博士論文

拡張性を備えた企業アプリケーションのモデルベース設計開発手法に関する研究

公立はこだて未来大学大学院 システム情報科学研究科
システム情報科学専攻

田中　明

2014 年 3 月

Doctoral Thesis

Study on Model-based Design and Development Method for Extensible Enterprise Applications

by

Akira TANAKA

Graduate School of Systems Information Science

Future University Hakodate

March 2014

要旨

企業システムは企業活動をIT側から支援し円滑で確実なビジネスの遂行に寄与するものである。IT分野における終わることなき技術革新への対応や少子高齢化に見られる社会状況の変化から、企業システムを頻繁に起きる外部要因に基づく変化に対応させるための仕組みが必要となっている。

本論文ではこの問題に対する一つのアプローチとして、企業システムの枠組みとしてのエンタプライズアーキテクチャの利用、抽象レベルを上げモデリングを活用したシステム定義、モデリング技術に基づいた変化に対応出来る仕組みの提案、適用事例とその評価、そしてモデリング技術を利用した下流工程への展開方法についての研究を行う。

最初に、システムアーキテクチャについて動向を概観し、大規模システムの設計に有効な「関心の分離」のサーベイを行い、この原則に基づくアーキテクチャである各種エンタプライズアーキテクチャの概略比較を行い、この課題に相応しいアーキテクチャを選択する。

次に、アーキテクチャの表現手段の比較検討を行い、自然言語より厳密性の高い記述が出来るモデリング技術の利用法、すなわち、モデル記述手法、モデル変換手法、実際に利用出来るオープンなモデリングツール、等についてサーベイを行う。

次に、変化に対応出来る拡張性を備えたアーキテクチャの実現方法についてモデリングを活用した拡張方法を提案する。この方法に基づき5種類の拡張例について検証を行い、本提案方法が有効であることを確認する。

更に、拡張を行った企業システムのモデルを開発プロセスの中で単なる中間生成物とせず、下流工程への意味ある入力とする手段についてモデルベース開発の手法を用いて検討する。

最後に本研究の成果を総括すると共に今後の課題について述べる。

Summary

Enterprise systems exist within enterprises as a supporting tool for its daily business executions. Pressure to support never-ending technology innovations, together with limited number of IT personnel, suggests that we should find a way to revise or develop enterprise systems much more efficiently. However, no effective and agreed solution has been found so far.

This paper presents one approach, which utilizes use of enterprise architecture framework, use of modeling technologies to enable software development at higher level of abstractions, and then introduce a proposed mechanism to embrace changes e.g. coming from supporting new technologies.

First, summary of well-known system architectures are described followed by a summary use cases of "separation of concerns" principle applied. A comparison of enterprise architecture frameworks is shown and one of them (RM-ODP) is selected as base enterprise architecture for this research.

Secondly, a comparison study among languages for representing enterprise architectures is explained, followed by summary of modeling technology domain, including modeling languages, model transformation technologies, and tools implementing those technologies.

Thirdly, a method is proposed to allow enterprise architectures to relatively easily incorporate external technologies. An experimental implementation is shown to demonstrate the effectiveness of this method, including the architecture, the model, the transformation, with some generated artifacts, and summary of this result.

Finally, a summary of this research with future work is presented.

Keywords:

Enterprise Architecture, RM-ODP, Modeling, Model Integration, Model Transformation

目次

図目次

表目次

第1章　　はじめに

本章では，本研究の背景，目的ならびに構成について述べる．

1.1 研究の背景

社会的背景について説明する。いまだ有効な Moore's Law の貢献もあり IT 分野においては継続的技術革新が当然のことと認識されており、実際日々新たな技術が開発・導入されている。例えばクラウド上の音楽データをモバイル端末にダウンロードし再生することやモバイル端末で撮影した写真データを即座にソーシャルネットワークに投稿することなど、人々は一昔前には考えられなかった数多くの利便性を享受している。しかしながら、ハードウェアや通信技術やシステム形態の進化は新たなソフトウェア開発の要求へと向かい、それらが既存システムに組み込まれることでシステム全体のアーキテクチャが整然としたものから混沌としたものになる危険性を秘めており、またこのような機能追加の繰り返しを行うと、類似機能の重複開発等とともにシステム仕様の複雑化という維持管理にあたっての大きな問題を生み出すことになる。

1.2 解決すべき問題点

主にベンダーが提供する基盤ソフトウェアではなく、企業の日々のビジネスを支える業務処理アプリケーション部分を中心にシステム開発を考える。

競合力を備えた企業システムを持ち続けるには、一貫したシステム設計が重要であり、そのためには抽象化レベルを上げた設計が必要となる。それにはどのような設計方法があるのか、システム仕様への新技術導入を実現する一貫した仕組みは存在するか、設計仕様がシームレスに開発部門に伝えられる有効な仕組みはあるのか、そしてそれらを支援するツール群にはどのようなものがありどう使うべきか、といった課題を解決してゆく必要があり、これらを本研究での解決すべき問題点として設定した。

1.3 研究の目的

本論文では上の課題に対する一つのアプローチとして、アーキテクチャ記述手法やモデリング技術を活用することで、システム仕様記述の抽象レベルを上げ、変化に対応出来るシステム仕様の策定方法を探ることを研究の目的とする。抽象度を上げる手段として、まずエンタプライズアーキテクチャを利用したシステム構造の共通化を行い、次にモデリング言語を用いて企業システムの仕様記述を行う。更に、外部要因でもある IT 技術環境の変化に対応できる仕組みとその仕様記述への組み込み方を提案し、拡張した結果のモデルを変換処理につなげる。

1.4 論文の構成

第１章で本研究の背景と目的を説明した後、第２章ではエンタプライズアーキテクチャとモデリング技術の動向について概要を述べる。

第３章では、システムアーキテクチャについて動向を概観し、企業システムのような大規模システムの設計時に有効な「関心の分離（separation of concerns)」に基づく設計方法や適用事例のサーベイを行う。更に、この原則に基づくアーキテクチャである Zachman Framework、TOGAF、FEA、DoDAF/MODAF、RM-ODP といった各種エンタプライズアーキテクチャの概略比較を行い、この課題に相応しいアーキテクチャを提案する。

第４章では、モデリング技術の概要を述べ、自然言語より厳密性の高い記述が出来るモデリング言語に関する分析を行う。具体的には、モデル階層、UML、DSL、モデル変換技術などについて述べる。

第５章では、モデリング技術を実際に適用するため、主にオープンソースの世界でのツールに関するサーベイを行い、本研究への適用可能性を検証する。モデル開発に関連するツール利用の流れや、UML はじめ各種ツールを用いた場合のモデル記述例についても触れる。

第６章では、変化に対応出来る拡張性を備えたアーキテクチャの実現方法について、メタモデル要素の分析方法、メタモデル要素の関連付けパターン、複合概念の分割と結合方法の提案を行う。そして、この提案方法を五つの具体例に適用し、提案するシステム記述の拡張が可能であることを示す。

第７章では、提案する方法をサンプルシステムに適用した場合、モデル変換を利用し下流工程が使えるようなコード生成の可能性につき検証し、コード生成に適した DSL モデルについて考察を行う。

最後に第８章で本研究の成果を総括すると共に今後の課題について述べる。

第2章　エンタプライズアーキテクチャとモデリング技術の動向

2.1 はじめに

エンタプライズアーキテクチャ技術およびモデリング関連技術について、現在の研究状況を概観する。

2.2 エンタプライズアーキテクチャ技術

エンタプライズアーキテクチャは、John A. Zachman 博士による Zachman Framework[1]（1987 年）ジャーナル記事が最初とされ、それ以降多くの研究が行われている。概念としては定着しており、現在では実践者と研究者向けに EABOK コンソーシアムが Enterprise Architecture Body of Knowledge というサイト（http://www2.mitre.org/public/eabok/）で普及をはかっている。現在普及しているといわれるエンタプライズアーキテクチャのフレームワークには、Zachman Framework、Federal Enterprise Architecture（FEA）、The Open Group Application Framework（TOGAF）、DoDAF/MODAF などがある。米国の FEA をはじめ、各国の政府機関でカスタマイズしたエンタプライズアーキテクチャが利用されている。また、本論文で取り扱った ISO 標準の Reference Model for Open Distributed Processing は分散処理標準化のための参照モデルとして策定されたが、これらのエンタプライズアーキテクチャフレームワークとほぼ同じ領域をカバーしている。

現在の研究対象としては、オントロジ技術の利用、サービス指向との統合、モデリングや DSL との関連づけ、シミュレーションなど、IEEE EDOC Conference の Trends in Enterprise Architecture Research Workshop をはじめとして多くの場で研究活動が続けられている。

2.3 モデリング技術

(1) モデル定義技術

モデル記述の標準として Unified Modeling Language（UML）があり、その定義言語用としてメタメタモデルを規定する Meta Object Facility（MOF）も標準化されている。UML や MOF の維持管理や種々の問題領域用の拡張は開発元の Object Management Group（OMG）が行っており、UML 拡張としてエンタプライズアーキテクチャ用の仕様も作成されている。

研究領域としては、対象領域を絞った Domain-Specific Modeling Languages、

Meta-Meta-Modeling Languages、Metamodeling、シミュレーション、厳密性（Formalism）、オントロジ技術との関連付け、モデル駆動ソフトウェア開発、ツール連携、要求定義のモデリング、など多くの領域があり ACM の SPLASH Conference（http://splashcon.org/）や MODELS Conference（http://www.modelsconference.org）そして国内では情報処理学会ソフトウェア工学研究会といった場を中心に活発に研究活動が続けられている。

(2) モデル変換技術

モデル変換技術の研究は OMG の Model Driven Architecture（MDA）イニシアティブにより活性化し、OMG ではモデルからモデルへの変換仕様である MOF Query/View/Transoformation、そしてモデルからテキストへの変換仕様である MOF Model to Text Transformation Language が標準化されている。なお、MOF QVT には手続き的な変換を記述する QVT Operations とモデル間の関係を記述する QVT Relations の二つの言語が含まれる。

モデルからモデルへの変換については、OMG での標準化提案に刺激を受けフランスの University of Nantes や INRIA が OCL の上に開発した ATLAS Transformation Language（ATL）が論文も広く知られている。また国内では国立情報学研究所における Bidirectional Graph (Model) Transformation の研究が双方向モデル変換に向けての新たな取り組みとして知られている。これは変換後のモデルに改訂を加えても逆変換の際に返還前のモデルへその改訂を反映させる仕組みをグラフ変換として実現する研究で、The BiG Project（http://www.biglab.org/index.html）として成果がまとめられている。

モデルからテキストへの変換については、どうしてもテンプレート言語方式となるため、新たな領域、言語、プラットフォームへの適用、そしてモデル駆動ツールチェーンでの役割などが主な研究課題となっている。

第3章　　システムアーキテクチャについての分析と選択

企業システムを大きくとらえる枠組みとしてのシステムアーキテクチャについて、関心の分離（separation of concerns）及びその応用としてのエンタプライズアーキテクチャについて説明し、本論文で用いるため選択したアーキテクチャの概要を説明する。

3.1 はじめに

企業システムを対象とする場合、過去からの継続性そして将来の拡張性を考え、企業オリジナルや標準に基づいた何らかのアーキテクチャを用いることが多い。もしアーキテクチャを使用しなければ、開発のたびに任意の構造を備えた新たなサブシステムが生み出され、他のサブシステムやコンポーネントなどとの統合や全体に渡っての維持管理が容易に行えないためである。すなわち、システムアーキテクチャを利用する目的は、ある共通のシステム構造を設けることで、開発作業や既存システムとの統合作業を容易にし、結果としてシステム開発効率や維持管理効率の向上に寄与するためである。

システムアーキテクチャをどう設計するかという観点では、例えば基盤部分と応用機能部分またはクライアントとサーバという各々二つの構成要素からなるアーキテクチャを想定したとして、与えられた機能要素をどちらに分類するかという例を考える。判断基準は「基盤部分かそれ以外（応用部分）か」「クライアントかそれ以外（サーバ）か」であり、他の要素はすべて排除し特定の関心事だけで判断を行うことが分かる。このように特定の関心事に注目すると、システムの特定の関心領域に特化した姿を浮かび上がらせることが出来る。この考え方は、システム設計において「関心の分離」として知られており、通常幾つかの関心を設定することで、それぞれの場合のシステム要素を抽出する。

関心の分離について幾つかの例として、アスペクト指向プログラミングにおける横断的関心事、ISO 42010 (IEEE 1471)[13] で標準化されているビューポイント、そして Zachman フレームワークのパースペクティブ等がある。Model, View, Controller の 3 要素で構成される MVC アーキテクチャもデータ要素を Model に限定するという意味で関心の分離の適用例であるし、Web システムにおける 3 階層アーキテクチャも類似の適用例である。

関心の分離を適用すると複数の一見独立した仕様が得られるが、これらは本来全体として元の仕様を表現するものであり、実際には独立したものではなく、何らかの手段によりそれらの間の関連付けを行う必要がある。例えば MVC の場合は三つのコン

ポーネント間にインタラクションを持たせることで、これを実現している。

関心の分離を大規模システムや企業システムに適用し、システムアーキテクチャ設計の枠組みとして整理したものはエンタプライズアーキテクチャと呼ばれている。代表的なものを列挙する。

3.2 Zachman Framework

Zachman Framework は John A. Zachman 博士が 1987 年にジャーナル記事として発表したのが最初とされ、その後も進化を続けている。6 行・6 列のマトリクスとして表現されるフレームワークないしはスキーマで、最新の第 3 版では副題をエンタプライズ・オントロジとし、行は Audience Perspective で Executive、Business Mgmt、Architect、Engineer、Technician、Enterprise の各 Perspective、列は Classification Names として 5W1H（What, How, Where, Who, When, Why）となっている（表 3-1）。この 6 種類の Perspectives は関心の分離を表し、各関心領域を更に 5W1H で細分化している。但し、このフレームワークは特定の記法を定めておらず、この考え方に基づきアーキテクチャ設計を行うという使い方が想定されている。

表 3-1 Zachman Framework

The Zachman Framework for Enterprise Architecture™
The Enterprise Ontology
Version 3.0

classification / perspective	What	How	Where	Who	When	Why	classification / model
Executive Perspective	Inventory Identification	Process Identification	Distribution Identification	Responsibility Identification	Timing Identification	Motivation Identification	Scope Cotexts
Bussiness Mgmt Perspective	Inventory Definition	Process Definition	Distribution Definition	Responsibility Definition	Timing Definition	Motivation Definition	Business Concepts
Architect Perspective	Inventory Representation	Process Representation	Distribution Representation	Responsibility Representation	Timing Representation	Motivation Representation	System Logic
Engineer Perspective	Inventory Specification	Process Specification	Distribution Specification	Responsibility Specification	Timing Specification	Motivation Specification	Technology Physics
Technician Perspective	Inventory Configuration	Process Configuration	Distribution Configuration	Responsibility Configuration	Timing Configuration	Motivation Configuration	Tool Component
Enterprise Perspective	Inventory Instantiations	Process Instantiation	Distribution Instantiation	Responsibility Instantiation	Timing Instantiation	Motivation Instantiation	Operations Instances
perspective / enterprise	Inventory Sets	Process Flows	Distribution Networks	Responsibility Assignments	Timing Cycles	Motivation Intentions	

3.3 The Open Group Application Framework (TOGAF)

メンバーシップにより構成されるコンソーシアムの The Open Group によると TOGAF[4]はエンタプライズアーキテクチャを開発するためのフレームワークで詳細の手法と一連の支援ツールから成る。その仕様は 7 部構成で、Introduction, Architecture Development Method (ADM), ADM Guidelines and Techniques, Architectural Content Framework, Enterprise Continuum & Tools, TOGAF Reference Models, Architecture Capability Framework である。このうち本研究と関連を持つのは ADM である（図 3-1）。また関連して Archimate[5]というモデリング言語も提供している。

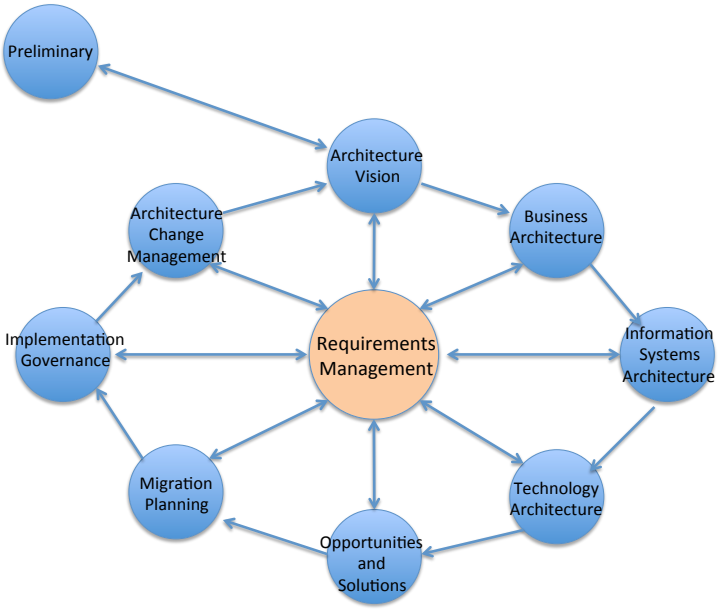

図 3-1 TOGAF ADM 概略図

TOG が推進しているため、コンソーシアム標準という位置づけのものとなる。

3.4 Federal Enterprise Architecture

FEA[2][3]は米国連邦政府の一部であり国家予算の開発を担当する Office of Management and Budget (OMB) が推進する米国政府のエンタプライズアーキテクチャであり、戦略的計画、ビジネス活動、データ及び情報、システムとアプリケーション、ネットワークとインフラストラクチャ他の切り口（関心事）で全ての政府部門の活動を体系化することで効率的な政府運営を目指すものである。大きな枠組みと各

種参照モデル（Performance Reference Model, Business Reference Model, Service Component Reference Model, Data Reference Model, Technology Reference Model 等）が含まれる。

日本政府も米国政府の FEA を受け、政府の業務を対象とした「業務・システム最適化計画」を策定している[8]。

3.5 DoDAF

DoDAF[6]は米国防衛省（Department of Defense）のアプリケーションフレームワークで前項 FEA に基づくものである。ここで DoDAF を取り上げた理由は、DoDAF が FEA の連邦政府組織内でのカスタム化の一例であることと、OMG で UML Profile for DoDAF/MODAF としてアーキテクチャ記法の標準が制定されていること、による（MODAF[7]は英国政府の DoDAF 相当のもの）。現在 OMG から ISO/TC 184/SC 4 Industrial data に PAS 提案され、国際標準化が進められている。

3.6 RM-ODP

Reference Model for Open Distributed Processing[9][10][11]は ISO と ITU-T が共同プロジェクトとして策定した国際標準であり、オープンな分散システムの仕様記述方法を定めている。オブジェクトベースの考え方で、５つの標準ビューポイント（関心事）として Enterprise, Information, Computational, Engineering, Technology を、また各ビューポイントにおいて仕様記述に必要な概念や規則を定めている[14-30]。この標準は記法についての規定が無かったため、後ほど Use of UML for ODP system specifications[12]という UML Profile 標準も制定されている。

最初に本研究で対象とするエンタプライズアーキテクチャを選択するため、これらの間の比較を行う（表 3-2）。比較項目として、対象領域、関心領域、構造化、記法、オープン性を用意した。

表 3-2 フレームワーク比較

	対象領域	関心領域	構造化	記法	オープン性
Zachman Framework	一般	エグゼクティブ ビジネス管理 アーキテクト エンジニア テクニシャン エンタプライズ	関心領域ごとに 5W1H 適用	自然言語 UML?	詳細は公開されていない（最も知られたフレームワーク）
TOGAF	一般	ビジネス データ アプリケーション テクノロジ	アーキテクチャ開発ステップとして個別詳細化	自然言語 UML	情報は公開されている
FEA	政府	戦略的計画 ビジネス活動 データと情報 システムとアプリケーション ネットワークとインフラストラクチャ	共通的な構造化手法は無く個別詳細化	自然言語 UML Profile?	情報は公開されている
DoDAF	政府	全視点 ケーパビリティ データと情報 オペレーショナル プロジェクト サービス 標準 システム	共通的な構造化手法は無くビューポイントごとに個別詳細化	自然言語 UML Profile	情報は公開されている
JEA	政府	政策・業務体系 データ体系 適用処理体系 技術体系	共通的な構造化手法は無く個別詳細化	自然言語 UML Profile	情報は公開されている
RM-ODP	一般	エンタプライズ 情報 コンピュテーショナル エンジニアリング テクノロジ	ビューポイント言語	自然言語 UML Profile	標準文書は ITU-T サイトで公開

3.7 システムアーキテクチャの選択

どれも実績のあるアーキテクチャではあるが、本研究で利用するため次の要件を用意した。

- アーキテクチャの技術的内容が文書として公開されていること
- アーキテクチャをサポートする UML Profile が規定されている場合は、そのデータが公開されていること
- 研究素材として利用して問題が出ないこと

これらを全て満足するものとして、本研究では RM-ODP を選択することとした。

3.8 エンタプライズアーキテクチャを用いた業務システム開発

一般的に、業務システム開発プロセスの上流工程で共通的な枠組みとして独自のものも含めアーキテクチャが導入される。次の図 3-2 はアーキテクチャの位置づけを示すため、開発プロセスの上流部分について主だった開発活動と成果物を示している。なお、実際のシステム開発ではこれ以外にも画面設計、出力帳票設計、非機能要件対応など多くの活動がある。

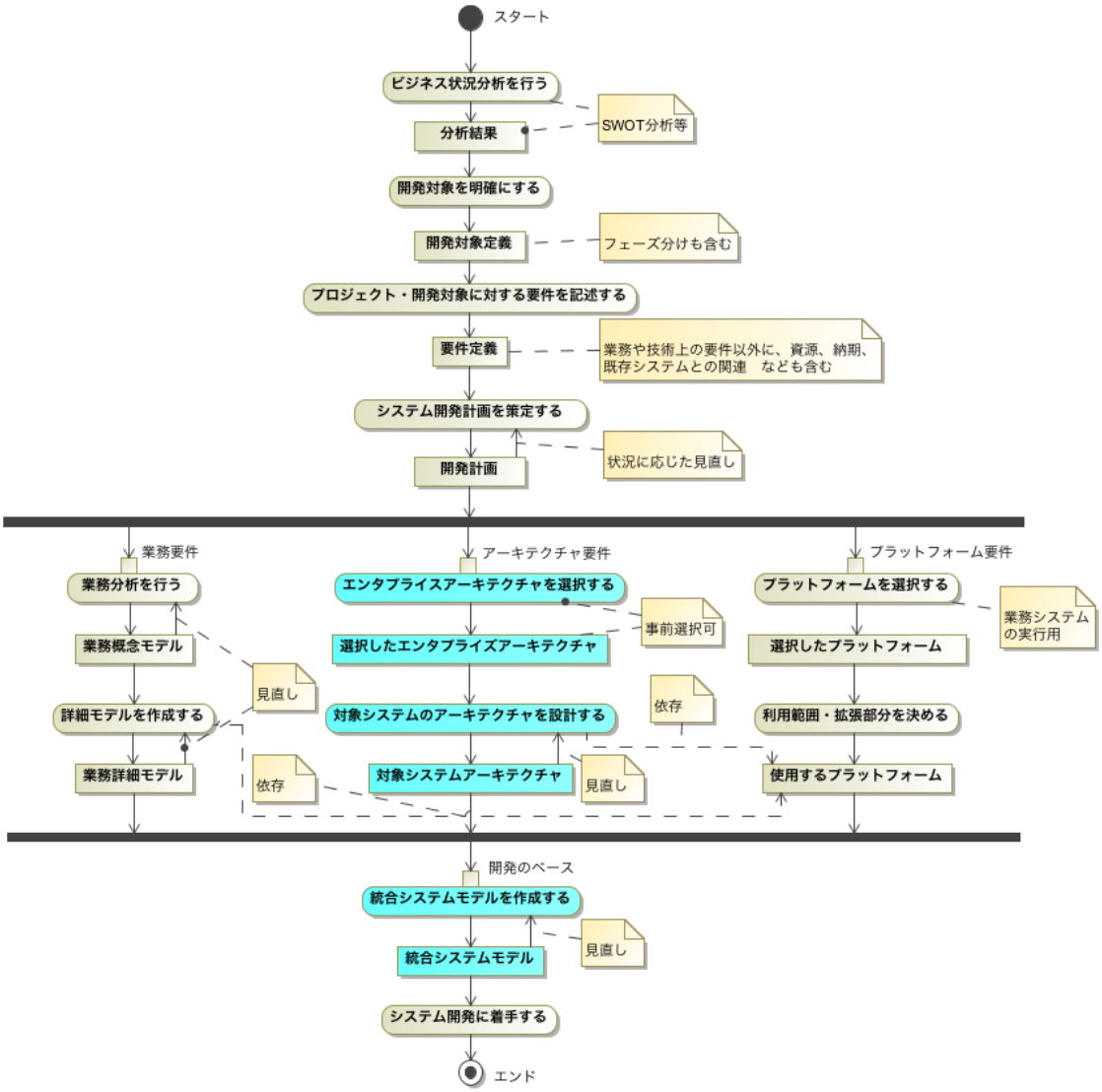

図 3-2 開発プロセス上流の概要

ここで改めて用いたアーキテクチャに関する言葉について整理する。

エンタプライズアーキテクチャは、企業システムを対象としたもので、ビジネス上の目標を実現するために、その企業にはどんな要素や制約があり、どのような実現プ

ロセスがあり、それを支えるためにどのような技術要素があり、どんな考え方や構造に基づいてそれらを組み合わせるのか、について総合的に記述したもので、複数の観点（関心の分離）から記述されることが多い。

システムアーキテクチャは、IT システムにまつわるアーキテクチャの記述であり、種々の技術要素をどのように構成することで当該IT システムを実現するかの記述でもある。

モデルは、対象から不要な詳細を削除し、対象の本質的な部分だけを抜き出して特定の言語（例：UML）を使い表現したものである。一般にアーキテクチャは通常文書として作成されるが、部分であってもモデル表現することで厳密さを加えることが出来る。上の図で、業務概念モデルは業務概念だけを対象としたモデルであり、業務詳細モデルは業務概念モデルでは削除していた詳細な属性や操作を追加したモデルであり、統合システムモデルは詳細化したシステムアーキテクチャも含めたエンタプライズアーキテクチャをモデルとして表現したものである。

図２-2 では、ビジネス状況分析には BMM 標準[36]を用い、青色を付した部分でエンタプライズアーキテクチャに基づくアーキテクチャ開発を行っている。ここでの成果物である、対象システムアーキテクチャ、業務概念モデル、業務詳細モデル、などは下流工程で利用出来るように標準記法（UML[31][32]）でモデルとして記述されるべきであり、エンタプライズアーキテクチャ用の UML Profile、汎用 UML Profile[38][39]やプラットフームに対応した UML Profile を用意するなどし、UML ベースの統合システムモデルにまとめ上げることで上流工程の判断を下流工程に滑らかに伝搬させることが出来る。

3.9 まとめ

本章では、企業システムが業務システム開発に利用できるエンタプライズアーキテクチャにはどのようなものがあり、どのような特徴を持つかを概観した。また、これを有効に利用するには開発プロセスの上流でどんな手順が想定されるか説明した。

第4章　　モデリング技術の概要

標準化動向も含め、現在活用出来るモデリング技術について概観する。モデル駆動開発（MD*）、モデル記述仕様（MOF[35], UML, DSL）、モデル変換仕様、などについても対象とする。

4.1 はじめに

物理学や工学の世界では、対象となる物理現象の仕組みを究明するため、数学などに基づくモデル（数式）を構築し、実際の現象を観測し、得られた結果とモデルのシミュレーション結果の傾向を比較し、それにより提案するモデルがどの程度現象を説明できているかを検証するという方法が行われる。ソフトウェアの世界におけるモデリングもこれに似たところがあり、現実のそれも人が生み出した概念や規則類に基づき動いている世界に対して、コンピュータを用いた処理を含んだモデルを作成し、そのモデルの妥当性検証はモデルに基づいたシステム実装後に頭の中で行ったシミュレーションの結果も含むテストデータを用いて比較検証する（ユニットテスト等）ということが行われる。別の見方をすると、複雑な対象を理解し仕様規定するため、対象の分割や抽象化といった複雑さの度合いを軽減する手法が用いられており、それを書き下したものがモデルと言える。そうしたモデルの意味付けについての研究の結果モデル階層の考え方が生まれ、モデル記述のためのモデリング言語や手法が発展し、その応用としてモデル駆動開発手法が研究され、実現するメカニズムとしてモデル変換が開発されたとも理解出来る。

4.2 モデル階層

データを説明するデータがメタデータと呼ばれるが、モデルを定義するためのモデル（抽象構文）はメタモデルと呼ばれる。モデリングの世界では、一般的にメタに関して以下の図 4-1 で示す 4 階層が用いられる。

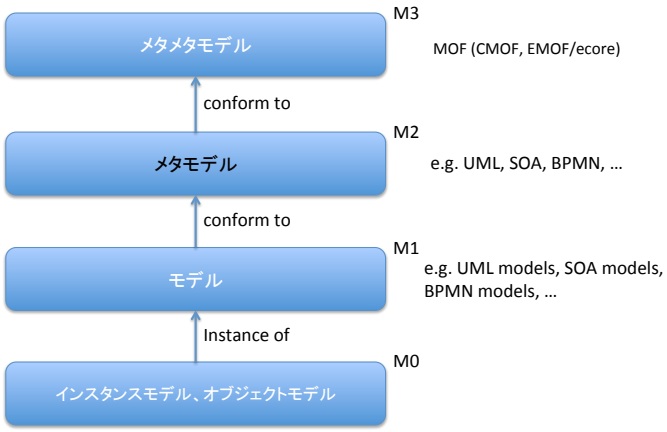

図 4-1 モデル階層

(3) メタメタモデル（M3）

メタモデルを記述するための抽象構文が必要であり、それがメタメタモデルと呼ばれる。標準としては後で説明する MOF がその仕様にあたり、記法としては UML Class 図のサブセットを用いる。オープン実装としては Eclipse Modeling Framework が MOF のサブセットである EMOF を実現している。

(4) メタモデル（M2）

モデルを記述する為の抽象構文がメタモデル（M2 モデル）であり、UML メタモデル、BPMN[73][74]メタモデルなど多数の実例がある。メタモデルの記述は UML クラス図のサブセットに制約や記法規定を加えたものとなる。規定されたそれぞれがモデリング言語であり、その仕様を実装したものがそのモデリング言語用ツールとなる。

(5) モデル（M1）

記述対象を抽象化しその型についての構造面及び振舞い面の定義情報、関連情報、制約情報などを M2 モデル規定に従った形で記述したもので、利用者の選んだモデリング言語に応じたツールを用い作成出来る。UML ツールを使い作成する UML モデルが最も一般的ではあるが、DSL に基づくモデルなど他の形態のモデルも存在する。

(6) インスタンスモデルまたはオブジェクトモデル（M0）

ある瞬間（時刻）におけるオブジェクトの値や関連をスナップショットの形で表現したもの。まず M0 モデルを作成し、これを抽象化することで M1 モデルを得るというモデル作成方法や、M0 モデルを作成することで、そのベースとなった M1 モデルの妥当性検証に利用することもある。いずれにせよある時点の値と関連であるため、単一のモデルだけでは常に正しいモデルに到達出来ない可能性があり、カバレッジを考え通常は幾つもの M0 モデルを作成する。

4.3 モデリング言語

モデルを作成するためにはモデル記述の為の言語が必要となる。数式を使ったモデルの場合モデリング言語は数学であり、グラフィカルモデルの場合はグラフィカルモデリング言語となり、テキスト型モデルの場合はテキスト型モデリング言語である。対象がソフトウェアの場合、一目で全体を見渡すことが出来るグラフィカルなモデリングが使われることが多いが、例えばプログラミング言語のソースコードでモデルを記述することも出来る。標準となったもの（例：UML や BPMN）もあれば、標準にはなってはいないもの（例：数式を用いるものや対象領域専用の言語など）もある。標準には多くの関係者が共通言語で書かれたモデルを理解出来るという点で優れているが、対象領域によってはベーシックなモデリング標準のカバーする範囲外の要求が現れる場合もある。モデリング言語の標準化動向に含まれない DSL（Domain Specific Language）[81][82][85][86]について少し補足する。DSL は特定の対象領域に特化したモデリング言語であり、次のような分類がある。一つ目は External DSL と Internal (or Embedded) DSL という分類で、母体となるプログラミング言語の有無がポイントとなる。例としては、Ruby on Rails[87][91]は Ruby 言語に基づく Internal DSL であり、SQL ステートメントはプログラミング言語に依存しない External DSL であると言える。更に、グラフィカル DSL とテキスト型 DSL という分類がある。先の Ruby on Rails や SQL はテキスト型 DSL であり、音楽の楽譜や化学記号などはグラフィカル DSL と言える。DSL の実装系については後で説明する。

4.4 標準化動向

ソフトウェアのモデルをどう表現するかは、主立ったモデリング手法から記法を中心に OMG で標準化が行われた UML（Unified Modeling Language）が業界標準となっている。UML には構造に着目したモデルを表現する部分と振舞いに着目したモデルを表現する部分がある。また、UML の提供する機能範囲に収まらないモデリングを行うために拡張機能（Profiling）が用意されている。UML 標準以外にも UML のメタモデルを定義するために用いられる MOF（Meta Object Facility）や XML 形式のデータ交換書式である XMI（XML Metadata Interchange Specification）[34]などが OMG で標準化され、主要な仕様は ISO 標準になっている。更に、モデルを作成した後に使われるモデル変換に関わるものとして、モデルからモデルへの変換の為の MOF/QVT 仕様とモデルからコードを含むテキストへの変換の為の MOF2Text 仕様が標準化されている。ただこれらは全て仕様であり、実際に利用者が手を触れて使えるものではない。実装には商用ツールやオープンソースツールが存在するが、研究に利用し易いも

のとして Eclipse Modeling Project に存在するオープンソース実装群がある。この詳細については後ほど説明する。

4.5 モデル駆動開発

ソフトウェア開発をモデルに基づき行うことをモデル駆動開発と呼ぶが、モデル駆動の呼び名次のように多くある：Model Driven Software Engineering, Model Driven Engineering, Model Driven Architecture, Model Driven Design and Development, … 。そのため全体をカバーする場合は MD*という呼び名を使うこととする。モデル駆動が現れた理由の一つは、モデリング活動とソフトウェア開発の間にあるギャップである。現実のソフトウェア開発プロジェクトでは、顧客の持つ要件をビジネスアナリストの役割を果たすコンサルタントなどが引き出し、それを文書や業務モデル等として作成し、成果物を開発者へと渡す。そして、開発者は要求仕様書やモデルを参考にしてプログラム開発を行う。プログラム開発が出来た段階で何段階ものテストを行い顧客の確認を受け納品する。この手順の問題は、顧客は技術的成果物であるモデルの検証を十分に行うことができないこと、そして作成されたモデルが作成に関与していないプログラマに渡され、その意味解釈を誤る可能性が加わること、そしてプログラマ独自の解釈に基づくコードが作成される可能性があることである。当然なんらかのチェック機能を用意するが、意味を 100%伝えるのは困難であることに加え、顧客要求自体が変化する可能性もある。従って、せっかく業務要件を厳密に表現したモデルを作成してもフィードバックループが弱く、更に以降の工程でモデルを有効に活用出来ていないということになる。この課題に対応するため、要求仕様を含むモデルを重視し、プログラムを書く段階で人による解釈の入る余地を出来るだけ排除しようというモデルベース開発ないしモデル駆動開発という考え方が生まれた。このきっかけとなったのが OMG の MDA イニシアティブである[40][41]。MDA では OMG の分散オブジェクト管理アーキテクチャ（OMA）とモデリング標準をベースとし、プラットフォーム独立のモデル（Platform Independent Model）、プラットフォーム依存のモデル（Platform Specific Model）、そしてそれらの間のマッピングに注目する。これは、OMG が分散オブジェクト標準 CORBA の普及推進の経験から、ミドルウェアプラットフォームの業界での標準化は困難という認識のもと、プラットフォーム依存性を排除したモデルとそれからプラットフォーム毎のモデルへとマッピングする仕様に論理的に分割するという着想を得ていたためである。MDA は次の要素から構成される。

モデリング言語：UML

CIM（Computation Independent Model）：純粋な業務モデル

PIM：業務モデルを計算機処理出来る形にしたもの

PSM：PIM をプラットフォーム対応にしたもの

モデル変換（モデルからモデル）：標準は MOF/QVT[33]で、例えば CIM から PIM、PIM から PSM へというようなモデルマッピングを行う仕様

モデル変換（モデルからテキスト）：標準は MOF2Text[42]で、例えば PSM から JEE 用 Java コードや XML ファイルを生成するような場合を想定したテンプレートエンジン入力仕様

MDA では反復的なプロセスを想定し、CIM から PIM、PIM から PIM または PSM、PSM から PSM またはテキストの各ステップがフィードバックループを含むものとなっている。考え方の概略を図 4-2 に示す。

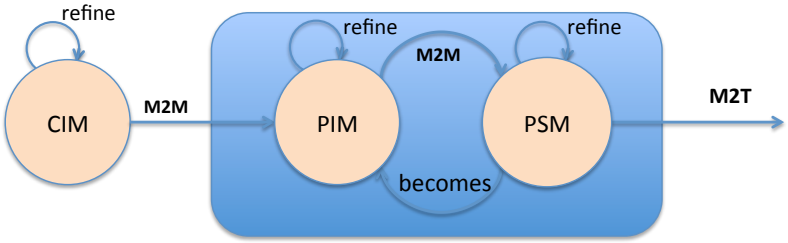

図 4-2 MDA の考え方

アーキテクチャ中心モデル駆動開発

ソフトウェアシステムの開発では業務分析フェーズや設計フェーズで過去の知見から得られた設計パターンに基づく構造をアーキテクチャとして利用することが多い[84]。例えばクライアント・サーバ、3 階層、MVC、Publish-Subscribe などをあげることができる。特定のアーキテクチャに基づくシステムモデルを作成する場合、アーキテクチャを単なるパターンとして設計することも可能であるが、維持管理の段階で正しく引き継がれない恐れがある。そのため、適用するアーキテクチャをメタモデルとして捉え、その UML マッピングとして UML Profile を定義し主要要素を stereotype とする方式がある。こうすることで、少なくともモデル要素に stereotype というマーク付けができ、必要な属性を定義するスロットを用意でき、また必要に応じ制約を付加することが出来る。これにより、維持管理の段階で誤解されるケースを確実に減らすことが出来る。UML モデルがこういった stereotype を含んでいる場合、モデル変換で stereotype 毎に個別の変換ロジックを記述できるためモデルのテキスト変換時に対象コンポーネント毎のテンプレートを記述出来る。この手法は先に述べたエンタプライズアーキテクチャに対しても適用でき、実際 OMG で UML Profile for DoDAF and MODAF[37]が標準化され、ISO で Use of UML for ODP system specifications が標

準化されている。

4.6 モデル記述手法

一般論

抽象化を行うということはある観点から対象を眺め、その観点とは無関係な詳細を捨て去ることでもある。モデル記述手法における観点は主に構造に関わるものと振舞いに関わるものに分類出来る。

UML では、構造的モデルと振舞い的モデルに分類されており、構造的モデルの図としては Package 図、Class 図、Object 図、Component 図、配備図などが、振舞い的モデルの図としては Activity 図、State Machine 図、Sequence 図、Timing 図、Use Case 図などがそれぞれの用途に合わせて利用出来る。UML を使わない場合、使用するモデリング言語に必要な構造的モデルと振舞い的モデルの記述手段が存在するかを確認し、必要なものが無ければ機能追加を行うか言語の変更などを検討する必要がある。

1) MOF

MOF モデルに基づくメタモデル記述には次の二通りの手法が用いられることが多いが、これで全てという訳ではない（DSL の項を参照）。

(1) UML クラス図を使った MOF モデル記述

MOF 仕様書には EMOF の場合と CMOF の場合に分けて UML クラス図で MOF モデルを記述する際の制約項目がリストされており、それらに従い UML ツールを用いクラス図として MOF モデルを作成する。作成した MOF モデルが EMOF モデルの場合、UML ツールによっては Ecore 形式での保存ないし Ecore 形式への export が可能で、その場合には次項につなげることが出来る。次の図 4-3 は UML クラス図を使った MOF モデル記述の例である。

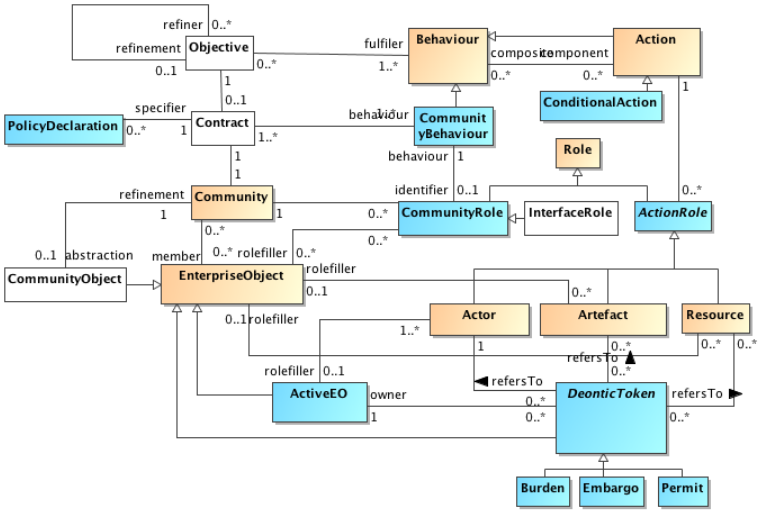

図 4-3 UML による MOF モデル例

(2) Ecore を使った Ecore モデル記述

オープンソースの統合開発環境である Eclipse には当初より Eclipse Modeling Framework（EMF）[47][48]を主要コンポーネントとして組み込まれている。この EMF で扱うモデルは Ecore 形式と呼ばれ、これは EMOF の XMI 形式表現に相当する。EMF を組み込んだ Eclipse を起動し、Diagram Editor for Ecore を利用すると Ecore ファイル（EMOF モデル）を UML のクラス図のようにグラフィカルに定義することが出来る（図 4-4）。またこの実装では、定義したモデルは本来 M2 レベルのモデルであり、そのルートノードから動的インスタンスを生成し M1 レベルのモデルを作成することも出来る。

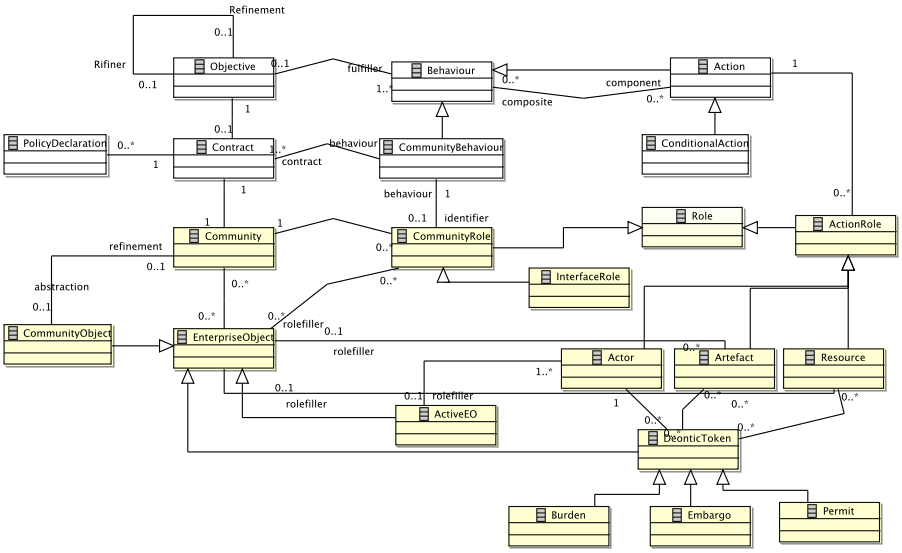

図 4-4 EMF による MOF モデル例

2)　UML

(1) UML モデル記述

一般的に UML モデルは専用ツールである UML ツールを用いて記述する。各種の商用ライセンス製品の他に ArgoUML や Eclipse Papyrus[50]などのオープンソースライセンスのものや、商用でもフリー版を持つものなど多数のツールが存在する。GUI はツールにより異なるが、基本的にはブランクダイアグラムにパレットからモデル要素の図形表現をドラッグ＆ドロップし。属性設定や他の要素との間の関連付けを行うことでモデルを完成させる。以下の図 4-5 は典型的な UML ツール（MagicDraw）の GUI を示す。

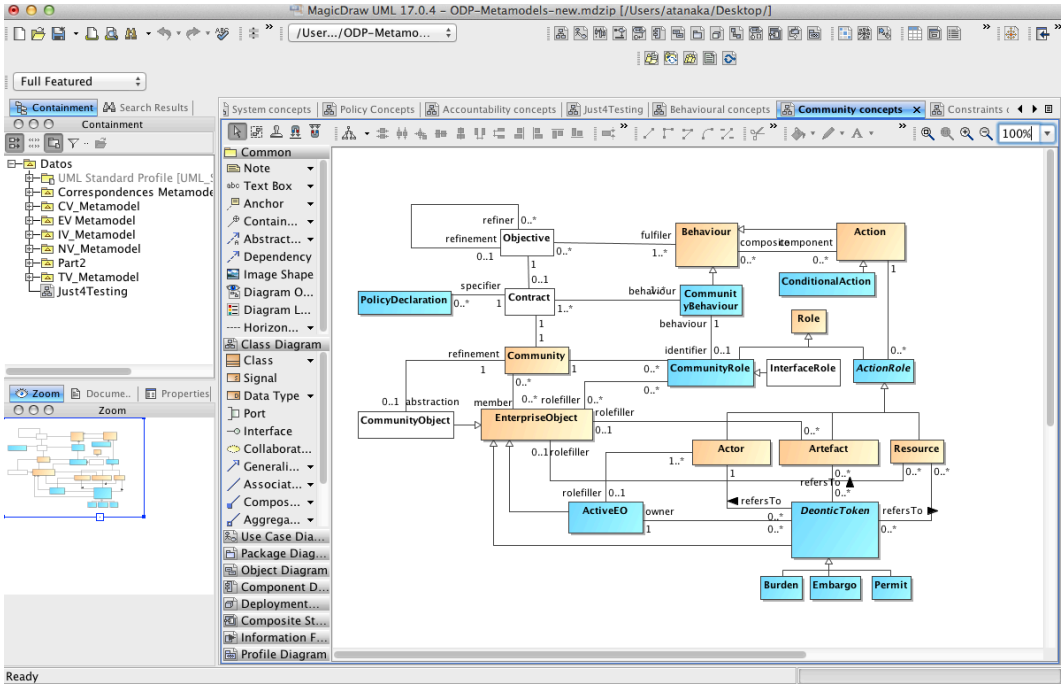

図　4-5 UML ツール GUI 例

(2) UML Profile 利用

MOF モデル（モデリング言語）に従った UML モデルを作成したい場合、UML ツールの機能範囲を超える場合 UML Profile を作成しそれをアドインやプラグインの形で UML ツールに組み込み利用することが出来る（ツール依存）。UML Profile は UML 仕様の一部であり、UML meta-class を拡張する stereotype を定義し属性や制約や関連を与えることで定義とする。OMG では多数の UML Profile 標準が開発されており、それらも同様に UML ツールに組み込み利用することが出来る。以下の図 4-6 と図 4-7 は、それぞれ UML Profile の定義例（RM-ODP のエンタプライズ言語用 Profile：着

色部分は UML のメタクラス）とその利用例（一部）である。

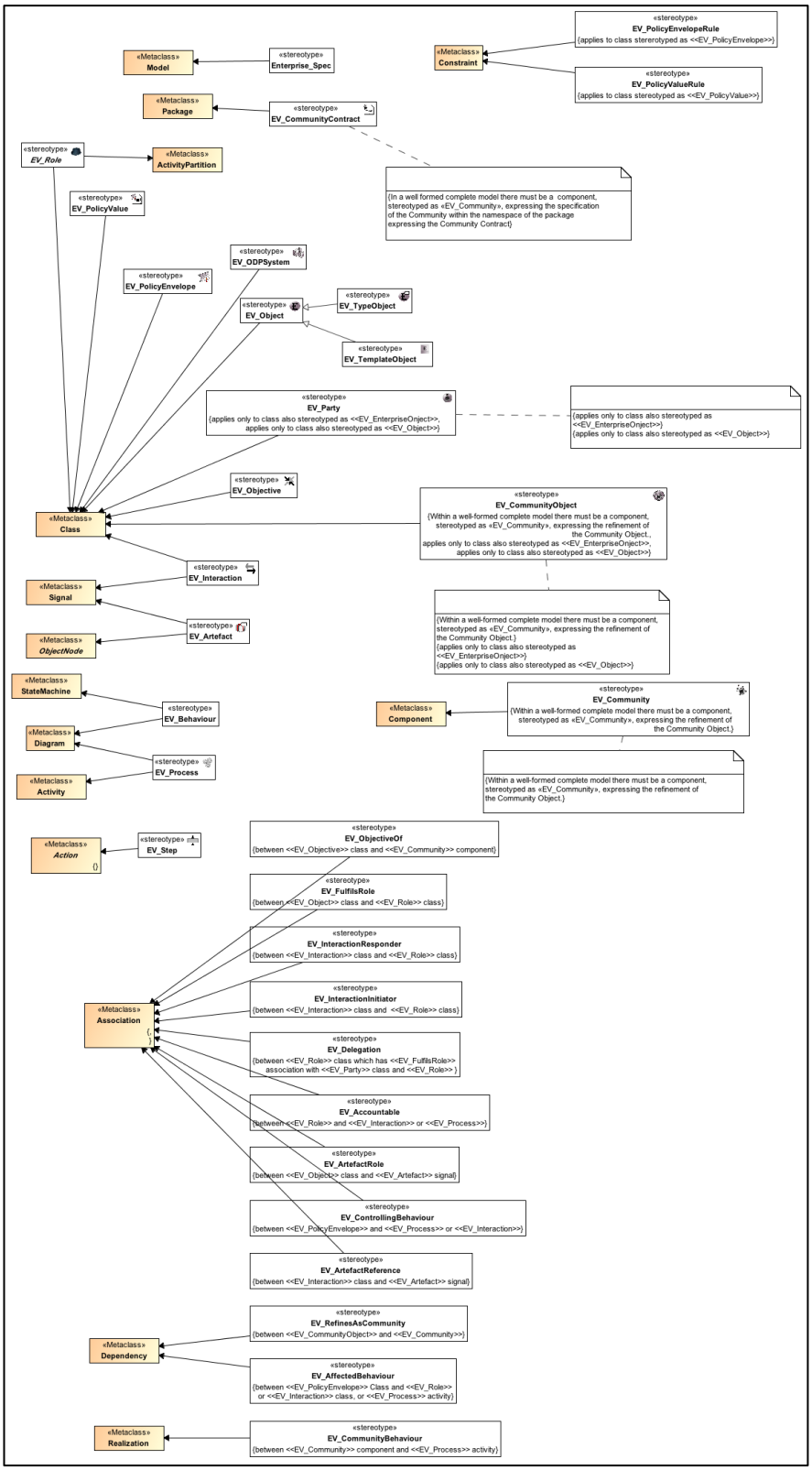

図 4-6 UML Profile 例

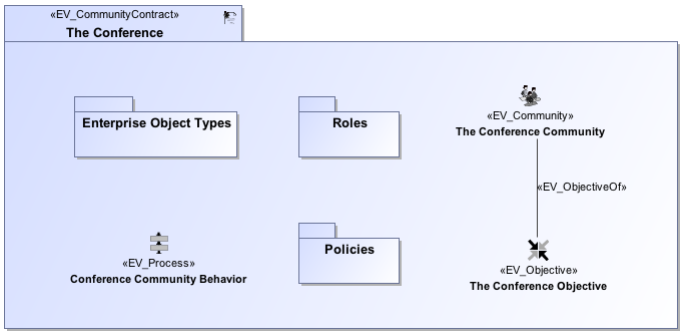

図 4-7 UML Profile 適用例

3) DSL

Domain Specific Language（DSL）は特に新しい概念ではなく以前からある特定領域のモデルを記述するための専用言語である。一般的には、例えば Relational Database の世界の SQL、Unix の世界の各種コマンド（例：sed）、音楽を記述する楽譜、化合物等を記述する化学記号、等々である。DSL には少なくとも以下のような 3 種類の分類がある。

(1) Graphical DSL

External DSL の一種、すなわち母体となるプログラミング言語は持たないもので、UML のような汎用のグラフィカルモデリング機能が必要ない場合、若しくは汎用のグラフィカルモデリング機能を使うとその後の処理（モデル変換やテキスト生成）が複雑になる場合に用いられる。商用ツールでは MetaEdit+[89]という製品がこの分野で良く知られている。オープンソースの世界では Eclipse の Graphical Modeling Framework（GMF）[49][51]とその周辺の各種ツール[52]がこれに相当する。パッケージに含まれている Diagram Editor for Ecore を用い EMOF モデル（Ecore ファイル）を作成し、モデル要素の図形との対応付けやパレット定義などを加えることでグラフィカルエディタを生成しドメインモデルを作成する。そうして作成したモデルは XMI 形式で保存出来るため、次のフェーズであるモデル処理への入力として引き継ぐことが出来る。

以下、図 4-8 は Ecore モデルをグラフィカルに定義する例、図 4-9 は同じモデルを Sample Reflective Ecore Editor で表示した木構造表現の例、図 4-10 はテキストベースの編集ツールで同じモデルを編集する例である。

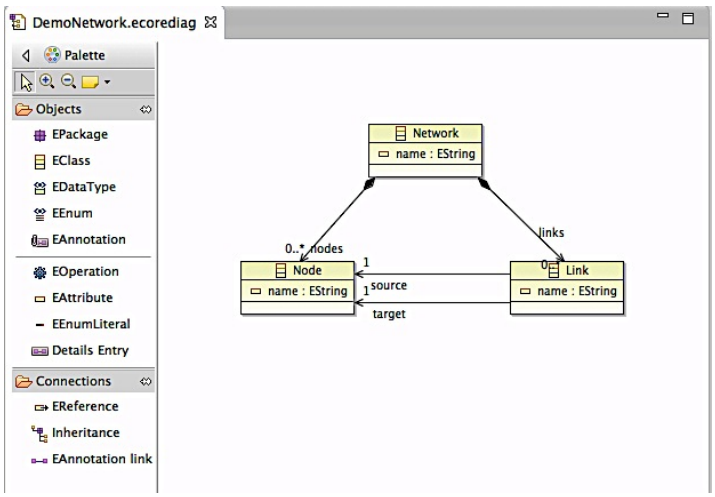

図 4-8 Ecore モデル例（1）

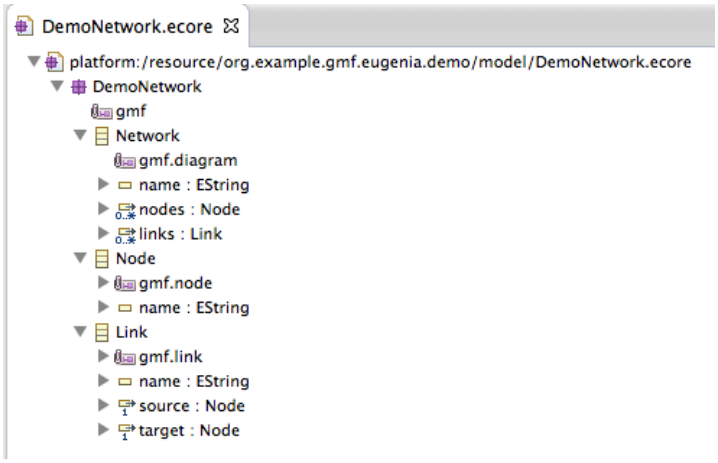

図 4-9 Ecore モデル例（2）

図 4-10 Ecore モデル例（3）

以下の図 4-11 と図 4-12 は、前掲の図 4-8 から図 4-10 までで示した Ecore モデルのグラフィカル DSL エディタを作成しモデルを記述した例とそのモデルの XML 表現例を示したものである。また図 4-13 は Eclipse のグラフィカルエディタ作成にあたり実績を持つ GMF のダッシュボード画面で、エディタ作成までの手順も示している。

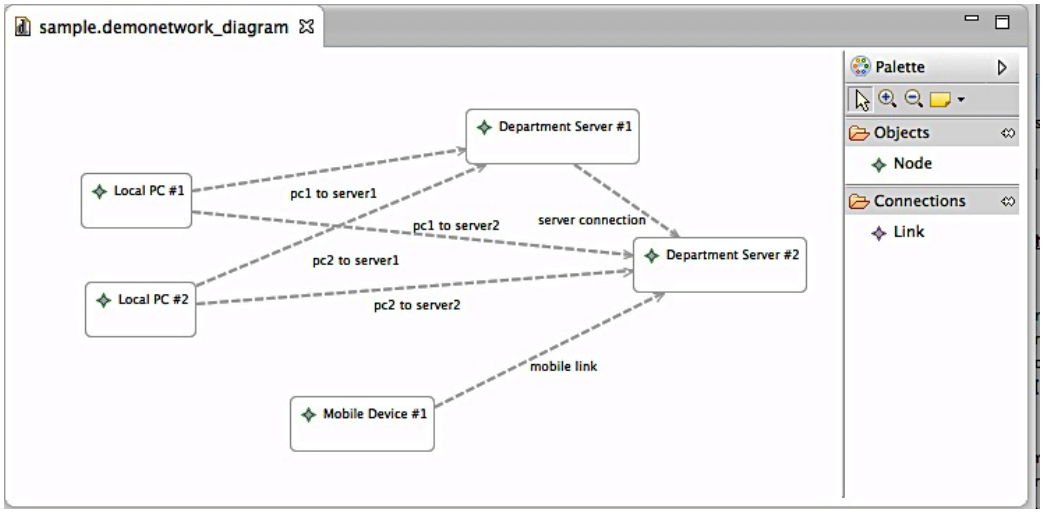

図 4-11 グラフィカル DSL 例

sample.xmi

```
<?xml version="1.0" encoding="UTF-8"?>
<dnet:Network xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:dnet="http://DemoNetwork" name="sample">
  <nodes name="Local PC #1"/>
  <nodes name="Local PC #2"/>
  <nodes name="Department Server #1"/>
  <nodes name="Mobile Device #1"/>
  <nodes name="Department Server #2"/>
  <links name="pc1 to server1" source="//@nodes.0" target="//@nodes.2"/>
  <links name="pc1 to server2" source="//@nodes.0" target="//@nodes.4"/>
  <links name="pc2 to server1" source="//@nodes.1" target="//@nodes.2"/>
  <links name="pc2 to server2" source="//@nodes.1" target="//@nodes.4"/>
  <links name="mobile link" source="//@nodes.3" target="//@nodes.4"/>
</dnet:Network>
```

図 4-12 グラフィカル DSL モデルデータ例

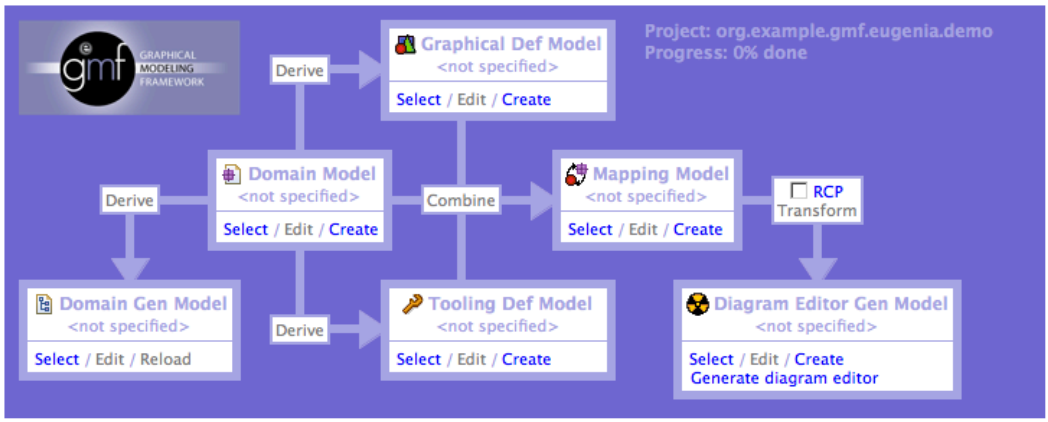

図 4-13 GMF ダッシュボード

(2) Textual DSL

同様に External DSL の一種、すなわち母体となるプログラミング言語は持たないものであるが、UML ツールのようなグラフィカルエディタではなく、テキストベースのエディタを生成する。代表的なものが Eclipse の Xtext[56-58]で、メタモデル相当部分は作成したい DSL の文法を拡張 BNF 形式に近い形式で定義し、これに基づきテキストベースのエディタを生成する。Eclipse プラットフォームの IDE 支援機能であるカラーリングやシンタクスチェックなど DSL モデルを記述する際に助けとなる種々の機能が組み込まれる。中間生成物として文法に相当する Ecore ファイルも作成され、これをグラフィカルモデルに変換表示し視覚的に文法の内容を確認することも出来る。また、エディタを作成するワークフローにはモデル変換用のテンプレートが含まれている。次の図 4-14 は文法定義例である。

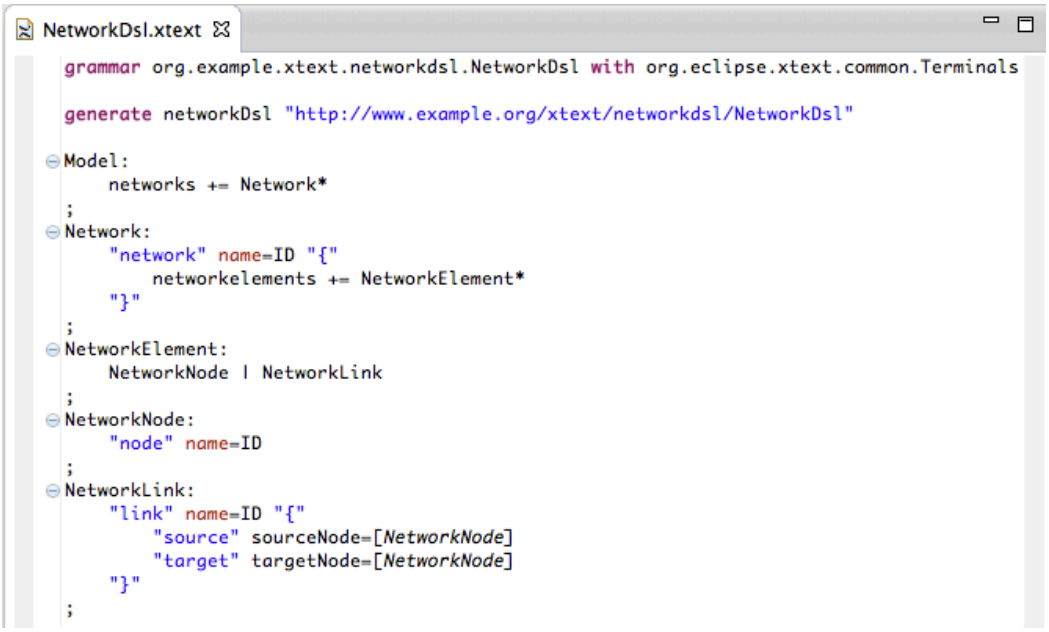

図 4-14 DSL 文法定義例

この文法定義に基づき生成される Ecore ファイル例を次の図 4-15 で示し、この文法に基づき生成されるエディタでモデル作成をする場合の一例を図 4-16 で示す。

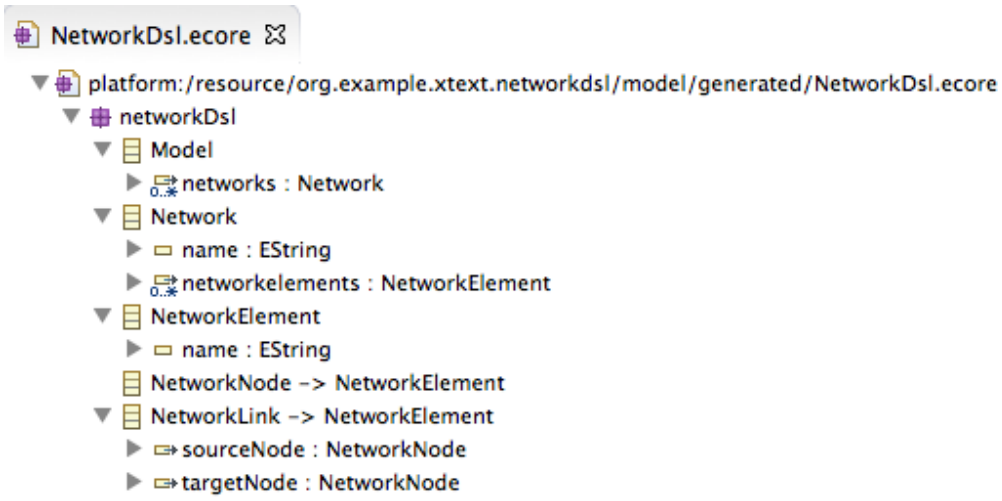

図 4-15 DSL 文法定義から生成された Ecore 例

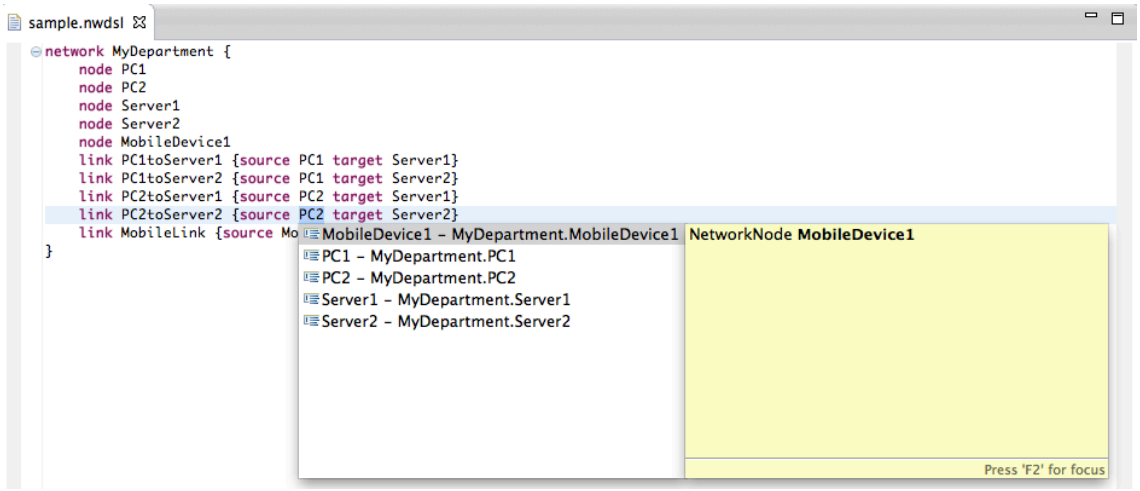

図 4-16 DSL エディタ例

このようにして作成したモデルのデータ構造を次の図 4-17 と図 4-18 で示す。図 4-17 は木構造エディタによるビューで、図 4-18 はモデルを XML 文書としてシリアライズしたデータのテキストとしてのビューである。

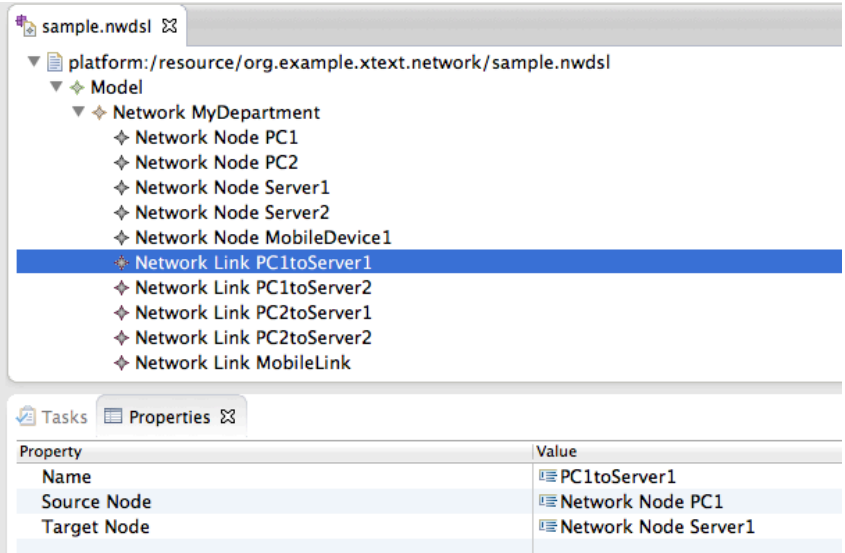

図 4-17 DSL エディタで作成したモデル例（1）

図 4-18 DSL エディタで作成したモデル例（2）

(3) Internal DSL

これは母体となるプログラミング言語を持つ DSL であり、例としては Ruby on Rails がある。テーブル定義と Active Record を使った Class 定義部分がモデルに相当する。Java 系で Ruby on Rails に相当するものとしては Groovy 言語に基づく Grails[92]があり、主にドメインクラス定義部分がモデルに相当する（図 4-19）。Internal DSL では母体となるプログラミング言語の機能を活用するため、短期間で開発出来るというメリットもあるが、最終的にはその言語に限定されたソリューションとなる。

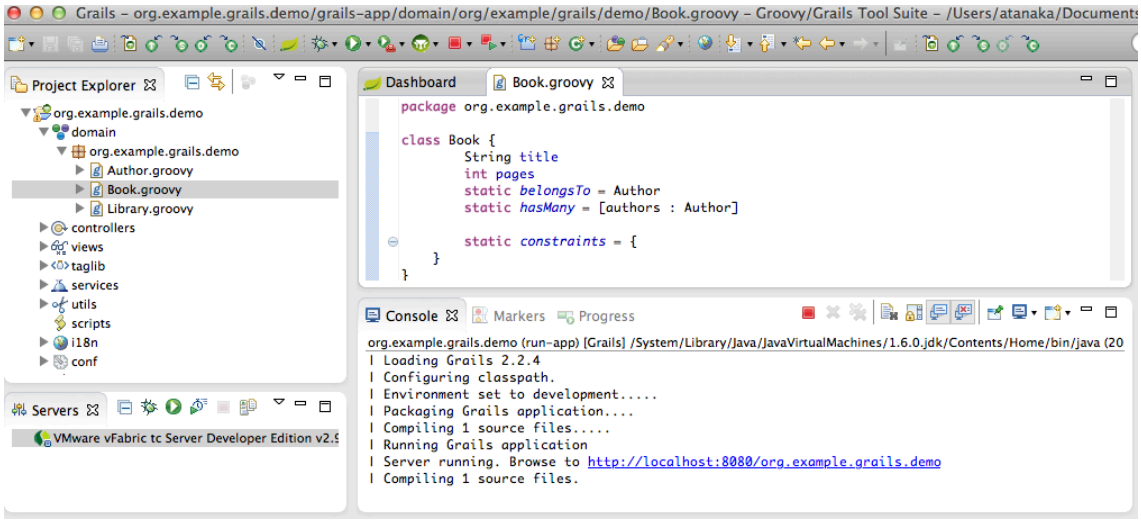

図 4-19 Grails 例

4) 数式モデル

物理現象や社会現象や経済活動などを説明する為に数式モデルが用いられる場合がある。この場合、数学がメタモデル、数式がモデル、観測される値がインスタンスモデル等と解釈することが出来る。数式モデルの特徴は、シミュレーションが比較的容易に実現出来る点にあり、数値計算手法を用い時系列にある値の集合の変化をシミュレートすることなどが出来る。

4.7 モデル変換手法

1) はじめに

モデル変換は、文字通りあるモデルを別のモデルへ変換することであるが、MD*の文脈においては非常に重要な役割を果たす。OMG の MDA[43-45]では、CIM、PIM、PSM の各モデル間の変換と PSM からテキスト(コード)への変換が想定されている。モデルからモデルへの変換ではより詳細またはよりプラットフォーム色の強いモデルへの変換が多く、その場合元のモデルに記述されていない情報を、デフォルトで与える、変換時のパラメタとして指定する、元のモデルにアノテーションとして追記する、といったメカニズムを用い詳細化を行う。モデルが変換出来るということは、変換元のモデルの範囲において変換先モデルの部分と意味的な対応付けが取れるということであり、その為にはそしてモデルレベルでの一貫した変換を実現するには、それぞれのモデルについて高次であるメタモデルレベルでの対応付けが出来る必要がある。

2)　モデルからモデルへの変換

モデルからモデルへの変換についての研究を早期に行った文献[55]によれば、モデルからモデルへの変換メカニズムは次のダイアグラム（図 4-20）で表現出来る。

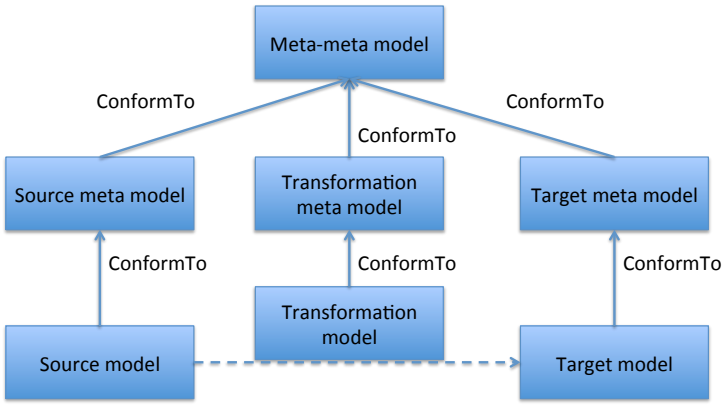

図 4-20 モデル変換概要

モデル変換を目的とした標準には MOF/QVT があり、オープン実装として Eclipse Modeling Project の QVT プラグイン[59][60]がある。また、QVT 標準化時に入力となったプロジェクトも現在 Eclipse ATL[54][55]として同じ Eclipse Modeling Project に存在する。更に、テキスト型 DSL の実装として例にあげた Eclipse Xtext ではモデル変換に使える言語の Xtend 言語が含まれ、またその前身であるテンプレート言語の Xpand[59]も利用可能なコンポーネントとして継続して提供されている。

モデル変換の事例として以下に三つの例を示す。図 4-21 は ATL によるモデル変換記述例、図 4-22 は QVT Operations によるモデル変換記述例、そして図 4-23 は QVT Relation によるモデル変換記述例、である。いずれの場合もアプローチの違いはあるものの、変換前後のメタモデルと変換前モデルを読み込み、変換ロジックを用い、出力モデルに変換結果を埋め込むという仕組みで動作している。

```
1  -- @path Families=/Families2Persons/metamodels/Families.ecore
2  -- @path Persons=/Families2Persons/metamodels/Persons.ecore
3
4  module Families2Persons;
5  create OUT : Persons from IN : Families;
6
7  helper context Families!Member def: familyName : String =
8      if not self.familyFather.oclIsUndefined() then
9          self.familyFather.lastName
10     else
11         if not self.familyMother.oclIsUndefined() then
12             self.familyMother.lastName
13         else
14             if not self.familySon.oclIsUndefined() then
15                 self.familySon.lastName
16             else
17                 self.familyDaughter.lastName
18             endif
19         endif
20     endif;
21
22 helper context Families!Member def: isFemale() : Boolean =
23     if not self.familyMother.oclIsUndefined() then
24         true
25     else
26         if not self.familyDaughter.oclIsUndefined() then
27             true
28         else
29             false
30         endif
31     endif;
32
33 rule Member2Male {
34     from
35         s : Families!Member (not s.isFemale())
36     to
37         t : Persons!Male (
38             fullName <- s.firstName + ' ' + s.familyName
39         )
40 }
41
42 rule Member2Female {
43     from
44         s : Families!Member (s.isFemale())
45     to
46         t : Persons!Female (
47             fullName <- s.firstName + ' ' + s.familyName
48         )
49 }
```

図 4-21 ATL 定義例

```
modeltype Families uses Families('http://families');
modeltype Persons uses Persons('http://persons');

transformation Family2PersonQVTo(in fm : Families, out Persons);

main() {
    fm.objects()[Member]->map members2males();
    fm.objects()[Member]->map members2females();
}
mapping Member::members2males() : Male when {not self.isFemale()}
{
    fullName := self.firstName + ' ' + self.familyName();
}
mapping Member::members2females() : Female when {self.isFemale()}
{
    fullName := self.firstName + ' ' + self.familyName();
}
query Families::Member::familyName() : String {
    return (
        if not self.familyFather.oclIsUndefined() then
              self.familyFather.lastName
        else
            if not self.familyMother.oclIsUndefined() then
                 self.familyMother.lastName
            else
                if not self.familySon.oclIsUndefined() then
                     self.familySon.lastName
                else
                     self.familyDaughter.lastName
                endif
            endif
        endif
    )
}
query Families::Member::isFemale() : Boolean {
    return (
        if not self.familyMother.oclIsUndefined() then
            true
        else
            if not self.familyDaughter.oclIsUndefined() then
                true
            else
                false
            endif
        endif
    )
}
```

図 4-22 QVTo 定義例

```
transformation Families2Persons(IN:Families, OUT:Persons) {

    top relation Member2Male {
        fn : String;
        checkonly domain IN member:Member {
            firstName = fn
        };
        enforce domain OUT male:Male {
            fullName = fn + ' ' + lastName(member)
        };
        when {
            not isFemale(member);
        }
    }
    top relation Member2Female {
        fn : String;
        checkonly domain IN member:Member {
            firstName = fn
        };
        enforce domain OUT female:Female {
            fullName = fn + ' ' + lastName(member)
        };
        when {
            isFemale(member);
        }
    }
    query isFemale(member: Families::Member) : Boolean {
        not member.familyMother.oclIsUndefined() or
        not member.familyDaughter.oclIsUndefined()
    }
    query lastName (member: Families::Member) : String {
        if not member.familyFather.oclIsUndefined() then
            member.familyFather.lastName
        else
            if not member.familyMother.oclIsUndefined() then
                member.familyMother.lastName
            else
                if not member.familySon.oclIsUndefined() then
                    member.familySon.lastName
                else
                    member.familyDaughter.lastName
                endif
            endif
        endif
    }
}
```

図 4-23 QVTr 定義例

モデルデータは XMI 形式の XML 文書として保存されるため、XML 文書間の変換として捉えることも可能であり、その意味で低レベルの処理になるが、モデル変換つまり XML 文書変換を XSLT で行うことも出来る。

3)　モデルからテキストへの変換

モデルからテキストへの変換も OMG で MOF2Text 仕様として標準化されており、オープン実装として Eclipse Acceleo[60]がある。機能的には、ベースとなる UML モデルに従う XMI 形式のモデルデータが入力となり、これを解析フィルタリングし任意のノードに移動し、そこへテンプレートを用いてテキストベースの出力を生成するというものである。MOF2Text 以外にも、Xtext ファミリーの Xpand/Xtend や Eclipse で最初から用いられている JET など、入力側のモデルデータの解析さえ出来れば各種テンプレートエンジンを利用してテキスト生成を行うことが可能である。UML モデルを前提にするなら Acceleo の利用例を図 4-24 に示す。

generate.mtl

```
[comment encoding = UTF-8 /]
[module generate('http://www.eclipse.org/uml2/4.0.0/UML')/]

[template public generate(aClass : Class)]
[comment @main /]
[file (aClass.name.concat('.java'), false)]
   public class [aClass.name.toUpperFirst()/] {

   [for (p: Property | aClass.attribute) separator('\n')]
   private [p.type.name/] [p.name/];
   [/for]

   [for (p: Property | aClass.attribute) separator('\n')]
   public [p.type.name/] get[p.name.toUpperFirst()/]() {
       return this.[p.name/];
   }
   [/for]

   [for (o: Operation | aClass.ownedOperation) separator('\n')]
   public [o.type.name/] [o.name/]() {
       // TODO should be implemented
   }
   [/for]

   }
[/file]
[/template]
```

図　4-24 Acceleo 定義例

モデルからモデルへの変換において、双方のモデルがテキスト型の場合がある。そのようなケースでは、モデル変換は実際にはテキスト変換技術を用いて実現することが出来る。報告者はビジネスプロセス（アクティビティ）モデルと SOA モデルの双方につきテキスト型 DSL を定義し、そのインスタンスモデル間の変換をテキスト変換の Xpand で行い相互変換が可能な領域を探る実験を行い報告した。次の図 4-25 は、ビジネスプロセスのモデル SOA モデルに変換[93]する際に用いたテンプレートの一部を示している。

```
«IMPORT jp::ac::fun::xtext::process::processDsl»
«EXTENSION templates::Extensions»
«REM» SoaML Service Architecture «ENDREM»
«DEFINE main FOR Process-»
«FILE this.name+".soa"-»
ServiceArchitecture «this.name» {
  «FOREACH this.lanes AS lane-»
    «FOREACH lane.processFlows.typeSelect(Step) AS node-»
    «IF node.fromNode.fnode.eContainer()==node.eContainer()-»
    «ELSE-»
      «IF node.fromNode.isArtifactProvidedFrom()-»
  ServiceInterface «node.fromNode.fnode.name» {
  }
      «ENDIF-»
    «ENDIF-»
    «ENDFOREACH-»
  «ENDFOREACH-»
  «FOREACH this.lanes AS lane»«IF lane.name=="Artifacts"»
    «ELSE-»
  Participant «lane.name» {
  }
    «ENDIF-»
  «ENDFOREACH-»
  «FOREACH this.lanes AS lane»
    «IF lane.name=="Artifacts"-»
      «FOREACH lane.processFlows AS msg-»
  Message «msg.name»
      «ENDFOREACH-»
    «ENDIF-»
  «ENDFOREACH-»
  «FOREACH this.lanes AS lane-»
    «FOREACH lane.processFlows.typeSelect(Step) AS node-»
    «IF node.toNode.tnode.eContainer()==node.eContainer()-»
    «ELSE-»
      «IF node.toNode.isArtifactProvided()-»
  ServiceContract «node.name» "" among {
    consumer «node.eContainer().getContainerName()»
      AsConsumer
      performed by «node.eContainer().getContainerName()»
    provider «node.toNode.tnode.eContainer().getContainerName()»
      AsProvider
      performed by «node.toNode.tnode.eContainer().getContainerName()»
```

図 4-25 Xpand 定義例

4) モデリング技術活用提案

従来モデリング技術は比較的特殊な領域に属するもので、比較的受け入れられた UML ツール以外はアクセスが困難であった。しかし、オープンソースの普及特に Eclipse Modeling Project とその関連プロジェクトの進展により、一般の利用者が種々のモデリングツールにアクセス出来るようになってきた。また一部ではモデル変換をサービスとして提供するところも現れている[46]。これにより、MD*のツールチェーンを検討することが現実的な課題となってきた。ここまで述べたモデルを中心に変換を重ねる場合のプロセスは概略次のようになる。

(1) モデリング言語を定義するという意味で、メタモデル作成ないし文法定義

(2) メタモデルの場合、基本的には UML Profile ルートとグラフィカル DSL ルート

の選択

(3) 文法定義の場合、テキスト型 DSL ルート

(4) UML Profile の場合、利用者による Profile 定義後、UML モデルエディタへ組み込む

(5) グラフィカル DSL の場合、利用者によるグラフィカルエディタ生成ツール（GMF, GMF+epsilon, MetaEdit+など）の利用

(6) 上記いずれの場合も、Ecore ファイルの保存とモデルの XMI ファイルの保存

(7) モデルからモデルへの変換が必要な場合、Eclipse QVT や ATL などのツールを用いモデル変換を行い XMI ファイルとして保存

(8) モデルからテキストへの変換では、Eclipse Acceleo や Xtend などのツールを用いテキストファイルを作成

このような MD*とは異なり、Trygve M. H. Reenskaug 教授の UML Virtual Machine 研究[95]のようなアプローチも存在する。主な違いはモデルの種類とターゲットとするプラットフォームにある。UML Virtual Machine 研究ではモデルは UML モデルであり、プラットフォームはプログラミング言語（例えば Java）の VM 相当である。これに対して MD*のモデルは UML モデルに限定されず、またターゲットプラットフォームも Web アプリケーションのためのインフラやモバイル機器用 OS（例えば Android Dalvik VM）である。また VM になると、Java VM の JIT コンパイラのような実行時コンパイラが普及する可能性もある。

4.8 まとめ

本章ではモデリング技術について概説した。モデル階層の説明を行い、モデリング言語の種類や標準化動向をのべ、モデル駆動開発という考え方を説明した。そしてモデル記述手法や変換手法として UML や MOF の説明とともに Eclipse で実装が進められている各種ドメイン特化言語ツールやテキスト変換ツールについて紹介した。

第5章　　モデリング技術活用方法についての分析

モデリング技術を実際に適用するにあたり、各段階で検討が必要となる事柄につき分析を行い、問題領域にあったモデリング手法についての提案を行う。また、モデルベース開発の場合の開発環境（ツール）と実行環境について現状を整理する。具体例として、先に述べたエンタプライズアーキテクチャの一つの RM-ODP を利用する。

5.1 メタモデリング技術

(1) 概要

メタモデルを作成するためには、対象ドメインの分析を行い、そのドメインで本質的な役割を果たす概念、関連を抽出することが必要である。この作業はメタモデル設計者が、そのドメインの教科書的書籍等の活用や、対象ドメインを良く知る専門家からのレクチャーと質疑、などインプットを受け様々な角度から整理を行った後に MOF モデルないし DSL 文法定義の形で表現する。実際には、最初スケッチ的なモデルから入り、検証と改訂を繰り返し、最終的には当該ドメイン専門家に確認を受ける。

RM-ODP の場合、標準として自然言語で記述された文書があり、概念の意味定義の他におおまかな概念モデルが含まれている。

メタモデル作成時に利用出来るツールには、OCL[97]を含む UML（クラス図作成）ツール、EMOF（Ecore）エディタ、オントロジ定義ツール、DSL 文法定義ツール[98]などがある。Eclipse EMF 環境との相互運用性があれば、多くの種類のツールが利用出来、データ交換の可能性が高いことから、別々の手法で作業を進めたとしても最終的に Eclipse 環境などでそういった部分的メタモデルを統合することも出来る。

(2) 分析

対象のドメインに現れる特有な概念の集合を分析し、概念の識別、概念に付随する属性の識別、概念に付随する他概念への参照の識別、参照の多重度、などをリストアップすることが必要である。最初の一歩としては、ドメイン分析でよく行われる名詞や動詞の抽出が利用出来る。また、抽出したあるもの（名詞）が概念なのか概念に付随する属性なのか迷うケースが出てくるが、それが更に属性を持つかという観点で判断する。また概念間の関連については、概念のマトリクスを作成し漏れを避ける必要がある。

(3) RM-ODP

RM-ODP は数多くの概念を定義しており、それ自身が個別標準を開発する際に参照されるべきモデルという位置づけにある。従って、それら定義された概念が個別標準や個別モデルのベース概念となるため意味的にはメタモデルと捉えることが出来る。

但し、当初から自然言語で記述されてきたため、まだ完全な RM-ODP の MOF モデルは作成されていない。

(4) ツール類

メタモデルを作成するツールには次のような各種のものがある。

UML ツール（MagicDraw or Eclipse Papyrus）

UML ツールをメタモデル作成に使う場合、利用出来るのは Class 図のサブセットと OCL 程度となる。メタモデルはそのままの形でモデル作成に使えないため、UML にマッピングするための仕様である UML Profile を定義・実装する必要がある。

Ecore エディタ（Diagram Editor for Ecore）

Ecore ファイルを作成する方法は幾つも提供されているが、比較的容易に作成出来るものがグラフィカルエディタの Diagram Editor for Ecore である。UML ツールの Class 図のサブセットに相当する部分を GMF により実装したもので、グラフィカルにモデル図を作成し保存すると、Ecore ファイルが作成・更新される。

GMF（EuGENia, Sirius）

メタモデル作成については Diagram Editor for Ecore が使われることが多く、GMF は Ecore モデルを入力としたグラフィカルエディタ作成ツールである。

DSL 定義ツール（Xtext）

メタモデル作成は拡張 BNF を使い DSL の文法定義という形で行う。作成した文法を解析し中間生成物の一つとして Ecore ファイルも生成する。

(5) 課題

メタモデル作成だけを目的とした場合、どの手段をとっても大差ないが、メタモデルだけでは次に進めない。下流工程のツールチェーンとの連続性を配慮すると出力データは Ecore ファイルそのもの（XMI 形式）、または容易に Ecore ファイルに変換できるものが望ましい。

5.2 モデリング技術

(1) 概要

前項で定義したメタモデルの種類に応じたモデル作成を行う。

UML ツールを使ったメタモデルを作成していた場合、次にそのメタモデルに対応する UML Profile を定義し、その UML Profile を UML ツールに組み込むことでモデル作成を行う。この時、UML の各種モデル要素が同時に利用出来るため、メタモデルでの規定以外の配慮点を含めることが出来る。例えば、Package を用いたグループ化や Profile の対象となっていない UML 図の利用など。

Ecore エディタを用いて Ecore ファイルを作成した場合、モデルの作成は同一 workspace における動的インスタンスによるか、当該 Plugin を組み込んだ Eclipse の別インスタンスを起動することで行う。これらは MOF インスタンスとして XMI 形式で保存することが出来る。

GMF を利用する場合、メタモデル（Ecore ファイル）作成はグラフィカルな Ecore エディタを用いるか、テキスト型の Ecore エディタを用いるかの二通りである。Ecore ファイル作成後は、GMF の定める手順に従いグラフィカルエディタを作成することになる。この手順はかなり手数を要するため、短縮化を実現する支援機能も別プロジェクトとして存在する。

DSL 定義ツールの Xtext を用いた場合、パーサジェネレーションの過程で拡張 BNF 形式の定義データから Ecore ファイルを生成する。モデルの編集は DSL 定義を行った Plugin を組み込んだ Eclipse の別インスタンスで IDE 支援機能を備えたテキストエディタで行える。テキストベースのため 2 次元的広がりはないが、一般的なプログラミング言語のソースコード作成時と類似の形態でモデル作成を行う。モデルデータはメモリ上に展開された木構造を持ち、XMI 形式で保存することが出来る。

ドメインが一般業務分野の場合、ドメイン駆動設計（Domain Driven Design）で提案されている Bounded Context による横方向分割（コンテキストの分割）や Layer による縦方向分割（UI、アプリケーション、ドメイン、インフラストラクチャ）、また Entity や Value Object そして Aggregate Root といったモデルを整理するパターンを使うことで、より実装を意識したモデルにすることが出来る。本来これらはメタモデルとして定義出来るモデル構造化概念であり、UML Profile として定義すれば、モデル作成時に複数の stereotype を適用する形でそれぞれの意味付けを反映することが出来る。

(2) 分析

モデルとメタモデルは「モデルがメタモデルに適合する（conform to）」という関係である。しかし実際には UML では UML Profile のメカニズムを利用するため「stereotype を与えられたモデル要素はそのベースのメタモデル要素に適合する」関係となり、UML モデル作成時に説明用などに stereotype を持たない通常の UML モデル要素をモデル内に記述することが良くある。そうなると、そのモデルのある部分はメタモデル要素のインスタンス相当だが、それ以外は一般の UML 要素からなるモデルとなる。つまり、このモデルは特定のメタモデルと UML のメタモデルの両方に対して適合していることになる。勿論、モデルに記述する全ての要素が stereotype を持つようなメタモデルと UML Profile を定めればこのような問題は発生しない。これ

に対し、MOF ベースのモデリングでは基本的に MOF モデル（メタモデル）要素のインスタンスしか利用出来ないため、本来の意味での適合関係が実現出来る。関係者の理解を得るためのモデルとして UML のケースにメリットは見られるが、この問題とツールチェーン（モデル変換）を考えた場合、追加した UML 要素が下流工程での複雑さを増加する可能性を含んでいる。

DDD を使う場合、考え方やパターンを付加的に利用すると UML Profile の形態が最も相応しいが、一般的な UML 要素の利用も見込まれるため、結果として上の UML の場合と同じ問題を抱えることになる。

(3) RM-ODP

RM-ODP に基づくモデルを UML で記述出来るようにするため、 UML Profile の国際標準化を行い、その後標準化のメンバが集まり幾つかの UML ツール向けに UML Profile データを作成公開している。

(4) ツール

UML ツール

RM-ODP 用の UML Profile 作成の際に用いたものは商用の UML ツールであり、ツールベンダの協力も得たことで成果物（Profile データやモデルデータ）を公開している。これとは別に研究レベルでオープンソース UML ツールである Eclipse Papyrus を利用した UML Profile 開発も行われた記録がある。モデリング作業の出力は UML Profile 要素を含む UML ファイルであり、データ交換には UML ベースの XMI を用いる。

Ecore エディタ（Diagram Editor for Ecore）

EMF は EMOF 実装であるという立場からは、Ecore エディタはグラフィカルな EMOF エディタであり、メタモデルエディタと言える。これを用いて RM-ODP の概念モデルを記述することが可能であるが、モデルは概念モデルのインスタンスだけとなる。モデリング作業の出力は Ecore ファイルであり、データ交換には Ecore ベースの XMI を用いる。

GMF（EuGENia）

GMF を使う場合と DSL ツールを使う場合、ベースとなるのは EMOF のインスタンスである Ecore ファイルである。余分な記述を追加出来ない点で、下流での複雑さを増加するようなことはない。

DSL 定義ツール（Xtext）

テキスト型の DSL 開発フレームワークで、概念記述をメタモデルではなく文法定義で行い、テキスト型のモデルエディタを作成出来る。モデリング作業の出力はテキス

トファイルであり、データ交換には Ecore ベースの XMI を用いる。

(5) 課題

UML Profile を適用した UML モデルを作成する場合、モデルの中には UML Profile を適用した部分とそうでない部分が混在することがあり、事前に「stereotype を適用した UML モデル要素と適用しなかった UML モデル要素」を明確にしておく必要がある。これは、stereotype 適用の有無にかかわらず、モデルデータ全体がモデル変換の入力となるためである。MOF モデルのインスタンスモデルの場合、基本的にはその MOF モデルを反映したモデル要素しか現れないが、他の MOF モデルなどの Import があり実際にインスタンスが使われる場合は、次のステップの為にそれらを明確にしておく必要がある。

メタモデルの定義ファイルを何らかの形で直接利用しモデル作成が出来るのは、Eclipse のモデリングツール系であり、UML ツールの場合は UML Profile の定義とプロジェクトへの組み込みが必要であり直接 MOF ファイルを利用出来ないのが現状である。しかし、普及しているのはむしろ UML ツールの方であり、双方を無事下流に流す仕組みが必要である。下流に渡されるデータは、Eclipse モデリングツール系の場合 Ecore ファイルをメタモデルとする XMI ファイル、UML の場合 UML ツールの Export 機能を用い UML メタモデルに適合した XMI ファイルとなる。

5.3 モデルからモデルへの変換技術

(1) 概要

UML または EMOF の XMI 形式文書を入力とし、別のメタモデルに基づくモデルの XMI 表現へと変換する。図 5-1 は図 4-20 に具体的な仕様名を記入したモデル変換概要図である。

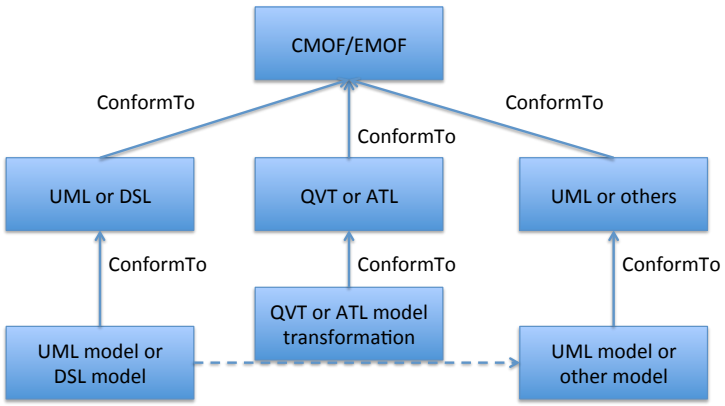

図 5-1 モデル変換概要

基本的な要素としては、変換前モデル、変換前モデルのメタモデル、変換機能、変換機能のメタモデル、変換後モデル、そして変換後モデルのメタモデルである。モデル変換は双方のメタモデルを理解する必要があり、変換ロジック記法としては QVT 標準や ATL の記法などがある。QVT 標準では Operations と Relations という 2 種類の言語を定めている。前者は命令後を並べる Imperative な記述を定め、後者は事前条件、事後条件、など Declarative な記述を中心に関連規則の記述を定めている。

これら以外にも、国立情報学研究所のグループが推進しているモデルをグラフとして変換する方式（Bi-directional Model Transformation）、モデルからテキストへの変換技術を利用する方式、XSLT を用いて XML レベルでの変換にしてしまう方式、更には Java などのソースコードを含める方式も考えられる。

(2) 分析

モデルの表現する意味は各様であるが、モデルを処理する段階では構造化文書としてシリアライズされたデータに対する処理となる。データ交換も考え、これが XML 文書の使われる理由となる。XML 文書の構造定義部分（スキーマ）と本体との関係はモデルとオブジェクトモデルの関係に相当するため、XMI 形式でモデルをシリアライズする場合には、MOF モデルと MOF モデルのインスタンス、つまり Ecore モデルと Ecore モデルのインスタンス、若しくは UML モデルと UML モデルのインスタンス、のいずれかの組み合わせになる。

構造化文書の処理手順は、スキーマに指定されたモデルに対応するリソースをその URI から読み込み、その構造情報に基づき文書の内部を分析する。実装に依存する部分であるが、文書の分析に必要となるインタフェースには高レベルのものや EMF の API を直接使うものもある。

(3) ツール

広く使われているのは前にも述べた Eclipse Modeling Project 系の ATL、QVTo であるが、それ以外にも幾つかの QVTo エンジンがある。他には XSLT ツールも利用されている。

(4) 課題

モデル変換が広く使われるようになるには、確かなオープン実装が必要である。また、上で触れた API やインタフェースのレベルが低すぎると、それも阻害要因となる。

5.4 モデルからテキストへの変換技術

(1) 概要

一般的なコード生成手法[99]とは異なり、メタモデル（Ecore ファイル）規定に適合

するよう作成された XMI 形式のモデルデータをルートからトラバースし、特定の条件に合致した木構造のノードにおいて、テンプレートのテキストとノードに指定した加工を加えた結果の値を埋め込みテキスト形式で書き出すという処理を行う。UML の場合も同様だが、UML 特有の処理として stereotype が指定されているか否かの判断を随所に入れる必要がある。

(2) 分析

モデルからテキストへの変換では、モデルの分析とテキストの生成の二つのフェーズに分けることが出来る。モデルの分析フェーズでは出来る限り EMF の API を隠すことが求められるが、この点については Acceleo や Xtend はほぼ同レベルにある。テキストの生成フェーズでは、テンプレートに埋め込む制御言語の質が問題となる。課題

(3) ツール

標準としては MOF2Text 仕様があるが、Eclipse の Acceleo がこれを実装している。Eclipse の Xtend 言語（旧 Xpand/Xtend 言語の発展したもの）は Java のソースコードをバックグラウンドで動的に生成するのを特徴とするテンプレート機能を含んだ言語である。更に、Eclipse の当初から存在する JET、オープンソースの世界では Apache Velocity や FreeMarker など幾つものテンプレートエンジンが存在し、それらはいずれもモデルからの変換時のテキスト形式書き出し部分に利用出来る。

(4) 課題

モデルからテキストへの変換は基本的にテンプレートを用いる方式になる。そのため、標準への適合性より、テンプレート記述がどれだけ容易か、という点で言語が選ばれる可能性がある。Acceleo や Xtend が現時点で利用者が多いと思われるが、この方向で設計された言語が出てくる可能性も残されている。なお、Xtend の利用する ”«»” 記号は国内では余り使われないため多少使いづらいところがある。

5.5 開発環境

(1) 概要

事前に決定している、または比較検討の上で決定したエンタプライズアーキテクチャに基づく自らのアーキテクチャ設計環境は、特殊な開発ツールではなく一般的な思索支援ツールであり、紙とペン、ホワイトボード、マインドマップ、メール・電話・SNS 的コラボレーション、情報共有メカニズム、リファレンス文書、そしてパターン集のような先人の知恵を生かしながら遂行する思考工程である。

次に、アーキテクチャを形のあるものとして、そして処理及び検索の対象になるも

のとして作成するための開発環境が必要で、メタモデル開発ツール、UML などのモデリングツール、モデル変換ツールなどをシームレスに連携させるモデル開発環境である。この部分で分断があると、一つのツールの出力を次のツールの入力にすることが困難となるため、ばらばらなツールの集合でしかなくなる。これを実現するためには情報交換のための共通モデルデータ形式が必要で、更には交換するデータの耐障害性を高めるため永続化メカニズムの利用が考えられる。このような考え方は欧州の ModelBus プロジェクト[61]で試行されており、そこではサービス指向コンピューティングとリポジトリを組み合わせ、モデルデータのリアルタイム共有と同期を実現している。

メタモデル開発ツールは、UML Class 図のサブセット相当が使われるため、独自のもの、Ecore を使うもの、UML を使うもの、EBNF を使った言語定義によるもの等がある。定まった入力は存在しないが、ほとんどが Ecore モデルを出力ないし export 対象としているため、モデル開発環境で Ecore モデルを出力するステップとして取り扱うことが出来る。

モデル開発ツールは、Ecore のインスタンスモデル、UML モデル、そして DSL（graphical or textual）モデルの定義・編集を行う。このステップの入力は Ecore モデルや UML Profile 定義であり、出力は XMI 文書である。

モデル変換ツールは、モデルデータと XMI の文書構造をトラバースしながら必要な処理を加える。このステップの入力は UML の場合も含めモデルの XMI 文書であり、出力は変換した XMI 文書または生成したテキスト（ソースコード類）である。但し、コード生成に至る前に中間的なモデルが必要となり多段のモデル変換（モデルからモデルへの変換を必要な数だけ行い、最後のモデルをテキストへ変換する）を行う場合もある。

モデル変換ツールの出力であるソースコード類を最後のソフトウェア開発環境に渡すことで、一般的なソフトウェア開発環境へとつながる。この接続が可能になることで、仕様変更時などにコード修正を先に行うのではなくモデル修正を先に行いコードへ展開するという仕組みが実現できる。この環境においても、生成したコードの断片がモデルのどの部分に対応するのか、という参照が必要なことが想定され、ツール側のサポートが期待される。

モデル開発に関連するツール利用の流れの概要を以下の図 5-2 で示す。

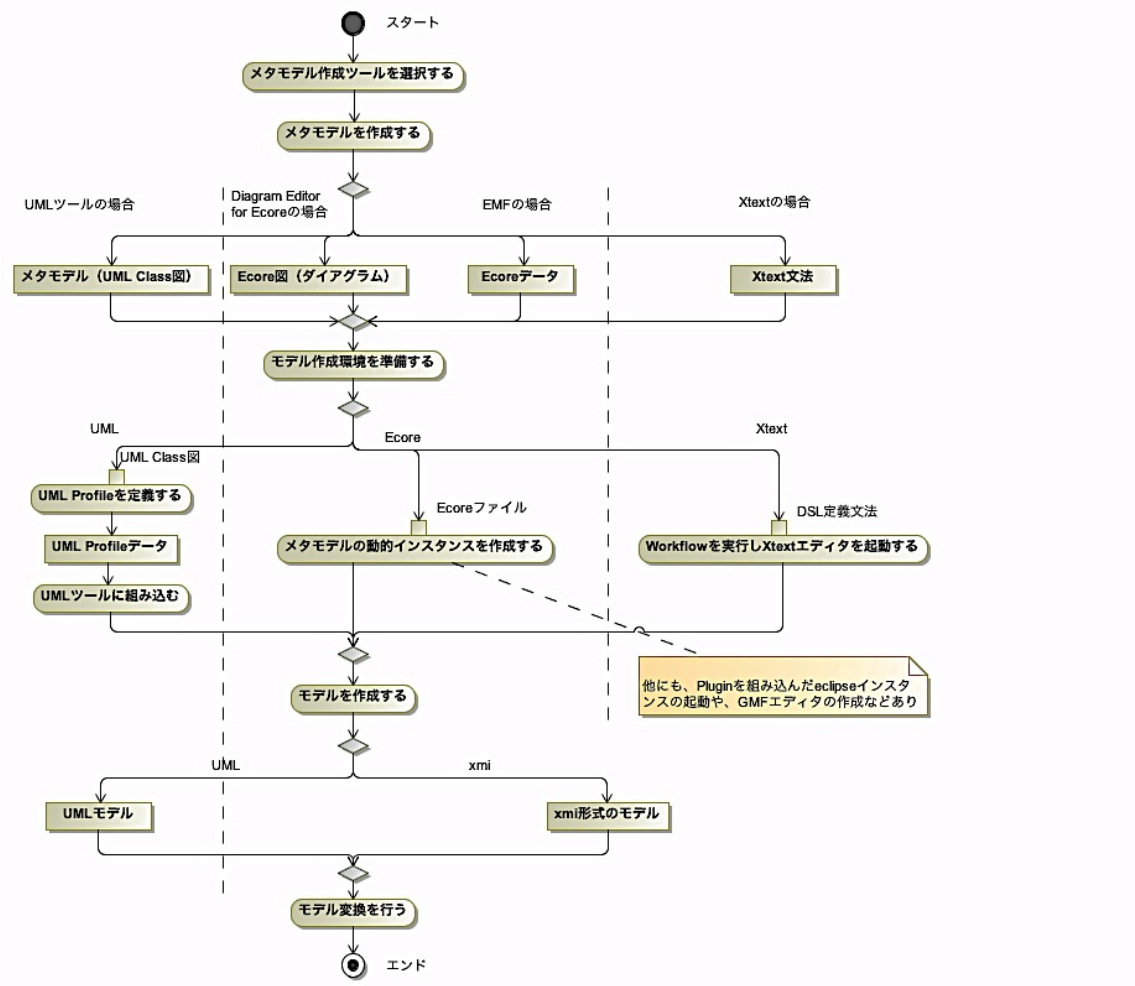

図 5-2 モデル開発に関連するツール利用の流れ

(2) 分析

先に述べた通り、これらのツールは連携出来ることが重要であり、そのためにはツール間のデータ形式に標準的なものを用いること、またツール自体のプラットフォーム間の親和性も重要となる。実装としては特定のベンダーによる全体の実装と、連携を実現した個別実装の集まり、の二つが多い。連携を指向する場合、ツールのプラットフォームを Eclipse にすると Eclipse Modeling Project の各種モデル操作ツールが利用出来る。その場合 Ecore ファイルと XMI データをほぼ標準データ形式として取り扱うことになる。UML ベースのモデル変換も可能であるが、UML ツールの実装状況に依存することもあり、この道を進む場合はそれが利用可能であることを確認の上用いることになる。

(3) 課題

上流から下流へという単一方向での開発はウォーターフォール型の開発である。テ

ストと確認に基づく反復型の開発との融合を考えると、テストデータを与えてモデルを実行し検証する仕組みが必要である。SPIN[100]という手法があるが振舞いについてのモデルだけが対象とされており、構造を記述したモデルの妥当性検証は今後の課題である。モデルからモデルへの変換については双方向モデル変換の研究[96]が進められていること、QVT でもシンプルな逆方向変換の開発は可能であること、等により生成したコードに基づきカスタマイズしたコードのリバースエンジニアリングも課題である。

5.6 実行環境

(1) 概要

ここで言う実行環境は、エンタプライズアーキテクチャに基づくモデルないしはそこから作成されたソースコード、但し本研究ではその型として Web アプリケーションに限定、の実行環境を意味することとする。Web アプリケーションの実行環境を MDA で言うプラットフォームと捉え、PIM から PSM への対応付けを検討することになる。但し、プラットフォームという言葉は用いる人や文脈により大きな幅がある。例えば次のようなプラットフォームのタイプが考えられる。

モデルインタープリタ：例えば UML の VM で、例えばアクティビティや状態遷移を実行ないしシミュレートする形態

アプリケーション寄りミドルウェア：アプリケーションに抽象度の高いインタフェースを提供することで、アプリケーション開発者の記述量が押さえられるもの。アプリケーション開発を楽にすると同時に細かな制御は許さない形態。

OS 寄りミドルウェア：アプリケーションに OS に近いローレベル・インタフェースを提供しアプリケーション製作者に最大限の自由度を与える（アプリケーションの記述量は増大する）形態。

モデル自体の抽象度の高さの違いや、何をアプリケーションと呼ぶかも立場や状況によって違うが、ここではエンタプライズアーキテクチャとして記述されたモデルを入力とし、モデルとプラットフォームのギャップ、つまりモデル変換機能の記述量を考えると、一般的には OS 寄りミドルウェアの場合が最もギャップが大きく、そのためギャップを埋めるモデル変換コードの量は最も多くなる。逆にギャップが小さく、その業務処理がプラットフォームの想定するアプリケーション形態に近い程、その間を埋めるモデル変換コードの記述量は少なくなる。

(2) 分析

モデルからモデルへの変換を繰り返しながらプラットフォームに近づけてゆく場合、

変換処理の中に次のレベルへの詳細化要素が含まれ、ターゲットとするプラットフォームの構造や振舞いを抽象化したものへと段階的に対応付けを行うことになる。これとは別に、コンポーネント化は同じ業務グループに属する複数のエンティティやないし責務をまとめることで、少し粒度の大きいコンポーネントを定義し、それにより総合的に全体構造の単純化を図るものである。

(3) 課題

モデル変換では MDA に CIM/PIM/PSM といった概念があるが、最も単純な CIM->PIM->PSM->code という図式で世の中に広まっているのが課題の一つである。極論をすると、CIM^{1}->CIM^{2}->CIM^{3}-> --- >PIM^{1}->PIM^{2}->PIM^{3}-> --- >PSM^{1}->PSM^{2}->PSM^{3}-> --- > code というように各 CIM/PIM/PSM の中でもモデル変換は存在しそれぞれより詳細化したモデルや見方を変えたモデルへと変換が行われる。モデル変換が多く発生するケースでは、変換の妥当性の検証手段（変換に対するテスト手法）や、変換の間の一貫性を確保する方法など課題である。

プラットフォームレベルの定義と分類が必要であるが、絶対評価が出来ず数値化困難である点が課題である。

5.7 エンタプライズアーキテクチャとして RM-ODP を利用した場合のモデリング

サンプルとして病院の予約管理システムを取り上げる。

(1) RM-ODP アーキテクチャ用メタモデル

RM-ODP アーキテクチャの概念定義は RM-ODP 標準で定義されており、そのメタモデルは Use of UML for ODP system specifications 標準に記載がある。同標準では、メタモデルに基づき UML Profile を定義し、RM-ODP に基づくモデルを UML モデルとして記述可能としている。本論文ではモデル変換に Eclipse プラットフォームを用いるため、そこで必要となる Ecore ファイルを作成する目的で EBNF ベースの定義を行う。次の図 5-3 はその定義の先頭部分であり、ベースとなる仕様構造、オブジェクト構造、振る舞い記述に関するものである。

```
Model:
    (elements+=ModelElement)* ;
ModelElement :
    Type | Object | Spec ;
Type:
    SimpleType | Enumeration ;
SimpleType:
    'type' name=ID ('extends' superType=[SimpleType])? ('{'
        (properties+=Property)*
    '}')? ;
Enumeration:
    'enum' name=ID '{'
        (valueItems+=STRING)*
    '}' ;
Spec:
    EV_spec | IV_spec | CV_spec | NV_spec | TV_spec | Corr_spec ;
FQN:
    ID ('.' ID)*;
Object:
    ObjectType=('EV_Object'|'IV_Object'|'CV_Object'|'NV_Object'|'TV_Object') name=ID ('extends' superObject=[Object|FQN])? '{'
        (properties+=Property)*
        ('{' stateMachine=StateMachine '}')?
        (interfaces+=Interface)*
        ('<' (innerObjects+=Object)+ '>')?
        ('implementing' nvObject=[Object|FQN])?
    '}' ;

Property:
    'property' name=ID ':' type=[ModelElement|FQN] (many?='[]')? ;

StateMachine:
    'events'
        (events+=Event)+
    'end'
```

図 5-3 EBNF による ODP 定義例（一部）

(2) RM-ODP モデル定義例

まず UML モデルで全体像を示し、構造に関する部分と振舞いに関する部分の代表例を取り上げ、Ecore のインスタンスモデル、そしてテキスト型 DSL による記述の三つの例を示す。

UML モデルは Use of UML for ODP system specifications が標準化した記法に基づき概要を示す。まず図 5-4 で、５つの視点で独立に記述したモデルとそれらの関連付けを行う Correspondence モデルから構成されることを示す。

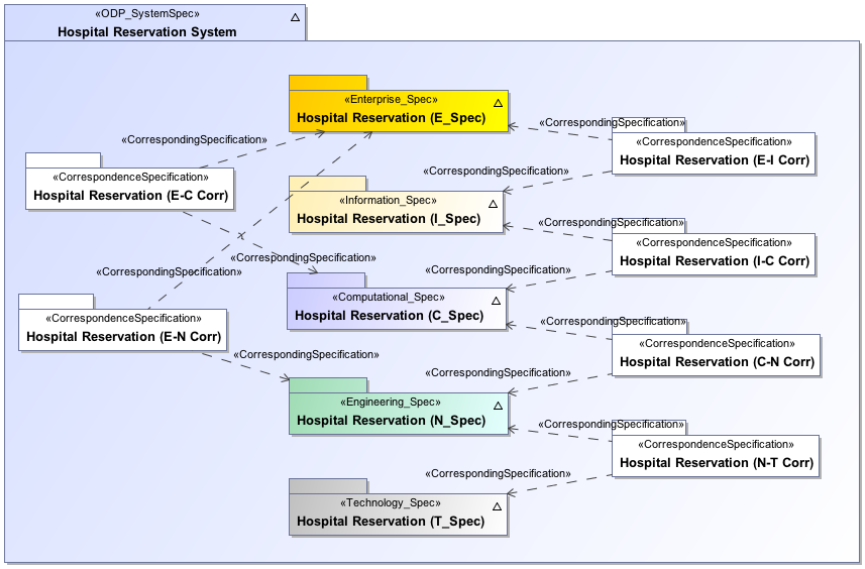

図 5-4 RM-ODP に基づく UML モデル構成例

図 5-4 のなかでエンタプライズ仕様の記述範囲を示すものが次の図である。

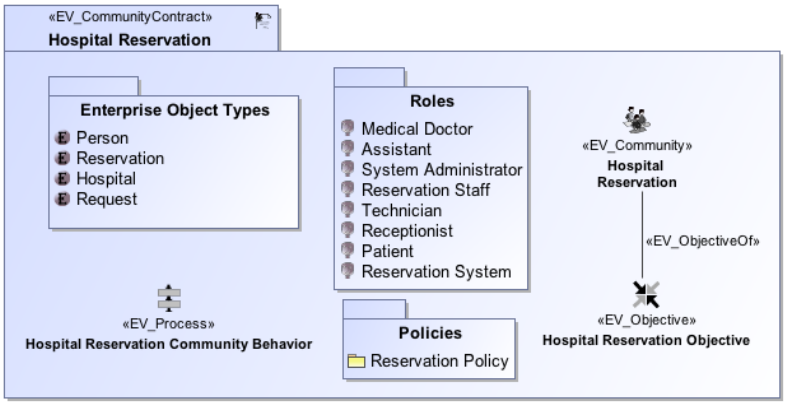

図 5-5 エンタプライズモデル概要図

次にエンタプライズ仕様として図 5-6 にロール定義例、図 5-7 にインタラクション定義例、そして図 5-8 にプロセス定義例を示す。

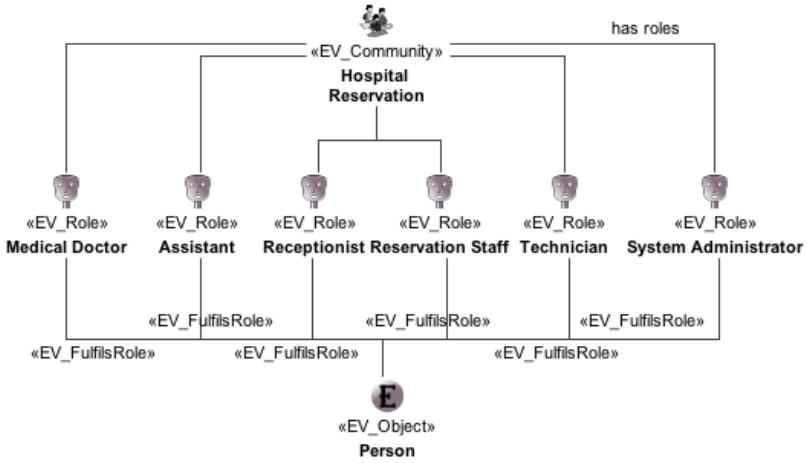

図 5-6 ロール定義例

図 5-6 は病院の予約管理に関わるロール（振る舞いを抽象化したもの）の種類とどのエンタプライズオブジェクトがその振る舞いを実行するかの対応付けをとっている。

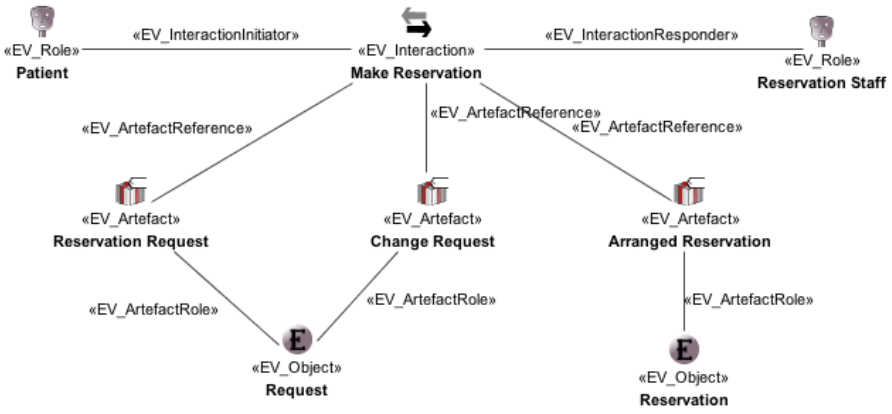

図 5-7 インタラクション定義例

図 5-7 は患者側からのはたらきかけで予約担当スタッフとインタラクションを行い、予約要求、変更要求、確定した予約、などが受け渡されることを示している。

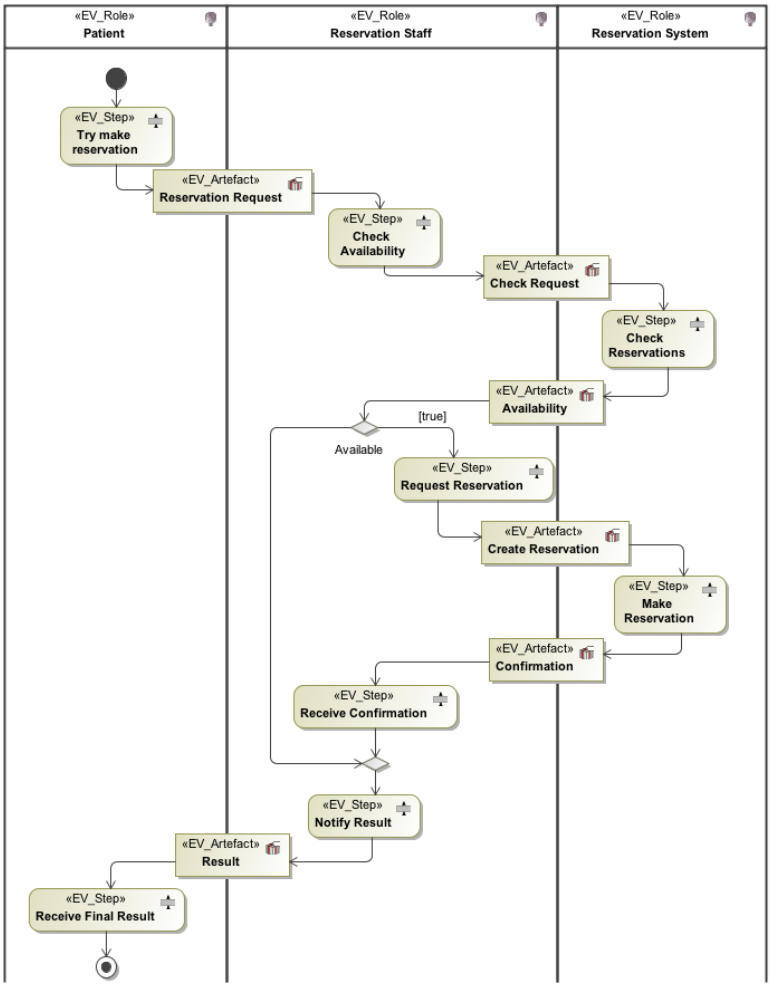

図 5-8 プロセス定義例

図 5-8 は、各ロールがどのような振る舞いを行い、そして予約をとるという仕事を成し遂げるかアクティビティ図ベースで示している。

次に情報の視点では、不変スキーマ定義の例をあげる。

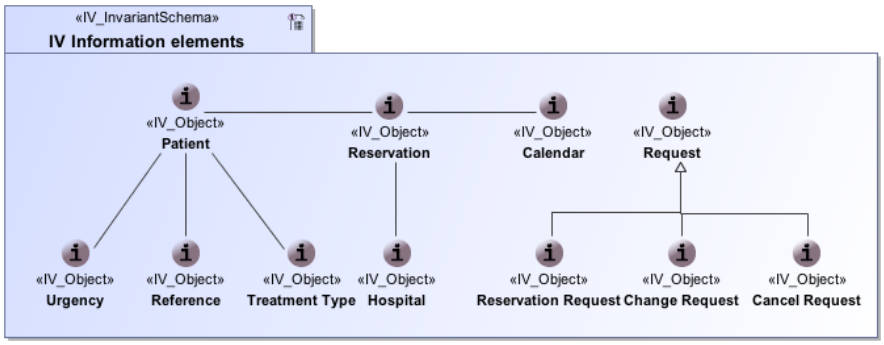

図 5-9 不変スキーマ例

RM-ODP では 3 種類のスキーマを定義する。常にその関係が保たれる不変スキーマ、状態遷移を規定する動的スキーマ、そしてある瞬間にそれらを満たすスナップショット値の集合としての静的スキーマである。図 5 -9 はそのうち不変スキーマの例で、UML で記述するデータモデルに相当する。

コンピュテーショナル視点では物理的な分散を意識せず機能要素に注目する。

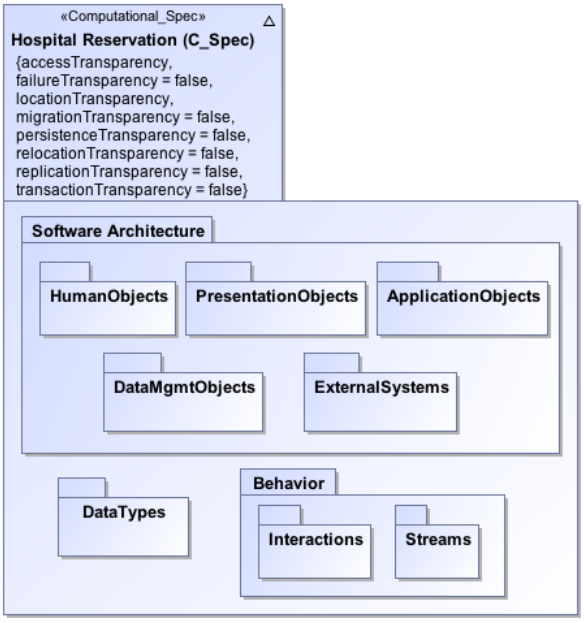

図 5-10 コンピュテーショナルモデル例

図 5 -10 はコンピュテーショナルモデルのなかで全体像を表すもので、どのようなシステムアーキテクチャ要素があり、どのような振る舞いが定義されるか、という関心範囲を示している。詳細は次のエンジニアリング視点でさらに分散を意識した形で提

示することになるため、ここでは構造にとどめる。なお、仕様の属性として多数の Transparency を明確にすることでモデルの前提を示している。

エンジニアリング視点では具体的なエンジニアリング要素やシステム構造を規定する。

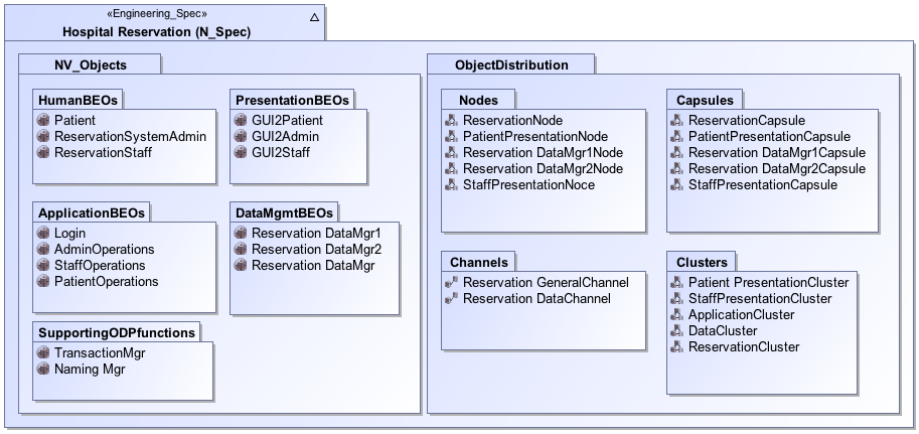

図 5-11 エンジニアリング要素例

図 5-11 はこの視点で必要となるエンジニアリングオブジェクト、ノード、論理的通信路などを列挙している。

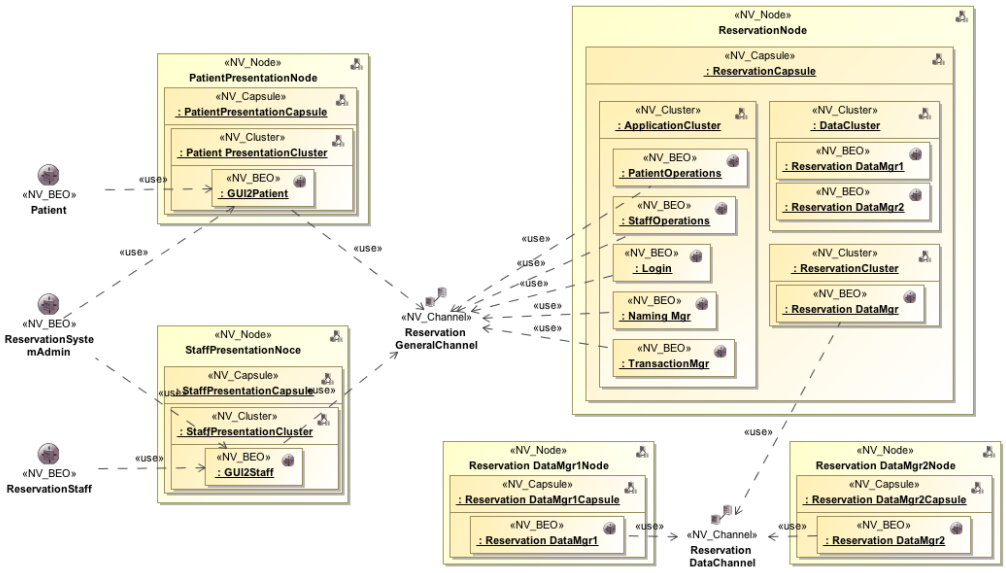

図 5-12 エンジニアリングモデル例

図 5-12 は図 5-11 の要素を用い、患者を代表するオブジェクトがどのようなコンポーネント連携により予約処理を実現するかの概略を示したモデル図である。各要素の振る舞いや要素間のインタラクション記述には UML のアクティビティ図やシーケンス図を用いる。

これら以外にも、具体的な技術要素との対応付けや異なる視点モデル間の関係づけモデルもある。

Ecore インスタンスモデル：

この手法では、RM-ODP のメタモデルを MOF（EMOF）により定義し、そのインスタンスを生成することでモデル作成を行うものである。エンタプライズビューポイントのコア部分につき Ecore モデルを作成した。ビジュアル表現では次のようになる。

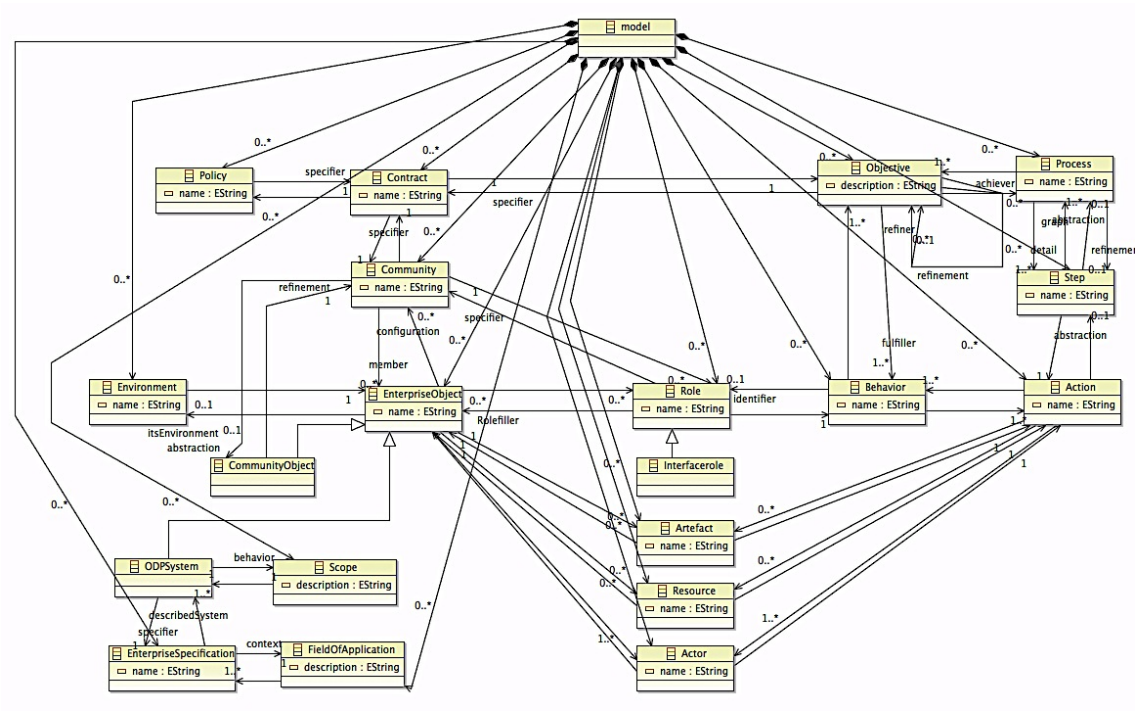

図 5-13 Ecore モデル図

図 5-13 に含まれる概念はエンタプライズビューポイントのシステム概念とコミュニティ概念の二つを統合したものである（Ecore では双方向の関連が存在しないため、この図では片方向の参照を両方向に与えている）。また図 5-13 ではインスタンスモデル作成のため、すべての概念を含む model 要素を追加的に導入している。この Ecore モデルを専用エディタで表示すると次の図のようになる。

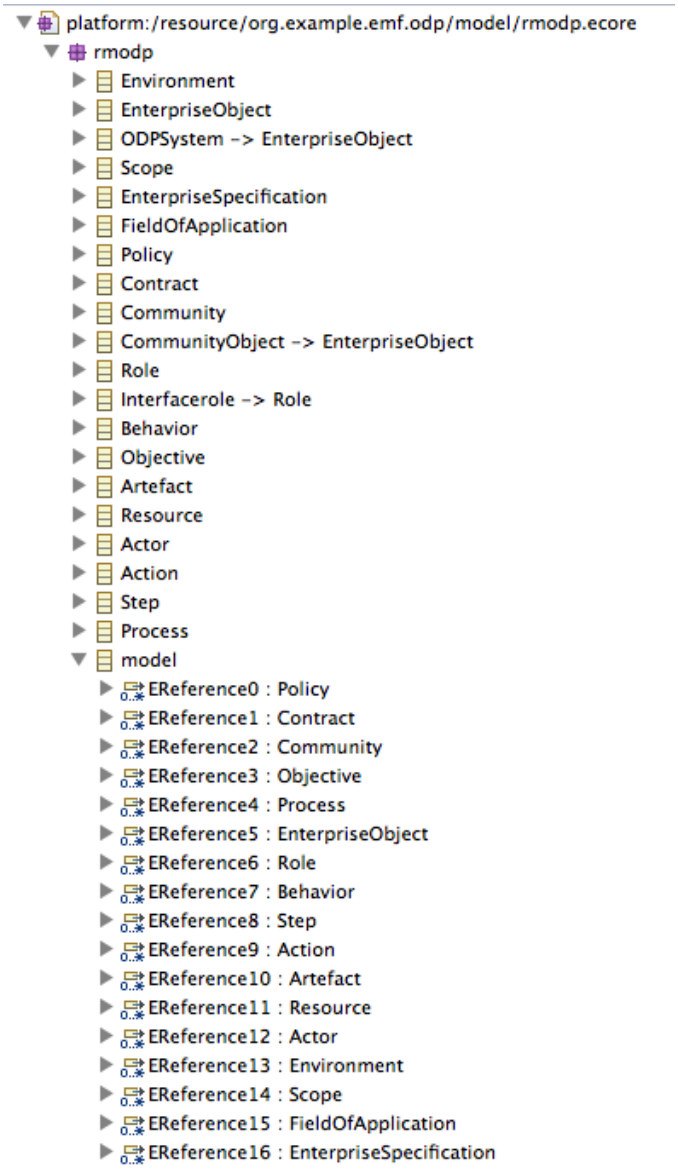

図 5-14 エンタプライズ概念主要部分の Ecore モデル

図 5-14 をベースとし、model 要素から動的インスタンスを生成し図 5-5 に対応するモデルを作成すると次のようになる。

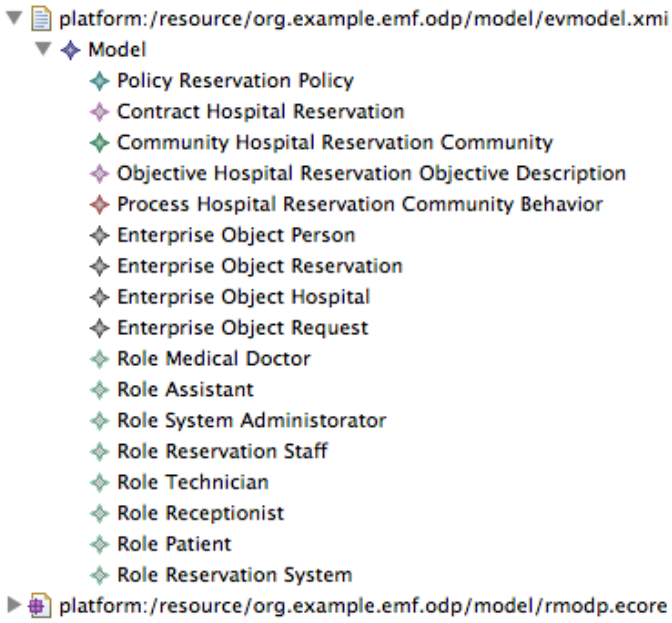

図 5-15 インスタンスモデル例

図 5-15 のモデルはすべての要素がフラットに並ぶもので、図 4-5 との違いは明白である。この違いは図 4-5 が UML Profile に基づき作成されたものであり、UML Profile を使いモデルを作成する場合 Package のような UML の標準的な要素は利用できること、UML Profile 標準化時にメタモデルにない要素を補完的に追加したこと、それに対してメタモデルは概念と関連の定義だけでインスタンス作成時にネスト構造を表現する包含関係も明示的に記述していないこと、などが原因となっている。

次に、図 5-8 のプロセス定義に対応した振る舞いに関するモデルは十分な記述が出来ないことがわかる。例えばアクティビティ図のレーンにはロールを当てはめることになるが、レーンに相当する概念はメタモデル側に存在しない。図 5-8 で記述できているのは UML Profile の定義時に、プロセスをアクティビティに対応付け、ステップをアクションに対応づける、などの UML 概念へのマッピングを行ったことにより、既に定義済みの UML 要素を利用したためである。

テキスト型 DSL モデル：

次の図 5-16 は RM-ODP に基づく Xtext 版モデル記述例の一部を Xtext が生成するエディタで表示したもので、図 5-17 は EMF の Sample Reflective Ecore Modeler を用いその構造を表示したものである。

```
enterprise Clinic_EV {
// Field of application
    EV_FieldOfApplication "To clarify the context in which appointment reservation is executed"
// Global Objects (Business Objects)
    EV_Object AppointmentManagementCommunityObject {
    }
    EV_Object TreatmentCommunityObject {
    }
    EV_Object AccountingCommunityObject {
    }
    EV_Object ClinicODPSystem {
    }
    EV_Object Person {
        property name : String
    }
// Community Specification
    EV_CommunityContract ClinicServiceContract {
// Community's objective (decomposable)
        EV_Objective clinic_objective "Provide fair and timely treatment to all the patient" {
            EV_Objective objective_of_this_year "Make it convenient for patients to visit the clinic"
        }
// Community specification
        EV_Community AppointmentManagement {
// Enterprise Objects with State Machines
            EV_Object Appointment {
                {
                events
                }
            }
            EV_Object IntroducedAppointment extends Appointment {
                property IntroducerID : String
            }
// Roles in the community
            EV_Role ExaminationAssistant
            EV_Role Receptionist
            EV_Role Patient
            EV_Role AppointmentReservationSystem
            EV_Role Doctor
// Start of "AppointmentReservationAbstract" Process Definition
            EV_Process AppointmentReservation {
// Behaviour: each lane of activity diagram representing Process is specified in sequence
// Behaviour (Decision/Action/Subprocess/Artefacts) of Patient
                EV_Role Patient {
                    start StartAppointReservation {
                        outgoing AppointmentRequestForm
                    }
                    EV_Artefact AppointmentRequestForm {
                        incoming StartAppointReservation
                        outgoing ShowReservations
                    }
                    EV_Step ProvidePrintedReservationConfirmation {
                        incoming
```

図 5-16 Xtext エディタを用いたモデル記述例（部分）

図 5-16 はテキストベースでモデル記述が可能であることを示している。UML 表現と比較すると、意味的には同等の内容を記述でき、またソースコードと同じくテキストの世界でコンパクトに設計を進められる可能性がある。しかし要素数が適度な範囲にあればグラフィカル表現の方が直観的に意味を伝え易いかもしれない。

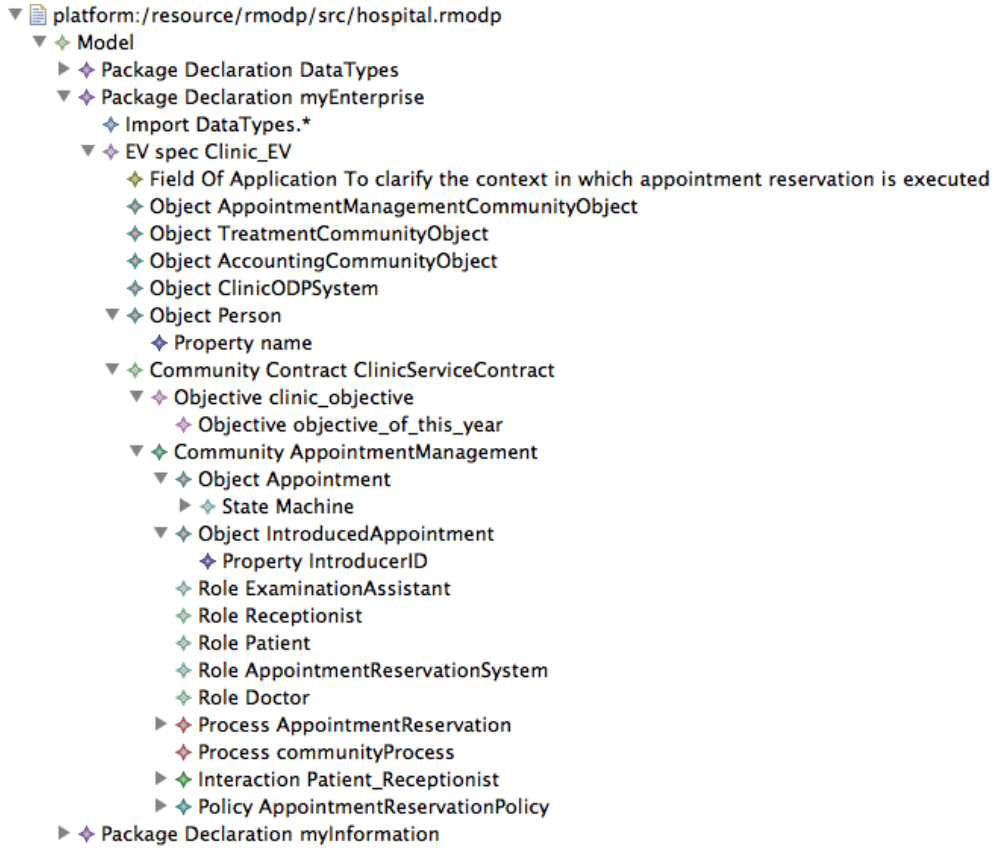

図 5-17 Xtext によるモデル記述例（部分）

図 5-16 および図 5-17 のモデル表現は、前項のインスタンスモデルで表現できなかったネスト構造を文法定義の中に取り込むことにより表現可能となったことを示している。また、振る舞いに関しては状態遷移機械による記述を可能としている。

```
EV_Object Appointment {
    {
    events
        Requested
        NotTaken
        AlreadyTaken
        ModificationRequested
        ModificationPossible
        CancelRequested
        CancelConfirmed
        AppointmentCompleted
    end
    commands
        Create
        Modify
        Cancel
        Complete
        Check
    end
    state initialState
        actions {Create}
        Requested => UnderChecking
    end
    state UnderChecking
        NotTaken => Reserved
        AlreadyTaken => initialState
        ModificationPossible => Reserved
    end
    state Reserved
        actions {Modify Cancel Complete Check}
        ModificationRequested => UnderChecking
        CancelRequested => Cancelling
        AppointmentCompleted => endState
        AlreadyTaken => Reserved
        NotTaken => Reserved
    end
    state Cancelling
        actions {Cancel}
        CancelConfirmed => endState
    end
    state endState
    end
    }
}
```

図 5-18 状態遷移機械による振舞記述例（部分）

このように DSL アプローチも UML 表現と同等の記述能力を持っていることがわかる。ただ、DSL アプローチでは例えば状態遷移機械自体を文法に組み込む必要がある。

(3) モデル変換例

M2T 変換メカニズムを利用したプロセスモデルのサービスモデルへの変換例を、図 5-19 に示す。M2T 変換ではテンプレートを用意しモデルデータから抽出した情報を埋め込むことになるため、具体的な設定方法は違うがほぼ次のような項目を変換にあたり指定する。それらは、変換ロジックを加えたテンプレートファイル、入力モデルファイル、入力モデルファイルのメタモデル（Ecore ファイル）、そして返還後のテキストやコードファイルの出力場所である。

```
«IMPORT jp::ac::fun::xtext::process::processDsl»
«EXTENSION templates::Extensions»
«REM» SoaML Service Architecture «ENDREM»
«DEFINE main FOR Process-»
«FILE this.name+".soa"-»
ServiceArchitecture «this.name» {
    «FOREACH this.lanes AS lane-»
        «FOREACH lane.processFlows.typeSelect(Step) AS node-»
        «IF node.fromNode.fnode.eContainer()==node.eContainer()-»«ELSE-»
        «IF node.fromNode.isArtifactProvidedFrom()-»
    ServiceInterface «node.fromNode.fnode.name» {
    }
        «ENDIF-»«ENDIF-»«ENDFOREACH-»
    «ENDFOREACH-»
    «FOREACH this.lanes AS lane»«IF lane.name=="Artifacts"»«ELSE-»
    Participant «lane.name» {
    }
    «ENDIF-»«ENDFOREACH-»
    «FOREACH this.lanes AS lane»«IF lane.name=="Artifacts"-»
    «FOREACH lane.processFlows AS msg-»
    Message «msg.name»
    «ENDFOREACH-»«ENDIF-»«ENDFOREACH-»
    «FOREACH this.lanes AS lane-»
        «FOREACH lane.processFlows.typeSelect(Step) AS node-»
        «IF node.toNode.tnode.eContainer()==node.eContainer()-»«ELSE-»
        «IF node.toNode.isArtifactProvided()-»
    ServiceContract «node.name» "" among {
        consumer «node.eContainer().getContainerName()»AsConsumer performed by
            «node.eContainer().getContainerName()»
        provider «node.toNode.tnode.eContainer().getContainerName()»AsProvider
            performed by «node.toNode.tnode.eContainer().getContainerName()»
        «node.eContainer().getContainerName()»AsConsumer->«node.toNode.tnode.eContainer()
            .getContainerName()»AsProvider «node.toNode.anode.name»
    }
        «ENDIF-»«ENDIF-»«ENDFOREACH-»
    «ENDFOREACH-»
}
«ENDFILE-»
«ENDDEFINE»
```

図 5-19 Xpand によるプロセスモデル変換例

図 5-19 はプロセス定義を処理しサービス定義に変換するものであるが、木構造のモデルデータから関心の対象となる要素を取り出し、ターゲットの対応箇所に埋め込むという意味で、Java コード生成などの場合と同等のロジックを含んでいる。

5.8 まとめ

本章では、モデリング技術の活用に関して、現時点で何が出来るかツールを含めた現状を説明した。また、UML を使った RM-ODP モデル定義例を説明するとともに、UML 以外の選択肢についても説明した。

第6章　　変化に対応出来るエンタプライズアーキテクチャの提案

堅固であることが特徴のエンタプライズシステムに変化への対応性を組み込む方式について提案する。提案は前章の分析結果をふまえたもので、対象が異なるアーキテクチャをモデリングの段階で親和性を良くする工夫について説明し、３種類の適用事例について解説する。更に、この方式が適用出来る場合と適用が困難な場合について検証する。

6.1 はじめに

Zachman Framework、TOGAF、FEA、そして RM-ODP などエンタプライズアーキテクチャに分類されるフレームワークがある。これらは、Zachman Framework の Perspective、 TOGAF の Architecture Domains、FEA の sub-architecture domain、そして RM-ODP のビューポイントというように、大きく複数の観点でアーキテクチャを分割して捉え、その個々のなかでまた別の観点から分類や構造化を行うという共通のアプローチをとり、規模が大きく複雑な対象物に対し divide and conquer という戦略を適用している。エンタプライズアーキテクチャはその性格上、非常に確かで安定した姿を記述できる。しかしそれはエンタプライズアーキテクチャとして想定した範囲内でのことであり、想定外の変更に対しても安定しているということを保証するものではない。昨今のように新技術が頻繁に開発され利用されると、使える新技術をどれだけ早くエンタプライズシステムに取り込めるかは企業にとって重要な課題である。一般にアーキテクチャは四角形などの図形とそれらを結ぶ線との組み合わせで表現されることが多く、新たな技術要素を追加する場合それを四角形の図形で表現して追加しても既存部分との関連は通常分からない。その点、モデル表現したアーキテクチャであれば、ある程度の詳細化が行われており、インテグレーション時に比較検討等が可能である。また、モデルベース開発が行われていれば、モデルレベルの変更は下流工程に伝達し易い。従って、変化に対応出来るエンタプライズアーキテクチャはモデル表現されたものを前提とする。

モデル表現するためにはメタモデルが必要で、それはエンタプライズアーキテクチャ専用メタモデルということになる。実際のモデリング技術には前章で述べた通り各種の方式があるが、本章ではその中から主に UML を用いた議論を行う。UML をベースに考える場合は、まずクラス図を用いてメタモデルを作成し、次に UML 要素との対応付けを規定し UML Profile とし、その上で UML モデルを作成することになる。UML 以外の場合には、メタモデルのインスタンスモデルや DSL を規定することにつ

ながる。UML ツールベンダによってはZachman Framework 用やTOGAF 用といったUML Profile を提供している場合もある。

6.2 柔軟性要件

最低でも、新技術の統合について検討出来る程度の抽象化と詳細化が、新技術とエンタプライズアーキテクチャの両方について必要となる。エンタプライズアーキテクチャ側はそれ自身がアーキテクチャとしての構造を持つため、その構造化を新技術に適用し比較出来るレベルの抽象化と詳細化が出来ることが要件となる。概念レベルの比較検討だけでなく、例えばコンポーネントレベルの統合検討が出来ることが次の要件となる。

参考になるのがUML の拡張性仕様であり、UML Profile を定義することで既存のモデルに新たな意味付けを持つモデル要素を追加出来る。UML Profile を定義する段階で、本当に既存のUML の範囲内で実現できないかが慎重に吟味される。

6.3 市場動向

エンタプライズアーキテクチャをターゲットとしたUML 他のツールというカテゴリがあり、幾つか有力なものもあるが市場を分け合っているというのが現状である。エンタプライズアーキテクチャの適用は規模が大きいことが多く、推進者（例えばFEA の場合の政府）からすると健全な競争のため複数のツールを入れることが望ましく、そこからツール間のデータ相互運用性の問題も出ており、最終的にはOMG で接続テストの場を用意するようなケースも起きている。

6.4 標準化動向

エンタプライズアーキテクチャの標準化としては、RM-ODP はエンタプライズアーキテクチャとは呼ばず開放型分散処理の参照モデルとして国際標準化されていること、またFEA の一つであるDoDAF とMODAF についてのUML Profile があり、現在ISO/TC 184/SC 4 Industrial data でOMG からのPAS 提案を受け標準化の議論が始まっている（ISO/DPAS 17729 Unified profile for DoDAF and MODAF (UPDM)）という程度である。

新技術の標準化は、その技術が新しければ新しい程なされていないことが多い。

6.5 検討事例

エンタプライズアーキテクチャの概念は20 年以上の歴史があり、当時存在せず現在使われている多くの技術が「新技術」のカテゴリに入る。この研究では特に次の五つ

を取り上げた。

6.5.1 クラウドコンピューティング

クラウドコンピューティングの発想は旧来からあったが、実際に企業活動にも使える技術として認知され利用が広まったのはここ数年である。従来のエンタプライズアーキテクチャになかった要素として以下を例としてあげることが出来る。例えば、SaaS/PaaS/IaaS や Public/Private/Hybrid といった区分は近年のものである。

6.5.2 モバイルコンピューティング

20 年前のモバイル機器はラップトップ機程度でありネットワーク環境も全く異なり、今日の携帯電話やタブレットのような真の移動体はアーキテクチャ要素として想定されていなかった。特に携帯電話は実質小型コンピュータであるにもかかわらず、ほとんどの場所への持ち込みが許されているのはリスクより利点を取ったとしか思えない現実である。携帯の機能では特に位置情報（緯度、経度、高さ、角度、移動速度など）関連機能やセキュリティを始めとする管理面などが注目される。

6.5.3 ソーシャルネットワーク

ソーシャルネットワークはTwitterやFacebookといったコンシューママーケットでの成功が無ければ企業に注目されなかったと思われる技術で、人のつながりがコミュニケーションの活性化や連携を生みだし、結果的に企業内でも活用が進み、ソフトウェア製品に組み込まれる事例が増えて来ている。特徴として、従来関心を持たれなかった人の要素に光を当て、これをシステム化している点があげられる。階層構造が常である企業内組織構造と比べ、非常に柔軟でしかも何らかの運用ルールや役割も存在する。

6.5.4 ビジネスプロセス定義記法

最新技術というものでは無いが、プロセスの中でも特にビジネスプロセス記述に特化した記法である OMG の BPMN（Business Process Model and Notation）は比較的利用されており、業務フローの定義が容易であるとされている。標準だけでなくオープンソースも含めある程度の数のツールが実装している。ISO/IEC 19510:2013 として国際標準にもなっている。

6.5.5 サービス指向モデリング言語

現時点で必ずしも広く使われている言語ではないが、OMG 標準の一つで SOA に基づく業務仕様作成を目的としたモデリング言語であり SoaML（UML Profile for Soa Modeling Language）[76-80]と呼ばれる。エンタプライズアーキテクチャとサービス指向の関係を調べるためここに含める。

6.6 変化に対応するためのモデル統合化手法の提案

(1) 用語の定義

各種の用語を用いて考察を行うため、一旦ここで用語の整理を行う。

- 概念：メタモデルの構成要素でメタクラスとして表現されるもの。名称と意味を持ち、他の概念との間に継承を含む関連を定義できる。
- 概念を集めたセット：あるメタモデルについて、含まれる概念（メタクラス）をメンバとする集合
- 概念定義：概念の意味について自然言語により定義を行うとともにメタモデルのなかで他の概念との関連、関連に関わる多重度、制約などを記述したもの。
- 制約：概念自体や概念間の関連に関しより厳密な定義とするため OCL 言語などを使いモデルのなかで守るべき条件を記述したもの。
- 概念が同一：二つの概念が同じ概念定義を持ち、メタモデルでそれら概念を入れ替えることが出来る関係にあるとき、それら概念は同一とする
- 概念が異なる：二つの概念が異なる概念定義を持ち、メタモデルでそれらを入れ替えることが出来る関係にないとき、それらの概念は異なるとする
- 概念のスーパークラス：概念間に継承の関連がある場合、上位の概念を下位の概念のスーパークラス、下位の概念を上位の概念のサブクラスと呼ぶものとする

(2) 一般論

モデルの統合を行うためには、その前提となるメタモデルレベルでの関連付けが必要である。エンタプライズアーキテクチャのメタモデルと新技術のメタモデルという二つが入力となる。メタモデルは対象とする問題領域においてある抽象レベルで仕様を記述する為に必要な概念群とそれらの間の関連や制約を規定したものであるが、二つのメタモデルの比較においては、まずそれら概念を要素とする二つの集合を考える。それらを、$Sa = \{x \mid x \in MMa\},\ Sb = \{x \mid x \in MMb\}$、ここに$MMa$と$MMb$はメタモデル、$x$はメタモデルを構成するクラス要素（メタクラス要素）とする。この時、二つの集合の関係は、①独立（$Sa \cap Sb = \emptyset$）、②両方に属する要素が存在するも片方がもう一方には含まれることはない（$Sa \cap Sb \neq \emptyset \wedge Sa \setminus Sb \neq \emptyset \wedge Sb \setminus Sa \neq \emptyset$）、③片方がもう一方に含まれる（$Sa \subset Sb \vee Sb \subset Sa$）、のいずれかの関係となる。これらの関係を次の図 6-1

に示す。

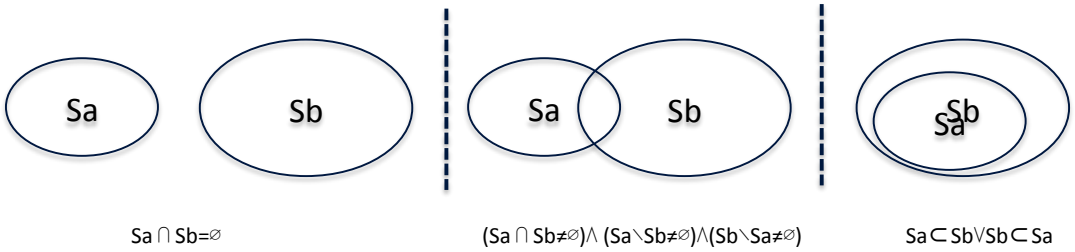

図 6-1　概念要素集合の関係図

上図中央の場合、共通の概念要素が存在することを意味するが、概念名が同じでも別々の意味を持ったり、別名でも同じ概念であったりすることもあるため、慎重な判断が必要となる。それら概念を意味の観点から比較し、二つの概念の関係を決定する。その関係には、別物である、共通の概念の異なる特殊化である（例としては、組織専用のメタモデルと作業者専用のメタモデルに階層構造という共通の概念があったとして、組織と作業者をパーティの特殊化としコンポジットパターンに当てはめることで統合するケース)、片方がもう一方の特殊化である（例としては、二つのメタモデルでワークフローという概念（用語）を用い、一方では抽象度の高いアクティビティ概念を意味し、もう一方ではビジネスプロセス概念や制御プロセス概念を意味していた場合)、同一である、などの分析結果を得ることが目標となる。この評価のための素材は入力とした二つのメタモデルである。

これらを評価しながら、メタモデルでの統合や UML Profile での統合について以降で検討する。

(2) メタモデル

エンタプライズアーキテクチャのメタモデルは、前に述べた通り RM-ODP の場合は標準の中に記載があり、これをそのまま利用する。

新技術に関するメタモデルは、それぞれの技術ごとに文献を調べたり新たに開発したりする。但し、過去の文献における検討結果は参考にすぎず、新技術が実際に企業内で使われる形態を想定して、本質的な概念を中心にメタモデルを作成する。メタモデルに限らず成果物の開発や作成については繰り返し開発の形態を前提とするため、まずコアとなる概念を押さえ、必要に応じ付加的な要素の追加を繰り返し実施する。

(3) 重複領域の解決方法

本研究では、エンタプライズアーキテクチャ側を母体とし不足分を補う形で拡張する、という前提を置く。これは、エンタプライズアーキテクチャの方が対象領域は広

く、また新技術が複数ある場合には必要な拡張項目だけを選択し適用出来る設計が望ましいためである。エンタプライズアーキテクチャのメタモデルを MMe、新技術のメタモデルを MMnt とする。次に各メタモデルの構成要素であるメタクラス要素からなる集合として、$Se = \{x | x \in MMe\}$及び$Snt = \{x | x \in MMnt\}$とすると、図 6-2 で示すような関係となる。

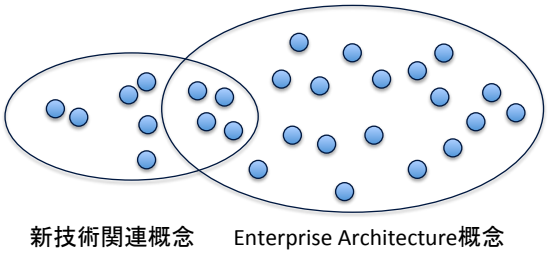

図 6-2 メタモデル要素集合の重複関係

クラス図を用いた例としては次のようなもので、二つのメタモデルの要素間に同一である可能性を見いだした場合を示す。図 6-3 で黒枠内が同一概念候補となる例を示している。

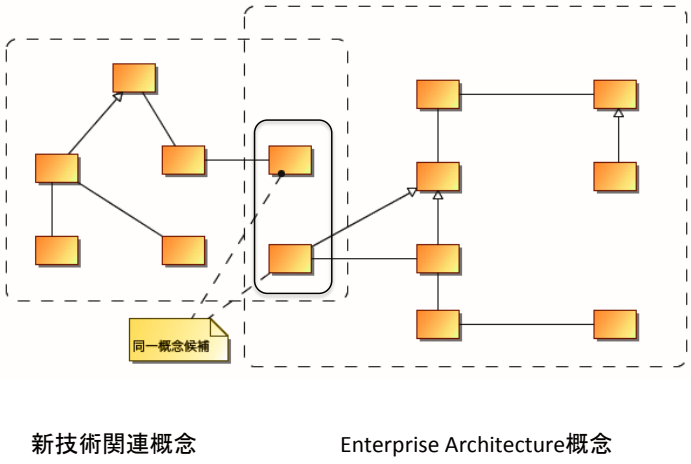

図 6-3 同一概念候補のメタモデル要素例

メタクラスだけからなる集合を考えるのは、最初に対応する概念（メタクラス）の存在を確認するためであり、メタモデル内の関連や制約は概念のコンテキストとして対応性を検証する際に使用する。SeとSntの関係については前述の分類と同様で、①$Se \cap Snt = \emptyset$または②$Se \cap Snt \neq \emptyset \wedge Se \setminus Snt \neq \emptyset \wedge Snt \setminus Se \neq \emptyset$または③$Se \subset Snt \vee Snt \subset Se$となる。

①の場合、新技術はエンタプライズアーキテクチャと全く接点を持たないことを意味する。技術的な観点からは関連付けた統合の対象とならないが、統合することに何らかの価値が見いだされれば、媒介となる概念群を導入し統合が行われる可能性があ

る。本論文では検討の対象外とする。

③の場合、エンタプライズアーキテクチャの中のある問題領域を、同じ概念を用いながら、異なる関連の定義や制約の定義を行い異なる構造の異なるモデル作成につながるものである。これは従来のエンタプライズアーキテクチャのスコープ内の話であり、内部構造の見直しと同じである。本論文では検討の対象外とする。

本論文の主な検討領域は②の分類が適用されるケースである。重複・類似する概念を集めた集合（$Se \cap Snt$）に含まれる概念につき、仮にある概念が関連や制約も含めて両方の世界でほぼ同じ意味を持つと判断できる場合、即ち当該概念の意味と周辺概念との関連や制約の全てにおいて二つのメタクラスは同じと確認できた場合、その概念をつなぎ合わすことでエンタプライズアーキテクチャのメタモデルを新技術対応に拡張することができる。この確認のためには、二つの概念が同一と仮定し接続を行ったり、メタモデルのサンプルのインスタンスモデル（つまりモデル）を作成し接続を行ったりし、その上で周辺の概念も含め検証を行う。確認できれば、二つのメタモデル間に要素の等価性を意味するリンクを確立できる。

上のプロセスの中で問題点に遭遇した場合、二つの概念が同じ名称を与えられていたとしても意味が異なり同一ではないということになる。但し、概念の抽象レベルや粒度の問題が残されている。同一の概念が異なる抽象レベルで使われた場合に互換性がなくなるのは当然であり、一方がもう一方の特殊化（サブクラス）または構成要素になる形での接続の可能性はある。また概念が複数のより基本的な概念の組み合わせからなる複合概念の場合、これを分解しその構成要素レベルで再度比較を行うことも出来る。

本論文では、エンタプライズアーキテクチャが使用されている状態、すなわちエンタプライズアーキテクチャに基づくモデルが既に存在するだけでなく、それに基づいたシステム開発が行われ、そのシステムが日々運用されている、という状態で新たな技術を導入する場合を想定している。そのため、既存システムへの影響を最小限に押さえるためエンタプライズアーキテクチャ側の概念を優先（出来るだけ変更点を少なくする）するという方針をとる。

以降ではより具体的な分析手順を次に示す。ここでは新技術関連概念の A とエンタプライズアーキテクチャ概念の X が同等または類似と判断しインターセクションに置かれたと想定する（図 6-3 では黒枠内）。

ステップ 1

概念 A と概念 X のそれぞれのアーキテクチャにおける意味を比較

定義文レベルの意味比較

周辺概念及びその関連の比較（多重度も含む）

制約があれば比較

なおこれは図 6-3 では黒枠内に置かれた概念間の比較であり、理論的には、独立または無関係、同一、または何らかの関連（通常の多重度付き関連や継承関係など）を持つ、のいずれかとなる。

ステップ 2

二つの概念間の関係が次のどのパターンとなるかを検証する。

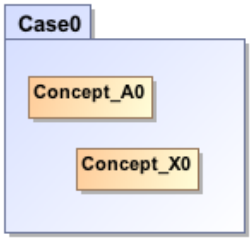

図 6-4 異なる概念であると認定した場合

図 6-4 は、当初同一または類似すると考えた概念が、分析の結果明白に異なることが判定された場合を表す。

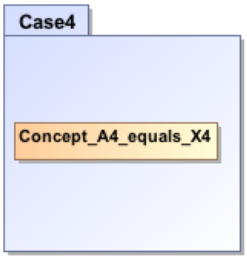

図 6-5 同一の概念と認定出来る（最も単純な）場合

図 6-5 は、当初同一または類似すると考えた概念が、分析の結果同一であると判定された場合を表す。

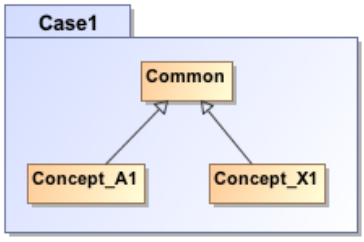

図 6-6 両方の概念に共通のスーパークラスが存在する場合

図 6-6 は、当初同一または類似すると考えた概念が、分析の結果共通の親から派生した共通点を持つが異なる概念であると判定された場合を表す。

異なる対象領域で同じ用語や概念が用いられる場合では、何らかの共通項と対象領域ごとの特殊性を加味したものとに分割出来ることがあり、このような構造を持つことがある。

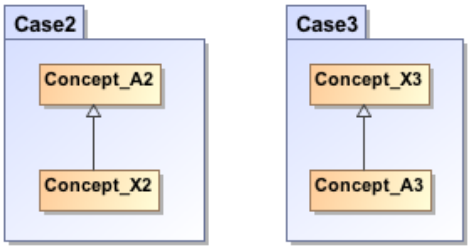

図 6-7 いずれかが他方のサブクラスとなる場合

図 6-7 は、当初同一または類似すると考えた概念が、分析の結果一方が他方を継承する関係にあることが判明した場合を表す。

片方がもう一方の詳細化を行っているような場合に、このような構造が現れる。

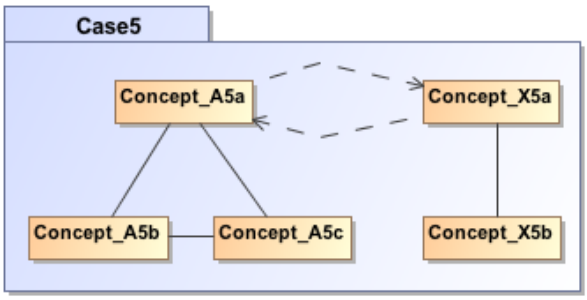

図 6-8 概念の粒度が異なり粒度の大きい概念の分割が必要な場合

図 6-8 の例は、概念 A と概念 X の比較において、概念 A が A5a と A5b と A5c から構成される複合概念 A5 として、また概念 X が X5a と X5b から構成される複合概念 X5 としてそれぞれ分割できた場合を示す。この場合、まずサブ概念要素を構成要素としたモデル（図 6-8 のクラス図）から関連、多重度、制約等を除きサブ概念要素だけを取り出した集合Sa及びSxを考える。図 5-9 は$Sa \cap Sx \neq \emptyset \wedge Sa \setminus Sx \neq \emptyset \wedge Sx \setminus Sa \neq \emptyset$の場合の集合関係を示しており、分割された新技術独自概念の集合は$Sa \setminus Sx$、同等と認められる概念要素の集合は$Sa \cap Sx$、そして分割された Enterprise Architecture 独自概念の集合は$Sx \setminus Sa$である。これは、先に説明したメタモデル要素集合間の比較（図 6-2 及び図 6-3 参照）と類似するが、概念要素を詳細化・分割して生まれる概念群の比較を行うものである。同じアプローチを適用すると図 6-2 に対応した次のような図を描くことが出来る。違いは、双方から同一概念候補を選ぶのではなく「同等と認められる概念要素」を特定した上で他の分割後の概念との関連を明確にすることである。

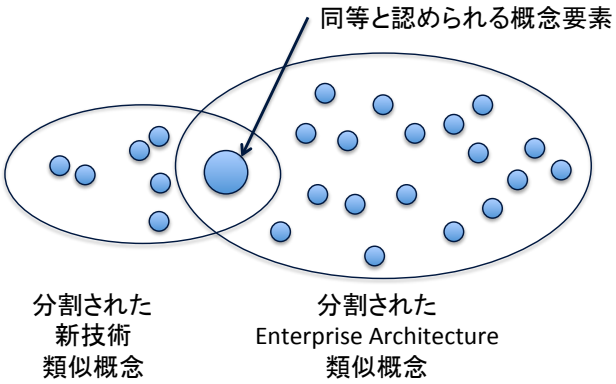

図 6-9 概念の更なる分割

一つのアプローチとして、$Sax = \{x|x \in Sa \cup Sx\}$をメンバ集合とする単一の複合概念（AX）を設けることが考えられる。その概念の持つ意味は、厳密には A5 の持つ意味とも X5 の持つ意味とも異なるが、新技術固有概念群（$Saa = \{x|x \in Sa \setminus Sx\}$に含まれる概念群）がエンタプライズアーキテクチャモデルに影響を与えず、エンタプライズアーキテクチャ固有概念（$Sxx = \{x|x \in Sx \setminus Sa\}$に含まれる概念群）が新技術のモデルに影響を与えない場合には、この複合概念（AX）で既存概念（A 及び X）を置き換えることが可能となる。ただ、これらの検証には時間が必要であり、またエンタプライズアーキテクチャに基づく企業モデル及びその実装に対する変更を最小限にとどめるため可能な範囲で既存概念 X を利用したいことから、既存概念 X を活かす検討を行った。

図 6-8 の場合、サブ概念要素の A5a と X5a を同一とした接続を行うため、結合出来たとして生まれる関連を表示すると次の図 6-10 のようになる。

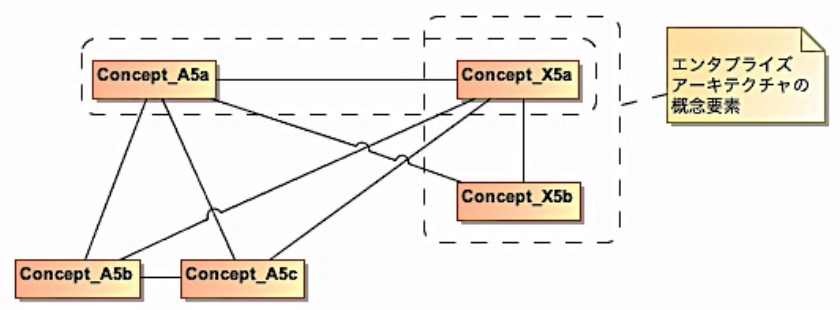

図 6-10 概念要素の同一性による結合例

上図をエンタプライズアーキテクチャ側から記述すると次の図 6-11 のようになる。

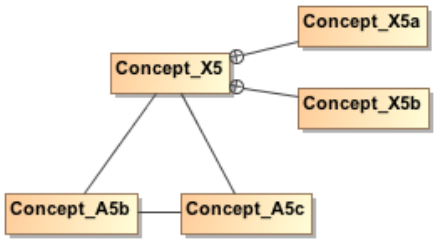

図 6-11 エンタプライズアーキテクチャ側優先の概念定義

ここにConcept_X5 は Concept_X5a (= Concept_A5a) と Concept_X5b の関連づけられた複合概念であり、エンタプライズアーキテクチャのメタモデルに最初から含まれていた概念である。これによるエンタプライズアーキテクチャ側の影響は、Concept_X5 が新たに持つことになった新技術に関する概念との関連であり、これら概念がエンタプライズアーキテクチャ側の既存概念と衝突が無い限り問題は発生しないはずである。なお、衝突が発生する場合というのは、例えば Concept_A5b がエンタプライズアーキテクチャ側の概念と共通点を持っていたという場合であり、それらをメタモデル要素集合のインターセクションに加え、上記手順を繰り返す必要がある。エンタプライズアーキテクチャ側優先の観点からは、上図のような形態の統合が良いものと考える。

逆に新技術側から記述すると次の図 6-12 のようになる。

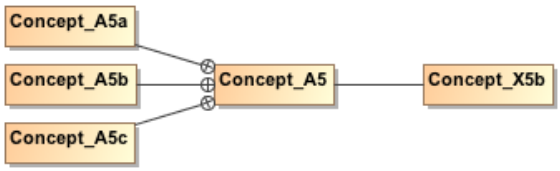

図 6-12 新技術側優先の概念定義

同様に、Concept_A5 は Concept_A5a (=Concept_X5a) と Concept_A5b 及び Concept_A5b の関連付けられた複合概念となり、Concept_A5 が新たに持つことになったエンタプライズアーキテクチャ概念との関連が新技術側の既存概念と衝突しない限り問題は発生しないはずで、衝突については前述同様やり直しが必要となる。

このように、厳密に対応するには同等と認められる概念要素の分割を行い新たなモデル要素を導入することになるため、既存モデルや既存システムに影響が出ないことを慎重に確認する必要がある。

UML の言語仕様書には、UML 言語を定義する目的で Package Merge という Package 間の関係が定義され用いられている。これには適用に幾つもの制約があるものの、二つの Package について共通のモデル要素がある場合その要素を中心に

Package を合成した構造を作り出すことを実現する。また、UML 言語を定義するために用いられていることから、メタモデル定義用のファシリティという位置づけとなり、一般的な UML モデル作成に利用することを目的とした機能ではない。本論文のベースの一つ[94]ではメタモデルの取り扱いにこの Package Merge を利用した。下の図 6-13 は Package A と Package B をマージし Package C を作り出すことを示す利用例である。

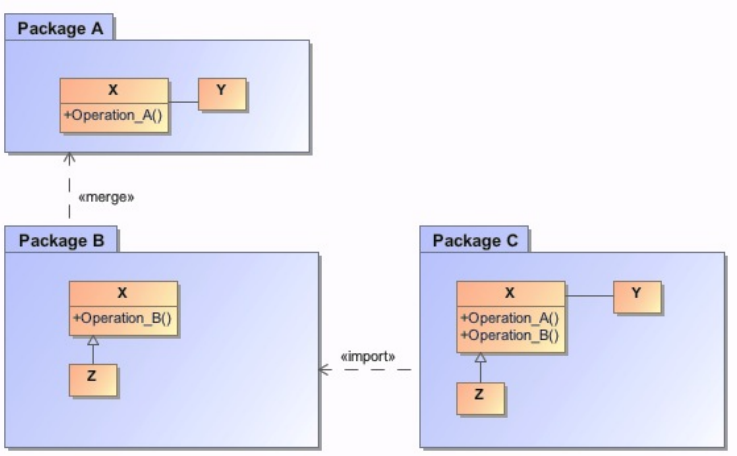

図 6-13 UML Package Merge 利用例

UML の定義（UML メタモデル）において Package を用いた機能分割が多用されたことで必要となった機能である。これが利用できると、次の Case6 のような構造を比較的容易に導き出すことが出来るが、UML 標準仕様に規定があるもののツールでの実行時マージはまだ研究段階にあるようである。なお、メタモデルの標準言語は UML のインフラストラクチャが使われており、Package、Class、OCL などが利用できるとともに、UML インフラストラクチャ仕様定義のために Package Merge が多用されており、本論文での複合概念メタモデルのマージ記述にも利用できる。

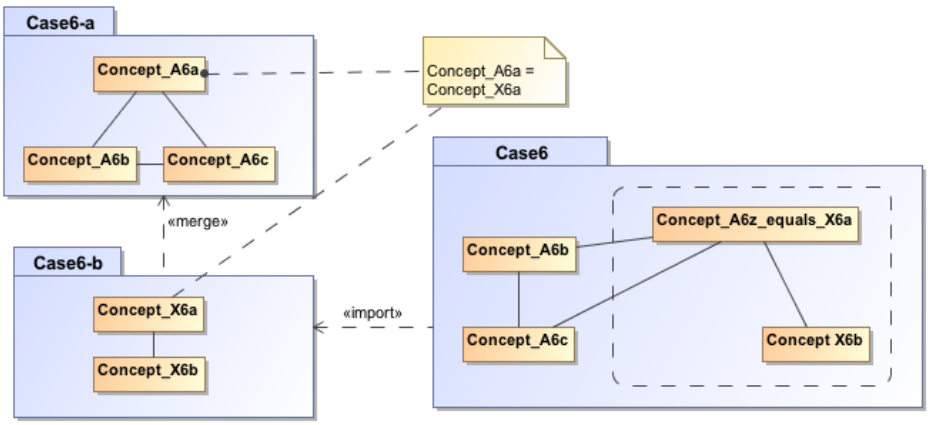

図 6-14 概念要素の分割&結合（または Package Merge を適用）した例

図 6-14 は Package Merge を利用することで、共通要素を持つ二つの複合概念を接続するという概念図である。

本論文の目的はエンタプライズアーキテクチャ側をベースに他の技術を表現する概念群を統合することであり、上の例では点線で囲まれた Concept X 系を優先し考える。Case 1-4 のような関係であれば、類似性を持つ二つの概念の関連は単純であり、二つのアーキテクチャを直接的に接続出来る。Case 5-6 のような関係は単純な追加という解は存在しないが、関連の評価や Package Merge の利用により既存概念に外部概念を付加した構造を導入出来る場合がある。

以上の手順を図 6-15 によりアクティビティ図の形で示す。

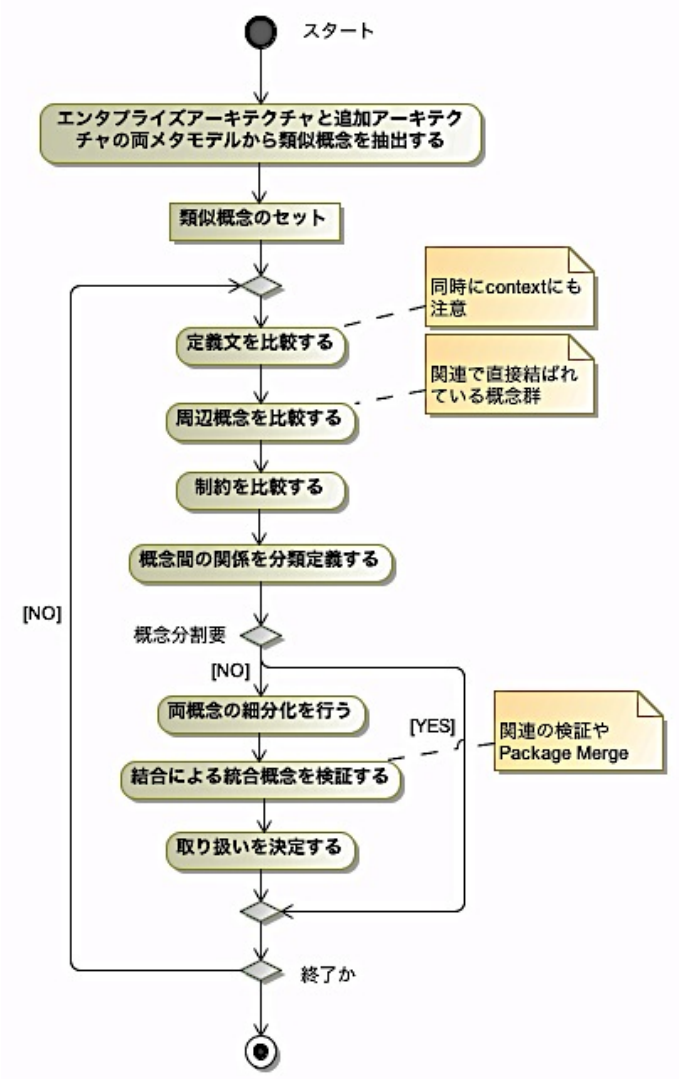

図 6-15 概念比較手順

ステップ３：

エンタプライズアーキテクチャに対する新技術要素の組み込み方は新技術要素ごとに独立したものとして用意する。そうしておけば、検討ベースではあるが必要に応じた組み合わせを選択出来る。

複数の新技術要素を組み込む場合で、概念の位置付けにコンフリクトが発生しそうな場合は、まず一件の組み込みを行い、その結果を新たなエンタプライズアーキテクチャメタモデルと考え、統合手順を最初から実行するものとする。

新技術の提供するものは、特定のクラスの利用者やオブジェクトに従来無かった能力を与える、という場合が多く、その場合 RM-ODP では物を表す Object や人を表す Party が関連づけの際の有力候補となる。しかし、ビジネスプロセスのように人でもオブジェクトでもない特定の概念や関心領域が対象となる場合もある。

ここからは、エンタプライズアーキテクチャのメタモデルを拡張する例を幾つかあげ、上記の考え方が実用的なものであるか否かを検証する。それに続き、メタモデルの UML Profile としての実現について述べる。

6.6.1 メタモデルの拡張検証

(1) モバイルコンピューティングのメタモデル例

モバイルデバイスの特徴はその名の通り移動体である点にある。移動するものは GPS 機能他を備え位置情報を持ちそれが移動に伴い刻々変化するという特性を持つ。位置情報には高度も含まれ、３次元空間の一点として捉えることが出来る。単位時間あたりの位置情報の変化からデバイスの移動速度も算出出来る。他にもジャイロセンサーによる角度検出も行えるものがある。このような特徴を備えたコンピュータと認識すると、時刻（location in time）及び位置（location in space）といった属性を持ち、一般的なライフサイクルにも従うオブジェクトと見ることが出来る。

Grassi 他の研究[65]によると、モバイルコンピューティングのメタモデルのコア部分は次のように捉えている。この図は UML Profile 定義だが、導入する要素は Place, NodeLocation, AllowedDeployment, CurrentDeployment であり、やはりロケーションを中心概念として UML の拡張を提案している。彼らの論文から該当するモデル定義部分を抜き出したものを、図 6-16 に示す。考え方としては、Node に移動を意識した Location を持たせるものである。

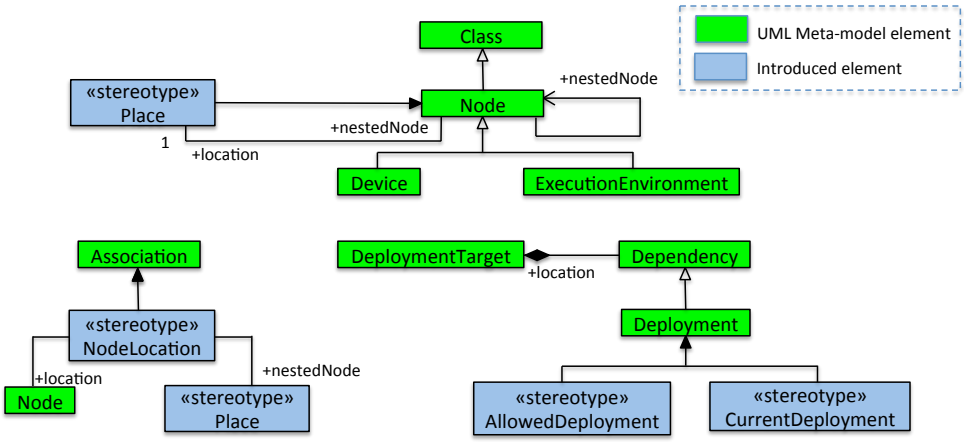

図 6-16 モバイルコンピューティングメタモデルのコア部分

また、他の論文でもこの方向性が認められる[66-68]。従って本論文では、移動体という意味で位置を持たせるに相応しい概念を選択し、時刻と位置のセットを持つように提案する。

Mobile Computing	RM-ODP	関係
Mobile Entity	Object	
Location	Location in space	

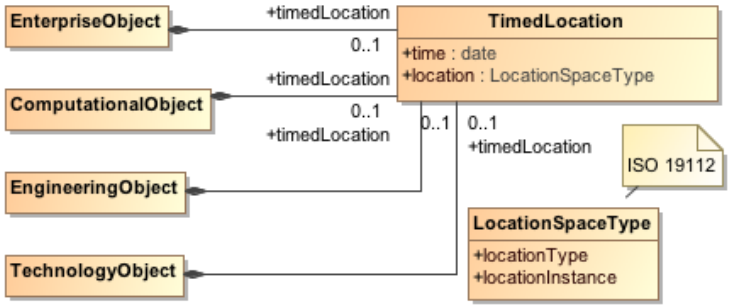

図 6-17 モバイル拡張

この考え方に基づき、モバイルコンピューティングのメタモデルと RM-ODP のメタモデルの関連づけを記述したものが図 6-17 である。RM-ODP で記述する移動可能な要素はすべてビューポイントオブジェクトであるため、これらに時刻ごとの標準位置情報を持たせることでモバイルオブジェクトの記述を可能にするものである。

(2) クラウドコンピューティングのメタモデル例

テレコム系の世界では電話同様に壁にあるソケットにケーブルを差し込むと欲しいサービスが受けられるというビジョンはかなり古くから存在した。かつて存在した TINA Consortium では交換機のネットワークを分散処理環境とし、そこで種々のテレコム系サービスを提供することを考えていた。現在のクラウドコンピューティングの Software as a Service（SaaS）はこの交換機ネットワークをインターネットに置き換えた物ということが出来る。

米国の NIST はかなり早い時期にクラウドコンピューティングの概念整理を行い、Special Report[64]として公開した。その中で、SaaS/PaaS/IaaS という分類や Public/Private/Hybrid などの形態を表現する言葉が定義され一般化した。NIST の出版した Cloud Taxonomy があり、その図を下に引用する。図 6-18 の Service Deployment 及び Cloud Service Consumer の二つのブランチをクラウドコンピューティングに関連するモデル要素のカテゴライズに重要な概念として取り入れた。

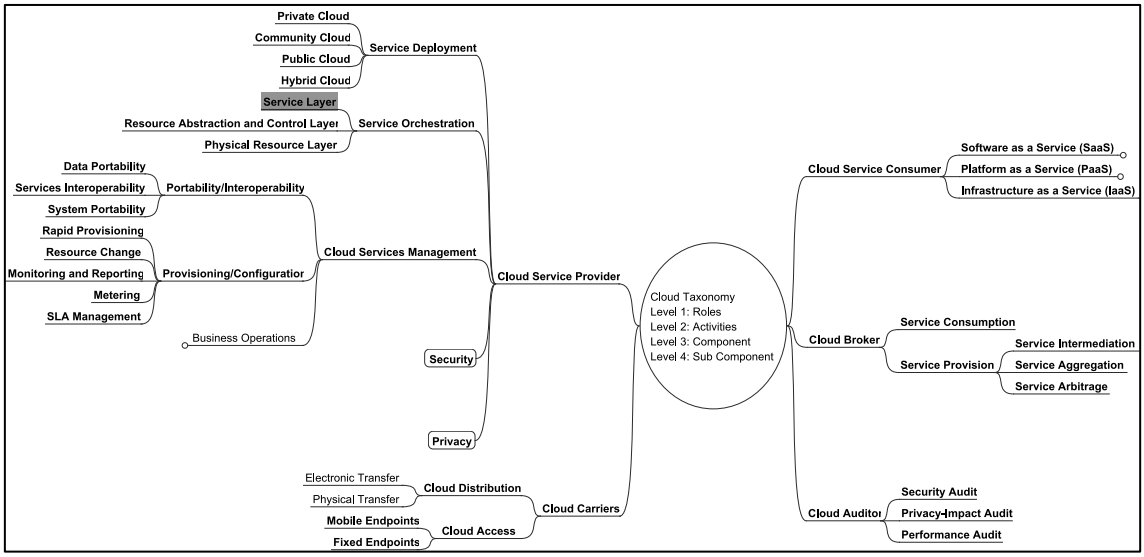

図 6-18 NIST Cloud Taxonomy

図 6-18 には他の要素も含まれるが、エンタプライズアーキテクチャとしての関心事としてはこれらを最初に表現できる必要があると考えたためである。

Cloud Computing	RM-ODP	関係
Cloud	Object	各ビューポイントにおける Object と対応
Service Deployment	Object	Object に付随
Cloud Service Consumer	Object	Object に付随

クラウドコンピューティングの特徴は、アプリケーションをサービスとして提供する、ミドルウェアプラットフォームやインフラストラクチャプラットフォームを提供しその上で顧客システムを稼働させる、といった従来にない構成で企業システム機能を提供できる点にある。実体は企業の外部に存在するため、従来のアーキテクチャからは有用なサービスを提供するブラックボックスが追加されたようにも見える。

これらサービスを提供する主体をエンタプライズアーキテクチャ側で考えるとオブジェクトとなる。メタモデルではクラウドコンピューティング属性をオブジェクトと関連づける。

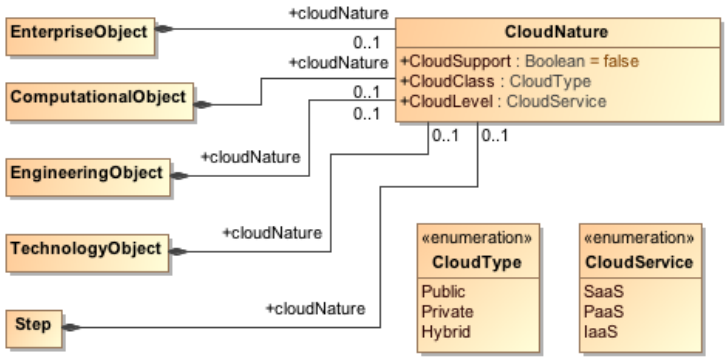

図 6-19 Cloud 拡張

この考え方に基づき、クラウドコンピューティングのメタモデルと RM-ODP のメタモデルの関連づけを記述したものが図 6-19 である。RM-ODP で記述する要素のうちクラウド上に配置できるものは、ビューポイントオブジェクトやそれらの間のビジネスプロセスなどであり、これらにクラウド特性を持たせることであるモデル要素がクラウド上に存在するという記述を可能にするものである。

(3) ソーシャルネットワークのメタモデル例

Twitter や Facebook[71]といったソーシャルネットワークはコンシューママーケットでまず成功を収めた。その成功要因を分析した結果、人と人のつながりの力を認識したベンダーは企業システム向けツールにソーシャルネットワークを組み込み始めた。

Parunak 他の研究[72]によると、ソーシャル構造の中心概念は Role, Agent, Group, Environment と見立てている[69][70]。以下の図 6-20 は注釈を付した簡略図である。ソーシャルグループに人々が参加し、人はその中で何らかの役割を果たしながらコミュニケーションをとるという見方である。これらに対応する概念はほぼすべて RM-ODP に存在するが、やはりソーシャルネットワークという文脈で使われるため違

いもある。

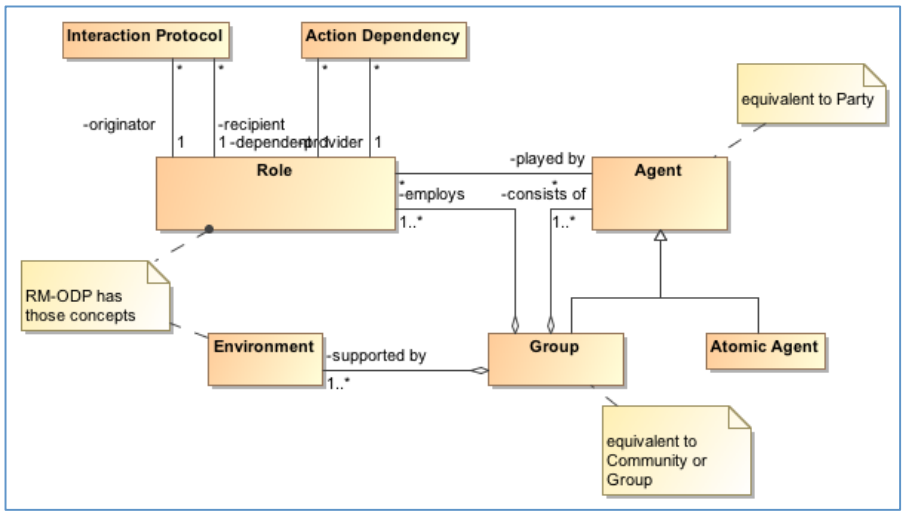

図 6-20 ソーシャルネットワーク概念図

ソーシャルネットワークの特徴は、人の要素に関連づけられるソーシャルコミュニティ、ソーシャルプロファイル、そしてソーシャルな関連である。これらを人の要素を持つオブジェクトの Party に関連づける。

Social Computing	RM-ODP	関係
Role	EV: Role	
Agent	EV: Party	
Group	EV: Community	
Environment	EV: Environment	

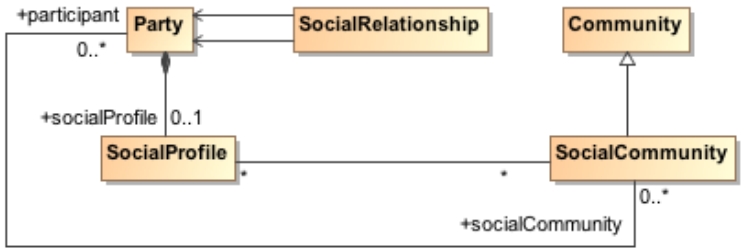

図 6-21 ソーシャル拡張

この考え方に基づき、ソーシャルネットワークのメタモデルと RM-ODP のメタモデルの関連づけを記述したものが図 6-21 である。RM-ODP で人を代表する概念の Party を中心に、Party 間の関連、Party のプロファイル、また Party が参加するソーシャルコミュニティは RM-ODP の Community を継承するものとして位置づけた。これによ

りソーシャルコミュニティに参加する Party オブジェクト、Party が果たすロール、Party の活動に関する Policy、Party 間のインタラクション、などエンタプライズアーキテクチャというコンテキストにおいて RM-ODP 概念の自然な拡張としてソーシャルネットワークを扱うことが出来るようになる。

(4) BPMN のメタモデル例

BPMN はエンタプライズアーキテクチャにおけるプロセスの概念を詳細化した仕様として捉えることが出来、ダイアグラム要素に対応する規定を含むこともあり、標準仕様の中で 100 を超えるメタモデル概念を定義している。対応付けが必要な主な概念は中心的役割を果たす概念だけであり以下をあげることが出来る。これはエンタプライズビューポイント言語の拡張と位置づけることが出来る。

BPMN	RM-ODP	関係
Process	EV: Process	置換可能
Participant	EV: Actor	置換可能
Activity	EV: Action or Activity	置換可能（ODP から BPMN へ）
Data Object	EV: Artifact or Resource	置換可能（ODP から BPMN へ）

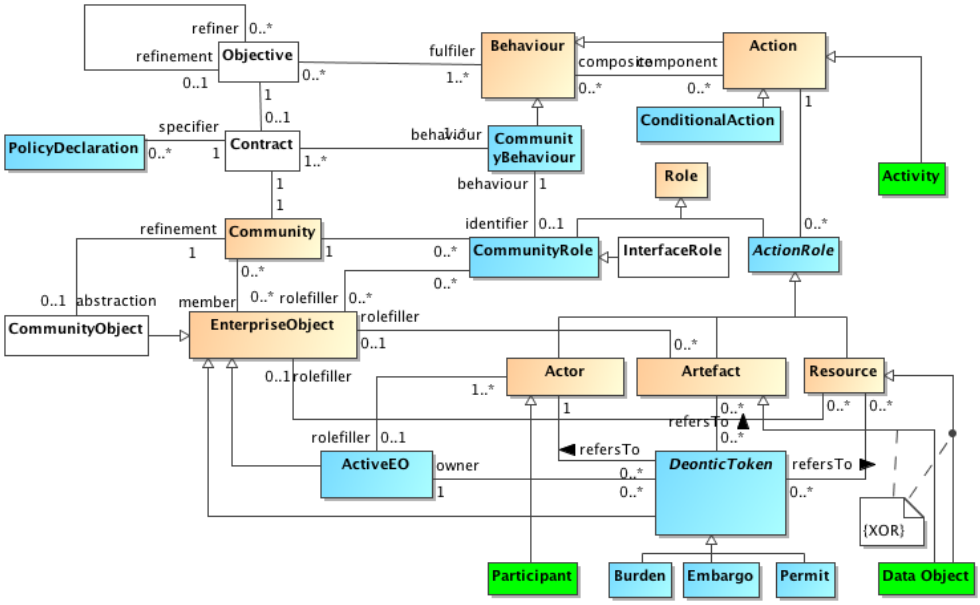

図 6-22 ビジネスプロセス拡張（1/2）

図 6-22 は BPMN の主要概念のうち、Activity、Participant、Data Object について RM-ODP の概念との対応を示すものである。

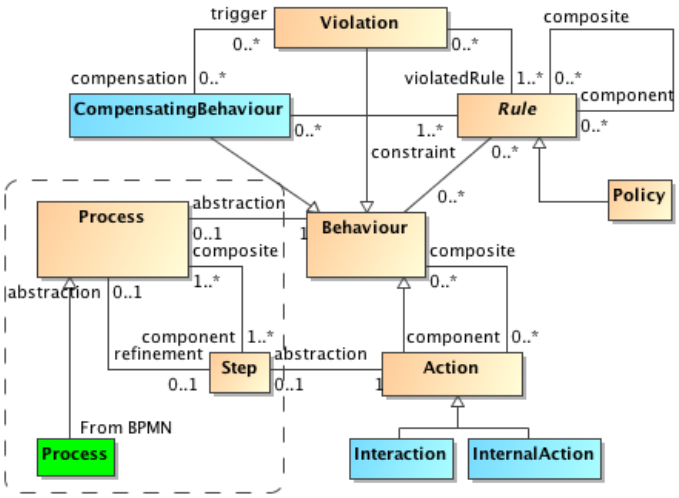

図 6-23 ビジネスプロセス拡張（2/2）

図 6-23 は BPMN の Process が RM-ODP の Process を継承し詳細化していることを示すものである。プロセスとしては同じものであるが、抽象レベルが異なることを示している。

上のような関係が成立した場合、用語として一致する Process を除き、二つのメタモデルの実質的な統合が実現できる。これは、RM-ODP の Process 概念の抽象度が比較的高いのに対し、BPMN の Process 概念がプロセス図記述レベルの具体的なものであること、また基本的な考え方も近いため、詳細化に近いメタモデル統合が可能になっているものと判断出来る。

(5) SoaML のメタモデル例

SoaML はサービス指向のシステムを記述出来るモデリング言語であり、25 種類のメタモデル概念を含んでいる。適用領域は Role に関連してエンタプライズビューポイント、そしてコンポーネントに関連してコンピュテーショナルビューポイントの二つが中心となる。

SoaML	RM-ODP	関連
Port	CV: OperationalInterface	置換可能（SoaML のコンテキストで）
Participant	EV: Actor CV: ComputationalObject	置換可能（SoaML のコンテキストで）
Collaboration	EV: 複数 Actor にまたがる Interaction	置換可能な場合有（SoaML のコンテキストで）

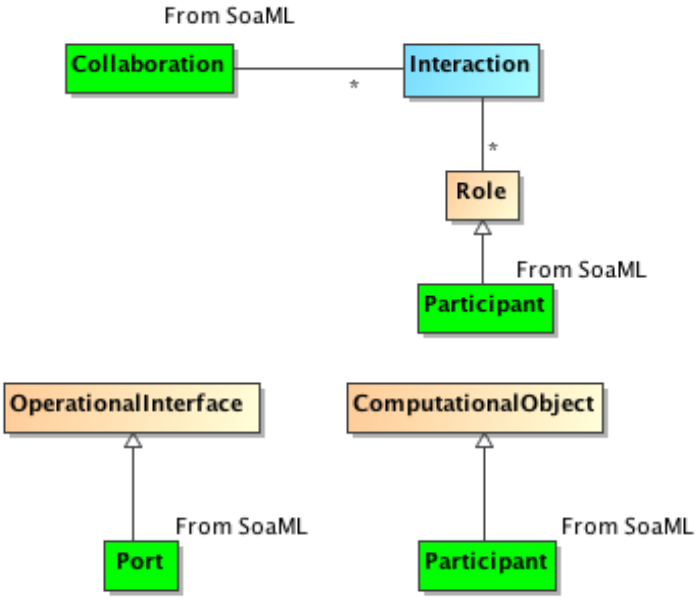

図 6-24 SoaML 拡張

図 6-24 は SoaML の概念と RM-ODP の概念の対応関係を示すものであるが、RM-ODP 側の Interaction と SoaML 側の Collaboration が異なる概念ながら近い関係にあることから、パターン的には図 6-6 に近く、関連する Participant、Port などを RM-ODP のコンピュテーショナル概念と対応づけることが可能である。

6.6.2 UML Profile の拡張検証

(1) UML Profile

通常 UML Profile の設計では、メタモデルを入力情報として利用し、そこに含まれる概念、関連、制約を UML 要素へと対応づける作業を行う。エンタプライズアーキテクチャとして RM-ODP を利用する場合、その UML Profile もまた Use of UML for ODP system specifications として標準化されているため、これをベースとして利用する。全体を引用すると量が多くなるため、関連する箇所のみ提示する。

メタモデルの統合の結果追加された新技術関連のメタモデル要素については新たに UML Profile 定義を行う必要がある。ここでは、上の五つの領域についてオブジェクトやパーティに関連づけたメタモデル要素をどう取り扱うか検討した。

(2) Stereotype 設計

既存のシステムモデルに与える影響を考慮し、既存の UML Profile 要素を改訂するというアプローチを採用した。この方式の場合、既存の stereotype に属性を追加し妥当と思われるデフォルト値を用意するため、UML Profile を入れ替えることで自動的

に新機能対応になる箇所もあり、既存のモデルがある場合は少ない修正量で更新が出来る。

他のアプローチには、新たなメタモデル要素に対応する UML Profile 要素は独立した stereotype として定義するとするものも考えられるが、多くの stereotype を生み出し、またモデル図の枚数も増えることが予想される。

(3) モバイルコンピューティング Profile

ベースとして用いる標準では、各ビューポイントオブジェクトはそれぞれが独立した stereotype として定義される。すなわち «EV_Object», «CV_Object», «NV_Object», «TV_Object» が定義されている。これらを拡張し time や location など新たな属性のセットを持たせる。

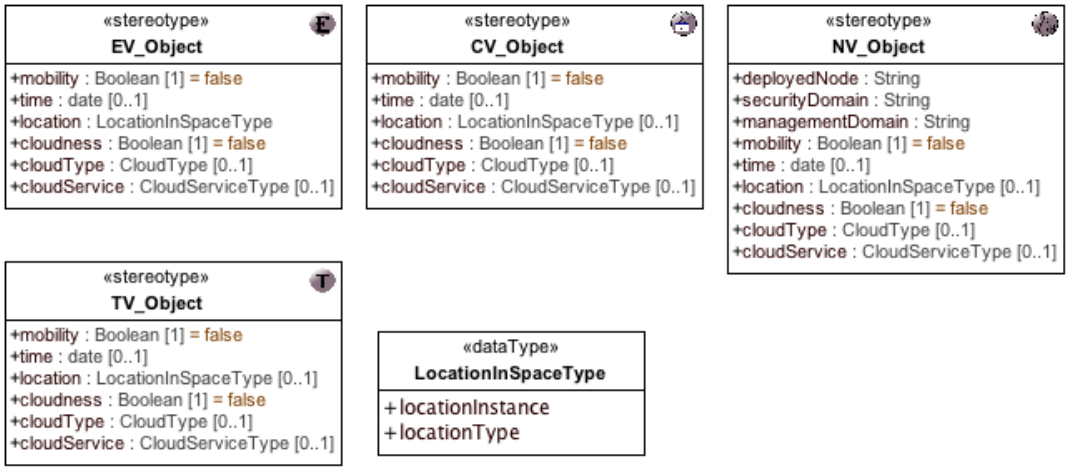

図 6-25 UMP Profile モバイル拡張

図 6-25 は LocationInSpaceType を導入するとともに、各ビューポイントオブジェクトに mobility と time を属性として持たせるよう Profile を拡張したことを示す。

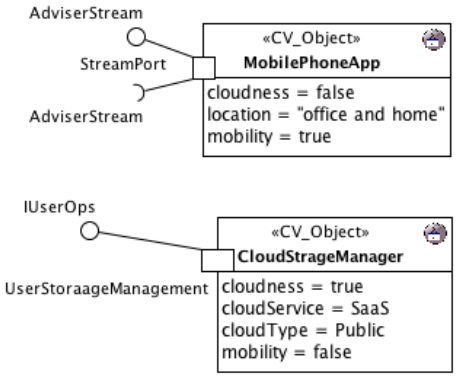

図 6-26 UMP Profile モバイル拡張利用例

図 6-26 は図 6-25 で修正したステレオタイプを利用した場合のモデル例である。

(4) クラウドコンピューティング Profile

ベースとして用いる標準では、各ビューポイントオブジェクトはそれぞれが独立した stereotype として定義される。すなわち «EV_Object», «CV_Object», «NV_Object», «TV_Object» であり、ステップは «EV_Step» が定義されている。これらを拡張し属性として CloudSupport, CloudClass, CloudLevel などを持たせる。

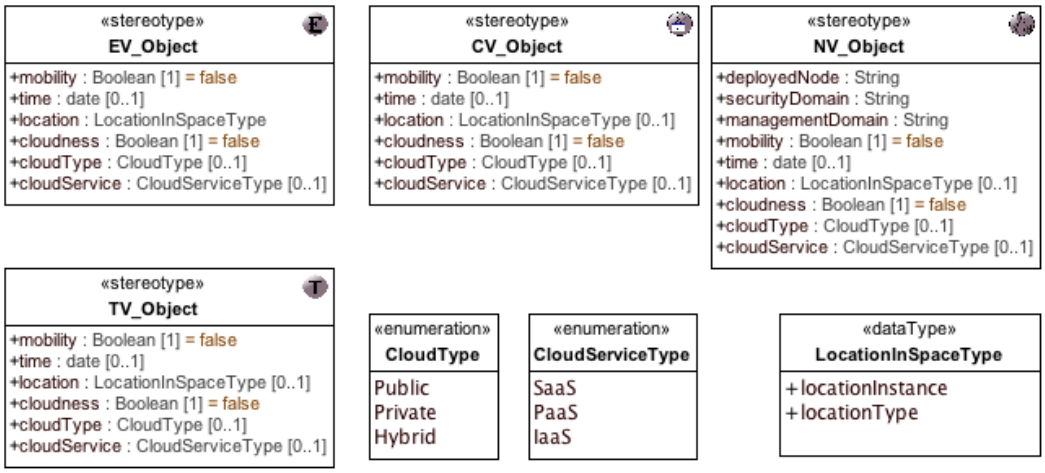

図 6-27 UMP Profile クラウド拡張

図 6-27 は CloudType と CloudServiceType を導入するとともに、各ビューポイントオブジェクトに cloudness、cloudType、cloudService を属性として持たせるよう Profile を拡張したことを示す。

なお、モバイルコンピューティングとクラウドコンピューティングの修正形態に類似性が高いため、両者を併せた拡張記述としている。

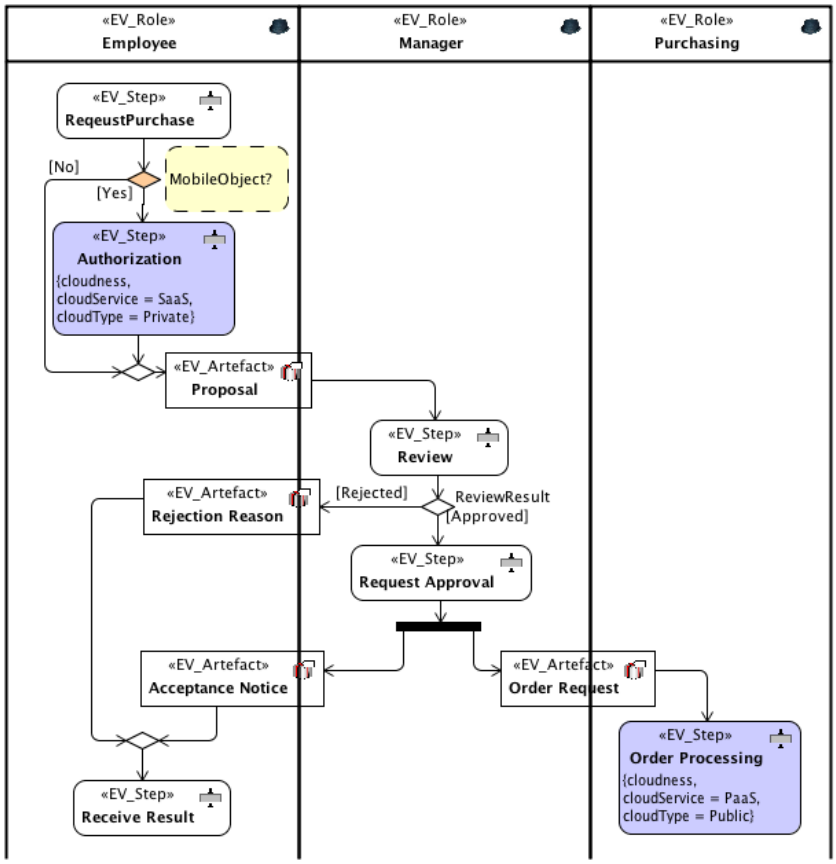

図 6-28 UML Profile クラウド拡張利用例

図 6-28 は図 6-27 で修正したステレオタイプを利用した場合のモデル例である。

(5) ソーシャルネットワーク Profile

ベースとして用いる標準では、パーティやコミュニティが stereotype として定義されている。すなわち «EV_Party», «EV_Community» である。これらを拡張し、更に SocialRelationship 要素は Association を拡張する新たな stereotype とする。

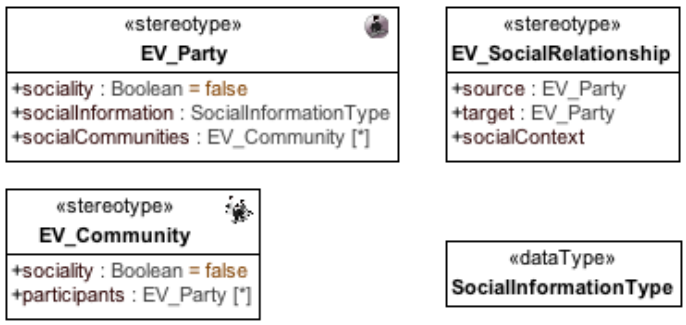

図 6-29 UML Profile ソーシャル拡張

図 6-29 は SocialInformationType を導入し、Community と Party をソーシャル対応に拡張する属性を加え、新たなステレオタイプである SocialRelationship を導入するという Profile 拡張を行うことを示す。

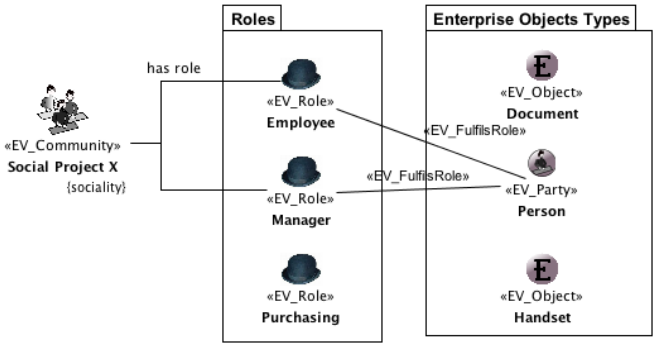

図 6-30 UML Profile ソーシャル拡張利用例

図 6-30 は図 6-29 で修正したステレオタイプを利用した場合のモデル例である。

（6）BPMN Profile との統合例

メタモデルでの類似概念の関連性より、以下のステレオタイプが両者の間をつなぐことになる。すなわち、BPMN は RM-ODP 概念の一部をスペシャライズしたものとして位置づける。

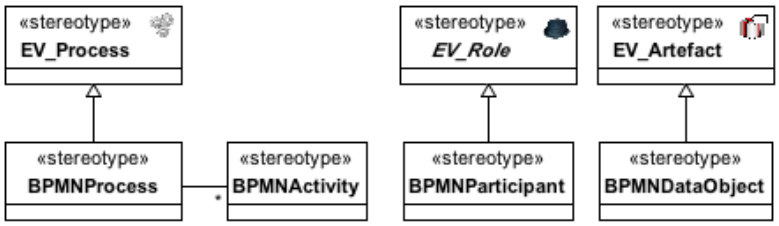

図 6-31 BPMN 用 UML Profile との関連づけ

図 6-31 は両者の UML Profile の関連づけ方を示したものであり、メタモデルレベルでの統合の考え方に基づいている。

モデル記述では、メタモデルと Profile の両方でコンフリクトを解決しているため、両 Profile をそのまま利用し Process 定義についてだけ BPMN Profile を利用することが出来る。両者の意味が重なっていることを明確にしたい箇所については、例えば Process に対し «EV_Process, BPMNProcess» というように二つのステレオタイプを適用することもできるが、意図や意味合いを明確にする必要がある。

（7）SoaML Profile との統合例

メタモデルの関連から BPMN と類似の関連を想像するが、この場合 RM–ODP と SoaML の UML Profile 設計方針の違い（メタクラス選択の違い）から単純な継承で結

ぶことは出来ない。そのためここでは両者を関連の形で接続し、RM-ODP のこれら概念がモデルの中で用いられている場合、関連する SoaML モデルが存在する可能性を示している。

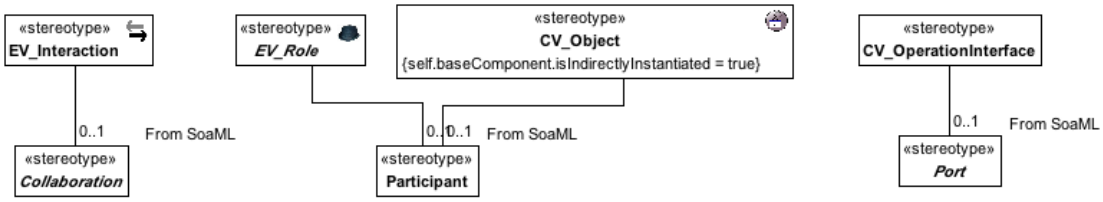

図 6-32 SoaML 用 UML Profile との関連づけ

図 6-32 は両者の UML Profile の関連づけ方を示したものであり、メタモデルレベルでの統合の考え方に基づいている。SoaML は RM-ODP の概念に近い概念を多く備えるが、UML へのマッピング方法が異なるという事実がある。上の関連による接続は、例えば RM-ODP の Interaction モデルのコア部分が SoaML の Collaboration モデルのコア部分に対応することを意味し、UML の世界では例えば継承による密な接続関係が築けず、ある Interaction モデルが全体としてある Collaboration モデルに対応することを意味しており、これは RM-ODP の Correspondence で結ばれる関係に近い。

6.7 まとめ

本章では、変化に対応出来るエンタプライズアーキテクチャの提案と題し、既に存在するアーキテクチャに新たなアーキテクチャを融合させる場合に発生する問題点の分析と取り組み方の提案、またモバイル、クラウド、ソーシャルという三つの新技術を例にとりモデリングに基づく統合方式に関する提案を行った。

第7章　　変化に対応出来るエンタプライズアーキテクチャの設計

提案する方式に従い、小規模なサンプル企業システム（部分）を用意し、幾つかの変化を与え、モデルベース開発のプロセスにおける影響を観察し、評価・考察を行う。

7.1 概要

5.7 の（2）で RM-ODP に基づくモデル記述例を紹介したため、ここでは先の記述例に基づきいくつかの拡張を考える。

4.7 で作成したモデルは次の通りであった。

- RM-ODP に基づく UML モデル構成例（図 5-4）
- エンタプライズモデル概要図（図 5-5）
- ロール定義例（図 5-6）
- インタラクション定義例（図 5-7）
- プロセス定義例（図 5-8）
- 不変スキーマ例（図 5-9）
- コンピュテーショナルモデル例（図 5-10）
- エンジニアリング要素例（図 5-11）
- エンジニアリングモデル例（図 5-12）

これに対し、次の三つの機能追加が行われたとする。

モバイル機器の利用が増えたため、モバイルを使った予約も可能とするが、その場合予約回数に制限を設ける。

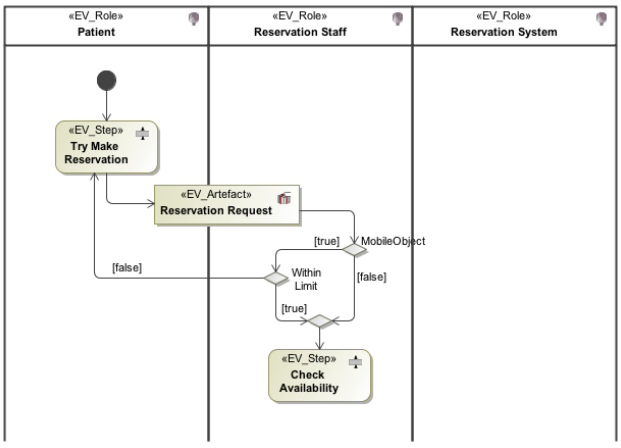

図 7-1 プロセス改訂例

図 7-1 はモバイル予約であり所定の予約可能回数を超えるかに関わるプロセスフローの一部である。

病院予約システムを SaaS プロバイダと契約することでクラウド化し院内でのシステム維持管理に割く費用削減を実現することにする。

図 7-2 コンポーネント例

図 7-2 は予約業務を遂行するコンポーネントがクラウド上に配置された場合のコンポーネント間の関係例を示す。

更に病院内のソーシャルネットを使い予約業務に関する利用者の声や改善要望などを担当科間で共有出来るようにした。

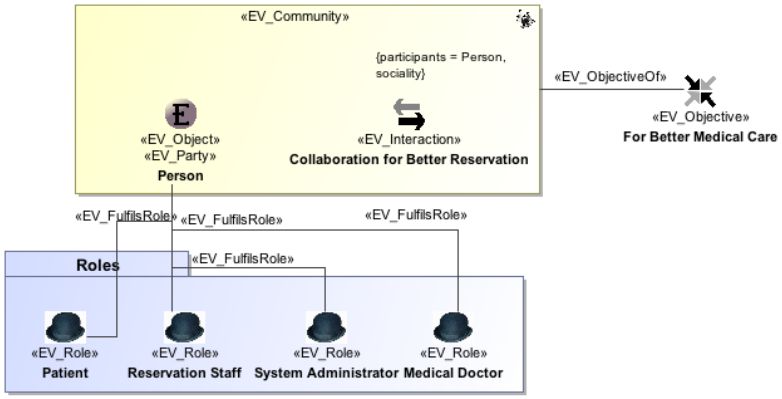

図 7-3 ソーシャルコミュニティ例

図 7-3 は、予約システムの利用者や担当者が予約処理に関する病院内ソーシャルネットに参加しより良い予約システムを実現するという例を示す。

7.2 モデル要素

基本機能の範囲では次のようなモデル要素を記述したとする。

コミュニティ：患者コミュニティ（患者、家族）、病院内コミュニティ（予約処理担当者、システム管理者、医師）

プロセス：予約作成プロセス、予約変更プロセス

情報モデル：患者、予約、病院、カレンダー、予約要求、予約変更要求、予約キャンセル要求、緊急度、紹介状、治療型

コンポーネント：GUI コンポーネント、患者用操作コンポーネント、予約担当者用

操作コンポーネント、ログインコンポーネント、ユーザ管理コンポーネント、トランザクション管理コンポーネント　他

分散処理インフラ：分散処理環境、DB 環境　他

ソフトウェア構成：PC (Windows + Browser), Server (Linux, Apache Server, Tomcat, MySQL, Web Application Framework)

機器構成：PC-LAN-Server-DB

7.3 改訂モデル要素

拡張機能を加えた範囲では、次のような改訂モデル要素を記述する。

改訂版プロセス：モバイル予約、クラウドサービス利用（追加）

改訂版情報モデル：モバイルデバイス（追加）

改訂版コンポーネント：クラウド予約サービス（追加)、モバイル予約コンポーネント（追加)、ソーシャルサービス（追加）

改訂版分散処理インフラ：クラウド対応分散処理環境、クラウド上 DB 環境　他

改訂版ソフトウェア構成：Mobile Device (Android + Browser)

改訂版機器構成：Mobile-WAN-Firewall-HTTP Server-LAN-Server-DB、Cloud 環境、Cloud 接続

7.4 モデル変換

モデルからモデルの変換は CIM/PIM/PSM のいずれのタイプのモデルであっても適用出来る。まず、上に書いた RM-ODP ベースのエンタプライズアーキテクチャモデルがこのどこに位置づけられるかを判定する必要がある。Enterprise と Information は ODP システムの存在以外に Computation に関わる概念が現れないため CIM と呼ぶことができるし、Computational と Engineering はコンポーネント相当を考えるという段階で PIM に近いと言える。そして Technology で具体的なソフトェアやハードウェアとの対応を取るため PSM に近いモデルと言える。従って上のモデルはこれら三つの混合物であり、Correspondence により要素間の対応付けがなされ、関心の分離により分割記述した仕様を再度つなぎ合わせている。

次に、目的のアプリケーションについて見てみる。例えば Web アプリケーションは Tomcat のようなサーブレットコンテナや EJB コンテナ、更に種々のフレームワークと合わせて実装される。プラットフォームに用いられる技術要素が変わる可能性を考慮に入れると、特定のプラットフォームを選択して一般的なモデル変換を論じること

が出来ないことが分かる。従って逆方向から、コードの種類に応じてどのようにモデルから導くことが出来るかを検証する。Web アプリケーションの開発時に作成すべきコード成果物には次のようなものがある。

(1) 情報定義：対応するコードで getter/setter 類を含む
(2) 入力妥当性検証のためのコード
(3) DB アクセス：SQL 文など
(4) DB とのマッピング：DAO など
(5) 情報モデルの参照・更新処理
(6) 振る舞い関連：業務制御フロー、業務処理ロジック、など
(7) 画面定義・出力帳票定義：サポートするデバイスに応じたコード
(8) メッセージ定義
(9) ログ関連コード
(10) Web サービス（クライアント側、サーバ側）
(11) プログラミング言語やフレームワークが要求するコード
(12) JUnit 等テスト用コード

上記以外にもシステム形態に応じた種々の成果物を作成することになる。これらに対し、コード生成の素材となるモデル情報との対応を調査した。

(1) 情報定義

素材は情報モデルでそれが概念モデルであればモデルの世界で要求される情報定義のレベルに合わせた詳細化を行う。情報モデリングは、問題に対して必ずしも絶対的な解がある訳ではないため、その品質はモデル開発者に依存する部分が多い。一つの提案は､概念モデルないし情報モデルの早期デバッグである｡情報モデルを直接 CRUD システムの対象として GUI のプロトタイピングを行うことで、関係者の間でのコンセンサス作り（仕様確定）を素早く実施出来る。モデリングとモデルからテキストへの変換を合わせて用いることでこの最後のタスクを容易にすることに貢献出来る。

(2) 入力妥当性検証のためのコード

素材は情報モデルに含まれる定義情報やビジネスルールの形で記述される（許される値の範囲などの）妥当性情報など。これらの情報は、通常の情報モデル定義に全て含まれる訳ではないため、情報モデル要素にリンクさせた制約またはビジネスルール利用になる。制約の場合、OCL で書かれていたとすると、OCL の実行環境は十分でないため、プログラミング言語で書き直すことになる。ビジネスルールであれば、ルール実行をビジネスプロセスとして扱うことでプロセス定義からルールに対応するテキストを生成することも出来る。

(3) DB アクセス

DB アクセスのベースは情報モデルで一般的な CRUD 用 SQL 文だけを生成し、複雑な SQL 文は生成の対象としない。これはデータベースの SQL サポートに差があるため、個別にマニュアルで対応せざるを得ないためである。

(4) DB とのマッピング

情報モデルと DB との対応付けが出来る場合には一般的なマッピングコード（DAO など）の生成はモデルからテキストへの変換で実現可能だが、カスタマイズやチューニングを行う場合はマニュアル変更が必要で変更箇所の管理が必要になる。

(5) 情報モデルの参照・更新処理

DB 更新処理は DBMS が担当するが、依頼側からはトランザクションの範囲指定が必要となる。制御能力が必要な場合、アクティビティの中の記述としてモデルで指定することが可能で、コード生成時にトランザクションの開始等を埋め込むことが出来る。CRUD を用いた業務処理については、後で述べるプロセスにおけるステップの詳細化モデル作成の中で検討する。

(6) 振る舞い関連：業務制御フロー、業務処理ロジック、など

素材は業務記述に含まれるシーケンス図、BPMN 図、アクティビティ図、状態遷移図など。これらのモデル図の中で、シーケンス図は精密に記述したものでないと使いづらい。モデル変換で厳密性が確保出来るのはアクティビティ図と状態遷移図である。BPMN 図もアクティビティ図相当であるが、UML とは異なるモデルデータの解析が必要となる。これらのモデルの問題点は、特にアクティビティやアクションの粒度と厳密性にあり、粒度が大き過ぎるとコードに変換できないし、プログラムの 1 ステップ相当のことを書かれるとモデル規模が大きくなり、書かれている内容がダイアグラムから読み取れなくなる恐れもある。適度な大きさで揃えることが出来る場合にだけモデル変換による業務処理ロジック生成が可能となる。

(7) 画面定義・出力帳票定義

素材は情報モデルだが基本的な部分しかカバー出来ない（画面フローに基づく開発形態を取るケースもある）。画面設計や出力帳票設計を情報モデルに基づき行うことは可能であるが、通常利用者はその程度で満足されずカスタマイズを行うことになる。そのためカスタマイズが容易な仕掛けが必要である（モデル変換の領域を超える）。

(8) メッセージ定義

素材は特に無く、プロジェクトの基準に従ったメッセージ一覧を用意する（振る舞い関連の図の判断箇所やエラーケースにメッセージが現れている場合もある）。この領域にモデリングの応用は余り考えられていないが、メッセージカタログなどを作成す

るツールを作るような場合には利用することも出来る。

(9) ログ関連コード

ログ機能に対応するコンポーネント、属性、stereotype 等をモデルに定義し、個々のモデルで指定するように設計する。

(10) Web サービス（クライアント側、サーバ側）

クライアント側の場合、Web サービス提供者側の WSDL や XML スキーマに対応する情報モデル要素を得られればモデル変換を利用出来る。

サーバ側の場合は、振る舞い関連モデルの該当箇所が素材となるが、通常は一塊の機能が該当する。振る舞いの入出力情報がインタフェース定義の中心となる。

(11) プログラミング言語やフレームワークが要求するコード

コード生成時には対象言語や対象フレームワーク等のプラットフォームが決まっている必要があり、それを素材とし、モデルからテキストへの変換テンプレートに要求されるコードを埋め込み利用する。

(12) JUnit 等テスト用コード

テストコードのためのモデルの生成は通常行わず、例えば EMF のダイナミックインスタンス機能等を用いテストデータを作成することは出来る。

モデル変換技術を適用した結果生成することになるコードには、主に構造面を記述する部分と振る舞いを記述する部分がある。ここではまず (1) の情報定義について考察する。今回のモデルの中で情報定義は、図 5-9 で示した不変スキーマである。ここでは単に UML モデルから情報オブジェクトを抜き出し Java クラスに変換することを行う。次のような手順を踏むことになる。

- 入力として UML ツール（MagicDraw）で作成したモデルを Eclipse UML2 (v4) XMI 形式にエキスポートしたものを用意する。
- 変換エンジンとして 3.7 の 3) で説明した Eclipse Acceleo を用いることとし、上記 XMI データをワークスペースに配置し、変換スクリプトを作成する。

Eclipse Acceleo にはサンプルの UML2Java 変換定義が付随するため、これをカスタマイズし、ステレオタイプが IV_Object である Class を抜き出した。

```
[template public generateElement(aClass : Class)]
[comment @main/]
[if aClass.hasStereotype('IV_Object')]
[file (aClass.name.concat('.java'), false, 'UTF-8')]
package org.example.hospital2java;
```

図 7-4 IV_Object 抽出箇所

図 7-4 の if 文で抽出を行っている。ただし、hasStereotype という機能は標準にないため query 機能として別途用意した。

次に、Class 間の関連が Class の属性として取り扱われるため、関連に基づく属性を除外した。

```
public class [aClass.name.toUpperFirst()/] {
[for (aProperty : Property | aClass.attribute)]
[if (aProperty.type.oclIsTypeOf(Class))][else]
private [if (aProperty.upper = -1 or aProperty.upper > 1)]List<String>
[else][aProperty.type.name.toUpperFirst()/][/if] [aProperty.name.toLowerFirst()/];
[/if]
[/for]
```

図 7-5 関連に基づく属性の除外

図 7-5 は、やはり if 文で属性が関連に基づくもので相手方が Class の場合を判定している。このような変換ロジックを用意することで、例えば次の図 7-6 のような定型的な JavaBean コードが各 IV_Object 対応に生成できる。

```
package org.example.hospital2java;

import java.util.*;

public class Request {
private Date requestedDate;
public Request() {
    super();
}
public Date getRequestedDate() {
    return this.requestedDate;
}

public void setRequestedDate(Date newRequestedDate) {
    this.requestedDate = newRequestedDate;
}
```

図 7-6 生成コード例

Class の操作部分については、枠を用意することは可能だが、その内容は操作のモデル化が出来ないため定型の CRUD などを除きマニュアル入力にならざるを得ない

別の観点から振る舞いについて考察する。例えば次のようなプロセス（一部）があったとする。

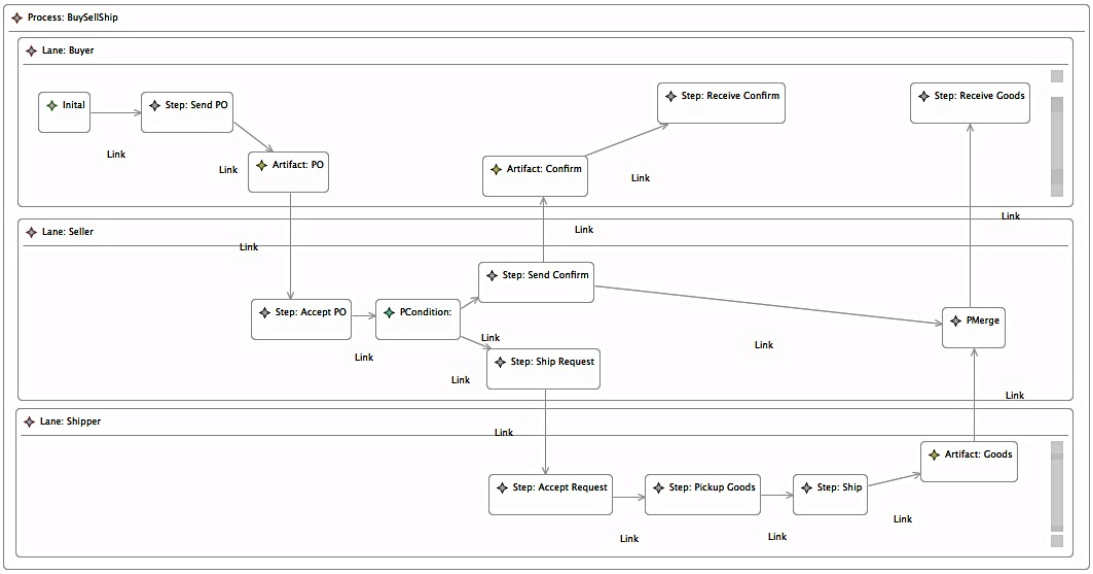

図 7-7 グラフィカルプロセス DSL エディタ

図 7-7 はモデリングツールで説明した手法に基づき、プロセス用のメタモデルを作成後にグラフィカルエディタに書き換え、テストデータを投入したものである。

このモデルは、下の図 7-8 のような XMI 形式のデータとして保存でき、従ってモデル変換の入力として利用出来る。

```
<?xml version="1.0" encoding="UTF-8"?>
<process:Model xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
  <processes name="BuySellShip">
    <lanes name="Buyer">
      <processFlows xsi:type="process:InitialNode"/>
      <processFlows xsi:type="process:Step" name="Send PO"/>
      <processFlows xsi:type="process:Artifact" name="PO"/>
      <processFlows xsi:type="process:Artifact" name="Confirm"/>
      <processFlows xsi:type="process:Step" name="Receive Confirm"/>
      <processFlows xsi:type="process:Step" name="Receive Goods"/>
      <links source="//@processes.0/@lanes.0/@processFlows.0" target="//@processes.0/@lanes.0/@processFlows.1"/>
      <links source="//@processes.0/@lanes.0/@processFlows.1" target="//@processes.0/@lanes.0/@processFlows.2"/>
      <links source="//@processes.0/@lanes.0/@processFlows.2" target="//@processes.0/@lanes.1/@processFlows.0"/>
      <links source="//@processes.0/@lanes.0/@processFlows.3" target="//@processes.0/@lanes.0/@processFlows.4"/>
    </lanes>
    <lanes name="Seller">
      <processFlows xsi:type="process:Step" name="Accept PO"/>
      <processFlows xsi:type="process:Step" name="Ship Request"/>
      <processFlows xsi:type="process:ParallelSplit"/>
      <processFlows xsi:type="process:Step" name="Send Confirm"/>
      <processFlows xsi:type="process:ParallelMerge"/>
      <links source="//@processes.0/@lanes.1/@processFlows.1" target="//@processes.0/@lanes.2/@processFlows.0"/>
      <links source="//@processes.0/@lanes.1/@processFlows.0" target="//@processes.0/@lanes.1/@processFlows.2"/>
      <links source="//@processes.0/@lanes.1/@processFlows.2" target="//@processes.0/@lanes.1/@processFlows.1"/>
      <links source="//@processes.0/@lanes.1/@processFlows.2" target="//@processes.0/@lanes.1/@processFlows.3"/>
      <links source="//@processes.0/@lanes.1/@processFlows.3" target="//@processes.0/@lanes.0/@processFlows.3"/>
      <links source="//@processes.0/@lanes.1/@processFlows.4" target="//@processes.0/@lanes.0/@processFlows.5"/>
      <links source="//@processes.0/@lanes.1/@processFlows.3" target="//@processes.0/@lanes.1/@processFlows.4"/>
    </lanes>
    <lanes name="Shipper">
      <processFlows xsi:type="process:Step" name="Accept Request"/>
      <processFlows xsi:type="process:Step" name="Pickup Goods"/>
      <processFlows xsi:type="process:Step" name="Ship"/>
      <processFlows xsi:type="process:Artifact" name="Goods"/>
      <links source="//@processes.0/@lanes.2/@processFlows.0" target="//@processes.0/@lanes.2/@processFlows.1"/>
      <links source="//@processes.0/@lanes.2/@processFlows.1" target="//@processes.0/@lanes.2/@processFlows.2"/>
      <links source="//@processes.0/@lanes.2/@processFlows.2" target="//@processes.0/@lanes.2/@processFlows.3"/>
      <links source="//@processes.0/@lanes.2/@processFlows.3" target="//@processes.0/@lanes.1/@processFlows.4"/>
    </lanes>
  </processes>
</process:Model>
```

図 7-8 グラフィカルプロセスモデルの XML 表現

但し、図 7-8 に含まれる情報は構造的なものだけであり、実際にモデルに含まれる

各アクティビティが何を行うのかの仕様規定が行われていない。このため、これでもまだ不十分である。

そこで、上の例ではノードのタイプでしか表していない Step について、構造化することでより詳細で限定的なアクションを指定出来るようにする方式が考えられる。その場合、明確に指定すべきは、対象物（例えば、XYZ テーブル）、アクション（例えば CRUD など指定出来ることを限定）、必要なパラメタ、結果のタイプ等が含まれることである。コード生成を実現するためにアクションに求める記述レベルという観点からは、例えば、販売プロセスの中に「注文を受け価格を計算する」というステップがあったとすると、次のようなステップを詳細化したモデル記述が考えられる。

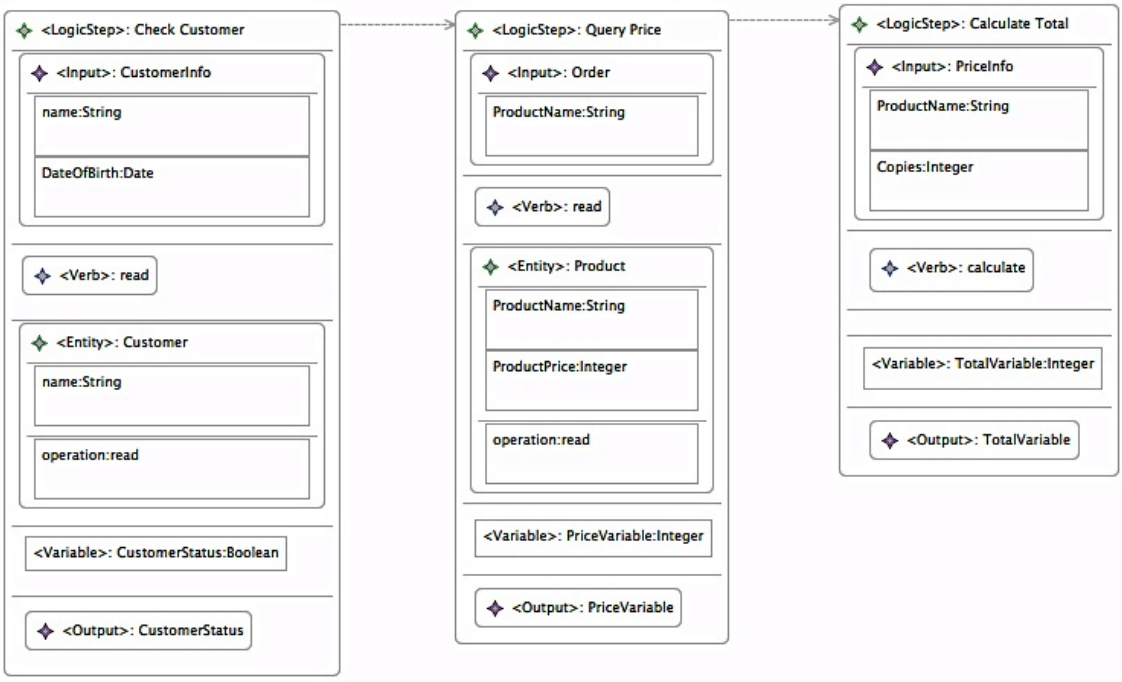

図 7-9 ロジック記述モデル例

上の図 7-9 のロジック記述例では、当該ステップを三つの連続実行するロジックステップに分解している。各ロジックステップは上から入力、動詞、エンティティ、変数、出力、というコンパートメントを持ち、それぞれのロジックステップから意味を持つ一塊のプログラム断片を導くことが出来る程度の抽象度とした。ハイレベル記述になりがちなプロセス定義の各アクションについて、それぞれの詳細化としてこのレベルのモデルを記述することが出来れば、システムの振る舞いについてモデルとコードの距離を近づけることが出来る。なお、上のモデル構造は次のメタモデルにより規定しており、必要があればプロセス定義のメタモデルにステップ構造として組み込む

ことも出来る。

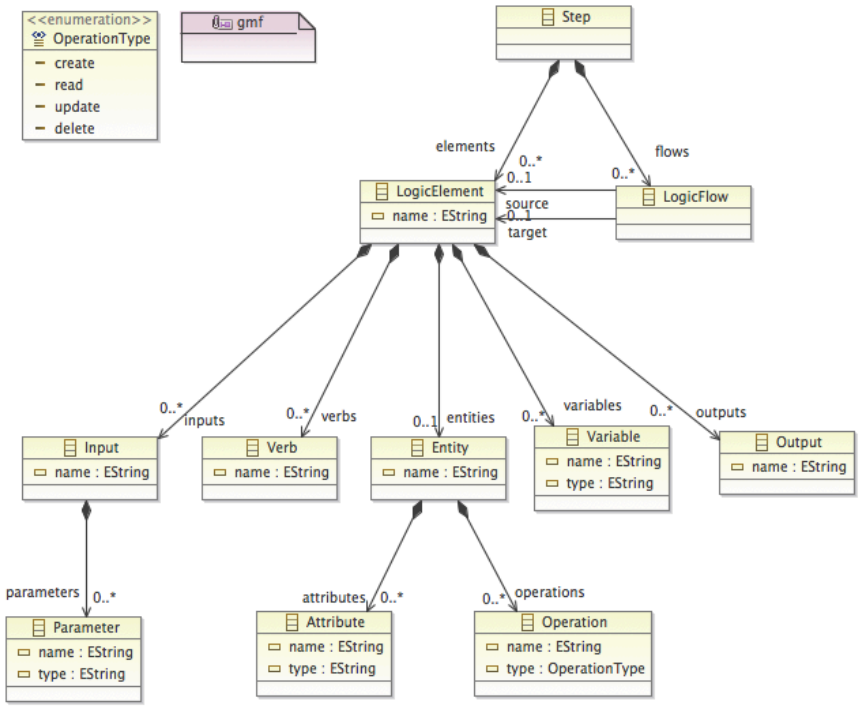

図 7-10 シンプルロジック構造モデル例

図 7-10 の新プロロジック構造モデルは Ecore モデルであり、先に紹介した Acceleo によるテンプレートに基づくテキスト変換という流れに乗せることが出来る。但し、検討すべき点として、どの程度の抽象度での記述が利用者にとってまた開発者にとって適切なのかが残されるが、これは継続検討課題とする。

7.5 他の手法との比較

本論文は、エンタプライズアーキテクチャの拡張を、モデルベース設計開発手法を利用し検討したものであり、以下の二つの手法の交叉する領域の一部と考えられる。

(1) エンタプライズアーキテクチャの開発手法

Zachman Framework を始めとして、策定したエンタプライズアーキテクチャの解説やその適用方法に関連する研究は多いが、アーキテクチャ拡張を前提にした設計手法に関する研究は見当たらない。エンタプライズアーキテクチャに基づくシステムアーキテクチャ開発について、例えば次のようなガイドがある。

“A practical guide to developing enterprise architecture

Step 1. Identify the purpose of your architecture

Step 2. Identify your business questions

Step 3. Identify assumptions and business rules

Step 4. Identify your framework

Step 5. Create a metamodel

Step 6. Identify the models needed in the architecture

Step 7. Integrate the architecture”

これらガイドは、エンタプライズアーキテクチャのフレームワークは既存のどれかを用い、その上で利用者の求めるアーキテクチャを作成する時に備えてというもので、メタモデルの作成を推薦しているものの、拡張ではなく一般的な使い方の範囲にとどまっている。

既存システムとの統合という意味では EAI (Enterprise Application Integration) が現実的なシステム接続方式であるが、エンタプライズアーキテクチャをターゲットとしたものではないためこれも比較対象にはならない。

(2) モデルベース設計開発手法

モデル駆動開発に関連しては多くの研究が行われ、ツール類もかなり利用出来る状況にある。本論文では幾つも提案されている方式から、オープンなツールを利用し実現出来るという範囲で Step の詳細モデルなどの実験を実施した。エンタプライズアーキテクチャを拡張後は特に新たな要素は導入していないため、その後は一般のモデルベース開発と同じ流れをたどる。

概念の比較に関する他の手法としてはオントロジ利用や概念マップなどもあるが、MOF/UML 標準に基づくメタモデル記述であるため、より厳密な分析が出来ると考える。

7.6 評価

前章で述べた変化に対応するモデル化手法をサンプル事例に適用し、拡張を加えた部分的なモデル記述を行った。このモデル記述に特段の問題は見つからず、本手法の適用可能性を示すことが出来た。また、改訂モデルをベースにコード生成する際の考察を行い、幾つかの問題点を抽出した。また業務ロジックについて、プロセス定義における Step 概念の拡張を議論し、コード変換が可能なレベルの仕様・モデル記述例を示した。

提案方式の適用がうまく行くか否かは、前章でのメタモデルまたは概念設計の問題が大きく、事前にどのようなモデル拡張を実現したいかが見えるまで繰り返しの検討が必要であることを確認した。このメタモデルまたは概念設計は、統合領域で出来る限り複合概念を構成概念に分割することが対策となる。そのためには、その概念の使われるコンテキストと周辺概念との関連などを十分に分析し、意味的な対応が取れる

構成概念が現れるまで繰り返すことになる。

7.7 まとめ

本章では、変化に対応出来るエンタプライズアーキテクチャの設計と題して、5 章で取り上げたモデルに対して前章でのべた 5 種類の新技術を導入する場合の変更点について具体的に検証した。コード生成については、出来る部分と困難な部分があることを示し、振る舞いについてはこれまでより実装に近いレベルでのモデリング（ロジックモデリング）という可能性を検討した。

第8章　　結論

本研究の成果を総括すると共に今後の課題について述べる

本研究はエンタプライズアーキテクチャに基づく企業システム仕様に新技術を導入する際、アーキテクチャ記述にモデリング技術を適用することで新技術導入を進めるための方式を提案するもので、適用事例とその評価、モデリング技術を利用した下流工程への展開方法とその考察についても含めた。

(1) システムアーキテクチャ

主なエンタプライズアーキテクチャを比較し、オープン性の観点から RM-ODP を本研究で使用した。TOGAF や DoDAF/MODAF を使っても同様の検討が可能である。企業システム開発で何らかのアーキテクチャを設定するのは良く行われているが、政府系や大規模開発以外でも、多様な視点からシステム全体を分類設計する基盤となるエンタプライズアーキテクチャは利用可能であり、エンタプライズアーキテクチャを利用する場合の開発プロセス上流の概要としてその流れを整理した。

(2) モデリング技術活用方法

OMG がモデリングに関する標準仕様の策定（標準化）を行っているが、その実装はメンバ会社に委ねられている。そこで、一般的な UML ツールと OMG のモデリング仕様をオープン実装している Eclipse の成果物を利用し、モデリング技術の概要と各種モデリングツールの適用について検討した。プログラミング言語の IDE としての Eclipse と比べると日本では比較的利用者・研究者の少ない領域であるが[62][63]、欧州を中心に現在も活発な活動が続いており[88][90]、抽象度を上げたソフトウェア開発の観点からもっと関心を持たれるべき領域と考える。本論文ではこの Eclipse Modeling Project に属する多くのプロジェクト成果物（EMF, GMF, Epsilon, Xtext, Acceleo, ATL, QVT 他）やサンプルを利用し、これらを組み合わせることで、モデリングを重視するソフトウェア開発を支援するツールチェーンとして利用できることを示した。

(3) 変化に対応するエンタプライズアーキテクチャ

エンタプライズアーキテクチャは大規模に使われるため一旦導入されると本当に新しい技術要素を追加するのが容易でなくなる。しかし新技術の企業 IT システムに与える影響範囲を明確にさせる意味で、エンタプライズアーキテクチャを導入し仕様をモデル表現しておくことが一つの対策になる。本論文では、新技術等が現れた場合、エンタプライズアーキテクチャの本質を表現するメタモデルないし概念モデルと新技術のメタモデルないし概念モデルを比較検討し、両者を接続する方法を検討し、共通と思われる概念について、幾つかの接続パターンに当てはまらないか調べ、当てはまら

なかった場合それら概念を合成概念と想定し、分割・詳細化を行い、細粒度の概念を抽出し、得られた細粒度要素からなるメタモデルを対象として上と同様の分析プロセスを適用するという方式を提案した。また、この際エンタプライズアーキテクチャ側を優先させて差分を追加する形でまとめる方式を提案した。そして、モバイル端末、クラウドコンピューティング、ソーシャルネットワーク、ビジネスプロセス、SOA の五つを例として取り上げ、統合のためのアプローチが適用出来ることの検証を行った。その結果、アーキテクチャが深いレベルでからみあっていなければ、幾つかの接続パターンから適用可能な接続形態を見つけることできるという感触を得た。

(4) モデル駆動開発

仕様がモデルとて表現されていると上流工程の成果物に基づきモデル変換などを利用して下流工程の開発を進めることにつながる。上流工程で作成したモデルを下流工程で処理の対象とするには、まずメタモデルとモデルデータをモデル変換の入力（変換前側）としたモデル変換を用意する。静的で主に構造面を表すコードをモデルデータに基づきコードを生成するのは既存の技術（テンプレートエンジン等）を用い比較的容易に実現出来る。しかし、動的な振る舞い面を表すコードを生成するに足る振る舞いモデルは一般的に利用されるアクティビティ図や BPMN 図の標準部分だけでは通常記述できず、この問題を解決する可能性を持つコード生成できる振る舞いモデルとしてあえて抽象レベルの低い DSL を試作した。適切な抽象度についての検討が今後の課題である。

(5) モデリングツール

メタモデル作成には、UML ツールや Eclipse の EMF 系ツールまたは DSL 系のツールが利用出来、UML の場合は UML Profile というステップを踏むものの、モデル作成にも同様のツールが利用出来る。モデル変換には QVT や ATL などモデルからモデルへの変換ツールと Acceleo のようなモデルからテキストへの変換ツールがあり、それぞれでメタモデルとモデルデータを入力とした変換処理を記述・実行することができる。実用性の観点でキーとなるのは最後のテキスト変換部分であり、十分詳細なコード生成を行うにはそれに相当する入力モデルが必要となるが、何らかの標準化が必要となる。

(6) 実用性検討

エンタプライズアーキテクチャに基づくシステム記述例について検討を行った。変化に対応するエンタプライズアーキテクチャとして準備した UML Profile を用い、有効な記述が出来ることを確認した。拡張例としてあげた五つのケースのうち四つのケースで実用性を検証した。唯一 SoaML との統合のケースで、メタモデルが UML

Profile 定義を前提とし UML メタモデルの拡張という形をとっている部分があることで、直接的な継承による接続は行えず、二つのモデルを併記し対応する要素を関連づけるに留まった。しかし SoaML と RM-ODP のエンタプライズ言語の場合、実質的には類似する概念が共通の親要素から派生しているケースに相当している。このような場合、類似概念ごとに対応関係を定義し、モデルレベルで関連付けを明記するような対応になると同時に、モデル変換では共通部分と個別部分に分けるような対応となる。コード生成という意味では、ソフトウェアの構造に関わるコード生成は比較的容易であるが、振る舞いに関わる部分はモデル側が標準だけでは十分な記述が出来ないため、抽象レベルを下げた DSL を試作した。適切な抽象レベルや作成分担については今後の課題となる。

なお、モデル駆動開発について次のような批判的見方があることは十分認識している。

組み込み分野での状態遷移駆動のシステムでは例があるが、モデル駆動開発を支援するツール類で、エンタプライズアーキテクチャで整理が必要になるようなシステムに適用され広く知れ渡っている例は聞こえてこない。その要因は次のようなものではないか。

A)　モデルとコードの間のギャップが大きい

エンタプライズレベルのモデルはシステム全体を捉えることが目的の一つであるため、かなり抽象度の高いモデルになっている。そこからのコード生成は困難。

B)　モデル作成が容易でない

これはエンタプライズシステムの複雑さや仕様の曖昧さが要因にあり、その中でツールや実行環境にあわせた最適のモデルを作成するのが困難という意味と解釈できる。

C)　モデル変換が完全でなくどうしてもコードを書かざるを得ない

そのため反復開発を行っていると、２度目以降のコード生成で手書きコードの扱いが難しくなる。

しかしながら、エンタプライズアーキテクチャの必要性を認識するような大規模なシステムでは、将来にわたって拡張出来る、知的財産としてのモデルを作成することが重要であり、それを十分に活用するためにもツールの整備によるモデルベースでのシステム維持開発の仕組み作りが重要な課題となる。

(7) 今後の課題

今回の研究ではエンタプライズアーキテクチャとして RM-ODP を利用したが、

TOGAF や DoDAF/MODAF もかなり仕様が公開されており、本研究のアプローチが異なるエンタプライズアーキテクチャでも適用出来ることの検証は今後の課題である。

メタモデル要素の分析をモデリング技術の世界で行ったが、分析でのオントロジ技術の活用はもう一つのアプローチであり、その検証も今後の課題である。

モデリングを活用したソフトウェア開発では、モデルに基づくコード生成とは別に、モデルを直接解釈実行する形態が考えられ、その方向の検討も今後の課題である。

標準化に関して、RM-ODP は ISO 標準・ITU-T 勧告であり、本論文の提案を標準化の場に持ち上げることが考えられる。課題としては、本提案はアドホックな拡張を生む可能性があるため、正式な標準修正手順を尊重する各国標準化参加メンバの合意を取り付けることは困難かもしれないことがある。ただ、同じく標準の UML 言語も言語拡張メカニズムを持っていること、また今日のように技術変化が激しい時代に数年を必要とする標準化プロセスが標準拡張に適したメカニズムなのか、といった点につき議論を進めてゆきたい。

参考文献

[1] Zachman, J.A., A Framework for Information Systems Architecture. IBM Systems Journal, Vol 26 (3) 1987

[2] FEA Reference Models, Consolidated Reference Model Version 2.3, http://www.whitehouse.gov/sites/default/files/omb/assets/fea_docs/FEA_CRM_v23_Final_Oct_2007_Revised. pdf.

[3] Federal Enterprise Architecture (FEA), http://www.whitehouse.gov/omb/e-gov/fea

[4] TOGAF, The Open Group, http://www.opengroup.org/togaf/

[5] The Open Group, ArchiMate®, http://www.opengroup.org/subjectareas/enterprise/archimate

[6] DoDAF, Dodd Application Framework, http://dodcio.defense.gov/dodaf20.aspx

[7] MODAF, MOD Application Framework, https://www.gov.uk/mod-architecture-framework

[8] 総務省, 業務・システム最適化の推進, http://www.soumu.go.jp/main_sosiki/gyoukan/kanri/a_01-02.html

[9] ISO/IEC 10746-2:2009, Information technology -- Open distributed processing -- Reference model: Foundations

[10] ISO/IEC 10746-3:2009, Information technology -- Open distributed processing -- Reference model: Architecture

[11] ISO/IEC 15414:2006, Information technology -- Open distributed processing -- Reference model -- Enterprise language

[12] ISO/IEC 19793:2008, Information technology -- Open Distributed Processing -- Use of UML for ODP system specifications

[13] ISO/IEC/IEEE 42010:2011, Systems and software engineering -- Architecture description

[14] Peter Linington, Zoran Milosevic, Akira Tanaka, A. Vallecillo. "Building Enterprise Systems with ODP -- An Introduction to Open Distributed Processing", Chapman & Hall/CRC Press, Sep 2011. (ISBN: 978-1-4398-6625-2)

[15] Janis Putman, “Architecting with RM-ODP”, Prentice Hall, 2001. (ISBN 0-13-019116-7)

[16] H. Kilov, P. Linington, J.R. Romero, A. Tanaka, A. Vallecillo. “The Reference Model of Open Distributed Processing: Foundations, experience and applications”. In Computer Standard & Interfaces 35(3): 247-256, March 2013

[17] J.R. Romero, J.I. Jaén, A. Vallecillo. “A Tool for the Model-Based Specification of Open Distributed Systems”. The Computer Journal, 2012. doi: 10.1093/comjnl/bxs021

[18] RM-ODP wiki, http://rm-odp.wikispaces.com/Main+Page

[19] 田中, オープン分散処理標準（RM-ODP）の概要と使い方, http://www.slideshare.net/at_husky/rm-odp

[20] 藤長昌彦,加藤聰彦,鈴木健二:ODP ビューポイントに基づく分散システム設計法とその通信システムへの適用,マルチメディア通信と分散処理 63-13 (1994)

[21] 中川路哲男, 田中 明, 浅野正一郎: "開放型分散処理の標準化の概要", 電子情報通信学会誌, Vol.77, No.3, pp.277-287, (1994)

[22] INTAP, UML による ODP システム記述ガイド, 平成 19 年度報告, https://sites.google.com/site/intapodpalumni/reports/01_UML による ODP システム記述ガイド.pdf?attredirects=0&d=1

[23] INTAP, 日本版 EA 対応 UML Profile に関する研究報告書, 平成 17 年度報告, https://sites.google.com/site/intapodpalumni/reports/01_日本版 EA 対応 UML Profile に関する研究報告書.pdf?attredirects=0&d=1

[24] INTAP, RM-ODP と UML Profile for EDOC の適用ガイド, 平成 15 年度報告, https://sites.google.com/site/intapodpalumni/reports/01_RM-ODP と UML Profile for EDOC の適用ガイド.pdf?attredirects=0&d=1

[25] Daisuke Hashimoto, Hiroshi Miyazaki, Akira Tanaka, “UML 2 Models for ODP Engineering/Technology Viewpoints”, Workshop on ODP Enterprise Computing (WODPEC 2005)

[26] Daisuke Hashimoto, Akira Tanaka, Masanori Yokoyama, “Case study on RM-ODP and Enterprise Architecture”, Workshop on ODP Enterprise Computing (WODPEC 2007)

[27] Akira Tanaka, Yoshihide Nagase, Yasuo Kiryu, Kanji Nakai. “Applying ODP Enterprise Viewpoint Language to Hospital Information Systems”, EDOC Conference 2001

[28] xODP: The MagicDraw plugin for RM-ODP and UML4ODP, http://www.uco.es/~in1rosaj/tool_mdplugin.html

[29] MDG Technology for ODP, Sparx Systems, http://www.sparxsystems.com.au/products/3rdparty/odp/index.html

[30] Lea Kutvonen, "Using the ODP reference model for Enterprise Architecture", Workshop on ODP Enterprise Computing (WODPEC 2007)

[31] OMG, Unified Modeling Language™ (UML®), http://www.omg.org/spec/UML/

[32] James Rumbaugh, Ivar Jacobson, Grady Booch, "The Unified Modeling Language Reference Manual, Second Edition", Addison Wesley, 2005

[33] OMG, Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, http://www.omg.org/spec/QVT/1.1/PDF/, January 2011

[34] OMG, XML Metadata Interchange (XMI®), http://www.omg.org/spec/XMI/

[35] OMG, Meta Object Facility (MOF) Core, http://www.omg.org/spec/MOF/

[36] OMG Business Motivation Model, http://www.omg.org/spec/BMM/1.0/

[37] OMG, Unified Profile for the Department of Defense Architecture Framework (DoDAF) and the Ministry of Defense Architecture Framework (MODAF) [UPDM], http://www.omg.org/spec/UPDM/

[38] UML Profile For Enterprise Application Integration (EAI), http://www.omg.org/spec/EAI/1.0/

[39] UML Profile For Enterprise Distributed Object Computing (EDOC), http://www.omg.org/spec/EDOC/1.0/

[40] OMG, Model Driven Architecture (MDA), http://www.omg.org/mda

[41] OMG, MDA Guide, http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf

[42] OMG, MOF Model To Text Transformation Language, http://www.omg.org/spec/MOFM2T/1.0/PDF

[43] David Frankel, Model Driven Architecture – Applying MDA to Enterprise Computing, OMG Press, 2003

[44] David Frankel & John Parodi, Editors, The MDA Journal, Meghan-Kiffer Press 2004

[45] Andrey Sadovykh, Philippe Desfray, Brian Elvesæter, Arne-Jørgen Berre, Einar Landre. "Enterprise Architecture Modeling with SoaML using BMM and BPMN – MDA Approach in Practice", Software Engineering Conference (CEE-SECR), 2010

[46] GenMyModel, http://www.genmymodel.com

[47] Eclipse Foundation, Eclipse Modeling Framework (EMF), http://www.eclipse.org/EMF/

[48] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, Ed Merks, “EMF Eclipse Modeling Framework”. Addison Wesley 2009

[49] Eclipse Foundation, Graphical Modeling Framework (GMF)

[50] Eclipse Foundation, Papyrus UML, http://www.eclipse.org/papyrus/

[51] Richard Gronback, “eclipse Modeling Project”. Addison Wesley 2009

[52] Eclipse Foundation, Epsilon, http://www.eclipse.org/epsilon/

[53] Eclipse Foundation, Model to Model Transformation (ATL and QVT), http://www.eclipse.org/mmt/

[54] Eclipse, ATL Transformations, http://www.eclipse.org/m2m/atl/atlTransformations/

[55] Jouault F., Allilaire F., Bezivin J., Kurtev I., ATL: A model transformation tool, Science of Computer Programming, Volume 72, Issues 1-2, Special Issue on Second issue of experimental software and toolkits (EST) (2008)

[56] Eclipse Foundation, Xtext - Language Development Framework, http://www.eclipse.org/Xtext/

[57] Eclipse Foundation, Xtend, http://www.eclipse.org/xtend/index.html

[58] 田中、細合、テキスト型 DSL 開発フレームワーク Xtext 入門、http://www.beta-publish.com/books.html

[59] Eclipse Foundation, Model To Text (M2T) (including Xpand), http://www.eclipse.org/modeling/m2t/

[60] Eclipse Foundation, Acceleo - transforming models into code, http://www.eclipse.org/acceleo/

[61] ModelBus project, http://www.modelbus.org/modelbus/

[62] Eclipse Modeling 勉強会, https://sites.google.com/site/eclipsemodelingsigjapan/re

[63] Xtext Users Japan, http://sites.google.com/site/xtextusersjapan/files

[64] National Institute of Standards and Technology, “The NIST Definition of Cloud Computing,” Special Publication 800-145, 2011

[65] V. Grassi, R. Mirandola, A. Sabetta, A UML Profile to Model Mobile Systems, «UML» 2004 – The Unified Modeling Language

[66] M. Wu and B Unhelkar, Extending Enterprise Architecture with Mobility,

2008
[67] C. Bouanaka, F. Belala, Towards a Mobile Architecture Description Language, 2008
[68] Khawar Hameed1, et al. An Enterprise Architecture Framework for Mobile Commerce, 2010
[69] Jacques FERBER, Olivier GUTKNECHT, “A meta-model for the analysis and design of organizations in multi-agent systems”, International Conference on Multi Agent Systems, 1998.
[70] Dreyfus, D and Iyer, B, “Enterprise Architecture: A Social Network Perspective,” 39th Hawaii International Conference on System Sciences, 2006
[71] facebook metamodel, Source: http://commons.wikimedia.org/wiki/File:Metamodel_of_Facebook.jpg
[72] H. Van Dyke Parunak and James J. Odell, Representing Social Structures in UML
[73] OMG, Business Process Model And Notation (BPMN) Version 2.0 http://www.omg.org/spec/BPMN/2.0/PDF/
[74] Eclipse Foundation, Stardust - Comprehensive Business Process Management For Eclipse, http://www.eclipse.org/stardust/
[75] Bruce Silver, BPMN METHOD & Style, CODY-CASSIDY PRESS 2011
[76] OMG, Service oriented architecture Modeling Language (SoaML®)
[77] OASIS, Reference Model for Service Oriented Architecture 1.0, http://docs.oasis-open.org/soa- rm/v1.0/soa-rm.pdf.
[78] OASIS, Reference Model for Service Oriented Architecture 1.0, http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf.
[79] Eclipse SoaML plug-in, http://alarcos.esi.uclm.es/MINERVA/TOOLS/soamlPlugin.htm
[80] Thomas Erl, “SOA Design Patterns”, Prentice Hall, 2009
[81] Fowler M., Domain-Specific Languages, Addison- Wesley, 2011
[82] Kleppe A., Software Language Engineering, Addison- Wesley, 2009
[83] Thomas Stahl, Markus Volter, Model-Driven Software Development, Wiley, 2005
[84] Fowler, Patterns of Enterprise Application Architecture
[85] Kleppe A., Software Language Engineering, Addison-Wesley, 2009

[86] Markus Voelter, DSL Engineering, dslbook.org, 2013
[87] Dave Thomas, David Heinemeier Hansson, “Agile Web Development with Rails”. The Pragmatic Bookshelf, 2005.
[88] Marco Brambilla, Jordi Cabot, Manuel Wimmer, “Model-Driven Software Engineering in Practice”, Morgan & Claypool Publishers, 2012
[89] MetaCase, MetaEdit+, http://www.metacase.com/ja/
[90] Obeo Designer, Obeo, http://www.obeodesigner.com
[91] Ruby on Rails, http://rubyonrails.org
[92] Grails, http://grails.org
[93] A. Tanaka and O. Takahashi, “Experimental Transformations between Business Process and SOA models”, International Journal of Informatics Society (IJIS) ISSN: 1883-4566, Volume: 4, No: 2, Date: August 2012, pp.93-102
[94] Akira Tanaka and Osamu Takahashi, “Extension Mechanism for Integrating New Technology Elements into Viewpoint based Enterprise Architecture Framework”, International Workshop on INformatics, 2013
[95] Trygve M. H. Reenskaug, Preliminary UML_VM, http://heim.ifi.uio.no/~trygver/themes/umlVM/umlvm-index.html
[96] NII, Bidirectional Computation, http://grace-center.jp/research/research_projects/prj_bimt-html
[97] Jos Warmer and Anneke Kleppe, “Object Constraint Language”, Pearson Education, Inc. 2003
[98] Terence Parr, “ANTLR Reference”, The Pragmatic Bookshelf, 2007
[99] Jack Herrington, “Code Generation in Action”, Manning, 2003
[100] SPIN, Formal Verification, http://spinroot.com/spin/whatispin.html

謝辞

本研究に取り組むにあたり，最後まで親身なご指導と励ましを賜りました指導教官である公立はこだて未来大学システム情報科学部情報アーキテクチャ学科高橋修教授に心より感謝いたします。更に、本論文の審査を引き受けて頂きました、同じく公立はこだて未来大学システム情報科学部情報アーキテクチャ学科の藤野雄一教授，姜暁鴻教授，大場みち子教授、そして湘南工科大学工学部情報工学科大谷真教授に心より感謝致します。また、IWIN の場で貴重なアドバイスを頂いた先生や研究者の方々に感謝申し上げます。

ほとんど函館にうかがう機会はありませんでしたが、目に見えない形で支えて頂いた高橋修研究室の皆様に感謝いたします。

最後に、この研究期間を支えてくれた家族に感謝します。

本研究に関する論文発表

筆頭論文

1. 論文誌

1) A. Tanaka and O. Takahashi, “Experimental Transformations between Business Process and SOA models,” International Journal of Informatics Society (IJIS), ISSN: 1883-4566, Volume: 4, No: 2, Date: August 2012, pp.93-102

2. 国際会議

1) Akira Tanaka and Osamu Takahashi, “Extension Mechanism for Integrating New Technology Elements into Viewpoint based Enterprise Architecture Framework,” International Workshop on Informatics, September 2013, pp.71-79

2) Akira Tanaka and Osamu Takahashi, “Experimental Transformations between Business Process and SOA models,” International Workshop on Informatics, September 2011, pp.105-113

3. 口頭発表

1) 田中明, 高橋修,「ビューポイント DSL を用いたシステム使用記述に関する考察」, 情報処理学会ソフトウェア工学研究会, 2010-SE-168(13),1-8 (2010-05-25)

2) 田中明, 高橋修,「ビューポイントに基づくシステム仕様記述に関する考察（１）」, 情報処理学会マルチメディア通信と分散処理（DPS）研究会, 2009-DPS-140(4),1-8 (2009-09-03)

共著論文

1. 論文誌

1) H. Kilov, P. Linington, J.R. Romero, A. Tanaka, A. Vallecillo. “The Reference Model of Open Distributed Processing: Foundations, experience and applications”. In Computer Standard & Interfaces 35(3): 247-256, March 2013

2) 中川路哲男, 田中 明, 浅野正一郎: "開放型分散処理の標準化の概要", 電子情報通信学会誌, Vol.77, No.3, pp.277-287, 1994

2. 国際会議

1) Hiroshi Miyazaki and Akira Tanaka, “Case Study on Human/System Interaction Specification using UML for ODP, ” Enterprise Distributed Object Computing Conference Workshops, 2008, pp. 413-421, September 2008

2) Daisuke Hashimoto, Akira Tanaka, Masanori Yokoyama, “Case study on RM-ODP and Enterprise Architecture”, Workshop on ODP Enterprise Computing (WODPEC 2007), pp.1-8, October 2007

3) Hiroshi Miyazaki and Akira Tanaka, “Study on representation of security aspects in each viewpoint using UML for ODP,” Workshop on ODP Enterprise Computing (WODPEC 2007), pp.38-44, October 2007

4) Daisuke Hashimoto, Hiroshi Miyazaki, Akira Tanaka, “UML 2 Models for ODP Engineering/Technology Viewpoints”, Workshop on ODP Enterprise Computing (WODPEC 2005), pp.32-38, September 2005

5) Akira Tanaka, Yoshihide Nagase, Yasuo Kiryu, Kanji Nakai, "Applying ODP Enterprise Viewpoint Language to Hospital Information System," 2001 5th IEEE International Enterprise Distributed Object Computing Conference, pp. 188-192, Fifth IEEE International Enterprise Distributed Object Computing Conference, September 2001

3. 著作

1) Peter F.Linington, Zoran Milosevic, Akira Tanaka, Antonio Vallecillo, “Building Enterprise Systes with ODP,” Chapman & Hall/CRC Press, September 2011. ISBN: 978-1-4398-6625-2

2) 田中 明, 細合晋太郎, “テキスト型 DSL 開発フレームワーク Xtext 入門”（電子ブック）,（株）テクノロジックアート, http://www.beta-publish.com/, 2012 年 4 月

3) 田中　明, ”情報処理ハンドブック：9 章 8.3 開放型分散コンピューティング, ” オーム社, pp.1108-1112, 1995 年 11 月

4. 特許

1) 田中　明, 小川　晶子, 特開 2004-258823, “ビジネスプロセス処理方法およびシステム並びにその処理プログラム”

2) 田中　明, 特開昭 63-213013, “計算機ネットワークシステムの時刻指定方式”