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

2005－SE－150（1）
2005／11／29

アカウンタビリティおよび進化容易性を持つソフトウェアアーキテクチャと 3 層モデルとの対応

早坂 良 †　堀 雅和 ††　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
†† インテック・ウェブ・アンド・ゲノム・インフォマティクス株式会社
ryoh@jaist.ac.jp, hori_masakazu@webgen.co.jp,
{fujieda,ochimizu}@jaist.ac.jp

概　要

本論文では，我々が提案しているアカウンタビリティおよび進化容易性を持つソフトウェアアーキテクチャと，広く一般に採用されているユーザインタフェース・プロセス管理・データベース管理から成る 3 層モデルとの対応を示し，提案しているアーキテクチャを用いることにより両性質を満たす 3 層モデルシステムが実現可能であることを示す．ユーザインタフェース層とプロセス管理層の間にインターセプタプロキシを配置することにより，既存の 3 層モデルシステムに大きな変更を加えずにアカウンタビリティ機能を付加できる．対象コンポーネントを置き換えることにより進化容易性を実現する．大学の履修管理システムを対象にプロトタイプ実装を行った．その動作例を示す．

Applying the Software Architecture with Accountability and Ease-of-Evolution to the Three-Tier Model

Ryo HAYASAKA †　Masakazu HORI ††　Kazuhiro FUJIEDA †　Koichiro OCHIMIZU †
† School of Information Science, Japan Advanced Institute of Science and Technology
†† INTEC Web and Genome Informatics Corporation

Abstract

We proposed the software architecture which enables Accountability and Ease-of-Evolution. In this paper, we show that the architecture can apply the commonly-used three-tier model systems, which divide into user interface, process management, and database management. By using Interceptor Proxy which intercepts the messages between user interface tier and process management tier and delegates them to the appropriate objects, the existing three-tier systems can be extended to hold Accountability without a lot of change cost. The extended functions called by Interceptor Proxy are modularized into components each of which encapsulates change requirements so that Ease-of-Evolution is realized by replacing the components. We developed the prototype implementation of university registration systems using the architecture. The implementation illustrates how the system works.

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．政治，行政，司法，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．したがって，電子社会システムは，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」（拠点リーダー：片山卓也） [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**アカウンタビリティ (Accountability)** および**進化容易性 (Ease-of-Evolution)** [*1]を対象としている．

社会には，人間や電子社会システムを含めて，属するものすべてが守らなければならない法令や，会社などの各組織が定めている各種規則があり，これらを社会規則と呼ぶことにする．電子社会システムは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，社会規則に従ってシステムが正しく構築されていることを保証できること，または社会規則に従ってシステムが正しく構築されていることを確認できる必要がある．一方，社会は常に変化しており，社会規則もそれに応じて改定される．電子社会システムは，社会規則の改定に応じてシステムを迅速に進化させていかなければならない．そのためには低いコストでシステムを変更できる必要がある．一般に社会規則は，条・項または章・節などから構成され，自然言語で記述されることが多い．また，システムは一定の組織によって管理・運用される．こういった特徴を持つシステムを片山は **Law-Defined システム**と呼んでいる．電子社会システムは Law-Defined システムの一種であり，本研究では Law-Defined システムを対象として議論を行う．

Law-Defined システムにおける**アカウンタビリティ**とは，さまざまな立場のユーザからの質問に対して，システムが従っている社会規則を根拠として説明可能であることをいう．ここでユーザとは，ある組織に属している Law-Defined システムに関わるステークホルダのことであり，組織外の一般利用者と，組織内の業務担当者，社会規則整備担当者，システム開発・保守担当者の組織内外合わせて 4 種類を考える．ステークホルダはそれぞれシステムに関わる目的や関心事が異なるため，質問内容も異なる．

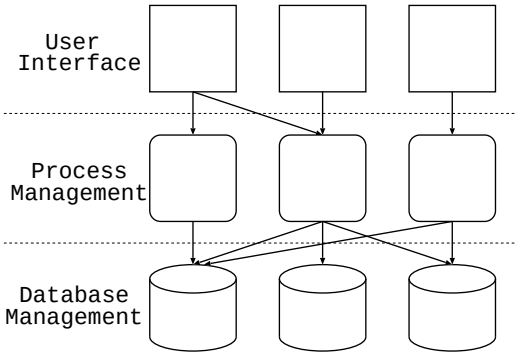

図 1: 3 層モデル

たとえば，一般利用者はシステムによる処理結果がなぜそうなるのかについて，業務担当者は規則をどのように解釈して業務判断を行ったらよいかについて，社会規則整備担当者は，規則が一貫しているか（矛盾はないか）について，システム開発・保守担当者は，規則がシステムのどこにどのように実現されているか，といった質問である．

Law-Defined システムにおける**進化容易性**とは，システムを取り巻く社会や環境の変化に応じて社会規則が改定されていくとき，社会規則の改定に応じてシステムを容易に変更でき，外部の社会環境に迅速に適応できることをいう．

本論文では，アカウンタビリティおよび進化容易性を実現するソフトウェアアーキテクチャと，Web サービスをはじめとして広く一般の情報システムに採用されている 3 層モデル [2] との対応を示す．これは，大学の履修管理システムを対象としたアカウンタビリティおよび進化容易性を実現するソフトウェアアーキテクチャ [3] を，3 層モデルにより設計された履修管理システムに適用することにより行う．図 1 にユーザインタフェース層，プロセス管理層，データベース管理層から成る 3 層モデルを示す．

アカウンタビリティおよび進化容易性を持つ Law-Defined システムを低い実装コストで実現で

[*1] ソフトウェアの進化性に加えて，われわれは進化のための変更コストを低減することも特に重要視しており，これを進化容易性と呼ぶ．

きることは非常に重要である．社会には既に多くの電子社会システムが存在し，これらのシステムに対して両性質を実現するために，本研究では，ユーザインタフェース層とプロセス管理層の間にインターセプタプロキシ [4] を配置するアプローチをとる．インターセプタプロキシは，ユーザインタフェース層とプロセス管理層との間の通信内容を変更することや，通信内容に応じて既存のシステムの外部で提供される機能を呼び出すことができる．これにより，既存の 3 層モデルシステムのうちプロセス管理層およびデータベース管理層に変更を加えることなく，アカウンタビリティ機能を実現できる．また，インターセプタプロキシによって呼び出される機能に対して，社会規則に記述されている各規則ごとにコンポーネントを作成することにより，社会規則が改定された際のシステムの進化に要する変更コストを低減できる．

論文の構成は以下のとおりである．まず，2 節でわれわれが提案しているアカウンタビリティおよび進化容易性を持つアーキテクチャについて述べ，3 節で 3 層モデルとの対応を示す．4 節では，履修管理システムを対象としたプロトタイプシステムの動作例を示す．最後に，5 節でまとめと今後の課題を述べる．

2 アカウンタビリティおよび進化容易性を持つソフトウェアアーキテクチャ

本節では，大学の履修管理システムを対象とした，アカウンタビリティおよび進化容易性を実現するアーキテクチャ [3] について述べる．

2.1 履修管理システムにおけるアカウンタビリティおよび進化容易性の定義

履修管理システムは，学生の履修登録データを管理しているシステムであり，学生からの履修登録要求などを処理する．履修管理システムは，大学が定める履修規則に従って正確に実現されていなければならないので，Law-Defined システムである．

図 2 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[*2]．ユースケースには，履修管理機能に関するものとアカウンタビリティ機能に関するものの 2 種類ある．システムに対する質問はステークホルダごとに異なるので，それぞれに対応したアカウンタビリティ機能をユースケースとしてモデル化している．図 2 中の “Ask the System Question about ...” という名前のユースケースがアカウンタビリティ機能である．

アクタは，学生，教職員 ＜学生課＞，教職員 ＜履修規則整備＞，システム開発・保守者の 4 つを考える．これらは履修管理システムに関するステークホルダである．

ステークホルダごとにシステムに関わる目的や関心事が異なるため，システムに対する質問もそれぞれ異なる．そのため，ステークホルダごとに異なる内容の説明を行う必要がある．各ステークホルダに対する履修管理システムのアカウンタビリティを以下に定義する．

- 学生：“Ask the System Questions about Research Proposal” ユースケース
  研究計画書の提出の受理／不受理の処理結果に関する質問に対して，なぜその決定が正しいのかについて理由を説明する．該当する履修規則を引用して行う．
- 教職員 ＜学生課＞：“Ask the System Questions about Law” ユースケース
  履修規則に関連する質問に対して, 履修規定改定の目的, 変更差分, 変更内容の概略, 変更理由などを説明する．
- 教職員 ＜履修規則整備＞：“Ask the System Questions about Consistency of Law” ユースケース
  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し説明を行う．
- システム開発・保守担当者：“Ask the System Questions about Implementation of Law” ユー

[*2] 履修管理機能は本来，より多くのユースケースを持つが，省略する．

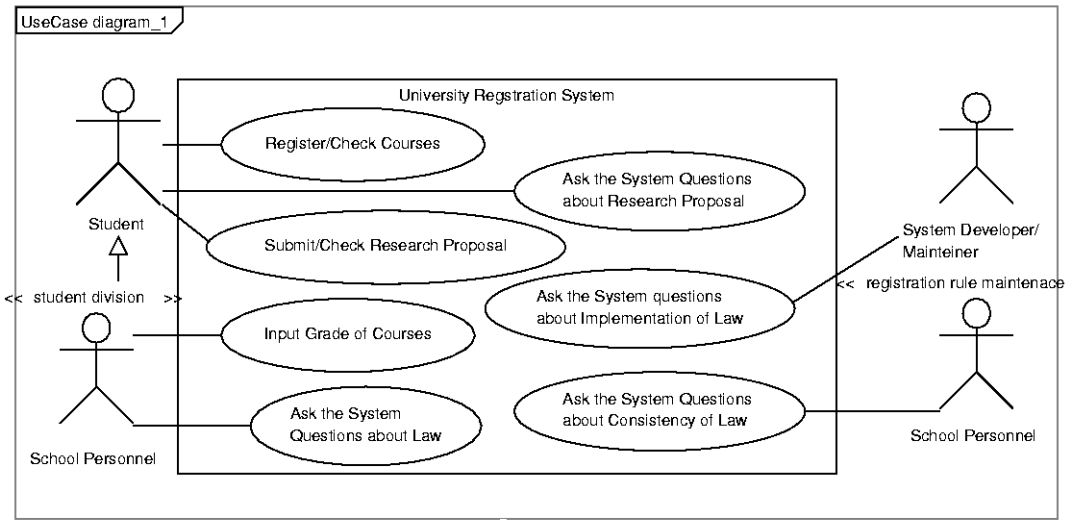

図 2: 履修管理システムのユースケース図

スケース
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, 分析モデル, 設計モデル, および実装モデルなどの要素間に定義されたトレース依存関係を利用して質問に答える.

次に，進化容易性について定義を行う．進化容易性とは，Law-Defined システムを取り巻く社会環境の変化および社会規則の改定に応じてシステムを容易に変更でき，外部の社会環境や社会規則に迅速に適応できることである．そのためには，以下の 3 つの要件が必要である.

- 追跡可能性
  システム外部の社会環境や組織が定めている社会規則および, Law-Defined 情報システムの仕様, 分析モデル, 設計モデル, 実装モデル, ソースコードなどのそれぞれの要素間にトレース依存関係を定義し, どの部分に何が対応しているのかを明らかにする.
- 変更波及解析
  システム外部の社会環境や組織が定めている社会規則が変化したとき, 対応する開発成果物 (仕様, 分析モデル, 設計モデル, 実装モデル, ソースコード, ドキュメントなど) 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.
- 変更範囲の局所化
  システム外部の社会環境や組織が定めている社会規則が変化した場合に, 必要なシステムの変更を最小限にするソフトウェア構成とする. これにより, 変更コストを削減でき, 迅速にシステムを進化させることができる.

2.2 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アーキテクチャ

アカウンタビリティおよび進化容易性を実現する履修管理システムのアーキテクチャを図 3 に示す.

アーキテクチャは，ユーザインタフェースサブシステムと履修管理サブシステムの間にインターセプタプロキシを配置し，ユーザインタフェースが履修管理機能を要求した際にはインターセプタプロキシは履修管理サブシステムを呼び出し，アカウンタビリティ機能を要求した際にはアカウンタビリティサ

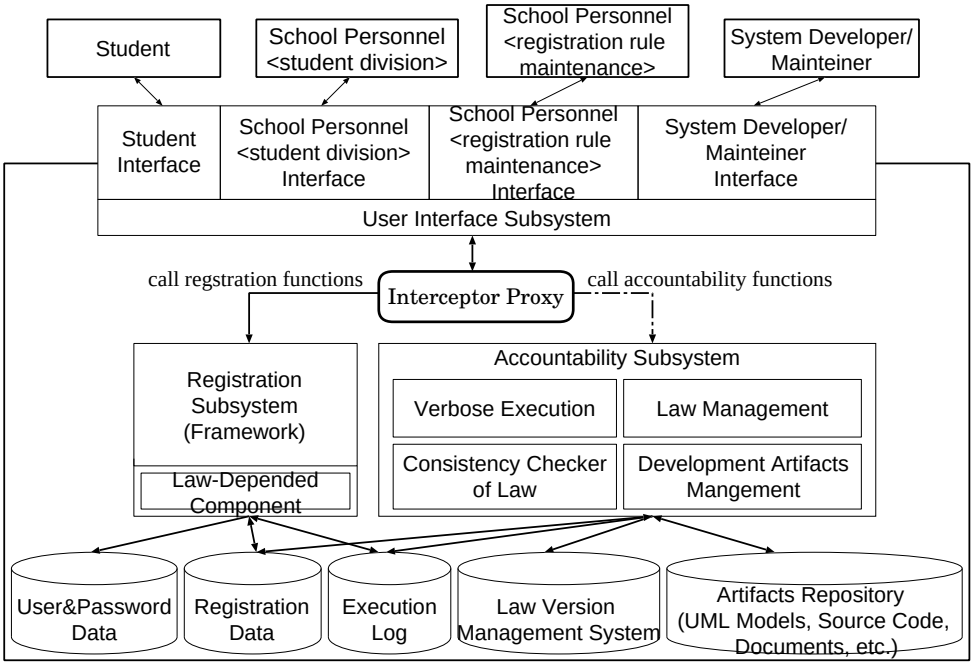

図 3: アカウンタビリティおよび進化容易性を持つ履修管理システムのアーキテクチャ

ブシステムを呼び出す構造になっている．この構造には，インターセプタプロキシを用いることにより既存の履修管理システムに大きな変更を加えることなくアカウンタビリティ機能を追加できる利点がある．

本アーキテクチャの中核の機構であるインターセプタプロキシ [4] の動作の概要を図 4 に示す．インターセプタプロキシは，インターセプトポイントと呼ばれるフックに拡張機能を追加するというインターセプタ [4] のバリエーションのひとつであり，プロキシ [5] の機能を併せ持つものである．プロキシとしてインターセプタプロキシは，ユーザインタフェースに対して履修管理機能およびアカウンタビリティ機能の両インタフェースを提供し，履修管理機能およびアカウンタビリティ機能に対して履修管理機能のインタフェースを提供する．インターセプタとしては，ユーザインタフェースからの呼び出しの際に，履修管理機能もしくはアカウンタビリティ機能の呼び出しの前後に，前処理・後処理を行う機能を提供する．本アーキテクチャの場合，インターセプトポイントは，インターセプタプロキシに実装

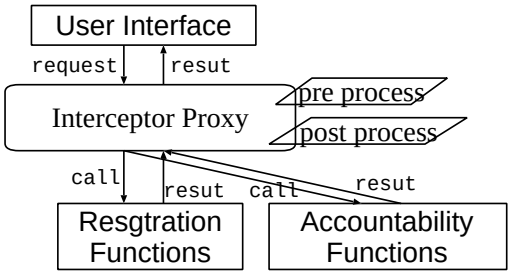

図 4: インターセプタプロキシ

されたアカウンタビリティ機能のインタフェースとなる．

ユーザインタフェースサブシステムは，4 つのステークホルダごと必要な専用のインタフェースを実現する．ユーザインタフェースサブシステムでは，各ユーザインタフェース共通に使用されるユーザ認証，アクセス制御，通信路の暗号化などの機能を提供する．

履修管理サブシステムでは履修管理機能を提供する．履修規則に依存しない処理をフレームワークとして構成し、履修規則に依存する処理をフレーム

ワークコンポーネントとする．この構成にすることにより，履修規則の更新の際にコンポーネントのみ新しいものに交換すればよいので，進化容易性の実現の一助となる．

アカウンタビリティサブシステムでは，ステークホルダごとに異なったアカウンタビリティ機能を実現するために，それぞれに対応したサブシステムを構成する．冗長実行サブシステムは，履修データおよび実行ログを用いて学生に対するアカウンタビリティ機能を実現する．規則整備サブシステムは，履修規則バージョン管理システムを利用して教職員＜学生課＞に対するアカウンタビリティ機能を実現する．規則の一貫性検証サブサブシステムは，履修規則バージョン管理システムを利用して教職員＜履修規則整備＞に対するアカウンタビリティ機能を実現する．開発成果物管理サブシステムは，成果物リポジトリを利用して，システム開発・保守者に対するアカウンタビリティ機能を実現する．

3　3 層モデルとの対応

3 層モデルとの対応を示すために，まず，3 層モデルによって構成した履修管理システムについて述べる．次に，それに対して 2.2 節で述べたアカウンタビリティおよび進化容易性を実現するアーキテクチャを適用する．

3.1　3 層モデルによる履修管理システムの構成

3 層モデルとは，図 1 に示した通り，ユーザインタフェース層，プロセス管理層，データベース管理層の 3 つの論理的な層に分離した構成のことである．3 層モデルは，クライアント・サーバ型の 2 層モデルにおいてビジネスロジックがクライアントおよびサーバの両方に分散してしまう点を改良したものである．

図 5 に 3 層モデルによる履修管理システムの構成を示す．中間層に履修登録ロジックを配置している．2.1 節で述べたように，履修管理システムは，学生の履修登録データを管理しているシステムであり，学生からの履修登録要求などを処理するシステムである．

学生は，まず，自分の ID とパスワードを使って

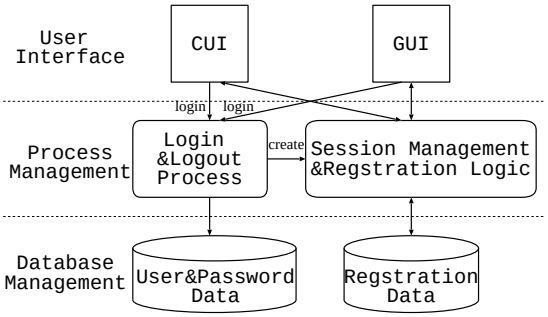

図 5: 3 層モデルによる履修管理システムの構成

システムにログインを行う．ユーザインタフェース層における「コマンドライン」もしくは「グラフィカルユーザインタフェース」からのログイン要求は，プロセス管理層の「ログイン＆ログアウト処理」に送られ，そこでデータベース管理層の「パスワードデータ」を用いて認証が行われる．合っていればその学生に対応する「セッション管理＆履修登録ロジック」が作成され，以後ユーザインタフェースは「セッション管理＆履修登録ロジック」に履修登録要求を送る．学生がログアウトすると，その学生に対応する「セッション管理＆履修登録ロジック」が削除される．

3.2　アカウンタビリティおよび進化容易性を持つアーキテクチャと 3 層モデルとの対応

3.1 節で述べた履修管理システムの構成に，2.2 節で述べたアーキテクチャを適用した図を 6 に示す．

図 3 中のインターセプタプロキシ，履修管理サブシステム，アカウンタビリティサブシステムを中間層とした，ユーザインタフェース (UI)，プロセス管理 (PM)，データベース管理 (DM) から成るの 3 層構造になっている．図 3 と異なる点は，2 点ある．

一つめは，履修管理サブシステムが図 5 のプロセス管理層で置き換わっている点である．履修管理サブシステムが 3 層モデルで構築されているので，プロセス管理層およびデータベース管理層に対して一切変更を加えずに本アーキテクチャにおいて再利用が可能になっている．

二つめは，ユーザインタフェース層に変更が必要である点である．ステークホルダがアカウンタビリ

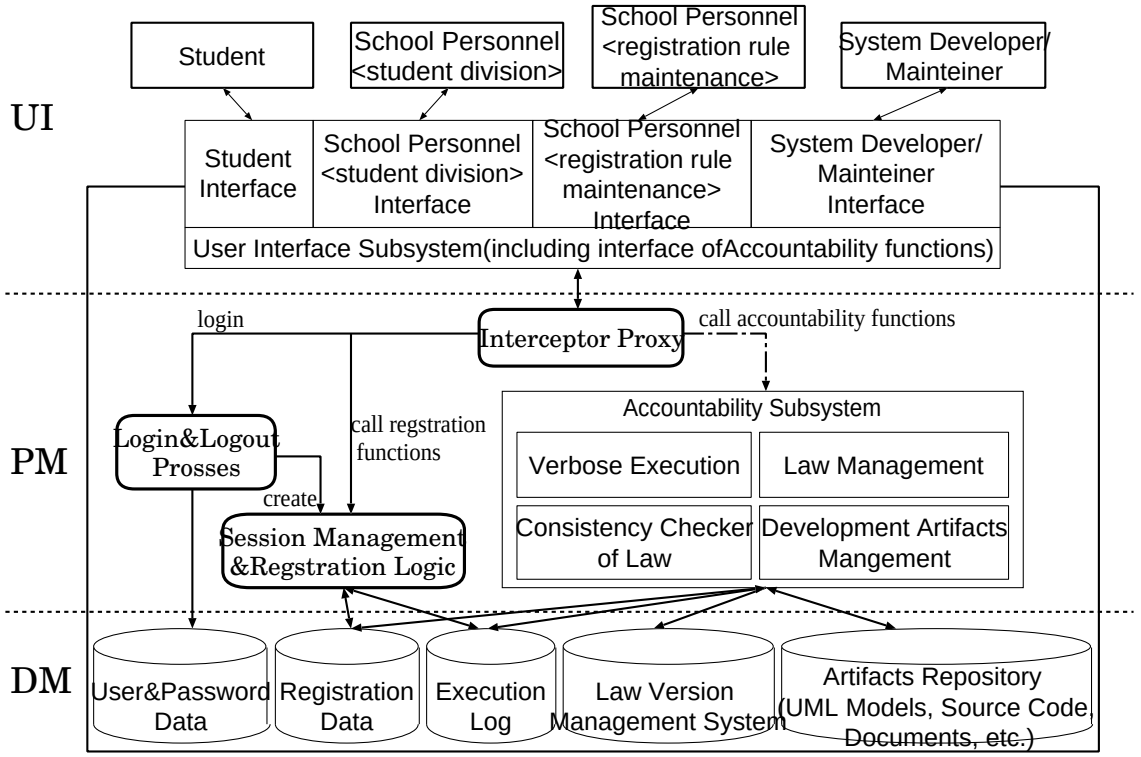

図 6: 3 層モデルで構成された履修管理システムにおけるアカウンタビリティおよび進化容易性を持つアーキテクチャ

ティ機能を利用するには，ユーザインタフェースがアカウンタビリティ機能を使用できるように設計されていなければならない．既存のユーザインタフェース層がラッパー等を使用することにより再利用が可能か，もしくは，スクラッチからの開発が必要になるかに関しては，既存のユーザインタフェース層の設計および実装法に依存する問題である．

インターセプタプロキシは，ユーザインタフェース層から呼び出されるメソッドによって，処理要求を送る先が異なる．例えば，ログイン要求は”Login&Logout Process”に要求が送られ，履修科目登録要求は”Session Management&Regstration Logic”に要求が送られ，研究計画書提出に関するアカウンタビリティ機能要求は，アカウンタビリティサブシステムに要求が転送される．

進化容易性の実現のために，履修規則の各項目ごとに規則コンポーネントを作成している．

以上の議論から，図 3 のアーキテクチャと 3 層モデルとは，うまく対応つけることが可能である．

4 プロトタイプシステムの動作例

4.1 開発法

オブジェクト指向スクリプト言語 Ruby [*3]を使用して，プロトタイプシステムを開発した．コード量は，合計して 400 行弱である．

プロトタイプシステムの開発は，以下のように行った．まず，図 5 の 3 層モデルに基づき，履修管理システムを開発した．このシステムでは，学生がユーザインタフェース (CUI) を用いて，図 2 中の履修管理機能に関するユースケースである”Register/Check Courses”および”Submit/Check Research Proposal”を行うことができる．

次に，3 層モデルの履修管理システムのコードには一切変更を加えずに，図 6 のアーキテクチャに基づいた，アカウンタビリティおよび進化容易性を

[*3] http://www.ruby-lang.org/

持つ履修管理システムを開発した．このシステムを使用すると，学生は上記履修管理機能に関するユースケースに加えて，図 2 中のアカウンタビリティ機能に関するユースケースである”Ask the System Questions about Research Proposal”を行うことができる．

本プロトタイプの場合，3 層モデルのユーザインタフェース層の再利用も可能であった．3 層モデルのユーザインタフェース層の中のそれぞれのステークホルダに対応するインタフェースクラスを継承して，差分プログラミングによりアカウンタビリティ機能のみを新たに実装した．よって，ほとんど実装コストは低く抑えることができた．

4.2 動作例

アカウンタビリティ機能の実行例を示す．研究計画書の提出を試みたが失敗したので，アカウンタビリティ機能を使用して，システムに説明を求めている．

```
# start RishuuManagementSystem
rms = RishuuManagementSystem.new
# student accesss
rmi = StudentAccountableRishuuManagementSysIF
      .new(rms)
puts "[STUDENT LOGIN]"
rmi.login("ryoh", "ryoh's password") do |session|
  session.submit_proposal(proposal)
  # use an accountability function
  puts "[ASK_ABOUT_SUBMIT_PROPOSAL]"
  session.ask_about_submit_proposal
end
puts "[STUDENT LOGOUT]"
```

以下は，上記コードの実行結果である．履修規則を引用 (4.4.3.4.2 節) して，説明を行っている．この機能は，インターセプタプロキシを経由して，アカウンタビリティサブシステムにより実現されている．

```
[SUBMIT_PROPOSAL]
==> "submit proposal failed."
[ASK_ABOUT_SUBMIT_PROPOSAL]
==> "[4.4.3.4.2] Requirements for Submit
     Proposal. The number of Senmon Kamoku
     must be grater than or equal to four."
==> "You have credited 3 Senmon Kamoku
     (Software Design, Operating System,
     Network Software).  Therefore, you
     could not submit proposal."
```

5 おわりに

本論文では，大学の履修管理システムにおける，アカウンタビリティおよび進化容易性を実現するアーキテクチャと 3 層モデルとの対応を示した．また，プロトタイプシステムを実装し，両性質を実現する 3 層アーキテクチャが実現可能であることを確認した．

今後の課題として，3 層以上の n 層モデルの場合や，より大規模で複雑な構成をとる場合，たとえばバス型（共通の通信バスで複数のサブシステムを結合）などの事例についても本アーキテクチャが適用可能かどうか検討していく予定である．

謝辞

本研究は，文部科学省科学研究費補助金（21 世紀 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] 片山卓也：特集 21 世紀卓越した情報研究拠点プログラムの目指す研究 (後編) 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] Schussel, G.: Client/Server: Past, Present and Future (1995), online.

[3] 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, CD-ROM (2005).

[4] Schmidt, D. C., Stal, M., Rohnert, H. and Buschmann, F.: *Pattern-Oriented Software Architecture Volume 2 – Networked and Concurrent Objects*, John Wiley and Sons (2000).

[5] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns: Elements of Reusable Object-oriented Software*, Addison Wesley, Reading (1996).