JAIST Repository
https://dspace.jaist.ac.jp/

Title	電子大学の履修管理システムを対象とした自己説明性および進化容易性を実現するためのソフトウェア構成手法の検討
Author(s)	早坂，良；藤枝，和宏；落水，浩一郎
Citation	Research report (School of Information Science, Japan Advanced Institute of Science and Technology), IS-RR-2005-005: 1-8
Issue Date	2005-03-28
Type	Technical Report
Text version	publisher
URL	http://hdl.handle.net/10119/8437
Rights	
Description	リサーチレポート（北陸先端科学技術大学院大学情報科学研究科）

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

電子大学の履修管理システムを対象とした自己説明性および進化容易性を実現するためのソフトウェア構成手法の検討

早坂 良, 藤枝 和宏, 落水 浩一郎

2005 年 3 月 28 日
IS-RR-2005-005

北陸先端科学技術大学院大学
情報科学研究科
〒 923-1292 石川県能美市旭台 1-1
{ryoh,fujieda,ochimizu}@jaist.ac.jp

©Ryo HAYASAKA, 2005

ISSN 0918-7553

電子大学の履修管理システムを対象とした自己説明性および進化容易性を実現するためのソフトウェア構成手法の検討

早坂 良　藤枝 和宏　落水 浩一郎
北陸先端科学技術大学院大学 情報科学研究科
923-1292 石川県能美市旭台 1-1
{ryoh,fujieda,ochimizu}@jaist.ac.jp

概要

電子社会システムは, われわれの生活を支えるインフラとしての役割を果たしており, 日増しにその重要性が増している. 本研究の目的は, 電子社会システムの安心性要件のうち自己説明性および進化容易性を実現するソフトウェア構造を研究することである. 本論文では, 電子大学の履修管理システムを対象として, 自己説明性および進化容易性の定義を行い, ソフトウェア構成手法による実現アプローチについて議論する. 既存のソフトウェア構成手法のうちアスペクト指向プログラミング, サブジェクト指向プログラミング, およびソフトウェアプロダクトラインを用いたフィーチャ指向開発を取り上げ, 自己説明性および進化容易性の実現手法の検討を行う.

1 はじめに

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押し進められている. 行政, 金融, 医療, 交通, 教育, 企業などが電子システム化され, それらが相互に接続されて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. したがって, 電子社会システムは, これまでの情報システムのように要求される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" 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, 自己説明性, 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おり, 本研究ではそのうち**自己説明性 (Accountability)** および**進化容易性 (Evolvability)** を対象とする.

自己説明性とは, 電子社会システムの行った処理結果や提供する機能に関するユーザからの質問に対して, システム自身が説明可能であることをいう. ここで, ユーザとは電子社会システムに関わる役割のことで, 一般利用者, サービス担当者, システム開発・保守担当者の 3 種類考えられる. ユーザは, 処理結果について, 役割に応じた根拠を知ることができ, その結果システムを信頼でき安心して生活できるようになる.

ユーザから電子社会システムに投入された処理依頼は, 人間が介在しない電子社会システム内部でさまざまな条件を判断しながら処理され, 結果がユーザに提示されることになる. ユーザが処理結果についての理由を知りたい場合, 多くの場合において処理過程に人間が介在しないことに加え, 処理過程のすべてを把握している管理者も存在しないため, システム自身がユーザからの質問に答えられることが電子社会システムとして重要である. ユーザからの質問に答えるには, システムに実装されたさまざまな条件・処理内容, およびその根拠 (法令など) を示す必要があるが, 従来の情報システムではそれらがシステム内部のさまざまな部分に分散して実装されていてそれらの間の対応関係が必要以上に複雑になってしまったり, もしくは対応関係が明確に定義されていないため, システム自身が処理

結果の理由を説明することができない.

進化容易性とは, 電子社会システムを取り巻く社会や環境の変化に応じてシステムを容易に変更でき, 外部の社会および環境に適応できることをいう. 社会や環境の変化に対応したシステムの変更が迅速に行われないと, ユーザに不便を強いることになり社会の進化を阻害してしまう. 社会や環境の変化に応じてシステムを迅速に変更可能であればシステムは的確に社会を支援でき, ユーザは安心して生活できるようになる.

一般に電子社会システムは, 外部の社会や環境が定めている規則 (法令など) を満たした上で, 十分なサービスを提供していなければならない. 外部の社会や環境は多かれ少なかれ常に変化しているものだが, 特に電子社会システムの実現の基本となっている規則に変更があった場合には, それに応じて電子社会システムを迅速に進化させていく必要がある. ソフトウェア進化 (Software Evolution) の研究分野ではさまざまなアプローチから研究が行われているが, 本研究では外部の社会や環境が定めている規則と電子社会システムの設計との間に対応関係を定義するアプローチをとる. この対応関係を用いることにより, 変更波及解析が可能になる. つまり, 規則に追加・削除・変更が行われたときに, 変更すべき電子社会システムの部分を明確に示すことができる. さらに, 電子社会システムは, 規則の変更に伴うシステムの変更箇所を局所化するソフトウェア構成をとることが望ましい. これにより, システムの進化にかかるコストを削減できる.

本論文では, 自己説明性および進化容易性を実現する電子社会システムのソフトウェア構成手法を研究するにあたり, 既存のソフトウェア設計・開発手法のうちアスペクト指向プログラミング [5], サブジェクト指向プログラミング [3], ソフトウェアプロダクトラインを用いたフィーチャ指向開発 [2] の 3 つを取り上げ, それぞれの手法を用いると自己説明性および進化容易性がどの程度実現できるのかについての考察結果を述べる.

電子社会システムのドメインとして, 電子大学の履修管理システムを対象として議論を行う. 履修管理システムは, 外部の社会や環境にあたる大学が定めた規則としての履修規定を満たした上で, 学生に対して履修登録や修了要件を満たしているかの確認などのサービスを提供するシステムである.

論文の構成は以下のとおりである. 第 2 節で自己説明性と進化容易性についての定義を行い, これらの要件を満たす履修管理システムの持つべき機能を明確にする. 次に, 第 3 節で, 本研究のとる自己説明性と進化容易性を持つ電子社会システムの実現アプローチについて述べる. 第 4 節で, 前述の 3 つの設計・開発手法について, 履修管理システムにおける自己説明性と進化容易性の実現の可能性を議論する. 第 5 節でまとめと今後の課題を述べる.

2 履修管理システムにおける自己説明性と進化容易性

本論文で取り上げる履修管理システムの概要を述べ, このシステムにおける自己説明性と進化容易性についての定義を行う. 履修管理システムは, 本学の平成 13 年度の履修規定を基にモデル化する.

2.1 履修管理システム

図 1 に簡略化した履修管理システムのユースケース図を示す. 履修管理システムのユースケースは本来もっと多いのだが, 省略する.

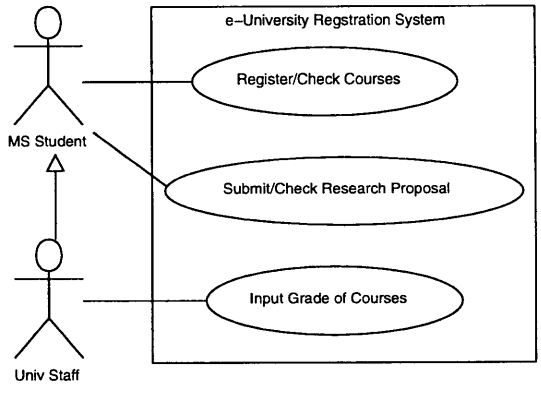

図 1: 履修管理システムのユースケース図

アクタは, 学生 (博士前期課程) と職員である. 学生に関連するユースケースは 2 つあり, 一つめの "Register/Check Courses" は受講したい科目の登録と既に単位を取得している科目の確認である. 二つめの "Submit/Check Research Proposal" は研究計画書の提出と研

究計画書を提出するための要件を満たしているかどうかの確認である. その要件とは,

専門科目 4 分野以上を履修していること.

である. 履修規定により, 講義は専門レベルによって入門・基幹・専門の 3 種類に, 専門分野によって 5 分野に分かれていると定義されている. この要件を満たす単位を取得していないと研究計画書を提出できない.

職員は, 学生に成り代わって上記の学生のユースケースをすべて実行できる. 加えて, 科目担当の教員から指示された学生ごとの成績をシステムに入力する "Input Grade of Courses" ユースケースがある.

2.2 自己説明性

自己説明性とは, 電子社会システムの行った処理内容や提供する機能に関するユーザからの質問に対して, システム自身が説明可能であることである. ユーザは, 一般利用者, サービス担当者, システム開発・保守担当者の 3 種類ある. ユーザを電子社会システムに関わる役割という広い意味で捉え, システム開発・保守担当者まで含めて考える. これにより, 電子社会システムに関わるすべての役割に対して, システムの自己説明性を用いて支援することができる.

ユーザごとに電子社会システムに関わる目的や関心事が異なる. そのため, ユーザごとに異なる内容で自己説明性を実現することが重要である. 一般利用者は, システムの提供するサービスの利用が目的であり, サービスの処理結果の正当性について主な関心がある. サービス担当者は, 電子社会システムを利用して一般利用者へのサービスの提供が目的であり, サービスの基本となっている規則 (法令など) の一貫性や規則やシステムの処理結果をいかに解釈して一般利用者に対してサービスするかに主な関心がある. システム開発・保守担当者は, 電子社会システムを開発し保守するのが目的であり, 規則に基づいた正しいサービスを実現できているか, 規則に変更があったときにその変更を反映するためにシステムのどこを変更すべきか, その変更の範囲はどこまでなのかなどに主な関心がある.

電子大学の履修管理システムの場合, ユーザは, 学生, 職員, システム開発・保守担当者となる. 各ユーザに対する履修管理システムの自己説明性を以下に定義する.

- 学生

  研究計画書の提出の受理／不受理の決定に関して質問があった場合に, なぜその決定が正しいのかについて説明する. 具体的には, 履修規定の該当する研究計画書提出要件を示し, さらに現在学生が修得している科目がどのようにその要件を満たしているか／満たしていないかを示す.

- 職員

  履修管理システムの基となっている履修規定の一貫性に関する質問があった場合に, 一貫性検証の結果を示す.

- システム開発・保守担当者

  履修規定のある項が履修管理システムのどの部分にどのように実現されているか, およびシステム内の処理の実装がどの履修規定の項に基づいているのかに関する質問があった場合に, 履修規定とシステムの仕様, 設計, および実装との間の関連を示す.

この定義は, 履修管理システムに限った場合の自己説明性の基本的な部分の定義であり, 電子社会システム一般を対象とした場合十分なものではない. たとえば, サービス担当者が関心のある, 規則やシステムの処理結果の解釈に関する自己説明性 (過去に行われた解釈の根拠を示すなど) などのより高度な内容の定義は含んでいない.

2.3 進化容易性

進化容易性とは, 電子社会システムを取り巻く社会や環境の変化に応じてシステムを容易に変更でき, 外部の社会および環境に適応できることである.

一般の情報システムと違い, 電子社会システムは, 外部の社会や環境が定めている規則 (法令など) を明示的に満たしていなければならないところに特徴がある. 電子社会システムも一般の情報システムと同様にビジネスロジックを実装しているが, 電子社会システムの仕様には外部の社会や環境が定めている規則が反映されていなければならない. その仕様を基に, 正確に設計・実装されている必要がある.

外部の社会や環境が定めている規則が変化すると, それに応じて電子社会システムも変化しなければならな

い. つまり, 社会の進化と呼応して電子社会システムも進化していく必要がある. したがって, 進化容易性は, 以下の 3 つの要件が必要である.

- 追跡可能性

  外部の社会や環境が定めているそれぞれの規則が, 電子社会システムの仕様／設計／実装のどの部分に対応しているのかが明らかである. これにより, 容易にシステムが明示的に規則を満たしていることが確認できる.

- 変更波及解析

  外部の社会や環境が定めている規則が変化したとき, 対応するシステムの仕様／設計／実装のどの部分まで変更しなければならない可能性があるのかを示すことができる. これにより, システムを進化させるためのコストの見積もりを行うことができる.

- 変更範囲の局所化

  外部の社会や環境が定めている規則が変化した場合に, 対応するシステムの変更を最小限にするソフトウェア構成となっている. これにより, システムの進化のためのコストを削減できる.

電子大学の履修管理システムの場合, 外部の社会や環境が定めている規則が履修規定にあたり, ビジネスロジックが図 1 における各ユースケース (を詳細に記述したもの) にあたる.

第 2.1 節で説明した平成 13 年度の履修規定における研究計画書を提出するための要件は, 平成 14 年度の履修規定では以下のように改定されている.

> 導入・基幹・専門講義科目から 4 分野 6 科目 (12 単位) 以上 (ただし, 導入講義科目は 3 科目まで) を修得していること.

この履修規定の改変に応じて, 上記の追跡可能性, 変更波及解析, および変更範囲の局所化が可能であることを電子大学の履修管理システムの変更容易性と定義する.

3 自己説明性および進化容易性の実現アプローチ

本研究では, ソフトウェア構造により自己説明性および進化容易性を実現するアプローチをとる. このソフトウェア構造を造り出すソフトウェア開発法の提案も行う. 電子社会システムの進化容易性の実現のためには, 開発の初期段階から保守, さらに進化までのすべてのソフトウェアライフサイクルの支援が必要となる. ソフトウェア開発法からの本研究のアプローチは, すべてのライフサイクルの支援が可能であり, 有効であると考えられる.

以下, 本研究のとるアプローチを自己説明性および進化容易性それぞれについて述べる.

3.1 自己説明性の実現アプローチ: ユースケース定義

自己説明性を持つ電子社会システムは, 行った処理に関するユーザからの質問に答えることができる. これを自己説明機能と呼ぶことにする. 自己説明機能はユーザが利用する機能なので, ユースケース駆動ソフトウェア開発では, 自己説明機能を使用して質問に答えるというユースケースを定義する必要がある. 図 2 に, これを考慮した履修管理システムのユースケース図を示す.

図 1 と異なっているのは, 履修管理システムとしての機能を提供する 3 つのユースケースをそれぞれ extend する, 自己説明性に関するユースケースがあるという点である. extend は, ある一定の条件を満たしたときに拡張ユースケースが取り込まれ実行されるという関係である. 履修管理システムとしての機能を提供するユースケースの実行の終わりにはユーザに処理結果を提示するわけだが, その際ユーザから質問があった場合には, 対応する拡張ユースケースである自己説明性に関するユースケースが実行され, ユーザは処理結果に関する質問の回答を得ることができる. ユーザからの質問がなかった場合には, 自己説明性に関するユースケースは実行せずに, 履修管理システムとしての機能を提供するユースケースが単に終了する.

たとえば, 学生が "Submit/Check Research Proposal" ユースケースを実行して研究計画書を提出しようとし

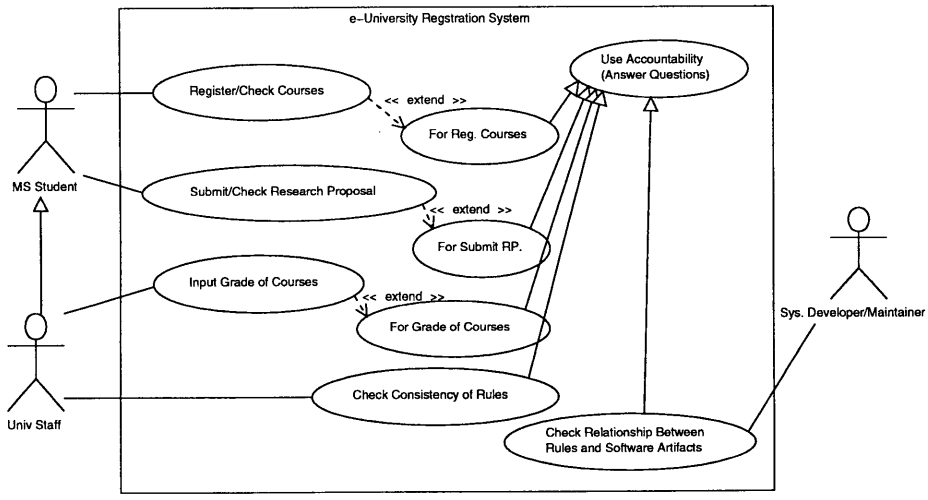

図 2: 自己説明性を持つ履修管理システムのユースケース図

たが, 履修管理システムは学生の履修済み科目を確認して, 不受理の決定を学生に提示したとする. その時, 学生がなぜ不受理の結果になったのか理由を知るために, システムに説明を求める. そうすると拡張ユースケースであり, "Use Accountability (Answer Questions)" ユースケースを継承している "For Reg. Courses" ユースケースが実行され, このユースケースの自己説明機能により, ユーザには履修規定に定められている研究計画書提出の要件 "専門科目 4 分野以上を履修していること" を示し, かつ実際にユーザが履修済みの科目とその分野をリストアップして, 研究計画書提出の要件を満たしていないことを具体的に説明する.

職員およびシステム開発・保守担当者は, 第 2.2 節で定義した自己説明機能を提供するユースケース "Check Consistency of Rules" および "Check Relationship Between Rules and Software Artifacts" をそれぞれ直接呼び出す.

履修管理システムの要求定義の段階から, 明確にアクタに応じた自己説明機能を考慮したユースケース定義を行う本アプローチの利点は, ユースケース図によって自己説明を持つ履修管理システムの機能分析を行うことができ, ユースケース駆動ソフトウェア開発法を基本として用いてシステムを開発できる点にある.

図 2 のユースケース図を基に設計を行うわけだが, 学生に対する自己説明性については, その手法を第 4 節で議論する. 職員およびシステム開発・保守担当者に対する自己説明性については, 自己説明機能を実現するためには以下に述べるさらなる研究を必要とする.

職員に対する "Check Consistency of Rules" ユースケースについては, 履修規則が自然言語 (日本語) で記述してあり, 各項の主張を論理式などの形式に変換し, その上で一貫性に関する推論を行い矛盾があればそれを検出しなければならない. 開発・保守担当者に対する "Check Relationship Between Rules and Software Artifacts" ユースケースについては, ソフトウェア開発の過程でつくられたさまざまな成果物 (仕様書, UML 設計図, 実装コード, ドキュメントなど) と履修規定との関連を保持するデータベースを構築し, そのソフトウェア開発で開発された製品としてのソフトウェアからそのデータベースを用いることによって, 第 2.2 節で定義した履修規定と成果物との間の対応関係を示すという, 開発・保守担当者に対する自己説明性を実現する必要がある.

3.2 進化容易性の実現アプローチ: フレームワークとコンポーネント

進化容易性を持つ電子社会システムは, 外部の社会や環境が定めている規則の改定に応じてシステムを容易に変更できる. これを実現するためには, 以下の 2 つのキーポイントがある.

一つめは, 規則とシステムの開発過程で作られる成

果物 (プログラムコードなど) との関連をどのように定義し, 両者がそれぞれ改定されていくときに関連をどのように維持するかである. たとえば, 履修管理システムの場合, 履修規定とシステムの開発過程で作成される成果物との関連である. このとき, 関係の維持が可能なように考慮した関係の定義を行う必要がある.

二つめは, 規則および成果物のバージョン管理の必要性である. 電子社会システムの進化を考えると, 外部の社会や環境が定めている規則に応じてシステムを変更していくだけでは不十分であり,n 古くなった規則や成果物も必要に応じて使用できなければならない. なぜならば, どのバージョンの規則でユーザからのリクエストを処理するのかが重要だからである. たとえば, 履修管理システムの場合, 学生の入学年度によって適用する履修規定のバージョンが異なる. 普通, 平成 13 年度入学の学生が 2 年生になっても, 平成 14 年度ではなく平成 13 年度の履修規定が適用されなければならない.

図 3 に進化容易性の実現アプローチの概念図を示す. 点線の上が時間軸における規則の進化を, 点線の下が電子社会システムのシステム構成をあらわしている. 各バージョン (年度) の規則は, 対応するバージョンのコンポーネントおよび成果物と対応づけられている.

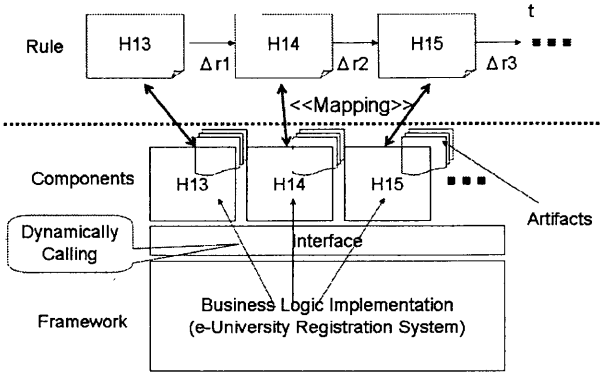

図 3: 進化容易性の実現アプローチ

履修管理システムを例に図 3 を説明する. 履修規定は改定ごとにバージョン管理を行い, 変更差分を記録する. このバージョンと対応するシステムの実装 (コンポーネント) および成果物との間に関連を作り, 維持する. .

履修管理システムは, 図 1 のユースケースで示したビジネスロジックを実装しており, その部分をオブジェクト指向・フレームワークとして構成する. 履修規定の各バージョンによって実装に影響を受ける部分をそれぞれコンポーネント化する. すなわち, 履修規定のバージョンごとに, その規定に基づいたコンポーネントを作成する. フレームワークからは, 学生の入学年度といった条件に応じて, 適切なコンポーネントを動的に呼び出し処理を行うことができる.

本アプローチは, 第 2.3 節で定義した進化容易性の 3 つの要件を満たすための基本的なシステム構成の枠組みとすることができると考えられる. 追跡可能性は, バージョン管理された規則と対応するコンポーネントとの間に定義されている関連を用いることにより実現できる. 変更波及解析は, 関連をつける両端のものの粒度に大きく依存するが, 粒度が小さいほど正確な変更波及解析ができる. 加えて, コンポーネント内の構成要素間の関連づけも必要である. 変更範囲の局所化は, 履修管理システムを不変部であるフレームワークと規則のバージョンによって可変部であるコンポーネントに分離することにより, 規則の変更の際のシステムの変更を一つのコンポーネントの中のみにカプセル化することが可能になる.

4 自己説明性および進化容易性の実現のための既存のソフトウェア構成手法の考察

本節では, 既存のソフトウェア構成手法であるアスペクト指向プログラミング, サブジェクト指向プログラミング, およびソフトウェアプロダクトラインを用いたフィーチャ指向開発の 3 つの手法に関して, 第 3 節での議論の観点から, 自己説明性と進化容易性を実現するためのソフトウェア手法にについて議論する.

4.1 アスペクト指向プログラミング

アスペクト指向プログラミングは, アスペクト (aspect) をプログラムのモジュール化の単位としてプログラミングを行う手法である. アスペクトは, クラスの継承構造を横断する複数のクラスから成っており, より抽象度の高いレベルからモジュール化を行うことができ

る. 従来行われてきた機能分割によるソフトウェア設計を行うと, 複数のクラスに類似するプログラムコードが分散して現れてしまい, 開発や保守に支障をきたす場合があった. それらを横断的関心事という観点からモデル化し, アスペクトとしてモジュール化することにより, 再利用性を向上させることができる.

一般に, オブジェクト指向プログラミングをベースとしてアスペクト指向プログラミングを支援するために, 専用のプログラム処理系を使用する. オブジェクト指向プログラミング言語 Java においては, AspectJ [1] が有名である.

アスペクトとして分離して記述したコードは, コンパイル時にプログラムコードに合成する必要がある. プログラムコード側にはジョインポイントと呼ばれる機能を拡張するフックが用意されいる. ジョインポイントは, メソッド呼び出し, フィールドのアクセス, またはオブジェクト生成などの特定のプログラムの実行のポイントで, アスペクトに記述したコードをそこで実行できる.

この処理系のメカニズムを利用すると, 既存のコードの変更なしで容易にロギング機能を実現できる. たとえば, 以下のコードは, Java プログラム内のすべてのメソッドの実行前に, そのメソッドのシグネチャを表示するアスペクトの定義である.

```
public aspect Log {
  pointcut method():
    execution(* *.*(..));
  before(): method() {
    System.out.println
      (thisJoinPoint.getSignature());
  }
}
```

ロギングアスペクトは, 第 3.1 節で議論した学生に対する自己説明性の実現の基本手法として応用できると考えられる. ロギングアスペクトに使用して, "Submit/Check Research Proposal" ユースケースの実行ログをとり, データベースに記録する. このログには, システムの処理の実行パスと, 研究計画書の受理／不受理を判断するための学生のデータ (履修済み科数など) および判断条件 (研究計画書提出要件) が含まれる. 学生から研究計画書の受理／不受理の決定に関する質問があったときに, このログを参照し実際使用されたデータを基に研究計画書提出要件をどのように満たしているか／満たしていないかの説明に利用できる.

4.2 サブジェクト指向プログラミング

サブジェクト指向プログラミングは, ソフトウェア開発およびサブシステムの変更を容易にすることを目的している手法である. この考え方の下では, 一つのシステムは複数のサブジェクトの集合のみで成り立っている. つまり, 一つのサブジェクトは一つのサブシステムを構成し, メインシステムにあたるシステムは存在しない. すべてのサブジェクトが対等である点が上記のアスペクト指向プログラミングにおけるアスペクトと異なる.

サブジェクトとは, ある特定の視点から観たときのオブジェクトの状態や振る舞いの集合である. たとえば履修管理システムの場合, ある特定の視点とは, 図 3 における, 平成 13 年度の履修規則の研究計画書提出要件の実現するコンポーネントおよび平成 14 年度の研究計画書提出要件の実現するコンポーネントからの視点である.

学生の科目履修状況を現わすオブジェクトについて, 前者はどの分野の専門科目をいくつ履修しているかを保持するデータ構造および操作を持っていればいいが, 後者は, 導入・基幹・専門講義を合計した履修科目数および単位数, 履修科目の分野数, 導入講義科目の科目数を保持するデータ構造および操作を持っていなければならない. このように, 異なる視点 (サブジェクト) から同じ役割を持つオブジェクトを観たときに, そのオブジェクトが持っているべき特徴や振る舞いが異なる. サブジェクト指向プログラミングにおいては, 他のサブジェクトに依存せずに個々のサブジェクトごとにサブシステムを作成し, 処理系の提供するサブジェクト統合機能により全体のシステムを構成する.

他のサブジェクトに依存せずにサブジェクトを設計／開発できる点を応用すると, 進化容易性の "変更範囲の局所化" 要件を実現に利用できると考えられる. 上記サブジェクトの説明で使用した履修管理システムの例のとおり, 規則の各バージョンの実現の視点をサブジェクトとしたとき, 履修管理システムのコンポーネントは他のコンポーネントやフレームワークに依存しないで開発が行える. 規則が改定されて新たなコンポーネントを開発する場合でも, その影響範囲は対応するコンポーネントのみに局所化することができる.

4.3 ソフトウェアプロダクトラインを用いたフィーチャ指向開発

ソフトウェアプロダクトラインは, 共通の特徴を持ち, 共通の再利用資産に基づいて作られるソフトウェア製品系列の体系的な開発手法である. 本手法の特徴の一つに, 共通性 (Commonality) および変動性 (Variability) の分析がある. 共通性は再利用資産として用いられ, 変動性は各製品の特徴 (フィーチャ) を産み出すのに用いられる.

共通性および変動性の分析は, フィーチャモデリング [4] によるものが有名である. Gomma の提案している PLUS 法 [2] もフィーチャモデリングを用いている. PLUS 法におけるフィーチャモデリングでは, ユースケースを用いて変動性分析を行った後, フィーチャを Common, Optional, Alternative, Parameterized, Prerequisite, Mutually Inclusive, Mutually Exclusive などに分類し, フィーチャ間の関連を静的にモデル化する. たとえば, ある製品を開発する際に, フィーチャ A を選択したときには同時にフィーチャ B も選択しなければならないという関連や, フィーチャ C とフィーチャ D は必ずどちらか片方を選択しなければならないという関連などを記述することができる.

PLUS 法で行っている上記フィーチャモデリングを変更容易性の変更波及解析の実現に応用できると考えられる. ある履修規定のバージョンで実現すべき機能のフィーチャモデリングを行い, そのフィーチャ間の関連を Alternative, Mutually Exclusive などを使用して静的に記述する. そのフィーチャモデルを基にコンポーネントの実装を行う. その後, 履修規定の改定があり新しいバージョンの履修規定ができたとき, 一つ古い履修規定を基に作成したフィーチャモデルに新しい履修規定の変更点をフィーチャモデリングしたモデルをマージし, その際フィーチャ間に張られている関連を利用して, フィーチャモデル上で変更波及解析を行うことができる. その解析結果を基に, 新しい履修規定に対応するコンポーネントを効率良く開発できる.

5 おわりに

本論文では, 安心できる電子社会システムを構築するため, 自己説明性および進化容易性の実現手法について, アスペクト指向プログラミング, サブジェクト指向プログラミング, およびソフトウェアプロダクトラインを用いたフィーチャ指向開発の検討を行った. 議論を具体的に行うために電子社会システムのドメインのシステムである電子大学の履修管理システムを対象として, 自己説明性および進化容易性の定義を行い, 実現アプローチについて議論を行った. 3 手法の概念やメカニズムを応用すると, 自己説明性または進化容易性のうち, 一部分を実現できる可能性があることを示した.

履修管理システムを対象とした自己説明性および進化容易性の定義および実現手法の検討は, 電子自治体といった他の電子社会システムを対象とした場合には修正や不足している部分があり得る. しかし, 比較的小規模のシステムである履修管理システムを対象として行った本論文の議論は, 基本的で本質的なものであると考えられる.

謝辞

本研究は, 文部科学省科学研究費補助金 (21 COE 特別研究員奨励費) 研究課題 "自己説明性および進化容易性を有する電子自治体シミュレータに関する研究" (採用年度・受付番号: 16・54141) の交付を受けて行われた.

参考文献

[1] AspectJ home page. http://eclipse.org/aspectj/.

[2] Hassan Gomma. *Designing Software Product Lines with UML*. Addison-Wesley, 2004.

[3] William Harrison and Harold Ossher. Subject-oriented programming (A critique of pure objects). In *Proceedings of the OOPSLA '93 Conference on Object-oriented Programming Systems, Languages and Applications*, pp. 411–28. IEEE Comput. Soc, Los Alamitos, CA, October 1993.

[4] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.

[5] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *ECOOP '97 — Object-Oriented Programming 11th European Conference, Jyväskylä, Finland*, Vol. 1241 of *Lecture Notes in Computer Science*, pp. 220–242. Springer-Verlag, New York, NY, June 1997.