修 士 論 文

# 法令実働化情報システムの進化容易性に関する研究

北陸先端科学技術大学院大学
情報科学研究科

箕輪 貴裕

2009 年 3 月

修 士 論 文

# 法令実働化情報システムの進化容易性に関する研究

指導教官　落水浩一郎 教授

審査委員主査　落水浩一郎 教授
審査委員　鈴木正人 准教授
審査委員　青木利晃 特任准教授

北陸先端科学技術大学院大学
情報科学研究科

0710067 箕輪 貴裕

提出年月: 2009 年 2 月

## 概 要

近年、我々の生活する社会は、電子社会システムによって支えられている。電子社会システムは、法律や条令などの各種規則に則って構築されており、こういった規則を完全に満たすように動作しなくてはならない。一方で、社会の変化に合わせて、規則は改定されていく。従って、システムは、規則の改定に合わせて、迅速にかつ低コストで進化させられなければならない。本研究では、LEIS を社会規則に依存するモデルと、機能要求に依存するモデルからなるものとして記述し、そのモデルを基に、進化を支援する方式を提案した。JAIST の履修規則と履修管理システムを用いてモデルの検証を行った。

# 目 次

# 第1章　序論

## 1.1　背景

近年、電子政府・電子自治体への取り組みをはじめとして、社会システムの電子化が急速に推し進められている。行政、金融、医療、交通、教育、企業などの活動の基盤部分が電子システム化され、それらがネットワークを介して相互に接続され巨大な電子社会システムが構築されつつある。

われわれの日常生活は、社会基盤としてのこのような電子社会システムの上に成り立っている。従って、電子社会システムは、これまでの情報システムのように機能を単に提供するだけでは十分ではなく、われわれが安心して生活できることを保証できるように設計・構築され、かつ運用・保守されている必要がある。

本学21世紀COEプログラム「検証進化可能電子社会」[1]では、安心できる電子社会システムが持っているべき要件として、正当性、アカウンタビリティ、セキュリティ、耐故障性、進化容易性の5つからなる安心性要件を提案している。この要件を満たす理想的な電子社会システムを法令実働化情報システム (Law Enforcing Information System, LEIS) と呼ぶ。

LEISは社会規則の適用を支援し、社会規則を完全に満たすように構築されていなければならない。さらに、社会規則に従って正しく構築されていることが保証され、かつ確認できる必要がある。社会規則は、法律や条令、組織内の各種規則といった、われわれが守らなければならない規則の総称である。しかし、社会は常に変化しており、社会規則はそれに合わせて改定される。従って、LEISには、前述した要件を満たすために、社会規則の改定に応じて、迅速に、かつ低コストで進化させられることが求められる。この特性を進化容易性と呼ぶ。

本研究では、LEISの進化容易性を扱う。進化容易性を実現するには、LEISを進化させようとするときに、システム開発者の持ちうる疑問に答える必要がある。その疑問としては、例えば「現行のシステムでは、社会規則がどのようにLEISに実現されているか」、「改定の結果、LEISの設計をどのように変更する必要があるか」といったことが挙げられる。これらの疑問を解消するには、対象としているLEISが、社会規則に対してどのように動作するのかを説明することが重要である。

また、社会規則を構造化して、制定理由を付加したものとして、アカウンタビリティ木が提案され、ソフトウェアアカウンタビリティの研究で利用されている [2]。アカウンタビリティ木は、社会規則の進化支援に有意義である。

## 1.2 アカウンタビリティ木

文献 [3] において、ゴール指向要求分析法を採用することにより、「規則群」と「規則を作る人が意図し、理解し、表現した世界」とを階層的に関連付けて構築する手法が提案されている。この手法では、社会規則の根源には、規則を作る人の意図した大きな目標があり、その目標に対して、それを達成するためのより具体的な目標がいくつか定められ、それらの目標に対しても更に具体的な目標が定められるといったことが続くと考える。そして、末端に来るのが、社会規則の条文に相当する。

この思考を図式化したものがゴール木 [3] である。ゴール木では、各目標、各条文をノードで表す。最終目標をルートとし、各目標の下に、その目標を達成するためのより具体的な目標、条文を配置する。従ってゴール木は、利害関係者からの社会規則の制定理由に関する質問に対して答えるために有用である。また、社会規則の改定を制御する役割も持つ。以降では、最終目標をゴール、中間の目標をサブゴールと呼ぶ。

社会規則の各条文は、規範としての意味を持つ。エックホフの法理論 [4] によれば、規範は、義務規範、権限規範、性質決定規範に分類でき、義務規範は、義務の内容によって、更に命令、禁止、許可、免除に分けられる。そして、いずれの種類の規範にも、主体と内容が定められる。また、規範間には、様々な種類の関連が存在する。エックホフの法理論を用いることで、各規範を型付けし、社会規則を構造化することができる。

ゴール木の葉に、エックホフの法理論に基づき型付けした規範を対応させたものを、アカウンタビリティ木と呼ぶ。アカウンタビリティ木により、社会規則の進化を支援することができる。なお、規範間の関連は、葉同士のクロスリンクとして表現する。図 1.1 に、アカウンタビリティ木の例を示す。なお、あるサブゴールの達成が複数の (サブ) ゴールの達成に寄与する場合があるため、ゴール木は必ずしも木であるとは限らない。

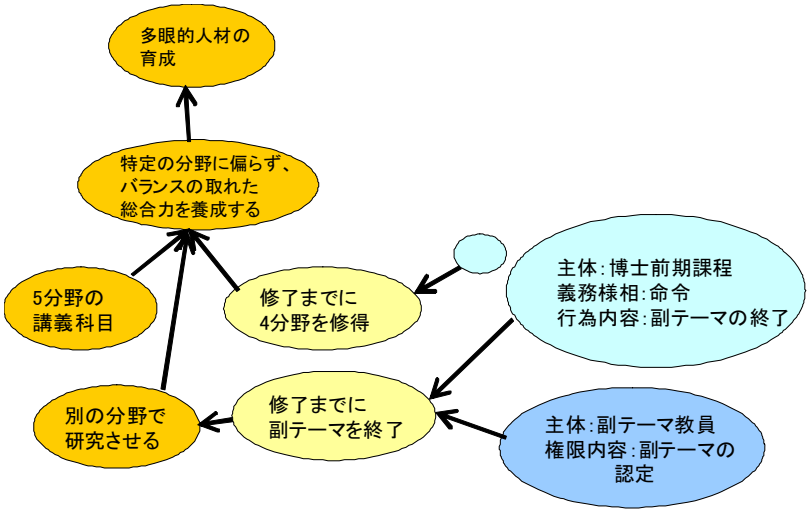


図 1.1: アカウンタビリティ木の例

## 1.3 本研究の目的

本研究の目的は、社会規則が改定された場合に必要となる LEIS の進化を容易にする方法を開発することである。

社会規則に由来する社会モデルと、機能要求に由来する機能モデルとが別個に存在すると考え、それぞれどのようなモデルにするべきかを考察し、それぞれの表現方法を定義する。そして、その 2 つのモデルから LEIS 全体の動作のモデルを導く方法を確立する。社会規則のモデル、機能要求のモデルから、LEIS 全体のモデルを導くことにより、進化容易性の実現を図る。

次に、アカウンタビリティ木と LEIS モデルに対して版管理モデルを定義する。LEIS モデルの版管理モデルは、アカウンタビリティ木の葉 (規範) および機能要求と、LEIS モデルとの構成要素との対応関係も記録する。これと LEIS モデルにより、社会規則と LEIS の設計との対応がとられる。

## 1.4 論文の構成

本論文の構成は以下のとおりである。

- 第 2 章 LEIS のモデル
  LEIS の進化支援の手法、社会モデル、機能モデルの定義、それらのモデルから LEIS 全体のモデルを導く方法を述べる。
- 第 3 章 JAIST 履修規則および履修管理システムの LEIS モデルによる表現
  第 2 章で述べた方法を用いて、本学の履修規則、履修管理システムのモデルの表現例を示すことにより、社会モデル、機能モデルの有用性を示す。
- 第 4 章 版管理モデル
  アカウンタビリティ木と LEIS モデルの版管理の方式を説明する。
- 第 5 章 結論
  本研究についてまとめ、今後の課題を述べる。

# 第2章　LEISのモデル

本章では、進化支援のための手法の提案して、その手法に必要となる、各モデルを定義する。

## 2.1　アプローチの概要

LEIS の目的は、社会規則の実社会への適用を支援することである。履修規則 (本学では履修案内と呼ぶ) は、修了までのプロセスと、プロセスの各ステップで達成しなければいけない条件を記述したものである。LEIS の一種である履修管理システムは、学生のプロセスの状態を記録し、各ステップの達成に必要な情報や状況を学生に提供するものである。

LEIS は、次のような手順により社会規則の適用を支援すると考えられる。図 2.1 にこの流れを示した。まず、機能が呼び出されると、実社会における、社会規則の適用を受け

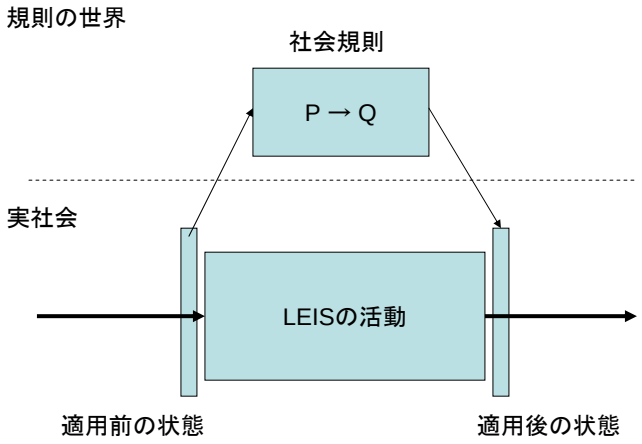


図 2.1: 一般的な LEIS の機能の動作

る人の「適用前の状態」を取得する。取得には、状態に関するデータを入力する、前もって登録されたデータを利用するというように、LEIS によって異なった方法が用いられる。次に、取得された状態に対して、社会規則を適用する。最後に、社会規則の適用結果を実社会に反映する。この手順も、結果を画面に表示する、通知書を印刷するというように、LEIS によって異なった方法によって行われる。

これらの手順を考慮すると、LEIS のモデルは社会規則を適用する部分と、実社会から適用前の状態を取得したり、社会規則の適用結果を実社会に反映したりする部分から成ると考えることができる。前者をモデル化したものを「社会モデル」、後者をモデル化したものを「機能モデル」と呼ぶこととする。

社会モデルは、社会規則のみに依存するモデルである。社会モデルは、対象とする LEIS とは無関係に、外部から入った「状態」のデータを基に、適用後の「状態」を出力する。一方、機能モデルは、機能要求、社会モデル、動作環境に依存する。機能モデルは、社会モデルと、コンピュータの入出力、記憶機能とを用いて、実社会における「適用前の状態」を取得して社会モデルに規則を適用させたり、社会規則の適用結果を実社会に反映したりする。

全体の構造は、社会モデルと、プラットフォーム、フレームワークといった動作環境とが土台として存在し、その上で機能モデルが両者の橋渡しを行うことで、LEIS 全体の機能を実現する形となる。

これら 2 種類のモデルを前もって記述しておき、それらを用いて進化を支援する。社会モデル、機能モデルは共に論理式の集合として記述する。社会モデルの論理式は条文毎、機能モデルの論理式は機能要求毎に記述する。いずれのモデルも版管理を行い、LEIS の進化が必要になった場合に論理式が取り出せるようにする。また、版管理モデルでは、論理式を保存するのみではなく、論理式と条文、機能要求との対応関係も管理する。これにより、社会規則や機能要求に変更があった場合に、社会モデル、機能モデルの修正すべき箇所をすぐに特定できるようにする。版管理モデルについては、第 4 章で述べる。

社会モデル、機能モデルを合成して、LEIS 全体の動作を表すようにしたものを、LEIS モデルと呼ぶ。LEIS モデルの導出は、今後の自動化を見込んでいる。LEIS モデルに対してモデル駆動型アーキテクチャ(MDA) などの開発手法を適用することで、LEIS を開発、メンテナンスできるようになることが見込まれる。

この考え方に基づき、新たに LEIS を開発する際の手順を以下のとおりに定めた。

1. 社会モデルを記述する。
2. 機能モデルを記述する。
3. 社会モデルと機能モデルから LEIS モデルを導出する。
4. LEIS モデルを用いて開発を行う。

社会規則が改定された場合の修正手順は以下のとおりに定めた。

1. 社会規則の改定に合わせて社会モデルを修正する。
2. 修正後の社会モデルと、既存の機能モデルから LEIS モデルを導出する。
3. LEIS モデルを用いて開発を行う。

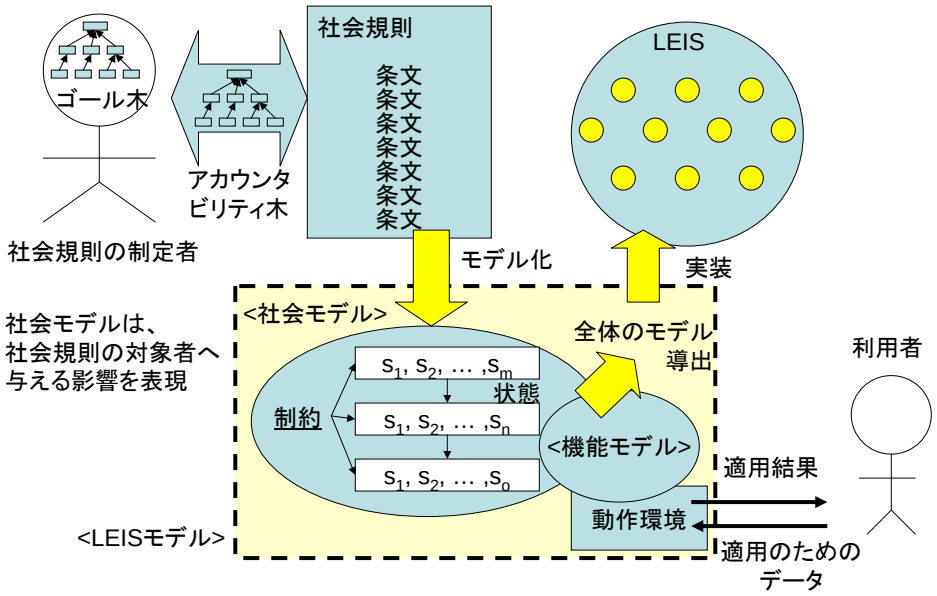


図 2.2: LEIS のモデル構造および LEIS の開発手順

以上の内容を図 2.2 にまとめた。

本研究では、上記を踏まえ、社会モデル、機能モデルを定義し、LEIS モデルの導出手法を提案することで、課題の解決を図る。

## 2.2 本手法の意義

本手法の意義は、社会規則を基にしたモデルから、LEIS の動作モデルを導くことにある。

LEIS の役割は社会規則の適用を支援することであるが、そのための手順には、適用のための情報の入手、内部での社会規則の適用、適用結果の出力がある。情報の入手の方法には、Web ページからの情報の入力、ディスクからの読み込みなどがある。適用結果の出力も同様である。このように、LEIS の動作は、単に社会規則の適用するのみではない。従って、社会規則の改定に合わせて LEIS を進化させるためには、LEIS から社会規則を適用する部分を特定して、その中から改定された条文に対応するものを見つける必要がある。

これを受けて本研究室で初めに提案されたのが、LEIS の開発に際して、LEIS の機能を実現する手順のうち社会規則と無関係の部分をフレームワークとして構築し、社会規則と関係のある部分をフレームワーク上で動作するコンポーネントとして構築する手法である [5]。これにより、社会規則の改定時に、LEIS の修正する必要のある部分が限定される。

また、社会規則の条文と、LEIS 構成要素 (クラス、メソッドなど) との対応表を作成し、社会規則や LEIS ソースコードと共に版管理することも提案されている。この対応表をクラス定義書と呼ぶ。社会規則が改定された時には、クラス定義書を利用して、改定された

条文に対応するLEIS構成要素を調べることにより、LEISの修正すべき部分を特定しやすくなる。

これらの両方を組み合わせることにより、進化容易性の実現が期待される。但し、修正すべき部分を特定して修正するには、対象とするLEISの仕組みをシステム開発者が理解している必要がある。また、社会規則に条文が追加された場合には、クラス定義書では対応できないといった問題もある。

今回提案した手法には、従来手法に対して特徴が2つある。まず、進化支援のために、社会規則の改定に応じて修正する必要のあるデータが、社会規則のみに依存していることである。本手法では、社会規則の改定があったならば、まず社会モデルを構成する論理式を修正し、版管理モデルに新しい社会モデルを登録する必要がある。しかしながら、社会モデルは社会規則が我々に与える影響を表現したものであるため、社会規則の改定内容を理解していれば、修正することが可能である。対象としているLEISや、使用するプラットフォーム、フレームワーク、言語などとは無関係であり、それらの知識は必要ない。この特徴は、本手法を実行するために必要となる、システム開発者への負担が軽いということを意味する。もう1つの特徴は、社会モデル、機能モデルから、LEIS全体の動作モデルを導出することである。現在、モデル駆動型アーキテクチャなど、ソフトウェアモデルを利用した開発手法が研究されている。本手法によって導かれるLEISモデルに、これらの開発手法を適用することにより、LEISを進化させられると考えられる。

以上、2つの特徴により、本手法がLEISの進化容易性に寄与すると考えられる。

## 2.3 社会モデル

社会モデルは、社会規則を適用することにより、我々がどのような影響を受けるかを表したモデルである。まず、社会モデルで取り扱うべきパラメータを検討する。そのためには、社会規則の適用対象となる「我々の状態」がどのようなものであるかを知る必要がある。その状態を構成する要素が、社会モデルで取り扱うべきパラメータである。

我々は、自らの意思によって行動する。社会規則の意図した内容を、我々に実行させるには、実行する意思を持たせる必要がある。そのため、社会規則は我々を感化させる条文を含んでいる。この条文が我々の状態に影響を与える。

例えばJAISTの学生は、履修規則の適用を受け、修了までに以下の一連のプロセスを実行する。

1. 一定の単位を修得して、副テーマを実行する
2. 副テーマを終了した上で、更に必要な単位を修得し、研究計画を立てて、研究計画提案書を提出する
3. 研究計画提案書を提出してから1年間修士研究を行い、修士論文審査に合格して修了する

履修規則はこのように、修了するまでのプロセスを定めている。

プロセスにはいくつかのステップが設けられ、あるステップを達成しなければ、次のステップには進めない。例えば、上記に示したプロセスの各項目を1つのステップとすると、上記ステップ1に関する条文は、「副テーマの開始要件」を定めている一方、上記ステップ2に関する条文では、研究計画提案書の提出の要件に、「副テーマの終了」を含めている。これにより、学生に1つずつステップを達成させ、確実にプロセスを実行させている。

上記ステップ2に注目する。このステップに関する条文では、研究計画提案書の提出要件として、副テーマの終了、一定の単位の修得、十分な研究計画の提案を定めている。つまり、これらの要件を満たしていた場合に、「研究計画提案書の提出」という活動を許可するという意味を持つ。これは、ある条件と、特定の活動の許可の有無との関係を決定している。従って、まず「特定の活動が許可される」ということを、社会モデルのパラメータとして取り扱うべきである。

なお、ある人にある活動が許可されることと、その人が実際にその活動をすることは、全く別物である。以降の説明では、この2つを区別する。

さて、この条文で述べられている要件を個別に調べると、他にも社会モデルで取り扱うべきパラメータが見つかる。副テーマの終了は、「特定の活動が既に実行された」という種類のパラメータであり、これも社会モデルで取り扱うべきである。一定の単位の修得は、その人の「修得単位数」というパラメータに対する条件である。そして、修得単位数は過去の活動の積み重ねによって決定されるパラメータの1つである。例えば、「修得単位数が6である」というのは、「単位数が2の科目を3つ修得する」という活動によって生み出されたと考えることができる。このようなパラメータを「現在のパラメータ」と呼ぶこととして、このパラメータも社会モデルのパラメータとして取り扱う。「現在のパラメータ」を「特定の活動が既に実行された」というパラメータの変形であると考えれば、「特定の活動が許可される」というパラメータの変形として、「許容されるパラメータ」というものも定義される。

以上で、我々を感化させる条文を取り上げたが、社会規則には、我々に直接影響を与える条文以外にも、これらの条文を通して間接的に影響を与えるものが存在する。それは、事実を定義する条文である。この条文は、「AはBの一種である」というように、物を分類したり、物に性質を付加したりする。例えば、「ソフトウェア設計論は、基幹講義に分類される」という条文がこれに該当する。この条文における事実を、社会モデルのパラメータとして取り扱うべきである。

以上で、社会モデルで取り扱うべきパラメータが示された。続いて、各種パラメータの表現方法を以下のとおりに定義した。

- 「特定の活動が既に実行された」、「特定の活動が許可される」、事実を定義する条文における事実
  これらのパラメータは、命題または述語を、社会モデルの記述時に定義して表現する。

- 現在の状態要素、許容される状態要素
  これらのパラメータは、変数または関数を、社会モデルの記述時に定義して表現する。

論理式の例として、「60 点以上取得した科目は合格とする」という条文は、以下の述語と論理式により表される。

- 述語
  - $Got(x)$
    科目 $x$ に合格した
  - $Score(x, y)$
    $y$ は科目 $x$ の成績である
- 論理式
  $Score(x, y) \wedge (y \geq 60) \rightarrow Got(x)$

## 2.4 機能モデルと動作環境

機能モデルは、LEIS の機能要求に定められた、各種機能を実現する手順のうち、社会規則の対象者の状態を取得して社会モデルに渡したり、社会モデルによる規則の適用結果を外部に出力して反映する部分のモデルである。但し、成績の登録機能のように、社会規則とは全く無関係のものも取り扱う。対象者の状態を取得したり、適用結果を出力したりするために必要な機能は、「動作環境」が提供する。機能モデルでは、社会モデルと、システムの動作環境とを橋渡しすることにより、LEIS の機能を実現する。

なお、動作環境は、対象とするシステムのフレームワークなどによって変化する。本論文における LEIS は、第 3 章で用いる履修管理システムと同じく、JBoss Seam を用いた Web システムに限定する。

機能は、LEIS の動作によって実現される。従って、機能は呼び出される前の状態に対して、処理終了後の状態を決定する。そのため、動作環境の各パラメータは前と後の状態を区別する。そのために、関数名、変数名、命題変数名、述語名の先頭に以下の接頭辞を付ける。

- Prev_ (述語名) または、 prev_ (関数名、変数名)
  述語の場合は、機能が呼び出される直前に、満たされていたということを表す。
  関数、変数の場合は、機能が呼び出される直前の時点における状態を表す。

- Next_ (述語名) または、 next_ (関数名、変数名)
  述語の場合は、機能の処理が終了した直後に、満たされることを表す。
  関数、変数の場合は、機能の処理が終了した直後の時点における状態を表す。

機能モデルの論理式中では、動作環境の命題変数、述語、関数、変数は、必ずこれらの接頭辞を付けなくてはならない。

続いて、機能モデルを記述するために必要である、「動作環境」について説明する。一般にLEISの動作環境は、入力機能、出力機能、記憶機能を持つ。JBoss Seamの場合、入力機能では、入力ボックスの文字列、ボタンやリンクのクリックなどが入力される。出力機能では、所定のページのテンプレートに、所定のデータが合成されて、利用者のWebブラウザに出力される。記憶機能は、コンテキストと呼ばれる領域に一時的に記憶するものと、データベースにアクセスするものとが存在する。

ボタンやリンクのクリックは、特定のセッションBeanの特定のメソッドの呼び出しという形で入力される。また、データはコンテキスト中にコンテキスト変数と呼ばれる形で記憶することができる。入力ボックスに入力された文字列などは、コンテキスト変数を通して入力される。出力されるページに合成されるデータもコンテキスト変数として出力する。データベースには、フレームワークが所定のクエリを発行し、その実行結果を通してアクセスする。このそれぞれに対して、動作環境の述語と関数を定義する。

これらを踏まえ、以下の述語を設ける。なお、以降の説明では、述語名、関数名の接頭辞は省略している。

- MethodCall( クラス名.メソッド名 )
  「指定のクラス名、メソッド名で表現されるセッションBeanが、ページのクリック操作によって呼び出されている。」
  この述語は、$Prev\_MethodCall$(クラス名.メソッド名) のように、必ず、Prev_接頭辞を付けて用いる。

- ShowData(コンテキスト変数名, インデックス, メンバ名...)
  「指定のコンテキスト変数名、メンバ名で表されたオブジェクトのメンバ変数の値が、表示中のページに含まれている。」
  当該変数が配列またはリストであるならば、インデックスに番号を設定し、単独の値のみを持つ変数ならば、インデックスは0とする。メンバ名はJavaの文法に従い、参照したいメンバを指すように記述する。この述語は、機能の処理終了時に表示されるWebページの内容に制約を与えるために用いる。従って、$Next\_ShowData$(コンテキスト変数名, メンバ名...) のように、必ず、Next_接頭辞をつけて用いる。

また、以下の関数を設ける。

- context(コンテキスト変数名, インデックス, メンバ名...)
  コンテキスト変数名、メンバ名で表されたオブジェクトのメンバ変数の値を返す。当該変数が配列またはリストであるならば、インデックスに番号を設定し、単独の値のみを持つ変数ならば、インデックスは0とする。但し、該当するオブジェクトやメンバ変数が無い場合はnullを返す。メンバ名はJavaの文法に従い、参照したいメンバを指すように記述する。

- data(クエリ, インデックス, メンバ名...)
  データベースに対してクエリを実行して得られたオブジェクトの内、インデックスに対応するオブジェクトのメンバ変数の値を返す。
  但し、該当するオブジェクトやメンバ変数が無い場合は null を返す。メンバ名は Java の文法に従い、参照したいメンバを指すように記述する。

## 2.5 汎用関数

以下の関数は、社会モデル、機能モデル共通で用いられ、値の加工を行う。
まず、文字列から 1 文字を取り出す関数を定義する。

- $charAt(s, i)$ ($s$ は文字列、$i$ は 0 以上の整数)
  文字列の (i-1) 文字目の文字を返す。

学生の修得単位数は、修得した科目それぞれの持つ単位数の合計値である。このような、合計値を表すことを可能とするために、下記の関数を定義する。この関数は図 2.3 のように、ある変数を複数の変数の合計値であると考え、そのそれぞれの変数に対して制約を与えることを可能とする。

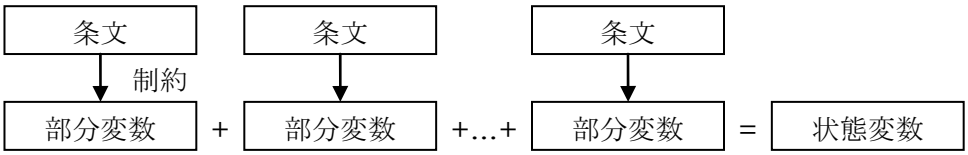


図 2.3: 変数の分割

- $partof(variableName, identifier)$ ($variableName$ は数値型の変数名、 $identifier$ は任意の型)
  $variableName$ で表される変数の値を、複数の変数の値の合計値であると考え、そのうちの 1 つの値を返す。このとき $identifier$ を、分割後の変数のそれぞれを識別する識別子として扱う。

## 2.6 LEIS モデルの導出

### 2.6.1 等式の代入

LEIS モデルは、社会モデル、機能モデルの論理式を組み合わせて、社会モデルの要素を消去した、新しい論理式を導くことで得られる。これを論理式の合成と呼ぶこととす

る。通常、新しい論理式は、元の 2 つの論理式に論理法則を適用して新しい論理式を推論することで得られる。しかし、社会モデル、機能モデルで取り扱う論理式には、「特定の等式または不等式が成り立つ」という意味の命題が含まれるものがある[1]。本節では、そのような式に用いられている関数、変数を消去するための、等式の代入の方法を述べる。なお、論理式が「特定の等式が成り立つ」という意味の命題である場合、その等式は他のいかなる論理式の等式、不等式にも明らかに代入可能である。これは、モデルを構成する論理式は、常に成り立つべき条件を表すからである。しかしながら、そのような命題を含む論理式が、その命題のみで構成されているとは限らない。本節で対象とする等式は、そのような論理式に含まれる等式である。

いかなる論理式も、乗法標準形に変換可能であることが知られている。乗法標準形に変換することで、LEIS が満たすべき条件を、複数の最大項に分けることができる。最大項のそれぞれを、新しい論理式と考える。

さて、新しい論理式の集合に次の 2 つ形の論理式が含まれていると仮定し、これらを合成することとする。前者の論理式には、代入したい等式、後者には代入先としたい等式が含まれている。但し、$M_b(y, x_{b,1}, x_{b,2}, ..., x_{b,p})$ は「引数の値を用いたある等式または不等式が成り立つ」という述語であり、その等式・不等式を代入先として、$A_i$、$B_i$ を論理式、$f(x_{a,1}, x_{a,2}, ..., x_{a,o})$ を式、$y$ を消去したい変数または関数とする。

$$A_1 \vee A_2 \vee ... \vee A_m \vee (y = f(x_{a,1}, x_{a,2}, ..., x_{a,o}))$$
$$B_1 \vee B_2 \vee ... \vee B_n \vee M_b(y, x_{b,1}, x_{b,2}, ..., x_{b,p})$$

これらの条件は、ド・モルガンの法則の適用と「$\rightarrow$」記号の導入により、それぞれ次のように表される。

$$(\neg A_1 \wedge \neg A_2 \wedge ... \wedge \neg A_m) \rightarrow (y = f(x_{a,1}, x_{a,2}, ..., x_{a,o}))$$
$$(\neg B_1 \wedge \neg B_2 \wedge ... \wedge \neg B_n) \rightarrow M_b(y, x_{b,1}, x_{b,2}, ..., x_{b,p})$$

前者の論理式は、条件 $A_1$、$A_2$、...、$A_m$ の全てが満たされた場合に限り、等式の正当性が保障されることを表す。従って、この論理式の等式を後者に代入するには、これらの条件を後者[2]の代入先を有効とする条件に追加する必要がある。条件を追加して、代入した結果は次のとおりとなる。但し、$M_{a,b}(x_{a,1}, ..., x_{a,o}, x_{b,1}, ..., x_{b,p})$ は代入結果を表す。

$$(\neg A_1 \wedge \neg A_2 \wedge ... \wedge \neg A_m \wedge \neg B_1 \wedge \neg B_2 \wedge ... \wedge \neg B_n)$$
$$\rightarrow M_{a,b}(x_{a,1}, ..., x_{a,o}, x_{b,1}, ..., x_{b,p})$$

これを「$\rightarrow$」記号を用いない形式に戻すと次のとおりとなる。これが合成結果である。

$$A_1 \vee A_2 \vee ... \vee A_m \vee B_1 \vee B_2 \vee ... \vee B_n \vee M_{a,b}(x_{a,1}, ..., x_{a,o}, x_{b,1}, ..., x_{b,p})$$

[1]本論文で取り上げる論理式では、成り立つべき等式・不等式を括弧で囲んで表現している。これは述語を定義して表現することも可能であり、その場合は式を構成する全ての関数、変数が述語の引数となる。

[2]「$\rightarrow$」の右側が代入先となる $M_b$ のみになるよう変換した。これは、条件を追加する際に、それの影響を代入結果である $M_{a,b}$ にとどめるためである。

次の形の論理式の合成結果も導出する。

$$A_1 \vee A_2 \vee ... \vee A_m \vee (y = f(x_{a,1}, x_{a,2}, ..., x_{a,o})) \tag{2.1}$$

$$B_1 \vee B_2 \vee ... \vee B_n \vee \neg M_b(y, x_{b,1}, x_{b,2}, ..., x_{b,p}) \tag{2.2}$$

この場合の合成結果は以下のとおりとなる。

$$A_1 \vee A_2 \vee ... \vee A_m \vee B_1 \vee B_2 \vee ... \vee B_n \vee \neg M_{a,b}(x_{a,1}, ..., x_{a,o}, x_{b,1}, ..., x_{b,p})$$

### 2.6.2 等式の代入例

等式の代入の例として、「メソッド checkSyuryou が呼び出されたら、データベースから副テーマの成績を読み出して社会モデルの要素と対応付ける」ということを表す論理式と、「副テーマの成績が 60 以上であって、一定の単位を修得していたら修了とする」ということを表す論理式との合成を取り上げる。まず、前者、後者の論理式はそれぞれ以下のとおりである。但し、副テーマの成績は、$prev\_data(クエリ, fuku)$ によってデータベースから読み出せるものとし、$subThemeScore$ を副テーマの成績を表す変数、$GotEnough$ を「一定の単位を修得した」、$Graduate$ を「修了する」という命題とする。

$$Prev\_MethodCall(checkSyuryou) \rightarrow (prev\_data(クエリ, fuku) = subThemeScore) \tag{2.3}$$

$$((subThemeScore \geq 60) \wedge GotEnough) \rightarrow Graduate \tag{2.4}$$

まず条件 (2.3)、(2.4) から「$\rightarrow$」記号を取り払い、後者に関しては、ド・モルガンの法則に基づき $\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$ の置き換えを適用して、乗法標準形に変換すると、それぞれ以下のとおりになる。

$$\neg Prev\_MethodCall(checkSyuryou) \vee (\mathbf{prev\_data(クエリ, fuku) = subThemeScore}) \tag{2.5}$$

$$\neg GotEnough \vee Graduate \vee \neg(\mathbf{subThemeScore \geq 60}) \tag{2.6}$$

条件 (2.5) の太字部分が代入したい等式である。条件 (2.6) の太字部分が代入先としたい不等式である。実際に代入すると不等式は以下のとおりとなる。これが、2.6.1 節における $M_{a,b}$ に相当する。

$$prev\_data(クエリ, fuku) \geq 60$$

さて、条件 (2.5)、(2.6) の組み合わせは、2.6.1 節で取り上げた条件 (2.1)、(2.2) の形に当てはまる。従って、2 つの条件を合成すると以下の以下のとおりとなる。

$$\neg Prev\_MethodCall(checkSyuryou) \vee \neg GotEnough \vee Graduate \vee \neg(\mathbf{prev\_data(クエリ, fuku) \geq 60})$$

これを「→」記号を用いた形式に変換すると以下のとおりとなる。

$$(Prev\_MethodCall(checkSyuryou) \land GotEnough \land (prev\_data(\text{クエリ}, fuku) \geq 60)) \\ \rightarrow Graduate$$

この論理式は、「メソッド checkSyuryou が呼び出されていて、一定の単位が修得済みであって、データベースから読み出された副テーマの成績が 60 以上であれば、修了とする」という意味を持つ。

# 第3章　JAIST履修規則および履修管理システムのLEISモデルによる表現

本章では、前章で提案したモデルを JAIST の履修規則および履修管理システムに適用することにより、提案手法を検証する。履修規則は、学生が本学を修了するまでに行わなければならない事項が書かれた社会規則である。履修管理システムは、学生の修得状況を管理する LEIS である。また、学生に対する機能として、「副テーマの提出」、「研究計画提案書」、「就職用推薦書の発行」、「修了」という各チェックポイントについて、通過するための各要件を満たしているかを表示する機能を持つ。先行研究において試験的に開発され、JBoss Seam をフレームワークとする Web システムとして実装されている。

## 3.1　履修規則に対する社会モデルの表現

ここでは、JAIST 履修規則に対応する社会モデルを表現する。一例として、研究計画提案書の提出要件に関する条文を取り上げる。その部分を図 3.1 に示す。

この部分では、研究計画提案書の提出要件が列挙されている。このうち、修得状況と関係するのは、1 と 2 である。また、要件 2 は科目数と分野数という 2 つの事項に言及しているため、2 つの要件に分けることができる。なお、ほとんどの科目の単位数が 2 であるため、今回は単位数の要件は外すこととした。これらの要件、即ち、副テーマ、科目数、分野数による要件のそれぞれを論理式で表現する。最後に、それらを統括して「研究計画提案書の提出」全体の要件を表現する。また、任意の科目についての科目番号と、共通、導入、基幹、専門、先端のいずれであるかとの対応が、別に規定されているので、図 3.2 に示す。

これらの条文に対応する論理式を記述するため、社会モデルにおける述語を表 3.1、変数を表 3.2、関数を表 3.3 のとおりに定めた。本章の表、図、及び説明では、$x$ は科目を表す変数、$i$ は整数を表す変数とする。

まず、「副テーマの研究が終了していること」という要件に対応する論理式を記述する。これは「副テーマの研究が終了していることが、研究計画提案書の提出要件の 1 つ目の要件をクリアしていることに等しい」という意味を持つ。「副テーマの研究が終了していること」は、$FinishedSubTheme$ と定義されている。「副テーマに関する要件をクリアして

4.4.4 研究指導 4. 主テーマ

2) 研究計画提案書の提出要件

【修士論文研究を履修する場合】

1. 副テーマの研究が終了していること。
2. 導入講義課目 (ただし、3 科目 6 単位まで)、および基幹・専門・先端講義科目から 4 分野 6 科目 12 単位以上単位を得ていること。
3. 研究計画の内容が十分であること。

図 3.1: 履修規則における研究計画提案書の提出要件

4.2 教育課程表

(注 1) 各記号の意味は以下のとおりである。

I1xx ... 導入講義課目

I2xx … 基幹講義課目

I4xx … 専門講義科目

I6xx … 先端講義課目

_0xx … 共通科目

図 3.2: 履修規則における科目番号の規定

表 3.1: 社会モデルにおける命題変数、述語

| 表記 | 説明 |
|---|---|
| $FinishedSubTheme$ | 副テーマの研究を終えた |
| $PermittedProposal$ | 研究計画提案書を提出できる |
| $Passed(x)$ | 科目 $x$ を修得した |
| $GotAsIntroFundProAdv(x)$ | 科目 $x$ が、導入、基幹、専門、先端科目のいずれかに当てはまり、なおかつ修得した |
| $GotField(x)$ | 導入、基幹、専門、先端科目から、分野 $x$ の科目を修得した |
| $ProposalPresentCondition_0$ | 研究計画提案書を提出するための要件のうち、副テーマに関する要件を既に満たしている |
| $ProposalPresentCondition_1$ | 研究計画提案書を提出するための要件のうち、科目数に関する要件を既に満たしている |
| $ProposalPresentCondition_2$ | 研究計画提案書を提出するための要件は 3 つあるが、分野数に関する要件を既に満たしている |
| $SubjectLevel(x, y)$ | 科目 $x$ が、共通、導入、基幹、専門、先端の分類方式では、$y$ に分類される |
| $SubjectField(x, y)$ | 科目 $x$ は、少なくとも分野 $y$ に属している |

表 3.2: 社会モデルにおける変数

| 表記 | 説明 |
|---|---|
| $introFundProAdvNumber$ | 導入、基幹、専門、先端科目から修得した科目数 |
| $introFundProAdvFieldNumber$ | 導入、基幹、専門、先端科目から修得した分野数 |

表 3.3: 社会モデルにおける関数

| 表記 | 説明 |
|---|---|
| subjectName(x) | 科目 x の科目番号を返す |

副テーマを終了したかを判定

$$FinishedSubTheme \equiv ProposalPresentCondition_0$$

図 3.3: 研究計画提案書提出の要件 1 に対応する論理式

いること」は、$ProposalPresentCondition_0$ と定義されている。従って、1 つ目の要件について、図 3.3 のような論理式が定められる。

続いて、2 つ目の要件は、「導入、基幹、専門、先端講義科目から 6 科目以上修得していること」である。これは、論理式で表す場合、3 つの段階に分けられる。

まず、各科目について、修得済みでなおかつ、導入、基幹、専門、先端講義科目のいずれかに当てはまるかを調べる。次に、それらの条件を満たす科目をカウントする。最後に、カウントした結果を基にして、本条件を満たしているかを確認する。

科目数のカウントは、$partof$ 関数を用いることにより、上記条件を満たす科目数を表す変数 $introFundProAdvNumber$ を分割し、分割結果のそれぞれを、1 つの科目に対応させる。条件を満たしている科目については 1 を、そうでなければ 0 を設定することで、分割した変数の合計値がカウント結果となる。

以上から、2 つ目の要件について、図 3.4 の論理式が定められる。

最後の要件は、「導入、基幹、専門、先端講義科目から 4 分野以上修得していること」である。この要件についても同様に、図 3.5 の論理式が求められる。

最後に、上記で述べた要件を全て満たしている場合に限り研究計画提案書を提出できるという論理式を記述する。「要件をすべて満たしている」ことを表現するためには、量化子 $\forall$ を用いる。論理式は図 3.6 のとおりである。

LEIS モデルの導出に必要であるため、科目番号の規定 (図 3.2) についても、論理式を求めた。図 3.7 のとおりである。

## 3.2 履修管理システムに対する機能モデルの表現

ここでは履修管理システムに対応する機能モデルを表現する。前節で取り上げた部分に関連して、研究計画提案書を提出するための各要件を満たしているかを表示する機能を取り上げる。この機能は、次のような仕組みになっている。

1. 利用者が、当該機能へのリンクをクリックすると、フレームワークは「checkpointAction」というセッション Bean の「checkKenkyu」というメソッドを呼び出す。

2. フレームワークは上記メソッドを呼び出す直前に、SQL 文を発行して、利用者の修得した科目の一覧を、上記 Bean の配列「userScores」に読み出す。

各科目が修得済みでなおかつ、導入、基幹、専門、先端講義科目のいずれかに当てはまるかを調査

$$\begin{aligned}((Passed(x) \quad &\wedge \quad (SubjectLevel(x, \text{"導入"}) \\ &\vee \quad SubjectLevel(x, \text{"基幹"}) \\ &\vee \quad SubjectLevel(x, \text{"専門"}) \\ &\vee \quad SubjectLevel(x, \text{"先端"}))) \\ &\equiv \quad GotAsIntroFundProAdv(x)\end{aligned}$$

上記に当てはまる科目をカウント

$$\begin{aligned}&(GotAsIntroFundProAdv(x) \rightarrow (partof(introFundProAdvNumber, x) = 1)) \\ \wedge \quad &(\neg GotAsIntroFundProAdv(x) \rightarrow (partof(introFundProAdvNumber, x) = 0))\end{aligned}$$

その科目数が 6 以上であるかを判定

$$(introFundProAdvNumber \geq 6) \equiv ProposalPresentCondition_1$$

図 3.4: 研究計画提案書提出の要件 2 に対応する論理式

各科目が修得済みでなおかつ、導入、基幹、専門、先端講義科目のいずれかに当てはまるかを調査

$$\begin{aligned}(Passed(x) \quad &\wedge \quad (SubjectLevel(x,\text{"導入"}) \\ &\vee \quad SubjectLevel(x,\text{"基幹"}) \\ &\vee \quad SubjectLevel(x,\text{"専門"}) \\ &\vee \quad SubjectLevel(x,\text{"先端"}))) \\ &\equiv \quad GotAsIntroFundProAdv(x)\end{aligned}$$

分野「ア」の修得が修得済みであるかを確認
(分野「イ」、「ウ」、「エ」、「オ」についても、強調部分のみ変更した論理式が入る)

$$\begin{aligned}\exists x(GotAsIntroFundProAdv(x) \quad &\wedge \quad SubjectField(x,\text{"}\mathbf{ア}\text{"})) \\ &\equiv \quad GotField(\text{"}\mathbf{ア}\text{"})\end{aligned}$$

分野数に「ア」をカウント (ア、イ、ウ、エ、オをそれぞれ 1 としてカウントする)
(分野「イ」、「ウ」、「エ」、「オ」についても、強調部分のみ変更した論理式が入る)

$$\begin{aligned}&(GotField(\text{"}\mathbf{ア}\text{"}) \\ &\rightarrow (partof(introFundProAdvFieldNumber,\text{"}\mathbf{fieldA}\text{"}) = 1)) \\ &\wedge(\neg GotField(\text{"}\mathbf{ア}\text{"}) \\ &\rightarrow (partof(introFundProAdvFieldNumber,\text{"}\mathbf{fieldA}\text{"}) = 0))\end{aligned}$$

分野数が 4 以上であるかを確認

$$(introFundProAdvFieldNumber \geq 4) \equiv ProposalPresentCondition_2$$

図 3.5: 研究計画提案書提出の要件 3 に対応する論理式

全ての要件を満たしている場合に限り研究計画提案書の提出を許可

$$\forall i(ProposalPresentCondition_i) \equiv PermittedProposal$$

図 3.6: 研究計画提案書提出要件全体の論理式

$$
\begin{aligned}
(charAt(subjectName(x), 1) =' 0') &\equiv SubjectLevel(x, \text{"共通"}) \\
(charAt(subjectName(x), 1) =' 1') &\equiv SubjectLevel(x, \text{"導入"}) \\
(charAt(subjectName(x), 1) =' 2') &\equiv SubjectLevel(x, \text{"基幹"}) \\
(charAt(subjectName(x), 1) =' 4') &\equiv SubjectLevel(x, \text{"専門"}) \\
(charAt(subjectName(x), 1) =' 6') &\equiv SubjectLevel(x, \text{"先端"})
\end{aligned}
$$

図 3.7: 科目番号の規定に対応する論理式

3. 副テーマを終了済みの場合には、上記一覧に「sub」という文字列を科目名として記録した項目が含まれる。これを利用して終了済みかを判定する。

4. 上記一覧には、科目毎に科目番号が記録されている。共通、導入、基幹、専門、先端科目のいずれであるかは、科目番号を基にして判定する。

5. メソッドは、各要件について、それが満たされているかの判定結果を「vectorlist」という名前のコンテキスト変数 (配列) に出力する。

6. メソッドの終了後、フレームワークが結果表示ページのテンプレートに「vectorlist」の内容を合成して利用者に示す。

手順 1 によれば、当該機能は「checkPointAction」というセッション Bean の「checkKenkyu」というメソッドで実現している。従って、この機能に関係する論理式はいずれも、「Prev_MethodCall (checkpointAction. checkKenkyu) $\rightarrow$ A」という形になる。これは、当該メソッドが呼び出された場合に、A に相当する動作をしなくてはならないということを表す。逆にそれ以外の場合には、A に相当する動作をする必要は無い。

手順 2 によれば、データベースに記録されている項目は、手順 3 で示した例外を除けば、全て修得した科目のデータである。(未履修、受講中、不合格のものは含まれない。) 従って、データベースの項目と、社会モデルの述語 $Passed(x)$ とが対応する。これについて、図 3.8 の論理式が定められる。

手順 3 は、データベースからの読み出し結果と、副テーマの研究を終えたかを表す社会モデルの命題変数 $FinishedSubTheme$ とを対応付ける。データベースに「sub」という科目名の項目が存在した場合に、$FinishedSubTheme$ を真にする。従って、図 3.9 の論理式が定められる。

手順 4 は、データベース上の科目番号と、社会モデル上の科目番号とを対応付ける。これに対応する論理式は図 3.10 のとおりとなる。

手順 5、6 は、各条件の判定結果の表示に関するものである。この手順は、結果を「vectorList」コンテキスト変数を通して、画面に出力するということを示している。JBoss

修得した科目について、データベースの内容と社会モデルの述語とを対応付ける

$$Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\rightarrow ((prev\_data(selectu fromUserScoreuwhereu.username =$$
$$: username, i, this) \neq null) \equiv Passed(x_i))$$

図 3.8: 手順 2 に対応する論理式

副テーマについて、データベースの内容と社会モデルの命題変数とを対応付ける

$$Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\rightarrow ((\exists m(prev\_data(selectu fromUserScoreuwhereu.username =$$
$$: username, m, course.title) = "sub")) \equiv FinishedSubTheme)$$

図 3.9: 手順 3 に対応する論理式

科目番号について、データベースの内容と社会モデルの関数とを対応付ける

$$Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\rightarrow (subjectName(x_i) = prev\_data(selectu fromUserScoreu$$
$$whereu.user.username =: username, i, course.coursename))$$

図 3.10: 手順 4 に対応する論理式

コンテキスト変数を、各要件を満たしているかの判定結果と対応付ける

$$Prev\_MethodCall(checkPointAction.checkKenkyu) \rightarrow$$
$$(next\_context(vectorlist, i, pass) \equiv ProposalPresentCondition_i)$$

上記コンテキスト変数の内容を結果ページに出力する

$$Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\rightarrow \forall i(Next\_ShowData(vectorlist, i, pass))$$

図 3.11: 手順 5、6 に対応する論理式

機能要求手順 4(乗法標準形に変換してある)

$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\vee(\mathbf{subjectName(x_i)} = prev\_data(selectu fromUserScoreu$$
$$whereu.user.username =: username, i, course.coursename))$$

図 3.12: 1 回目の合成対象 1

Seam を用いたシステムでは、画面に結果を表示するには、このようにコンテキストを通す必要がある。また、出力される Web ページのテンプレートで、そのコンテキスト変数の該当するメンバの値を出力するように設定されている必要がある。従って、図 3.11 の論理式が定められる。

## 3.3 履修管理システムに対する LEIS モデルの導出

ここでは、社会モデルと機能モデルから、LEIS モデルの論理式を導出する一例を示す。

初めの論理式としては、機能要求手順 4(図 3.10) を選択した。まず、これと科目番号の規定関係 (図 3.7) のそれぞれとを合成する。これらの論理式を図 3.12、図 3.13 に抜き出した。これらの図で強調した部分が、消去しようとしている要素である。

科目番号の規定側が「charAt」関数で変数 $x$ から 2 文字目を選んでいるのに対し、機能要求手順 4 側は、文字列全体を参照しているためこのままでは代入できない。しかし、文字列全体が等しければ、2 文字目も等しいはずである。そこで、機能要求手順 4 側の文字列の比較部分を 2 文字目の比較に置き換えると次のようになる。

$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$

科目番号の規定 (乗法標準形に変換して分割してある)

$$
\begin{aligned}
\neg(charAt(\mathbf{subjectName}(\mathbf{x}),1) =' 0') &\vee SubjectLevel(x,\text{"共通"})\\
(charAt(\mathbf{subjectName}(\mathbf{x}),1) =' 0') &\vee \neg SubjectLevel(x,\text{"共通"})\\
\neg(charAt(\mathbf{subjectName}(\mathbf{x}),1) =' 1') &\vee SubjectLevel(x,\text{"導入"})\\
(charAt(\mathbf{subjectName}(\mathbf{x}),1) =' 1') &\vee \neg SubjectLevel(x,\text{"導入"})\\
\neg(charAt(\mathbf{subjectName}(\mathbf{x}),1) =' 2') &\vee SubjectLevel(x,\text{"基幹"})\\
(charAt(\mathbf{subjectName}(\mathbf{x}),1) =' 2') &\vee \neg SubjectLevel(x,\text{"基幹"})\\
\neg(charAt(\mathbf{subjectName}(\mathbf{x}),1) =' 4') &\vee SubjectLevel(x,\text{"専門"})\\
(charAt(\mathbf{subjectName}(\mathbf{x}),1) =' 4') &\vee \neg SubjectLevel(x,\text{"専門"})\\
\neg(charAt(\mathbf{subjectName}(\mathbf{x}),1) =' 6') &\vee SubjectLevel(x,\text{"先端"})\\
(charAt(\mathbf{subjectName}(\mathbf{x}),1) =' 6') &\vee \neg SubjectLevel(x,\text{"先端"})
\end{aligned}
$$

図 3.13: 1 回目の合成対象 2

$$
\begin{aligned}
\vee(charAt(\mathbf{subjectName}(\mathbf{x_i}),1) = charAt(prev\_data(selectu from\\
UserScoreuwhereu.user.username =: username, i, course.coursename),1))
\end{aligned}
$$

今回は、機能要求手順の後半 ($\vee$ より後) の等式を、科目番号の規定側の等式 ($\vee$ より前) に代入する。合成の結果は図 3.14 のとおりとなる。これを「合成結果 1」と呼ぶこととする。

続いて、合成結果 1 と研究計画提案書提出の要件 2(図 3.4) の 1 つ目とを合成する。後者を図 3.15 に抜き出した。図 3.14、図 3.15 の中の強調した部分が消去しようとしている述語である。今回は、次の一般的な推論を用いて、この述語を消去する。

$$((A \to B) \wedge (B \to C)) \Rightarrow (\neg A \vee C)$$

これは、以下の推論と等価である。

$$((\neg A \vee B) \wedge (\neg B \vee C)) \Rightarrow (\neg A \vee C)$$

$B$ を現在消去しようとしている述語、$A$、$C$ をその他の要素と考え、$\Rightarrow$ より前を変換前、それより後を変換後として適用する。組み合わせは、まず図 3.14 のグループ 1 の各論理式と、図 3.15 のグループ 1 との各論理式の組み合わせがある。もう 1 つ、図 3.14 のグループ 2 の各論理式と、図 3.15 の論理式 2 との組み合わせがある。こちらは、後者の論理式に、前者のグループの各論理式を合成する。

合成結果 1
グループ 1(この形の論理式が 5 つ)

$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\vee\neg('0' = charAt(prev\_data(selectu fromUserScoreu$$
$$whereu.user.username =: username, i, course.coursename), 1)$$
$$\vee\mathbf{SubjectLevel}(\mathbf{x_i}, \text{"}\mathbf{共通}\text{"})$$

グループ 2(この形の論理式が 5 つ)

$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\vee('0' = charAt(prev\_data(selectu fromUserScoreu$$
$$whereu.user.username =: username, i, course.coursename), 1)$$
$$\vee\neg\mathbf{SubjectLevel}(\mathbf{x_i}, \text{"}\mathbf{共通}\text{"})$$

(注) 科目番号の規定関連の 10 個の論理式それぞれを合成したので、実際には合成結果はいずれのグループにもあと 4 つ存在する。

図 3.14: 2 回目の合成対象 1

研究計画提案書提出要件 2 の 1 つ目 (乗法標準形に変換して分割してある)
グループ 1

$$\neg Passed(x) \vee \neg\mathbf{SubjectLevel}(\mathbf{x}, \text{"}\mathbf{導入}\text{"}) \vee GotAsIntroFundProAdv(x)$$
$$\neg Passed(x) \vee \neg\mathbf{SubjectLevel}(\mathbf{x}, \text{"}\mathbf{基幹}\text{"}) \vee GotAsIntroFundProAdv(x)$$
$$\neg Passed(x) \vee \neg\mathbf{SubjectLevel}(\mathbf{x}, \text{"}\mathbf{専門}\text{"}) \vee GotAsIntroFundProAdv(x)$$
$$\neg Passed(x) \vee \neg\mathbf{SubjectLevel}(\mathbf{x}, \text{"}\mathbf{先端}\text{"}) \vee GotAsIntroFundProAdv(x)$$

論理式 2

$$\neg GotAsIntroFundProAdv(x) \vee \mathbf{SubjectLevel}(\mathbf{x}, \text{"}\mathbf{導入}\text{"})$$
$$\vee\mathbf{SubjectLevel}(\mathbf{x}, \text{"}\mathbf{基幹}\text{"}) \vee \mathbf{SubjectLevel}(\mathbf{x}, \text{"}\mathbf{専門}\text{"}) \vee \mathbf{SubjectLevel}(\mathbf{x}, \text{"}\mathbf{先端}\text{"})$$

$$\neg GotAsIntroFundProAdv(x) \vee Passed(x)$$

図 3.15: 2 回目の合成対象 2

合成結果 2
グループ 1 (実際には、'0' の部分を'1'、'2'、'4'、'6' に変更した各論理式が存在する。)

$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\vee\neg('0' = charAt(prev\_data(selectu fromUserScoreu$$
$$whereu.user.username =: username, i, course.coursename), 1))$$
$$\vee\neg Passed(x_i) \vee \mathbf{GotAsIntroFundProAdv(x_i)}$$

論理式 2

$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\vee('0' = charAt(prev\_data(selectu fromUserScoreu$$
$$whereu.user.username =: username, i, course.coursename), 1))$$
$$\vee[\text{中略}]$$
$$\vee('6' = charAt(prev\_data(selectu fromUserScoreu$$
$$whereu.user.username =: username, i, course.coursename), 1))$$
$$\vee\neg\mathbf{GotAsIntroFundProAdv(x_i)}$$

図 3.16: 3 回目の合成対象 1

研究計画提案書提出要件 2 の 2 つ目 (乗法標準形に変換して分割してある)

$$\neg\mathbf{GotAsIntroFundProAdv(x)} \ \vee \ (partof(introFundProAdvNumber, x) = 1)$$
$$\mathbf{GotAsIntroFundProAdv(x)} \ \vee \ (partof(introFundProAdvNumber, x) = 0)$$

図 3.17: 3 回目の合成対象 2

合成の結果は図 3.16 のとおりとなる。グループ 1 が前者の合成結果、論理式 2 が後者の合成結果である。これを「合成結果 2」と呼ぶこととする。

続いて、合成結果 2 と研究計画提案書提出の要件 2(図 3.4) の 2 つ目とを合成する。後者を図 3.17 に抜き出した。図 3.16、図 3.17 の中の強調した部分が消去しようとしている述語である。2 回目の合成と同じ推論を用いる。合成の結果は図 3.18 のとおりとなる。これを「合成結果 3」と呼ぶこととする。

続いて、合成結果 3 と、機能要求手順 2 とを合成する。後者を図 3.19 に抜き出した。後者は、乗法標準形への変換により 2 つの論理式に分けられたが、図 3.19 における 1 つ目の論理式のみが合成の対象となる。前者は、消去したい述語を含む 1 つ目の論理式のみを対象とするが、もう 1 つの論理式はそのまま次に引き継ぐ。これを 2 回目の合成と同じ推論を用いて合成すると次のとおりとなる。

$$\begin{aligned}
&\neg Prev_MethodCall(checkPointAction.checkKenkyu)\\
&\ \vee\neg('0' = charAt(prev_data(selectufromUserScoreu\\
&\quad whereu.user.username =: username, i, course.coursename), 1))\\
&\ \vee(partof(introFundProAdvNumber, x_i) = 1)\\
&\ \vee\neg(prev_data(selectufromUserScoreuwhereu.username =\\
&\quad : username, i, this) \neq null)
\end{aligned}$$

この式には $prev\_data$ 関数を用いた等式と不等式が含まれる。この等式が成り立つ場合には、不等式は必ず成り立つ。この等式、不等式をそれぞれ $A$、$B$、その他の項をまとめて $C$ としたとき、この論理式は $\neg A \vee \neg B \vee C$ と表される。これを変形すると、$\neg(A \wedge B) \vee C$ となる。$A \rightarrow B$ が成り立つから、$A \wedge B$ は $A$ と等価である[1]。従って、この式は $\neg A \vee C$ に書き換えることができる。合成の結果は図 3.20 のとおりとなる。これを「合成結果 4」と呼ぶこととする。

続いて、合成結果 4 と、研究計画提案書提出の要件 2(図 3.4) の 3 つ目とを合成する。

[1] このことは、自然演繹により証明可能である

合成結果 3
(実際には下記の 1 つ目の論理式の類型として、'0' の部分を'1'、'2'、'4'、'6' に変更した各論理式が存在する。)

$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\vee\neg('0' = charAt(prev\_data(selectufromUserScoreu$$
$$whereu.user.username =: username, i, course.coursename), 1))$$
$$\vee\neg\mathbf{Passed(x_i)} \vee (partof(introFundProAdvNumber, x_i) = 1)$$

$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\vee('0' = charAt(prev\_data(selectufromUserScoreu$$
$$whereu.user.username =: username, i, course.coursename), 1))$$
$$\vee[\text{中略}]$$
$$\vee('6' = charAt(prev\_data(selectufromUserScoreu$$
$$whereu.user.username =: username, i, course.coursename), 1))$$
$$\vee(partof(introFundProAdvNumber, x_i) = 1)$$

図 3.18: 4 回目の合成対象 1

機能要求手順 2(乗法標準形に変換して分割してある)

$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\vee\neg(prev\_data(selectufromUserScoreuwhereu.username =$$
$$: username, i, this) \neq null) \vee \mathbf{Passed(x_i)}$$

$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\vee(prev\_data(selectufromUserScoreuwhereu.username =$$
$$: username, i, this) \neq null) \vee \neg\mathbf{Passed(x_i)}$$

図 3.19: 4 回目の合成対象 2

合成結果 4
(実際には下記の 1 つ目の論理式の類型として、'0' の部分を'1'、'2'、'4'、'6' に変更した各論理式が存在する。)

$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\vee\neg('0' = charAt(prev\_data(selectu fromUserScoreu$$
$$whereu.user.username =: username, i, course.coursename), 1))$$
$$\vee(partof(introFundProAdvNumber, x_i) = 1)$$

$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\vee('0' = charAt(prev\_data(selectu fromUserScoreu$$
$$whereu.user.username =: username, i, course.coursename), 1))$$
$$\vee[\text{中略}]$$
$$\vee('6' = charAt(prev\_data(selectu fromUserScoreu$$
$$whereu.user.username =: username, i, course.coursename), 1))$$
$$\vee(partof(introFundProAdvNumber, x_i) = 1)$$

図 3.20: 4 回目の合成結果

研究計画提案書提出要件 2 の 3 つ目 (乗法標準形に変換して分割してある)

$$\neg(introFundProAdvNumber \geq 6) \vee \mathbf{ProposalPresentCondition_1}$$
$$(introFundProAdvNumber \geq 6) \vee \neg\mathbf{ProposalPresentCondition_1}$$

図 3.21: 5 回目の合成対象 1

機能要求手順 5、6 の 1 つ目 (乗法標準形に変換して分割してある)

$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\vee\neg next\_context(vectorlist, i, pass) \vee \mathbf{ProposalPresentCondition_i}$$
$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\vee next\_context(vectorlist, i, pass) \vee \neg\mathbf{ProposalPresentCondition_i}$$

図 3.22: 5 回目の合成対象 2

後者を図 3.21 に抜き出した。変数 $introFundProAdvNumber$ を消去しようとしている。合成結果 4 の消去しようとしている部分には partof 関数が用いられている。このように、partof 関数が用いられている場合、合成することができない。そのため、この組み合わせでの合成は行わず、後者の論理式を対象として続けていく。なお、partof の使われている変数を処理しようとする際、その変数を引数とする partof 関数呼び出しを含む論理式が、これまでに用いた論理式以外に存在しないことを確認する必要がある。もしも存在するならば、その論理式も今回の合成処理のターゲットの 1 つとみなす必要がある。

続いて、その論理式と、機能要求手順 5、6(図 3.11) の 1 つ目とを合成する。後者を図 3.22 に抜き出した。図 3.21、図 3.22 の中の強調した部分が消去しようとしている述語である。合成の結果は図 3.23 のとおりとなる。これを「合成結果 5」と名付ける。

以上の結果、合成結果 4(図 3.20) と合成結果 5(図 3.23) とを併せて、1 つのとなった。論理式は 2 つに分かれているが、LEIS の動作内容を読み取ることが可能である。これらの論理式は併せて次のような意味を持つ。

- メソッド「checkKenkyu」が実行されたならば、データベースにある修得科目の一覧のうち、科目番号の 2 文字目が 1、2、4、6 のいずれかであるものを数えて、その結果が 6 以上であるかどうかを、コンテキスト変数「vectorlist」の 2 つ目の要素のメンバ「pass」に出力しなくてはならない。

合成結果 5

$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\vee \neg next\_context(vectorlist, i, pass) \vee (introFundProAdvNumber \geq 6)$$
$$\neg Prev\_MethodCall(checkPointAction.checkKenkyu)$$
$$\vee next\_context(vectorlist, i, pass) \vee \neg(introFundProAdvNumber \geq 6)$$

図 3.23: 5 回目の合成結果

# 第4章　版管理モデル

本章では、アカウンタビリティ木、LEIS 関連モデル (社会モデル、機能モデル) に対する版管理モデルについて説明する。

## 4.1　版管理の方式

アカウンタビリティ木、LEIS 関連モデルは XML で表現することとする。XML は、テキストファイルであり、任意のエディタで編集可能である。この特徴を活かすため、新しい版を追加する際には、版データをエディタで作成した上で、版管理システムに登録する方式をとることとする。従って、版管理システムは、新しい版の登録、任意の版の取得、変更点の取得という 3 つの機能のみを持つシンプルなものとなる。

記録方式は、データ量の抑制のため、差分を記録する方式とする。XML の差分の記録方法については、文献 [6] で方法が提案されている。その方法は、第 2 版以降の登録において、新しく登録する XML の版に、直前の版と全く同一の部分木があった場合に、その部分を前の版の部分木への参照要素に置き換える。

しかし、この方式の場合、ある要素の下位に部分木が複数あったとして、そのうちの 1 つでも変更されていた場合、その要素をルートとする部分木に対しては置き換えが適用できない[1]。そこで、変更のなかった複数の部分木が同じ要素の下に連続して存在し、なおかつ、前の版においても、それらの部分木が同じ順番で連続して存在する場合には、「前の版の、この部分木から、この部分木までの参照」という参照要素に置き換えることとする。

それぞれの版データには、内部にあるそれぞれの要素を識別するための情報が必要になる。これは、以下に挙げる理由による。

- 新しい版を登録する際に、差分を取るために、新しい版と直前の版との間で、要素同士の対応関係を取る必要がある
- 変更点の取得機能を実現するために、対応関係を取る必要がある。

なお、理由から分かるように、新しい版については、新しく追加された要素以外についてのみ識別できれば十分である。

[1] 但し、変更のあった部分木を除けば、下位の部分木はいずれも全く変更されていないわけだから、そのそれぞれを参照要素に置き換えることは可能である。

要素の識別のための情報としては、HTML において id 属性が用いられている [7]。id 属性は、識別したい要素の id 属性をユニークな値に設定するものである。この id 属性を新しい版の登録時に版管理システムが自動的に付与し、改版時には、版の作成者は最新版を取り出して変更を加えることとする。付与する値は、重複を避けるため、「(付与したときの版番号).(通し番号)」とする。

以上から、第 1 版を登録する手順は、以下のとおりとなる。

1. アカウンタビリティ木、LEIS モデルの版データをエディタなどで作成する。
2. 版の作成者は、版管理システムの登録コマンドを実行する。
3. 版管理システムは、版データを受け付けて、データベースに記録する。このとき、次の版の登録に備えて、全ての要素に id 属性を付与する。

第 2 版以降を登録する手順は、以下のとおりとなる。

1. 版の作成者は、版管理システムの版取得コマンドを最新版に対して実行する。
2. 版管理システムは、最新版の版データを出力する。
3. 版の作成者は、取り出した版データに、変更を加え、新しい版データとする。
4. 版の作成者は、版管理システムの登録コマンドを実行する。
5. 版管理システムは、版データを受け付けて、1 つ前の版との差分をデータベースに記録する。このとき、新しい要素が出現した場合は、その要素に id 属性を付与する。

最後に、データベースに記録される版データに用いる参照要素の書式を以下のように定める。

- 直前の版の 1 つの部分木を参照する場合

<ref to=”参照先” />

- 直前の版の複数の連続した部分木を参照する場合

<ref begin=”最初の参照先” end=”最後の参照先の最後のインデックス値” />

参照先は、直前の版の最上位 (ルートの 1 つ上) から、目的の部分木のルートまでのパスをピリオドで区切って書く。各階層における要素の選択は 1 から始まるインデックスによるものとする。。

(例) $1^{2}$.2.3.1

[2]XML の最上層には、1 つしか要素を置けないことになっているため、先頭は必ず 1 となる

## 4.2 版データの書式

### 4.2.1 アカウンタビリティ木の表現

アカウンタビリティ木を表現するには、まず対象とするアカウンタビリティ木のゴール木部分が木になっていない場合には、重要でない枝を取り除くことにより、木へと加工する必要がある。この結果は、ゴール (最終目標) が 1 つであれば 1 つの木に、複数あればその数の木になる。その結果、出来上がった木は、そのまま XML の階層構造に変換する。取り除かれた枝は、下位のサブゴールから上位の (サブ) ゴールへのリンクとして表現する。アカウンタビリティ木の版データの基本構造を図 4.1 に示す。

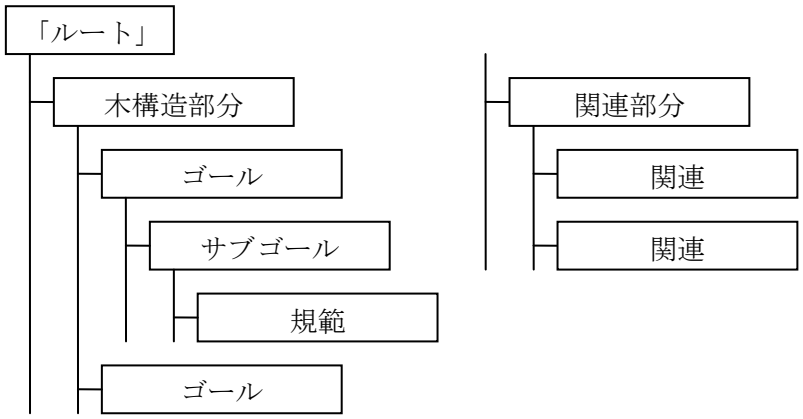


図 4.1: アカウンタビリティ木の版データの基本構造

まず、最上位にルートがある。この部分の書式は、図 4.2 のとおりに定めた。

この書式の trees 要素が、図 4.1 における木構造部分を表しており、ここに前述したそれぞれの木に対応するデータを配置する。ゴールが複数あれば、そのそれぞれに対応する要素を全て、この下に配置する。relations 要素は、関連部分を表しており、ここには、規範間の関連を表すデータが入る。

ゴールの書式について説明する。各 (サブ) ゴールを図 4.3 に示した書式で表現する。goal 要素の下に、その (サブ) ゴールに関する情報を配置する。まず、ゴールの説明を description タグで記述する。下位にサブゴールがあれば、そのサブゴールについても、同じ書式でこの goal 要素の下に記述する。但し、ここでのサブゴールは、前述した要領で一部の枝を取り除いた結果、残ったもののみを含める。もともとのアカウンタビリティ木において、上位の (サブ) ゴールが複数あった場合には、1 つを除いてそれらのゴールとの対応関係が除かれているはずである。そのような場合には、取り除かれたそれぞれに対応する parent タグを配置し、それの nodeId 属性に、対応する goal 要素の id 属性の値を設定する。

もしも、下位が規範であれば、サブゴールの代わりに、次に説明する規範の情報を配置する。規範は、図 4.4 に示した書式で表現する。model 要素の下に、その規範に関する情

```
<accountabilityTree>
    <trees>
        [ゴール (最終目標)(サブゴールの書式に従う)]
        ...
    </trees>
    <relations>
        [関連]
        ...
    </relations>
</accountabilityTree>
```

図 4.2: ルートの書式

```
<goal>
    <description>(サブ) ゴールの説明</description>
    (以下は、下位がサブゴールである場合)
    [下位のサブゴール (中間目標)]
    ...
    (以下は、下位が規範である場合)
    [下位の規範 (規範記述部分の書式に従う)]
    ...

    <parent nodeId="上位の (サブ) ゴールの ID" /> (上位のゴールが複数ある場
合に限り、取り除かれた枝に対応するものを記述する)
    ...
</goal>
```

図 4.3: サブゴールの書式

```
<model type="規範の種類 (本文参照)" kind="義務の種類 (本文参照)">
        (kind 属性は義務規範の場合のみ)
    <object>誰に対して義務を与えるか/誰に対して権限を与えるか
        /何のカテゴリなのか</object>
    <description>どのような [義務/権限/もののこと] なのか</description>
    <kind>どの種類の義務であるか</kind> (義務規範の場合のみ)
    <relation nodeId="関係する関連記述部分の ID"/>
        (関連の情報は別途記述する)
    ...
</model>
```

図 4.4: 規範記述部分の書式

```
<relation>
    <description>関連の説明</description>
    <link nodeId="関係する規範の ID"/>
    ...
</relation>
```

図 4.5: 関連記述部分の書式

報を配置する。まず、type 属性に規範の種類を設定する。値は、義務規範であれば duty、権限規範であれば authority、性質決定規範であれば definition とする。規範間の関連は、図 4.1 の関連部分に別途記述するが、その関連を表す要素へのリンクを relation タグとして記述し、その要素に付いた id 属性の値を nodeId 属性に設定する。

以下は、規範の種類別に説明する。

義務規範の場合は、誰に対して義務を与えるかを object タグとして、どのような義務であるかを description タグとして記述する。また、kind 属性を、義務の種類が命令であれば order に、禁止であれば ban に、許可であれば parmit に、免除であれば donothave に設定する。

権限規範の場合は、誰に対して権限を与えるかを object タグとして、どのような権限であるかを description タグとして記述する。

性質決定規範の場合は、何のカテゴリであるのかを object タグとして、そのカテゴリにどのようなものが含まれるのかを description タグとして記述する。

木構造部分は以上である。続いて、図 4.1 の関連部分の説明に移る。関連部分には、それぞれの関連の情報を配置する。1 つの関連を、図 4.5 に示した書式で表現する。まず、関連の説明を description タグで記述する。次に関係する全ての規範について link タグを配

置し、nodeId 要素を、その規範を表す model 要素の id 要素の値に設定する。

### 4.2.2 LEIS モデルの表現

LEIS モデルの版データの構造は、社会規則の条文や機能要求との対応をとれるように設計した。図 4.6 に基本構造を示す。

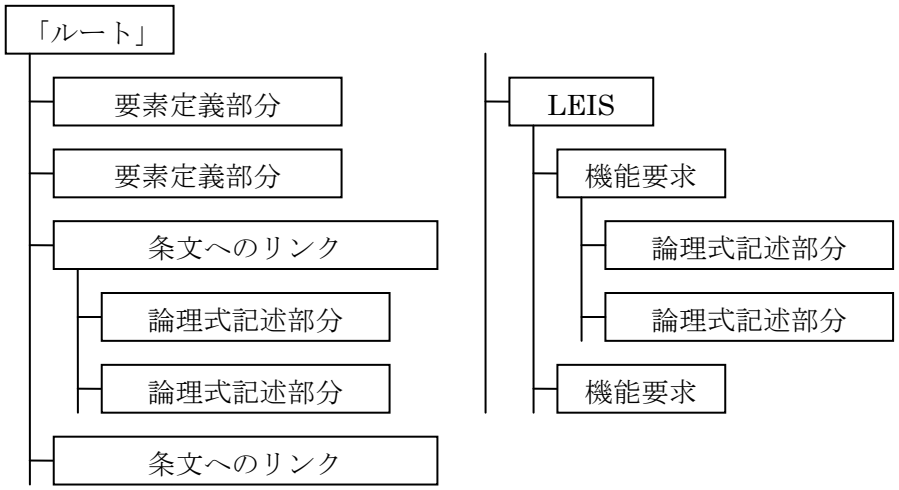


図 4.6: LEIS モデルの版データの基本構造

まず、「要素定義部分」で、各論理式で用いられ、社会モデルの状態を表す各命題、述語、関数、変数を定義する。論理式は、根拠となる条文、機能要求毎にまとめて記述する。「条文へのリンク」は、アカウンタビリティ木の 1 つの規範を指し、その規範を根拠とする全ての論理式を下位の要素に持つ。「LEIS」は 1 つの LEIS を表す。「機能要求」は同様に 1 つの機能要求を表す。「論理式記述部分」は、論理式を 1 つ格納する。以上の構造におけるルート、条文へのリンク、LEIS、機能要求、論理式記述部分の具体的な書式を図 4.7、図 4.8、図 4.9、図 4.10、図 4.11、図 4.12 のとおりに定めた。

論理式は、そのままではテキストで表現できないため、変換を行う。論理符号、比較演算子、量化符号を表 4.1 に示したとおりに置き換える。

```
<classDefinition>
    [条文へのリンクの一覧]
    [LEIS の一覧]
</classDefinition>
```

図 4.7: ルートの書式

```
<element>
    <notation>命題、述語、関数、変数の表記</notation>
    <meaning>この命題、述語、関数、変数の説明</meaning>
</element>
```

図 4.8: 要素定義部分の書式

```
<provision refferenceto="アカウンタビリティ木の葉の識別 ID">
    [論理式記述部分の一覧]
</provision>
```

図 4.9: 条文へのリンクの書式

```
<leis>
    <name>LEIS の名前</name>
    [機能要求の一覧]
</leis>
```

図 4.10: LEIS の書式

```
<function>
    <description>機能の説明</description>
    [論理式記述部分の一覧]
</classDefinition>
```

図 4.11: 機能要求の書式

```
<modelRestriction>
    [論理式]
</modelRestriction>
```

図 4.12: 論理式記述部分の書式

表 4.1: 論理式をテキストで表現するための、符号の変換規則

| 元の論理式、等式・不等式 | 変換後の論理式、等式・不等式 |
|---|---|
| $A \geq B$ | A >= B |
| $A \leq B$ | A <= B |
| $A \neq B$ | A != B |
| $\forall x A(x)$ | Forall(x, A(x)) |
| $\exists x A(x)$ | Exists(x, A(x)) |
| $A \wedge B$ | A AND B |
| $A \vee B$ | A OR B |
| $\neg B$ | Not(B) |
| $A \rightarrow B$ | A THEN B |

# 第5章　結論

本研究では、LEIS の進化を支援するため、社会規則に対応する「社会モデル」と、機能要求に対応する「機能モデル」を定義し、両者から LEIS 全体の動作モデルとして「LEIS モデル」を導出する方法を提案した。また、アカウンタビリティ木、各モデルの版管理モデルを定義した。

上記のモデルを検証するため、JAIST 履修規則と履修管理システムについて、それぞれ社会モデル、機能要求モデルを記述し、LEIS モデルを導出した。その結果、これらの論理式の記述が可能であり、履修管理システムに求められる動作内容を導けることが示された。

今後の課題として以下が挙げられる。

- モデル化の、JAIST 履修規則、履修管理システム以外への適用
  本研究では、JAIST 履修規則および履修管理システムを対象として検証を行った。しかし、社会規則、LEIS には様々な種類のものがあり、それら全てに適用できるかは不明である。従って、それらに対して検証を行い、本研究で述べたモデルの定義が最適なものであるかを調べる必要がある。

- LEIS モデル導出の自動化
  本研究では、LEIS 全体の動作に対する論理式を導出するために、社会モデル、機能モデルの論理式を合成して、新しい論理式および、根拠となる社会規則を導出する方法を考案した。しかし、この手順は煩雑である。そこで、論理式の導出を自動化することが望まれる。

# 謝辞

本研究について主指導教員としてご指導くださいました落水浩一郎教授に深く感謝いたします。

審査員として助言をくださいました鈴木正人准教授、青木利晃特任准教授に感謝いたします。

本研究を進めるにあたって相談にのっていただき、助言をくださった早坂良氏に感謝いたします。

本論文を執筆するにあたってご協力をくださった齋藤彰儀氏に感謝いたします。

# 参考文献


[1] 片山卓也. 検証進化可能電子社会-情報科学による安心な電子社会の実現-. 情報処理, vol. 46, No. 5, pp.515-521, 2005.

[2] 秋山裕俊. Law Enforcing Information System にソフトウェアアカウンタビリティ機能を追加する機構の研究. 北陸先端科学技術大学院大学情報科学研究科 修士論文 2008/3.

[3] 山本修一郎. 要求を可視化するための要求定義・要求仕様書の作り方. ソフトリサーチセンター, 2006.

[4] T. エックホフ, N.K. ズンドビー. 法システム. ミネルヴァ書房

[5] 早坂良、藤枝和宏、落水浩一郎. アカウンタビリティおよび進化容易性を持つ履修管理システムの設計. 日本ソフトウェア科学会第 22 回大会論文集.

[6] Shu-Yao Chien, Vassilis J. Tsotras, Carlo Zaniolo. Efficient Management of Multversion Documents by Object Referencing. Proceedings of the 27th International Conference on Very Large Data Bases.

[7] World Wide Web Consortium. Links in HTML documents. http://www.w3.org/TR/html401/struct/links.html.