博 士 論 文

サービス指向システム開発方法論の研究

D2018SE002 山本 里枝子

指導教員 青山 幹雄

2019 年 2 月

南山大学大学院 理工学研究科 ソフトウェア工学専攻

A Service-Oriented Systems Development Methodology

D2018SE002 Rieko Yamamoto

Supervisor Mikio Aoyama

February 2019

Graduate Program in Software Engineering
Graduate School of Science and Engineering
Nanzan University

要約

Web サービスは企業情報システム開発の新たなプラットフォームである．Web サービスを利用した企業情報システム開発における生産性向上と品質確保に関する新たな課題を解決するためにソフトウェア工学の枠組みが求められている．

本研究は，Web サービスを提供・利用するサービス指向システム開発の生産性と品質の向上を目的に，生産性を阻害する課題を明らかにし，その解決を図る開発方法論を提案する．提案する方法論の核技術として，メタモデル駆動の業務プロセスモデリング，Web サービス開発向けパターン体系，Web API の利用者観点での品質モデルを提案する．

メタモデル駆動の業務プロセスモデリングは，業務プロセスモデルの再利用を容易にするために，メタモデル，業務プロセス再利用方法，再利用可能な業務プロセステンプレートに基づく業務プロセスモデリング方法論と，それを支援する統合環境 BPM-IE(Business Process Modeling Integrated Environment)を提案する．業務プロセステンプレートを伴う BPM-IE が，業務プロセスモデルの作成と再利用を支援する．提案した方法と BPM-IE を実際の顧客の企業ソフトウェア開発プロジェクトに適用し，その実証的研究から提案した方法論と BPM-IE が生産性の 46%以上向上に貢献したことを確認した．本研究は業務プロセスモデリングの成果物の再利用を実現することで，サービス指向の要求分析プロセス技術の発展に貢献する．

Web サービス開発のパターン体系は，新しいソフトウェア開発技術の発信の試みとして，Web サービス開発向けの具体的な技術である XML を取り上げて，XML を利用するシステムの開発ノウハウをパターン体系として構築した．本パターン体系を実際の企業システム開発の 50 プロジェクトに適用して一定の効果を確認した．この結果，今後現れる新たな技術要素に対しても，5 つの入口（アーキテクチャ，テンプレートモデル，詳細ノウハウ，コンポーネント，開発方法論）から成るパターン体系が有効であることを示した．また，企業において実践可能なパターン体系の開発プロセスと開発体制も提示した．パターンの概念を Web サービスの再利用へ発展させ，それを通した企業内における新技術の普及に貢献した．

Web API の利用者観点での品質モデルは，Web API が持つインタフェース定義と実装の乖離や，リモートで実行されてユーザと独立に変更される等の特性により，ソフトウェア工学へ提起された新たな問題への品質面からの提案である．Web API の特性を捉えるために，Web API を利用するアプリケーション開発者のパースペクティブから，非機能要求フレームワークを用いて，ユーザビリティの習得容易性と保守性の安定性を品質特性として導出し品質モデルを定義して提案した．提案する品質モデルを，実際の Web API に適用し，ユーザビリティの習得容易性の測定値が開発者による定性的な評価と同じ傾向を示すこと，及び，保守性の安定性が API コンシューマの保守に必要な工数の度合いを示すことから，その有効性を確認した．Web API の利用において新たに重要性が認識されている品質特性のモデル化とその定量的評価方法の確立，ならびに，従来のソフトウェア製品品質モデルを Web API を利用した企業情報システムの品質モデルへ拡張する一つのアプローチを提示した点で意義がある．

本研究の成果はソフトウェア工学の発展に関して，次の３つの点で貢献する．

(1) サービス指向システム開発の生産性を阻害する課題を明らかにし，その解決を図る開発方法論を提案していること
(2) ソフトウェア工学で重要な，モデル駆動型開発，ソフトウェアパターン，品質モデルを技術の核として，ソフトウェア工学分野で実践的な新たな提案としていること
(3) 提案した開発方法論を実際の企業のシステム開発や Web API に適用し，多面的な評価尺度を用いてデータを収集して，その効果を定量的に評価していること

Abstract

Web services are the platform for enterprise information system development, a software engineering framework is required to solve new problems concerning the productivity improvement and quality assurance in enterprise information system development using Web services.

In this research, in order to improve the productivity of service-oriented system development to provide and utilize web services, the authors propose a development methodology to clarify issues that impede productivity and solve them. This research proposes a metamodel-driven business process modeling, a pattern system for Web service development, and a quality model from the viewpoint of users of Web APIs as the core technology of the proposed methodology.

To help developers reuse the business process models and best practices, the authors propose a methodology and an integrated environment for business process modeling driven by the metamodel. Furthermore, the authors propose a process-template design method to unify the granularity and separate the commonality and variability of business processes so that business process models can be reused across different enterprise software systems. The proposed methodology enables to create a set of reuse-oriented business process templates before the business process modeling. To support the proposed methodology, the authors developed an integrated environment for creating, reusing and verifying the business process models. As the key techniques, this research proposes the methodology and its integrated environment, including a meta-model and notations. The authors applied the methodology and integrated environment to an actual enterprise software development project, and evaluated that the productivity of business process modeling is improved by at least 46%. As the conclusion, this research contributes to prove the effectiveness of the meta-model driven business process modeling methodology for the reuse of business process models.

The pattern system of Web service development, as an attempt to disseminate new software development technology, provides development know-how of a system that uses XML, that is a specific technology for Web service development. The authors applied the proposed pattern system to 50 projects of actual enterprise system development and confirmed certain effect. The authors showed that a pattern system consisting of five entrances, including architecture, template model, detailed know-how, components, and development methodology, is effective in dealing with new technical elements that will emerge in the future. In addition, this research also presented the development process and development system of a pattern system that can be practiced in enterprises.

This research developed the concept of patterns into the reuse of Web services, and contributed to the spread of new technologies within the enterprise through it.

Web APIs differ from conventional APIs in that they execute remotely on different servers and may be changed independently of their users. These unique characteristics introduce new problems in the software engineering of Web APIs, and impose risks to the users, especially those using enterprise Web APIs. To solve these problems, in this paper, the authors propose a quality model for Web APIs that reflects their unique characteristics. As the main characteristics of this quality model, this research proposes the concept of Web API learnability to use and stability to change, from the perspective of Web API users. Based on this quality model, we also propose a set of measures and a quantitative evaluation method. In this study, the authors applied the proposed quality model and evaluation method to four types of actual Web APIs. To validate the proposed model, the authors also conducted an empirical study of the usability of the Web APIs. Our comparison of the proposed quality statistics with those from the empirical study validates the effectiveness of the proposed quality model and its associated measures of the learnability and stability of Web APIs.

The results of this research contribute to the development of software engineering in the following three

points.

(1) To clarify issues that impede productivity of service-oriented system development and to propose a development methodology to solve the problems

(2) To make it a reproducible proposal in the software engineering field with model driven development, software pattern, quality model important as the core technologies in the software engineering

(3) To apply the proposed development methodology to the actual system development of the enterprise and Web APIs, and to quantitatively evaluate the effectiveness with the collected data using multifaceted evaluation scale

目次

1 はじめに

1.1 研究の背景

Web サービスは企業情報システム開発で重要な要素技術である．2000 年代の XML や Web サービスの登場により，サービス指向システムが開発されるようになった．複数の企業が Web サービスとして提供する機能を，サービスインテグレータが統合して Web アプリケーションとしてエンドユーザに提供するシステム開発形態が普及している．Web サービスを提供する側と利用する側で，それぞれ新しい技術にいち早く取り組み，ビジネスの差別化を図っている．

近年は Web サービスを Web API として提供する形態が一般化し，Web API が企業情報システム開発のコア資産となっている[3] [38]．Web API はインターネットを介した Web サービスを提供する手段であり，REST (REpresentational State Transfer)アーキテクチャスタイルに準拠する[21]．Web API のグローバルなディレクトリ Programmable Web[69]には 2019 年 1 月現在で 2 万個を超える Web API が登録され，今後も急速に増加すると見込まれている．さらに，Web API を基礎とするソフトウェアとビジネスのエコシステムが国内外で成長している[3][94]．また，企業がエンタープライズ Web API として戦略的に Web API をパートナやコミュニティに公開し，新しいサービスやアプリケーションの開発を促し，Web API ビジネスを展開する動きがある[38]．しかし，Web API は従来の同一システムで利用されるシステム API と異なる特性を持つことから，多くの新たな問題を提起している[19]．

また，サービス指向システムのアーキテクチャ進化の観点では，細粒度で独立したサービスとして実装し提供されるマイクロサービス[24]がある．しかし，マイクロサービスもインタフェース定義は Web API となり，Web API が提起する問題点を共有する．

1.2 研究の目的

Web サービスは企業情報システム開発のプラットフォームであり，Web サービスを利用した企業情報システム開発における生産性向上と品質保証に関する新たな課題を解決するためにソフトウェア工学の枠組みが求められている．本研究では，Web サービスを利用する企業情報システム，及び，利用される Web サービスの両者を含むサービス指向システムの開発の生産性向上を目的に，生産性を阻害する課題を明らかにし，その解決を図る開発方法論を提案する．Web サービスの開発と利用を複数の企業アプリケーション開発で実施した経験から得たサービス指向システム開発方法論の技術を俯瞰し，提案する方法論の核技術として，メタモデル駆動の業務プロセスモデリング，Web サービス開発向けパターン体系，Web API の利用者観点での品質モデルを提案する．

本研究は，サービス指向要求分析における業務プロセスの再利用，Web サービス開発におけるパターンとソフトウェアコンポーネントの再利用を可能とすることで，生産性の向上と技術の企業内への浸透を期待する．また，企業情報システム開発に他者が Web API を利用する際の新たな研究課題を提起し，Web API が持つ従来のシステム API と異なる性質に着目した品質モデルを明らかにすることを目標に，他者が開発した Web API を実際に利用する前の判断に活用することを想定して提案する．

Web サービスに関するソフトウェア工学の研究が必要とされている中，その研究は少なく萌芽的な段階にある[88]．本研究はソフトウェア工学のアプローチにより，サービス指向システム開発の生産性向上，ならびに，その開発技術の発展への貢献が期待できるものである．あわせて，ソフトウェア工学の発展とその領域の拡大に寄与するものである．

1.3 本論文の構成

本論文の構成を以下に示す.

第２章では研究課題について説明する．第３章は関連研究について論じて，第４章でアプローチを示す．第５章はサービス指向の要求分析の生産性向上を目標とする業務プロセスモデリング技法を示す. 第６章は Web サービス開発の品質と生産性向上を目標にノウハウを体系化したパターン体系について説明する．第７章は Web サービスの提供形態である Web API の特徴を捉えた品質モデルを示す．第８章は提案する開発方法論についての考察を述べる．第９章は今後の課題，第 10 章は本研究のまとめを述べる.

2 研究課題

2.1 サービス指向システム開発の問題領域

企業情報システム開発の領域では，XML や Web サービスの提案以来，多くの企業がサービス指向システム開発に取り組んできている．これまで Web サービスの開発と利用を複数の企業アプリケーション開発で実施した経験から得たサービス指向システム開発方法論の技術を俯瞰して図 2-1 に示す．サービスのインテグレータが Web サービスを統合し新しいサービスとしてエンドユーザに提供する．企業では限られたパートナと企業情報システムを開発する組織形態が多い．また，ソフトウェア開発技術としてオブジェクト指向開発技術を基本とした．

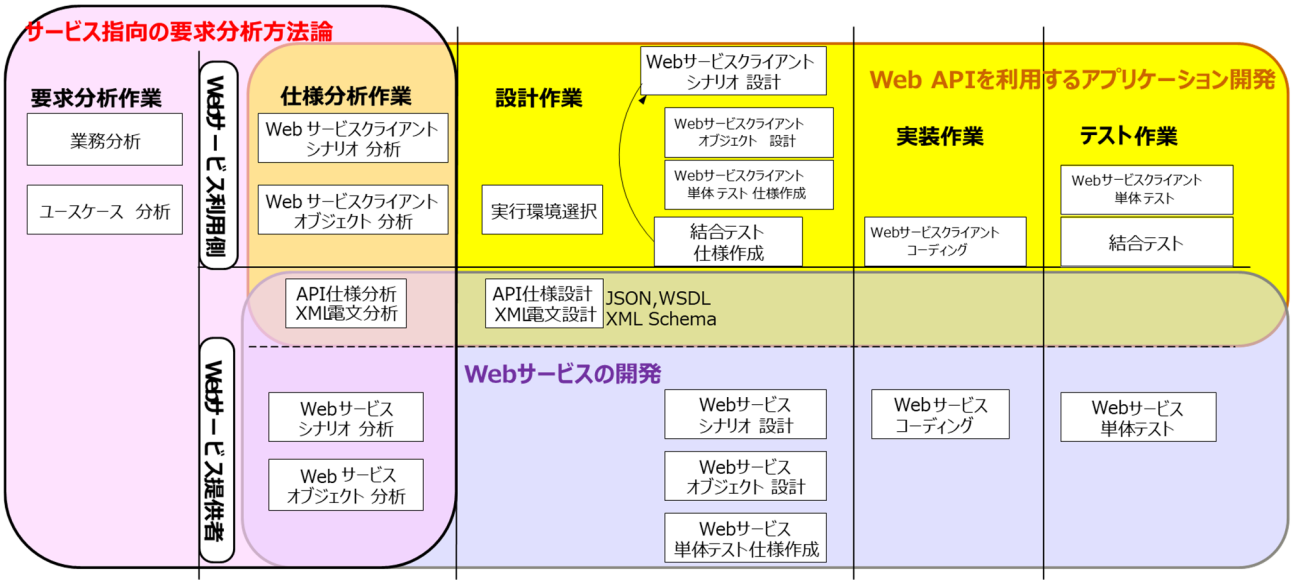

図 2-1　サービス指向システム開発方法論の全体概要

開発作業の概要とポイントを述べる．

要求分析プロセスでは，エンドユーザ側の業務分析と，エンドユーザをアクタに想定したユースケース分析を行なう．続く仕様分析プロセスで，各ユースケースを実装するためにインテグレータ側は「Web サービスクライアントシナリオ分析」を行い，利用する Web サービスの役割分担を明確にする．また，企業間のシステム連携の場合は，業務的な役割分担も合わせて合意する必要がある．この結果に基づき，各 Web サービスのインタフェースの仕様が決まる．企業間／システム間のインタフェースとデータの設計は，アプリケーション本体の開発より先行して行い，OpenAPI/Swagger, WSDL (Web Services Description Language)，XML Schema 等で定義されたそのインタフェース仕様を早期に合意し共有することが望ましい．一方，Web 上に公開されている Public な Web サービスを探索し，その仕様とサービス品質が確認できれば，それを利用するアプローチもある．

Web サービスをアクセスするためのインタフェース設計ではデータ構造の検討が特に重要である．例えば XML メッセージの設計で標準ボキャブラリの調査や既存システムのデータ項目との整合など既存データから，先行して分析モデルを開発し，共通に用いるべき型や要素をボキャブラリとして定義し，それらを部品としてプロジェクト内で共有する場合もある．XML メッセージはそれらの部品を組み合わせて設計する．XML Schema 設計に UML を適用する場合は表記上の工夫も必要で，クラスに XML 設計むけのステレオタイプ等を導入した設計モデルで表記する．そのような UML クラス図から一定の変換規則を適用して XML Schema

を作成することも可能である. REST における JSON を用いたデータの受け渡しに関してもデータの内容の理解が必須であり，データ設計は重要である．しかし，後述するようにデータ型の強い制約がない等の理由でインタフェース仕様と実装間の不整合が起こりやすい.

Web サービス利用側（インテグレータ側）は，GUI を含む Web サービスクライアントを設計，実装する．クライアントコードは，Web API の外部仕様を前提に開発する．WSDL が提供されていればそれをもとに開発ツールでプロキシクラスを生成し，開発効率化が図れる．REST では，API ドキュメントと実装が独立であることから，後述する課題が生じている.

Web サービス提供側は，規定されたインタフェースを実装する Web サービスの設計，実装を行なう．新規開発であれば，規定されたインタフェースを実装するサービスを開発する．既存アプリケーションを利用する場合は，規定されたインタフェースとの整合を図るため変換アダプタの開発が必要な場合がある．また，Web サービスの開発では，従来のプログラミング技術に加えて，例えば REST や XML 処理等の新しい技術やノウハウが求められる場合が多い.

上記で概説したサービス指向開発方法論の主要な技術領域を, 実践経験から図 2-1 に示す 3 領域に分割する．各領域での課題の概要は以下である．以降，各課題を背景も含めて説明する.

(1) サービス指向の要求分析方法論：企業情報システムのどの部分に Web サービスを用いるかを分析する要求分析作業の生産性と品質を確保する
(2) Web サービスの開発：Web サービスの開発に特徴的な新規技術への対応期間を短縮する
(3) Web API を利用するアプリケーション開発：Web サービスを Web API で提供する場合が増加しているが，企業情報システム開発に他者が開発した Web API を利用するには多くの課題があり[14][19]，アプリケーションの品質の確保にむけた取組みが必要である．アプリケーション開発の初期に他者が開発した Web API の利用可否を判断することが重要であるので，その判断に用いる品質モデルを想定する．具体的には図 2-1 における“API 仕様分析”に用いる.

2.2 サービス指向の要求分析方法論

企業情報システムの開発は，デジタルビジネスのためにその企業のビジネスを再設計する必要に直面している[46]. 業務プロセスは企業ソフトウェアシステムの核心であるため、次のような共通の状況下で業務プロセスを再設計するという課題がある.

(1) 企業は，業務プロセスおよびソフトウェアシステム全体を再検討し，業務モデル，業務プロセスおよびソフトウェアシステムを調整する必要がある．その根底にある動機は，企業が新しい業務モデルとその運用環境を，より短い期間に，より低いコストで創出する必要があることである.
(2) 企業は，今日の急速に変化するビジネス環境の中で生じる新たな問題に取り組み，企業全体に影響を及ぼす可能性のあるソリューションを開発する必要がある.

このような要求を満たすために，BPM (Business Process Management) の主要技術として業務プロセスのモデリング，ビジュアライゼーション，モニタリングのための多くの方法が提案されている[15][87]. さらに，企業ソフトウェア開発の要求分析における業務プロセスを明確にすることの重要性が認識されている．サービス指向システムの開発において，業務プロセスは企業情報システムが提供する機能群の仕様を与え，機能の一部を Web サービスで実装する際の適用判断の根拠ともなる.

これまで ARIS[13]などの業務プロセスモデリングのための技術とツールが提案されてきた[26]. さらに，多くの ERP (Enterprise Resource Planning) 製品は，テンプレートの形で様々な業務プロセスモデルを提供している．しかし，そのような業務プロセスモデルは実際の顧客ビジネスと乖離がある．さらに，業務分析者や開発者は，独自の業務プロセス分析と技術開発を採用する傾向があり，企業ソフトウェア開発における業務プロセスの再利用を妨げている[45]. 業務プロセスモデルを統一することが重要である.

本研究では，サービス指向の要求分析方法論として，業務プロセスモデルを統一して業務プロセスの再利用を可能とする方法の提供を課題とする.

2.3 Web サービスの開発

Web サービスの開発には，XML，RESTful，Web API など，新しいソフトウェア開発技術が登場している．新しい技術を活用したビジネスチャンスを獲得するために，各企業はこれらの技術を適用したアプリケーション開発を確立しなければならない．

しかし新しいソフトウェア開発技術を導入し，一定の期間が経過しても，システム開発に関わる部署の中に新技術に関する全般的な知識の獲得が進まないという課題がある．また新技術に関する知識を持つ技術者が各開発部署にいるとは限らない．そのため，人材不足のために新しいソフトウェア開発技術の利用を検討できない，または新技術を利用することの開発リスクが大きいため利用を見送る，といった事態も起こる可能性がある

新しいソフトウェア開発技術に関する知識やノウハウの集約と発信は新しいソフトウェア開発技術全般が必要としている課題である．この解決策として企業内で対象技術の相談室を設立しコンサルティング環境を構築する活動[30]などが行われている．

本研究ではこの課題に対する一解決アプローチとして，新しいソフトウェア開発技術を用いたシステム開発のノウハウをパターン化し体系的かつ相互に関連づけた形態，すなわち新技術を利用するシステム開発のパターン体系を提案する．

2.4　Web API を利用するアプリケーション開発

2.4.1　Web API の基本的な概念

Web API の定義は必ずしも統一されていないが，本稿では “HTTP プロトコルを利用してネットワーク越しに Web サーバをアクセスする API (Application Programming Interface)” とする[55]．Web API に関する基本的な概念は以下である．

リソース(Resource) : URI(Uniform Resource Identifier)でアクセス可能な Web 上の全てのプログラム，データ等をリソースと呼ぶ．URI を指定してアクセスした結果は表現(Representation)と呼ばれ，その形式には HTML，JSON，XML が用いられる[5]．

REST: REST は HTTP をベースとしたアーキテクチャスタイルであり，幾つかの設計原則が知られている[21]．そのなかでインタフェース定義に関わる主要な設計原則は，1)すべてのリソースは URI で表される識別子により参照できる，2)そのリソースにアクセスする「よく定義された操作のセット」，“GET”，“POST”，”PUT”，”DELETE”を提供する，等である．

2.4.2　Web API の課題を説明する事例

Web API のインタフェース定義の具体的な事例を図 2-2 に示す．この事例は，ある企業の顧客として登録されたユーザが，ボタンを押下するだけで商品を発注できるサービスを想定し，発注，発注取消，等の Web API を公開している，とする．図は，ボタンを押下して注文する際に用いる Web API のインタフェース定義である．このインタフェース定義には，HTTP メソッドが “POST” であること，リソースにアクセスするには “/v1/orders”という形式の URI をとること，リクエストメッセージには 1 個のボディパラメータが必要で，そのボディパラメータはユーザが押下するボタンの ID であること，レスポンスメッセージには”buttonID”に関連付けられたユーザの情報，注文された商品の情報，商品の配送に関わる情報が含まれること，が記述されている．また，このインタフェース定義は，リクエストとレスポンスのサンプルが含まれている．このサンプルには，リクエストやレスポンスの要素の値のサンプルとレスポンスが JSON 形式であることが記述されている．

POST /orders

Order Request

Order Requestエンドポイントでは，予めAromaボタンに登録されている商品の注文を行います．

Resource

POST /v1/orders

Authentification

OAuth 2.0 bearer token with the request scope.

Body Parameters

Name	type	optional	Description
buttonID	string		注文に使うAromaボタンのボタンID

Response

Name	type	Description
buttonID	string	注文に使われたAromaButtonのボタンID
userID	string	注文者のユーザID
userName	string	注文者の氏名
orderID	string	注文伝票ID
itemID	string	注文商品ID
itemName	string	注文商品名
amount	int	注文数量
orderDate	string	注文日
deliveryAddress	string	配送先の住所
deliveryDate	string	配送予定日
deliveryTime	string	配送予定時間

Error Response

Name	type	Description
errors	string	エラーのリスト
errors.statusCode	int	レスポンスのステータスコード
errors.code	string	レスポンスコード
errors.title	string	レスポンスのタイトル

Error Response一覧

Status	Code	Title
400	insufficient_inputs	The input is not sufficient.
400	invalid_value	The input's value is invalid.
503	service_unavailable	Aroma Develper Site is not availabled.

Example

Example1

Request

```
curl -X POST \
  -H 'Authorization: Bearer <token>' \
  -H 'Content-Type: application/json' \
  -d '{ \
    "buttonID": "b123" \
  }' \
  https://api.aroma.com/aromaAPI/v1/orders
```

Response

Status Code: 200 OK


```
{
  "buttonID": "b123",
  "userID": "ABCDE",
  "userName": "富士　通男",
  "orderID": "order98765",
  "itemID": "productA",
  "itemName": "紙おむつ(M) 50枚入り",
  "amout": "2",
  "orderDate": "2018/12/01",
  "deliveryAddress": "神奈川県川崎市中原区上小田中4-1-1",
  "deliveryDate": "2018/12/05",
  "deliveryTime": "時間指定無し"
}
```


図 2-2 Web API のインタフェース定義例

この Web API を利用する開発者は，インタフェース定義を読み取り，Web API の仕様に合致した HTTP リクエストメッセージを送信し，レスポンスを処理するためのプログラムを実装する．

```
public class DeliveryItemsApiServiceImpl extends DeliveryItemsApiService {
    @Override
    public Response deliveryItemsPost(CreateDeliveryItemParam createDeliveryItemParam, SecurityContext securityContext) {
        try {
            checkParams(createDeliveryItemParam);
            DeliveryItemCreatedResponse responseData = registerDeliveryItem(createDeliveryItemParam);
            return Response.ok().entity(new ApiResponseMessage(ApiResponseMessage.OK, responseData.toString())).build();

        } catch(Exception e) {
            return Response.notAcceptable().entity(new ErrorResponse(e)).build();
        }
    }
```

図 2-3: Web API の実装

図 2-2 の Web API のサーバ側の実装の一部を図 2-3 に示す. Java を実装言語としたプログラム例である．図 2-3 に実装されたサーバ側のプログラムにアクセスするため，クライアント側のプログラムでは図 2-2 から以下のような HTTP の POST メソッドを表現する文字列を作成し，サーバへリクエストとして送信する．

POST /v1/orders HTTP/1.1
…(略)…
{"buttonID": "b123"}

このリクエストからは, レスポンスコードと図 2-2 に示した JSON 形式の文字列がレスポンスとして返る．Java の Java Remote Method Invocation であれば，メソッドを呼び出すのも Java で記述できて，型定義によるチェックが可能である．しかし，REST ではこの様にリソースが文字列として表現される等，型定義も明示的でない．

2.4.3 Web API の品質モデルに関する研究課題

他者が開発した Web API を企業情報システム開発で利用するために，Web サービスの Web API の品質を定量的に測定し，保証する必要がある．しかし，Web API は従来のシステム API と異なる以下のような特徴を持つため，その違いを考慮した議論が重要である．

(1) 実装言語と独立したインタフェース定義：従来のシステム API は，その実装言語でインタフェースを定義することから，インタフェース定義を生成するツールも提供されている．しかし Web API のインタフェース定義は，実装言語とは独立に REST の原則[21]に従うリソースの表現として文字列で定義される．そのためツール化されておらず，インタフェース定義の作成は人手に依存している．さらに，そのリソース表現は，同一リソースであっても，JSON や XML など複数の形式を取り得て，自己記述メッセージ(Self-descriptive Message)として交換される．このため，型チェックなどが適用できない．このようなインタフェース定義の品質モデルは未確立である．

(2) ユーザと独立した進化：Web API はインターネットを介した Web サービスを提供する手段である．従来のシステム API がそのユーザの計算機資源の中に取り込んだライブラリ等をアクセスするのに対し，Web API はユーザの計算機資源とは異なる環境で開発，修正され，遠隔にサービスとしてアクセスされる．そのため，ユーザと独立して，機能の変更，追加が可能である．一方，利用できていた Web API が，ユーザが知らない間に変更され，その Web API を使ったアプリケーションが動作しなくなる，という問題にもつながる[85].

本研究では, システム API と異なる上記の特徴を持つ Web API を利用したアプリケーション開発において，アプリケーションの品質確保のために，アプリケーション開発の初期に上記の特徴を測定可能とすることを目的とする．開発の初期は，実際に Web API を使い始める前の作業が必要であり，Biehl[5]が"Test the API"と称したように，Web API に関する情報を収集してその適否を判断する．そして実際に使うまでの時間や仕様変更時の対応工数等を判断して，開発工数の見積りを可能とする．具体的には図 2-1 の "API 仕様分析" で実施

することを想定する．

そのため，以下の研究設問を設定する．

RQ1: システム API と異なる特徴を捉えるための Web API の品質特性と測定方法は何か?

Web API を利用したアプリケーションの品質を確保するために，従来のシステム API と異なる Web API の特徴を捉えた，Web API の品質モデルを定義する．本稿では，Web API を用いたアプリケーション開発の初期を想定して取り組む．Web API のインタフェース定義が実装言語と独立なリソースのテキスト表現であること，ユーザとは独立した開発サイクルが可能であること，等の品質に関わる新たな課題を解決するための品質モデル，品質特性，尺度とその測定方法を明らかにする．

RQ2: 提案する測定方法は実際の Web API に有効か?

実際に利用されている Web API を対象に，提案した品質モデルとその尺度，測定方法を適用し，妥当性，有用性を実証する．

3 関連研究

3.1 サービス指向の要求分析方法論

3.1.1 業務プロセスモデリング技法

業務プロセスモデリングと業務プロセスマネジメントに関する研究成果は数多く報告されている．

これらの中で，Eriksson と Penker は UML のアクティビティ図に“Process”のセマンティクスを導入した業務モデリング方法を提案した[16]. “Process”は”ActionState”のステレオタイプである．彼らはまた業務プロセスモデルのパターン群も記述している．RUP (Rational Unified Process)の中で，Eriksson-Penker のモデリング方法論は業務プロセスモデリングアプローチの代替として推奨されている[39]．しかし，これらのアプローチは業務プロセスモデリングにのみ焦点をあてており，業務プロセスモデリングの成果物の再利用は言及していない.

ISO/IEC OMG BPMN[64][32]は，標準化された一般的な業務プロセスモデリング言語である．もう一つ注目すべき表現には EPC（Event Driven Process Chain）[8]がある．どちらも業務プロセスのコンポーネントを表現するサブプロセスを持つが，再利用可能な業務プロセスモデルコンポーネントは含まない.

ISO / IEC OMG BPMN は業務プロセスモデリングの標準として広く受け入れられているが，実際にその表記を使うには様々な問題がある[47]. BPMN を使う業務プロセスモデリングにはまだ多くの問題が存在している[1][89].

3.1.2 業務プロセスモデルの可変性と再利用

業務プロセスモデルの再利用のためには，プロダクトラインソフトウェア工学のような業務プロセスモデルの共通性と可変性を管理するために，業務プロセスを構造化する必要がある[70]．Marshall はどのようにアクテビィティ図を分割し簡単化できるかを記述した[50]．しかし，彼はアクテビィティ図を再利用する方法については提案していない.

業務プロセスモデルの多くのバリアントで構成される業務プロセスファミリをモデル化する分解主導型の方法が提案されている[53]．その方法では，トップダウン分割で共通部分とバリアントをトレードオフすることを提案し，2 つのケーススタディから重複の最大 50%を削減できると主張している．この研究の概念は本研究と同じ方向であるが，本研究のテンプレートのような具体的なフレームワークは提供されていない.

3.1.3 業務プロセスモデリングツール

業務モデリングのためのツールは，学術/オープンソースと商用ツールの両方がある程度の数存在する．しかしそれらは, 機能, 開発のためのモデルや使用目的の点で異なる[76]．例えば, APIS Toolset は EPC と BPMN を使う業務モデリングツールを提供している[13]. Enterprise Architects[78] や Cacoo[59]等の他の作図ツールも UML と BPMN をサポートしている．これらのツールは，標準の再利用可能な業務プロセスコンポーネントに基づく再利用可能なプロセステンプレートはサポートしていない.

Rational Software Architect は RAS (Reusable Asset Specification) 形式のアセットのインポートとエクスポートができる[29][39]．しかし，プロセスモデリングの工数を削減するためのプロセステンプレートのサポートはできない.

3.2 Web サービス開発向けパターン体系

3.2.1 パターン体系

F. Buschmann らは「パターンは，それ単独に存在するものではない．パターン間には多くの依存関係が存在する．しかし，全てのパターンを平板に列挙したログのようなリストでは，これらの多種多様な関係は反映できない．このような方法ではなく，パターン体系の中に 1 つずつパターンが織り込まれるように記述されるべきである」と述べている[7]．しかし Buschmann らの文献を含め，実際に「パターンが織り込まれたパターン体系」の開発例を見つけることは難しい．例えば，R. Keller らはネットワーク管理インタフェースについてパターン体系を開発したと述べている[42]が，パターンのリストアップに留まり，パターンを織り込んでパターン体系にするところまでは至っていない．これは文献中で彼らも認めている．本研究では後述する（6.1.2）5 つの入口によってパターンを織り込むことを目指した．

R. Johnson らは「相当数のパターンが蓄積されてからでなければ，パターンコミュニティがパターンを体系化することは難しい」という意味の発言をしている[80]（3.3 章）．本研究ではパターン体系の対象範囲を絞ることで，最初にパターン化すべきノウハウを列挙することができ，初めからパターン体系化を念頭にパターンを構築できることを示す．

3.2.2 サービス指向システムのパターン

Fowler は企業がアプリケーションを開発する際の設計に役立つパターンを約 50 件のパターンカタログとして提供した[23]．自社内でソフトウェアを開発する想定で，OR (Object-Relational) マッピングなど具体的な実装をイメージできる．Erl は SOA の設計ノウハウを独自のパターンテンプレートを用いて整理し，SOA のためのパターンカタログとパターンランゲージを提唱した[17]．また REST による SOA に関して，原則，設計パターンをカタログにして，特にサービスの組み合わせに関して論じている[18]．Daigneau は，Web サービスの基本的な設計思想をパターン化し，実装コードも紹介している[12]．

Woolf らは，システムインテグレーションのためのパターンをカタログ化し，非同期メッセージングによる疎結合インテグレーションを対象ドメインに，疎結合に関する実際的な事例も含めて提示した[90]．

本研究では XML 処理に着目した Web サービスの実装を，実行可能なソフトウェア部品とともに提供するパターン体系を構築している．これらパターンカタログで論じられたノウハウを実際に動く形で提供している点が大きく異なり，より実践的な形態であるといえる．また，Erl や Daigneau の設計パターンとは補完するものとなる．

3.3 Web API の特徴と品質

3.3.1 Web API の急速な普及とその課題

Web API の急速な普及に伴い，その価値の認識[38][94]と共にその技術課題も明らかになってきた[19]．しかし，従来のシステム API と比較し Web API に関する研究は極めて少なく，萌芽的段階にあるといえる．さらに，従来主としてスマートフォンアプリケーションや Web アプリケーション向けであった Web API に対してエンタープライズ Web API と呼ばれる企業情報システム開発や B2B 向けの Web API が提供されるようになり，Web API の品質や Web API を用いた企業情報システム開発が重要な課題となっている[88]．

3.3.2 API の品質とアプリケーション開発への影響分析

Web API の品質はその仕様記述ドキュメントの品質として捉えられている．Web API の品質は，それに関

与するステークホルダのパースペクティブによる[52]．API 品質はそのユーザであるアプリケーション開発者に最も影響を及ぼすことから開発者のパースペクティブから評価すべきであると考えられている[14]．API 品質がそれを用いたアプリケーション開発の生産性に影響していることが実態調査と実証実験により明らかになっている．その中で API ユーザビリティの重要性が認識され，その研究が活発におこなわれてきた．ソフトウェア製品のユーザビリティの概念をシステム API へ拡張した API ユーザビリティの概念が提唱された[51]．さらに，この概念を Web API に適用した Web API ユーザビリティの概念が最近提唱されている[58]．Web API ユーザビリティがアプリケーション開発の生産性に及ぼす影響は開発者へのアンケート調査[84]や実証実験[77]によって明らかになっている．これらの結果から，Web API を用いた開発における開発者の開発経験として DX (Developer eXperience)の重要性が認識されるようになっている[14]．

しかし，これらの研究においては，ユーザビリティを含む幾つかの品質特性が示されているに留まり，Web API の品質特性に関する包括的な定義は未確立である．

3.3.3 API 仕様記述における例示の効果

システム API の利用において，その仕様記述に例示を含むことの有効性が示されている[79]．この成果を Web API に適用し，Web API 仕様記述に例示を含む効果を開発比較実験により実証している[77]．また，アンケート調査によって適切な例示の必要性も示されている[73]．これらの結果から Web API 仕様記述のスタイルガイドラインの検討も提案されている[57]．しかし，従来の研究では例示の有無に留まり，その構造や内容などに関する議論は見られない．

3.3.4 Web API の進化の問題

システムそのものの進化に関してはこれまで多くの研究がある[4]．この一領域としてインタフェースの進化，すなわちシステム API の進化に関する一連の研究がある[27]．これに対して Web API が異なる本質的特性の一つに“実行時の独立した進化”がある．すなわち Web API のコンシューマが Web API を利用中にその Web API のプロバイダが独立に変更，削除する可能性があることである．これは Web サービス共通の特性として知られているが，サービス中断などの深刻な問題を引き起こすリスクとなる[22]．さらに，Web API が変更されるとそれを利用するアプリケーションも変更せざるをえない状況がでてくる．例えば，Li らは 226 件の Web API の変更を 16 のパターンに分類し，その中に従来のシステム API にない新たな変更パターンを 6 つ指摘した[49]．この研究を発展させ，Wang らは StackOverflow 上での質疑から，21 の変更パターンを特定し，その中で，7 つが新たなパターンとしている[85]．また，約 82%の変更が改版の途中で行われていることを指摘し，変更が改版によらず常時行われていることも指摘している．近年のアジャイル開発や DevOps の普及により，この傾向は一層顕著になる可能性がある．

しかし， Web API 進化のこれまでの研究は変化のパターン化に留まり，Web API の進化がその品質特性として捉えられるまでには至っていない．

3.3.5 ソフトウェア品質モデル

ソフトウェア品質の体系である品質モデルはソフトウェア工学の基礎的研究課題として長年多くの成果がある[43][68][11]．これらの成果と測定に関する国際規格[31]を統合して，ISO/IEC 25000 シリーズが広く認知され，利用されている[34][35][36]．しかし，これらの成果は，単一システムの品質を対象としている．これに対して，Web サービス，Web API などのネットワークで連携するシステム，あるいは，それらが構成するエコシステムでは，前述した新たな問題が提起され，その品質保証にも新たな課題があることが指摘されている[41]．しかし，このような Web API システムに対する品質モデルに関する研究は少なく，その品質モデルの確立が求められている．

4 アプローチ

4.1 アプローチの全体像

本研究は，2 章で示したサービス指向システムの開発方法論における領域と各々課題に関して，図 4-1 に示すアプローチで取り組む．以降でアプローチの詳細を述べる．

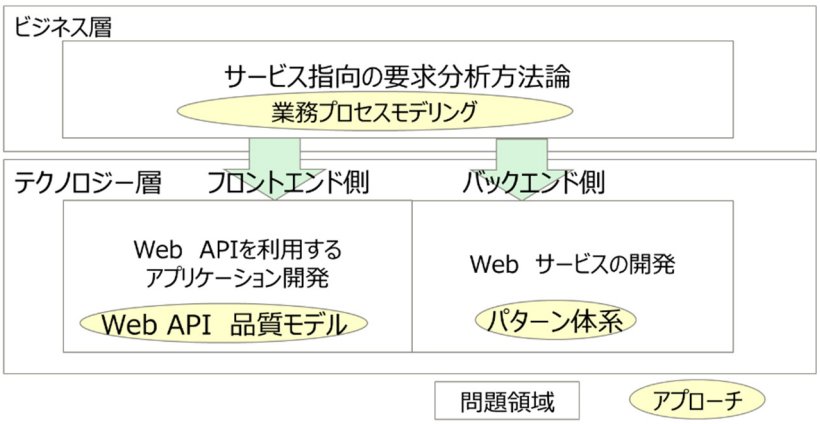

図 4-1　アプローチの全体像

4.2 業務プロセルモデリングの目標

業務プロセスモデルの再利用を可能にするために，標準表記法として広く使われている UML（Unified Modeling Language）[62]を拡張して，業務プロセスモデリングの方法を提案する．本研究のアプローチは，表現のための ISO / IEC OMG BPMN（業務プロセスモデルと記法）のアプローチに沿っている[64][32]．本研究では業務プロセスのモデリングをガイドする拡張メタモデル，及び，BPM テンプレートの集合を持つ再利用方法と支援する IE（統合環境）を提供する．

本研究での業務プロセスモデリングの目標は，顧客の業務プロセスを再設計して，その業務プロセスをサポートする企業ソフトウェアの開発期間を短縮することである．そのため，業務プロセスの可視化と再利用に焦点を当て，再設計をアジャイルに行うことを可能とする．業務プロセスの再利用可能性を目標に置いてモデル化されている場合，そのプロセスを再利用でき，かつ，プロセスを迅速に再設計し，企業ソフトウェアの開発時間を短縮することができる．

再利用を可能にするために，以下の再利用可能な業務プロセスモデルの性質を認識した．

(1) 客観性(Objectivity): 操作上の意味が明確に定義されていなければならない．
(2) 一貫性(Consistency): モデリング方法が統一されていなければならない．
(3) 保守性(Maintainability): 保守の継続的な制御が容易でなければならない．
(4) 再利用性(Reusability): 出力ドキュメントは業務プロセス分析のための入力として再利用できる．
(5) 詳細の記述性(Descriptiveness of details): 再利用可能なレベルに十分に詳細に記述する必要がある．

これらの要求に合致する業務プロセスモデリング方法論と統合環境を提案する．

本稿では 5 章で，提案する業務プロセスモデリング方法論とそれを支援する統合環境，業務プロセス再利用を可能とするプロセステンプレート開発方法，開発方法論，及び，統合環境を適用した際のソフトウェア成果物を説明する．

4.3 パターン体系の対象と前提

Web サービスの開発に必要なソフトウェア開発技術として本研究では XML を取り上げる．XML に関する基本仕様，API は標準化され, 様々な応用分野で利用されている. システムの新規開発やリプレースでは XML の利用を前提とすることが当たり前になった．また後述するように著者らは XML をアプリケーション開発の一つの革新であると認識しており，さらに XML を利用するシステム開発では XML のスキーマ設計が新たに中核的な作業であると考えている．このことから XML を利用するアプリケーション開発に関してパターン体系化すべきと考えた.

XML を利用するアプリケーション開発の特徴を以降で述べる．これらを前提に 6 章でパターン体系を提案する.

4.3.1 従来のデータ表現形式との処理方式の違い

XML は「一つの新たなデータ表現形式」である．XML データの処理方式が，これまでのデータ表現形式の処理方式と異なるのは，データ構造の変更（データの中の項目の増減）に伴うプログラム変更が少ない点である.

これまでのデータ表現形式，例えば固定長電文や CSV では，データの中の項目が増減してデータ構造が変更になるたびにデータを扱うプログラム（データを作成する部分やデータを解釈する部分）を変更する必要があった.

異なるデータ処理方式としてデータオブジェクトを用いる方法がある．Java などのオブジェクト指向言語を用いてデータとそのロジックをオブジェクトの中に隠蔽し，データ構造の各項目へのアクセスインタフェース介して外部からアクセス可能とする方法がある．この方法を用いることでデータ構造を変更しても，データを扱うプログラム（無変更の項目を扱う部分）は変更する必要がなくなった．しかしデータを表す Java クラス自体はデータ構造の変更にともなって変更し，再コンパイルする必要がある．この手間を省くために開発現場で行われたのは，データの中に用途の決まっていない予備の項目をいくつも用意しておいてデータ項目の増加に備えることだった.

XML はデータ自体を処理するプログラムが DOM[28], SAX[75], JDOM[40]などのライブラリとして用意されており，データオブジェクトのように再コンパイルする必要ない（もちろんデータ構造を変更してもデータを扱うプログラムを変更する必要がない）． データ構造の変更が XML 自体を処理するプログラムから分離できているのは，XML の仕様がデータの項目を表すタグを必ず付ける仕様であることと，データの項目を木構造で格納する仕様であることが理由である.

XML 処理が汎用化されており作り直す必要がないという XML の特徴は革新的である．本研究でパターン体系として提供する情報の一つである.

4.3.2 XML のスキーマ設計作業

XML のスキーマ（データ構造）の設計作業は早い段階で行われる場合が多い．疎結合システム間でのデータ交換に XML を用いる場合，特に異なる企業間でのデータ交換に XML を用いる場合は，開発の非常に早い段階で XML のボキャブラリ（XML のデータ構造と XML の各要素の意味）を決める．また，既存のシステムの置き換えのためデータ交換する内容がほぼ確定している場合や，既存の RDB テーブルの内容を XML データに変換する場合など，XML のボキャブラリを早くから決めることが可能な場合も多い.

一方 XML のスキーマ設計はデータの構造や型を意識する必要がある．CSV などと異なり，XML はデータを木構造で構造化して格納できる．また XML Schema[20]や RELAX N[72]など DTD より後に出て広まったスキーマ言語は，XML の中の各データに型を付けることができる.

構造や型を意識したスキーマ設計をどのように行うかも，本研究のパターン体系で提供すべき重要な情報である.

4.4 Web APIの特徴を捉える品質モデル

本研究では，ソフトウェア製品の品質モデルを基礎として，Web API の特性とそれによって提起されている品質に関する新たな課題を解決するように品質モデルを拡張するアプローチをとる．そのための着眼点とそれに基づく具体的なアプローチを以下に示す．これらを前提に 7 章で Web API の特徴を捉える品質モデルを提案する．

4.4.1 Web API 品質に関与するステークホルダとそのパースペクティブの設定

一般に，品質要求は ISO/IEC/IEEE 29148[37]で規定するようにステークホルダ毎に異なる．また，品質モデル国際規格 ISO/IEC 25010 (以下，ISO 25010 と略記)では，ステークホルダとして 1 次，2 次，間接の 3 つのユーザを設定し，各ユーザのパースペクティブから品質特性を議論している．しかし，このユーザモデルは単一システムが前提となっており，Web API では適切とはいえない．そのため，Web API において一般に知られている次の 3 つのステークホルダ[5][58]を定義し，そのパースペクティブを起点として品質を捉える．

(1) API プロバイダ: Web API を実装し提供するバックエンドサービスの提供者
(2) API コンシューマ: Web API を用いたアプリケーション／プロダクトの開発者
(3) アプリケーションユーザ: Web API を利用したアプリケーションのユーザ

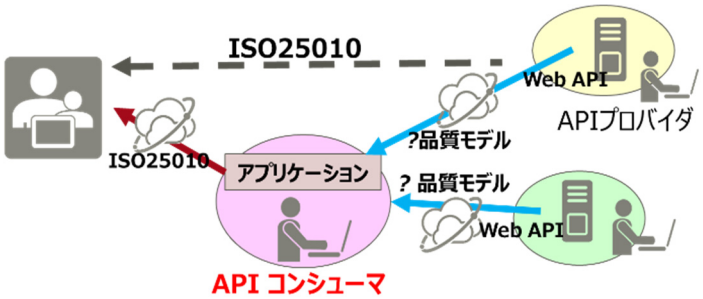

図 4-2 Web API 品質に関与するステークホルダ

本節では，Web API を使用するアプリケーション開発における API コンシューマの視点からの品質に焦点を当てる．図 4-2 に，3 つのステークホルダの関係を示す．

ISO25010 はソフトウェア製品の提供者とそのユーザ間での品質を規定する．しかし，API プロバイダが提供する Web API はアプリケーションユーザが直接利用するわけではなく，一般にはそれ単独のユーザインタフェースも持たない．API コンシューマが開発するアプリケーションが Web API を利用し，そのアプリケーションがアプリケーションユーザに提供される．API コンシューマは開発者であり，開発者の体験(Developer eXperience)の観点からも，ISO25010 を適用するには議論が必要である．

4.4.2 ソフトウェア製品の品質モデルの拡張

ソフトウェア製品の品質モデルの国際規格 ISO 25000 シリーズは広く受け入れられ，実践においても普及している．しかし，上述のように Web API では開発，利用がシステム API と異なることから，品質モデルの見直しが必要となっている．本研究は，この品質モデルを基礎とし，Web API の提起する課題に対する品質特性を特定し拡張するアプローチをとる．これによって，従来の品質モデルとの整合性をとり，実践における導入や活用を容易にする．

5 メタモデル駆動の業務プロセスモデリング

本章は，業務プロセスモデリング方法論とそれを支援する統合環境，業務プロセス再利用を可能とするプロセステンプレート開発技法，開発方法論，及び，統合環境を適用した際のソフトウェア成果物を提案する．

5.1 業務プロセスのメタモデル概要

5.1.1 企業の業務プロセス記述内容の反映

提供する業務プロセスモデリング方法論は企業の実際の顧客システム開発に適用可能でなければならない．そのために本研究では，現実の課題を把握しそれを解決する方法をとることとし，複数の顧客システム開発プロジェクトの業務プロセスフローの実態をまず調査した．

開発プロジェクトの多くは，顧客業務を UML のアクティビティ図に似たフロー形式で記述しており，顧客が理解可能なアイコン（絵）も使う．業務プロセスフローを用いて顧客業務の内容やシステム化範囲を確認・決定するために，開発者と顧客で共有するためである．それらの表記はプロジェクト内では統一しているが，プロジェクト間では統一されていなかった．また，納品物としての可読性を向上させるため，表計算ソフトウェアを用いて方眼紙の様に升目を作成し，記述要素の上下左右を整頓するような工夫もしていた．

現場の必要性の高い，UML のアクティビティ図に含まれていない記述として以下を抽出した．

(1) 業務プロセスに関連する時間
(2) データの処理に関わる“手段”．例えば“書類をバイク便で送る”を表現する際の“バイク便”に相当する．なお，“送る”はプロセスに，“書類”はオブジェクトとして従来のアクティビティ図のメタモデルで対応できる．
(3) 業務プロセスの実施者と組織構造の関係

これらをメタモデルに反映して，UML アクティビティ図を拡張した記法を開発した．開発した記法は，調査対象プロジェクトが記述した業務プロセスを本記法で記述することで適用可能であることを確認した上で，開発プロジェクトに記述内容を確認してもらうことで，採用可能との合意を得た．

5.1.2 メタモデルと BPME(業務プロセスモデリング言語)

ISO / IEC OMG BPMN[64][32]や個別に開発された言語[44][53]を含む多くの BPML（Business Process Modeling Language，業務プロセスモデリング言語）が提案されている．しかし，それらの多くが，抽象的過ぎる，および/またはワークフローに制限されており，共通理解が欠如しているため組織全体で再利用することは困難である．本研究では，業務プロセスモデリング言語とそのセマンティクスの共通の基礎を提供するメタモデルである BPML メタモデルを提案する．図 5-1 に，提案する BPML メタモデルの一部を示す．この BPML メタモデルは UML 2.4.1 superstructure[63][33]のメタクラスをベースとして，プロセスプロファイルを拡張した．表 5-1 に BPML メタモデルのクラスと UML 2.4.1 superstructure のクラスとの関係をまとめて示す．UML 2.4.1 superstructure は，specification の中に superstructure を統合した最新の UML 2.5.1 specification[62] をベースとすることに留意されたい．従って，提案した BPML メタモデルは UML 2.5.1 と一貫している．

メタモデルは業務プロセスモデリングの評価用のインデックスを含む．本研究では，前述した顧客システム開発の現場の必要性から，メタクラス名“Process”を持つ業務プロセスの一般的な評価基準として，以下の属性を定義した．

(1) Duration1 and duration2: 業務プロセスに必要な時間（例：平均時間，最大時間）
(2) Cost: 実行された時のコスト
(3) Condition: 起動条件

また，業務プロセスフロー中のデータに対して手段(Mean)を導入した．業務プロセスモデリング言語の表記法では，UML との一貫性を高いレベルで維持する必要である．後述する業務プロセスフロー図は，UML アクティビティ図を追加のアイコンで拡張したものである．業務構造図，業務データ図，及び，組織図は，UML クラス図にアイコンを追加して拡張している．顧客システム開発の現場の必要性から，業務プロセスの実施者と組織構造の関係を“組織図”で記述可能としたものである．

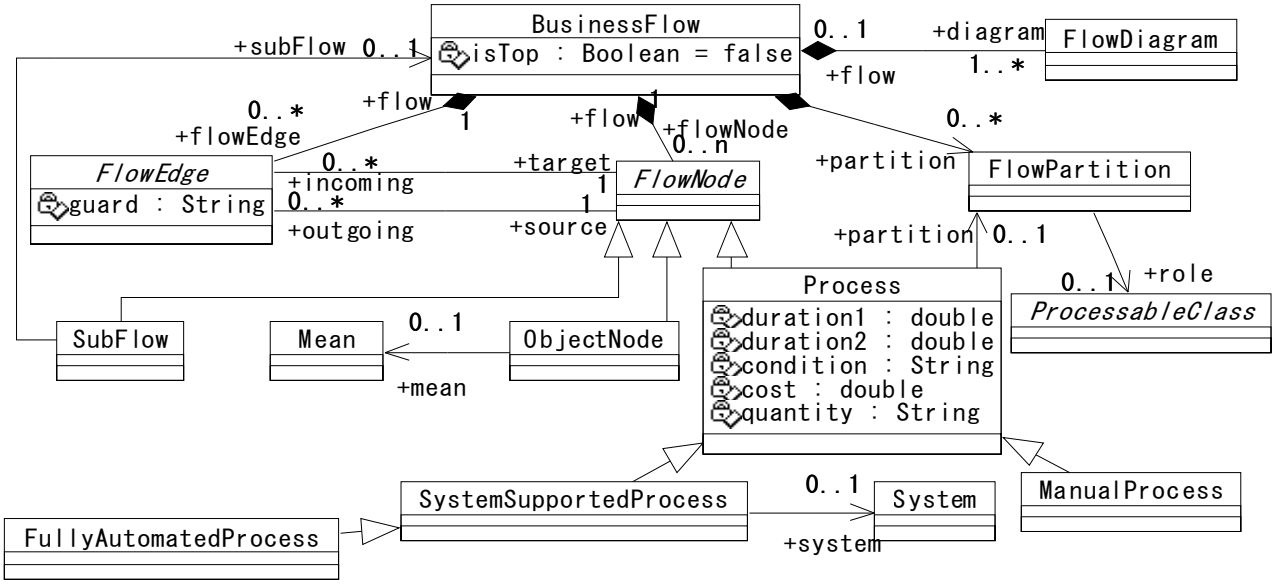

図 5-1　BPML Metamodel　(一部)

表 5-1　提案する BPML メタモデルと UML 2.5.1 superstructure の対応

業務プロセスモデリングのメタクラス	対応する UML 2.5.1 のクラス
BusinessFlow	Activity
FlowNode	ActivityNode
FlowEdge	ActivityEdge
Process	Action (stereotype)
ManualProcess	Action (stereotype)
SystemSupportedProcess	Action (stereotype)
FullyAutomatedProcess	Action (stereotype
SubFlow (SubActivity)	CallBehaviorAction
ObjectNode	ObjectNode
Mean	Class (stereotype)
FlowPartition (Swimlane)	ActivityPartition
FlowDiagram	(no correspondence)

5.1.3　業務プロセスモデルの要素

BPML メタモデルに基づいて，再利用を容易にするための業務プロセスモデルの主要要素を定義する．以下で説明するように，業務プロセスモデルは(1)業務全体構造と，(2) 業務プロセスフロー (3) 業務データ (4) ロール (5) BPM プロパティで構成する業務プロセスの詳細構造，の 2 階層で構成される．

(1)　業務構造

業務構造は，階層内の業務プロセス間の階層化された構造と関係でモデル化される．「業務構造図」は UML クラス図とパッケージ図に基づく．対象とする業務構造を記述することを目的とする．各業務プロセスモジュールは，図中の UML パッケージとして表される．業務構造図の詳細は，以下の業務プロセスフローおよび業務データによって記述される．

(2)　業務モジュール内の業務プロセスと業務プロセスフロー

作業の単位は一般に“業務プロセス(Business Process)と呼ばれる[15]．業務プロセスは，作業単位の順序付けられたフロー，すなわち業務プロセスのフローに構造化される．これが“業務プロセスフロー図”で記述される．業務プロセスフロー図は UML アクティビティ図に基づく．図 5-2 に示すように，業務プロセスモデルの一単位を，業務プロセスのワークフローとして詳細化する．

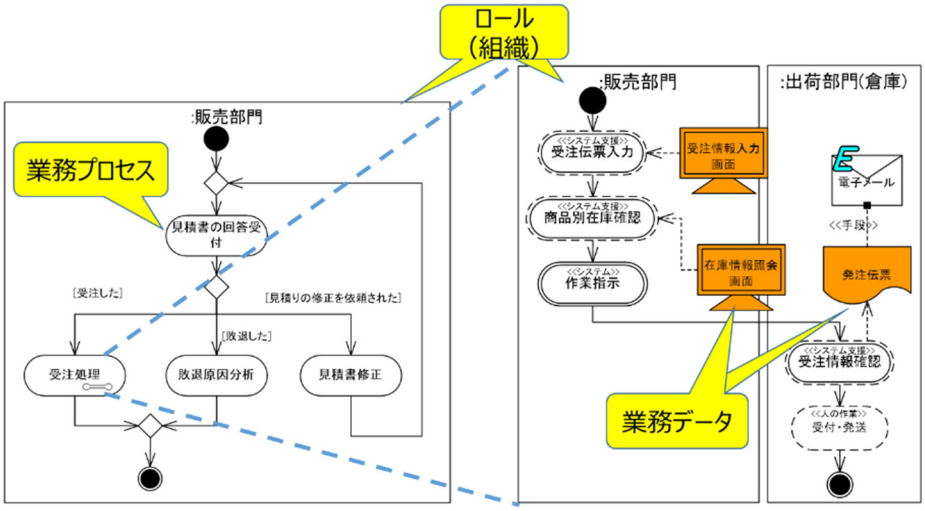

図 5-2　業務プロセスフローの例

条件は，あるフローを複数の並列フローとなるように制御するよう，フローの分岐に関連付けられる．業務プロセスフローをさらに詳細化する必要がある場合は，1 つの業務プロセスの詳細を示すプロセスフローを用いることができる．

図 5-3 は，図 5-2 に示す「受注処理」業務プロセスを詳細化した業務プロセスフローを示す．業務プロセスは，コンピュータのシステムだけでなく，人によっても実行される作業単位である．これを，各業務プロセスに，<<人の作業>>，<<システム支援>>，<<システム>>の 3 つのステレオタイプに型付けして表現する．

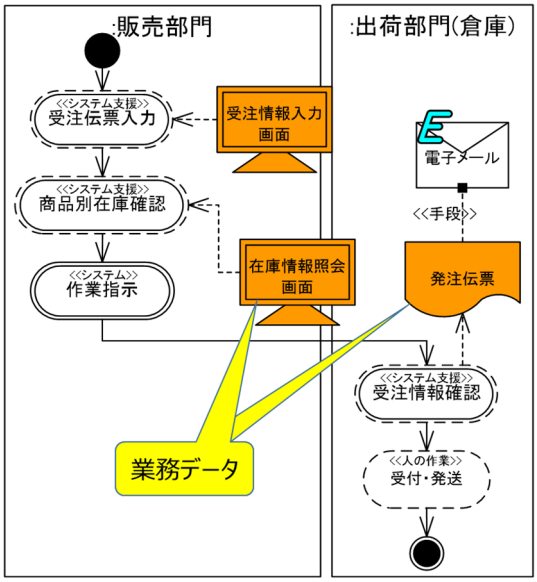

図 5-3　図 5-2 の“受注処理”を詳細化した業務プロセスフロー

(3)　業務データ

業務データは，業務プロセスの入力および/または出力であり，プロセスによって作成および/または変更される．「業務データ図」は，業務プロセスに関連するデータの全体像を表す．これは“業務”を意味的に拡張した UML クラス図に基づく．業務データは，業務データに固有のタイプの集合を含む「クラス」として表される．図 5-3 に例示したように，データの種類を示すある特定のアイコンを用いる．必要であれば，データ送信手段が記述される．図 5-3 では「発注伝票」を送信するために電子メールが使用されている．

(4)　組織内のロール(役割)

ロールは，業務プロセスを実行する組織要素である．UML クラス図に基づく「組織図」は，組織内のロール（役割）を記述するために用いる．この図で役割は「クラス」で表現される．

(5)　BPM プロパティ

BPM を通じて業務プロセスを改善するために，評価基準を提供する必要がある．評価基準と関連する関連する統計は，業務プロセス，業務データ，及び，組織に定義することができる．　業務プロセスの場合，完了に必要な時間とコスト，起動条件（例えば毎月の初日），が定義される．

5.2 メタモデル駆動の業務プロセスモデリング方法論とサポート統合環境

5.2.1 BPM 方法論の目的

4.2 節で述べた目標に基づいて、メタモデル駆動型 BPM 方法論と BPM-IE（Business Process Modeling Integrated Environment，業務プロセスモデリング統合環境）の具体的な目的を設定した．

(1) BPML メタモデル

BPML メタモデルを使用して，5.1 節で説明した各エンティティの意味を明確にする．その結果，メタモデルが，異なる業務プロセスモデル中のエンティティ間の一貫性を保証し，業務プロセスモデルを BPM-IE を介して共有されるデータとして再利用することを可能とする．メタモデルは，モデルの変更を管理することによって業務プロセスモデルの保守性を向上させる．特に，メタモデルは顧客からの要求を満たすために，UML superstructure のように業務プロセスモデルの拡張に役立つ．メタモデルに対するモデルの拡張を検証することにより，拡張されたモデルの一貫性，完全性，及び，非あいまい性を検証することが可能となる．

(2) メタモデル駆動型業務プロセスモデリング方法論

メタモデル駆動の業務プロセスモデリングとは，業務プロセスモデリングのプロセス全体が以下に説明する BPM-IE のメタモデルによって管理されることを意味する．提案した方法論は業務プロセスの再利用に重点を置いており，メタモデルは上記のように一貫性を保証できる．

(3) BPM-IE (Business Process Modeling Integrated Environment)

BPM-IE は，メタモデルのガバナンス下で業務プロセスのモデリングを支援する．メタモデルにより，BPM-IE は作成された業務プロセスの一貫性，完全性，非あいまい性を保証する．従って，業務プロセスモデリング方法論に基づいて業務プロセスを作成し再利用するためには，BPM-IE が不可欠である．

5.2.2 業務プロセスモデリング方法論

図 5-4 に業務プロセスモデリング方法論の 3 ステップのプロセスの概要を示す．

(1) 準備作業

“準備作業”では，業務プロセスモデリングプロジェクトを開始し，対象とする組織，業務プロセス，及び，明確化するべき BPM プロパティを含む業務プロセスのコンテキストを決定する．プロジェクトを開始後，その組織で使用されている語彙を収集して定義し，評価基準を設定し，業務プロセスモデリングに関わるステークホルダ間で合意を形成する．

(2) 業務プロセスモデリング作業

“業務プロセスモデリング作業” はこの方法論の中心的なプロセスである．“業務プロセスフローの分析”と“業務データの分析”の 2 つのアクティビティで構成される．2 つのアクティビティは，同じ業務プロセスモデルの 2 つの側面，フローとデータを記述し，反復的に実行する．

業務プロセスフローの分析”アクティビティでは，モデル要素を抽出し，ワークフローとともに業務プロセスフローを分析する．業務プロセスは 3 つの実行ステレオタイプに分類される．次に，業務・データを，各業

務プロセスの入力および/または出力として抽出する．また，各業務プロセスに対してプロパティを識別する．

業務データを抽出した後，“業務データの分析”アクティビティで，データを分析してその属性を定義する．最後に，業務プロセスフローの入力および出力の形式（画面や帳票，等）を指定する．

(3)　結果のまとめと承認作業

“結果のまとめと承認作業”は，業務プロセスモデルを検証して，業務プロセスモデリングに関わるステークホルダとモデルに関するコンセンサスを作成するプロセスである．

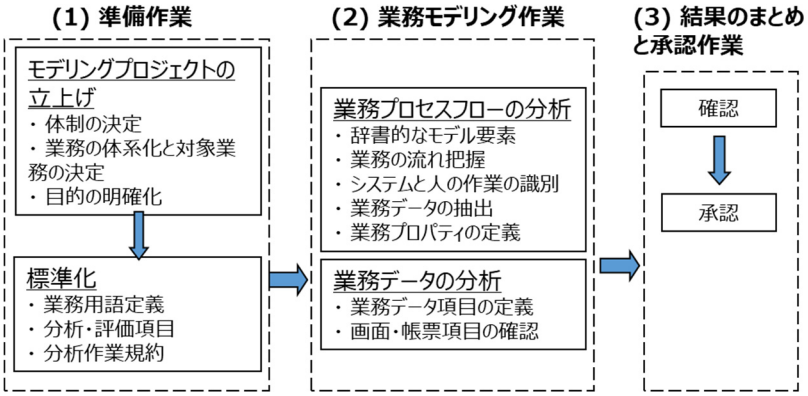

図 5-4　業務プロセスモデリング方法論の 3 つのステップ（概要）

提案した方法論は，以下の特徴を備えている．

(1)　“準備作業”は，方法論の目的と範囲を設定する上で重要である．顧客を含む主要なステークホルダ，すなわち業務プロセスを担当する人々，業務分析者，及び，開発者は，業務プロセスモデリングの目標を事前に設定している．目標に基づいて，分析及び評価基準をカスタマイズできる．

(2)　“準備作業”の終わりに，業務プロセスモデリングの目標と範囲について合意することが期待される．スコープクリークを防止し，業務プロセスモデルの共有と再利用を促進するには，合意が重要である．例えば，業務プロセスフロー図には一連の制約が含まれているため，スコープと実行セマンティクスを UML アクティビティ図よりも明確に定義できる．

(3)　BPM-IE は、“結果のまとめと承認作業”において，エラーが発生しやすい作業である業務プロセスモデルの一貫性，完全性，非あいまい性の検証を可能にする．

5.2.3　BPM-IE (Business Process Modeling Integrated Environment)

本研究では，図 5-5 に示す，BPM-IE (Business Process Modeling Integrated Environment，業務プロセスモデリング統合環境)を開発した．

図 5-5 の中央のウィンドウには、CRM（Customer Relationship Management）の業務プロセスフロー図が表示されている．左上のウィンドウには業務プロセスモデルのリポジトリのディレクトリが表示されている．このリポジトリのルートディレクトリが業務全体を示す．ディレクトリの下は，リポジトリに格納された業務プロセスモデルの階層である．左下のウィンドウには，CRM の業務データの定義が表示される．別の左下のウィンドウには，業務プロセスのパフォーマンスのメトリックと目標値を含む BPM プロパティが表示されている．

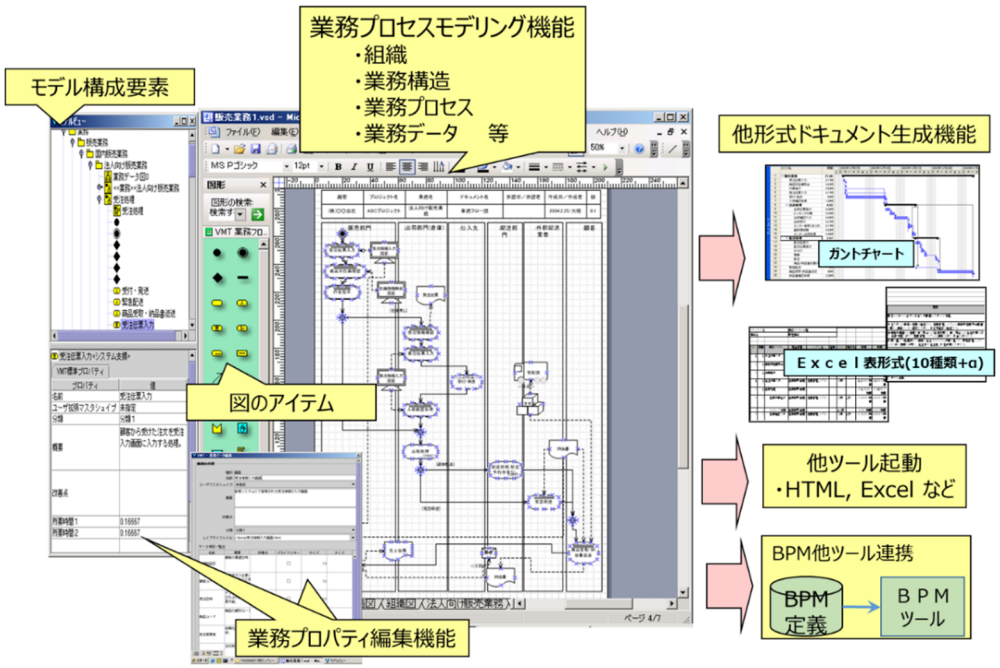

図 5-5　BPM-IE (Business Process Modeling Integrated Environment)の画面ビュー

この BPM-IE は以下の機能を持つ.

(1)　業務プロセスモデリング

BPM-IE は，業務構造図，業務プロセスフロー図，業務データ図，組織図の生成と編集を統合して支援する.

(2)　BPM プロパティ設計

BPM-IE は，業務プロセスと業務データの詳細な情報の生成と編集を支援する.

(3)　業務プロセスモデルの検証

BPM-IE は，業務プロセスモデルが非あいまい，かつ，一貫していることを保証することを支援する．つまり，業務プロセスモデルの図には不一致がない．検証は，メタモデル及び表記法によって定義された制約に基づいて実施される．例えば，業務プロセスフロー図の条件分岐で条件が指定されていない場合，BPM-IE はその問題と考えられる解決方法を提示する.

(4)　複数視点で文書を生成

業務プロセスモデルのレビューと承認をサポートするために，複数の視点及び表記の文書を生成する．例えば，業務プロセスのプロパティをリストした表形式文書を生成することができる．さらに，ユーザがドキュメントの書式や表示される BPM プロパティの選択をカスタマイズすることができる.

(5)　モデルの拡張

モデル拡張機能は，メタモデルのメタクラス単位に，ユーザ固有の BPM プロパティを追加し，デフォルトの BPM プロパティを変更することによって，“準備作業”で業務プロセスモデリングの機能を拡張するために使用する．

(6) BPM-IE API によるツール統合

この BPM-IE は API を通じて機能を提供する．他のツールや IE は，API を通じてモデリング成果物にアクセスできる. 例えば, 業務プロセスフローのデータは, 業務プロセスの実行を監視するための BPM 監視ツールに提供される[91].

5.3 BPM テンプレートを用いた業務プロセス再利用方法

5.3.1 業務プロセス再利用方法の原則

前節で述べた業務プロセスモデリング方法論に基づいて，BPM テンプレートを用いた業務プロセス再利用方法を提案する．検証された業務プロセスモデルを再利用することで，業務プロセスモデリングの機敏性，生産性，品質を向上させることが期待できる．

5.2 節で説明した方法論と BPM-IE を使用すると，モデリング成果物を統合された形式でリポジトリに格納することができる．BPM-IE では，業務プロセスモデルが完全で一貫性があり，かつ，明確であることが保証されている．

しかし，このような成果物が，顧客固有の変更や拡張および情報を単純に排除することによって再利用することができないことに気付いた．再利用を容易にするために，関心事の分離（separation of concerns）の原則に基づいて以下の 2 つの原則を導入した[6].

(1) 業務プロセスとデータコンポーネントの統一した粒度を定義する必要がある．
(2) 共通性と変動性の分離：再利用可能な共通情報と顧客固有の情報を分離する必要がある[70]．変化点はホットスポットとして仕様化される．
この原則が OMG RAS（Reusable Asset Specification）[61]の根拠と一致することに注意されたい．

これらの原則に基づいて業務プロセスの再利用を実現するために，モジュール性基準と業務プロセスフローの階層の抽象度とレベルを定義した．さらに，業務プロセスモデルを再利用することによる生産性向上を評価した．

5.3.2 BPM テンプレートを用いた業務プロセス再利用方法

業務プロセスの再利用の原則を具体化するために，3 階層の BPM(Business Process Model，業務プロセスモデル)テンプレートを定義した．そのテンプレートを用いて，BPM テンプレートの開発と適用の方法を提案する．最初に，この方法は再利用可能な業務プロセスフローを記述することを可能とする．

また，実際のプロジェクトのために再利用可能な業務プロセスモデルを開発するために，業務プロセスモデリングチームのメンバの役割も定義した．

表 5-2 に，再利用可能な業務プロセスモデルの階層 3 の構造の概要を示す．

階層 1 と階層 2 の業務プロセスフローは，業務プロセスフローの抽象化であり，複数の業務プロセスにおける共通性を表す．つまり，それらが業務プロセステンプレートとして再利用される．階層 3 の業務フローは，階層 2 で技術された業務プロセスの詳細を定義する．従って，階層 3 の業務プロセスフローが変動性を表す．

表 5-2　再利用指向の業務プロセスモデルの階層

業務フローの階層	目的	共通性の基準	作成者	記述内容の指針
階層 1/ Level 1 (概観業務フロー)	主要業務の構成	業種・業態で共通	リーダ	- 業種で共通な主要業務とその順序を記述する - "部署（ロール）"と "データ"は書かない
階層 2/ Level 2 (典型的なフロー)	標準的な業務の流れ	業種・業態で共通	リーダ	- 階層 1 の業務プロセスの入出力となる業務データ - 上記の入力から出力までの手順となる業務プロセス - 業務に関わる主要部署
階層 3/ Level 3 (システム化業務フロー)	システム化範囲の定義を含む，業務の流れ	顧客別	担当者	- 階層 2 の業務プロセスを実現するシステム機能．“業務プロセス”がシステムの“ユースケース”となる - 機能の入力データ，出力データ - 機能の業務上の使い方

5.3.3　BPM-IE の再利用のための機能

業務共通部分と顧客個別部分を分離することで業務プロセス再利用方法を支援するため，以下のツール機能を BPM-IE で提供する．

(1)下位レベルの業務プロセスの付け替えによる再利用支援

下位レベル業務プロセスは，より上位レベルの業務プロセスとして定義された業務プロセステンプレートを詳細化する．上位レベルの業務プロセスフローを再利用して，下位レベルの業務フローを顧客の業務に合わせるために，適切なサブアクティビティグラフをテンプレートの一部に置き換える．

(2) RAS によるモデルコンポーネントの管理

本研究では，業務プロセスモデルが業務プロセスのサブフローの候補となる分割された業務プロセスフローの集合である業務プロセスモデルのコンポーネント群を，RAS[61]の「デフォルトプロファイル」を使用して管理する．RAS は，資産のメタ情報を提供するための，再利用可能な資産の構造と XML の記述形式を規定する．

本研究の業務プロセスモデルでは，ある SubFlow オブジェクトは，図 5-6 に例示する詳細化された業務プロセスフローを持つような業務プロセスインスタンスを記述する．

詳細化されたフローは，SubFlow オブジェクトにリンクされている BusinessFlow オブジェクトで表現される．本研究の BPM-IE には，RAS に準拠した XML で書かれた以下の管理情報が格納されている:

a) SubFlow インスタンスにリンクするための詳細化された BusinessFlow オブジェクトは複数の候補を持つ，または，今後持つことになるような，SubFlow インスタンス群.
b) SubFlow インスタンスにリンクできるような業務フローインスタンス

このかなり情報は RAS のデフォルトプロファイルの XML “artifact” 要素と“artifact-dependency”要素で表現することが可能である．

例えば，図 5-7 に示す XML 記述は，図 5-6 の SubFlow オブジェクトと BusinessFlow オブジェクトの関係を表している．図 5-6 では， Id＝"287"，Id＝"050"，及び Id＝"E90"の BusinessFlow オブジェクトの候補の集合から，Id = "287"の BusinessFlow オブジェクトが選択されて，Id = "F67"の SubFlow オブジェクトとリンクしている．

(3) RAS を用いたモデルコンポーネント選択

業務プロセスモデルテンプレートは，業務プロセスモデルコンポーネント間の関係に関する情報が含まれる．本研究の BPM-IE はモデルテンプレートをインポートした後に以下を実施する．

a) ホットスポット，詳細化された業務プロセスフローを顧客個別の情報で置き換えることができるポイ

ントを強調表示して，

b) ツールユーザがホットスポットを選択したときに代替候補を示し，

c) ユーザがある使用固有の業務プロセスでの代替を要求した場合，その詳細化された業務プロセスフローで置き換える

業務プロセスフローはオブジェクトノード，業務アクション，業務データオブジェクトへの参照を含む．提案する BPM-IE はさらにその含まれるオブジェクト群や参照するフローを詳細化することを支援する．

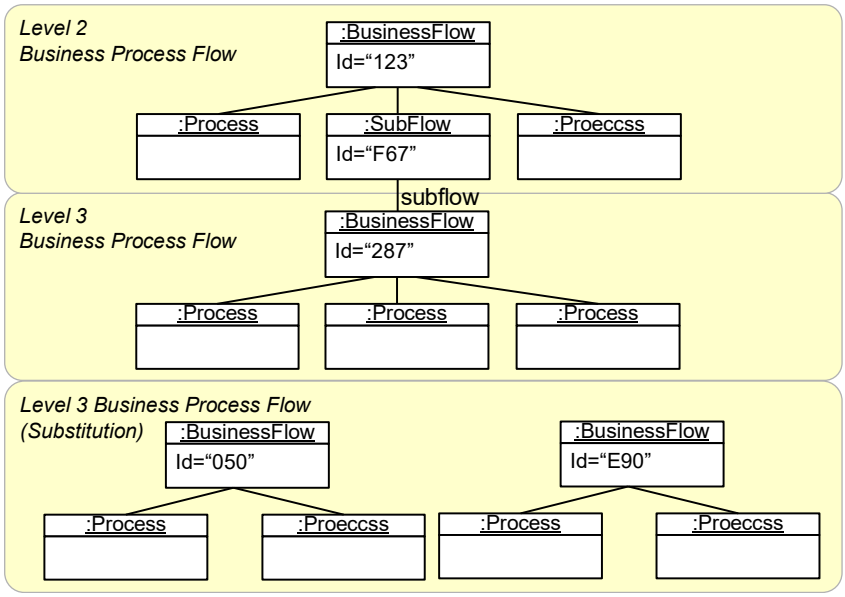

図 5-6　業務プロセスモデルのオブジェクト図

```
1: <asset name="Sales-1" date="2005-01-11Z" id="124">
     .. .. ..
2:   <solution>
3:     <artifact id="840" type="vsd" name=".¥SalesTemplate.vsd">
4:       <artifact-type type="model-file"/>
5:       <artifact id="120" type="Package">
6:         <artifact-type type="template-base"/>
7:         <artifact id="F67" type="SubFlow">                    (Substitution Point)
8:           <variability-point id="A88"/>
9:           <artifact-type type="template-plug"/>
10:          <artifact id="287" type="BusinessFlow">             (Candidate)
11:            <artifact-dependency artifact-id="A88"/>
12:            <artifact-type type="template-part"/>
13:          </artifact>
14:        </artifact>
15:      </artifact>
16:    </artifact>
17:    <artifact id="550" type="vsd" name=".¥partsForSalesTemplate.vsd">
18:      <artifact-type type="model-file"/>
19:      <artifact id="050" type="BusinessFlow">                 (Candidate)
20:        <artifact-dependency artifact-id="A88"/>
21:        <artifact-type type="template-part"/>
22:      </artifact>
23:      <artifact id="E90" type="BusinessFlow">                 (Candidate)
24:        <artifact-dependency artifact-id="A88"/>
25:        <artifact-type type="template-part"/>
26:      </artifact>
27:    </artifact>
28:  </solution>
39: </asset>
```

図 5-7　図 5-6 中のオブジェクト間の関係の XML 記述.

5.4 企業ソフトウェア開発の実プロジェクトによる評価

5.4.1　プロジェクト概要と評価方法

提案したモデリング方法論，BPM-IE，及び，BPM テンプレートを用いる業務プロセス再利用方法を実際の顧客企業むけソフトウェア開発プロジェクトに適用して，提案方法と BPM-IE の有効性を評価した．

プロジェクトは流通業の顧客むけプロジェクトで，業務プロセス分析の対象業務は 11 業務である．業務プロセスモデリングのプロジェクトメンバは，プロジェクトマネージャ 1 名，リーダ 2 名，担当者 9 名で構成する．5.3 節で述べた業務プロセステンプレート開発方法を適用するため，別な要員 1 名を“事前準備”プロセスに割り当てた．

“事前準備”プロセスで，プロジェクトは表 3 に示すように 5 つの業務カテゴリに対して一連の業務プロセステンプレートを定義した．階層 1，階層 2，階層 3 のテンプレートを「受注」と「出荷」の 2 つの業務カテゴリに対して定義した．一方，「発注」「入荷」「在庫」の 3 業務カテゴリは，階層 1 と階層 2 を準備した．テンプレートの数と準備に必要な工数（人日）を表 3 に示す．階層 2 までであれば，テンプレートを準備する作業負荷が軽いことを確認した．

表 5-3　　準備の結果

業務カテゴリ		受注	出荷	発注	入荷	在庫
作成数	階層 1	1	1	1	1	1
	階層 2	4	9	3	5	3
	階層 3	13	16	-	-	-
工数(人日)		32	28	0.8	0.8	1.5

5.4.2　BPM-IE とテンプレートの定量的評価

前節の準備作業後，プロジェクトの 11 業務（サブシステム）で業務プロセスフローを開発した．全業務のプロセスフローの作成時間(人日)を，条件別に平均したデータを表 5-4 に示す．プロジェクトメンバに，(1)実際の作成時間，(2)BPM-IE は使うがテンプレートはなし，の場合の想定工数，(3)BPM-IE 無しかつテンプレートなし，の場合の想定工数，に関してヒアリングを実施した．

表 5-4　開発時間（人日）

	階層 3 までを準備 (2 業務平均)	階層 2 までを準備 (3 業務平均)	テンプレートなし (6 業務平均)	合計
(1)作成時間実績値	6.6	5.4	3.2	46.4(11 業務)
(2)BPM-IE あり，テンプレートなし	8.2	7.5	NA	38.8(5 業務)
(3)BPM-IE なし	10.6	10.0	25.1	76.3(11 業務)

11 業務のモデリングで集めたデータを基に，以下の効果を定量的に求めた．

(1) BPM-IE による生産性向上：業務プロセスフローモデリングの工数が，BPM-IE の利用で 24.4%削減

(2) BPM-IE とテンプレートの組合せによる生産性向上：業務プロセスフローモデリングの工数が，BPM-IE と階層 1, 2 のテンプレート利用で 45.8%削減，BPM-IE と階層 1, 2, 3 のテンプレート利用

で 47.1%削減.

生産性の向上が，再利用可能な業務プロセスフローモデリングの実際的な有効性を証明すると結論付けることができる．BPM-IE とテンプレートの組合せにより，改善効果は 2 倍，24.4%から 47.1%となる．したがって，業務プロセスフローを再利用するためにテンプレートを作成して使用することによって高い効果を得られる． テンプレートを使用することの効果は，業務プロセスの再利用の効果を明確に示唆している．

5.4.3 BPM-IE とテンプレートの定性的評価

プロジェクトのユーザから意見を収集して BPM-IE とテンプレートの効果を評価した．

(1) この BPM-IE は，統一的に記述された業務プロセスフローを保証するのに役立つ．

(2) この BPM-IE は，業務プロセスフロー中の内容（要素）を統一的に表現するのに役立つ．

加えて，ユーザのほとんどが BPM-IE とテンプレートを次以降の自分たちのプロジェクトでも使いたい，と述べた．

以上から，提案した BPM-IE とテンプレートが実際的に役立ち，かつ，効果的であると結論づけられる．

5.4.4 業務プロセス再利用方法の評価

業務プロセスの再利用方法を評価するために，図 5-8 に示すように，"受注"と "出荷"の成果物を分析した．この 2 つの業務カテゴリでは，階層 2 と階層 3 の両方のテンプレートが提供された．テンプレートを実際の顧客むけのソフトウェアに再利用するための業務プロセスモデルの変更に関して詳細を分析した．

⑴ 階層 2 では，すべての業務プロセスがそのまま再利用された．プロセス(Process)の追加とオブジェクト(ObjectFlowState)の追加が確認できた．但し，そのような業務プロセスモデルの拡張は，再利用されたコンポーネントを変更する必要がなく，提案方法の優れた拡張性を示している．

⑵ 階層 3 では，テンプレートで想定した業務範囲と合致したプロセスフローでは，業務プロセスの再利用率が 50%であった．テンプレートの詳細化は軽微な修正が主であった．しかし，想定した業務範囲が合致しない場合，再利用率は 20%未満に低下した．また，企業組織を細分化して追加（39 個），担当者ごとに業務データを個別に追加（337 個）していた．この分析は，階層 3 はその業務範囲が定義され合意されていれば，かなり再利用可能であることを示している．また，業務データの詳細化は，業務プロセス定義の後に行うと有益な効果が得られることも確認した．

結論として，提案した業務プロセスの再利用方法は，実際の業務プロセスモデリングに有効であると結論づけることができる．

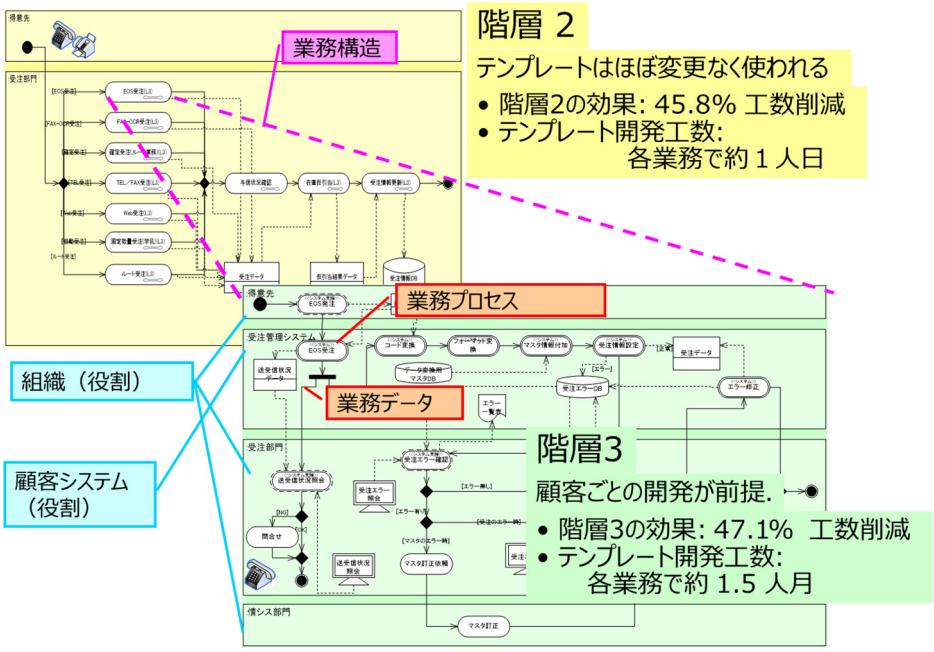

図 5-8 BPM-IE とテンプレートによる生産性向上の定量的評価

5.5 提案する業務プロセスモデリングの考察

本研究の貢献を関連研究との比較を通じて議論する.

(1) メタモデル駆動業務プロセスモデリング方法論の貢献

提案する方法論の独自性はメタモデルとメタモデル駆動モデリングにある．業務プロセスモデリングの最も一般的な表記は ISO / IEC OMG BPMN[64] [32]である．BPMN は明確に定義された表記であるが，実用上のいくつかの制限や問題が指摘されている[47]. 本研究は，メタモデル駆動型ビジネスモデリング方法論の次の 2 つの技術における問題に貢献している.

a) 業務プロセスモデリングのための業務メタモデル : BPMN は UML メタモデルに基づいたメタモデルを持ち，表記のために BPMN エンティティのエンティティと関係を定義する．しかし，業務プロセスモデルの具体的な意味論は定義されていない．図 5-1 に示したように，本研究では “BusinessFlow”から始める業務メタモデルを定義した．これにより具体的な方法で業務プロセスモデルを定義することを支援できる．本研究のアプローチは BPMN の拡張と言うことができる.

b) 業務プロセスモデリングを実際に機能させるための一連の支援技術 : BPMN は BPD（Business Process Diagram，業務プロセス図）に焦点を当てており，業務用語と組織構造の定義を支援していない．提案した方法論は，業務用語，業務プロセスの特性，及び，組織構造の定義を支援する．この支援は業務プロセスモデルとその運用コンテキストを明確に定義するために必要である．業務プロセスモデリングを実際に機能させるためには，業務プロセスモデリング表記法以外にもさらに支援が必要であることを，本研究が示唆している.

(2) BPM テンプレートと BPM-IE(統合環境)による業務プロセス再利用方法の貢献

提案した業務プロセス再利用方法は以下の２つの技術で業務プロセスの再利用に貢献する.

a) 業務プロセスか可変性の構造的分解方法: 業務プロセスモデルをトップダウンにモデル化するのが

一般的である[15][87]．しかし，指摘されているように[53]，多くの業務プロセスモデルは類似性を持つ．そのため，プロダクトラインソフトウェア工学と同様に，業務プロセスモデル間の共通性と可変性を識別して分離する必要がある[8][70]．本研究では，可変性を一連の基準で分解するために業務プロセスモデルの 3 階層のフレームワークを提案した．これは業務モデルを再利用するための基礎となる．

b) BPM (Business Process Model)テンプレート：業務プロセスモデルの再利用を支援するために，著者らは 3 階層のフレームワークに基づいてテンプレートを定義した．特定のドメインが特定のテンプレートを必要とするという意味で，一部のテンプレートはドメイン固有のものである可能性はあるが，テンプレートは再利用のために具体的かつ実体的な業務プロセスコンポーネントとして機能する．実証的な研究により，提案した再利用方法が業務プロセスモデリングの生産性を 46%向上させることを実証した．分解駆動アプローチのケーススタディでは，最大 50%の重複が排除されたと報告されている[53]．これらの統計が，業務プロセスモデリングにおける再利用のベースラインを示唆している可能性がある．

c) 業務プロセス再利用のための BPM-IE：業務プロセスモデリングは複雑な作業である．従って，ツールの支援は不可欠である．業務プロセスをモデリングするために多くのツールがある[1]．しかし，ほとんどのモデリングツールはモデルの描画に焦点を当てている．再利用をサポートするツールはほとんどない．本研究で提案した BPM IE は，BPM テンプレートとリポジトリと共に業務プロセスモデルの再利用を支援する．実証的研究により生産性が 46%向上することを実証した．実際の企業ソフトウェア開発における実証的研究から，本研究の BPM IE の有効性を確認した．

6 Web サービス開発むけパターン体系

6.1 パターン体系

パターン体系を Buschmann らは文献[7]の 5 章において次の通り定義している：「ソフトウェアアーキテクチャのためのパターンのコレクションであり，その実装のためのガイドラインやソフトウェア開発における実際の利用やガイドラインを含んでいる.」

本研究で開発したパターン体系はこの定義の内容に加え，企業でのソフトウェア開発体制に合わせた情報やガイドラインを取捨選択して盛り込んでいる．以下に詳細を示す.

6.1.1 対象範囲

本研究のパターン体系がカバーする範囲は Web サービスの実装に XML を利用するビジネスアプリケーションであり，その中でもサーバ側の開発に必要な技術に絞っている．これはアプリケーション開発の中で XML を利用することで変更が生じ，ノウハウを提供しなければならない技術がクライアント側（ユーザインタフェース側）ではなくサーバ側に多くあると考えたためである．実際，ユーザインタフェース側は各社のさまざまなコンポーネントやアプリケーションが提供されており，これらを使えば要求を満たせる場合が多い.

6.1.2 パターン体系の構成

提案するパターン体系は個々の技術ノウハウを相互にリンクしたドキュメントとして提供し，膨大なノウハウから必要なノウハウだけを効率的に見ることができる．開発の状況に応じて以下の様々な入口（アーキテクチャ，テンプレートモデル，詳細ノウハウ，コンポーネント，開発方法論）からリンクをたどり，粗粒度のノウハウから細粒度のノウハウに至ることができる．一覧から直接細粒度のノウハウに至ることもできる.

同じビジネスアプリケーションの開発に関わる人間でも，例えばプログラマと SE ではアプリケーションに対する観点や必要とするノウハウが異なる．上記のように様々な入口と経路を設けることで，全ての人が「最適なノウハウに最短で」到達できる体系を目指した.

また細粒度のノウハウが詳細設計や実装に関するノウハウについては，そのサンプルコードとコンポーネントをそのまま利用できる形で提供している.

以下にパターン体系を構成する 5 つの入口を示す（図 6-1）.

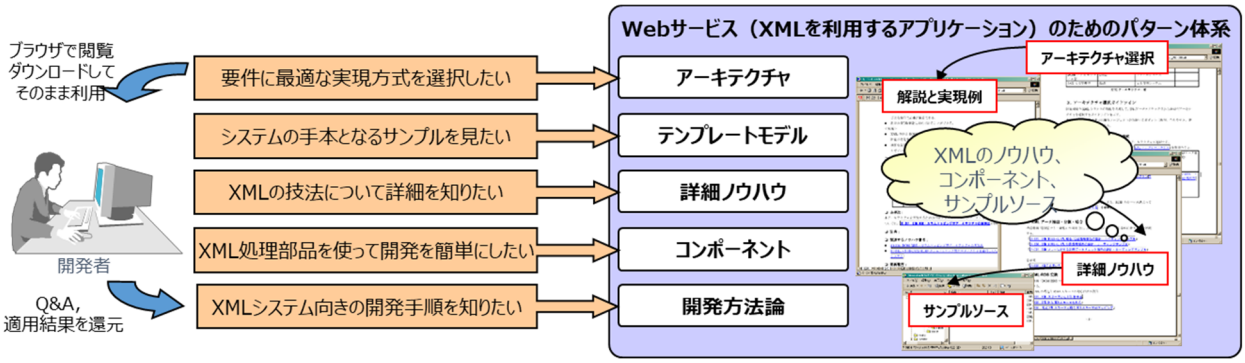

図 6-1 パターン体系の全体構成

(1) XML システムのアーキテクチャ

XML を入出力とするシステムの概要モデルを解説している．XML 自体を処理する方式とデータを格納する方式をそれぞれ分類して特徴やトレードオフを解説している．システムの要求に応じて方式を選択するためのガイドラインも提供している．また各アーキテクチャで使用できる実際の製品名を挙げている.

ここから詳細化したアーキテクチャやテンプレートモデル，詳細ノウハウ，コンポーネントに至ることができる．

実装した内容は，分類整理した XML システムアーキテクチャのうち，実際に商談に適用を推奨する 8 種類とした．後述する評価では企業内のシステム開発では十分であると判断できた．

(2) テンプレートモデル

システムのタイプ別にアプリケーションの実現例とサンプルコード，および，それを用いた開発に必要な作業手順を提供している．

ここから関連するアーキテクチャやテンプレートモデル，詳細ノウハウ，コンポーネントに至ることができる．

実装した内容は先行商談 4 例のテンプレートモデルで，本パターン体系の適用した実商談をテンプレートモデル化して拡充した．

(3) 詳細ノウハウ

XML アプリケーションの開発に必要な個々の技法に関する知識とサンプルコードを提供している．ノウハウはテーマ別（RDB 格納法など）に分類してある．

ここから関連するコンポーネントに至ることができる．

実装した内容は当初 13 種類で XML-RDB マッピングやトランザクション管理など多岐に渡る．

(4) コンポーネント

使用頻度の高い XML 処理を部品化しコンポーネントとして提供している．あわせて設計情報，動作保証情報，使用法を示すサンプルコードも提供している．コンポーネントの設定を外部ファイル（XML）で与えてカスタマイズを可能にしてある．

実装した内容は当初約 30 種類であり，ツールも含めて順次拡充していった．

(5) 開発方法論

実践経験のあるオブジェクト指向開発方法論[92]をベースに XML を利用するアプリケーション開発のための作業手順（分析作業からテスト作業まで）とドキュメント（UML ベース）を開発した．

開発方法論はパターン体系が提供する 5 つの入口やパターンの実際の使い方を示している．この方法論は入口やパターンをどのソフトウェア開発フェーズで使うか，各開発フェースでどのように使うか，開発フェーズではどこまで（例えばどの詳細度まで）作業すべきか，という情報を提供する．また，代表的なアプリケーション開発を例に全ドキュメント事例も提供している．

このパターン体系のパターンのたどり方の例を簡単に示す（図 6-2）．

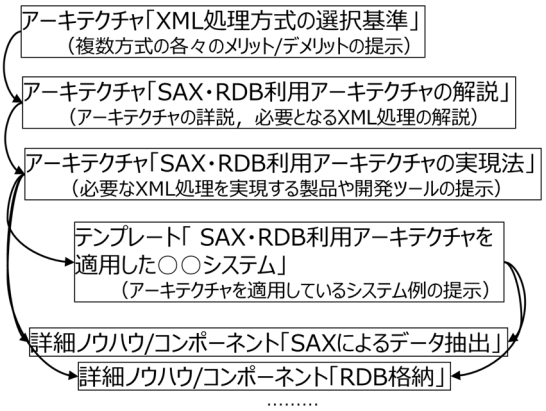

図 6-2 パターンのたどり方

入口「XML システムのアーキテクチャ」から「XML 処理方式の選択基準」が書かれたパターンにたどりつくことができる．このパターンに従って処理方式を選択し，パターンに張られたハイパーリンクをたどって

選択したアーキテクチャ（例えばSAXとRDBを利用するXML処理アーキテクチャ）やその実現方法に関するパターンを参照できる．さらに，このパターンからアーキテクチャを利用したテンプレートモデルやアーキテクチャを利用するための詳細ノウハウ（例えば図 6-3 の SAX に関するノウハウ）やコンポーネントのパターンにたどり着くことができる．

4.3 節であげたパターン体系で提供すべき情報のうち，4.3.1 節で示したXMLの技術的特徴に関する情報は，アーキテクチャ，ノウハウ，コンポーネントを入口とするパターンとして提供している．4.3.2 節で示したな開発方法や作業手順に関しては，開発方法論で情報を提供している．

6.1.3　パターンテンプレート

パターンを統一された形式で記述するために，パターン体系用のパターン記述テンプレート（以降パターンテンプレート）を用意した．記述する項目は社内での利用に必要なものを列挙した．

パターンの例を図 6-3 に示す．

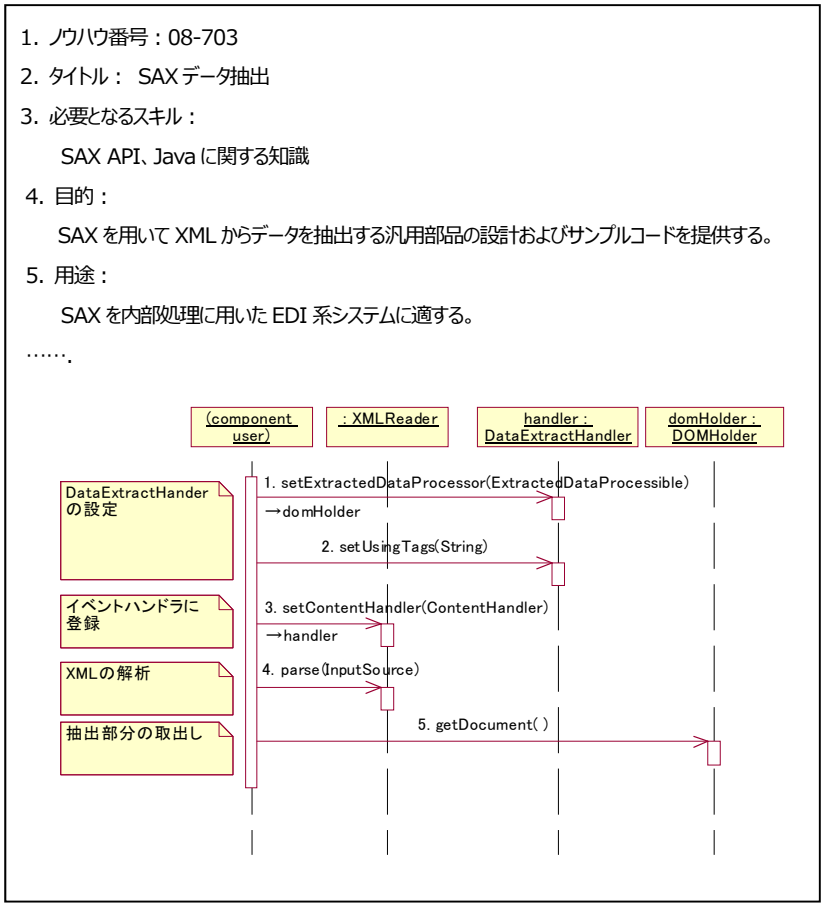

図 6-3　パターンの例

本研究のパターンテンプレートが持つ項目の中で，一般のパターンテンプレート（たとえば文献[7]の 1 章に示されている記述テンプレート）にない項目として「必要となるスキル」がある．この項目には例えば「DOM API，JDBC，Java に関する知識」と書かれている．この項目を設けることで習得技術の異なる各部署の開発担当者は現在の技術知識で読むべきか否かを判断できる．なお，前述の一般のパターンテンプレートに記述する項目として列挙されているのは，名前，別名，例，前提，課題，解決策，静的側面，動的側面，実装，補足，

バリエーション，適用例，結論，参考，である．

また本研究のパターンには，GoF デザインパターン[25]で「動機」に書かれる内容，すなわちパターンが解決すべき問題の文脈をほとんど書いていない．代わりに 6.1.2 に示した 5 つの入口のうち「アーキテクチャ」，「テンプレートモデル」にこの内容が書いてあり，説明文中，図，表，リストから適宜ハイパーリンクでパターンに到達できる．またこの形式により関連するパターンにより簡単に到達できると考える．

その他パターンの項目として，タイトル，ID 番号，目的，用途，概要，外部仕様，内部仕様，関連するパターン（同時に本文中で適宜関連パターンへハイパーリンクを張っている），効果などを用意した．

詳細設計や実装に関するパターンには，理解の助けのためにサンプルコードを動作する完全形で提供した．サンプルコードはそのまま流用することも可能である．またすべてのサンプルコードには UML などで記述した設計情報を用意しておりサンプルコードの利用プログラム言語（Java）とは異なるプログラム言語の再利用を助けている．

6.2 開発作業

パターン体系は現場での経験を元に内容をまとめるか，仮説をもとに適用範囲や内容をしぼり，実際に適用してフィードバックしなければ，活用できない．パターン体系を開発する部門の技術者に XML に関する経験が多くなかったため，次の 3 ステップで開発を行った．

(1) 知識の集約，第 0 版のパターン体系の作成，サンプル業務システムへの適用
(2) XML を用いる業務システム開発の支援，パターン体系の適用，実際に必要で一般性のある技術要素のパターン化
(3) パターン体系へのフィードバック

以下，それぞれのステップの詳細を述べる．

6.2.1 第 0 版のパターン体系の作成

6.1.1 で述べたように本研究のパターン体系の対象はアプリケーションシステムのサーバ側である．パターン開発メンバの多くは，今回の開発以前にサーバ側のアプリケーション開発支援を数多く経験し，また他のパターン体系の開発と試行も経験している[92][83][93][86]．その経験からパターン体系の開発に関して，仮説（6.1.2 で示した 5 つの入口）を持って取り組んだ．

XML の基礎技術や応用方法についての知識は研究開発を行ってきた部署やすでに数多く出版されていた書籍や Web 上の文献から得た．

これらの経験と知識をもとに，第 0 版のパターン体系を作り，それを実際に適用した．このパターン体系作成ステップでは次の 3 つの作業を行った．

(1) 知識の集約
XML 技術に関する文献と，開発メンバがこれまで開発支援してきた経験，特に DB に関する経験から，XML の処理方式や DB 格納法を組み合わせ， XML を用いるシステムのアーキテクチャをまとめた．同時に XML に関する技術要素をリストアップした．
リストアップした技術要素からパターンとすべき内容を選び，パターン体系の入口（アーキテクチャ，テンプレートモデル，詳細ノウハウなど）とテーマごとに大項目番号を振り，個々のパターンごとに小項目番号を振った．
ここで付けた番号は変更を不可としており，この番号で別のパターンから参照される．

(2) パターンの作成
パターンを作成し，開発メンバ内レビューや第三者レビューを繰り返した．

(3) サンプル業務システムへの適用

具体的な業務を想定し，そのシステムの開発に作成したパターンを適用した．パターン開発者とは異なる第三者に開発してもらい，パターンテンプレートの不備，解説している技術要素に関する不備を洗い出した．

6.2.2 パターン体系の適用

第0版のパターン体系をWeb上で公開し社内で自由に利用できるようにした．同時にパターンの開発に携わるメンバがいくつかのシステム開発プロジェクトの方式設計作業に参加し，技術情報の提供を行った．提供した内容は， XMLの適用の可否やXML処理方式を決定するための情報，開発手順，XML処理方式や詳細設計のプロトタイプ， コンポーネントなどである．またドキュメントやXMLボキャブラリ案の提示なども行った．

6.2.3 パターン体系へのフィードバック

プロジェクトの方式設計作業に参加したことで，多くのフィードバックが得られた．

本研究で提供したノウハウには，パターン化していなかったノウハウや，パターン化していたが内容に不足があったノウハウがあった．これらのノウハウについて，内容を検討しパターンの洗練や追加を行った．

またパターン体系の全体構成やパターンテンプレートの見直しが必要なフィードバックを受け，次の(1) から(6)の変更を検討した．

(1) ダウンロード形式の変更

パターン体系全体をひとつにまとめたダウンロードファイルを新たに公開した．

当初，「利用者はパターン体系をある程度オンラインでたどっていき，必要な部分だけダウンロードする」と判断して，小単位でパターンをひとかたまりにダウンロード単位としていた．例えば詳細ノウハウに関するパターンをテーマ別にひとかたまりにしてダウンロード単位とし，アーキテクチャやテンプレートに関するパターンはそれぞれ一つのパターンをダウンロード単位としていた．

しかし実際は，複数に分かれたダウンロードファイルをすべてダウンロードし，手元に置いておくという使われ方が多いことが分かった．

(2) 性能に関する情報の追加

XML処理方式ごとの性能に関するデータを求められた．基本的なXML処理性能測定値や性能改善ノウハウを追加した．

当初，処理性能測定値を載せなかったのは次の理由による．処理性能測定値は，それぞれの開発プロジェクトが実運用かそれに近い環境（OSの種類，メモリサイズ，CPUの種類，数，動作クロック，ネットワーク，ハードディスク性能）で測定する必要があり，開発するシステムの要求ごとに測定すべき観点も異なる．またXML処理方式単体の処理性能ではなく，実際のシステムに近いコンポーネントの組み合わせで処理性能を測定する必要がある．これらのさまざまな要因の組み合わせを測定するには工数がかかりすぎる．

開発プロジェクトごとに性能測定するのが最適，という方針は変わっておらず，基本的なXML処理方法を除き，性能に関する情報は加えていない．

(3) サンプルコードのコンポーネント化

使用頻度の高いXML処理を部品化しコンポーネントとして提供した．使用法のドキュメントや，サンプルコード，シーケンス図も追加した．

(4) 動作確認情報の付加

上記のコンポーネントについて動作確認がとれた環境の情報（ミドルウェアやXMLプロセッサなどの種類やバージョン）を追加した．

(5) パターンの分割

コンポーネントとして部品化した処理に関するパターンを2種類に分割した．一つはコンポーネントを使うユーザの観点でノウハウを記述したパターン，もう一つはコンポーネントで行っているXML処理内容

についてのノウハウを記述したパターンである.
部品化したパターンのいくつかは,「処理が複雑でパターンの対象読者が誰なのか明確でない」という指摘を受けていた. この指摘への対応として行った.

(6) ツール群の提供
実際の開発プロジェクトで使われていたツールが便利であったため, 許可を得て汎用化し公開した.
また開発プロジェクトからの要望が多かった, コンポーネントの設定を行う外部ファイル (XML) のエディタを開発した.

6.3 開発体制

提供したパターン体系はXMLを利用するアプリケーションの開発に関するノウハウを集めている．実用に耐えるパターン体系に成熟させるために実際の開発プロジェクトに参加して効率的に必要な情報（求められているノウハウ，提供しているパターンの技術情報の過不足など）を収集する必要がある．そのためには本研究の提案者らを含め次のチーム構成が必要だった．

6.3.1　パターン体系開発チーム

業務開発支援経験を持ち，第0版のパターン体系を作成．実際の開発プロジェクトに参加し必要な情報を得てパターン体系を成熟させていくチーム．提案者らはここに属する．

6.3.2　商談支援チーム

開発プロジェクトとパターン体系開発チームの橋渡しを行うチーム．開発プロジェクトからの開発支援要請を受け，必要に応じてパターン体系開発チームを開発プロジェクトに参加させる．

通常は開発支援要請を受けても，商談支援チームが持つXML関連の技術情報（パターン体系を含む）に基づき質問に回答し，また，適切な開発部隊やパターン体系を紹介する．開発プロジェクトにパターン体系開発チームが必要だと判断する，またはパターン体系開発チームにとってプロジェクトへの参加が望ましいと判断すると，パターン体系開発チームを開発プロジェクトに紹介する．

6.3.3　コンテンツ展開チーム

パターン体系を含む，XML関連の技術情報を全社に周知，展開していくチーム．技術情報の説明会やパターン体系を適用した業務テンプレートの開発を行う．

また商談支援チームと業務開発に関する情報を共有しており，これに基づいたパターン体系のレビューも行っている．

6.3.4　教育チーム

パターン体系などXML関連の技術の教材開発と教育体系の整備を行うチーム．たとえば，さまざまなスキルレベルを持つ人に対応できる形で，パターン体系のe-Learning化を行っている．

6.3.5　コア製品開発チーム

DOM，SAXなどXMLプロセッサやXML関連の仕様など，XMLを利用するアプリケーションの開発の基盤となる技術や情報を開発調査しているチーム．

パターン体系開発チームが実際の開発プロジェクトに入っていて，XML処理方式の性能比較の実データやXMLの新しい仕様の詳細など基盤時術に関わる情報が必要なときこのチームからの支援が必要となる．

ここに挙げたチームが互いをサポートし合うことで，それぞれの目的，パターン体系開発チームにとってはパターン体系開発を達成することができる．

XMLは常に新しい技術情報が出てくる状態であるため，それに追随するためにも常にパターン体系の改訂が必要である．この開発体制がうまく機能したため，適度な負荷で十分な情報やフィードバックを収集することができ，ほぼ3ヶ月ごとに改訂を行うことができた．

6.4 パターン体系の評価

6.4.1　利用状況収集と効果

パターン体系を社内に公開し，そのアクセス数，適用目的，適用状況等に関して調査した．調査は，アクセス時のアンケートによる所属や利用目的等の収集，時間をおいて行った確認アンケートによる利用方法や効果等の情報の収集を基本とした．これらのアンケートは，本研究の提案者らが所属する会社の様々なノウハウを共有するシステムとして提供される．このアンケートに追加して，いくつかの代表的な適用プロジェクトへのヒアリングを実施した．

パターン体系は過去 16 ヶ月で数千件のアクセスがあった．利用法で多いのは，提案書を含むドキュメント作成の参考，プロトタイプ構築，新技術の導入の参考，コンポーネントとしての利用，XML 技術の学習教材としての利用などである．また設計を採用し他の言語で実装した例もある．

現在 50 プロジェクトに適用済みで，大きなコスト削減効果があった．パターンを体系化したことで様々な観点で利用者が情報へたどり着くことができるため，上記のとおり様々な形で利用され，コスト削減効果を高めたと考える．

また，プロジェクトの立ち上げ時の学習教材としての利用も数多くあり，当初の目的である新技術の社内普及という観点からも効果が確認できた．

6.4.2　パターン体系の周辺への展開

パターン体系を全面適用したアプリケーション事例をコンテンツ展開チームが開発して社内に公開した．こちらも再利用によるコスト削減効果をもたらしている．

また普及への別なアプローチとして，このパターン体系を元に e-Learning 教材を開発した．こちらは教材提供から約 1 年間で約 1,000 人が利用し，社内の開発要員の新しい技術の獲得に貢献できた．

6.5　パターン体系の考察

6.5.1　パターン体系の構成

提案するパターン体系に 5 つの入口を設け，様々な観点から必要なパターンにたどり着ける構成とした．この構成の意義と課題について考察する．

評価で述べたようにこの構成によってパターンの利用機会が増えており，多くの入口を設けた効果があったと考える．

アーキテクチャ，テンプレートは，上流の開発作業で参照する入口として用意する必要があった．提案方法は新しい技術を導入するプロジェクトを進める上での技術リスクや不安を取り除く上で必要であった．また，詳細ノウハウ，コンポーネントは，すでにこのパターン体系の利用経験のあるユーザがリファレンスマニュアルのように直接詳細なノウハウの確認に使っており，これらも用意する必要があった．

課題について言及は難しいが，利用者のフィードバックから新たな入口の議論は起こっていない．

6.5.2　開発体制

パターン開発に参加した各チームが様々な役割を分担したことで，開発（プロジェクトの方式設計作業への参加やプロジェクトへのヒアリングを含む）により多くの時間を割けた．この結果，不足している情報やパターンの理解のしやすさなどのフィードバックを得て，それに対応する形でパターンの追加，洗練を繰り返すことができた．現在までに 7 回のパターン全体の定期的な改版を繰り返している．

特に，商談支援チーム，コンテンツ展開チームからのフィードバックがなければ，パターン体系や各パターンを成熟させることはできなかった.

ただし，パターン体系開発の初期段階からしばらくの間は各チーム自身がパターン体系の理解とパターンの内容の理解を行う必要があった．この期間を短くすることは今後の課題である．解決策として，例えばパターン体系開発の初期段階から参加する，などがある.

6.5.3 開発・展開に必要な要件

XML 技術のパターン体系を開発・展開し効果を確認できた理由として，本研究のケースでは次の要因があると考える.

(1) 開発体制が整ったこと
(2) 専任の担当者が就いたこと
(3) トップダウンのプロジェクトに組み込めたこと．XML を普及させるという全社的な動きと連動できた
(4) XML に特化したこと．そのため短期間で提供内容の充実を図れた

6.5.4 開発プロジェクトの理解の促進にむけた改善点

我々が開発プロジェクトに入ったとき，またパターン体系を利用者に説明したとき，こちらの想定していなかった質問や意見をいくつか受けた．これらから，次の点を改善すべきと考える.

(1) 対象範囲外との連携方式の提示

クライアントサーバシステムのクライアント側で XML を利用する方式は，提供したパターン体系の対象範囲外である．しかし対象範囲外のシステムとの連携に関する質問を何度も受けた.
企業で利用されるパターン体系に特有の要求であろうが，パターン体系の範囲内と範囲外の境界部分についてもノウハウを提供する必要がある.

なお，本研究のパターン体系に関しては，コンテンツ展開チームの開発したパターン体系適用例の中で，境界部分のノウハウを提供した.

(2) 企画作業 （方向付け）

本研究のパターン体系は分析作業からテスト作業までの方法論を提供する．しかし，顧客からいつから XML の使用を本気で検討すべきかなどの質問を受けた，といった，本研究の対象範囲外の質問への回答例を尋ねられた.

このような質問に答えられるところまでパターン体系の範囲を広げる必要はないと考えている．仮に本研究のパターン体系を含む知識共有体系を作ったときは，本研究のパターン体系からのリンクを用意すべきである.

7 Web API の特徴を捉える品質モデル

7.1 対象とする品質特性の分析

7.1.1 API コンシューマの開発の観点からの分析

従来，Web API のユーザビリティなどの品質特性[51] [52] [58]の概念が個別に提示されているが，ソフトウェア製品の品質モデルの国際規格 ISO 25010[34]の分類と照らした議論は不十分である

品質を評価する対象に関して，一般に，Web API を含む API は，その“インタフェース定義”と“実装”に分けられる．2.4 節の図 2-2 が“インタフェース定義”を例示し，図 2-3 が“実装”を例示する．さらに，品質特性分類に ISO 25030[36]で規定されている製品品質の外部品質と利用品質の概念がある．例えば，利用品質の一つである効率性(efficiency)は，開発者にとっては Web API を利用した開発の効率性となり，アプケーションユーザにとってはそれを利用して，ユーザがタスクを遂行する効率性を意味することから，異なる定義となる．このことは，Web API の品質特性として効率性の再定義が必要であることを意味している．

さらに，本研究では，Web API を利用したアプリケーション開発の初期工程で，実際に Web API をアクセスする前に判断すべき品質特性に着目する．実際に Web API をアクセスするには一般に ID 取得などの手続きや使用料支払いなどの費用が必要となる．このような作業を行う前に，その必要性を判断し，また，利用上のリスクに応じて対応作業を予測できることが望まれている．これによって，その後のアプリケーション開発作業の見積りの適正化や工数確保等の準備ができる．そのため，Web API の“インタフェース定義”からその品質を評価する対象とし，“API コンシューマからみた外部品質”の観点で議論する．

7.1.2 非機能要求モデルに基づく品質特性の構造モデル化

API コンシューマのパースペクティブから Web API の品質特性を構造化し，ISO 25010 の品質特性に対する拡張性を議論するために，非機能要求の構造モデルである Chung らの非機能要求フレームワーク[9]を導入する．

非機能要求フレームワークに基づき，ソフトゴールと課題（非機能要求フレームワークではクレームと呼ぶ）と品質特性との関係を定義した結果を図 7-1 に示す．表記法は非機能要求フレームワークに基づいているが，後述するように，一部拡張を行った．

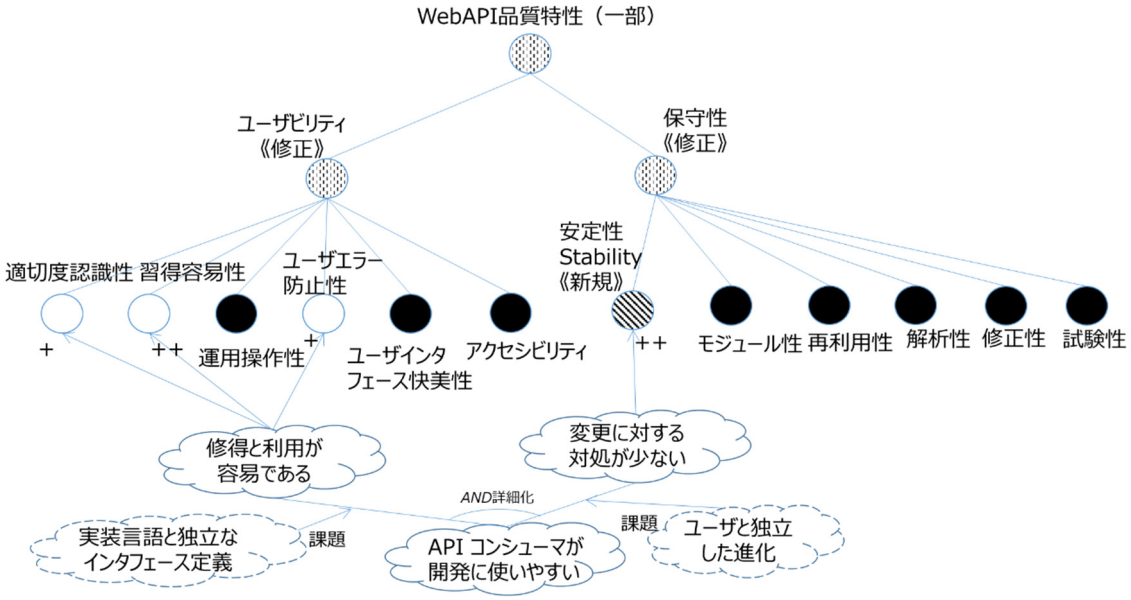

図 7-1 API コンシューマの開発初期の課題に対応する品質特性

本研究では“API コンシューマが開発に使いやすい”をソフトゴールとして，前述したシステム API と異なる Web API の 2 つの特徴“実装言語と独立なインタフェース定義”と“ユーザと独立した進化”を Chung らの課題と捉える．これらを計測するための品質特性を ISO25010 のユーザビリティと保守性に基づき定義した．図の下部に API コンシューマが開発初期に Web API の品質を判断するためのソフトゴールを定義し，先に記述した Web API の特徴が提起する課題を関連づけている．Chung らの表記を用いて，ソフトゴールが副特性に positive に作用することを+で，negative に作用することを-で表現し，その度合いを＋と-の数で表した．図の上部は ISO25010 の特性の一部と副特性の階層を記述した．図中，拡張した表記法として，API コンシューマが開発初期の課題に貢献する副特性は白色，関与しない副特性は黒色，本稿で新規に貢献するとして提案する副特性は斜め線で示した．結果として現在の ISO25010 を修正した特性は縦縞で示した．また，品質特性の≪修正≫と≪新規≫を UML(Unified Modeling Language)のステレオタイプで表記した．

インタフェース定義を対象にして，ソフトゴール“API コンシューマが開発に使いやすい”を前述の課題に対応するソフトゴールに分解し，“習得と利用が簡単である”，“変更に対する対処が少ない”を定義した．

前者は品質特性“ユーザビリティ(Usability)”の副特性の“習得容易性(Learnability)”が強く関与する．ユーザビリティの他の副特性に関しては，Web API がアプリケーションプログラムからアクセスされるため，アプリケーションユーザとのユーザインタフェースを主に想定した運用操作性，ユーザインタフェース快美性，アクセサビリティは関与しないと判断できる．適切度認識性とユーザエラー防止性は，“インタフェース定義”だけでなく“実装”の評価も必要となるので，関与の度合いが習得容易性よりも低いと判断した．

後者は，品質特性“保守性(Maintainability)”が関与すると判断した．しかし，“インタフェース定義”を対象に，ネットワーク越しのリソースをアクセスするため実際のソフトウェアのプログラムコードやバイナリコードが入手できない状況に適用できる副特性はないと判断した．一方，本研究では，“インタフェース定義”を対象に仕様の変化への対応を可能とするための Web API の新たな品質特性として“安定性(stability)”を提案する．安定性の定義は後述する．なお，“実装”を対象とすれば，プログラムコードが入手できない状況であっても解析性，修正性，試験性は関与すると考えられる．

安定性と関連する品質特性として，保守性から発展した“進化可能性(Evolvability)”がある．例えば，システムが要求，環境，実現技術の変化に対応できる性質として定義されている[10]．これは，Web API が短期間に，あるいは，継続的に変更される場合も発生している[74][85]ことから，Web API を利用するとき変更を肯定的かつ積極的に受け止めて対応すべきとの意味づけがされている．これに対して，API コンシューマのパースペクティブからは“変更への対処が少ない”ことが求められていることから，それに対して positive に強く関与する品質特性を明確にするため，安定性を採用する．

7.1.3 開発初期の課題のための Web API 品質特性の定義

図 7-1 と前述の議論から，Web API と従来のシステム API の特徴の差を捉えた品質特性として，Web API を用いたアプリケーション開発の初期に，API コンシューマにとって重要となる品質特性として以下に定義する二つを導出した．

(1) 習得容易性（Learnability）：習得容易性とは，ある特定のユーザがある特定の目標を達成するために Web API を使用することによる有効性，効率性，リスクと満足度の程度である．

ISO 25010 で定義されているソフトウェア製品に対する品質特性の習得性の拡張であり，ユーザビリティの一特性である．3.3.2 で述べたように，システム API での API ユーザビリティの重要性は実証されており，また，Web API でも API ユーザビリティの重要性が認識されつつあるため，ユーザビリティを対象とする．システム API がその実装言語でインタフェースを定義するので一意の理解となるのに対し，Web API のインタフェース定義は実装言語と独立に REST の原則に従うリソースの表現として定義し，同一リソースでも複数の表現をとり得る．この性質の違いに着目して，ユーザビリティの中でもその起点となる習得容易性を本稿の対象とする．

(2) 安定性(Stability)：安定性とは，Web API のインタフェースがある時間にわたって変化する度合いである．これは，Web API コンシューマが変更に対応するための開発期間への制約となる．この定義は，ライブラリの安定性[71]を Web API に拡張したものである．3.3.4 で述べたように，Web API が変更されるとそれを利用するアプリケーションも変更せざるをえない状況が出てくる．Web API はシステム API と異なり，API コンシューマと独立にリモートで開発されて，実行時に“ユーザと独立した進化”をする特徴を持つ．これは，企業情報システムに大きなリスクとなることから，開発の初期段階で Web API の変更を予測する必要がある．このために，本稿では安定性を対象とする．

7.2 Web API の品質評価モデルと品質評価方法

7.2.1 メタモデル駆動型 Web API 品質評価モデル

関連研究で述べたように，Web API を用いる開発の広まりとともに，本研究を含む様々な品質特性の議論が期待される．そのため，Web API の品質モデルの全体像として，概念，スコープ，語彙を定義するためのメタモデルとして，Web API 品質特性メタモデルを定義する．次に，各 Web API 品質特性の品質モデルをメタモデルのインスタンスとして定義する．メタモデルは Web API 品質モデルの基礎となり，モデルの一貫性を保証するために用いる．

7.2.2 Web API 品質評価メタモデル

ソフトウェア開発の視点から捉えた品質特性とその測定の共通認識の基礎として，ソフトウェアの汎用的な品質評価のメタモデルをまず定義する．ISO 15939 測定情報モデル[31] (プロジェクトで必要な情報を測定可能なプロセスやプロダクトと関係づけるモデルの定義)を参照し，品質特性とその評価に必要な要素のモデルを図 7-2 に示す．

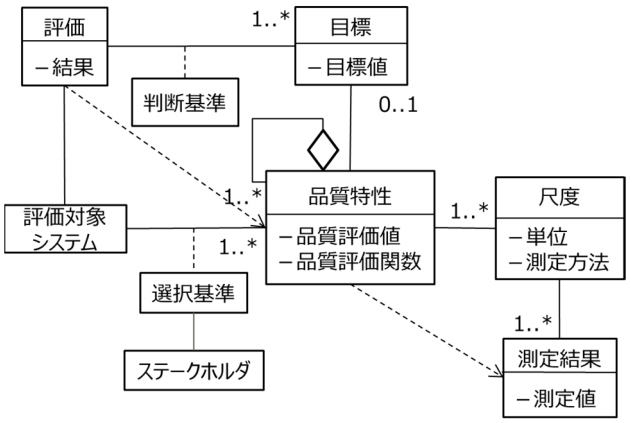

図 7-2 Web API 品質評価メタモデル

品質特性は階層的に構成しその最下層の品質特性が測定対象となるように定義する．評価対象システムは必要とする品質特性を選択し，測定結果と目標値を比較して品質を評価する．

7.2.3　習得容易性の評価

7.2.3.1　習得容易性の評価モデル

Web API を用いたアプリケーション開発では，API コンシューマは使用する Web API に対してある程度習熟している必要がある．そのため，開発に利用する Web API を新たに選択する場合，習得しやすいものを選択することが望まれる．本研究では，Web API の習得しやすさ(以後，習得容易性と呼ぶ)に注目して品質特性を具体化した．前節に示したメタモデルの具体化に際しては，開発初期の動作環境が整備される前であっても利用可能な情報，例えば API ドキュメントを用いた尺度を定義することを目指した．

Web API の習得容易性の品質モデルを図 7-3 に示す．この品質モデルでは，習得容易性を以下の 3 つの品質特性に詳細化した．各品質特性の概要と品質モデルの中で取り上げた理由を以下に示す．

(1) Web API サンプル充実性：Web API がその仕様だけでなくサンプルでも示されている程度．API コンシューマが Web API の選択や習得し始めようとした際にまず参照するのがサンプルだと想定したためである．
(2) Web API 標準適合性：Web API が設計原則の標準である REST [21]に則っていること．標準設計原則に則った Web API であれば，その仕様の理解が容易で，開発にも利用しやすいと考える．
(3) サポート充実性：API コンシューマに対するサポートが充実している程度．Web API に限らず何かを習得しようとする際には，疑問点や意図したとおりにならないことが発生する．そのような場合に効率的に解決できることはアプリケーション開発に重要であるため，本品質特性を取り上げた．ここで，サポートとは API コンシューマ向けの情報提供サイトや FAQ 等である．

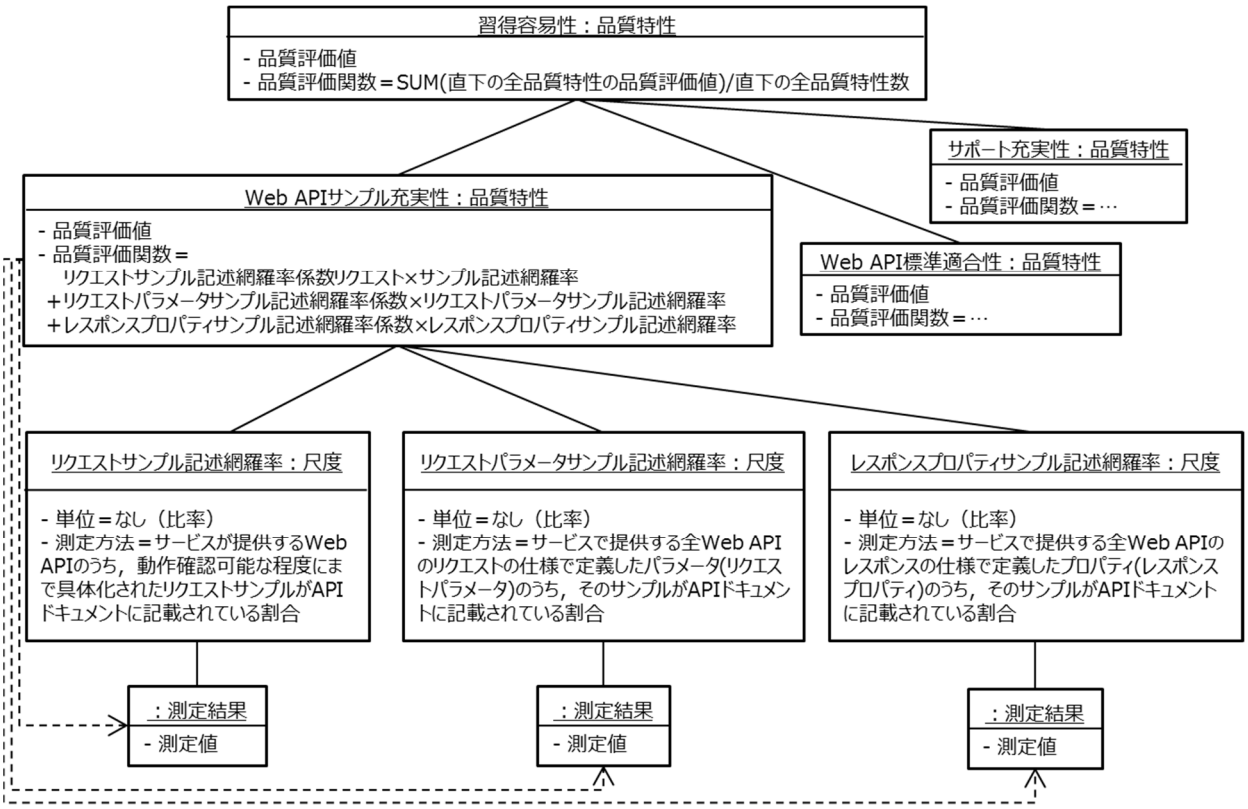

図 7-3　サービス評価品質モデルの一部(習得容易性)

これら 3 つの品質特性のうち，以降では“(1) WEB API サンプル充実性”について議論する．この品質特性を取り上げた理由は，品質特性の達成度合いの機械的な判断に最も適していると判断したからである．

7.2.3.2 習得容易性の評価尺度と評価方法

3.3 節で述べたように API のサンプルが習得容易性に寄与することが Uddin ら[84]や Sohan ら[77]により実証されている．本研究においてもサンプルの重要性を認識し，前節の品質特性「Web API サンプル充実性」に注目して表 7-1 に示す 3 つの尺度を定義する．

表 7-1　習得容易性の尺度

		粒度	
		全体	構成要素
対象	リクエスト (Req.)	(1) リクエストサンプル記述網羅率 (ReqSampleCoverage)	(2) リクエストパラメータサンプル記述網羅率 (ReqParamCoverage)
	レスポンス (Res.)	–	(3) レスポンスプロパティサンプル記述網羅率 (ResPropCoverage)

尺度は次の二つの観点から整理した．

・対象：サンプルが Web API のリクエストとレスポンスのいずれを対象としているか

・粒度：サンプルが，Web API のリクエストのサンプルであるか，あるいはリクエストやレスポンスを構成するパラメータプロパティのサンプルであるか

以降，リクエストを Req.，レスポンスを Res.と略記する．

表中の“リクエストサンプル”とは，Web API の動作確認をするために最低限必要な HTTP メソッド名，エンドポイントのパス，全必須パラメータと任意個のオプショナルなパラメータのサンプルの組み合わせである．

これら習得容易性の 3 つの尺度の定義を示す．

(1) リクエストサンプル記述網羅率(ReqSampleCoverage)

API コンシューマによる動作確認が可能な程度に具体化したリクエストのサンプルが提供されている程度を表す．定義を式(1)に示す．値域は 0 から 1．全ての Web API に対してリクエストのサンプルが存在すると 1，全く存在しないと 0 となる．

NumOfAPIs：サービスが提供する Web API の総数

NumOfAPIsWithSample：サービスで提供している全 Web API のうち，そのままで動作確認可能なリクエストサンプルを提供する API の数

$$\mathrm{ReqSampleCoverage} = \frac{NumOfAPIsWithSample}{NumOfAPIs} \tag{1}$$

(2) リクエストパラメータサンプル記述網羅率(ReqParamCoverage)

リクエストに用いるパラメータの仕様に対して，その使い方を理解するためのサンプルが提供されている程度を表す．定義を式(2)に示す．値域は 0 から 1．全パラメータにサンプルが存在すると 1，全く存在しないと 0 となる．

NumOfParams：サービスで提供する全 Web API のリクエストの仕様で定義したパラメータの総数

NumOfParmsWithSample：サービスで提供する全 Web API の全リクエストパラメータの中で，そのサンプルを API ドキュメントに示したパラメータの数

$$\mathrm{ReqParamCoverage} = \frac{NumOfParmsWithSample}{NumOfParams} \tag{2}$$

リクエストのパラメータの仕様とそのサンプルの例を図 7-4 と図 7-5 に示す．また，図中の 2 個の Web API の例を用いてリクエストサンプル記述網羅率とリクエストパラメータサンプル記述網羅率の計算方法を示す．

まず，リクエストサンプル記述網羅率の計算例を示す．Web API は 2 個なので，NumOfAPIs は 2 である．各図の(a)に示した Web API の仕様に対して(b)に示したサンプルがあるので，開発されるアプリケーション毎に異なる部分，図中の<TOKEN>部分，以外はそのまま使用して動作確認が可能なサンプルは 2 個である．従って NumOfAPIsWithSample は 2 である．NumOfAPIs と NumOfAPIsWithSample から，ReqSampleCoverage の値として 1 が得られる．

次に，リクエストパラメータサンプル記述網羅率の計算例を示す．

図 7-4(a)には 1 つのパラメータ buttonID，図 7-5(a)には 4 つのパラメータ buttonID，userID，itemID，amount の仕様が記述されている．2 つの図で合計 5 個のパラメータの仕様が記述されているので，NumOfParams は 5 である．次に，図 7-4(b)にはパラメータ” buttonID”のサンプルが含まれているが，図 7-5(b)にはパラメータ” itemID”と” amount”の 2 つはサンプルに含まれていない．サンプルを持つパラメータは合計 3 個なので NumOfParmsWithSample は 3 である．NumOfParams と NumOfParmsWithSample から ReqParamCoverage の値として 3/5 が得られる．

Name	Type	Description
buttonID	string	Aroma ボタンの識別子

(a) パラメータの仕様の例

```
curl -X POST ¥
  –H 'Authentification: Bearer <TOKEN>' ¥
  -H “Content-Type: application/json” ¥
  -d ‘{ ¥
  “buttonID“ : “b123 ” } ¥
  https://api.aroma.com/aromaAPI/v1/orders
```

(b) リクエストサンプルの例

図 7-4　Web API の定義例 1

Name	Type	Description
buttonID	string	注文に用いる Aroma ボタン ID
userID	string	注文者のユーザ ID
itemID	string	注文する商品の商品 ID
amount	int	注文数量

(a) パラメータの仕様の例

```
curl -X POST ¥
  –H 'Authentification: Bearer <TOKEN>' ¥
  -H “Content-Type: application/json” ¥
  -d ‘{ ¥
  “buttonID“ : “b123 ”, ¥
  “userID”: “userA” } ¥
  https://api.aroma.com/aromaAPI//v1/deliveryItems
```

(b) リクエストサンプルの例

図 7-5　Web API の定義例 2

(3) レスポンスプロパティサンプル記述網羅率(ResPropCoverage)

レスポンスを構成するプロパティの仕様を理解するためのサンプルが提供されている程度を表す．定義を式(3)に示す．値域は 0 から 1．全プロパティにサンプルが存在すると 1，全く存在しないと 0 となる．

NumOfProps：サービスが提供する Web API のレスポンスで，仕様として定義したプロパティの総数

NumOfPropsWithSample：サービスで提供する全 Web API の全レスポンスプロパティの中で，そのサンプルを API ドキュメントで示したレスポンスプロパティの数

$$\mathrm{ResPropCoverage} = \frac{NumOfPropsWithSample}{NumOfProps} \tag{3}$$

ResPropCoverage の計算に必要なデータは，ReqParamCoverage と同様にして API ドキュメントから取得する．

7.2.4 安定性の評価

7.2.4.1 安定性の評価モデル

3.3.4 で述べたように Web API は変更が改版によらず常時行われているため，変更が発生した場合の追従に想定外の工数が発生する可能性がある．そのため，Web API を利用する前に保守の期間や工数を予測できることが重要である．このための品質特性として Web API の安定性(Stability)を定義する．従来のシステム API の安定性は，Raemaekers らにより“ソフトウェアライブラリのユーザが対応を要するかもしれない，時間の経過に伴い積み重なっていくインタフェースや実装の変更の度合い”として定義され，削除メソッド数，既存メソッド内の変更割合，新旧メソッドの変更割合比，新メソッドの割合の 4 つの指標が提案されている[71].

それらの指標値の算出には実装上の API の変更を抽出する必要があるが 2.4 節で述べたように Web API のインタフェース定義は実装言語とは独立に REST の原則に従うリソースの表現として定義されるため，従来のシステム API に対する測定方法はそのままでは適用できない．そこで Li ら，Wang らによって提案された Web API の変更のパターン分類[49][85]と同様に，Web API の構成要素をもとに Web API の変更を評価する方法を提案する．そして Web API に関しても，Raemaekers らの安定性の定義と同様に Web API ユーザである API コンシューマが必要となるであろう API 変更への対応の度合いを安定性として定義する．

安定性の品質モデルを図 7-6 に示す．安定性の定義において，API コンシューマに必要となる対応が少ない場合は安定性が高く，多い場合は安定性が低いとする．そこで Web API の変更数を互換性有無の観点で区別

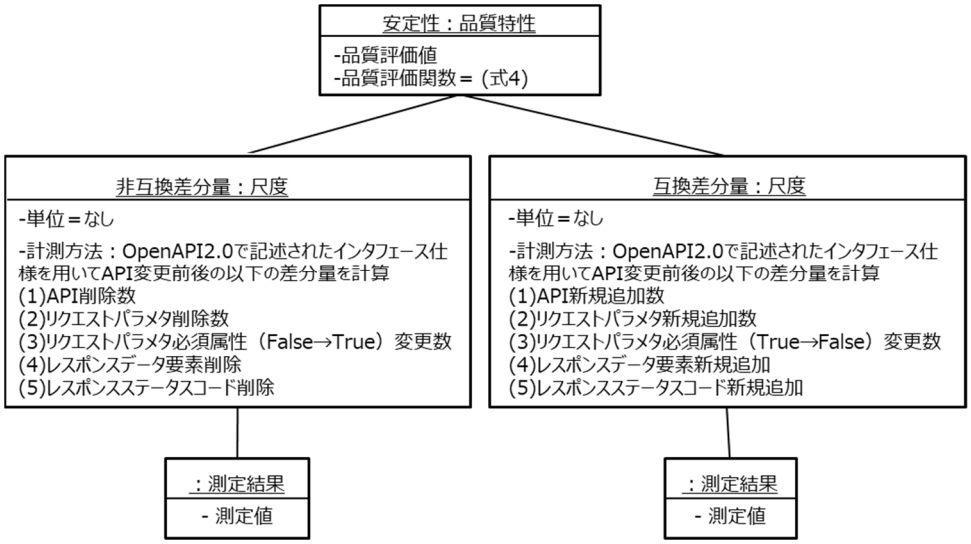

図 7-6　サービス評価品質モデルの一部(安定性)

して安定性の算出に利用する．特に，互換性のない変更は安定性の低下が大きく，互換性のある変更は低下が小さくなるように定義する．互換性のある変更は API コンシューマに影響を与えないと考えることもできるが，インタフェース上は変更がない場合においても振る舞いが異なるために API コンシューマの対応が必要な場合があるため，本定義では安定性の低下をもたらすものとした．

7.2.4.2 安定性の評価尺度と評価方法

Web API の変更量を算出するために，変更前後の 2 つの Web API の差分を抽出する必要がある．Web API は従来のシステム API とは異なり，ネットワーク上に存在する機能をリソースとして表現した文字列によって呼び出す．このため，従来の構文解析による抽出方法の適用は困難であり，Web API のインタフェース定義の差分抽出方法をとることが多い．本研究においてもインタフェース定義を利用して API の差分を抽出する方法を採用する．以降では安定性の品質モデルで取り上げた非互換差分量と互換差分量の算出方法について説明する．

(1) 非互換差分量

Web API に変更が発生した場合に API コンシューマが変更を必要とするような非互換な変更の要素数を非互換差分量と定義し，以下の測定値の総和とする．

a) API 削除数
b) リクエストパラメータ削除数
c) リクエストパラメータ必須属性(False→True)変更数
d) レスポンスデータ要素削除数
e) レスポンスステータスコード削除数

(2) 互換差分量

Web API に変更が発生した場合に API コンシューマが変更を必要としないような互換性のある変更の要素数を互換差分量と定義し，以下の測定値の総和とする．

a) API 新規追加数
b) リクエストパラメータ新規追加数
c) リクエストパラメータ必須属性(True→False)変更数
d) レスポンスデータ要素新規追加数
e) レスポンスステータスコード新規追加数

ソフトウェアで変更が発生した箇所は，最初は不安定な状態だが変更を重ねると安定する傾向をとる性質がある．このような時間的変化の傾向が指標値に利用されることがある[48]．従来のシステム API の安定性についても“時間の経過に応じて変更の影響は小さくなる”と仮定し，変更発生からの時間が重みとして利用されている[71]．この仮定は Web API においても成立すると考え，安定性の品質特性の算出に Web API に変更が発生してからの時間を重み $hw(s)$として導入する．ここで，s はソフトウェアの特定のスナップショットを示す番号であり，1 から始まり過去にさかのぼるにつれて増加する．なお重み $hw(s)$には「時間経過と共に影響が小さくなる」という仮定から，現在からの時間に反比例する 1/t 関数とすることも可能である．

しかし，直近の過去の影響が大きくなり，経過時間によって影響がより小さくなるように，Raemaekers ら[71]の計算式を参考に時間の指数関数の逆数に比例する$\frac{1}{2^s}$を利用する．また，7.2.4.1 で述べたように，API コンシューマへの影響の大きさは(2)互換差分量よりも(1)非互換差分量が大きくなる．このため，安定性の品質評価関数において，(1)と(2)の重みを変えるために非互換差分量係数 k_i，互換差分量係数 k_cを導入する．

以上より，安定性の品質特性の評価関数を式(4)で定義する．値域は 0 から 1 であり，1 に近ければ安定しており，0 に近ければ安定していないことを示す．この評価値により保守工数の推定が可能となる．評価関数で用いる変数を表 7-2 に示す．

$$安定性の品質評価関数 = \frac{1}{1+\sum_{s=1}^{|S|-1}(k_i I(s)+k_c C(s))hw(s)} \quad (4)$$

表 7-2　評価式(4)の変数定義

記号	説明
s	ソフトウェアの特定のスナップショットを示す番号，1 から始まり過去にさかのぼるにつれて増加する
S	すべてのスナップショットの集合
k_i	非互換差分量係数
k_c	互換差分量係数
$I(s)$	非互換差分量(s+1 から s の間に発生した差分量)
$C(s)$	互換差分量(s+1 から s の間に発生した差分量)
$hw(s)$	時間係数 $\frac{1}{2^s}$

7.3　品質モデルと評価方法の適用評価

7.3.1　習得容易性の適用評価

習得容易性を表 7-3 に示す 4 つのサービスの Web API に対して測定した. 測定したサービスの概要や特徴, 尺度の測定方法，および測定結果を示す.

表 7-3　測定対象のサービス

サービス名(版)	概要や特徴
Uber (v1.2)	配車サービス向けのサービス
WordPress(V2)	Web ページやブログの作成を支援するサービス
OpenStack (Version Pike)	オープンソースで開発されているクラウドの IaaS 環境を構築するためのソフトウェアで，多くのコンポーネントから構成されている．本研究では，Ceilometer, Cinder, Designate, Glance, Heat, Keystone, Neutron, Nova, Swift のコンポーネントを利用した．外部に API を公開しているだけでなく，API 開発者も API を利用していることが特徴である
メディア処理 API(v1)	社内向けに公開されている画像や音声等の各種メディアを処理する Web API 群である．API ドキュメントは，Swagger UI で公開されている[81]．本研究の著者らは，本 API 群の API プロバイダに対し，API コンシューマの立場で良いドキュメントとは何かについて指導を行っていた

各サービスに対し，以下の尺度を用いて測定した.

(1) “リクエストサンプル記述網羅率”

Web 上に公開されている API ドキュメントを参照しながら人手で測定した．OpenAPI Specification (OAS) 2.0 形式[60]のファイルを入力とし，API ドキュメントを生成するツール[81]が公開されている．このツールを利用して API ドキュメントを公開する場合，ツール機能を用いてリクエストサンプルを生成できる．このようなツールで API ドキュメントを公開し，リクエストサンプルが生成される場合は，“サンプルを有する”と判断する. メディア処理 API の API ドキュメントは, Swagger UI で公開しているため,

本研究で提案した方法を用いて測定を行った.

(2) “リクエストパラメータサンプル記述網羅率”
Web 上に公開されている API ドキュメントを参照しながら人手で測定した.

(3) “レスポンスプロパティサンプル記述網羅率”
OAS2.0 形式のファイルに記載されたレスポンスプロパティの仕様とそのサンプルを取得して測定した．レスポンスは構造が複雑であることが多く人手では測定困難なためツールを作成した．ツールでの測定手順を図 7-7 に示す．なお，API プロバイダから OAS2.0 形式のファイルが公開されていない場合は，ファイルを作成した上で測定した.

Web API サンプル充実性の品質評価関数には各尺度の重要度を示す係数が含まれる．本研究では，一対比較法を行い各尺度の重要度を決定し，その値を品質評価関数の係数として用いることにした[56].

測定にあたり，前述のサービスが提供する Web API の一部を対象とした．測定対象の Web API 数を表 7-4 に示す.

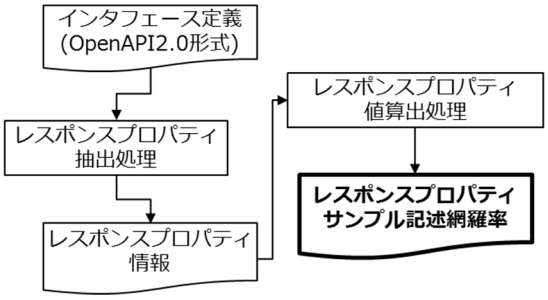

図 7-7 レスポンスプロパティサンプル記述網羅率の測定手順

表 7-4 測定対象の Web API 数

サービス名	尺度(1)(2)の Web API 測定数	尺度(3)の Web API 測定数
Uber	10	4
WordPress	10	10
OpenStack	11	993
メディア処理 API	10	34

各尺度の係数，尺度の測定結果，および品質特性の品質評価値を表 7-5 に示す.

表 7-5 測定結果

	尺度			品質特性
	(1)	(2)	(3)	Web API サンプル充実性
係数 / サービス名	0.77	0.16	0.06	−
Uber	1.00	0.82	0.91	0.96
WordPress	0.90	0.02	0.00	0.70
OpenStack	0.00	0.37	0.95	0.12
メディア処理 API	1.00	0.87	0.25	0.92

7.3.2　安定性の適用評価

OpenStack の中心的なサービス群から Nova (Compute Service)，Cinder (Block Storage)，Glance (Image Service)，Trove (Database Service), Ironic (Bare Metal Provisioning Service), Sahara (Big Data Processing Framework Provisioning)，Manila (Shared Filesystems)を選択し，安定性を測定した．測定対象として OpenStack を採用した理由は以下である．

(1) Web API の時系列変化が確認できるデータが取得可能

(2) Web API のインタフェース定義が機械的に処理可能なフォーマットで管理されている

OpenStack では Web API のインタフェース定義は各サービスのソースリポジトリに格納されている．またフォーマットは Python の reStructuredText を用いており，Web API の仕様を OpenStack 独自のルールに従って記載している[66]．測定対象期間は本フォーマットでのインタフェース定義の管理が始まった 2016 年夏頃(サービスによって若干異なる)とする．

図 7-8 に測定手順を示す．まず，サービスのリポジトリから各月のスナップショットとなるソースコードを取り出す．次に Web API のインタフェース定義である reStructuredText (.inc)ファイルを利用し，Web API の仕様記述として普及している OAS2.0 の形式に変換する．OAS2.0 に変換する理由は，Web API 差分を抽出するツールなどの周辺ツールが揃っており，将来様々な Web API が OAS2.0 形式のインタフェースを公開した際に本方法が適用できるようにするためである．ただし，サービスによっては reStructuredText (.inc)ファイルが不完全なものが存在しており，OAS2.0 形式へ変換できないものも見られた．今回の評価では，1 サービスあたり 20～60 の reStructuredText (.inc)ファイルを数ヶ月にまたがって処理する必要があり，一つ一つ完全なファイルを手作業で作成することが困難であったことから，そのような不完全なファイルは調査対象外として除外した．除外したファイルは全体の 7.2%であり，適用評価全体への影響はない．

そして，安定性の尺度である，非互換差分量，互換差分量を算出するために各月に発生する Web API 差分を抽出する．差分抽出には OSS である swagger-diff[82]を参考に，本品質特性計算用にツールを開発し，適用した．

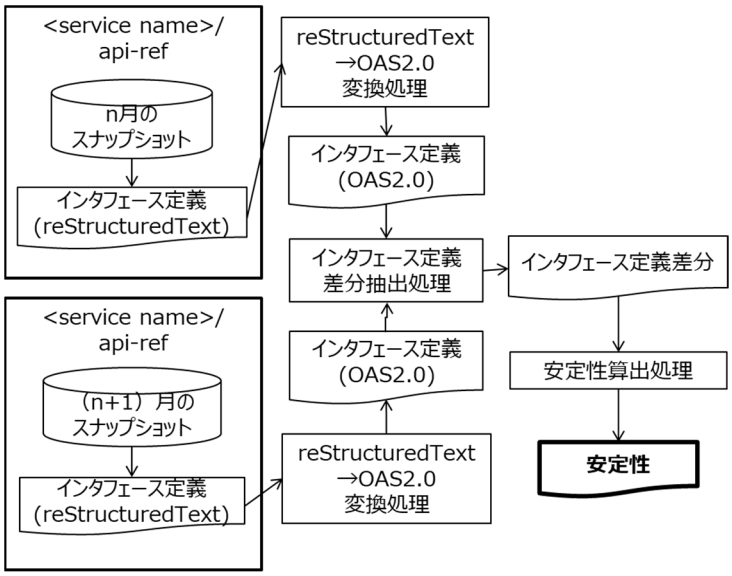

図 7-8　安定性の測定手順

表 7-6 に最新月における重み付き非互換差分量，重み付き互換差分量，安定性を示す．差分量は，式(4)に示す，時間と Web API コンシューマへの影響の重みを考慮に入れた差分量である．7.2.4.1 で述べたように非互換の変更は API コンシューマに大きな影響を与えるため安定性への係数に反映する必要がある．そこで，一対

比較法[56]で用いられる 9 段階尺度(等しく重要な場合は 1 点，とても重要な場合は 9 点をつける方法)を参考に，互換差分量を 1 と捉えた場合に非互換差分量を 9(とても重要)と捉え，互換差分量係数:非互換差分量係数=1:9 となるように互換差分量係数 k_c=1/10，非互換差分量係数 k_r=9/10 とした.

測定結果より Glance，Trove，Sahara は重み付き非互換差分量，互換差分量ともに小さく安定性が高いことがわかった．逆に Cinder は重み付き非互換差分量が多く安定性が低いことがわかった．これらと比較すると， Nova，Ironic，Manila は安定性が中程度であることがわかる.

表 7-6　安定性の測定結果

サービス名	重み付き 非互換差分量	重み付き 互換差分量	安定性
Nova	1.18	0.13	0.43
Cinder	11.35	2.76	0.07
Glance	0.00	0.03	0.97
Trove	0.00	0.00	1.00
Ironic	0.22	0.26	0.67
Sahara	0.01	0.00	0.99
Manila	0.37	0.15	0.66

7.4 提案する品質モデルの考察

7.4.1 RQ1: システム API と異なる特徴を捉えるための Web API の品質特性と測定方法は何か?

Web API を“HTTP プロトコルを利用してネットワーク越しに Web サーバをアクセスする API”と定義した上で，システム API と異なる Web API の特徴として“実装言語と独立したインタフェース定義”と“ユーザと独立した進化”を指摘した．Web API は Java などのプログラミング言語を用いて Web サーバ上で実装される．それをアクセスするためのインタフェース定義は REST に代表されるリソース表現であるため，システム API のような実装言語による型チェック等の厳格なチェックができない．また，リモートでアクセスするため，アクセス元のアプリケーションプログラムが動いていようと，独立して修正や削除が行われる．アプリケーション開発においてこれらの特徴は開発工数を増やす原因となる．そのため，アプリケーション開発に関わるステークホルダのうち，API コンシューマによるアプリケーション開発のしやすさに着目した．開発初期で実際に Web API を使い始める前にリスクを検知する目的で，非機能要求フレームワークに基づき，“API コンシューマが開発しやすい”をソフトゴールと捉えてゴール分析を実施した．実際に Web API をアクセスする前なので，API ドキュメントを対象に測定可能であることを重視して，ISO 25010 の品質特性をベースに，ユーザビリティに関して習得容易性を特定し，また，保守性の新しい副特性として安定性を提案した．

習得容易性は，API ユーザビリティの一特性である．Web API の“インタフェース定義”が REST の原則に従うリソース表現として XML や JSON 等で定義され，ソフトウェアの仕様記述と類似の性質を有することを品質モデルに反映した．

安定性は，システム API と異なり実行時に API ユーザとは独立に進化する Web API の特徴に着目して，Web API の構成要素の変更量に基づいて品質モデルを定義した．

なお，今後も継続する品質特性の議論のために品質モデルのメタモデルを最初に定義した．この考え方は，GQM[2]による品質特性から尺度を導き出すアプローチの一実現手段ともいえる．本研究で提案したメタモデルを，ビジネスアプリケーションの UX 評価方法の開発に適用し，メタモデルに従って品質特性とその尺度，測定方法を定義した[60]．実プロジェクトで，定義した尺度と測定方法が実測可能であることを実証した[60]．これにより，メタモデルとして妥当であることも確認した．

習得容易性に関して 3 つの尺度を定義し，その中で達成度合いを機械的な評価に適していると考えられる Web API サンプル充実性の尺度を定義し，実測可能であることを示した．安定性では，互換性に着目した尺度と測定方法を定義し，実測可能であることを確認した．

7.4.2 RQ2: 提案方法は実際の Web API に有効か?

7.4.2.1 習得容易性

品質評価関数の有効性を確認するため，API コンシューマ 7 名を被験者としてサービスの習得容易性を評価する実証実験を実施した．全ての被験者は，評価対象とした 4 サービスの Web API を用いたアプリケーション開発の経験はない．しかし，全被験者は一般的な API ドキュメントに記述される内容やその意味の知識を有している．また一部の被験者は API ドキュメントを参照しながら Web API を用いたアプリケーション開発の経験がある．

被験者は各 API ドキュメントを最大 20 分間参照した上で，開発の容易性を 3 段階で評価した．3 段階で評価した理由は，段階数が多いと被験者が段階を選ぶ際に迷いが生じ，その結果選ぶ段階に誤差が生じると考えたからである．表 7-3 に示したサービスに対し，習得が容易と見なしたものを 1，困難を 0，中間を 0.5 でスコアリングを行い，回答結果の平均値を算出した．それらの結果を表 7-7 に示す．表中の<番号>は品質評価値の順位を示す．本稿では“習得容易性”のうち“Web API サンプル充実性”に着目した．以後の議論では“Web API サンプル充実性”の品質評価値を“習得容易性”の品質評価値とみなす．

表 7-7　習得容易性の評価結果の比較

サービス名	習得容易性 (Web API サンプル充実性)の評価結果	実証実験での評価結果
Uber	0.96 <1>	0.9 <1>
WordPress	0.70 <3>	0.3 <3>
OpenStack	0.12 <4>	0.2 <4>
メディア処理 API	0.92 <2>	0.8 <2>

7.3.1 で算出した習得容易性の品質評価値と実証実験での評価値の順位が一致した. 従って, 尺度, 測定方法, 品質評価関数は妥当であると判断できる．また，習得容易性の尺度の値を求める際に API ドキュメントのみを用いたので，開発初期に適用可能となる.

各サービスの品質評価値について考察する.

Uber は全リクエストにサンプルがあり，必須のパラメータやプロパティ以外は 1 個以上のサンプルが記載されていることが高評価につながった.

OpenStack の品質評価値が低いのは，リクエストのパラメータが階層構造を持っている Web API だけにしかサンプルが記載されていないからある．OpenStack の API コンシューマは OpenStack の API プロバイダでもあり，現状は問題視されていないと推察している．しかし，一般の API コンシューマにとっては習得が困難であると判断した.

WordPress はリクエストのサンプルはあるものの, オプショナルなパラメータに全くサンプルを提示しない. さらにレスポンスのプロパティはサンプルのみならず定義も全く記載しないため品質評価値が下がる.

メディア処理 API は，API ドキュメントとしての記述項目の充実を図るための指導を受けていた．その指導によりリクエスト側の記述が非常に充実し，高評価につながった．一方，レスポンス側のサンプルが不十分であることが明確になったので，API プロバイダに対して改善に向けた提言を行うことができた.

この結果を Uddin らの実証結果[84]と比較する．Uddin らは API ドキュメントの問題を 10 個挙げ，それらを発生頻度，影響の大きさで分類している．その中で深刻さが高い問題として不完全さ(Incompleteness)，曖昧さ(Ambiguity)，説明不足(Unexplained examples)，不正確さ(Incorrectness)の 4 つを挙げている．これらの中で，曖昧さと説明不足の問題は，ドキュメント中のサンプルによって軽減できることが実証されている．これら二つの問題を，本稿で示した習得容易性によって定量的に評価できるようになる．なお，不完全さは他の Web API とのインタラクションの説明不足の問題，不正確さは API ドキュメントと実際の動作に差異があることによる問題である．不完全さは複数の API のドキュメントの横断的な分析が必要であり，不正確さの評価は Web API の実際の動作を確認する必要があるため，本稿で示した習得容易性では定量化できない.

7.4.2.2　安定性

安定性に関して，Web API の実際の変更内容の詳細解析による確認，及び，Web API プロバイダ（提供者）が公表していた成熟度との比較，の 2 通りの方法で有効性を検証する.

(1) Web API の実際の変更内容の詳細解析による確認

7.2.4.1 で述べた通り“Web API を利用する前に保守の期間や工数を予測できるか”との観点で算出した安定性について考察する.

図 7-9 は安定性を算出した 7 サービスについて，各月における安定性をプロットし，時系列変化を示している．7.3.2 では，安定性を高い，低い，中程度の 3 種類が観察できた．時系列変化も，安定性が高くかつ変動量が少なく推移しているもの，低くかつ変動量が少なく推移しているもの，安定性が変動しているもの，の 3 種類が確認できる．そこで，それぞれから代表的な 1 サービスをサンプルとし，実際に発生している Web API の変更量を確認する.

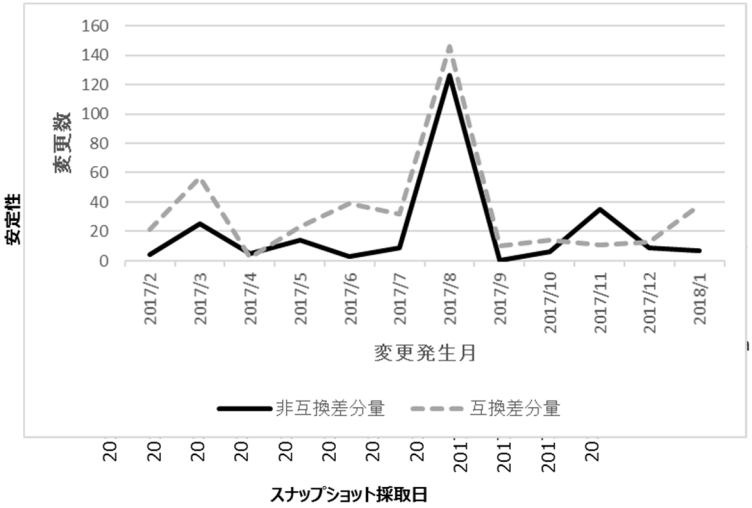

図 7-9　安定性の月ごとの推移

(a) 安定性が高くかつ変動量が少なく推移しているもの：Glance

図 7-10 は Glance の非互換差分量と互換差分量の推移である．非互換差分は 2017 年 3 月に 1 件発生しているのみである．また互換差分は 2017 年 3 月に 1 件，7 月に 4 件，2018 年 1 月に 1 件と数ヶ月に一度程度でしか発生していない．Web API コンシューマは，非互換変更によるインタフェース変更への対応はほとんど必要なく，また，互換変更による品質影響の確認もほとんど必要ない状態であるといえる．従って，安定性の高さが Web API コンシューマの保守工数の低さを十分に表しているといえる．

(b) 安定性が低くかつ変動量が少なく推移しているもの：Cinder

図 7-11 は Cinder の非互換差分量と互換差分量の推移である．非互換差分量は互換差分量よりも少ないものの毎月 10 件前後は発生しており，API コンシューマが常にインタフェース変更へ注意する必要があることが読み取れる．また，互換差分量も常に 20 件前後は発生している状態であるため，インタフェース変更に現れない内部品質にも注意が必要な状態であると推測できる．

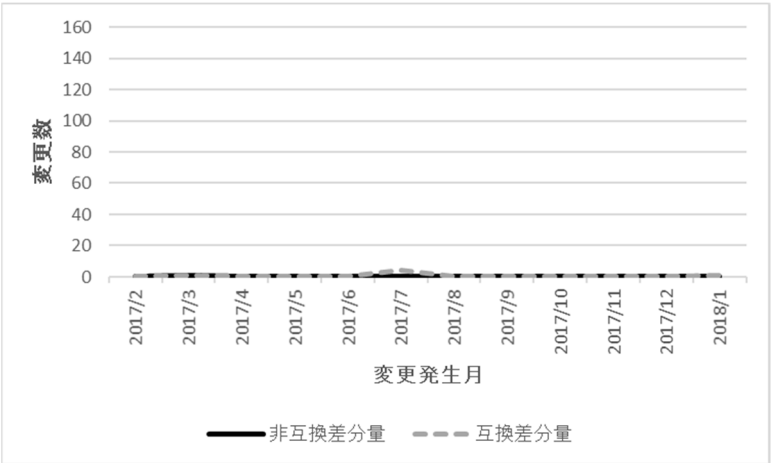

図 7-10　Glance の変更量の時系列変化

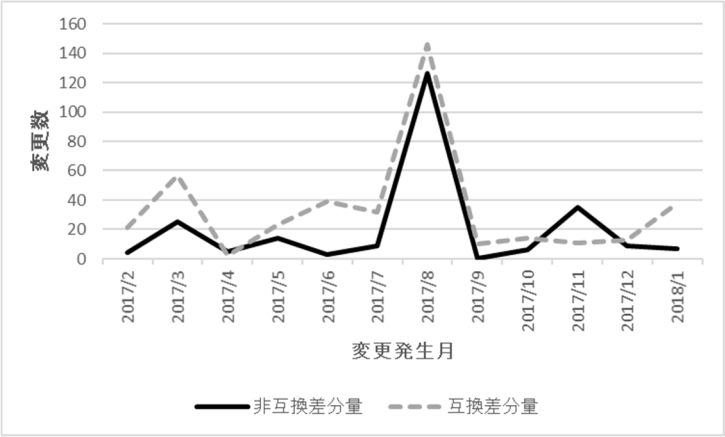

図 7-11　Cinder の変更量の時系列変化

(c) 安定性が変動しているもの：Manila

図 7-12 は Manila の非互換差分量と互換差分量の推移である．非互換変更に着目すると，2017 年 2 月から 7 月までは 0 件，また 11 月から 2018 年 1 月まででは 1 件しか発生していない．しかし，2017 年 8 月～10 月は合計で 16 件も非互換変更が発生している．互換変更については非互換変更と完全に重なっているわけではないが，非互換変更と同様に変更が発生している期間と発生していない期間がある．

このように時期によって変更量が大きく異なる場合は，その該当時期においては Web API コンシューマの対応工数が多くなることが推測できる．

図 7-13 は図 7-9 から Manila の安定性のみを抽出したものである．前述の通り対応工数が多くなることが推測できる 2017 年 8 月～10 月には安定性が大きく低下しており，安定性が対応工数の増減を表していることが読み取れる．

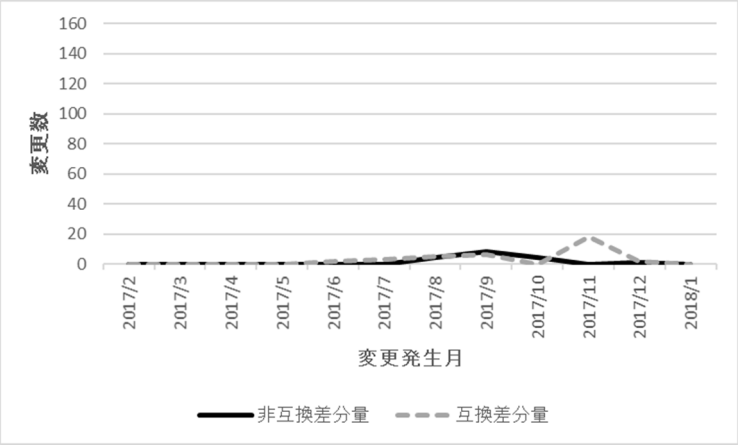

図 7-12　Manila の変更量の時系列変化

上記の議論から，品質特性として定義した安定性により Web API コンシューマの保守に必要な対応工数の度合いを示すことができていると言える．この結果から安定性の品質モデルは，Web API の変更の評価尺度として妥当であり，開発初期での API コンシューマにとって有用な情報を提供できるといえる．

また，今回の測定において(c)で見られた「変更が発生する時期」について調査したところ，OpenStack のメジャーリリースの時期と重なっていることがわかった．例えば，2017 年 8 月はメジャーリリース「Pike」の初期リリースが行われた時期となる．メジャーリリースでは大規模な機能追加が行われることが多い．そのような機能追加時はインタフェースの互換性を保つことが難しいため非互換変更が発生し，Web API コンシューマ

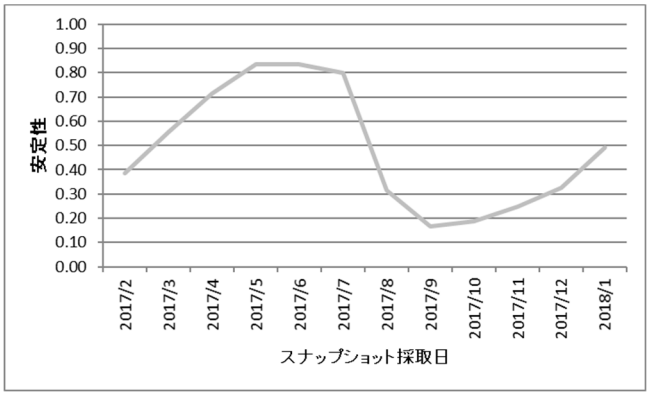

図 7-13　Manila の安定性の推移

の対応が必要となることが多いと推測できる．この事例より，安定性の時系列変化を確認することで安定性の変動の傾向を捉えることができ，そのような兆候が見られる場合は事前に対応工数の上積みを行う等の対策をとることが可能となる．

(2) Web API の成熟度との比較

OpenStack の公式な情報には安定性を示すものはないが，成熟度を示す Maturity が 2018 年 8 月まで公開されていた[67]．これはサービス利用の判断に使う目的で，自動または手動でつけられたタグをベースに 7 段階の値で表現されており，安定性(Stability)と持続可能性(Sustainability)を示している．この特性は本稿で提案した品質特性の安定性と類似しているため，Maturity と安定性の値を比較することで，安定性の妥当性について考察する．

図 7-14 に Nova，Cinder，Glance の Maturity をリリースごとにプロットした．各サービス間の Maturity を比較すると，次の順序関係になっている．

Nova＞Cinder＞Glance　(5)

安定性については表 7-6 に示した通り次の順で高くなっている．

Glance＞Nova＞Cinder　(6)

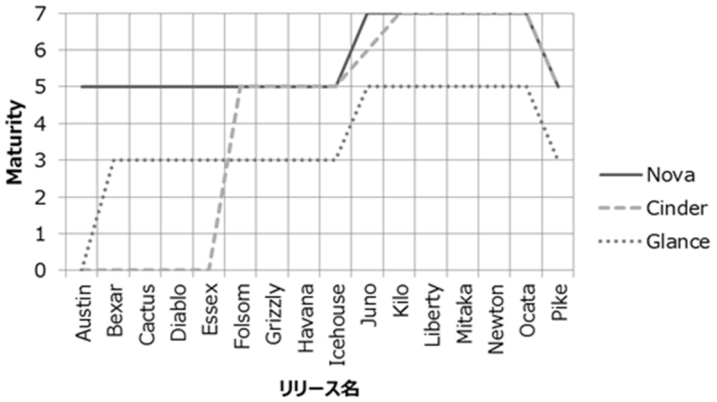

図 7-14　公式サイト提供のリリースごとの Maturity

Nova＞Cinder については同じ傾向を示しているが，Glance は，Maturity は低いが安定性は高いという結果となった．これは Maturity が安定性だけでなく持続可能性も評価しているのに対して，安定性はインタフェース定義の変更量のみを評価しているためと推測される．

また，リリースを重ねるごとに Maturity は上昇していく．そこで，安定性にも同じ傾向があるかを各月における安定性を算出して確認した(図 7-15)．安定性に関しても上昇傾向が見られる．ただし，Nova，Cinder，Glance ともに 2017 年 7 月の周辺で一時的に安定性が低下している．これは 2017 年 8 月 30 日の Pike のリリースと関係があると推測できる．図 7-15 は Pike のリリース後に公式サイトで確認した Maturity であるが Pike で低下していることが確認できた. 他のリリースと比較してリリースからの経過日数が少ないことに起因すると推測されるが，安定性においても同様な傾向が確認できた.

上記の議論から，公式サイトが提供する安定性を示す Maturity と品質特性として定義した安定性は類似の傾向を示すと言える．この結果から安定性の品質モデルは，Web API の変更の評価尺度として妥当であり，開発初期での API コンシューマにとって有用な情報を提供できるといえる.

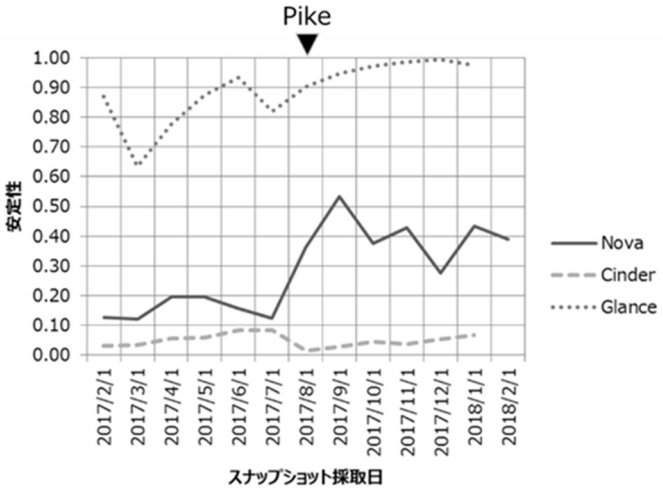

図 7-15　安定性の月ごとの推移

7.4.3　提案した品質モデルの課題

本稿の測定では尺度の計算で利用する重みについて，一対比較法[56]を用いて業務経験に照らし合わせて，重み設定時の条件を満たす値を選択した．今後さまざまなドメインでの測定で同様な考え方で調整して，適切な値を見分ける必要がある．また，本稿で定義した習得容易性と安定性は，アプリケーション開発の初期を想定して API ドキュメントを解析対象とした．Web API のインタフェース定義と実際の動作に齟齬がある実情があり，テストや動作確認を伴う“実装”に関する品質特性の議論が今後必要である.

8 考察

8.1 メタモデル駆動の業務プロセスモデリングの意義

従来の業務プロセスモデリングと業務プロセスマネジメントに関する研究成果が知られている[16][39]．これらのアプローチは業務プロセスモデリングの技術にのみ焦点をあてており，本研究のような業務プロセスモデリングの成果物の再利用は言及していない．また，プロダクトラインソフトウェア工学のような業務プロセスモデルの共通性と可変性の研究例が，実際に共通部分とバリアントをトレードオフすることで効果を確認している[53]．本研究と方向は同じであるが本研究のテンプレートのような具体的なフレームワークは提供されていない．

提案する方法論の独自性はメタモデルとメタモデル駆動モデリングにある．業務プロセスモデリングを実際に機能させるための一連の支援技術を提供したことで，業務プロセスモデリング表記法以外にもさらなる支援を提供することが必要であることを示している．また，提案する BPM テンプレートと BPM-IE(統合環境)による業務プロセス再利用方法は，業務プロセスの共通性と可変性を識別して分離する際の基準，及び，可変部分を分解するための業務プロセスモデルの 3 階層のフレームワークを示した．これは業務プロセスモデルを再利用するための基礎となる．併せて，BPM テンプレートが再利用のための業務プロセスコンポーネントとして機能し，業務プロセス再利用のための BPM-IE とともに提供すると実プロジェクトの生産性が 46%向上することを，実証した．

以上の議論から，本研究は，業務プロセスモデリングへ従来になかった新たなアプローチを提示し，その有効性を実証している点で業務プロセスモデリングの発展に貢献する成果であると言える．

8.2 企業情報システム開発におけるパターン体系の意義

新しいソフトウェア開発技術に関する知識やノウハウの集約と発信は新しいソフトウェア開発技術全般が必要としている課題である．本研究ではこの課題に対する一解決アプローチとして，新しいソフトウェア開発技術を用いたシステム開発のノウハウをパターン化し体系的かつ相互に関連づけた形態，すなわち新技術を利用するシステム開発のパターン体系を提案した．

従来のパターンカタログは，デザインパターンから始まり[25]，様々な領域を対象にノウハウを広く普及させる文献として提供されている．Buschmann ら[7]はパターン間の多くの依存関係を指摘しているが，それを実際に実現したパターン体系の開発例を見つけることは難しい．本研究では新しいソフトウェア開発技術の発信の試みとして，Web サービス開発向けの具体的な技術である XML を取り上げ，XML を利用するシステムの開発ノウハウのパターン体系を開発し，実際に適用して一定の効果を得た．

本研究は，今後現れる新たな技術要素に対処する際にも 5 つの入口（アーキテクチャ，テンプレートモデル，詳細ノウハウ，コンポーネント，開発方法論）から成るパターン体系が有効であることを示した．パターンの概念を Web サービスの再利用へ発展させ，それを通した企業内における新技術の普及にも貢献した．

以上の議論から，本研究はパターン技術の新たな枠組みを提案し，その応用を拡張した点でソフトウェア工学における基礎技術であるパターン技術の発展に貢献する成果であると言える．

8.3 Web API の品質モデルの意義

クラウドの普及に伴い様々なサービスを Web API として提供し，アプリケーション開発に活用するようになっている．本研究では企業システム開発における Web API の利用を想定して，従来のソフトウェアシステムの製品品質モデルを拡張して，Web API の特性を捉えた品質モデルを提案した．Web API を利用したアプリケーション開発に関わるステークホルダを整理した上で，API コンシューマの観点から重要な品質特性として，API ユーザビリティの習得容易性と変更の安定性の概念に着目し，習得容易性と安定性の品質モデル，尺度，測定方法を提案した．提案した品質特性の測定方法を実際の Web API に適用して，測定可能であること，

及び，人による評価や保守に必要な対応工数の度合いを示すことを示した．このことから，提案した品質モデルはアプリケーション開発の初期段階で Web API の品質評価に有効であることを確認した．

このように，本研究は，Web API の利用において新たに重要性が認識されている品質特性のモデル化とその定量的評価方法の確立，ならびに，従来のソフトウェア製品品質モデルを Web API を利用した企業情報システムの品質モデルへ拡張する一つのアプローチを提示し，品質モデルの発展に貢献した点で意義がある．

9 今後の課題

本研究を発展させていくために，次の 2 点を課題としてあげる

9.1 Web API 品質モデルの拡充

Web API はシステム内の API とは基本的に異なる特性を持つことから，従来の品質モデルは不適合である．本研究で提案した Web API の品質モデルは，Web API を利用するアプリケーション開発の初期を想定して，Web API の特性を捉えて開発リスクを早期に把握することで他者が開発した Web API の採用可否の判断につなげることを目標とした．

今後，Web API を利用したアプリケーション開発における開発プロセスにおける品質や，サービス提供の継続性などの SLA（Service Level Agreement）に関わるアプリケーションの運用時の品質へも取り組む必要がある．その際，API マネジメントシステムで計測可能な尺度を活用することも検討すべきである．

また，品質モデルの拡充のもう一つの議論点として，これまでのシステム内の API の品質特性との差異の整理があると考える．システム内の API はソフトウェアの製品品質[34][35][36]を前提に議論されているが，Web API の品質モデルとしても適用可能な特性と尺度，新たに提案すべき特性と尺度を明らかにして，それらを整理することが必要である．

Web API の品質モデル体系が確立されれば，品質の安定した Web API を選択することで，サービス指向システムの品質確保の基礎技術として貢献することが期待できる．

9.2 実行可能なパターン体系の検討

クラウドの登場により，様々なソフトウェア実行環境を早期に試行できる状況となった．開発ノウハウは，本研究が対象としたようなソフトウェア成果物，方法論をパターンとして提供するだけではなく，CI（Continuous Integratioin）/ CD（Continuous Delivery）環境を実現するスクリプト群も含めて提供可能となっている．

ソフトウェアに関連する様々なノウハウをパターンカタログとして提供することは無論継続すべきではある．しかし，頻繁に進化するソフトウェアを積極的に利用する開発においては，その開発ノウハウは全てをソフトウェアとして提供して，入手後すぐに試行できる環境とともに提供することが今後，必要になると考える．ソフトウェアや実行環境の変化に追随し続けることは工数と時間を要する．そのため，利用するソフトウェアや実行環境の変化を自動的に解析して，ソフトウェア化したノウハウに反映する技術の開発が期待される．変化を反映することにより，パターンの実践を品質面から保証し続けることが可能となる．

10まとめ

Web サービスは企業情報システム開発の重要な技術となっている．本研究は，Web サービスを提供・利用するサービス指向システム開発の生産性向上を目的に，生産性を阻害する課題を明らかにし，その解決を図る開発方法論を提案した．提案する方法論の核技術として，メタモデル駆動の業務プロセスモデリング，Web サービス開発向けパターン体系，Web API の利用者観点での品質モデルを提案した．

メタモデル駆動業務プロセスモデリングは統合開発環境と共に，サービス指向要求分析でこれまで困難であった業務プロセスの再利用を可能とし，実プロジェクトの生産性が 46%向上することを実証した．

Web サービス開発向けパターン体系はパターンの概念を Web サービスの再利用へ発展させ，それを通した企業内における新技術の普及にも貢献した．

さらに，Web サービスのインタフェースである Web API によるシステムの実現を効率化するためには，インタフェース定義と実装の乖離等の新たな問題がある．本研究は Web API を利用するアプリケーション開発者の観点からそれらの問題を捉え，ユーザビリティの概念からその品質特性を分析して，習得容易性と安定性の品質モデルを提案した．提案した品質モデルを実際の Web API に適用し，有効性を確認した．

本研究はソフトウェア工学のアプローチにより，サービス指向システム開発の生産性向上，ならびに，その開発技術の発展に貢献するものである．あわせて，Web が基盤となっている現在のソフトウェア開発におけるソフトウェア工学としてサービス指向システム開発技術の体系化に貢献するものと考えている．

11 謝辞

本研究は，南山大学大学院理工学研究科ソフトウェア工学専攻青山研究室において，青山幹雄教授のご指導の下に実施されたものです．本研究をまとめる全過程において親身のご指導と適切なご助言を賜りました．青山幹雄教授に深く感謝し，心より御礼申し上げます．

本論文を審査頂きました，南山大学理工学研究科ソフトウェア工学専攻，野呂昌満教授，沢田篤史教授，ならびに，東京工業大学情報理工学院情報工学科 佐伯元司教授におかれましては，本論文について詳細なアドバイスを頂きました．深く感謝するとともに厚く御礼申し上げます．

本論文は著者が富士通研究所と富士通株式会社において行った研究に基づいています．本研究を進めるにあたってご協力を賜りました，富士通研究所 山本晃司氏，大橋恭子氏，福寄雅洋氏，関口敦二氏，上原忠弘氏，松尾昭彦氏，富士通株式会社 猪股順二氏，松田友隆氏をはじめとするソフトウェア工学の研究開発部門の皆様，さらに，富士通株式会社の社内展開にご協力頂きました関連部署，技術評価にご協力頂きました顧客プロジェクト部隊の皆様に心から感謝いたします．

参考文献

[1] Aagesen, G. and Krogstie, J., BPMN 2.0 for Modeling Business Processes, vom Brocke, J. and Rosemann, M. (eds.) Handbook on Business Process Management 1, 2nd ed., Springer, 2015, pp. 219-250, https://doi.org/10.1007/978-3-642-45100-3_10

[2] Basili, V. R. and Rombach, H. D., The TAME Project: Towards Improvement-Oriented Software Environments, IEEE Tran. on Software Engineering, Vol. 14, No. 6, Jun. 1988, pp. 758-773.

[3] Basole, R. C., Accelerating Digital Transformation: Visual Insights from the API Ecosystem, IEEE IT Professional, Vol. 18, No. 6, Nov.-Dec. 2016, pp. 20-25.

[4] Bennet, K. H., and Rajlich, V., Software Maintenance and Evolution: A Roadmap, The Future of Software Engineering, ICSE 2000, ACM, May 2000, pp. 75-87.

[5] Biehl, M., RESTful API Design, API-University Press, 2016

[6] Bug Prediction at Google, http://google-engtools.blogspot.jp/2011/ 12/bug-prediction-at-google.html.

[7] Buschmann, F., Meunier, R., Rohnert, H., and Stal, M., Pattern-Oriented Software Architecture: A System of Patterns, Wiley, 1996

[8] Ceatano, A., Silva, A. R., and Tribolet, J., Business Process Decomposition: An Approach Based on the Separation of Concerns, *International Journal of Conceptual Modeling*, Vol. **5, No.** 1, 2010, pp. 44-57, https://doi.org/10.18417/emisa.5.1.3.

[9] Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J., The NFR Framework in Action, Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J.(eds), Non-Functional Requirements in Software Engineering, Springer, 1999, pp. 15-45.

[10] Ciraci, S., van den Broek, P., Evolvability as a Quality Attribute of Software Architectures, Proc. of Int'l ERCIM Workshop on Software Evolution 2006, Apr. 2006, pp. 29-31.

[11] Côté, M.-A., Suryn, W. and Georgiadou, E., In Search for a Widely Applicable and Accepted Software Quality Model for Software Quality Engineering, Software Quality J., Vol. 15, No. 4, Dec. 2007, pp. 401-416.

[12] Daigneau, R., Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services, Addison Wesley, 2011

[13] Davis, R., and Brabander, E., ARIS Design Platform, Springer, 2007.

[14] De, B., API Management: An Architect's Guide to Developing and Managing APIs for Your Organization, APRESS, 2017.

[15] Dumas, M., La Rosa, M., Mendling, J., and Reijers, H. A., Fundamentals of Business Process Management, 2nd ed., Springer, 2018.

[16] Eriksson, H-E. and Penker, M., Business Modeling with UML, John Wiley & Sons, 1999.

[17] Erl, T., SOA Design Patterns, Prentice Hall, 2008.

[18] Erl, T., Carlyle, B., Pautasso, C. and Balusubramanian, R., SOA with REST, Prentice Hall, 2012.

[19] Espinha, T., Zaidman, A., and Grosset, H., Web API Growing Pains: Loosely Coupled Yet Strongly Tied, J. of Systems and Software, Vol. 100, Feb. 2015, pp. 27-43.

[20] Fallside, D. C.: XML Schema Part 0: Primer, http://www.w3.org/TR/xmlschema-0/, 2001.

[21] Fielding, R. T., Taylor, R. N., Erenkrantz, J. R., Gorlick, M. M., Whitehead, J., and Khare, R., Reflections on the REST Architectural Style and "Principled Design of the Modern Web Architecture", Proc. of ESEC/FSE 2017, ACM, Sep. 2017, pp. 4-14.

[22] Fokaefs, M., Mikhaiel, R., Tsantalis, N., Stroulia, E. and Lau, A., An Empirical Study on Web Service Evolution, Proc. of ICWS 2011, IEEE, Jul. 2011, pp. 49-56.

[23] Fowler, M., Patterns of Enterprise Application Architecture, Addison-Wesley, 2002.

[24] Fowler, S. J., Production-Ready Microservices, O'Relly, 2017

[25] Gamma, E., Helm, R., Johnson, R. and Vlissides J., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[26] Havey, M., Essential Business Process Modeling, O'Reilly, 2005.

[27] Henkel, J. and Diwan, A., CtachUp! Capturing and Replaying Refactorings to Support API Evolution, Proc. of ICSE 2005, May 2005, IEEE, pp. 274-283.

[28] Hors, A. Le, Hegaret, P. Le, Wood L., Nicol, G., Robie, J., Champion, M. and Byrne, S.: Document Object Model (DOM) Level 2 Core Specification, http://www.w3.org/TR/DOM-Level-2-Core/, 2000.

[29] IBM, Rational Software Architect, https://www.ibm.com/developerworks/downloads/r/architect/index.html.

[30] 井上 健, 桐嶋 正光, 林 康二, 企業内オブジェクト指向相談室の設立と運用経験, オブジェクト指向最前線 2002, オブジェクト指向シンポジウム 2002 論文集, 情報処理学会/近代科学社, Sep. 2002, pp.27-34.

[31] ISO/IEC 15939:2017, Systems and Software Engineering - Measurement Process, 2017

[32] ISO/IEC 19510:2013, Information Technology – Object Management Group Business Process Model and Notation, 2013.
[33] ISO/IEC 19505-2:2012, Information Technology – Object Management Group Unified Modeling Language (OMG UML) – Part 2: Superstructure, 2012.
[34] ISO/IEC 25010:2011, Systems and software engineering —Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models, 2011.
[35] ISO/IEC 25020:2007, Software Engineering- Software Product Quality Requirements and Evaluation (SQuaRE) – Measurement Reference Model and Guide, 2007.
[36] ISO/IEC 25030:2007, Software Engineering- Software Product Quality Requirements and Evaluation (SQuaRE) - Quality Requirements, 2007.
[37] ISO/IEC/IEEE 29148:2011, Software and Systems Engineering- Life Cycle Process - Requirements Engineering, 2011.
[38] Iyer, B. and Subramaniam, M., The Strategic Value of APIs, Harvard Business Review, Jan. 2015, https://hbr.org/2015/01/the-strategic-value-of-apis.
[39] Jacobson I., Booch, G., and Rumbaugh, J., The Unified Software Development Process, Addison-Wesley, 1999.
[40] JDOM, http://www.jdom.org/.
[41] Jensen, S., and van Capelleveen, G., Quality Review and Approval Methods for Extension in Software Ecosystems, Jensen. S., Brinkkemper, S., and Cusumano, M. A. (eds.), Software Ecosystems, Edward Elgar, 2013, pp. 187-211.
[42] Keller, R. , Tessier, J. and Bochmann G. Von: A pattern system for network management interfaces, CACM 41 (9), pp. 86-93, 1998
[43] Kitchenham, B., and Pfleeger, S. K., Software Quality: The Elusive Target, IEEE Software, Vol. 13, No. 1, Jan. 2006, pp. 12-21.
[44] Koschmider, A., Caporale, T., Fillmann, M., Lehner, J., and Oberweis, A., Business Process Modeling Support by Depictive and Descriptive Diagrams, *Proceedings of 6th International Workshop on Enterprise Modeling and Information Systems Architecture (EMISA 2015)*, Sep. 2015, pp. 31-44, https://dl.gi.de/handle/20.500.12116/2035.
[45] Koschmider, A., Fellmann, M., Schoknecht, A., and Oberweis, A., Analysis of Process Model Reuse: Where are We Now and, Where Should We Go from Here?, *Decision Support Systems*, Vol. 66, 2014, pp. 9-19
[46] Lederer, M., Knapp, J., and Schott, P., The Digital Future Has Many Names: How Business Process Management Drives the Digital Transformation, *Proceedings of the 6th International Conference on Industry Technology and Management (ICITM 2017)*, IEEE, Mar. 2017, pp. 22-26. https://doi.org/10.1109/ICITM.2017.791788
[47] Leopold, H., Mendling, J., and Günther, O., Learning from Quality Issues of BPMN Models from Industry, *IEEE Software*, Vol. 33, No. 4, Jul./Aug. 2016, pp. 26-33, http://doi.ieeecomputersociety.org/10.1109/MS.2015.8
[48] Lewis, C. and Ou, R., Bug Prediction at Google, http://google-engtools.blogspot.jp/2011/ 12/bug-prediction-at-google.html.
[49] Li, J., Xiong, Y., Liu, X. and Zhang, L., How Does Web Service API Evolution Affect Clients?, Proc. of ICWS 2013, IEEE, Jun.-Jul. 2013, pp. 300-307.
[50] Marshall C., Enterprise Modeling with UML, Addison-Wesley, 1999.
[51] McLellan, S. G., Roesler, A. W., Tempest, J. T. and SpinuzziLellan, C. I., et al., Building More Usable APIs, IEEE Software, Vol. 15, No. 3, May/Jun. 1998, pp. 78-86.
[52] Menascé, D. A., QoS Issues in Web Services, IEEE Internet, Vol. 6, No. 6, Nov.-Dec. 2002, pp. 72-75.
[53] Milani, F., Dumas, M., Ahmed, N., Matulevičius, R., Modelling Families of Business Process Variants: A Decomposition Driven Method, *Information Systems*, Vol 56, Mar. 2016, pp. 55-72.
[54] Mili, H., Tremblay, G., Jaoude, G. B., Lefebbre, É., Elabed, L., and Boussaidi, G. E., Business Process Modeling Languages: Sorting Through the Alphabet Soup, *ACM Computing Surveys*, Vol. 43, No. 1, Article No. 4, Nov. 2010, pp. 1-56.
[55] 水野 貴明, Web API: The Good Parts, O'Reilly, 2014.
[56] Mu, E. and Pereyra-Rojas, M., Practical Decision Making using Super Decisions v3, Springer, 2017.
[57] Murphy, L., Alliyu, T., Macvean, A., Kery, M. B. and Myers, B. A., Preliminary Analysis of REST API Style Guidelines, Proc. of PLATEAU 2017/SPLASH 2017, ACM, Oct. 2017, pp. 1-9, https://2017.splashcon.org/track/plateau-2017.
[58] Myers, B. A. and Stylos, J., Improving API Usability, CACM, Vol. 59, No. 6, Jun. 2016, pp. 62-69.
[59] Nulab, Cacoo, https://cacoo.com/.
[60] Ohashi, K., Katayama, A., Kurihara, H., Yamamoto, R., Doerr, J. and Magin, D. P., Focusing Requirements Elicitation by Using a UX Measurement Method, Proc. of RE 2018, IEEE, Aug. 2018, pp. 348-357.
[61] OMG, Reusable Asset, Ver. 2.2., 2005, https://www.omg.org/spec/RAS/About-RAS/
[62] OMG, Unified Modeling Language Specification (OMG UML), Ver.2.5.1, 2017, https://www.omg.org/spec/UML/2.5.1/.
[63] OMG, Unified Modeling Language Superstructure, Ver.2.4.1, 2011, https://www.omg.org/spec/UML/2.4.1
[64] OMG, Business Process Model and Notation, Ver.2.0, 2011, https://www.omg.org/spec/BPMN/2.0/.
[65] Open API Specification, Version 2.0, 2014, https://github.com/OAI/ OpenAPI-Specification/blob/master/versions/2.0.md#schema.
[66] OpenStack API Documentation, https://docs.openstack.org/doc- contrib-guide/api-guides.html.

[67] OpenStack Project Navigator, https://www.openstack.org/ software/project-navigator/
[68] Ortega, M., Pérez, M., and Rojas, T., Construction of a Systemic Quality Model for Evaluating a Software Product, Software Quality J., Vol. 11, No. 3, Jul. 2003, pp. 219-242.
[69] ProgrammableWeb, https://www.programmableweb.com/.
[70] Rosa, M. L., van der Aalst, W. M. P., Dumas, M., and Milani, F. P. (2017) Business Process Variability Modeling: A Survey, ACM Computing Survey, 50 (1), Article 2, 1-45, http://dx.doi.org/10.1145/3041957
[71] Raemaekers, S., Deursen, A. v. and Visser, J., Measuring Software Library Stability through Historical Version Analysis, Proc. of ICSM 2012, IEEE, Sep. 2012, pp. 378-387.
[72] RELAX NG home page, http://relaxng.org/.
[73] Robillard, M. P., What Makes APIs Hard to Learn? Answers from Developers, IEEE Software, Vol. 26, No. 6, Nov./Dec. 2009, pp. 29-34.
[74] Romano, D. and Pinzger, M., Analyzing the Evolution of Web Services Using Fine-Grained Changes, Proc. of ICWS 2012, IEEE, Jun. 2012, pp. 392-399.
[75] SAX, http://www.saxproject.org/.
[76] Snoeck, M., de Oca, I. M.-M., Haegemans, T., Scheldeman, B., and Hoste, T., Testing a Selection of BPMN Tools for Their Support of Modeling Guidelines, Proceedings of IFIP Working Conference on the Practice of Enterprise Modeling (PoEM 2015), LNBIP Vol. 235, Springer, Nov. 2015, pp. 111-125, https://doi.org/10.1007/978-3-319-25897-3_8.
[77] Sohan, S. M., Maurer, F., Anslow, C. and Robillard, M. P., A Study of the Effectiveness of Usage Examples in REST API Documentation, Proc. of IEEE VL/HCC 2017, IEEE, Nov. 2017, pp. 53-61.
[78] SPARX Systems, Enterprise Architect, http://sparxsystems.com/.
[79] Stylos, J., Faulring, A., Yang, Z. and Myers, B. A., Improving API Documentation Using API Usage Information, Proc. of VL/HCC 2009, IEEE, Sep. 2009, pp. 119-126.
[80] 鈴木 純一，長瀬 嘉秀，田中 祐，松田 亮一，ソフトウェアパターン再考，日科技連，2000.
[81] Swagger UI, https://swagger.io/swagger-ui/.
[82] Swagger-diff, https://github.com/civisanalytics/swagger-diff.
[83] 上原 忠弘，山本 里枝子，吉田 裕之，パターン指向開発とパターン自動適用ツール，オブジェクト指向最前線 '98，オブジェクト指向 '98 シンポジウム論文集，情報処理学会/朝倉書店, Sep. 1998, pp.29-36.
[84] Uddin, G. and Robillard, M. P., How API Documentation Fails, IEEE Software, Vol. 32, No. 4, Jul./Aug. 2015, pp. 68-75.
[85] Wang, S., Keivanloo, I. and ZouS Y., How Do Developers React to Restful API Evolution?, Proc. of ICSOC 2014, LNCS Vol. 8831, Springer, Nov. 2014, pp. 245-259.
[86] 鷲崎 弘宜，丸山 勝久，山本 里枝子，ソフトウェアパターン，近代科学社，2007
[87] Weske, M., Business Process Management: Concepts, Languages, Architectures, Springer, 2012.
[88] Wittern, E., Ying, A., Zheng, Y., Laredo, J. A., Dolby, J., Young, C. C., and Slominski, A. A., Opportunities in Software Engineering Research for Web API Consumption, Proc. of WAPI 2017/ICSE 2017, IEEE, May 2017, pp. 7-10.
[89] Wohed, P., van der Aalst, W. M. P., Dumas, M., ter Hofstede, A. H. M., and Russell, N., On the Suitability of BPMN for Business Process Modelling, *Proceedings of the 4th International Conference on Business Process Management (BPM 2006)*, LNCS Vol. **4102**, Sep. 2006, pp. 161-176, https://doi.org/10.1007/11841760_1
[90] Woolf, G. and Hoepe, B., Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison Wesley, 2003.
[91] Yano, K., Nomura, Y., and Kanai, T., A Practical Approach to Automated Business Process Discovery, Proceedings of the 17th IEEE International Enterprise Distributed Object Computing Conference Workshop (EDOCW 2013), Sep. 2013, pp. 43-62, http://dx.doi.org/10.1109/EDOCW.2013.13
[92] 吉田 裕之，山本 里枝子，上原 忠弘，田中 達雄，UML によるオブジェクト指向開発実践ガイド，技術評論社, 1999.
[93] 山本 里枝子，上原 忠弘，大橋 恭子，中山 裕子，吉田 裕之，ビジネスアプリケーションむけパターン体系とその適用，第 124 回ソフトウェア工学研究会, Vol. 99-SE-124，情報処理学会, Oct. 1999, pp. 27-34.
[94] 岡部 一詩, API 経済圏，日経コンピュータ, 2016/5/26 号, pp. 18-31.

研究実績

(1) R. Yamamoto, K. Ohashi, K. Yamamoto, J. Inomata, and T. Matsuda, Lessons Learned from Requirements Analysis for Implementing Integrated Services, Proc. of SoRE 2004 (International Workshop on Service-Oriented Requirements Engineering), IEEE, Sep. 2004, 10 pages (査読有).

(2) 山本 晃治, 上原 忠弘, 松尾 昭彦, 大橋恭子, 猪股 順二 松田 友隆, 山本 里枝子, XML アプリケーション開発のためのパターン体系の開発, 情報処理学会論文誌, Vol. 45, No. 6, Jun. 2004, pp. 1584-1592 (査読有).

(3) P. van Eck, R. Yamamoto, J. Gordijn, and R. Wieringa, Cross-Organizational Workflows: A Classification of Design Decisions, M. Funahashi and A. Grzech (Eds.), Challenges of Expanding Internet: E-Commerce, E-Business, and E-Government, 5th IFIP Conference E-Commerce, E-Business, and E-Government (13 E’ 2005), Springer, Oct. 2005, 449-463 (査読有).

(4) R. Yamamoto, K. Yamamoto, K. Ohashi, and J. Inomata, Development of a Business Process Modeling Methodology and a Tool for Sharing Business Processes, Proc. of APSEC 2005 (Asia-Pacific Software Engineering Conference), Dec. 2005, IEEE, pp. 679-686 (査読有).

(5) 小高 敏裕, 松塚貴英, 野村 佳秀, 村上 雅彦, 山本 里枝子, SIP アプリケーションフレームワークの開発と適用, 情報処理学会論文誌, Vol. 48, No. 8, Aug. 2007, pp. 2674-2683 (査読有).

(6) K. Ohashi, H. Kurihara, Y. Tanaka and R. Yamamoto, A Means of Establishing Traceability Based on a UML Model in Business Application Development, Proc. of 19th IEEE International Requirements Engineering Conference (RE 2011), IEEE, Sep. 2011, pp. 279-284 (査読有).

(7) 宗像 聡, 安家 武, 上原 忠弘, 山本 里枝子, キャプチャデータからのテスト条件抽出に基づく OSS Web アプリのテストセットの評価と強化, コンピュータソフトウェア, Vol. 35, No. 1, Jan. 2018, pp. 124-139 (査読有).

(8) R. Yamamoto, K. Yamamoto, K. Ohashi, J. Inomata, M. Aoyama, A Metamodel-Driven Business Process Modeling Methodology and Its Integrated Environment for Reusing Business Processes, Journal of Software Engineering and Applications, Vol. 11, No. 8, Aug. 2018, Special Issue on Enterprise Software Technology, Scientific Research, pp. 363-382 (査読有).

(9) K. Ohashi, A. Katayama, N. Hasegawa, H. Kurihara, R. Yamamoto, J. Doerr and D. P. Magin, Focusing Requirements Elicitation by Using a UX Measurement Method, Proc. of 26th IEEE International Requirements Engineering Conference (RE 2018), IEEE, Aug. 2018, pp. 348-357 (査読有).

(10) 山本 里枝子, 大橋 恭子, 福寄 雅洋, 木村 功作, 関口 敦二, 梅川 竜一, 上原 忠弘, 青山 幹雄, Web API 品質モデルの提案とその定量評価, SES(ソフトウェアエンジニアリングシンポジウム) 2018 論文集, 情報処理学会, Sep. 2018, pp. 111-120 (査読有).

(11) R. Yamamoto, K. Ohashi, M. Fukuyori, K. Kimura, A. Sekiguchi, R. Umekawa, T. Uehara, and M. Aoyama, A Quality Model and its Quantitative Evaluation Method for Web APIs, Proc. of 25th Asia-Pacific Software Engineering Conference (APSEC 2018), IEEE, Dec. 2018, pp. 598-607 (査読有).

(12) 山本 里枝子, 大橋 恭子, 福寄 雅洋, 木村 功作, 関口 敦二, 梅川 竜一, 上原 忠弘, 青山 幹雄, WebAPI の特徴を捉える品質モデルとその定量評価方法の提案と適用評価, 情報処理学会論文誌 [査読中].