OWL-Sを用いたWebアプリケーションの検査と生成

平成18年1月27日

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
海津 智宏

目 次

第 1 章　はじめに **1**
1.1 背景 1
1.2 研究の目的 2
1.3 本論文の構成 2

第 2 章　T-Web と OWL-S **3**
2.1 T-Web 3
2.2 OWL-S 5

第 3 章　セマンティック T-Web **10**
3.1 システムの概要 10
3.2 ST-Web での Web 遷移図 11
3.2.1 Web 遷移図抽象モデル 11
3.2.2 Web 遷移図詳細モデル 13
3.3 自動的なパラメータの受け渡し 14
3.4 意味的不整合の検査 15
3.4.1 事前条件と効果 16
3.4.2 パラメータの存在確認 18
3.5 検査に必要な計算量 18

第 4 章　実装・実験 **21**
4.1 実装 21
4.1.1 Web 遷移図エディタ 21
4.1.2 Web 遷移図の RDF 表現 23
4.1.3 OWL-S 検査 25
4.1.4 Web アプリケーション生成 26
4.2 実験 27
4.2.1 掲示板 Web アプリケーション 27
4.2.2 ショッピングカート Web アプリケーション 29

第 5 章　関連研究 **31**
5.1 従来型 T-Web 31
5.2 OWL-S プロセスモデルの LTL 検査 31
5.3 BPEL を用いた Web サービス合成 32

5.4 比較 33

第 6 章 おわりに **35**
6.1 結論 35
6.2 今後の課題 35

謝辞 **37**

付録 A Web 遷移図の RDF 表現の実例 **40**
A.1 ショッピングカート Web アプリケーションの Web 遷移図 40

付録 B ST-Web の実装 **43**
B.1 出力する CGI 43

第1章

はじめに

1.1　背景

情報社会の発達に伴い，近年では非常に多くの情報がコンピュータで処理されるようになった．コンピュータの能力向上に加えてネットワーク技術も著しく発達し，さまざまな情報がネットワーク上でやりとりされるようになっている．

ネットワークを用いるアプリケーションの多くはクライアント・サーバ型というシステムである．クライアント・サーバ型のシステムでは，サーバがクライアントからの要求に応じて情報を加工、記録、送信、提示する．実際のアプリケーションとしては，クライアントに Web ブラウザを用いる Web アプリケーションや，XML 形式で情報をやりとりする Web サービスが多く用いられている．

複数の企業やシステム間で情報を共有し，クライアントからの要求を処理するようなシステムでは，Web サービスと Web アプリケーションを組み合わせたシステムを用いることになる．バックグラウンドでの情報共有や情報の処理には Web サービスが利用され，最終的なクライアントとの対話では Web アプリケーションが利用される．この形態のシステムは多くの業務に適用でき，高い需要が期待されている．

このように Web サービスと Web アプリケーションを組み合わせたシステムでは，ビジネスロジックの実装が Web サービスとして分離されるため，Web アプリケーション部分は Web サービスを利用しないものよりも開発が容易になる．Web アプリケーションはユーザーインターフェースが重視されるため，通常のアプリケーション開発者ではなく Web デザインを専門とする開発者によって開発することが考えられる．

しかし，このような Web アプリケーションを開発するには，Web サービスを呼び出す順序や渡すべきパラメータなどの仕様をきちんと把握している必要がある．Web アプリケーションの開発時には Web サービスに関して型チェック程度の検査しか行えないため，コンパイルが成功しても実行時にエラーとなってしまう場合がある．

そのため，より知識の少ない開発者でも意味的不整合のない Web アプリケーションを容易に開発できるような開発手法が求められている．

1.2 研究の目的

Web アプリケーション開発を支援するための手法として，Web アプリケーションに特化したプログラミング言語やフレームワーク，Web アプリケーション生成系などが提案されている．特に Web アプリケーション生成系では，Web アプリケーションのふるまいを記述したダイアグラムから Web アプリケーションを生成できるため，Web アプリケーションを容易に開発することができる．

しかし，従来の Web アプリケーション生成系では Web サービスの呼び出しが表現できない場合が多く，Web サービスを呼び出すことができる場合でも Web サービスを呼び出す順序や入出力パラメータの意味に関しての支援はできなかった．

そこで，本研究では，Web サービスの仕様を OWL-S という言語で記述し，その仕様に基づいて Web アプリケーションの検査と生成を行う Web アプリケーション生成系を提案する．Web アプリケーションの設計時に検査を行うことで実行時に意図しない動作が行われる可能性を排除することができ，仕様に基づいてパラメータの受け渡しの設定を自動化することで開発者の作業を軽減できる．

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章で本研究を行うための基盤技術となる T-Web および OWL-S について述べる．第 3章で提案するセマンティック T-Web システムの構造および設計時検査の方法について述べる．第 4章で実際に実装したシステムの詳細な動作と，実装したシステムを用いて作成した Web アプリケーションの例について述べる．第 5章で関連する他の研究について述べ，提案手法との比較を行う．最後に，第 6章で結論および今後の課題について述べる．

第2章

T-WebとOWL-S

本章では，従来の Web アプリケーション開発手法である T-Web 生成系と，Web サービスの仕様を記述するための言語である OWL-S について述べる．

2.1　T-Web

T-Web[4, 5, 6, 7, 8, 9, 10, 11] は Web 遷移図と呼ばれるダイアグラムから Web アプリケーションを生成するソフトウェアである．Web 遷移図にテンプレートを適用することで，開発者がプログラムコードを記述することなく Web アプリケーションを生成できる．以下では，Web 遷移図とテンプレートの適用について述べる．

Web 遷移図

Web 遷移図は Web アプリケーションのふるまいを記述するダイアグラムである．Web アプリケーションは Web ページ間の遷移関係と，プログラムによるデータフローの制御という 2 つの側面を有するが，Web 遷移図ではこの 2 つの流れを 1 つの図に同時に記述することで，Web アプリケーション全体のふるまいの直感的な理解を容易にする．

Web 遷移図のノードには固定 Web ページ，出力 Web ページ，サーバープログラム，データベースの 4 種類があり，リンクにはページ遷移リンクとデータフローリンクの 2 種類がある．

固定 Web ページは静的な Web ページ文書で表現される Web ページであり、出力 Web ページはプログラムによって動的に出力される Web ページである。Web 遷移図では，固定 Web ページは太い線の長方形で，出力 Web ページは細い線の長方形で表される．この 2 種類の Web ページノードは，クライアントの画面で表示される Web ページを表す．

サーバープログラムノードはサーバーで行われる処理を表す．サーバーではクライアントの送信した情報を受け取り，入力値の検査，データベースの更新，データベースからの情報取得などを行い，出力 Web ページを出力する．サーバープログラ

ムノードはWeb遷移図では横長の楕円で表される．

データベースノードはサーバーに置かれるデータベースのテーブルを表す．Web遷移図では円筒形で表される．

ページ遷移リンクはWebページ間のハイパーリンクを表現する．静的Webページや固定Webページから，静的Webページへ遷移することができる．Web遷移図では矢印で表される．

データフローリンクはプログラムで制御されるデータフローを表現する．Webページ内のフォームからサーバープログラムへの入力値の受け渡し，サーバープログラムとデータベース間でのデータのやりとり，サーバープログラムからの出力Webページの出力をデータフローリンクで表現する．Web遷移図では先端に線を加えた矢印で表される．

この他にも，拡張されたWeb遷移図では，開始点，終了点，セッションオブジェクトノード．メールサーバーノードなどが用いられる場合がある．

さらに，それぞれのノードやリンクの挙動を記述するために詳細モデルと呼ばれるモデルを使用する．詳細モデルでは，各Webページノード内にフォーム要素やハイパーリンク・表示するパラメータを記述する．また，データベースノードにはそのテーブルのフィールドを記述し，データフローリンクには受け渡しされるパラメータと「成功時」「エラー発生時」といった遷移の条件を記述する．

テンプレート方式

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、プログラムコード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ことによりWebアプリケーションを自動生成する方法をいう。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プログラムコードが得られる。

T-Webシステムではあらかじめプログラムテンプレートが定義されており，Webアプリケーション開発者はWeb遷移図のプログラムノードごとにテンプレートを選択するだけで典型的なアプリケーションを生成できる．典型的なアプリケーションではデータベースの入出力が主な処理となるので，データベースへの入出力を処理するテンプレートが主に用意される．以下にテンプレートの例を示す．

ADD 1 データベーステーブルに新規レコードを追加する．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はレコードを追加しない．

ADD 2 データベーステーブルに新規レコードを追加する．ただし内容が完全に一致するレコードがある場合はレコードを追加しない．

ADD 3 データベーステーブルに新規レコードを追加する．ただし指定カラムの値が既存レコードと同一の場合は、既存レコードに新規レコードを上書きする．

SHOW 1 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．

SHOW 2 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．

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．

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．

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．

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．

SENDMAIL 電子メールを送信する．

2.2 OWL-S

OWL-S は Web サービスに対して情報を付加するための語彙体系である．OWL-S はセマンティック Web と呼ばれる技術に基づいており，セマンティック Web の文法と語彙を利用して Web サービスの仕様や連携方法を記述できる．以下では，セマンティック Web の全体的な枠組みについて述べ，セマンティック Web を構成する技術について順に述べる．

セマンティック Web

セマンティック Web[18] はコンピュータ処理可能な情報を記述・処理するための枠組みであり，次世代の Web として研究が進められている．セマンティック Web は，情報を記述するための RDF，語彙を定義するための OWL，ルール記述のための SWRL，クエリ言語の SPARQL といった複数の規格の組み合わせによって実現される．

セマンティック Web の大きな特徴は情報リソースを URI で識別すること，リソース間の関係を RDF と呼ばれる 3 つ組みで表現することの 2 点である．セマンティック Web におけるリソースとは，インターネット上でアクセス可能なものや不可能なもの全てを含む，あらゆるモノをいう．言及対象となるあらゆるものを URI で名前付けし，それらリソース間の関係を定義していくことで不特定多数の人間の記述した情報を統合し，推論することを可能とする．

RDF

Resource Description Framework(RDF)[19] はリソースに関する情報を記述するための枠組みである．RDF モデルでは「主語」「述語」「目的語」の 3 つ組みで情報を記述する．主語にはリソースを，述語にはプロパティと呼ばれる特別なリソースを，目的語にはリソースもしくはリテラルと呼ばれる値を用いる．リテラルには文字列や数

```
<rdf:RDF>
  <rdf:Description rdf:about="http://www.example.org/">
    <dc:creator>John Smith</dc:creator>
  </rdf:Description>
</rdf:RDF>
```

図 2.1 RDF/XML の例

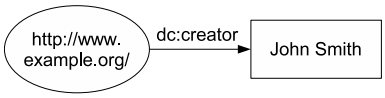


図 2.2 RDF グラフの例

値を記述できる．例えば,「http://www.example.org/ の製作者は John Smith である」という情報は，<http://www.example.org/> <http://purl.org/dc/elements/1.1/creator> ”John Smith”. という 3 つ組みで表現される．http://purl.org/dc/elements/1.1/creator はダブリン・コア [15] と呼ばれる語彙体系で定められた「製作者」を表すプロパティ，”John Smith”は文字列リテラルである．

RDF の知識を表現するための文法としては，XML を用いる RDF/XML[20] や，よりシンプルに 3 つ組みを記述する N3[17]，有向グラフによる図示などの記述法が定められている．図 2.1は RDF/XML で記述した情報，図 2.2は有向グラフで記述した情報の例である．この例では，ネームスペース宣言など一部の記述を省略している．RDF グラフでは，リソースが楕円，プロパティが矢印，リテラルが長方形で表される．リソース間やリソース・リテラル間の矢印によって，矢印の始点を主語，終点を目的語とする 3 つ組が記述される．

OWL

RDF で記述した情報を複数のシステムで共有するためには，語彙の定義が必要となる．前述の <http://www.example.org/> <http://purl.org/dc/elements/1.1/creator> ”John Smith”. という例で言えば,「http://purl.org/dc/elements/1.1/creator」という URI が「製作者」を表す，という知識が情報提供者と利用者の間で共有されている必要がある．そのため，セマンティック Web では，語彙を定義するための技術が決められている．それが Web Ontology Language(OWL)[21] である．

OWL では，概念やプロパティ間の関係や制約を記述することで，RDF で用いる語彙を定義する．概念の同一性や階層関係，プロパティの定義域，地域，反射律や対称律や推移律，逆関数との関係などが定義できる．

OWL では記述力の違いにより OWL Lite，OWL DL，OWL Full の 3 種類が定義されている．OWL Lite は記述できることが少ない反面，推論に必要な計算が少ない．OWL DL は記述論理という論理体系に基づいており，より多くのことを記述可能としながらも有限時間内に計算が終了するように定義されている．OWL Full は最大の記述力を持つが，計算可能性は保障されていない．

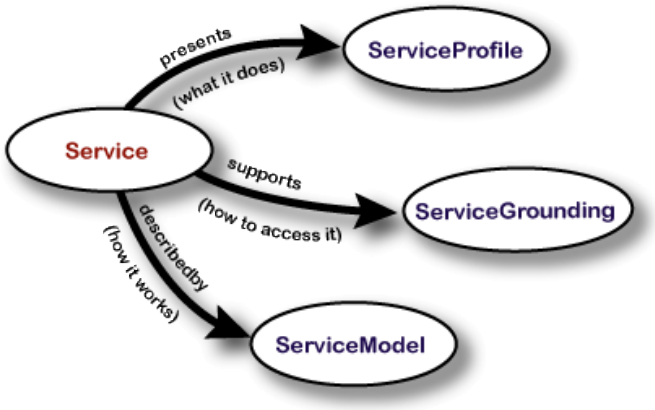


図 2.3 OWL-S トップレベルオントロジー

SWRL

Semantic Web Rule Language(SWRL)[13] はルールを定めるための言語である．「P が成り立つならば常に Q である」という形の知識が記述できる．前件部および後件部の論理式には，以下の 5 種類を任意の個数指定する．

classAtom: C(x) リソース x はクラス C のインスタンスである

individualPropertyAtom: P(x,y) リソース x とリソース y の間にプロパティ P の関係が成り立つ

datavaluedPropertyAtom: P(x,y) リソース x とリテラル y の間にプロパティ P の関係が成り立つ

sameIndividualAtom: sameAs(x,y) リソース x とリソース y が同一リソースである

differentIndividualsAtom: differentFrom(x,y) リソース x とリソース y が異なるリソースである

それぞれの x,y には URI で指定されるリソースもしくは変数が利用できる．

OWL-S

OWL-S[14] は OWL で定義された Web サービス用の語彙である．OWL-S では Web サービスに対してサービスプロファイル・サービスグラウンディング・サービスモデルの 3 種類の情報を付加する（図 2.3）．OWL-S の用語として，1 つのサイトで提供される一群の Web サービス操作をまとめてサービスと呼び，単一のリクエストで行われる操作をプロセスと呼ぶ．

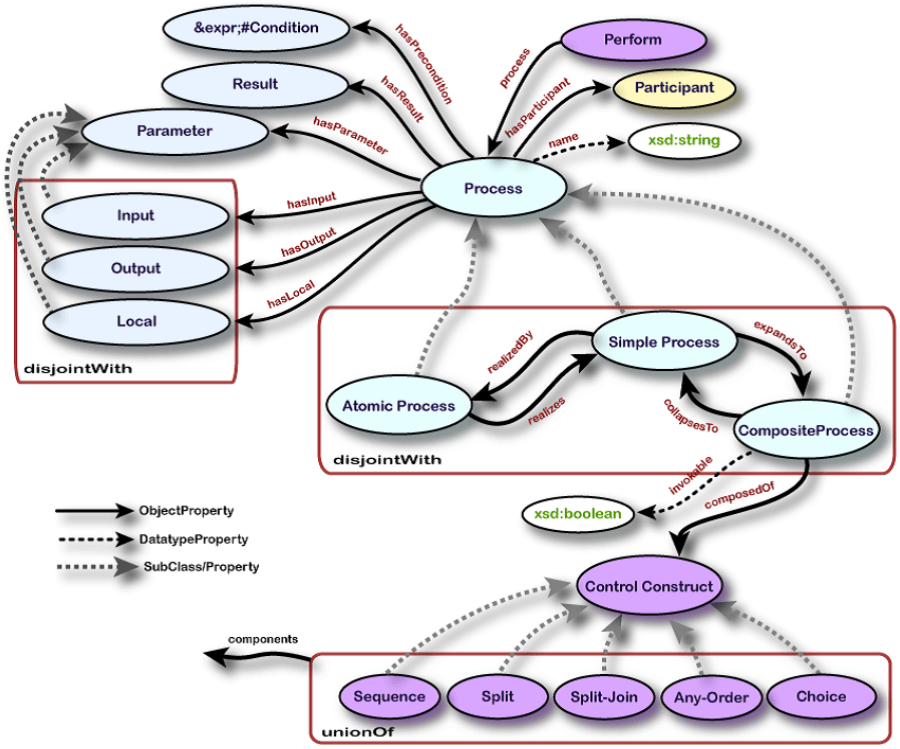


図 2.4 OWL-S プロセスオントロジー

サービスプロファイルではサービスが何をするかを記述する．この情報は主にサービスの発見に利用されることを想定している．製作者の情報，自然言語によるサマリ，サービスのカテゴリなどが記述できる．

サービスグラウンディングでは OWL-S で記述した情報と WSDL 記述 [22] とをマッピングする．OWL-S のプロセスと WSDL のオペレーションをマッピングし，OWL-S のパラメータと WSDL のパラメータをマッピングすることで，OWL-S を用いて発見・合成した Web サービスを，WSDL 記述に従って起動できるようにする．

サービスモデルではそれぞれのプロセスの詳細を記述する．プロセスは WSDL のオペレーションと対応するアトミックプロセスと複数のプロセスを連携させるコンポジットプロセスに分けられる．アトミックプロセス・コンポジットプロセスとも，入出力パラメータ，実行のための前提条件，実行後の結果を記述できる．

コンポジットプロセスでは，Sequence，Choice などのコントロールフローを用いて Web サービスの結合方法を記述する．複数のプロセスが結合されたコンポジットプロセスは，全体として粒度の大きい新しい Web サービスオペレーションとして外部に提供される．(図 2.4)

前提条件や結果は通常 SWRL の式として記述される．また，入出力パラメータは SWRL の変数として扱われる．このように OWL-S はセマンティック Web の技術を積極的に利用しており，セマンティック Web で記述される豊富な語彙を用いて Web サービスの動作を記述することができるようになっている．

Web サービスを用いる Web アプリケーションではサービス全体の実行途中で適宜ユーザーとの対話が行われるため，OWL-S が想定しているコンポジットプロセスとはふるまいが異なる．そのため，本研究では，アトミックプロセスの入力・出力・事前条件・効果のみを用いる．

第3章

セマンティックT-Web

本章では，セマンティック T-Web の構造及び設計時検査の方法について述べる．

3.1　システムの概要

本研究で提案するセマンティック T-Web(ST-Web) とは，Web 遷移図及び OWL-S メタデータを用いて Web サービスを利用する Web アプリケーションを自動生成するシステムをいう．

Web アプリケーションの自動生成のための従来手法である T-Web システムでは，Web アプリケーションが直接データベースにアクセスし，データの追加や参照等の処理を行うことを前提としていた．ST-Web では Web アプリケーションの本質的な処理が Web サービスとして分離されていることを前提とし，Web サービスを用いた Web アプリケーションを容易に開発可能にすることを目的とする．

処理を Web サービスとして分離することで，従来の T-Web では生成できなかったような高度な機能を持つ Web アプリケーションが生成可能になるほか，複数の企業やシステムの提供するサービスを組み合わせて Web アプリケーションを構成することが可能になる．利用される Web サービスはあらかじめ別途作成され，OWL-S によって適切にパラメータの意味や前提条件・効果が記述されているものとする．

提案する ST-Web は，Web 遷移図エディタ，OWL-S に基づく静的検査，Web アプリケーション生成の 3 つの機能を有する．

Web 遷移図エディタでは Web サービス呼び出しに特化させた Web 遷移図を記述する．Web サービスのパラメータの受け渡しは OWL-S の記述に従って自動的に行われるため，開発者はパラメータの受け渡しを記述する必要はない．開発者が Web 遷移図を変更するとシステムによって静的検査が行われ，問題点が検出されると警告が Web 遷移図上に表示される．この機能により，実行時に発生する可能性のある問題を設計時に把握することができ，効率のよい開発が可能となる．問題がないことを確認して生成機能を実行すると，Web アプリケーションを生成できる．ST-Web システムの全体的な構成は図 3.1のようになる．

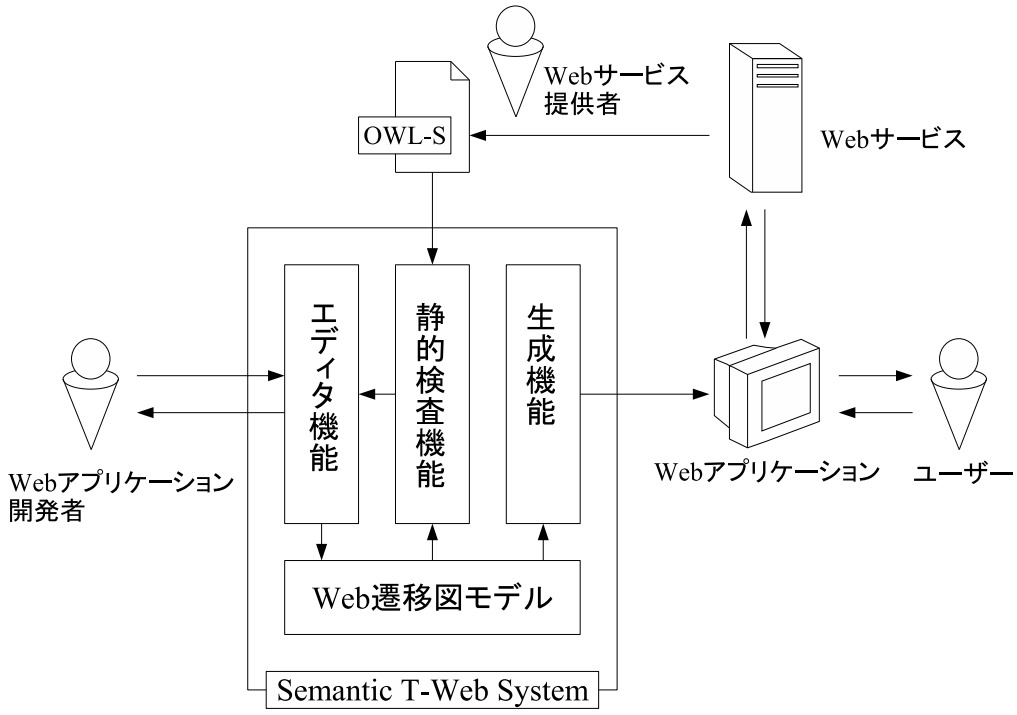


図 3.1 ST-Web システムの全体図

3.2 ST-Web での Web 遷移図

3.2.1 Web 遷移図抽象モデル

ST-Web では，Web サービスを利用する Web アプリケーションの生成に特化するため，利用する Web 遷移図に制限を加える．具体的にはサーバー側のプログラムを Web サービス呼び出しとページ出力処理に限定し，Web 遷移図のプログラムノードを Web サービス呼び出しと対応させる．

本研究では，Web 遷移図として抽象モデルと詳細モデルの 2 種類を定義する．抽象モデルは Web アプリケーション全体のふるまいを記述し，さらに詳細モデルによってそれぞれの Web ページや Web サービスの挙動を記述する．

ST-Web での Web 遷移図抽象モデルでは，要素として以下のノードとリンクを用いる．

Web ページノード 出力される Web ページを Web ページノードで表す．Web 遷移図では長方形で表現される．

Web サービスノード Web サービスの呼び出しを表す．Web 遷移図では楕円で表現される．

開始点 セッションを開始する位置を指定する．他のサイトからのリンクやブックマークからの訪問が開始点からのリンクとして表される．Web 遷移図では黒

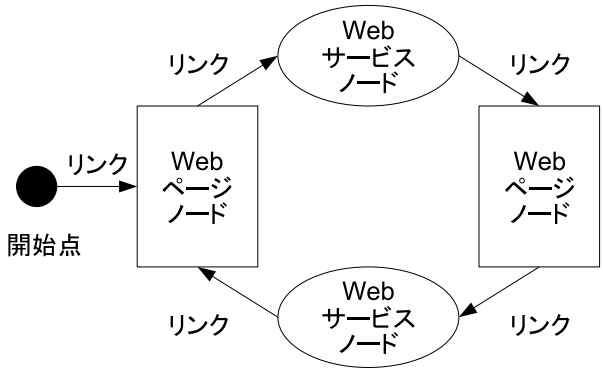


図 3.2 Web 遷移図抽象モデル

く塗りつぶされた丸で表現される．

リンク HTML のハイパーリンク及びフォーム送信時・Web ページ生成時のデータフロー，複数のサービス呼び出しの実行順序をリンクとして表現する．Web 遷移図では矢印で表現される．

図 3.2は Web 遷移図抽象モデルの例である．
この Web 遷移図のモデルは以下のように定義できる．

$$
\begin{aligned}
WTD &:= (P, E, s, L) \\
P &:= \{p1, p2, ..., p_n\} \\
E &:= \{e1, e2, ..., e_m\} \\
L &\subseteq (P \cup E \cup \{s\}) \times (P \cup E)
\end{aligned}
$$

Web 遷移図 WTD は Web ページノードの集合 P，Web サービスノードの集合 E，開始点 s，リンク L から構成される．リンクは Web ページノード，Web サービスノード，開始点のいずれかを始点とし，Web ページノードもしくは Web サービスノードを終点とする．

開始点を始点とするリンクは 1 本のみであり，リンク先のページが生成する Web アプリケーションのトップページとなる．Web サービスノードを始点とするリンクは Web サービスノードごとに 2 本であり，それぞれ Web サービスが成功した場合と失敗した場合にそのリンク先に遷移する．Web 遷移図では，失敗した場合のリンクの図示を省略できる．失敗した場合のリンクを省略した場合には，規定のエラーページがその遷移先となる．エラーページはエラーの発生を示すページであり，トップページへのリンクを持つ．Web ページノードを始点とするリンクは Web ページノードごとに 1 本以上存在する．これらは Web ページ上でハイパーリンクもしくはボタンとして表示され，複数のリンク先の中から 1 つを Web アプリケーションのユーザーが選択する．

3.2.2 Web 遷移図詳細モデル

Web アプリケーションを自動生成するためには，前述の抽象モデルだけでなく，それぞれのノードの挙動を指定する詳細モデルが必要となる．具体的には，Web ページノードで入力もしくは表示するパラメータ，Web サービスノードで起動する Web サービスといった情報を詳細モデルで記述する．

Web アプリケーションでユーザーが入力を行うパラメータは，HTML の form 要素単位でサーバーに渡される．そこで，詳細モデルでは新たにフォームノードという概念を追加する．フォームノードは出力される HTML の form 要素を表す．HTML において form 要素を利用しないハイパーリンクに関してもボタン 1 つのみのフォームと同一視してフォームノードを挿入する．つまり，抽象モデルにおいて Web ページノードを始点としていたリンクは詳細モデルでは常に Web ページノードからフォームノードへのリンク，フォームノード，フォームノードを始点とするリンクの 3 要素に置き換えられる．

Web 遷移図詳細モデルは以下のように定義される．

$$
\begin{aligned}
WTD' &:= (P', F, E', s', L_p, L_e, w) \\
P' &:= \{p1', p2', ..., p'_n\} \\
F &:= \{f1, f2, ..., f_m\} \\
E' &:= \{e1', e2', ..., e'_l\} \\
L_p &\subseteq (\{s\} \times (P' \cup E')) \cup (P' \times F) \cup (F \times (E' \cup P')) \\
L_e &\subseteq E' \times (P' \cup E') \\
w &: L_e \rightarrow \{ok, ng\} \\
p'_k &:= (name, displayvars, autosetvars) \\
f_k &:= (name, displayvars, autosetvars, inputvars) \\
e'_k &:= (name, wsdlurl, opname, inputparams, outputparams) \\
s' &:= (condition)
\end{aligned}
$$

Web 遷移図 WTD' は Web ページノードの集合 P'，フォームノードの集合 F，Web サービスノードの集合 E'，開始点 s'，決定的リンク L_p，非決定的リンク L_e，リンクの種類を表す関数 w の 7 組で構成される．

Web ページノード p'_k は $name$，$displayvars$，$autosetvars$ の 3 組で構成される．$name$ は Web ページの名前，$displayvars$ はこのページで表示されるパラメータの一覧，$autosetvars$ はこのページが表示される際に自動的に値を割り当てるパラメータと値の組の一覧である．

フォームノード f_k はこれらに 3 要素に $inputvars$ を加えた 4 組で構成される．$name$ はフォームの名前，$displayvars$ は form 要素内で表示するパラメータの一覧，$autosetvars$ はこのフォームをユーザーが選択した際に自動的に値を割り当てるパ

ラメータと値の組の一覧，$inputvars$ はユーザーが入力を行うパラメータの一覧である．

Web サービスノード e'_k は $name$, $wsdlurl$, $opname$, $inputparams$, $outputparams$ の 5 組で構成される．$name$ は Web サービスノードの名前，$wsdlurl$ は呼び出す Web サービスの WSDL 文書を参照できる URL，$opname$ は呼び出す Web サービスのオペレーションの名前，$inputparams$ は Web サービスに渡すパラメータの一覧，$outputparams$ は Web サービスの結果として受け取るパラメータの一覧である．

開始点 s' にはセッション外の状態 $condition$ を設定する．

Web サービスを始点とするリンク L_e は Web サービスの成功もしくは失敗によってリンク先が異なるため，関数 w によってリンクごとに ok もしくは ng というラベルを設定する．Web ページノード，フォームノード，開始点を始点とするリンク L_p は決定的もしくはユーザーが選択を行うものであり，このラベルは使用しない．

なお，Web 遷移図上のリンクの意味は以下の通りである．

開始点からのリンク　リンク先がトップページであることを示す．Web アプリケーションサーバーにセッションが記録されていない場合，トップページが表示される．

Web ページノードからのリンク　Web ページ内の要素としてフォームを出力することを示す．1 つの Web ページノードとそこからリンクされる任意の個数のフォームノードを合わせて 1 つの HTML 文書が出力される．

フォームノード及び Web サービスノードからのリンク　Web アプリケーションサーバーでの処理の流れを示す．フォーム内のボタンがクリックされてサーバーにリクエストが送られると，Web 遷移図のリンクに従って順に Web サービスが起動される．リンク先が Web ページノードの場合には HTML が生成され，ユーザーに返される．

3.3　自動的なパラメータの受け渡し

OWL-S では Web サービスのパラメータについての知識が記述できる．これを用いることで，同一の値を入れるべきパラメータを自動的に判断することができる．この機能により，ST-Web では Web サービスノードに対してどのパラメータを渡すかを開発者が明記する必要がなく，ST-Web によって自動的に受け渡しが行われる．具体的には，OWL-S 内で同じ RDF リソースとマッピングされたパラメータおよびマッピングされた RDF リソースどうしが OWL の sameAs プロパティで結ばれている場合，それらのパラメータを同一視する（図 3.3）．この機能により，例えばログイン Web サービスでセッション ID が返され，その後の各サービスの実行のためにそのセッション ID を渡す必要がある場合でも開発者が明示的に値の受け渡しを記述する必要がなく，効率的な開発が可能となる．

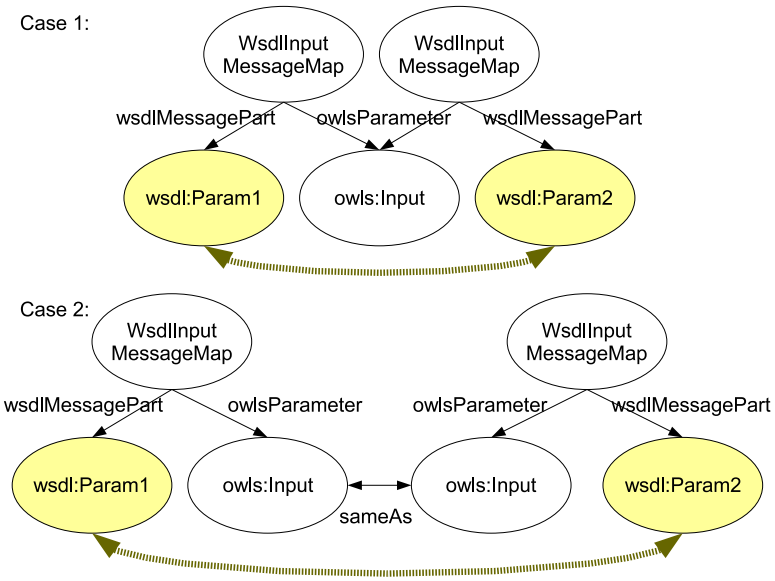


図 3.3　同一とみなされるパラメータ（入力パラメータの場合）

3.4　意味的不整合の検査

Web サービスでは，単一のリクエストに対して単一のレスポンスが返されるという 1 往復の通信で要求が処理される．しかし，Web サービスの提供すべき機能の多くは単一のリクエストのみで完結するものではなく，複数のリクエストを定められた順番で送信する必要がある場合が多い．例えば，ログインが必要な Web サービスでは，最初にログイン Web サービスを起動し，目的の処理を行う Web サービスを起動し，最後にログアウト Web サービスを起動しなければならない．このようなサービス間の依存関係は Web サービス記述言語である WSDL だけでは記述できないため，従来の開発手法においてはもしログインが必要なサービスをログインせずに起動しようとしたとしても開発時に問題を発見することができず，テスト時もしくは運用時に初めて問題が発見されるという事があった．

これは，WSDL では Web サービスの起動の仕方のみが記述され，Web サービスの処理内容に関する意味記述がなされていないため発生する問題であると考えられる．本論文では，処理を行うために必要となる前提条件を満たさないまま Web サービスを起動しようとしてしまう事を意味的不整合の発生と呼ぶ．

ST-Web では，Web アプリケーションの設計時に OWL-S を用いて意味的不整合を検出することで，Web サービスを利用する Web アプリケーションの開発を効率化する．

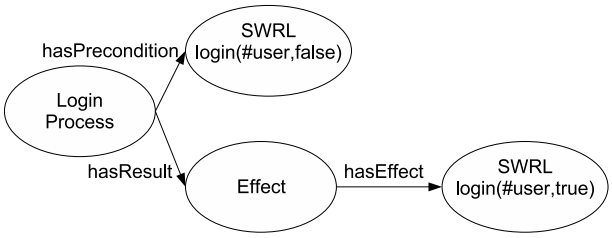


図 3.4　OWL-S を用いて記述された事前条件と効果

3.4.1　事前条件と効果

OWL-S では，Web サービスの事前条件と効果を記述することができる．事前条件と効果によって，Web サービスの処理内容が記述できる（図 3.4）．ST-Web では，OWL-S の記述を用いて，Web サービスの起動時に事前条件が満たされているかどうかを検査する．この検査のため，Web アプリケーションの実行中に Web サービスのセッション状態がどのような状態になるかを Web 遷移図と OWL-S の効果の記述に従って計算し，Web サービスごとに事前条件が成り立つかどうかを調べる．セッションの状態はサービスの実行時ににによって変化するので，ノード間にあるリンクごとに状態を計算することとする．

各リンクの状態を得るために，開始点から順にリンクをたどって状態を更新する．ここでは，各ノードを終点とするリンクをそのノードの入力リンク，始点とするリンクをそのノードの出力リンクと呼び，それぞれの状態を入力状態，出力状態と呼ぶ．Web サービスノードの出力状態は，入力状態に Web サービスの効果を加えた状態となる．

OWL-S では，Web サービスに対し複数の結果（Result）を記述できる．それぞれの結果に対し，状態変化を効果として hasEffect プロパティで記述し，出力パラメータの意味を withOutput プロパティで記述する．それらの結果の発生条件を inCondition プロパティで記述する．これに従い，ST-Web における状態変化では，各結果ごとに inCondition プロパティを入力状態と比較し，その条件が成り立つ場合には hasEffect プロパティに従って出力状態に効果を追加する．

効果として記述される式は，実行後に成り立つ状態のみが記述され，入力状態から削除すべき項目は記述されない．例えば，入力状態で「ユーザーがログアウト中」，効果が「ユーザーがログイン中」であった場合，そのまま効果を入力状態に追加すると「ユーザーはログアウト中かつログイン中」という状態が得られてしまい，期待通りの出力状態が得られない．この結果が期待と異なる理由は「ログアウト中」という状態と「ログイン中」という状態が両立しえないからだと考えられる．OWL を用いて語彙を定義しているならば，このような状態に関して語彙の定義の中で両立できないことを記述できる．ログイン中かどうかを示すプロパティが，true もしくは false のどちらか 1 つのみの値を持つと定義されていれば，その定義に従って出力状態から「ユーザーがログアウト中」という式を削除できる．

ST-Web では，OWL 記述に従い，以下の場合について出力状態から入力状態の内

容を削除する．

- functional なプロパティの値が効果で指定された場合にはそのプロパティの他の値を削除する．
- あるインスタンスがクラス C に含まれるという効果が発生した場合にはそのインスタンスがクラス C と disjoint なクラスに含まれるという記述を削除する．
- インスタンス I1 と I2 が同一，もしくは異なるとする効果が発生した場合にはそれ以前の I1 と I2 の同一性の記述を削除する．

また，Web 遷移図には通常分岐や閉路が存在するため，1 つのリンクに到達するパスが複数存在する．それらのパスごとに状態は異なる場合が多く，設計時にリンクの状態について完全な知識を得ることは困難である．そこで，ST-Web では効率的な検査を行うため，各リンクについて必ず成り立つ状態のみを考える．同一のノードが複数の入力リンクを持つ場合，それぞれのリンクの持つ状態の共通部分が必ず成り立つ状態であると考えられるので，各リンクの状態の共通部分を全体の入力状態と考えて，出力状態を考える．

Web 遷移図が閉路を含む場合には，閉路内のノードの出力状態の計算がそのノード自身の入力状態に影響を与える．そのため，各ノードの出力状態を計算する時点でそのノードの入力状態が確定できない場合がある．ST-Web では，計算しようとするノードに状態が計算されていない入力リンクがある場合でも，状態の計算済みである入力リンクのみを考えて出力状態を計算し，状態の計算を続ける．その後入力リンクの状態に変化があった場合にはもう一度状態の計算をやり直し，全体の状態が確定するまで繰り返す．つまり，以下のアルゴリズムで計算を行う．

1. 開始点からのリンクの状態を「セッション外の状態」とし，それ以外の全てのリンクの状態を unknown という状態に初期化する
2. 更新すべきノードの集合 S を考え，開始点からのリンク先を S に加える
3. S が空集合となるまで以下を繰り返す
   - 3.1. S からノード v を 1 つ選ぶ
   - 3.2. v の unknown でない入力の共通部分を取り出し，OWL-S の記述に従って効果を追加した状態を出力リンクに代入する
   - 3.3. 出力リンクの状態が計算前と変わっていれば，リンク先のノードを S に追加する
   - 3.4. S から v を削除する

操作 3.2 での出力リンクの状態の計算の例を図 3.5に載せる．この計算によって各リンクの状態を求めた後，Web サービスノードの入力状態が Web サービスの前提条件を満たしてない場合，警告を表示する．なお，開始点から到達不可能なリンク

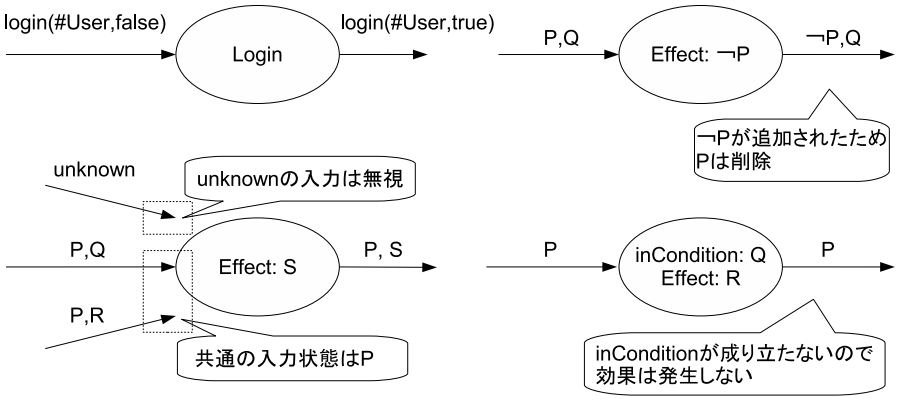


図 3.5　状態の計算の例

はこの計算が終了した後も状態が unknown のままとなる．この場合は前提条件とは無関係に，トップページから到達不可能であるという警告を表示する．

開発者はこれらの問題点を解決するように Web 遷移図を修正することで，意味的不整合の発生しない Web アプリケーションを得ることができる

3.4.2　パラメータの存在確認

ST-Web では，パラメータの自動的な受け渡しのため，同じ意味を持つ変数を同一視する機能を持つ．この機能により変数の同一視を行ったうえで，Web サービス起動時に必要なパラメータが揃っていない可能性がある場合には警告を表示することができる．このパラメータの存在確認は，事前条件・効果の検査と同時に行うことができる．フォーム内でユーザーによってパラメータの値が入力される場合や Web サービスの出力パラメータとして値が返される場合に，出力状態にそのパラメータが既知であるという記述を追加する．この記述は OWL-S の効果から得られる記述と全く同じように扱う．さらに，Web サービスの事前条件として各パラメータが既知である，という条件を追加し，未知のパラメータが存在する場合には警告を表示する．

以上のように Web 遷移図と OWL-S を用いて設計時に問題点を検出することで，Web アプリケーション開発を効率化できる．

3.5　検査に必要な計算量

3.4節で述べた Web 遷移図の検査に必要な計算量について考察する．

ここでは，全てのノード・リンクが開始点から到達可能であるような Web 遷移図を仮定する．もし到達不可能なノードやリンクが存在する場合，それらに対しては状態の計算が行われないため，ここでの考察よりも短い時間で計算が終了する．以下では，ノードの個数を N と置く．

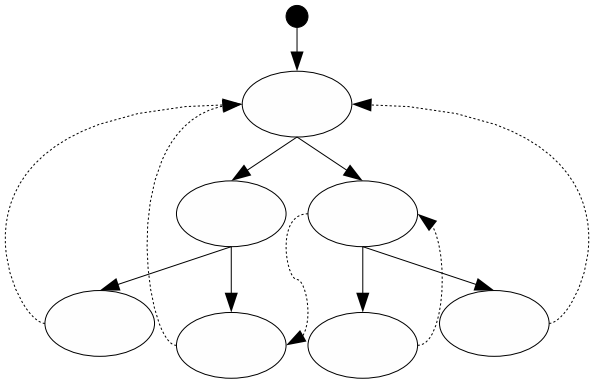


図 3.6 開始点を根とするツリー

まず，全てのノードが開始点から到達可能である Web 遷移図に対して，開始点と各ノードとをできるだけ少ない本数のリンクで結ぶことを考える．これは開始点を根とし，全てのノードを節もしくは葉として持つツリー構造として表現できる．このツリー構造のリンクは元の Web 遷移図のリンクの部分集合となる．図 3.6がツリー構造の例である．実線の矢印がツリーを構成するリンク，破線の矢印がそれ以外のリンクである．このツリー構造部分のみを Web 遷移図とみなして状態の計算を行うと，開始点から順に各ノードを 1 回ずつ処理することで全ての状態が求められる．すなわち，計算量は $O(N)$ となる．

次に，ツリー構造に含まれないリンクを考える．ツリー構造に含まれないリンクを処理した場合にはツリーのみの場合とは出力状態が異なるため，連鎖的に計算済みのノードを再計算する必要が生じる．

まずは，ツリー構造上で開始点に近いノードをリンク先として持つリンクを優先的に処理することとする．開始点に近いノードから順に状態を決めていくため，ツリー構造に含まれるリンクの状態は不変のものと考えてよい．実際のアルゴリズムでは更新すべきノード S から処理するノード v を選ぶ際に任意の 1 つを選択するが，これについては後で述べる．

開始点以外の各ノードはツリー構造に含まれる入力リンクを必ず 1 本持つため，ノード全体の入力状態は各入力リンクの状態の共通部分となる．以下では，ツリー構造に含まれる入力リンクの状態を親状態と呼ぶ．ここで，特定の式 P に着目すると，各状態では以下の 3 種類のいずれかになっていることがわかる．

- P が成り立つ.（状態に P が含まれる）
- P が成り立たない.（状態に P と両立しえない式 $\neg P$ が含まれる）
- 不明.（状態に P も $\neg P$ も含まれない）

もし親状態で P が成り立つ場合，それ以外の入力リンクが全て P を含めば全体の入力状態は P になり，不明もしくは $\neg P$ を含む入力リンクがあれば全体の入力状態は不明となる．もし親状態で P が成り立たない場合，それ以外の入力リンクが全

て $\neg P$ を含めば全体の入力状態は $\neg P$ になり，不明もしくは P を含む入力リンクがあれば全体の入力状態は不明となる．もし親状態で P に関して不明であれば，他の入力リンクの状態に関わらず全体の入力状態は不明となる．

まとめると，全体の入力状態は親状態と同一か不明となるかのどちらかであるといえる．もし全体の入力状態が親状態と同一であれば，出力状態はツリー構造に含まれないリンクを考えない場合と同一となり，再計算は必要とならない．全体の入力状態が不明の場合，新しい出力状態で状態を再計算する必要がある．ツリー構造に従って状態を再計算するには，$O(N)$ の計算時間がかかる．

OWL-S 記述における inCondition には P もしくは $\neg P$ が指定されているか，P に関する記述がないかのどれかである．P に関する記述がない場合には，入力状態に関わらず効果が発生する．従って，入力状態が変化することで効果が異なる場合というのは最初に計算した際の入力状態が P もしくは $\neg P$ で効果が発生し，計算しなおした際に入力状態が不明であって効果が発生しない場合のみとなる．つまり，計算をしなおしている間には発生する効果が元の計算よりも少なくなるため，元の計算の時に不明であった状態 Q に対して「Q が成り立つ」もしくは「$\neg Q$ が成り立たない」，といった知識が増えることはない．従って，再計算を終えた際に元のノードへ戻ってくるリンクが存在しても，さらに再計算が必要となることはなく，1 回のみの再計算でよいことがわかる．

1 つのノードに 3 本以上の入力リンクがある場合でも，一度入力状態を不明として出力状態を出力した後では再計算は起こらない．したがって，各ノードに関して最大 1 回しか再計算は必要とならない．ノードの個数が N 個，再計算に必要な計算時間が $O(N)$ であるから，全体の計算量は $O(N^2)$ である．

次に，開始点からの距離に関係なく任意のノードから処理することを考える．この場合にも，特定のノードの入力状態は再計算のたびに「不明」が増加する．しかし，式 P が成り立つことを仮定して再計算をした後に親状態で P が不明になることが考えられるので，再計算の回数は 1 回よりも増加する．最も再計算の回数が増える場合を考えると，独立する状態 $P, Q, R, ...$ のうちで 1 つずつ「不明」となり，再計算を行うことが考えられる．Web 遷移図内で使用する，独立して定められる式の種類を M 種類と置くと，ノードごとに最大 M 回の再計算が必要になり，ノードの個数が N 個，再計算に必要な計算時間が $O(N)$ であるから，全体の計算量は $O(N^2M)$ である．

以上より，ST-Web での検査に必要な計算時間はノード数を N，式の種類を M として $O(N^2M)$ であり，処理するノードの選択方法を工夫すれば $O(N^2)$ で計算が可能となる．Web 遷移図のノード数は数十程度であるから，十分に短い時間で検査を行うことが可能である．

第4章

実装・実験

本章では，実際に行ったST-Webの実装の内容と，ST-Webを用いて作成したWebアプリケーションの例について述べる．

4.1　実装

ST-WebはJava 1.5で実装した．実装したST-Webには，Web遷移図を編集する機能，OWL-S記述に従って検査を行う機能，Webアプリケーションを生成する機能を持つ．以下ではそれぞれの機能について述べる．

4.1.1　Web遷移図エディタ

ST-WebではWeb遷移図の詳細モデルを編集できる．ただし，エディタに表示される図はフォームを省略することで，抽象モデルに近い形となっている．

図4.1がST-Webの動作画面である．

左側にボタンがあり，開始点，Webページノード，Webサービスノード，リンクを選択できる．これらの要素を選択し，Web遷移図上でクリックすると，選択した要素が追加される．リンクの場合には始点から終点までをドラッグすることでリンクを作成する.「Edit」ボタンを選択中はWeb遷移図に要素は追加されない.「Edit」ボタンを選択し，各要素をドラッグすることで各要素の位置を変更することができる．以上の機能により，抽象モデルに相当する図はボタンによる要素の選択と，マウスのクリック・ドラッグのみで記述可能となっている．

Fileメニュー内にはWeb上もしくはファイル上からOWL-S文書を読み込むためのメニュー項目がある．OWL-S文書を読み込むと，読み込んだOWL-Sに記載されているWebサービスを認識し,「Web Service」ボタン下のドロップダウンリストに項目が追加される．Webサービスノードを追加する際には，あらかじめこのドロップダウンリストから追加するWebサービスを選択しておく．Webサービスノード内の名前はWebサービスのオペレーション名と同一となり，楕円の中にオペレーション名が表示される．入出力パラメータはOWL-Sの記述から取得可能なので，Web

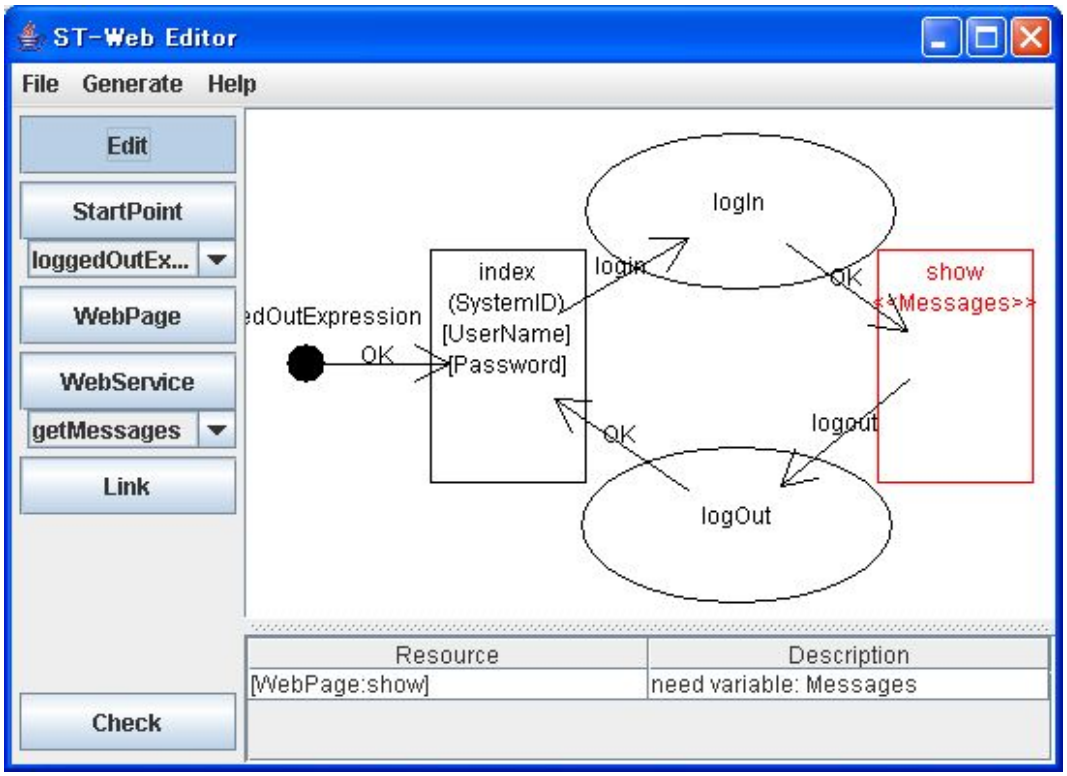


図 4.1 ST-Web システムの動作画面

サービスノードに関して開発者が行わなければならないことはサービスの選択のみである．

「Start Point」ボタン下のドロップダウンリストはセッション外で成立する条件を選ぶためのものであり，OWL-S 文書を読み込んだ際に OWL-S 文書中から利用可能な式が自動的に抽出される．開始点を追加する際にはこの中からセッション外で成り立つ項目を選択しておく．エディタ上では黒丸の上部に式の名前が表示される．

各要素を右クリックすると選択した要素に応じたポップアップメニューが表示される．

開始点もしくは Web サービスノードを右クリックすると「Delete」「Change Resource」の 2 項目が表示される．「Delete」を実行すると選択中のノードと選択中のノードを始点とするリンクが削除される．「Change Resource」を実行すると Web サービスやセッション外の状態を変更できる．Web サービスを変更する場合，前章で述べたアルゴリズムを用いて入力状態を計算し，事前条件が満たされているサービスのみを提示する．これにより，開発者は少ない選択肢の中からサービスを発見できるようになる．

Web ページノードを右クリックすると，「Delete」「Edit WebPage」の 2 項目が表示される．「Delete」を実行すると選択中のノードと選択中のノードを始点とするリンクが削除される．「Edit WebPage」を実行すると Web ページノード・フォームノードを編集するための Web ページ編集ダイアログボックスが表示される．

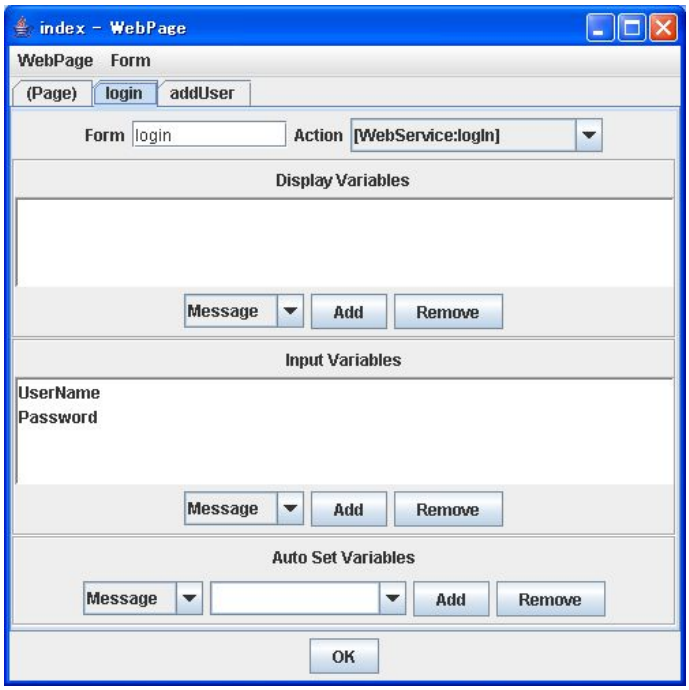


図 4.2 Web ページ編集ダイアログの動作画面

リンクを右クリックすると「Delete」のみが実行され，リンクを削除できる．

図 4.2が Web ページ編集ダイアログの動作画面である．

Web ページノードとページ内のフォームノードがタブとして表現される．Web 遷移図上で Web ページを始点とするリンクを作成すると，フォームノードは自動的に作成される．Web ページノードを編集するタブでは Web ページノードの名前，表示されるパラメータ，自動的に割り当てるパラメータが編集できる．フォームノードを編集するタブではフォームノードの名前，リンク先，表示されるパラメータ，ユーザーが入力すべきパラメータ，自動的に割り当てるパラメータが編集できる．

Web サービスノードを始点とするリンクに関して，最初に作成したリンクは自動的に「OK」となり，2 本目に作成したリンクは自動的に「NG」となる．このラベルが Web 遷移図のリンク上に表示される．

エディタ上で Web ページを始点とするリンクは詳細モデルではリンク 2 本とフォームノードとして扱われる．エディタ上ではフォームノードの名前がリンク上に表示される．

開始点を始点とするリンクには,「OK」のラベルが表示される．

4.1.2 Web 遷移図の RDF 表現

File メニュー内にはダイアログの保存と読み込みのためのメニュー項目があり，記述中の Web 遷移図を保存できる．保存形式は RDF/XML 規格に従って以下のように定義する．

Diagram クラス 保存ファイル中に 1 インスタンス記述される．Web 遷移図全体を表す．Diagram クラスを定義域とするプロパティには以下のものがある．

プロパティ	値域	意味
seeAlso	rdfs:Resource	利用している OWL-S ファイルを参照する
hasNodes	rdf:List (of Node)	Web 遷移図内のノードの一覧
hasLinks	rdf:List (of Link)	Web 遷移図内のリンクの一覧

Node クラス WebPage クラス，Form クラス，WebService クラス，StartPoint クラスの抽象親クラスとなる．

WebPage クラス Web ページノードを表す．Node クラスの子クラスである．WebPage クラスを定義域とするプロパティには以下のものがある．

プロパティ	値域	意味
hasName	xsd:string	Web ページノードの名前
positionX	xsd:integer	Web 遷移図上での X 座標
positionY	xsd:integer	Web 遷移図上での Y 座標
hasDisplayVariables	rdf:List (of Variable)	表示するパラメータの一覧
hasVariableBinding	VariableBinding	自動的に設定されるパラメータ

Form クラス フォームノードを表す．Node クラスの子クラスである．Form クラスを定義域とするプロパティには以下のものがある．

プロパティ	値域	意味
hasName	xsd:string	フォームノードの名前
positionX	xsd:integer	Web 遷移図上での X 座標
positionY	xsd:integer	Web 遷移図上での Y 座標
hasDisplayVariables	rdf:List (of Variable)	表示するパラメータの一覧
hasInputVariables	rdf:List (of Variable)	入力するパラメータの一覧
hasVariableBinding	VariableBinding	自動的に設定されるパラメータ

WebService クラス Web サービスノードを表す．Node クラスの子クラスである．WebService クラスを定義域とするプロパティには以下のものがある．

プロパティ	値域	意味
hasName	xsd:string	Web サービスノードの名前
positionX	xsd:integer	Web 遷移図上での X 座標
positionY	xsd:integer	Web 遷移図上での Y 座標
hasProcess	process:AtomicProcess	起動する OWL-S プロセス

StartPoint クラス 開始点を表す．Node クラスの子クラスである．StartPoint クラスを定義域とするプロパティには以下のものがある．

プロパティ	値域	意味
hasName	xsd:string	開始点の名前
positionX	xsd:integer	Web 遷移図上での X 座標
positionY	xsd:integer	Web 遷移図上での Y 座標
hasExpression	expr:Expression	セッション外で成立する状態

Variable クラス SWRL の Variable クラスと同一視される．ST-Web 内での変数を表す．

VariableBinding クラス Web ページノード，フォームノード通過時に自動的に割り当てるパラメータを表す．hasVariableBinding プロパティは funtional プロパティではなく，1 つのノードに複数指定することができる．変数の順序が意味を持たないため，値域を List としなかった．VariableBinding クラスを定義域とするプロパティには以下のものがある．valueSource と value はどちらか一方のみを記述する．

プロパティ	値域	意味
toVariable	Variable	割り当てる変数
valueSource	Variable	代入元となる変数
value	rdfs:Literal	代入する値

Link クラス リンクを表す．Link クラスを定義域とするプロパティには以下のものがある．hasName プロパティは Web サービスノードを始点とする場合に OK または NG を指定する．

プロパティ	値域	意味
hasName	xsd:string	リンクにつけられるラベル
from	Node	リンクの始点
to	Node	リンクの終点

Web 遷移図の RDF 表現の例を付録 A.1に載せる．

4.1.3 OWL-S 検査

3章で述べた OWL-S に基づく検査を実装した．RDF・OWL の処理には Jena Semantic Web Framework を用い，OWL-S・SWRL の処理は独自に実装を行った．ST-Web 画面上に「Check」ボタンがあり，このボタンを押すと検査が実行される．また，Web 遷移図編集時にも自動的にバックグラウンドで検査が実行されるようになっており，リアルタイムに問題点を確認することができる．

問題点が発見されると Web 遷移図エディタ上の該当する箇所が赤く表示され，画面下部の警告一覧に内容が表示される．エディタ上で警告対象の要素を選択すると警告一覧の対応するレコードも選択される．また，警告一覧中のレコードを選択す

ると，エディタ上の対応する要素も選択される．このようにエディタ画面と警告一覧が対応して動作することで，問題点を効率的に把握できる．

警告の内容には以下の種類がある．

unreachable 到達不可能なノード・リンクを警告する．検査時にリンクの状態・ノードの入力状態が unknown である場合に表示される．

no outgoing link 出力リンクの存在しないノードを警告する．

too many outgoing link 出力リンクの多すぎるノードを警告する．出力リンクの数は開始点では 1 本，Web サービスノードでは 1 本もしくは 2 本である．

need precondition 事前条件を満たさない Web サービスノードを警告する．

need variable 必要なパラメータの揃っていないノードを警告する．Web サービスの入力パラメータが未知の場合や Web ページに表示するパラメータが未知の場合に表示される．

4.1.4 Web アプリケーション生成

「Generate」メニューを実行すると，Web 遷移図に基づいた Web アプリケーションが生成される．今回の実装では，テンプレート方式は用いず，単一の Perl スクリプトと設定ファイルを出力する形式とした．

Perl スクリプトはリクエストを受け取り，Web 遷移図に従って Web サービスを実行し，Web ページを出力する．OWL-S に関する処理は行わない．なお，リクエストのクエリ処理には CGI モジュールを使用し，Web サービスの起動には SOAP::Lite モジュールを使用した．出力する Perl スクリプトを付録 B.1に載せる．

設定ファイルには，実行に必要な Web 遷移図の情報が記載される．形式は Perl からの扱いやすさを考慮してタブ区切りのテキスト形式とした．link.txt，page.txt，ws.txt の 3 ファイルが出力される．

link.txt はリンクの情報である．始点，ラベル，終点の 3 つのフィールドを持つ．ラベルは始点が Web サービスノードの場合には詳細モデルで定義される「OK」または「NG」の値である．始点が Web ページノードもしくはフォームノードの場合にはラベルの値がボタンとして表示される．始点が開始点の場合には常に「OK」となる．

page.txt は Web ページノードおよびフォームノードの情報である．ノードの名前，入力パラメータのリスト，出力パラメータのリストの 3 つのフィールドを持つ．リストの各要素はカンマで区切られ，自動的に設定される値はパラメータ名の後に等号に続けて設定する値もしくは代入元となるパラメータ名を記述する．

ws.txt は Web サービスノードの情報である．ノードの名前，起動する Web サービスの URI，入力パラメータのリスト，出力パラメータのリストの 4 つのフィールドを持つ．起動する Web サービスは WSDL の URL とオペレーション名を「#」記号で結んだものとする．リストの各要素はカンマで区切られる．

4.2 実験

ST-Web で制作できるアプリケーションの例として，掲示板およびショッピングカートの Web アプリケーションを作成した．

4.2.1 掲示板 Web アプリケーション

この例ではユーザー認証を必要とする掲示板を作成する．ユーザーはログイン後に過去の投稿の閲覧や新規投稿を行うことができる．単一の Web サービスで複数の掲示板を運営できるようにするため，各オペレーションにシステム ID というパラメータを設け，ログイン時にどの掲示板にログインするかを選択可能とした．

この例では，Web サービスは以下の 5 種類のオペレーションを提供する．

login ユーザーのログイン処理を行う．システム ID，ユーザー名，パスワードを入力とし，セッション ID を返す．ユーザーがログイン中であることを前提条件とし，ユーザーがログアウト中であることを効果とする．

logout ユーザーのログアウト処理を行う．システム ID とセッション ID を入力とする．ユーザーがログアウト中であることを前提条件とし，ユーザーがログイン中であることを効果とする．

getMessages 過去の投稿を取得する．システム ID とセッション ID を入力とし，メッセージの一覧を返す．ユーザーがログイン中であることを前提条件とする．

addMessage 新規に投稿を追加する．システム ID，投稿内容，セッション ID を入力とする．ユーザーがログイン中であることを前提条件とする．

addUser 新たにユーザーを追加する．システム ID，ユーザー名，パスワード，メールアドレスを入力とする．前提条件や効果は指定しない．

これらの Web サービスを利用して，ST-Web で Web アプリケーションを作成した．図 4.3が作成した Web 遷移図である．トップページからはユーザー名とパスワードを入力してログインができる．また，ユーザー追加ページへ遷移でき，ユーザー追加ページではユーザー名，パスワード，メールアドレスを入力してユーザーの追加ができる．ユーザーの追加が成功するとそのままログイン処理も行われるよう，addUser Web サービスノードの遷移先が logIn Web サービスノードとなっている．ログインが成功するとメッセージ一覧が取得され，表示ページでメッセージ一覧が表示される．表示ページからはメッセージの追加ページへ遷移できる．メッセージの追加後はメッセージ一覧が取得され，表示される．また，表示ページからはログアウト処理も可能となっている．

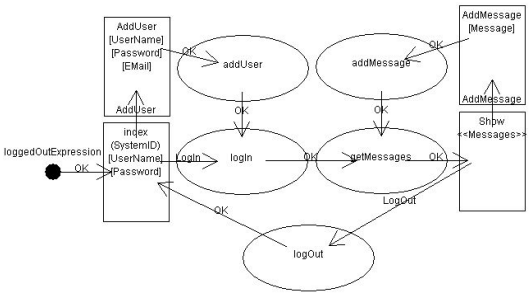
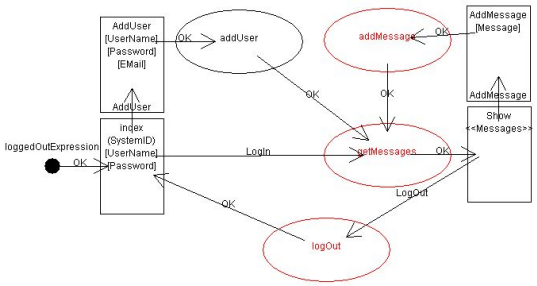


図 4.3 掲示板 Web 遷移図

図 4.4 掲示板動作画面



Resource	Description
[WebService:getMessages]	need precondition: loggedIn(TheWebAppUser,true)
[WebService:getMessages]	need variable: SessionID
[WebService:addMessage]	need precondition: loggedIn(TheWebAppUser,true)
[WebService:addMessage]	need variable: SessionID
[WebService:logOut]	need precondition: loggedIn(TheWebAppUser,true)
[WebService:logOut]	need variable: SessionID

図 4.5 掲示板 Web 遷移図での警告表示

この Web 遷移図には問題はなく，警告表示はない．Web アプリケーション生成を行うと，掲示板システムが生成される．図 4.4はメッセージ一覧ページの動作画面である．

Web サービスで使われているセッション ID に関して，この Web 遷移図では一切記述を行っていない．OWL-S の記述から自動的に ST-Web が変数の受け渡しを行い，login Web サービスで返された値をそのまま getMessages や addMessage で利用する．システム ID に関してもトップページで値をセットしている他では記述の必要がない．

もし Web アプリケーション開発者が login Web サービスの利用を忘れてしまった場合，ST-Web では図 4.5のように，getMessages，addMessage，logout の各 Web サービスが警告対象となる．警告内容は「ログイン中」という事前条件が満たされていないこととセッション ID が未知となっていることである．このように警告メッセージが表示されるので，開発者は login Web サービスを追加することで問題を解消できる．

4.2.2 ショッピングカート Web アプリケーション

この例ではショッピングカートを作成する．ユーザーは商品の一覧を閲覧し，商品をカートに追加，削除，個数変更を行うことができる．

この例では，Web サービスは以下の 9 種類のオペレーションを提供する．

Shop_getGoodsList 商品の一覧を取得する．入力パラメータはなく，商品の一覧を出力とする．前提条件や効果は指定しない．

Shop_createCart ショッピングカートを作成する．入力パラメータはなく，カート ID を出力とする．カートを持っていないことを前提条件とし，カートを持っていることを効果とする．

Shop_add ショッピングカートに商品を追加する．カート ID，商品 ID，個数を入力とする．カートを持っていることを前提条件とする．

Shop_change ショッピングカートの中の商品の個数を変更する．カート ID，商品 ID，個数を入力とする．カートを持っていることを前提条件とする．

Shop_remove ショッピングカートから商品を削除する．カート ID，商品 ID を入力とする．カートを持っていることを前提条件とする．

Shop_getCart ショッピングカートの中身を取得する．カート ID を入力とし，カートの中身の一覧を出力とする．カートを持っていることを前提条件とする．

Shop_getCount ショッピングカートの中の特定の商品の個数を取得する．カート ID，商品 ID を入力とし，個数を返す．カートを持っていることを前提条件とする．

Shop_destroyCart ショッピングカートを破棄する．カート ID を入力とする．カートを持っていることを前提条件とし，カートを持っていないことを効果とする．

Shop_buy ショッピングカートの中身を購入する．カート ID，名前，住所，電話番号，カード番号を入力とし，購入 ID を返す．カートを持っていることを前提条件とし，カートを持っていないことを効果とする．

これらの Web サービスを利用して，ST-Web で Web アプリケーションを作成した．図 4.6が作成した Web 遷移図である．トップページからこの Web ショップに入ると，カートを作成して商品一覧を表示する．商品一覧ページからはカートの表示とカートへの追加が可能である．カートへ商品を追加した場合もカート表示ページへ遷移する．カート表示ページからは商品一覧ページの表示，カートの中の商品の

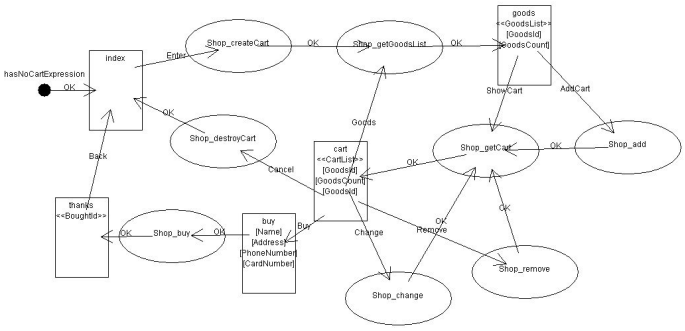


図 4.6 ショッピングカート Web 遷移図

図 4.7 ショッピングカート動作画面

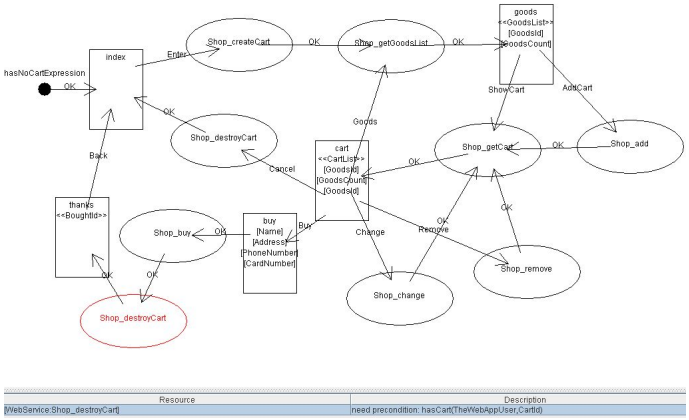


図 4.8 ショッピングカート Web 遷移図での警告表示

個数変更，削除，カートの破棄，購入ページへの遷移が可能となっている．カートを破棄するとトップページへ戻る．購入ページでは名前・住所・電話番号・カード番号を入力し，Shop_buy Web サービスを呼び出し，購入 ID を表示したあとトップページへ戻る．図 4.7はカート表示ページの動作画面である．

この例でも，カート ID に関しては Web 遷移図では記述する必要がない．また，カートを持っていることを事前条件とする Web サービスをカートを持っていない状態で起動しようとすると警告が表示される．例えば，この Web サービスでは商品購入時に自動的にカートが破棄されるが，それを知らずに Web 遷移図内で購入の後にカートの破棄をしようとすると図 4.8のように警告メッセージが表示される．

掲示板 Web アプリケーション，ショッピングカート Web アプリケーションのどちらの場合でも，実験環境（Pentium4 3.06GHz，1.00GB RAM）において検査は 1〜2 秒程度で終了し，開発者の作業を妨げることはなかった．これらの例を作成することにより，OWL-S を用いてパラメータの受け渡しの自動化や事前条件の検査を行うことで Web アプリケーション開発を支援できることが確認された．

第5章

関連研究

本章では，ST-Web による Web アプリケーション生成と関連研究との比較を行う．

5.1　従来型 T-Web

従来型 T-Web では，Web 遷移図から Web アプリケーション生成を行うことができる．ST-Web は従来型 T-Web を参考に作られており，類似点も多い．

ST-Web と従来型 T-Web では，対象とする Web アプリケーションに大きな違いがある．従来型 T-Web では，データベースの入出力を行う Web アプリケーションに特化しており，入力される値をそのまま入出力する形式の Web アプリケーションしか開発できなかった．ST-Web では，ロジック部分を Web サービスとして切り離すことで，複雑な計算を行ったり複数のシステムで通信を行ったりすることができるようになった．従来型 T-Web でも Web サービスノードを記述できるようにする提案はあったが，変数の受け渡しなどを開発者が設定する必要があった．ST-Web では，OWL-S で記述されたメタデータを利用することで変数の受け渡しを自動化し，事前条件をあらかじめ検査できるようになった．

しかし，ST-Web では Web 遷移図を Web サービスに特化した仕様としたため，Web サービスが別途開発されていなければ Web アプリケーションが開発できない仕様となっている．Web サービスが存在し，OWL-S メタデータとともに配布されているような状況であれば効率的な Web アプリケーション開発が可能だが，Web アプリケーションが存在しない場合には ST-Web は利用できない．

将来的には，ST-Web の機能が従来型 T-Web に組み込まれる形で Web サービスの利用とデータベースの利用がともにでき，OWL-S がある場合には Web サービスの事前条件や効果が利用できるようなシステムが作られるのが理想的であるといえる．

5.2　OWL-S プロセスモデルの LTL 検査

OWL-S を使って Web サービス合成の検査を行う研究として，Ankolekar らの研究 [1] がある．この研究では OWL-S のコンポジットプロセスを検査対象とし，線形

時相論理式（LTL）の検査を行う．Web サービスの合成結果を OWL-S のコンポジットプロセスで記述し，検査したい内容を LTL で記述することにより，合成時の問題を検出することができる．

この研究では SPIN[3] というモデル検査器を用いて検査を行う．SPIN では Promela という言語で記述されたモデルが LTL 式を満たすかどうかをチェックすることができる．そこで，OWL-S コンポジットプロセスを Promela 言語に変換する変換器が提案されており，変換された Promela 言語と LTL を SPIN に渡し，LTL 式が満たされているかどうかを検査する．

この研究は OWL-S を用いて検査を行うという点では ST-Web と同一だが，検査対象，検査条件ともに ST-Web とは異なっており，相補的な関係となっている．ST-Web では検査対象は Web 遷移図であり，Web サービスの単純な合成ではなく，途中にユーザーからの入力を含み，状態遷移もコンポジットプロセスよりも複雑となる．また，検査項目は ST-Web では Web サービスの事前条件，入力パラメータから自動的に決められる．検査式に LTL を使う方式では検査を行う人が自由に検査項目を決められる反面，LTL に関する知識が必要となる．また，LTL 式は状態遷移モデルを前提とし，Web サービスの合成結果を念頭に置かなければ作成できないが，事前条件や効果は Web サービスごとに考えることができるため，記述がより容易であると考えられる．

以上より，Web アプリケーション開発を容易にするという目的では ST-Web の方式に価値があるものと考えられる．ただし，より検査項目を増やしたい場合には Web 遷移図に関する LTL 検査を導入する事も有用であると考えられる．

5.3 BPEL を用いた Web サービス合成

Web サービスの合成という観点では，OWL-S ではなく BPEL(Web Services Business Process Execution Language)[16] という規格を用いた Web サービス合成が用いられることが多い．BPEL は OASIS という標準化団体によって標準化作業の行われているワークフロー記述言語である．BPEL では，OWL-S コンポジットプロセスと同様に，複数の Web サービスの合成方法を記述し，合成された結果を新たな Web サービスとして公開することができる．BPEL で記述されたビジネスプロセスはビジネスプロセス・エンジンによって実行する．

BPEL では Web アプリケーションの生成は考えられていないが，Web ブラウザをパートナーの一種としてとらえることにより BPEL プロセスを Web アプリケーションとして実行することができる [12]．この場合，Web サービス提供者が Web サービスの使用法として BPEL 記述を提供していれば，BPEL 記述から容易に Web アプリケーションを生成できる．BPEL では事前条件の検査はできないが，Web サービスの起動順や変数の受け渡しを記述できる．ただし，OWL-S を用いる場合と同様，Web サービス提供者によって情報が提供されていることを前提とする．ST-Web では実際に Web アプリケーションの中でどのように Web サービスを起動するかは Web アプリケーション開発者にまかされており，合成結果の検査のみを行う．BPEL

を用いる場合には，BPEL によって提供されている合成方法がそのまま使われる．従って，BPEL のほうが開発が容易であり，ST-Web のほうが記述力が高いといえる．ST-Web 内で OWL-S コンポジットモデルの処理を可能とすることで，OWL-S を用いても BPEL と同等に開発を容易にすることが可能であると考えられる．

BPEL のビジネスプロセスを Promela に変換して，SPIN による LTL 式の検査を行う手法も提案されている [2]．この場合，OWL-S コンポジットモデルの LTL 検査とほぼ同じ検査が可能となる．

5.4 比較

本章で挙げた各方式を用い，Web サービスを利用した Web アプリケーションを開発する際の特徴を比較する．

	ST-Web	従来型 T-Web	OWL-S LTL 検査	BPEL LTL 検査
モデル	Web 遷移図	Web 遷移図	OWL-S	BPEL
Web アプリ生成	○	○	△	△
検査	○	×	○	○
- 検査項目	OWL-S	-	LTL	LTL
- 検査項目の自動決定	○	-	×	×
- 検査項目の変更	×	-	○	○
- 検査器	ST-Web	-	SPIN	SPIN
合成方法の自動決定	×	×	○	○
合成方法の変更	○	○	×	×
変数の自動的な受け渡し	○	×	○	○

この比較表では，OWL-S や BPEL は Web サービス提供者によってあらかじめ提供されているものとする．Web 遷移図と LTL 式は Web アプリケーション開発者が作成する．

「モデル」はそれぞれの方式で Web サービスの連携方法を記述する言語である．ST-Web および T-Web では Web 遷移図を用いるが，LTL 検査の対象はでは OWL-S コンポジットプロセスや BPEL ビジネスプロセスを用いる．「Web アプリ生成」は各モデルから Web アプリケーションを生成できるかを示す．Web 遷移図は Web アプリケーション生成のためのモデルであり問題なく Web アプリケーションを生成できるが，OWL-S コンポジットプロセス及び BPEL ビジネスプロセスは Web サービス生成のためのモデルであり，Web アプリケーション生成のための変換が必要となり，元のプロセスの持つ意味が完全に保存されるわけではない．「検査」はそれぞれの方式で合成結果の検査を行う手段を提供するかを示す．「合成方法の自動決定」は Web サービスの合成が自動的に行われるかどうかを示す．「合成方法の変更」は Web サービスの合成方法を Web アプリケーション開発者が変更できるかどうかを示す．「変数の自動的な受け渡し」は複数の Web サービス間でのパラメータの受け渡しが自動的

に行えるかどうかを示す．

表からわかるとおり，Web サービスを利用する Web アプリケーションの開発手法として，ST-Web は従来型 T-Web よりも改善されていることがわかる．LTL 検査を用いる方式と比べた場合，それぞれに得意分野が異なる．検査項目や合成方法が自動的に決定されると開発は容易になるが，柔軟性が失われる．現状では二者択一となっているが，今後の課題として，標準的には検査項目や合成方法が自動的に決定され，必要に応じて変更もできるシステムが求められるのではないかと思われる．

第6章

おわりに

6.1　結論

本論文では，Web サービスを利用する Web アプリケーションの開発を容易にするため，Web 遷移図と OWL-S を用いた Web アプリケーションの検査と生成を行う ST-Web システムを提案した．

OWL-S 記述を用いて Web サービス間での変数の受け渡しを自動化し，事前条件を設計段階で検査することにより，Web サービスを利用する Web アプリケーションを従来手法よりも容易に開発できるようになった．掲示板 Web サービスやショッピングカート Web アプリケーションを ST-Web で生成することにより，ST-Web の有用性が確かめられた．

6.2　今後の課題

今後の課題を以下に挙げる．

XML パラメータの処理 ST-Web ではあまり変数の型を意識していないが，Web サービスに用いられるパラメータは XML であり，Web サービスを用いる Web アプリケーションは XML を正しく処理できる必要がある．本研究で出力される Web アプリケーションは XML を出力できるが，XML を入力したり，XML の一部分を取り出したり編集したり，XML の要素ごとに繰り返し処理行ったりする機能を持たない．これらの機能をきちんと実装する必要がある．

複数の状態の重ね合わせ 本研究ではリンクごとに 1 つの状態しか考えていないため，検査が完全とはいえない．1 つのリンクに対してものリンクへ至る複数のパスを考え，複数の状態を持つことを考慮できるようにすべきであると思われる．現状のアルゴリズムでリンクに複数の状態を持たせると状態数が非常に多くなってしまい現実的ではないが，モデル検査の技術を取り入れることで高速化できる可能性がある．

セッションと無関係な事前条件・効果の考慮 本研究では事前条件や効果の状態がWebサービスのセッションごとに定まることを仮定しているが，OWL-Sの仕様上はセッションと対応しない一般的な記述も可能である．ST-Webでは，このような記述があると意味的不整合のない箇所でもエラーとなってしまう場合があるため，改善の必要がある．

動的な事前条件の検査 本研究での検査は設計時に行う静的検査を考えており，Webアプリケーション生成後の動的検査は行わない．しかし，実行時にも検査を行い，実行不可能なWebサービスへのリンクを自動的に非表示にすることで，Web遷移図の記述を減らすことができる．現状では場合に応じてWebページノードを増やさなければならなかったものが，単一のWebページノードで兼用できるようになり，Web遷移図をより容易に記述できるようになる．

関連研究と比べ劣っている機能の追加 関連研究と比較すると，任意のLTL式を検査する機能や，OWL-Sコンポジットプロセスや BPELビジネスロジックを用いてWebサービスの合成方法を自動的に決定する機能が不足している．これらの機能を追加することは有用であると考えられる．

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，鈴木徹也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] Anupriya Ankolekar, Massimo Paolucci, and Katia Sycara. Towards a Formal Verification of OWL-S Process Models. *Proc. of the 4th International Semantic Web Conference*, pp. 37-51, 2005.

[2] Xiang Fu, Tevfik Bultan, Jianwen Su. Analysis of Interacting BPEL Web Services. *Proc. of the WWW Conference 2004*, pp. 621-630, 2004.

[3] Gerard J. Holzmann. The Model Checker SPIN. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 1997.

[4] K.Jamroendararasame, T.Suzuki, T.Tokuda. A Generator of Web-based Transaction Systems Using Web Transition Diagrams. *Proc. of the 17th Japan Society for Software Science and Technology*, 2000.

[5] K.Jamroendararasame, T.Matsuzaki, T.Suzuki, T.Tokuda. Generation of Secure Web Applications from Web Transition Diagrams. *Proc. of the IASTED International Symposia Applied Informatics*, pp. 496–501, 2001.

[6] K.Jamroendararasame, T.Matsuzaki, T.Suzuki, T.Tokuda. Two Generators of Secure Web-Based Transacion Systems. *Proc. of the 11th European-Japanese Conference on Information Modelling and Knowledge Bases*, pp. 348–362, 2001.

[7] K.Jamroendararasame, T.Suzuki, T.Tokuda. JSP/Servlet-Based Web Application Generator. *18th Conference Proceedings Japan Society for Software Science and Technology*, 2001.

[8] K.Jamroendararasame, T.Suzuki, T.Tokuda. A Visual Approach to Development of Web Services Providers/Requestors. *Proc. of the 2003 IEEE Symposium on Visual and Multimedia Software Engineering*, pp. 251–253, 2003.

[9] M.Taguchi, T.Susuki, T.Tokuda. Generation of Server Page Type Web Applications from Diagrams. *Proc. of the 12th Conference on Information Modelling and Knowledge Bases*, pp. 117–130, 2002.

[10] M.Taguchi, T.Suzuki, T.Tokuda. A Visual Approach for Generating Server Page Type Web Applications Based on Template Method. *Proc. of the 2003*

IEEE Symposium on Visual and Multimedia Software Engineering, pp. 248-250, 2003.

[11] M.Taguchi, K.Jamroendararasame, K.Asami, T.Tokuda. Comparison of Two Approaches for Automatic Construction of Web Applications: Annotation Approach and Diagram Approach. *Proc. of the 4th International Conference on Web Engineering*, pp. 230-243, 2004.

[12] 永井 隆. 副作用のある Web サービスを用いた Web アプリケーションの構成法. 2005.

[13] DAML Services. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. `http://www.daml.org/2003/11/swrl/`, 2003.

[14] DAML Services. OWL-S 1.1. `http://www.daml.org/services/owl-s/1.1/overview/`, 2004.

[15] Dublin Core Metadata Initiative. DCMI Metadata Terms. `http://dublincore.org/documents/dcmi-terms/`, 2005.

[16] IBM. Business Process Execution Language for Web Services version 1.1. `http://www-128.ibm.com/developerworks/library/specification/ws-bpel/`, 2005.

[17] Tim Berners-Lee. Notation 3. `http://www.w3.org/DesignIssues/Notation3.html`, 2001.

[18] World Wide Web Consortium. Semantic Web. `http://www.w3.org/2001/sw/`.

[19] World Wide Web Consortium. RDF Primer. `http://www.w3.org/TR/rdf-primer/`, 2004.

[20] World Wide Web Consortium. RDF/XML Syntax Specification. `http://www.w3.org/TR/rdf-syntax-grammar/`, 2004.

[21] World Wide Web Consortium. OWL Web Ontology Language Semantics and Abstract Syntax. `http://www.w3.org/TR/owl-semantics/`, 2004.

[22] World Wide Web Consortium. Web Services Description Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`, 2001.

付録 A

Web 遷移図の RDF 表現の実例

A.1　ショッピングカート Web アプリケーションの Web 遷移図

shop.stweb.xml

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#">
<stweb:Diagram xmlns:stweb="http://www.asukaze.net/ttlab/stw
eb/diagram/">
<stweb:seeAlso rdf:resource="http://localhost/shop/wsshop.ow
l.xml" />
<stweb:hasNodes rdf:parseType="Collection">
  <stweb:WebPage rdf:ID="Pindex_0">
    <stweb:hasName>index</stweb:hasName>
    <stweb:positionX>126</stweb:positionX>
    <stweb:positionY>145</stweb:positionY>
  </stweb:WebPage>
  <stweb:WebPage rdf:ID="Pgoods_1">
    <stweb:hasName>goods</stweb:hasName>
    <stweb:positionX>646</stweb:positionX>
    <stweb:positionY>73</stweb:positionY>
    <stweb:hasDisplayVariables rdf:parseType="Collection">
      <stweb:Variable rdf:about="http://localhost/shop/wssho
p.owl.xml#GoodsList" />
    </stweb:hasDisplayVariables>
  </stweb:WebPage>
  <stweb:WebPage rdf:ID="Pcart_2">
```

```
    <stweb:hasName>cart</stweb:hasName>
    <stweb:positionX>426</stweb:positionX>
    <stweb:positionY>232</stweb:positionY>
    <stweb:hasDisplayVariables rdf:parseType="Collection">
      <stweb:Variable rdf:about="http://localhost/shop/wssho
p.owl.xml#CartList" />
    </stweb:hasDisplayVariables>
  </stweb:WebPage>
  <stweb:StartPoint rdf:ID="S">
    <stweb:hasName>hasNoCartExpression</stweb:hasName>
    <stweb:positionX>18</stweb:positionX>
    <stweb:positionY>140</stweb:positionY>
    <stweb:hasExpression rdf:resource="http://localhost/shop
/wsshop.owl.xml#hasNoCartExpression" />
  </stweb:StartPoint>

  (中略)

  <stweb:Form rdf:ID="FOK_23">
    <stweb:hasName>OK</stweb:hasName>
    <stweb:positionX>401</stweb:positionX>
    <stweb:positionY>363</stweb:positionY>
    <stweb:hasInputVariables rdf:parseType="Collection">
      <stweb:Variable rdf:about="http://localhost/shop/wssho
p.owl.xml#Name" />
      <stweb:Variable rdf:about="http://localhost/shop/wssho
p.owl.xml#Address" />
      <stweb:Variable rdf:about="http://localhost/shop/wssho
p.owl.xml#PhoneNumber" />
      <stweb:Variable rdf:about="http://localhost/shop/wssho
p.owl.xml#CardNumber" />
    </stweb:hasInputVariables>
  </stweb:Form>
</stweb:hasNodes>
<stweb:hasLinks rdf:parseType="Collection">
  <stweb:Link>
    <stweb:from rdf:resource="#S" />
    <stweb:to rdf:resource="#Pindex_0" />
  </stweb:Link>
  <stweb:Link>
```

```
    <stweb:from rdf:resource="#Pindex_0" />
    <stweb:to rdf:resource="#FEnter_13" />
  </stweb:Link>
  <stweb:Link>
    <stweb:from rdf:resource="#FEnter_13" />
    <stweb:to rdf:resource="#XShop_createCart_6" />
  </stweb:Link>

  (中略)

  <stweb:Link>
    <stweb:from rdf:resource="#FOK_23" />
    <stweb:to rdf:resource="#XShop_buy_8" />
  </stweb:Link>
</stweb:hasLinks>
</stweb:Diagram>
</rdf:RDF>
```

付録B

ST-Webの実装

B.1　出力するCGI

index.cgi

```
#!/Perl/bin/Perl.exe -w
use CGI;
use Error qw(:try);
use SOAP::Lite;
use strict;

print "content-type: text/html\n\n";

my $q = new CGI;
my %links = ();
my %pages = ();
my %services = ();
my %params = ();

# 設定読み込み
&readconf(\%links, \%pages, \%services);

# GET/POSTパラメータ取得
foreach ($q->param) {
  if($_ !~ /^_/){
    $params{$_} = $q->param($_)
  }
}

# 現在地の取得
```

```
my $node = $q->param('__next');
if(!$node){
  $node = 'S';
}

# ページ遷移
while($node !~ /^P/){
  if($node =~ /^X/){
    # サービス起動
    my $svc = $services{$node};
    my @svcparams = ();
    my $wsdl = $svc->{'wsdl'};
    my $op = $svc->{'op'};
    my $in = $svc->{'in'};
    my $out = $svc->{'out'};
    my $okng = "OK";
    try{
      foreach (@$in) {
        if(exists($params{$_})){
          @svcparams = (@svcparams, $params{$_});
        }else{
          $okng = "NG";
        }
      }
      my $result;
      if($okng eq "OK"){
        $result = SOAP::Lite -> service($wsdl) -> $op (@svcp
arams);
        if($out){
          $params{$out} = $result;
        }
      }
    }catch Error with {
      $okng = "NG";
    };
    $node = $links{$node}{$okng};
    if(!$node){
      if($okng eq "OK") { $node = 'S'; }
      else { $node = 'Perror'; }
    }
```

```
  }else{
    $node = $links{$node}{"OK"};
    if(!$node){ $node = 'P'; }
  }
}


# ページ生成
my $output = "";
my $pagein = $pages{$node}{'in'};
foreach (@$pagein){
  if(/^(.*?)="(.*)"$/){
    $params{$1} = $2;
  }elsif(/^(.*?)=(.*)$/){
    $params{$1} = $params{$2};
  }
}
my $pageout = $pages{$node}{'out'};
foreach (@$pageout){
  $output .= &outputvar("", $_, $params{$_});
}
$output .= "<hr />\n";

my $nexts = $links{$node};
my %nexts = %$nexts;
foreach my $linkname (keys(%nexts)){
  my $formnode = $nexts{$linkname};

  $output .= qq(<form action="index.cgi" method="post">\n);
  $output .= qq(<input type="hidden" name="__next" value="$l
inks{$formnode}{$linkname}" />\n);
  my $pagein = $pages{$formnode}{'in'};
  my $pageout = $pages{$formnode}{'out'};

  # 内部保持データの埋め込み
  foreach (sort(keys(%params))) {
    if(! &pageinexists($_, @$pagein)){
      $output .= qq(<input type="hidden" name="$_" value="$p
arams{$_}" />\n);
    }
```

```
  }

  # 出力情報
  foreach (@$pageout){
    $output .= &outputvar("", $_, $params{$_});
  }

  # 入力情報
  foreach (@$pagein){
    if(/^(.*?)="(.*)"$/){
      $output .= qq(<input type="hidden" name="$1" value="$2
" />\n);
    }elsif(/^(.*?)=(.*)$/){
      $output .= qq(<input type="hidden" name="$1" value="$p
arams{$2}" />\n);
    }else{
      my $name = $_;
      if($name =~ /^(.*)_\d+$/){
        $name = $1;
      }
      $output .= qq($name : <input type="text" name="$_" val
ue="$params{$_}" /><br />\n);
    }
  }
  $output .= qq(<input type="submit" value="  $linkname  " /
>\n);
  $output .= qq(</form>\n);
  $output .= "<hr />\n";
}

# ページ出力
my $skinhtml = "skin/skin.html";
$node =~ /^!?(.*)$/;
if(-e "skin/$1.html"){
  $skinhtml = "skin/$1.html";
}
my $nodename = $node;
if($node =~ /^P(.+)_\d+$/){
  $nodename = $1;
}
```

```
open IN, $skinhtml or throw Error::IO;
while(<IN>){
  s|<stweb:title.*?/>|$nodename|g;
  s|<stweb:form.*?/>|$output|g;
  print;
}
close IN;

# input 配列中に存在するか
sub pageinexists {
  my ($var, @arr) = @_;
  foreach (@arr){
    $_ =~ s/(=.*$)//;
    if($_ eq $var){
      return 1;
    }
  }
  return 0;
}

# どんな変数でも HTML 化してしまう万能関数
sub outputvar {
  my ($indent, $name, $val) = @_;
  my $out = "";
  my $ref = "";
  if(ref($val)){
    if($val =~ /^(.+\=)?([A-Z]+)/){
      $ref = $2;
    }
  }
  if($name =~ /^(.*)_\d+$/){
    $name = $1;
  }

  if($ref eq 'SCALAR'){
    my $class = &getclass($name);
    $out .= "<div class=\"$class\">$indent$name : ".sanitize
($$val)."</div>";
  }elsif($ref eq 'ARRAY'){
```

```
    my $i = 1;
    foreach (@$val){
      $out .= outputvar($indent, "$name #$i", $_);
      $i++;
    }
  }elsif($ref eq 'HASH'){
    my $class = &getclass($name);
    $out .= "<div class=\"$class\">$indent$name :";
    foreach (sort(keys(%$val))){
      $out .= outputvar($indent."  ", $_, $val->{$
_});
    }
    $out .= "</div>";
  }else{
    my $class = &getclass($name);
    $out .= "<div class=\"$class\">$indent$name : ".sanitize
($val)."</div>";
  }
  return $out;
}

# タグの除去と改行の br 化
sub sanitize {
  $_ = $_[0];
  s/&/&amp;/g;
  s/</&lt;/g;
  s/>/&gt;/g;
  s|\n|<br />\n|g;
  return $_;
}

# 英数字部分の抽出
sub getclass {
  my ($name) = @_;
  my $class = "output";
  if($name =~ /^([a-zA-Z][a-zA-Z0-9]*)/){
    $class = $1;
  }
  return $class;
}
```

```
# 設定ファイルの読み込み
sub readconf {
  my ($links, $pages, $services) = @_;

  open IN, "app/link.txt" or throw Error::IO;
  while(<IN>){
    if(/^(.*?)\t(.*)\t(.*)$/){
      $links->{$1}{$2} = $3;
    }
  }
  close IN;

  open IN, "app/page.txt" or throw Error::IO;
  while(<IN>){
    if(/^(.*?)\t(.*?)\t(.*)$/){
      my @in = split(',', $2);
      my @out = split(',', $3);
      my %page = ();
      $page{'in'} = \@in;
      $page{'out'} = \@out;
      $pages->{$1} = \%page;
    }
  }
  close IN;

  open IN, "app/ws.txt" or throw Error::IO;
  while(<IN>){
    if(/^(.*?)\t(.*?)\#(.*?)\t(.*?)\t(.*)$/){
      my %svc = ();
      $svc{'wsdl'} = $2;
      $svc{'op'} = $3;
      my @in = split(',', $4);
      $svc{'in'} = \@in;
      $svc{'out'} = $5;
      $services->{$1} = \%svc;
    }
  }
  close IN;
}
```