D \cup \overline{D}} \wedge (p', \Delta) \in \delta(p, e) \wedge (d \in \Delta \vee \overline{d} \in \Delta)\});$
$foreach\ p \in P\ where\ p' \in P \wedge \Delta \in 2^{D \cup \overline{D}} \wedge (p, \Delta) \in \delta(p', e)$
$document \leftarrow Branch(e.Transition);$
$document \leftarrow Template('session_change', (P, \delta, s, F));$
$document \leftarrow Transit(p);$
end
end

*3.3 (*動的 *Web* ページ文書*)*

$foreach\ p \in P\ where\ p' \in P \wedge e \in E \wedge \Delta \in 2^{D \cup \overline{D}} \wedge (p, \Delta) \in \delta(p', e)$
$document.name = p.name;$
$document \leftarrow Template('link', \{p' \mid p' \in P \wedge (p', \phi) \in \delta(p, \epsilon)\});$
$foreach\ e' \in E\ where\ \delta(p, e') \neq \phi$

	HTML	XML
Web ページ文書	静的 Web ページ文書 (*.html) 動的 Web ページ文書 (*.jsp など)	静的 Web ページ文書 (*.xml) 動的 Web ページ文書 (*.jsp など)
サーバプログラム文書	ソースコード (*.java など)	ソースコード (*.java など)
スタイル定義文書	CSS 文書 (*.css)	CSS 文書 (*.css) XSL ファイル (*.xsl)
スキーマ定義文書		DTD 文書 (*.dtd)

表 5.3 ページ・プログラム分離型 Web アプリケーションを構成する文書

$$document \leftarrow Template('inputform', (p.InputForm \cap e'.In);$$
$$end$$
$$document \leftarrow Template('singleview', p.SingleView);$$
$$document \leftarrow Template('multiview', p.MultiView);$$
$$end$$

ページ・プログラム分離型アーキテクチャでは，どの処理プログラムノードからも遷移先となっていない Web ページノードに対して，1 つの静的 Web ページ文書が生成される．静的 Web ページ文書は，ハイパーリンクと Web フォームの記述を含む．また，処理プログラムノードに対して，1 つのサーバプログラム文書が生成される．サーバプログラム文書は，図 5.3 に示した計算を実行する処理プログラムのソースコードを有する．さらに，処理プログラムノードからの遷移先となっている Web ページノードに対して，1 つの動的 Web ページ文書が生成される．動的 Web ページ文書は，ハイパーリンク，Web フォーム，データ表示の記述を含む.

ページ・プログラム分離型 Web アプリケーションを構成する文書の種類を表 5.3 に示す．また，生成されるサーバプログラム文書，動的 Web ページ文書の記述例を図 5.9 示す.

(a) サーバプログラム

```
import javax.servlet.*;
import javax.servlet.http.*;

public class sample extends HttpServlet {
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response) throws Exception {
        String name = request.getParameter("name");
        if(name == null) name = new String("");
        String email = request.getParameter("email");
        if(email == null) email = new String("");
        String comment = request.getParameter("comment");
        if(comment == null) comment = new String("");
        boolean _typecheck = true;
        _typecheck = _typecheck & name.matches("^.*$");
        _typecheck = _typecheck & email.matches("^[^@]+@[^.]+\\..+$|^$");
        _typecheck = _typecheck & comment.matches("^(.|\n|\r)*$");
        ------- 省略 -------
        _stmt.executeUpdate(_query);
        _stmt.close(); _conn.close();
        RequestDispatcher dispatcher = getServletContext()
                                       .getRequestDispatcher("/success.jsp");
        dispatcher.forward(request,response);
    }
}
```

(b) サーバページ

```
<%@ page language="java" contentType="text/html"%>
<html>
<head><title>success</title></head>
<body>
    <h1>success</h1>
    <a href="list.jsp">view</a><br/>
</body>
</html>
```

図 5.9 ページ・プログラム分離型 Web アプリケーションの文書例

第6章

クライアント協調型Webアプリケーションの生成

本章では，拡張した Web 遷移図からクライアント協調型 Web アプリケーションを自動生成する手法および生成システムについて述べる．

6.1　生成対象アプリケーション

クライアント協調型 Web アプリケーションの生成システムについて説明する前に，提案システムが生成対象とする Web アプリケーションについて述べる．広義のクライアント協調型 Web アプリケーションは，クライアント側プログラムとサーバ側プログラム間の処理分担について何ら制限はなく，クライアント側で任意の計算が実行可能である．したがって，データマイニングなどの複雑な統計処理をクライアント側で行うことも考えられる．しかし，本研究で対象とする典型的なデータ処理中心の Web アプリケーションでは，クライアント側プログラムが以下の処理（またはその一部）を分担することを想定する．

インターフェイス制御 Web ページの状態に応じてデータや Web フォームの表示を切り替えるプログラムである．適切なインターフェイス制御により，利用者の操作ミスを防止しユーザビリティの向上を図ることができる．

入力値検査 Web フォーム等の送信イベントを受けて，HTTP リクエストの送信前に入力値の検査を行うプログラムである．入力値検査に失敗した場合，HTTP リクエストは送信せず利用者に警告を行う．利用者の操作ミスによる意図しないサーバ側プログラムの実行を防止するとともに，Web サーバの負荷を軽減することができる．

セッション管理 サーバ側プログラムと協調してセッション状態を管理し，Web ページの表示とサーバ側が保持するデータとの間の一貫性欠如を防止するプログラムである．とくに，Web ブラウザが保持する Web ページ履歴を使用した場合の一貫性欠如を防止し，ユーザビリティの向上を図ることができる．

取得データの操作 Web ページ文書の一部として取得したデータに対して，さらに計算を行うプログラムである．表示項目や表示順序の切り替え，表示データの絞込み等を，Web サーバとの通信を行わずに取得済みデータの処理によって実現する．Web ブラウザのイベントに対する応答速度を向上させ，Web サーバの負荷を軽減することができる．

Web クライアントのハードウェア資源を活用して上記処理をクライアント側で実行することにより，システム全体の処理効率を高めるとともに，Web アプリケーションの品質を向上させることができる．ただし，インターフェイス制御については，セキュリティ維持やセッション管理と密接に関係するものに生成対象を限定する．これは，ユーザビリティの向上等を目的とするものは，Web ページ文書の生成後に開発支援ツール等を用いて実装することができるからである．

上記クライアント側プログラムと協調を行うサーバ側プログラムとして，以下のデータ処理を実行する処理プログラムを対象とする．これ以外の計算を行う処理プログラムは，サーバ側でのみ実行する．

- **Get** データを記憶領域から取得するプログラムである．取得するデータの条件を入力として，取得したデータを識別する識別子を出力する．
- **Create** データを記憶領域に追加するプログラムである．追加するデータの値を入力として，追加したデータを識別する識別子を出力する．
- **Modify** 記憶領域が保持するデータを更新するプログラムである．更新するデータの条件とデータの値を入力として，更新したデータを識別する識別子を出力する．
- **Delete** 記憶領域からデータを削除するプログラムである．削除するデータの条件を入力として，出力を返さない．

上記処理プログラムは，典型的なデータ処理中心の Web アプリケーションの主要な部分を構成する．その他の処理プログラムとしては，複数のデータ間で値の一致/不一致を検査するプログラム等がある．

6.2 生成システムの概要

本研究で提案するクライアント協調型 Web アプリケーションのための生成システム（以下，本章で T-Web システムという）は，第 3 章で述べたクライアント協調型 Web アプリケーションを記述するための Web 遷移図（以下，本章で Web 遷移図という）と各ノードに関する付加的記述から，クライアント協調型 Web アプリケーションを自動生成するシステムである．

第 5 章で述べたサーバ集中型 T-Web システムは，サーバ側でのみデータ処理を行う Web アプリケーションを対象として，実装アーキテクチャに依存しない体系的な生成手法を提供する．サーバ集中型 T-Web システムを使用することで，実装アーキテクチャや実装技術の違いによるプログラム構造等の差異を意識することなく，典型的な Web アプリケーションを作成することができる．ここで，上記生成手法の実装アーキテクチャ非依存という特徴を生かし，さらにクライアント協調型 Web アプリケーションへの応用を行う．Web クライアントの処理能力を活用する Web アプリケーションでは，多様な Web クライアントに対応する必要があるため，複数の実装技術に基づく処理プログラム等を用意する必要がある．したがって，生成システムがクライアント協調型 Web アプリケーションを自動生成することにより開発効率が大きく向上する．

クライアント協調型 T-Web システムは，サーバ集中型 T-Web システムと同様，あらかじめ用意された処理プログラムテンプレートに Web 遷移図等の記述を適用することで，実

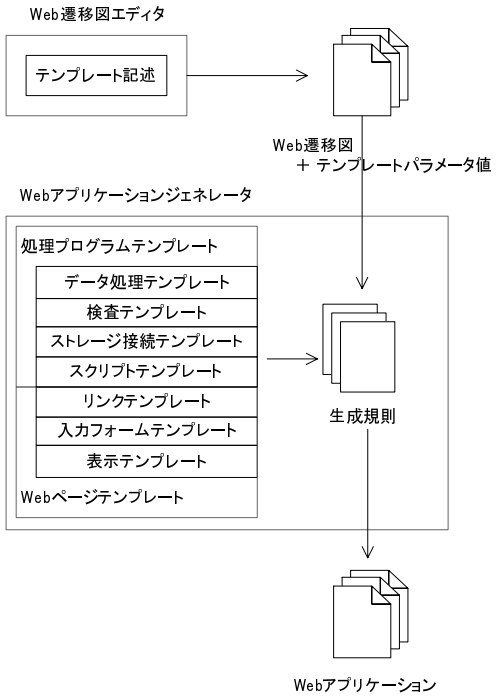

図 6.1　クライアント協調型 T-Web システムの構造

行可能な Web アプリケーションを自動生成する．ただし，1 つの処理プログラムノードに対して，サーバ側プログラムのほかクライアント側のデータ処理プログラムも必要に応じて生成される．また，Web ブラウザのイベントに反応してデータ処理プログラムを起動するためのクライアント側スクリプトも生成される．生成される Web アプリケーションは，サーバ集中型 Web アプリケーション以上のセッション管理機能とセキュリティ保持機能を備える．なお，クライアント協調型 Web 遷移図はサーバ集中型 Web 遷移図の特徴も備えるため，クライアント協調型 Web 遷移図からサーバ集中型 Web アプリケーションを生成すことが可能である．

T-Web システムの全体構造を図 6.1 に示す．T-Web システムは，Web 遷移図の記述を支援する Web 遷移図エディタと，実行可能な Web アプリケーションを生成する Web アプリケーションジェネレータの 2 つのサブシステムから構成される．Web アプリケーションジェネレータは，Web 遷移図の記述とサーバ側実装アーキテクチャを指定するパラメータ値のほか，クライアント側プログラムの実装技術を指定するパラメータ値を入力とする．ここで，サーバ側実装アーキテクチャはクライアント側プログラムの実装技術とは独立に選択することができる．Web アプリケーションジェネレータは，指定された実装アーキテクチャおよび実装技術に応じて生成規則を切り替え適切なテンプレートを適用する．クライアント協調型 T-Web システムは，処理プログラムテンプレートの一部として，クライアント側スクリプトのモジュールを記述したスクリプトテンプレートをさらにもつ．また，サーバ集中型生成システムと比べて，リンクテンプレート，検査テンプレート，ストレージ接続テンプレートが追加されている．

6.3 テンプレート

T-Web システムは，プログラムと Web ページに関する記述中の可変的要素をパラメータ化した汎用的なテンプレートを使用する．汎用的テンプレートの各パラメータに具体的な値を適用することにより，具体的計算手順を記述したコードおよび具体的な Web ページに関する記述を得る．クライアント協調型 T-Web システムは，サーバ集中型 T-Web システムと同様のテンプレートのほか，さらに以下のテンプレートをもつ．

スクリプトテンプレート

スクリプトテンプレートは，Web ブラウザのイベントに反応してクライアント側のデータ処理プログラムを呼び出すためのクライアント側スクリプト記述である．データ処理プログラムは，JavaApplet プログラムなどのダウンロード型コンポーネントとして実装される．クライアント側スクリプトは，Web ブラウザのイベントとダウンロード型コンポーネントのメソッド呼び出しとを関連付ける．JavaApplet プログラムのメソッド呼び出しの場合，JavaScript の LiveConnect を用いて実現する．例えば，Web フォームの送信イベントが発生すると，JavaScript プログラムの関数が実行され JavaApplet プログラムのメソッド呼び出しが行われる．そして，JavaScript プログラムは JavaApplet プログラムから計算結果を受け取り，計算結果に応じて HTTP リクエストの送信や Web ページの表示変更等の処理を行う．

Web フォームのイベントを受け取るスクリプトテンプレートの記述例を以下に示す．このプログラムは，Web フォームの記述中に埋め込まれる．

```
onSubmit="return /*FORM*/_/*METHOD*/()"
```

データ処理プログラムのメソッドを呼び出すスクリプトテンプレートの記述例を以下に示す．このプログラムは，Web ページ文書に含まれるかまたは Web ページ文書が参照するスクリプト文書に含まれる．

```
function /*FORM*/_/*METHOD*/() {
    _result = true;
    /** _result = _result && document./*APPLET*/
            ./*APPMETHOD*/(document./*FORM*/./*VALUE*/.value); **/
    if(_result) {
        return true;
    } else {
        /*RESPONSE*/; return false;
    }
}
```

リンクテンプレート

リンクテンプレートとして，Web ページ文書からクライアント側のデータ処理プログラムを参照する Web ページ記述を追加する．データ処理プログラムの実装技術に応じた Web ページ記述が用意される．

データ処理プログラムを参照するリンクテンプレートの記述例を以下に示す．このテンプレートは，JavaApplet プログラムを参照する．

```
<applet code="/*FILE*/" width="0" height="0" name="/*APPLET*/">
</applet>
```

検査テンプレート

検査テンプレートとして，クライアント側でセッション管理を行うためのプログラムモジュールを追加する．クライアント側プログラムは，Web フォームからの HTTP リクエストの送信とそれに対する HTTP レスポンスの受信状態を管理することで，Web ページの表示とサーバ側のセッション状態との間の一貫性欠如を防止する．

クライアント側プログラムのためのセッション管理テンプレートの記述例を以下に示す．このプログラムは，Web フォームの状態を管理する．

```
function startForm/*FORM*/() {
    _state = getCookie(/*FORMID*/);
    if(_state == "") {
        setCookie(/*FORMID*/, "0");
    } else if(_state == "1") {
        /*RESPONSE1*/; transit(/*TARGET*/);
    } else if(_state == "2") {
        /*RESPONSE2*/; transit(/*TARGET*/);
    } else if(_state == "3") {
        /*RESPONSE3*/;
    } else if(state == "-1") {
        /*RESPONSE-1*/;
    }
}
```

ストレージ接続テンプレート

ストレージ接続テンプレートとして，Web ブラウザが取得したデータをデータ処理プログラムが操作するために必要なプログラムモジュールを追加する．このプログラムは，Web ページ文書に含まれる単一データ要素または複数データ要素を文字列として保持し，データ処理プログラムに対してデータアクセス手段を提供する．なお，JavaApplet プログラムの場合，Web ページの表示切り替えにより保持するデータが破棄されるのを防止するため，Applet クラスの静的変数に取得したデータを挿入する．

クライアント側プログラムのためのストレージ接続テンプレートの記述例を以下に示す．このテンプレートは，複数データ要素中の現在参照している位置から一定個数のデータを取得する．

```
private static String /*DATA*/_data = new String("");
private int /*DATA*/_ptr = 0;
```

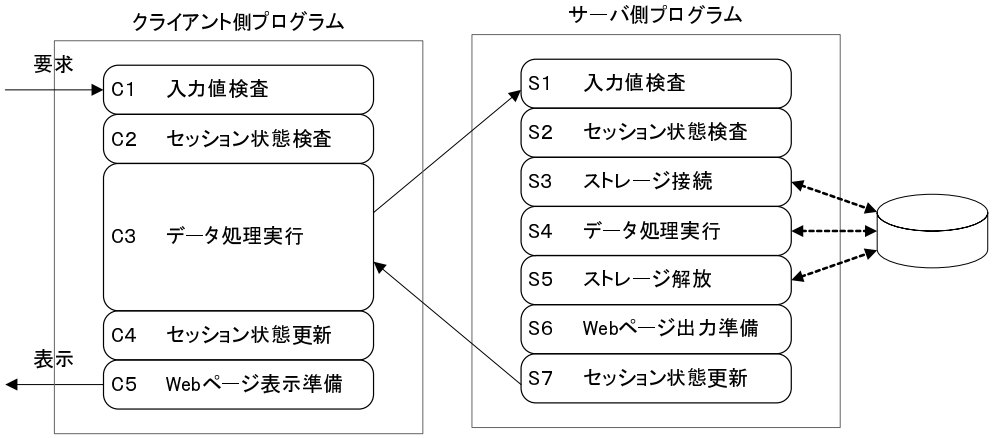

図 6.2 処理プログラムが実行する計算の流れ

```
private int /*DATA*/_size = 0;

public String get/*DATA*/Data() {
    StringBuffer _result = new StringBuffer();
    String[] rows = /*DATA*/_data.split("/*ROWTAG*/");
    for(int i = /*DATA*/_ptr; i < /*DATA*/_size
                          && i < /*DATA*/_ptr + /*AMOUNT*/; i++) {
        _result.append(rows[i] + "/*ROWTAG*/");
    }
    _result.append(rows[rows.length - 1]);
    _return result.toString();
}
```

6.4 クライアント側プログラム

Web アプリケーションジェネレータは，Web 遷移図等の記述を処理プログラムテンプレートと Web ページテンプレートに適用することで，実行可能な Web アプリケーションを生成する．具体的には，テンプレート中の”/*”と”*/”で囲まれた部分を対応するパラメータ値に置換し，”/**”と”**/”で囲まれた部分を対応するパラメータ値の個数だけ繰り返した文字列に置換する．

生成されるクライアント側プログラムは，実装技術にかかわらず，一般に図 6.2 に示した計算実行手順に従う．

1. **入力値検査** クライアント側プログラムはまず，Web フォーム等からの入力値に対して文字列パターン検査，制御文字検査，最大文字列長検査，空文字検査を行う．具体的な計算手順は，サーバ側プログラムの入力値検査と同様である．入力値検査に失敗した場合，クライアント側プログラムはデータ処理を実行せず，Web ページ表示を更新して利用者に警告を行う．

	HTML	XML
Web ページ文書	静的 Web ページ文書 (*.html)	静的 Web ページ文書 (*.xml)
	動的 Web ページ文書 (*.jsp など)	動的 Web ページ文書 (*.jsp など)
プログラム文書	サーバコード (*.java など)	サーバコード (*.java など)
	クライアントコード (*.java など)	クライアントコード (*.java など)
	スクリプト文書 (*.js など)	スクリプト文書 (*.js など)
スタイル定義文書	CSS 文書 (*.css)	CSS 文書 (*.css)
		XSL ファイル (*.xsl)
スキーマ定義文書		DTD 文書 (*.dtd)

表 6.1 クライアント協調型 Web アプリケーションを構成する文書

2. **セッション状態検査** クライアント側プログラムは，利用者が開発者の意図した順序に従って Web ページを遷移しているか，セッション状態の検査を行う．とくに，Web フォームの送信イベントに対してデータ処理を許可すべきか否か検査する．セッション状態検査に失敗した場合，クライアントプログラムはデータ処理を実行せず，Web ページの表示を更新して利用者に警告を行う．

3. **データ処理実行** 処理プログラムは，Web フォーム等の入力データとクライアント側プログラムが保持するデータから，Web サーバへのアクセスが必要か否か判断する．アクセスが必要な場合，処理プログラムは HTTP リクエストを送信し Web ページ文書を取得して Web ページの表示を切り替える．アクセスが不要な場合，処理プログラムはデータ処理を実行し，計算結果に応じて Web ページの表示の一部を更新する．具体的な計算手順は，選択されたデータ処理テンプレートの規定に従う．

4. **セッション状態更新** 処理プログラムは，ステップ 3 の計算結果に従いセッション状態の更新を行う．例えば，クッキーが保持する値の更新を行う．具体的な計算手順は，セッション管理テンプレートの規定に従う．

なお，クライアント側プログラムが入力値検査を行う場合でも，サーバ側プログラムは別途入力値検査の機能を備える必要がある．これは，開発者の意図した Web ページから HTTP リクエストが送信されるとは限らないからである．

T-Web システムが生成する文書の種類は，使用するサーバ側実装アーキテクチャ，クライアント側実装技術，Web ページ記述言語によって異なる．クライアント協調型 Web アプリケーションを構成する文書の種類例を表 6.1 に示す．以下，クライアント側プログラムとサーバ側プログラムが協調して処理を実行する例を説明する．

6.4.1 入力値検査

入力値検査を行うクライアント側プログラムは，Web フォームの送信イベントに反応して実行されるクライアント側スクリプトと，入力値検査の計算を行うダウンロード型コンポーネントから構成される．Web 遷移図の各入力フォームに対して，スクリプト文書とプログラム文書が生成される．また，入力フォームをもつ Web ページノードから生成される

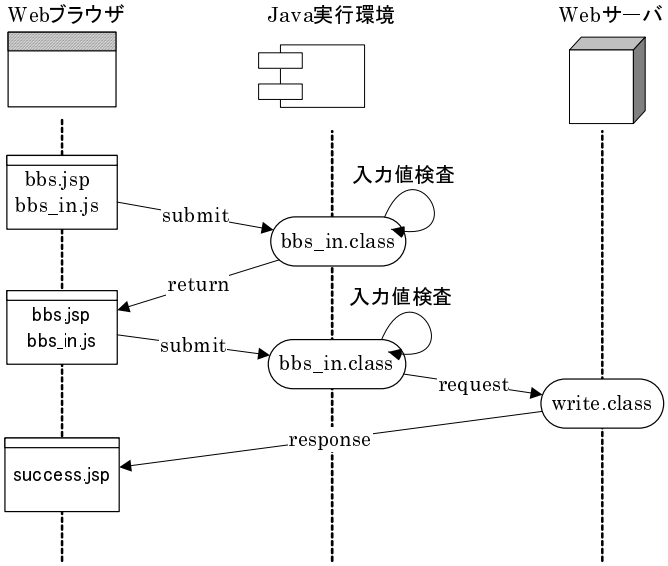

図 6.3　入力値検査を行うクライアント側プログラムの実行の流れ

図 6.4　入力値検査を行うクライアント側プログラムの実行例

Web ページ文書（または Web ページ記述を含むプログラム文書）には，上記スクリプト文書とダウンロード型コンポーネントを参照する Web ページ記述が埋め込まれる．

入力値検査を行うクライアント側プログラムの実行の流れを図 6.3 に示す．このプログラムは，第 3 章で示した Web 遷移図の記述例を T-Web システムの入力として与えた場合に出力されるプログラムである．Web ページ”bbs”の Web フォームから値を入力する場合，例えばメールアドレス欄の入力値が不適切であるとクライアント側プログラムの入力値検査に失敗し，Web ページにエラーメッセージが表示される．このとき，Web サーバに対して HTTP リクエストは送信されない．すべての入力値について入力値検査が成功した場合のみ，HTTP リクエストが送信される．なお，入力値検査が成功した場合に，入力値の確認を利用者に促した後 HTTP リクエストを送信することも可能である．入力値検査を行うクライアント側プログラムの実行例を図 6.4 に示す．また，生成されるクライアント側プログラムの記述例を図 6.5 に示す．

(a) スクリプト

```
function in_check() {
    _result = true;
    _result = _result && document.bbs_in.checkName(
                                    document.in.name.value);
    _result = _result && document.bbs_in.checkEmail(
                                    document.in.email.value);
    _result = _result && document.bbs_in.checkComment(
                                    document.in.comment.value);
    if(_result) { return true; }
        else { return false; }
}
```

(b) 処理プログラム

```
public class bbs_in extends java.applet.Applet {
    public boolean checkName(String value) {
        boolean _result = true;
        _result = _result & value.matches("^.*$");
        _result = _result & value.matches(
                  "^[^\\x00-\\x09\\x0B\\x0C\\x0E-\\x1F\\x7F]*$");
        _result = _result & value.length() <= 50;
        return _result;
    }
    ------- 省略 -------
}
```

図 6.5 入力値検査を行うクライアント側プログラムの記述例

6.4.2 セッション管理

セッション管理を行うクライアント側プログラムは，Web フォームの送信イベントに反応して実行されるクライアント側スクリプト，Web フォームの表示時に実行されるクライアント側スクリプト，セッション管理の計算を行うダウンロード型コンポーネントから構成される．Web 遷移図の各入力フォームに対して，スクリプト文書とプログラム文書が生成される．また，入力フォームをもつ Web ページノードから生成される Web ページ文書（または Web ページ記述を含むプログラム文書）には，上記スクリプト文書とダウンロード型コンポーネントを参照する Web ページ記述が埋め込まれる．

クライアント側とサーバ側の Web フォームの状態遷移を図 6.6 に示す．この状態は，サーバ側プログラムの実行時に各 Web フォームのインスタンスに対してそれぞれ割り当てられる．サーバ側の状態の意味は以下の通りである．

- **Server0**: Web フォームからデータを受信していない．
- **Server1**: Web フォームからデータを受信し，データ処理の実行中である．
- **Server2**: Web フォームからデータを受信し，処理結果を Web ブラウザへ送信済である．

また，クライアント側の状態の意味は以下の通りである．

- **Client0**: Web フォームからデータを送信していない．
- **Client1**: Web フォームからデータを送信し，Web サーバに到達したか不明である．
- **Client2**: Web フォームからデータを送信し，Web サーバに到達している．
- **Client3**: Web フォームからデータを送信し，結果を受信済である．

サーバ側プログラムは，Web フォームからの HTTP リクエストを受信すると以下のセッション管理を行う．

1. 状態を表す値が存在しない場合，一意な ID を新規に発行して Web フォームを送信し，状態を Server0 に設定する．
2. 状態 Server0 の場合，受信したデータに基づきデータ処理を開始し，状態 Server1 とする．データ処理が終了すると処理結果を一時記憶領域に保存してから Web ブラウザへ返信し，状態を Server2 とする．
3. 状態 Server1 の場合，データ処理の実行中である旨を返信する．
4. 状態 Server2 の場合，一時記憶領域から処理結果を取得し返信する．

一方，クライアント側プログラムは，Web フォームの表示および送信イベントに対して以下のセッション管理を行う．

1. 状態を表す値が存在しない時に Web フォームを受信した場合，状態を Client0 に設定する．

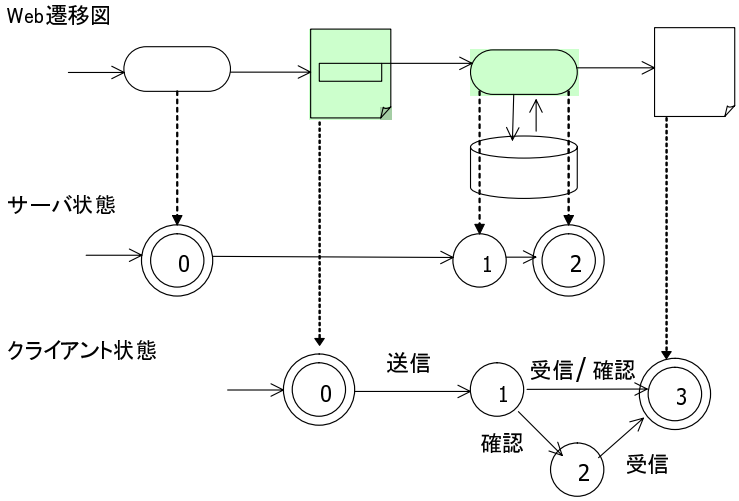

図 6.6　クライアント側とサーバ側のセッションの状態遷移

2. 状態 Client0 のときに送信イベントが発生した場合，状態を Client1 とし，データを Web サーバへ送信する．Web サーバから処理結果を受信した場合，状態を Client3 とする．

3. 状態 Client1 のときに送信イベントが発生した場合，確認要求メッセージを Web サーバへ送信する．データ処理の実行中である旨を受信した場合は状態を Client2 とし，処理結果を受信した場合は状態を Client3 とする．

4. 状態 Client2 のときに送信イベントが発生した場合，確認要求メッセージを Web サーバへ送信する．データ処理の実行中である旨を受信した場合は状態を Client2 とし，処理結果を受信した場合は状態を Client3 とする．

5. 状態 Client3 のときに送信イベントが発生した場合，送信しない．

なお，クライアント側プログラムは，状態 Client2 のとき一定時間ごとに確認要求メッセージを Web サーバへ送信することも可能である．また，状態 Client3 のときに送信イベントが発生した場合，警告メッセージを表示し利用者に確認を促すことも可能である．上記方法により，セッション管理の戻り問題，2 度書き問題，2 度押し問題を解決することができる．また，上記セッション管理方法により，Web ページ文書を取得する前に HTTP 接続が切断された場合でも適切に処理を継続することが可能である．

セッション管理を行うクライアント側プログラムの実行例を図 6.7 に示す．また，生成されるクライアント側プログラムの記述例を図 6.8 に示す．

6.4.3　取得データの操作

データ操作を行うクライアント側プログラムは，Web フォームのイベントに反応して実行されるクライアント側スクリプトと，データ処理の計算を行うダウンロード型コンポーネントから構成される．Web 遷移図の各複数表示要素に対して，スクリプト文書とプログラム文書が生成される．また，複数表示要素をもつ Web ページノードから生成される Web

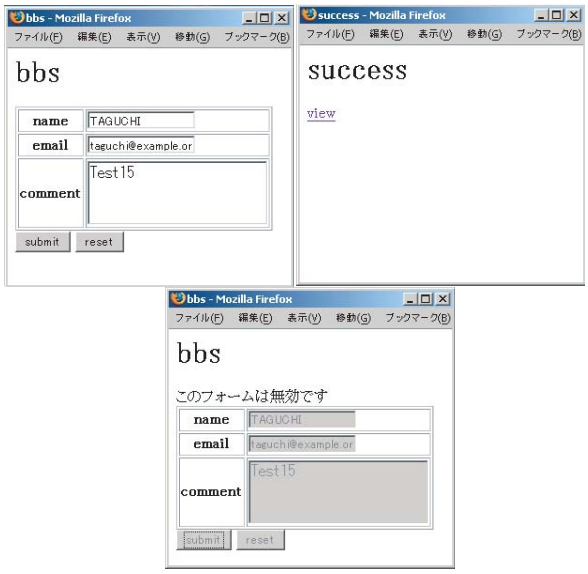

図 6.7 セッション管理を行うクライアント側プログラムの実行例

```
function start_form() {
    state = get_cookie(formID);
    if(state == "") {
        enable_form();
        set_cookie(formID, "0");
    } else if(state == "1") {
        disable_form_in();
        document.getElementById("in_msg").innerHTML
                                = "サーバにアクセス中です";
    } else if(state == "2") {
        disable_form_in();
        document.getElementById("in_msg").innerHTML
                                = "サーバで要求を処理中です";
    } else if(state == "3") {
        disable_form_in();
        document.getElementById("in_msg").innerHTML
                                = "このフォームは無効です";
    }
}
------- 以下省略 -------
```

図 6.8 セッション管理を行うクライアント側プログラムの記述例

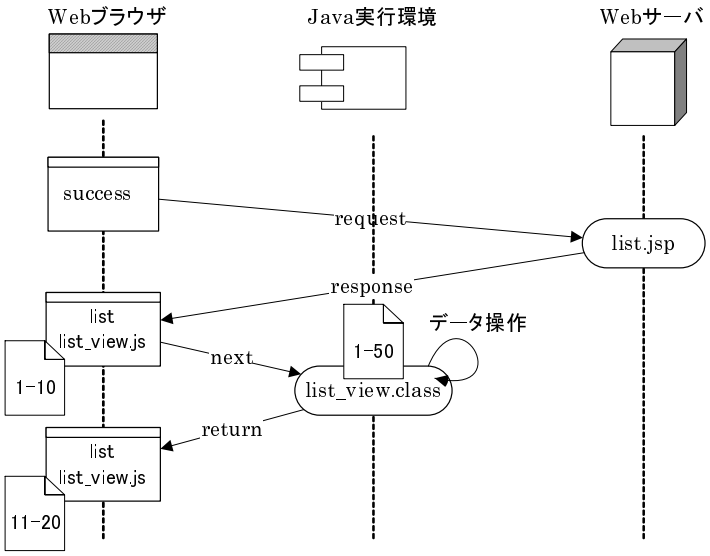

図 6.9　データ操作を行うクライアント側プログラムの実行の流れ

ページ文書（または Web ページ記述を含むプログラム文書）には，上記スクリプト文書とダウンロード型コンポーネントを参照する Web ページ記述が埋め込まれる．ただし，複合データ型のメタデータで”S-”と規定されているデータに対しては，上記クライアント側プログラムを生成しない．そのデータをクライアントによる操作から保護する必要があるからである．

データ操作を行うクライアント側プログラムの実行の流れを図 6.3 に示す．このプログラムは，第 3 章で示した Web 遷移図の記述例を T-Web システムの入力として与えた場合に出力されるプログラムである．Web ページ文書”list”がもつ複合データ”book”のデータ数が 1 ページとして表示すべき個数を超える場合，クライアント側プログラムがデータの分割表示を行う．処理プログラムは一括して取得したデータを保持し，その範囲内でのページ切り替えは Web ページ記述の一部の書き換えにより実現する．処理プログラムが保持するデータの範囲を超えるページ切り替えが発生すると，処理プログラムは HTTP リクエストを送信する．なお，取得したデータに対して，Web フォームの入力値に基づいたデータ検索を行うことも可能である．データ操作を行うクライアント側プログラムの実行例を図 6.4 に示す．また，生成されるクライアント側プログラムの記述例を図 6.5 に示す．

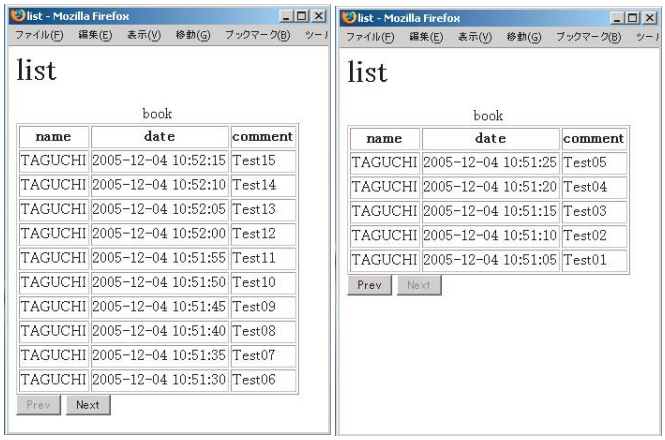

図 6.10　データ操作を行うクライアント側プログラムの実行例

(a) スクリプト

```
function view_getData() {
    document.getElementById("view").innerHTML
                = document.list_view.getData();
}
function view_prev() {
    document.list_view.prev();
    view_getData();
}
function view_next() {
  document.list_view.next();
  view_getData();
}
```

(b) 処理プログラム

```
public class list_view extends java.applet.Applet {
    private static String data = new String("");
    private static final int amount = 10;
    private int pointer = 0;
    private int size = 0;

    public String getData() {
        StringBuffer result = new StringBuffer();
        String[] rows = data.split("</tr>|</TR>");
        for(int i = pointer; i < size && i < pointer + amount; i++) {
            result.append(rows[i] + "</TR>");
        }
        return result.toString();
    }
    public void prev() {
        pointer = pointer - amount;
        if(pointer < 1) pointer = 1;
    }
    public void next() {
        pointer = pointer + amount;
    }
    ------- 省略 -------
}
```

図 6.11 データ操作を行うクライアント側プログラムの記述例

第7章

評価

本章では，提案生成手法および生成システムについて評価を行う．まず提案手法の特徴について述べ，T-Web システムを用いた Web アプリケーションの開発例を示す．また，学生被験者による評価実験について説明し，実験結果の分析を行う．さらに，Web アプリケーションの開発支援に関する関連技術との比較を行う．

7.1　T-Web システムを用いた開発

7.1.1　生成システムの対象

提案生成手法および生成システムは，典型的なデータ処理中心の Web アプリケーションを容易かつ迅速に作成することを支援する．T-Web システムがもつ生成規則やテンプレートを適宜切り替えることで，多くの Web アプリケーション開発に柔軟に対応することができる．しかし，生成手法が有する特徴から，とくに以下の対象に対してその有効性を発揮することができる．

利用対象者 T-Web システムは，Web アプリケーション開発に精通していない技術者や非技術者を主な利用対象者とする．データ処理中心の Web アプリケーションを作成するためには，一般的なプログラミングに関する知識のほか，データベースの利用方法，ネットワークセキュリティ，Web ページのデザインなど広範囲にわたる知識を必要とする．しかし，これら技術的知識を短期間に習得することは困難である．T-Web システムは，このうちプログラミング，データベース，ネットワークセキュリティ等の実装技術に関する知識を隠蔽する．一方，T-Web システムの利用者は，Web ページの構成や用意すべき入出力インターフェイスなど，生成するアプリケーションの利用に関する知識を必要とする．したがって，Web アプリケーションの利用者や，利用者に近い立場にある管理者，プロトタイプ開発者等の利用にとくに有効である．

対象アプリケーション データ処理中心の Web アプリケーションの中でも，とくに Web コンテンツの検索・追加・更新を行う，情報共有のための Web アプリケーション開発に有効である．T-Web システムは，専用のパッケージ製品を利用する場合と比べ，柔軟な Web ページの構成や多様な Web コンテンツの管理を実現することができる．T-Web システムでは，例えば以下のような Web アプリケーションを生成することができる．

- 日記サイトや個人の情報提供ページなど，個人がもつ情報を効率的に管理し発信するための Web アプリケーション．
- 掲示板やグループの予定管理システム，会議室予約ページなど，複数人が特定の情報を共有するための Web アプリケーション．
- 会員登録ページや会議運営システムなど，Web ページ開設者と Web 利用者との間の対話を実現するシステムにおいて，フロントエンドとして動作する Web アプリケーション．この場合，一般に外部プログラムや他のネットワーク技術との連携が必要となる．

対象利用場面 T-Web システムは，容易かつ迅速な Web アプリケーション開発が求められる場面で有効である．例えば，短期プロジェクトの予定管理を行う場合や一時的なイベントにより大量のデータを共有する場合など，迅速な複数人間の情報共有を実現する方法として利用することができる．また，中規模以上のソフトウェア開発において，開発の初期段階でのプロトタイプ生成にも利用することができる．なお，一時的な情報共有のための Web アプリケーションでは，高性能な Web サーバを設置する場合は少ないと考えられることから，クライアント協調型 Web アプリケーションを利用することによる Web サーバの負荷軽減が有効である．

7.1.2 生成システムの特徴

提案生成手法および生成システムが有する特徴について述べる．

データ中心開発 T-Web システムが採用する生成手法では，データ中心型の開発プロセスをとることが多い．典型的なデータ処理中心の Web アプリケーションでは，まずシステムが保持すべきデータの設計を行うことで，必要となる処理プログラムを比較的容易に抽出できるからである．また，典型的な Web アプリケーションで頻繁に使用されるデータ型が多く存在することから，過去のデータ設計を再利用することもできる．なお，サーバ集中型 T-Web システムとクライアント協調型 T-Web システムでは，Web フォームやデータ表示手段等の定義は複合データ型に対応付けて管理されるため，開発中のデータ型の変更も柔軟に行うことができる．例えば，複合データ型の定義を変更すると，それに対応して Web フォームにも自動的に変更が反映される．一方，T-Web システムでは既存のソフトウェア資産やシステムとの連携は限定的となる．

安全性維持機能 T-Web システムが生成する Web アプリケーションは，典型的な攻撃から Web アプリケーションを防御する標準的な安全性維持機能をもつ．例えば，第 2 章で述べた典型的な攻撃方法に対して，T-Web システムは以下に示す効果をもつ．

- **バッファオーバーフロー** すべての入力値に対して最大文字列長を検査するため原則として防御可能である．
- **クロスサイトスクリプト** スクリプトタグを含む HTML タグに対して無害化処理を行っているため多くの攻撃は防御可能である．
- **パラメータ改ざん** すべての入力値に対して型検査を行っているため想定していない入力値は受け付けない．

- **バックドアとデバッグオプション** T-Web システムは脆弱性の原因となるデバッグ用の機能を生成することはない.
- **強制的ブラウズ** Web サーバ上の文書配置の問題であり T-Web システムは防御手段を提供しない.
- **セッションハイジャック/リプレイ** 解読が容易なセッション ID を意識的に使用することはないが，安全性は Web アプリケーションの実行環境に依存する.
- **パスの乗り越え** 既定のテンプレートはローカルファイルを参照するプログラムを含まないが，そのようなテンプレートを追加した場合には危険性が高まる.
- **SQL の挿入** SQL 文を区切る文字に対して無害化処理を行っているため多くの攻撃は防御可能である.
- **OS コマンドの挿入** 既定のテンプレートは OS コマンドを実行するプログラムを含まないが，そのようなテンプレートを追加した場合には危険性が高まる.
- **クライアント側コメント** T-Web システムが生成する Web アプリケーションは不必要なクライアント側コメントをもたない.
- **エラーコード** T-Web システムが生成する Web アプリケーションは，すべてのエラーに対してエラー処理を行うためエラーメッセージがそのまま Web ブラウザへ出力されることはない.

生成アプリケーションの制限 T-Web システムは，ダイアグラムの可読性を高め迅速な Web アプリケーションの生成を実現するため，生成可能な Web アプリケーションに一定の制限を加えている．T-Web システムではマルチフレームを扱っていない．マルチフレームを記述できるように Web 遷移図を拡張することは可能であるが，この場合，複数のフレーム間での相互作用に関する記述が複雑になり Web 遷移図の可読性が低下するからである．また，クリッカブルマップのようにデータフローとマルチメディアデータとが密接に結合する Web アプリケーションの生成には適さない．T-Web システムの生成手法では，データフローが定義済の Web ページ文書を生成後に修正して音声・画像等のデータを参照させるという開発プロセスをとるためである.

なお，T-Web システムの実装では HTTPS など HTTP 以外のプロトコルを使用する Web アプリケーションを直接の生成対象としていない．しかし，ハイパーリンクの URL に HTTPS プロトコルを指定し，生成アプリケーションを実行する Web サーバの設定を変更することで，HTTPS プロトコルを使用した Web アプリケーションを作成できる．ただし，HTTPS プロトコルにより保護された Web ページと保護されていない Web ページとの間をハイパーリンクを辿って往復すると，十分な安全性が維持できない場合がある．そこで，HTTPS プロトコルを用いた Web アプリケーションを生成する場合，同一セッション内で HTTP と HTTPS とを混在させるべきではない.

また，T-Web システムは，クライアント側プログラムとサーバ側プログラムが非同期通信を行うクライアント協調型 Web アプリケーションを生成対象としていない．これは，現在の Web ブラウザの実装では，クライアント側プログラムにより非同期通信が行われると，Web ページの履歴が残らないという問題があるためである．ただし，既に取得したデータの範囲を超えてページ遷移をするときに利用者にボタン等を押してもらうことで，例えば GoogleMaps 等に類似する機能を実現することが可能である.

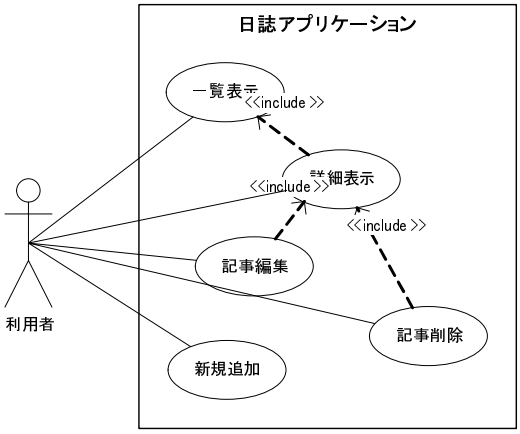

図 7.1 日誌アプリケーションのユースケース図

7.1.3 Web アプリケーションの生成例

本節では，試作実装した T-Web システムを用いて Web 遷移図から Web アプリケーションを自動生成する例を示す．ここでは，基本的な情報共有機能を有する，複数人が書き込み可能な日誌アプリケーションを作成する．生成システムとしてクライアント協調型 T-Web システムを用い，同一ダイアグラムからそれぞれページ・プログラム分離型 Web アプリケーションと，さらにクライアント側プログラムを有するクライアント協調型 Web アプリケーションを生成する．Web アプリケーションの仕様は以下の通りである．また，これに対応するユースケース図を図 7.1 に示す．

- 利用者は新規記事を追加することができる (日時は Web アプリケーションにより自動的に付与される)．
- 利用者は記事一覧を取得することができる (書誌的事項のみ)．
- 利用者は指定した個別記事の詳細を見ることができる．
- 利用者は指定した記事を編集し保存することができる．
- 利用者は指定した記事を削除することができる．
- 保存されるべきデータは（名前，電子メールアドレス，タイトル，日付，本文）である．

上記要求仕様から Web 遷移図を構成する．クライアント協調型 T-Web システムでは，まず XML Schema の文法にしたがって基本データ型および複合データ型を定義する．ただし，T-Web システムは Email 型や DateTime 型など典型的な Web アプリケーションで頻繁に使用される基本データ型をもつため，ここではこれら基本データ型を使用する．仕様に従い 5 つの基本データ型から構成される複合データ型”book”を定義する (図 7.2)．

日誌アプリケーションの Web 遷移図を図 7.2 に示す．各 Web ページは，ユースケース図の各ユースケースに対応付けて導出されたものである．利用者は，記事一覧が表示される Web ページ”diary”からセッションを開始する．利用者は記事一覧から 1 つの記事を選択し，Web ページ”detail”で選択した記事の詳細を見ることができる．さらに利用者はその記事の

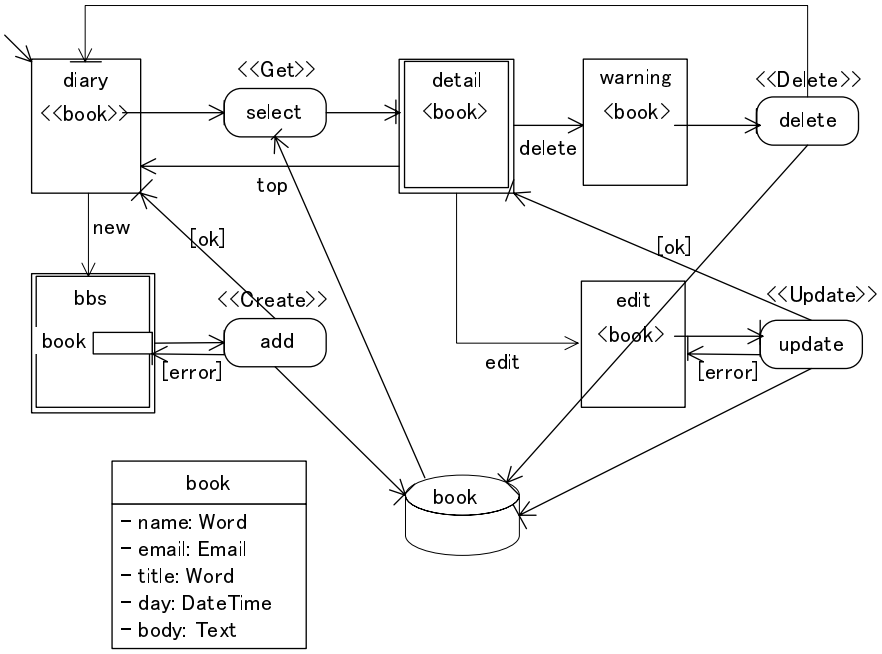

図 7.2 日誌アプリケーションの Web 遷移図

	サーバ集中型	クライアント協調型
サーバページ	6 (688 行)	6 (689 行)
サーバプログラム	5 (536 行)	5 (536 行)
クライアントプログラム	0	2 (155 行)

表 7.1 日誌アプリケーションで生成されるファール数と行総数

削除を確認するための Web ページ”warning”か編集を行うための Web ページ”edit”に遷移することができる．また，セッション開始ページから記事の追加を行う Web フォームをもつ Web ページ”write”へ遷移することもできる．この Web アプリケーションでは，データの取得（Get），追加（Create），更新（Update），削除（Delete）を意味する処理プログラムノードがそれぞれ 1 つずつ用いられている．

上記 Web 遷移図をクライアント協調型 T-Web システムの入力として与えたときに生成された文書の数とコード行総数を表 7.1 に示す．サーバ側実装アーキテクチャとしてページ・プログラム分離型アーキテクチャを用いているため，各 Web ページノードに対して 1 つのサーバページ文書が生成される．また，各処理プログラムノードに対して 1 つのサーバプログラム文書が生成される．クライアント協調型 Web アプリケーションの場合，さらにクライアント側プログラムとして Web フォームの入力値検査を行うプログラムと記事一覧の表示を切り替えるプログラムが生成される．なお，生成されるコードが 1 文書あたり平均 100 行を超えているのは，一貫性や安全性維持に必要なコードが多くなるからである．生成された Web アプリケーションの実行例を図 7.3 に示す．

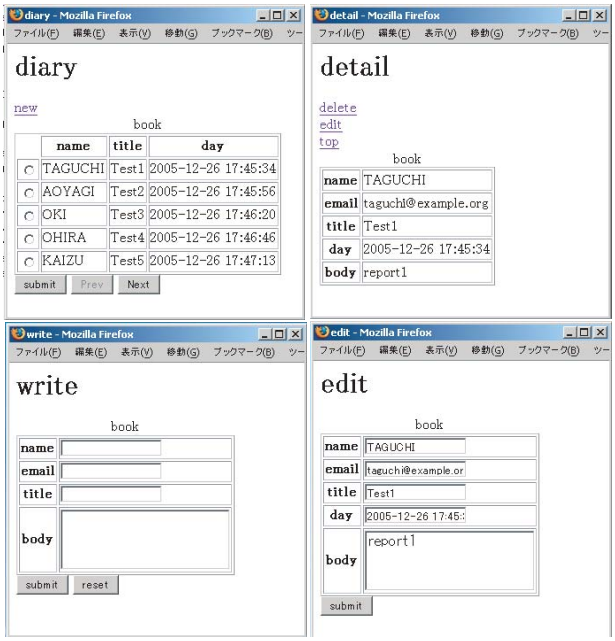

図 7.3　日誌アプリケーションの実行例

7.2　評価実験

本研究では，試作実装した T-Web システムを用いて学生被験者による評価実験を行った．ここでは，行った実験の方法について説明し，実験結果の分析を行う．

7.2.1　実験方法

評価実験では，Web アプリケーションの開発経験がある 10 名の大学生および大学院生に被験者として協力してもらい，T-Web システムを用いて自動生成した場合と従来の開発支援技術を用いた開発を行った場合とで，開発に要する時間の違いを計測した．T-Web システムを用いた開発時間が，従来方法による開発時間に比べてどの程度短縮されたかを計測することで，T-Web システムおよびその生成手法の有用性を確認する．また，被験者からのコメントにより提案手法の利点と問題点を明らかにすることも評価実験の目的である．

客観的に比較可能な計測時間を得るため，被験者全員に同一 Web アプリケーションを T-Web システムを用いた場合と従来手法による場合との 2 通りで作成してもらった．作成する Web アプリケーションは，本章で例示した日誌アプリケーションである．被験者には，課題をフェーズ 1 とフェーズ 2 の 2 回に分けて実験をしてもらった．実験の具体的指示内容は以下の通りである．

フェーズ 1　簡易個人日記アプリケーションを既存のコンポーネント等を利用せずに新規に開発する．フェーズ 1 では，サーバ側プログラムのみを用いて実現する．要求は以下に挙げる 5 項目である．

1. 利用者は新規記事を入力することができる．
2. 利用者は過去の記事の一覧を取得することができる．

3. 利用者は指定した記事を編集し保存することができる．
4. 利用者は指定した記事を削除することができる．
5. システムが保持すべきデータは（タイトル，日付，本文）とする．

問題を簡単にするため，認証等の機能は不要とした．また，仕様の曖昧な部分は任意に解釈してよいこととした．

フェーズ 2 フェーズ 1 で作成した個人日記アプリケーションを拡張し，複数人で共有する日誌アプリケーションを作成する．また，クライアント側プログラムを用いたデータ処理も実装する．要求は以下に挙げる 3 項目である．

1. 個人日記アプリケーションを複数人共有の日誌アプリケーションに拡張するため，保存データとして（名前，電子メールアドレス）を追加する．これにともない，Web フォームの入力欄等も変更する．
2. クライアント側プログラムを用いて，電子メールアドレスの入力値検査を行う．Web フォームの入力値が電子メールアドレスの文字列パターンに合致しないときは，Web サーバに HTTP リクエストを送信しない．
3. クライアント側プログラムを用いて，記事一覧のページ切り替えを実現する．Web 文書中のデータが一定個数以上ある場合には，Web ページの表示を分割する．

フェーズ 1 と同様，仕様の曖昧な部分は任意に解釈してよいこととした．また，要求 3 については，一定時間を経過しても完成の見込みがない場合は，未完成でもよいこととした．

上記課題を行う具体的な手順は以下の通りである．

1. 被験者全員に対して，T-Web システムの操作方法の説明を行う．約 1 時間かけて Web 遷移図の記述方法と Web アプリケーションの生成手順をデモンストレーションを行いながら説明する．

2. 説明を受けた被験者は，実験に着手する前に各自 T-Web システムを使用するトレーニングを行う．トレーニングは被験者がシステムを理解できたと思うまで継続し，とくに時間は指定しないが，トレーニングに要した時間も計測してもらう．この間，不明な点があれば T-Web システムの開発者に質問してよい．なお，実行環境を統一するため，T-Web システムの実行には専用の実験機を用意した．また，厳密を期すため，トレーニング期間中はまだ被験者に課題を教えていない．

3. 上記トレーニングが終了した者から順次フェーズ 1 の課題を行い，所要時間を計測してもらう．ここで，T-Web システムを用いた開発と従来手法による開発の行う順序による影響を相殺するため，被験者をそれぞれ 5 人ずつの 2 つのグループに分け，グループ 1 の人には従来手法を先に，グループ 2 の人には T-Web システムを先に使用して実験を行ってもらう．なお，従来手法とは，汎用的な開発環境のうち各被験者がもっとも慣れているものを使用した開発方法である．ただし，T-Web システムが生成したコードを見ることや，個人が所有するライブラリ等を使用することは禁止する．なお，フェーズ 1 の時点ではフェーズ 2 の課題は被験者に教えていない．

4. フェーズ 1 について両手法による開発が終わった者から順次フェーズ 2 の課題を行ってもらう．T-Web システムと従来手法の順序は，フェーズ 2 のときと同じグループ分けに従う．フェーズ 2 ではクライアント側プログラムの実装も行うが，サーバ側プログラムと同様，各自のもっとも慣れた実装技術を使用してもらう．なお，課題のうちサーバ側の実装に関する課題 1 の時間と，クライアント側の実装に関する課題 2，3 の時間とを分けて計測してもらう．

5. フェーズ 2 の実験が終了した被験者には，あらかじめ配布しておいたアンケート用紙に，実験の開発時間，従来手法として使用した技術やツール，それまでの開発経験，その他自由なコメントを記入して提出してもらう．

なお，今回の被験者の技術水準は以下の通りである．

- 被験者のプログラミング経験歴は，最大が 15 年，最小が 2 年，平均で 6.2 年であった．なお，グループ 1 の平均は 5.6 年，グループ 2 の平均は 6.8 年であり，両者に著しい差異はなかった．
- Web アプリケーションの開発歴は，最大が 5 年，最小が 1 年，平均で 2.7 年であった．こちらも，グループ 1 の平均は 2.6 年，グループ 2 の平均は 2.8 年であり，両者に著しい差異はなかった．
- Web アプリケーションのサーバ側実装技術のうち最も使用経験があるものは，Java が 5 人，PHP が 2 人，ASP.NET が 2 人，Perl が 1 人であった．
- Web アプリケーションのクライアント側実装技術のうち最も使用経験があるものは，10 人全員が JavaScript であった．ただし，JavaScript の使用経験が十分にある人は 2 人であり，8 人はクライアント側の実装技術の使用経験が少なかった．
- 従来手法として使用した開発環境は，eclipse が 5 人，PHP エディタが 2 人，Visual-Studio.NET が 2 人，テキストエディタが 1 人であった．

7.2.2 実験結果

T-Web システムと従来手法を用いた場合のそれぞれの開発に要した時間を，表 7.2 に示す．実験では，設計のための時間と実装のための時間を分けてアンケートを行った．しかし，10 人中 9 人が，従来手法の設計時間の方が T-Web システムでの設計時間よりも長いか同じであり，従来手法と T-Web システムの実験順序による差異が現れなかった．また，設計時間は実装時間に比べ相対的に小さかった．これは，T-Web システムでは Web 遷移図を描く作業の中でページ遷移の設計を行っているからだと考えられる．そこで，表 7.2 では，設計時間と実装時間を合計した値を開発時間としている．フェーズ 1 では，10 人中 8 人が T-Web システムを用いたほうが開発時間が短く，残りの 2 人は両手法ともほぼ同程度の時間であった．また，フェーズ 2 では被験者全員が T-Web システムを用いた方が開発時間が短く，両者の差も大きかった．これは，Web アプリケーションの開発経験を有する人でも，クライアント側の実装技術を使用する機会があまり多くないためであると考えられる．なお，フェーズ 2 の課題 3 は，10 人中 2 人が未完成であった．

被験者	フェーズ 1 (分)		フェーズ 2 (分)		フェーズ 1+2 (分)	
	T-Web	従来手法	T-Web	従来手法	T-Web	従来手法
A	60	145	5	130	65	275
B	22	105	2	167	24	272
C	85	83	17	291	102	374
D	20	130	3	60	23	190
E	7	150	25	80	32	230
G1 平均	38.8	122.6	10.4	145.6	49.2	268.2
F	21	83	4	111	25	194
G	131	125	15	300	146	425
H	50	300	90	540 以上	140	840 以上
I	36	113	37	160	73	273
J	40	90	10	20 以上	50	110 以上
G2 平均	55.6	144.2	31.2	226.2	86.8	368.4
総平均	47.2	133.4	20.8	185.9	68	318.3

表 7.2 評価実験の開発時間一覧

表 7.3 は，従来手法の開発時間を 100 としたときの T-Web システムの開発時間の相対値である．つまり，T-Web システムを用いることによる従来手法に対する労力軽減の割合であり，数値が小さいほどその効果が大きいことを意味する．上記結果から，サーバ側の実装のみの場合で開発労力が約 41 ここで，フェーズ 2 の従来手法のサーバ側の変更のみの時間と T-Web システムの時間との比較を行う．フェーズ 2 の T-Web システムを用いた作業は，データ型定義の変更作業が中心だからである．従来手法の時間を 100 としたときのスコアの平均は 101.6 であり，若干従来手法のほうが有利であった．一方，10 人中 7 人が T-Web システムによる時間のほうが従来手法による時間よりも短かった．このように，今回は作業量が小さく作業時間も短いため，T-Web システムを用いたときの変更容易性について優位な差は得られなかった．

次に，従来手法による開発と T-Web システムによる自動生成で得られる成果物について，ファイル数と行総数の分析を行う．表 7.4 は，両手法について成果物を回収することができた 7 人分のファイル数と行総数である．フェーズ 1 とフェーズ 2 ともに，T-Web システムの生成物のほうがファイル数，行総数ともに大きい．とくに行総数の両者の差異は大きい．これは，T-Web システムの生成する Web アプリケーションがさまざまな一貫性と安全性維持のための機能を有しているからである．一方，フェーズ 1 の従来手法とフェーズ 2 の従来手法でファイル数にほとんど変化がないのは，被験者の多くが Web ページ文書の一部として直接 JavaScript のコードを記述したからである．なお，T-Web システムを用いた場合でも，被験者により生成ファイル数が異なることがわかる．これは，Web 遷移図を用いた Web ページの遷移の設計に十分な自由度があるためと考えられる．

ここで，計測した開発時間に影響を与える可能性がある要因について分析を行う．T-Web システムのトレーニング時間は平均 45 分であった．一方，平均時間以上のトレーニングを行ったが T-Web システムによる開発時間が平均時間以上かかっている被験者が 10 人中 3 人であった．よって，トレーニング時間の多寡と T-Web システムによる開発時間との相関は

被験者	フェーズ 1 (%)	フェーズ 2 (%)	フェーズ 1+2 (%)
A	41.4	3.8	23.6
B	21.0	1.2	8.8
C	102.4	5.8	27.3
D	15.4	5.0	12.1
E	4.7	31.3	13.9
G1 平均	37.0	9.4	17.1
F	25.3	3.6	12.9
G	104.8	5.0	34.4
H	16.7	16.7	16.7
I	31.9	23.1	26.7
J	44.4	50	45.5
G2 平均	44.6	19.7	27.8
総平均	40.8	14.6	22.5

表 7.3 評価実験の開発相対時間一覧

	フェーズ 1		フェーズ 2	
被験者	T-Web	従来手法	T-Web	従来手法
A	11 (923 行)	7 (200 行)	15 (1420 行)	8 (274 行)
B	7 (796 行)	6 (149 行)	10 (1086 行)	6 (195 行)
E	10 (1016 行)	8 (274 行)	13 (1216 行)	8 (428 行)
F	10 (1051 行)	4 (234 行)	12 (1203 行)	4 (295 行)
G	12 (1202 行)	9 (406 行)	16 (1467 行)	9 (590 行)
I	9 (994 行)	5 (242 行)	11 (1143 行)	5 (316 行)
J	10 (810 行)	9 (331 行)	13 (1229 行)	10 (383 行)
平均	9.9 (970 行)	6.9 (262 行)	12.9 (1252 行)	7.1 (354 行)

表 7.4 評価実験の成果物ファイル数と行総数

弱いと考えられる．これは，T-Web システムの学習コストが比較的低いことを反映している．次に，プログラミング経験の長い人上位 3 人と短い人 3 人について開発時間の比較を行うと，長い人 3 人の T-Web システムの平均は 79 分，従来手法の平均は 298 分であった．一方，短い人 3 人の T-Web システムの平均は 97 分，従来手法の平均は 441 分であった．よって，プログラミング経験の多寡は T-Web システムを用いた開発の時間にも影響を与える．しかし，従来手法で 1.5 倍の差があった開発時間が T-Web システムを使用することで大きく縮まっている．同様の傾向は Web アプリケーション開発の経験の多寡に対しても見つけられる．このように，T-Web システムは開発経験の少ない者の利用に対してとくに有効であるといえる．

7.3 比較

本節では，ダイアグラムを用いる生成手法とは異なるアノテーション方式による生成手法との比較を行う．また，ダイアグラムを用いる他の開発手法として，汎用的なソフトウェア開発手法の 1 つであるモデル駆動型開発を Web アプリケーション開発に応用する場合との比較を行う．

7.3.1 アノテーション方式

アノテーション方式による Web アプリケーションの生成とは，Web ページ文書と Web ページ間の遷移関係に着目し，静的 Web ページ文書に宣言的記述を付加することで Web アプリケーションを自動生成する手法である [2, 3]．アノテーション方式に基づく Web アプリケーション生成システムとして，A-Web システムが提案されている．静的 Web ページ文書に付加的記述を与えることで Web アプリケーションの振る舞いを定義する点で，サーバページ型 Web アプリケーションと類似する．しかし，サーバページ型 Web アプリケーションでは手続き的記述を用いるのに対し，アノテーション方式では制約式や SQL 文などの宣言的記述を用いる．アノテーション方式による Web アプリケーション開発は，一般に以下の手順に従う．

1. まず視覚的ツール等を用いて静的 Web ページ文書を作成する．Web ページ中の動的に値が埋め込まれる部分は特別な文字列を使って区別しておく．
2. アノテーションを Web ページ文書中の動的に値が変化する部分に対応付ける．アノテーションは，入力検査やセッション管理，記憶領域がもつデータの操作，外部プログラムとの通信など，Web サーバ側でのデータ処理に関する記述である．
3. 生成システムが，アノテーションが付与された Web ページ文書からサーバ側プログラムを自動生成する．

以下，提案手法が採用するダイアグラム方式とアノテーション方式の比較を行う．

対象アプリケーション

アノテーション方式では，サーバ側プログラムを生成する前に Web ページ文書を作成することから，HTML 文書等を編集可能なツールを用いて柔軟なページレイアウトを容易に

実現することができる．この点，ダイアグラム方式ではデータフローがすでに定義された生成文書に対して修正を加えることから，Web ページのレイアウト変更に関しても一定の制限がある．一方，アノテーション方式では静的部分が少なく動的部分が多い Web ページ文書の作成・編集には適切でない．ダイアグラム方式では文書としての Web ページが存在しなくても抽象的なレベルで振る舞いを把握することができる．このため，抽象的モデルからの詳細化が必要な大規模 Web サイトなどの場合，ダイアグラム方式のほうが柔軟性に優れている．

開発プロセス

アノテーション方式では，どのような Web ページが必要かを決定すると，Web フォームなどの具体的な Web ページ要素に着目する．また，2 つの Web ページ間の遷移としてデータフローをとらえる．一方，ダイアグラム方式ではデータと Web ページを抽象的概念としてとらえ，Web アプリケーション全体のデータフローに着目する．このような違いから，アノテーション方式では Web ページの具体的なレイアウト等について漸進的な開発プロセスを取ることができる．一方，ダイアグラム方式ではアプリケーション全体の振る舞いを踏まえた迅速な生成を可能とする．

維持管理

アノテーション方式でもダイアグラム方式でも，Web アプリケーション全体の一貫性を維持しつつデータ定義やナビゲーション等の変更を行うには，ツールによる支援が不可欠である．例えば，ハイパーリンクの参照先の検出や，データ入出力に関する整合性検査などがある．しかし，大規模な Web アプリケーションでは，詳細なデータフローに基づいて維持管理を行うことは作業負担が大きいことから，ダイアグラム方式による抽象レベルでの振る舞い把握が有効である．

対象利用者

アノテーション方式では，付加的記述として制約式などを与えることから，利用者はデータの整合性等に関する計算の知識を必要とする．そのような記述方式に慣れることできわめて簡潔な記述から Web アプリケーションを生成することができる．一方ダイアグラム方式では，モデルを用いて Web アプリケーションを記述することから，利用者はアプリケーション領域に関する問題を抽象的にとらえ概念化する力を必要とする．ただし，実装に関する技術的知識は必要ない．

7.3.2 UML による Web アプリケーション開発

ダイアグラムを用いた Web アプリケーションの開発支援法として，さまざまな設計言語や設計手法が提案されている．Web アプリケーションのハイパーメディアシステムとしての側面に着目し，ハイパーメディアがもつ特徴を表現するための特別なノード等を用いて Web アプリケーションのナビゲーション設計を行う方法がある．例えば，設計言語として WebML，設計手法として OOHDM などがある [7, 31]．また，UML の記述方法を採用した

設計手法も提案されている [19]．これらの方法では，ハイパーメディアアプリケーション特有のナビゲーション構造を簡潔に表現できる．一方，Web ブラウザからの入力に基づいてサーバ側のデータを更新する処理に関しては，直感的な記述が難しいという問題がある．

一方，近年ソフトウェア開発において，標準化団体 OMG (Object Management Group) が提唱するモデル駆動型開発のソフトウェアアーキテクチャMDA (Model-Driven Architecture) が注目されている．MDA は，コード主導開発からモデル主導開発へ移行することで，モデルの可搬性，相互運用性，再利用性を高めることを目的する．MDA で最も多く使用される設計言語は UML(Uniform Modeling Language) である [4]．しかし，UML は記述力の高い汎用的なソフトウェア記述言語であるため，適用ドメインごとの記述方法の指針が重要になる．この点，Web アプリケーション開発のために拡張した UML が提案されている [10]．ここでは，拡張された UML を用いて MDA に基づき Web アプリケーションを開発する場合との比較を行う．

拡張した UML では，UML のクラスや関連にステレオタイプを適用することで，Web アプリケーションの構成要素や構成要素間の関係を表現している．例えば，ステレオタイプとして，Web サーバ上に配置されるプログラムを表すクラス <<server page>>，Web クライアントが表示する Web ページを表すクラス <<client page>>，Web フォームを表すクラス <<form>> があり，client page クラスは form クラスをコンポジションとしてもつ．form クラスから server page クラスに HTTP リクエストを表す関連 <<submit>> があり，server page クラスから client page クラスに HTTP レスポンスを表す関連 <<build>> がある．また，クラス図を用いて Web アプリケーションの構成要素間の関係を定義し，シーケンス図を用いてプログラムの振る舞いを記述する．拡張した UML を用いて MDA を適用した Web アプリケーション開発は，以下の特徴を有する．

クライアントページとサーバページの分離記述 拡張した UML では，Web サーバがもつ Web ページテンプレートと Web ブラウザが表示する Web ページとを分離して記述しており，サーバページがクライアントページを生成するという構造をもつ．これにより，クライアント側の表示状態の遷移とサーバ側のセッション状態の遷移とを独立に記述することができ，複雑な Web ページ切り替えを行うクライアント側プログラムを記述することも可能である．しかし，クラス図で記述すべき関連の範囲が曖昧となりやすい．例えば，HTTP リクエストを表す関連 submit は静的構造だけでなく振る舞いの意味も含んでいる．また，クラス図とシーケンス図との間の整合性を強く意識する必要がある．一方，Web 遷移図ではクライアント側プログラムの振る舞いを独立して記述しない．そのため，クライアント側プログラムが実行できる処理は，サーバ側プログラムの振る舞いに大きく依存する．しかし，1 種類のダイアグラムのみを使用することでダイアグラムの可読性が高まり，迅速な Web アプリケーション開発を実現することができる．

オブジェクト指向開発 拡張した UML を用いた Web アプリケーション開発は，一般的な UML を用いたソフトウェア開発と同様，オブジェクト指向分析・設計との融和性が高い．例えば，form クラスは client page クラスから分離して存在するが，同様の入力フォームをもつ client page クラスがある場合には再利用が可能である．また，server page クラスは他の server page クラスと共有するサーバページの断片を参照することもできる．一方，Web 遷移図および T-Web システムでは，このような部品指向の開発は行わず，Web ページノードを下位概念の複数の部品に分割して記述することもしな

い．Web アプリケーションでは Web ページ単位でページ遷移を設計するのが最も自然であると考えるからである．また，これにより記述するノードの粒度を統一し，可読性の高いダイアグラムとすることができる．

モデルによるロジック記述 MDA では，コードによる記述を排除し，すべてをモデルにより記述することが理想とされる．ここで，実行可能なプログラムをモデルから生成するためには，モデルがプログラムと同等の情報を有している必要がある．しかし，シーケンス図等の振る舞い図によりロジックのすべてを記述することは，コードを直接記述する以上の労力を必要とする．一方，提案手法ではデータフローの全体的な流れはモデルにより記述するが，詳細なロジックはモデル外部に存在するテンプレートを用いて実現する．これにより，モデルの記述が開発者の負担となることを防止し，迅速な Web アプリケーションの生成を実現している．

第8章

おわりに

本研究ではまず，サーバページ型 Web アプリケーションの全体的振る舞いを記述するための Web 遷移図を提案し，この Web 遷移図に基づいてサーバページ型 Web アプリケーションを自動生成する手法を提案した．提案ダイアグラムは，サーバページ型 Web アプリケーションの全体的振る舞いを直感的に理解・記述することを容易にした．また，提案生成手法は，一貫性と標準的安全性を有するサーバページ型 Web アプリケーションを容易に作成することを可能にした．生成システム T-Web を試作実装し，提案手法が典型的な Web アプリケーションの開発に有用であることを確認した．

本研究では次に，サーバプログラム型 Web アプリケーションとサーバページ型 Web アプリケーションの生成手法を一般化・体系化し，サーバ側の実装アーキテクチャに非依存な生成手法を提案した．上位概念化した Web 遷移図は，ダイアグラムの理解容易性を維持しつつ，開発の初期段階から生成段階へのモデルの詳細化を円滑にすることを実現した．また，提案生成手法は，Web アプリケーションの多様な実行環境への柔軟な対応を可能にした．一般化した生成システム T-Web を試作実装し，提案手法が典型的な Web アプリケーションの開発に柔軟に対応できることを確認した．

本研究ではさらに，実装アーキテクチャに非依存な生成手法を応用し，クライアント協調型 Web アプリケーションを自動生成する手法を提案した．上位概念化した Web 遷移図に付加的記述を与えることで，サーバ集中型 Web アプリケーションとクライアント協調型 Web アプリケーションを同時に生成することを可能にした．多様な Web クライアントの処理能力を適切に活用することで，Web アプリケーションのセッション管理機能や安全性維持機能を強化し，アプリケーション全体の処理効率を向上させることができた．拡張した生成システム T-Web を試作実装し，セッション管理やデータ処理を行うクライアント側プログラムを生成できることを確認した．

また，実装した T-Web システムを使用して Web アプリケーション開発経験のある学生による評価実験を行った．提案生成手法が従来の開発手法と比べて典型的な Web アプリケーションの開発に有用であることを確認した．とくに，プログラミング等の経験が比較的少ない学生の開発時間を大幅に短縮することができた．

今後の課題として，Web 遷移図の検査が挙げられる．Web アプリケーションの生成前に Web 遷移図の静的意味検査を行うことで，Web 遷移図の記述が仕様を満たしていることを容易に確認することができ，生成される Web アプリケーションのさらなる品質向上を図ることができる．また，提案生成手法を応用して，メタデータを利用した柔軟なコンテンツ管理を実現することも今後の課題である．大規模データを Web アプリケーションで効率的に

管理するためには，データ追加時にメタデータを自動抽出することが有用である．ここで，メタデータの抽出をサーバ側で行うと，Web サーバの負荷が大きく増加する．そこで，クライアント側プログラムがメタデータの抽出を行い，その結果を Web サーバへ送信するというアーキテクチャが考えられる．このように，メタデータの抽出を行う Web アプリケーションでは，クライアント協調型 Web アプリケーションの利点を生かすことができる．さらに，Web アプリケーションへの攻撃に対するさらなる防御方法の検討も今後の課題である．

謝辞

本研究にあたり，熱心にご指導いただきました指導教官の徳田雄洋教授に深く感謝いたします．また，野呂智哉助手には常日頃から多大な協力と助言を頂きました．そして，いつも貴重な意見をいただき実験にも協力していただいた徳田研究室の皆様に深くお礼申し上げます．

参考文献

[1] ActionScript.org. Macrodedia Flash Resources and Tutorials. http://www.actionscripts.org/.

[2] Kazuhiro Asami and Takehiro Tokuda. Generation of Web Applications from Annotation-Based Definitions. *Proc. of Engineering Information Systems in the Internet Context*, pp. 69–79, 2002.

[3] Kazuhiro Asami and Takehiro Tokuda. Generation of Web Applications from HTML Page Templates with Annotations. *Proc. of the IASTED International Conference*, pp. 295–300, 2002.

[4] Grady Booch, Ivar Jacobson, James Rumbaugh, and Jim Rumbaugh. *The Unified Modeling Language User Guide*. Addison-Wesley Pub Co, 1st edition, 1998.

[5] Simon Brown, Robert Burdick, Jayson Falkner, Ben Galbraith, Rod Johnson, Larry Kim, Casey Kochmer, Thor Kristmundsson, and Sing Li. *Professional JSP*. Wrox Press Inc., 2nd edition, 2001.

[6] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture*. John Wilery & Sons, 1996.

[7] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web Modeling Languages (WebML): a modeling language for designing Web sites. *Proc. of the 9th World Wide Web Conference*, 2000.

[8] Patrick Chan and Rosanna Lee. *The Java(TM) Class Libraries, Volume 2: java.applet, java.awt, java.beans*. Addison-Wesley Pub Co, 2nd edition, 1997.

[9] David Chappell. *Understanding ActiveX and OLE*. Microsoft Press, 1996.

[10] Jim Conallen. Modeling Web Application Architectures with UML. *Communications of the ACM*, Vol. 42, No. 10, pp. 63–70, 1999.

[11] World Wide Web Consortium. Cascading Style Sheets. http://www.w3.org/Style/CSS/.

[12] World Wide Web Consortium. Extensible Stylesheet Language. http://www.w3.org/Style/XSL/.

[13] World Wide Web Consortium. HyperText Markup Language. http://www.w3.org/MarkUp/.

[14] World Wide Web Consortium. Hypertext Transfer Protocol. http://www.w3.org/Protocols/.

[15] World Wide Web Consortium. XForms. http://www.w3.org/MarkUp/Forms/.

[16] World Wide Web Consortium. XML Schema. http://www.w3.org/XML/Schema/.

[17] Bill Evjen, Scott Hanselman, Farhan Muhammad, S. Srinivasa Sivakumar, and Devin Rader. *Professional ASP.NET 2.0.* Wrox Press Inc.

[18] David Flanagan. *JavaScript: The Definitive Guide.* O'Reilly & Associates, 4th edition, 2001.

[19] Rolf Hennicker and Nora Koch. A UML-based Methodology for Hypermedia Design. *Proc. of UML 2000 Conference*, pp. 410–424, 2000.

[20] Alex Homer, Dave Sussman, and Brian Francis. *Professional Active Server Pages 3.0.* Wrox Press Inc., 1998.

[21] Jason Hunter and William Crawford. *Java Servlet Programming.* O'Reilly & Associates, 2nd edition, 2001.

[22] Microsoft Inc. Internet Explorer. http://www.microsoft.com/windows/ie/.

[23] Kornkamol Jamroendararasame, Tomohiro Matsuzaki, Tetsuya Suzuki, and Takehiro Tokuda. Generation of Secure Web Applications from Web Transition Diagrams. *Proc. of the IASTED International Symposia Applied Informatics*, 2001.

[24] Kornkamol Jamroendararasame, Tetsuya Suzuki, and Takehiro Tokuda. A Generator of Web-based Transaction Systems Using Web Transition Diagrams. 日本ソフトウェア科学会 第 17 回全国大会, 2000.

[25] Kornkamol Jamroendararasame, Tetsuya Suzuki, and Takehiro Tokuda. JSP/Servlet-Based Web Applicaiton Generator. 日本ソフトウェア科学会 第 18 回全国大会, pp. 2C–1, 2001.

[26] Kornkamol Jamroendararasame, Tetsuya Suzuki, and Tokuda Tokuda. A Diagram Approach to Automatic Generation of JSP/Servlet Web Applications. *Proc. of the 6th IASTED International Conference Software Engineering and Applications*, pp. 292–297, 2002.

[27] Adrian Kingsley-Hughes, Kathie Kingsley-Hughes, Paul Wilton, Brian Francis, Brian Matsik, Erick Nelson, Piotr Prussak, Dan Read, Carsten Thomsen, Stuart Updegrave, Antonio De Donatis, and Susanne Clark. *Vbscript Programmer's Reference.* Wrox Press Inc., 1st edition, 1999.

[28] Rasmus Lerdorf and Kevin Tatroe. *Programming PHP.* O'Reilly & Associates, 1st edition, 2002.

[29] Tomohiro Matsuzaki, Tetsuya Suzuki, and Takehiro Tokuda. A Pipe/Filter Architecture Based Software Generator PF-Web for Constructing Web Applications. *Computer Software*, pp. 293–306, 2002.

[30] Mozzila Projects. Mozilla.org. http://www.mozilla.org/.

[31] Gustavo Rossi and Daniel Schwabe. Designing Computational Hypermedia Applications. *Journal of Digital Information*, Vol. 1, No. 4, 1999.

[32] Larry Wall, Tom Christiansen, and JonOrwant. *Programming Perl*. O'Reilly & Associates, 3rd edition, 2000.

[33] 若林憲志. Web アプリケーションのセッション管理に関する研究. 東京工業大学 学士論文, 2002.

[34] 情報処理推進機構. 情報セキュリティスキルマップ. 2002.