ゴール指向要求分析駆動によるUML設計手法

- ゴール指向ユースケースモデリングとロバストネス分析 -

本田 耕三

電気通信大学大学院情報システム学研究科

博士（工学）の学位申請論文

2017年3月

ゴール指向要求分析駆動によるUML設計手法

- ゴール指向ユースケースモデリングとロバストネス分析 -

博士論文審査委員会

主査　　大須賀 昭彦　教授

委員　　田中 健次　教授

委員　　田原 康之　准教授

委員　　古賀 久志　准教授

委員　　石川 冬樹　客員准教授

著作権所有者

本田 耕三

2017年3月

Goal-Oriented Requirements Engineering Driven UML Design Approach

- Goal-Oriented Use case Modeling and Robustness Analysis -

Kozo Honda

Abstract

The software development process progresses from requirements engineering on the most upper stream to the design process. Requirements engineering requires to identify what to need exhaustively, consistently, and unambiguously in the user's perspective. Meanwhile, the design process defines how to realize the structure and behaviors of the system for the user requirements, and then realize them in the designer's perspective. Therefore, the applicability and inter-traceability are needed between them.

In requirements engineering and the design process, Goal oriented requirements analysis methodology and Object oriented design methodology with UML are available respectively. Goal oriented requirements analysis methodology elicits the requirements systematically, logically, and with reasoning without depending on designer's knowledge, heuristics, and inspiration. But, in the requirements engineering side, the work for reflecting systematically the requirements definition by goal-oriented definition process to design and implementation process is depends on the designer for many parts, e.g. the designer is required to understand the requirements definition model and reflect them to design process manually. Meanwhile, in the design process side, some object oriented design and implementation methodologies are applicable to reflect the requirements defined to the design and implementation process for practical projects, but the eliciting and analyzing the requirements systematically are not put importance on. Therefore, the gap which the reflecting the requirements analysis model to design start point model causes, i.e. the leakage of the requirements information when the reflection, is recognized as important problem.

From the above descriptions, building the approach which reflects the requirements de-

fined systematically and logically to the design and implements for them is one of the fundamental problems for software development. The approach which reflects requirements definition information between goal oriented requirements analysis model and use case driven object oriented design process model is difficult to build because their models have different definition each other.

The approach proposed in this study aims at resolving this problem. This study adopts goal oriented requirements analysis methodology KAOS, UML driven object oriented design and implementation process ICONX for the requirements engineering, the design implementation process, respectively. The transformation processes based on the meta-models from KAOS model to use case model which is start point model of ICONIX process or robustness diagram which is preliminary design model are proposed. The proposed approach is the systematic one as I avoid manual process as possible.

I confirmed that the proposed approach was available one through evaluating a case study on ATM system in the United States and the international airline reservation system. The case of ATM system in the United States shows the exclusion effect from the dependency on the designer and the inheritance of refinement patterns when the transformations. The case of the international airline reservation system also shows the matching well the robustness diagrams to the KAOS model.

The advantages of the proposed approach are as follows. Through the adoption of useful existing method, the usability of such methods and the mitigation of the load for the familiarizing the designer with new methods become available. Furthermore, the mitigation of the influences from heuristics and so on is expected by the exclusion effect from the dependency on the designer. The inter-traceability between both models is also expected through the model transformation with refinement patterns inherited.

ゴール指向要求分析駆動による UML 設計手法

- ゴール指向ユースケースモデリングとロバストネス分析 -

本田 耕三

概要

ソフトウェア開発は，最上流の要求定義工程から設計工程へと進む．要求定義工程では，ユーザの視点で何が必要かを，網羅的に矛盾なく，曖昧さを排除して定義することが求められる．一方，設計工程では，要件定義に対応するユーザの視点から見たシステム内部の構造と振舞いを，“どう実現するか”について決定し，設計者の視点で具体化していく．このように，要求定義と設計の要素間には，論理的根拠に基づく妥当性と相互追跡性が必要となる．

要求定義工程，設計工程にはそれぞれゴール指向要求分析手法，UML によるオブジェクト指向設計手法という有効な手法が存在する．ゴール指向要求分析手法は，設計者の知識，経験，発想力などに過度に頼ることなく，要求を体系的，論理的，明確な根拠のもとに抽出する．しかし，要求定義工程から見ると，ゴール指向による要求定義を体系的に設計に反映し実装する仕組みは，設計者による要求定義モデルの理解と設計への反映を必要とするなど，設計者に依存する部分が多く課題が残っている．

逆に，設計工程から見ると，要求定義を設計に反映し実装する仕組みは実践的に使用されているものがあるが，その要求を体系的に抽出し分析した成果とすることに重点は置かれていない．

そのため，抽出された要求を，これらの要求分析モデルから設計工程の入力定義モデルに反映するときに発生するギャップ，すなわち抽出した要求情報が漏れてしまうことが，問題となっている．このことから，論理的，体系的に抽出した要求定義を設計に反映し実装する仕組みを構築することは，ソフトウェア開発における基本的な課題のひとつであると考えられる．

ゴール指向要求分析手法とユースケース駆動オブジェクト指向設計プロセスという異なった手法に基づくモデル間で，要求定義情報を体系的に反映するアプローチは，モデルの定

義が異なることから一般的には困難である．本研究で提案するアプローチはこの課題の解決を目指すものである．要求定義工程，設計・実装工程において，それぞれ実績あるゴール指向分析手法 KAOS，UML によるオブジェクト指向設計・実装プロセス ICONIX を活用する．

KAOS ゴールモデルは，トップゴールとしてシステムの最終的な目標を設定し，AND/OR グラフを用いてそれを論理的に詳細化する．詳細化は体系的に実施され，システムに対する要求がリーフゴールとして抽出される．ICONIX プロセスは，ドメインモデルとユースケースモデルによる要求定義を出発点として，ユースケース駆動による設計・テストまでを範疇とする実践的なプロセスである．

ユースケースモデル，ロバストネス図は，それぞれ ICONIX プロセスの要求定義モデル，予備設計モデルである．KAOS の成果物であるゴールモデルからこれらのモデルへの，変換テンプレートを介した，両者のメタモデルに基づく，変換アプローチを提案する．

KAOS ゴールモデルで暗黙的に表現された振舞いを明示的に抽出し，ユースケースモデルやロバストネス図へ如何に効率よく継承するかが具体的な課題のひとつである．また，ゴールモデルによる振舞いは要求として分析・定義されたものであり，要求定義工程からの一貫した要求の情報として，設計者によって変更されることなく，ユースケースモデルやロバストネス図に反映されるべきものである．すなわち，属人性を排した継承とする必要がある．これがもうひとつの具体的な課題となる．これらふたつの課題に対し，変換テンプレートを介して規則的に変換できるように工夫した．

戦術的な振舞いのシナリオに基づいたゴール分解方法の一般的表現として洗練パターンが知られている．基本的な振舞いの AND 分解 6 パターンからなり，これらの振舞いを暗黙的・暗示的に表現する．変換テンプレートは，洗練パターンによる戦術的な振舞いのシナリオを，それぞれのモデルの規則に従って明示的に定義したものである．洗練パターンに準じた一般的な表現になっているが，振舞いのシナリオを明確に定義するためにそれぞれのモデル要素を規則に従って構成している．

変換元モデルの変換テンプレートを変換先モデルの変換テンプレートにマッピングし，さらにモデル要素をマッピングすることによって，振舞いのシナリオを規則的に効率よく体系的に継承できる．これによって，振舞いの効率的な継承と変換における属人性の排除

を実現した．

米国の ATM システムや国際航空券予約システムを事例とした適用実験の結果，提案アプローチの有用性を確認した．米国 ATM システムの事例では，基準のユースケースモデルに対する適合率や再現率の評価結果による属人性の排除効果や，変換における洗練パターンの継承を確認した．また，国際航空券予約システムの事例では，KAOS モデルとロバストネス図との対応が正しく取られていることを確認した．

この結果，実績ある既存手法を採用できることで，それら手法の有用性活用，設計者の新手法習熟に対する負荷の軽減が可能となる．さらに，モデル変換における属人性の排除により，経験則等の影響軽減を期待できる．また，洗練パターンを継承したモデル変換により，両モデル間の追跡性確保を期待できる．

目 次

図 目 次

表 目 次

第1章　序論

本章では，本研究の背景を述べた後，本論文の目的と貢献を説明する．その後，本論文の構成について述べる．

1.1　本研究の背景

ソフトウエア開発の最上流工程である要求定義工程では，ユーザの視点で何が必要かを，網羅的に矛盾なく（一貫性），曖昧さを排除して定義することが求められる [55, 39]．要求間の整合性等の一貫性を保つことは，機能，性能，運用方法などに矛盾のないシステムを構築する上で重要であり，加えて要求定義においては根拠と追跡可能性，完全性，さらに明確な優先度が必要とされる [8, 39]．あわせて，外部環境とシステムとの境界を確定することによって，その実現手段を構築する上での前提を明確にする [47]．一方，要求定義工程を引き継ぐ設計工程では，要件定義に対応するユーザの視点から見たシステム内部の構造と振舞いを，“どう実現するか”について決定し，開発者の視点で具体化していく [55, 16]．

このように，要求定義工程と設計工程それぞれの要件を考え合わせると，要求定義から設計への移行は，ユーザの視点で定義した要求をシステム内部の構造と振舞いに置換え，その実現方法を具体化することである．設計の出発点は，要求の実現を可能とする内部要素に実現すべき要求を割付けることであり，要求定義と設計の要素間には，論理的根拠に基づく妥当性と相互追跡性が必要となる [16, 55]．

要求定義工程と設計工程にはそれぞれ実績ある手法が存在する．要求定義工程には，要求を網羅的，論理的に抽出・分析して体系的に定義できると定評のあるゴール指向要求分析手法がある．KAOS[18], i*[38]，および NFR フレームワーク [6] はその代表的な手法である．KAOS は AND/OR グラフを用いて要求を論理的に抽出する．i*は，人，組織，システム，役割などをアクターと見做し，ゴール達成に関するアクター間の依存関係およびア

クター内の目的・実現手段関係と貢献・依存関係をネットワークで表現する．NFR フレームワークは，ソフトゴールの AND/OR 依存グラフにより親ソフトゴールの満足された度合いを評価し，対象システム設計の妥当性を判断する．このように，ゴール指向要求分析手法では発想力に過度に頼ることなく，抽象目標から論理的に要求を抽出できる．このようにして抽出した要求は，必要となる根拠や要求間の関係が体系的にモデル化されていて，その合理性が確保されている．しかし，こうして合理的に定義できた要求を，完全性を保って設計に体系的に反映して実装する仕組みは，まだ十分に確立してはいない．

設計工程では，要件定義から設計工程，実装工程までを幅広くサポートし，静的構造と振舞いの実現方法を効果的に記述する統一モデリング言語（以降，UML）[29, 41, 40] を用いたオブジェクト指向開発手法が広く利用されている．その多くは，ユースケースモデルやシナリオで記述された要求定義の中からオブジェクトを抽出し，その静的構造や振舞いをそれぞれクラス図やシーケンス図，コミュニケーション図，またはアクティビティ図などで表現することで，システムの実現手段へと詳細化している．実務的な場面ではプロジェクトチームの環境に合わせて独自にアレンジした設計プロセスを使うことが多いが，ICONIX プロセス [51, 35] や Rational Unified Process（以降，RUP）[15] のようなソフトウエア開発全般に渡る実践的開発プロセスとして提供されている手法の活用も増えている．ともに，ユースケース駆動によるプロセスである．ICONIX プロセスは，ドメインモデルとユースケースモデルによる要求定義情報を入力として，ユースケース駆動による設計・テストまでを範疇とする実践的なプロセスである．RUP も，ユースケースで要求分析を行い，ユースケースをベースに設計を進めるユースケース駆動によるソフトウエア開発プロセスであるが，チームによる反復開発が柱となっている．アーキテクチャをベースとしてビジネスモデリングから導入までを反復開発するソフトウエア開発のためのベストプラクティスである．ICONIX プロセスよりも少し大きな枠組みといえる．例えば，RUP において反復する具体的なプロセスとして，ICONIX プロセスやプロジェクトでアレンジしたプロセスを使うという運用が考えられる．

このように，要求をユースケースモデルで定義し体系的に設計・実装へと具体化する手法は，ICONIX プロセスや RUP のように確立されたものがあり，実務で活用されている．しかし，ユースケースモデルのみによる要求の抽出・分析は，要求の体系化や論理的根拠

の面で十分とは言えず，シナリオやCRCカード等を利用することが多い．この場合，体系化は難しく，その成果物の品質は発想力や直観力等個人の能力や経験に左右される．

以上述べたように，要求定義工程，設計工程にはそれぞれゴール指向要求分析手法，UMLによるオブジェクト指向設計手法という有効な手法が存在する．しかし，要求定義工程から見ると，ゴール指向による要求定義を漏れなく体系的に設計に反映し実装する仕組みには，知識や経験則など設計者に依存しているなど課題が残っている．逆に，設計工程から見ると，要求定義を設計に反映し実装する仕組みは実践的に使用されているものがあるが，その要求を体系的に抽出し分析した成果とすることに重点は置かれていない．そのため，抽出された要求をこれらの要求分析手法からユースケースモデルに反映するときに発生するギャップ，すなわち抽出した要求の情報が漏れてしまうことが，問題となっている．さらに，要求分析の成果をアーキテクチャの決定に反映させる方法も体系立てられているとは言えず，設計者の知識や経験に依っているところが大きい．

筆者ら [44] は，KAOS，UMLそれぞれによって要求分析から設計を実施し，その特徴を比較した．その結果，KAOSで要求定義工程を実施し，その結果をユースケースやロバストネス図に反映してUMLによる設計工程に移行することが効果的であるとの結論を得た．これらのことから，論理的，体系的に抽出した要求定義を設計に反映し実装する仕組みを構築することは，ソフトウエア開発における基本的な課題のひとつであると考える．そして，この基本的な課題を解決するための要件は，これまでの議論から以下のようになる．

1. 要求定義工程における体系的，論理的な要求の抽出・分析・定義
2. 設計・実装工程への入力である定義された要求を，設計から実装へと体系的に反映し具体化していく仕組み
3. 要求定義工程の成果を，設計・実装工程への入力である定義された要求に，体系的に反映する仕組み

また，要求定義工程から設計・実装工程にかけて，網羅性，整合性，一貫性（無矛盾性），完全性，または追跡可能性といった品質特性を保持することが重要となる．

1.2 本研究の目的と貢献

本稿で提案するアプローチは，1.1 節で議論した要件を満たすためのものである．体系的，論理的に抽出・分析・定義した要求で設計を駆動する．要件の 1. と 2. を満足させるために実績のある手法を利用する．具体的には，要求定義工程と設計・実装工程に，ゴール指向要求分析手法 KAOS と UML による設計・実装プロセス ICONIX をそれぞれ利用する．すなわち，要求定義工程と設計・実装工程それぞれに実績ある手法を採用することで，既存手法の有効性を活用し，新規手法習得に対する開発者の負荷を軽減する．

また要件 3. については，KAOS モデルをユースケースモデルやロバストネス図に体系的に変換するアプローチを提案することにより，KAOS の成果である分析されて定義された要求の情報を ICONIX プロセスの入力となる定義された要求に反映し，両モデル間の追跡性を確保する．具体的には，次の手順で KAOS の成果である要求モデルを ICONIX プロセスの入力モデル（ユースケースモデル）や予備設計モデル（ロバストネス図）に変換し，要求定義モデルの情報を ICONIX プロセスに反映する．

以下に提案アプローチの概要を示す．

1. KAOS のゴールモデル（要求モデル）を入力とし，操作モデルに変換する（責任モデル，オブジェクトモデルの要素を含む）．
2. 操作モデルをユースケース図に変換する．
3. ユースケースに相当するリーフゴール（要求）をシナリオを表現するサブゴールに洗練し，さらにシナリオを継承した操作モデルに変換する．
4. シナリオを継承した操作モデルをイベントフロー図に変換する．
5. シナリオを継承した操作モデルをロバストネス図に変換する．

提案アプローチの特徴は次の 2 点である．

- KAOS モデリングとユースケースモデル（ユースケース図，イベントフロー図）やロバストネス図への変換は，洗練パターンを根拠に作成した変換テンプレートを介して実施する．

- イベントに着目して，ユースケースにおけるアクターとシステム間のインタラクション（シナリオに相当する）を抽出し，システム機能の振舞いを特定する．

洗練パターンとは，Lamsweerde[20] が KAOS モデリングにおいて戦術的な振舞いのシナリオに基づいた洗練を実施するガイドとして推奨しているゴールの典型的な分解パターンである．典型的な振舞いの AND 分解 6 パターンからなり，これらの振舞いを暗黙的・暗示的に表現する．さらに，これら暗黙的・暗示的に表現された振舞いは，操作モデルで明示的にモデリングされる．

変換テンプレートは，洗練パターンによる戦術的な振舞いのシナリオを，それぞれのモデルの規則に従って明示的に定義したものである．洗練パターンに準じた一般的な表現になっているが，振舞いのシナリオを明確に定義するためにそれぞれのモデル要素を規則に従って構成している．この変換テンプレートをベースに KAOS モデルを作成することによって，ユースケースモデルやロバストネス図の要素へのマッピングを容易にする．また，操作の原因や結果に対応するイベントをキーとすることでインタラクション抽出の根拠となり，それに対応するシステム機能の振舞いを特定できる．

本研究の課題は，これらそれぞれの工程で有効性が確認され実務にも使用されている既存の手法を最大限に活用し，体系的・論理的に抽出・分析・定義された要求を漏れなく実現できるようなシステムを設計し実装する仕組みを構築することである．その課題の解決を図るために，KAOS による要求モデルを ICONIX プロセスの入力であるユースケースモデルや予備設計モデルであるロバストネス図に，体系的に変換する仕組みを構築することを目的とする．

提案アプローチでは，KAOS モデルのセマンティクスを，変換テンプレートを介して ICONIX プロセスに反映する．モデル間の変換を，それぞれのセマンティクスを含む変換テンプレートを介して実施することにより，変換テンプレートで意味づけられた要求モデルの情報（振舞いのシナリオ）を，それぞれのモデル間の変換で逐次継承しユースケースモデルやロバストネス図に反映する．すなわち，ゴールモデルを構成する洗練パターンを根拠にした変換テンプレートを媒体として，それぞれのモデル間の変換を逐次駆動することにより，ゴールモデルの体系と論理的根拠をユースケースモデルやロバストネス図に反映することができる．KAOS モデルの体系や情報は，追跡性が確保されながら，ユースケー

スモデルさらにロバストネス図に継承される．また，変換テンプレートを介したモデル間の変換それぞれは変換規則に従って実施されるため，変換の多くの部分を自動化でき，モデル作成における属人性の排除を期待できる．

本研究の貢献を纏めると以下のようになる．

1. 実績ある既存手法を最大限に活用できる仕組みを提案している．すなわち，体系的，論理的に要求を分析できるKAOSを要求定義工程に，設計から実装まで実務に効果的なICONIXプロセスを設計工程に活用する．

2. ゴールモデル（要求定義）の体系と論理的根拠を設計工程に反映している．

3. KAOSモデルを規則に基づいてユースケースモデルやロバストネス図に変換するため，多くの部分について自動化が期待できる．また，変換における属人性を低減している．

4. KAOSモデルで表現された振舞いのシナリオを，変換を通してユースケースモデルやロバストネス図に継承している．すなわち，KAOSモデルとユースケースモデルまたはロバストネス図間で情報の追跡性を確保している．

5. 米国のATMシステムや国際航空券予約システムを事例とした適用実験の結果，KAOSモデルをICONIXプロセスのユースケースモデルやロバストネス図に反映する提案手法の有用性が確認された．

1.3 本論文の構成

2章以降の内容について説明する．まず，2章では，既存の技術を説明する．最初に2.1節でゴール指向要求分析の代表的な手法のひとつであるKAOSを「遮断バーの安全踏切」のモデルを使って説明する．ここでは，KAOSを概説した後，ゴール洗練化の有効なガイドラインとなる洗練パターンについても紹介する．続いて，2.2節ではユースケースモデルとロバストネス図について，オブジェクト指向開発プロセスにおける位置付けと記述内容を説明する．そのあと，2.3節でユースケース駆動のオブジェクト指向開発であるICONIXプロセスのワークフローの概要について説明する．

次に，3 章では，ゴール指向による要求分析・定義モデルを設計工程に一貫性を持って反映させる試みについて議論している関連研究とその残る課題について紹介する．まず，3.1 節で KAOS での要求分析に基づいたオブジェクト指向設計へのアプローチについて言及し，さらに KAOS でのゴール分解への洗練パターンの適用について時相論理による記述手法を 3.2 節で説明する．ついで，ゴール指向要求分析のもうひとつの代表的な手法である i*による一貫した開発手法 Tropos について 3.3 節で説明する．

次の 4 章では，ゴール指向要求分析による要求定義モデルを設計工程の入力となるユースケースモデルおよびロバストネス図に変換する提案アプローチの全体像を説明する．まず，4.1 節で提案アプローチの全体的な流れを説明した後，4.2 節で提案アプローチのキーとなる変換テンプレートを説明する．

つづいて，5 章では，4.1 節で概説した変換アプローチの各 STEP について，新規に提案する変換アルゴリズムの説明と自動または自動後手動アルゴリズムの疑似コードによる具体化を行っている．5.1〜5.6 節で，それぞれ要求ゴールを含む一次の AND グラフ抽出，ゴールモデルから操作モデルへの変換，操作モデルからユースケース図への変換，ユースケース図の統合，操作モデルからイベントフロー図への変換，および操作モデルからロバストネス図への変換について説明する．さらに，5.7 節で変換アルゴリズムの適用範囲について議論する．

5 章に基づき,「遮断バーの安全踏切」を具体例として提案アプローチ全体の流れを 6 章で説明する．6.1〜6.9 節で，それぞれ要求ゴールを含む AND グラフ抽出，ゴールモデルからの操作モデル変換，操作モデルからのユースケース図変換，ユースケース図と KAOS モデル相互洗練，ユースケース図の統合，洗練パターンによる要求ゴール分解，ゴールモデルからの操作モデル変換，操作モデルからのイベントフロー図変換，および操作モデルからのロバストネス図変換を順を追って説明する．

次の 7 章では，提案アプローチの有効性を確認するため，ふたつの評価実験を実施した．7.1 節と 7.2 節で，それぞれ米国 ATM システムと国際航空券予約システムを事例として評価した結果について説明する．

最後に，8 章で結論を述べる．8.1 節で本研究をまとめ，8.2 節で今後の課題について述べる．

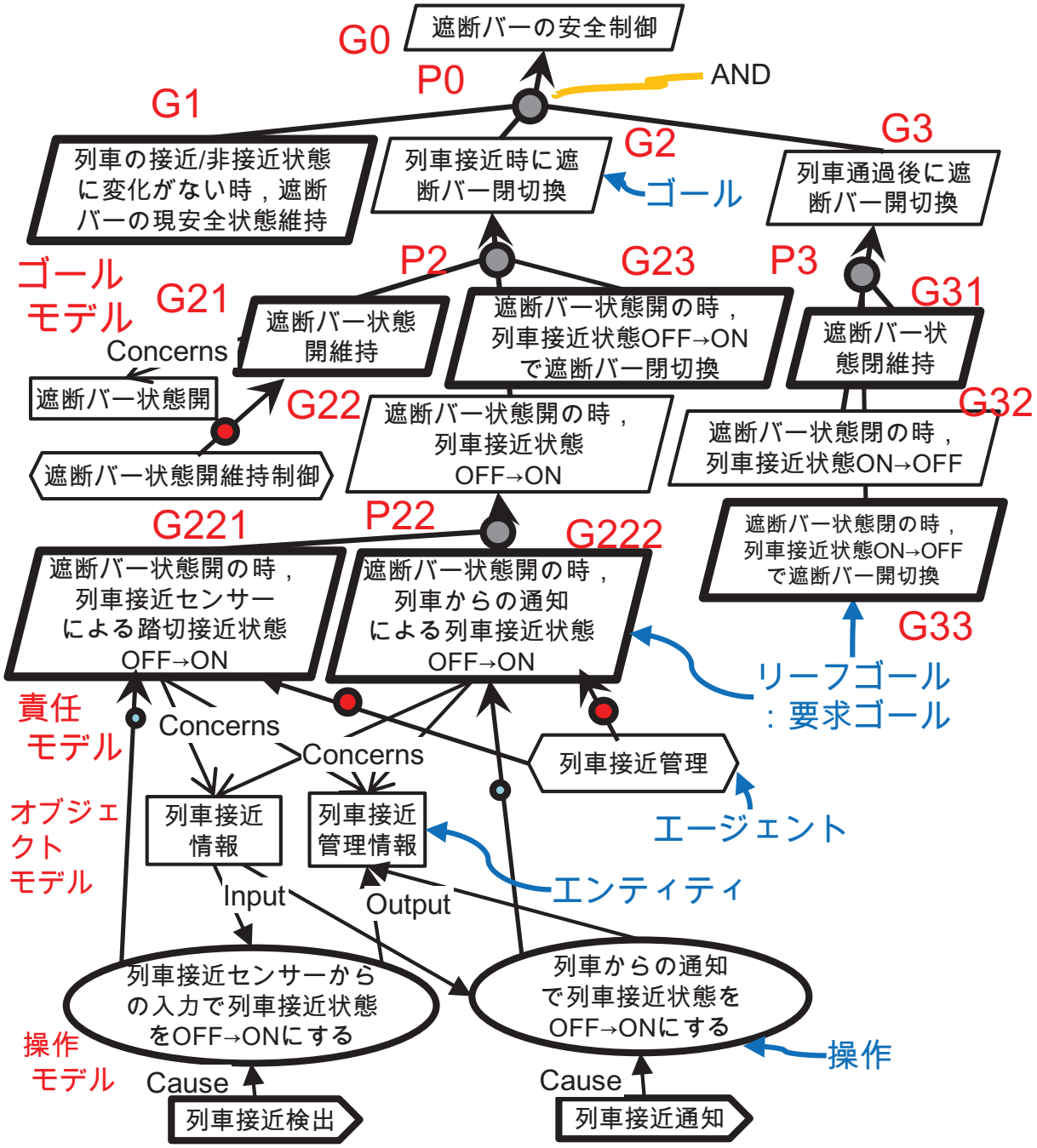
第2章　既存技術

本章では，ゴール指向ユースケースモデリングの核となる従来技術について説明する．まず，2.1 節で本論文の基盤であり実績ある要求分析手法として定評のあるゴール指向要求分析手法 KAOS と，その中でゴール分解の定型的なガイドラインとして使用される洗練パターンを紹介する．続いて，2.2 節では，オブジェクト指向設計の出発点に位置けられ，設計に対する入力，予備設計の結果としてそれぞれ使用される，ユースケースモデル，ロバストネス図を説明する．さらに 2.3 節で，実務的なオブジェクト指向設計プロセスである ICONIX を説明する．

2.1　ゴール指向要求分析手法 KAOS

KAOS はゴール指向要求分析の代表的な手法のひとつである．抽象的なゴールを最終目標に設定し，それを AND／OR グラフで洗練・詳細化することによってシステムに対する要求を分析・抽出する [18]．システムの構成員たるエージェントに達成の責務を割当てることができるリーフゴールが，システムに対する要求である．本稿では以降，説明のため，リーフゴールとして抽出した要求を要求ゴールと呼ぶ．KAOS は，ゴール（Goal），責任（Responsibility），オブジェクト（Object），操作（Operation）の基本 4 モデルで構成され，ゴールモデルを中心に要求を分析する．KAOS はゴールから操作要求を導くのに有効である [21]．また，要求モデリングのみならず設計モデリングにも踏み込んで使用されている [19, 22]．

KAOS モデルの例として，図 2.1 に鉄道踏切を安全に制御する踏切コントローラの一部である遮断バー安全制御のモデルを示す．踏切システムは踏切コントローラと遮断バーコントローラで構成される．踏切コントローラは，列車の接近／通過に基づき，遮断機の開閉をコントロールする遮断バーコントローラに遮断バー開閉の指示を出す．遮断バーの開



注： 1. 列車接近情報は，列車検出情報や列車接近通知情報を含む．
2. 列車接近管理情報は，列車接近状態ONを含む．
3. 一部省略．

図 2.1: 遮断バー安全制御の KAOS モデル例 (一部省略)

閉を安全に制御するためには，遮断バーの開閉を列車の接近／通過（非接近）に合わせて開閉すること，およびその開閉の状態を列車の接近／非接近の状態が変わらない限り維持することが必要になる．列車の接近／非接近に応じた遮断バーの安全な開閉制御に関する要求モデルであるが，本提案の説明において冗長と判断した部分は省略している．

図 2.1 を使って KAOS モデルの構成を概説する．まず，平行四辺形のゴールによるモデルがゴールモデルである．最終目標であるトップゴールを AND ／ OR で詳細化する．詳細化は大きい丸の付いた矢印で示される．階層的に詳細化した最下層の太枠の平行四辺形が，システムに対する要求となる．要求は，その実現に責任を持つ単一のエージェントを見つけることによって特定でき，その時点でゴールの詳細化は終了する．

次に，平たい六角形で示したエージェントを，中サイズの丸の付いた矢印で，実現責任を持つ要求ゴールに対応付けているのが責任モデルである．ゴールモデルと責任モデルを例で説明する．ゴール G22「遮断バー状態開の時，列車接近状態 OFF → ON」は，G221「遮断バー状態開の時，列車接近センサーによる踏切接近状態 OFF → ON」と G222「遮断バー状態開の時，列車からの通知による踏切接近状態 OFF → ON」という 2 つの要求ゴールに AND で詳細化され，列車接近管理エージェントがこれら 2 つの要求ゴール実現に責任を持つ．さらに，オブジェクト間の関連を示すのがオブジェクトモデルである．長方形はゴール実現に必要なエンティティを示す. 関係するゴールからは Concerns の矢印で関連付けられている．

最後に，図 2.1 の下層部分が操作モデルである．最下層にある楕円と五角形は，それぞれ要求ゴールを実現するための操作とその操作を起動するイベントである．操作はその上側にある長方形のエンティティを入出力しながら実行され，要求ゴールを実現する．その操作によって実現される要求ゴールは小さい丸の付いた矢印で関連付けられている．オブジェクトモデルと操作モデルを例で説明する．ゴール G221「遮断バー状態開の時，列車接近センサーによる踏切接近状態 OFF → ON」を実現するための操作「列車接近センサーからの入力で列車接近状態を OFF → ON にする」は,「列車接近検出」のイベントにより駆動され，エンティティ「列車接近情報」から情報を入力し，操作結果に基づいてエンティティ「列車接近管理情報」を更新する．

図 2.1 から分かるように，ゴールモデル，責任モデル，オブジェクトモデル，および操

表 2.1: ゴールモデルの洗練パターン一覧

洗練パターン名	説 明
マイルストーン駆動	・いくつかのマイルストーンを経由して目標状態に到達するパターン ・順番にマイルストーンを実現するサブゴールに分解する.
ケース分解	・複数の独立したサブ状態から成る目標状態を実現するパターン ・それぞれのサブ状態を実現するサブゴール（ケース）に分解する.
ガード条件導入	・ガード条件成立時に目標状態に到達できるパターン ・「ガード条件成立」,「ガード条件成立時に目標状態到達」,「現状態維持」に分解する.
分割・統治	・単純なサブゴールに分解してそれ毎に実現するパターン ・実現し易い単純なサブゴールに分解する.
モニタ不能駆動	・関心事のモニタ能力がない時にモニタ機能を外部に割振るパターン ・「代役からモニタ情報入手」と「モニタ情報によるゴール実現」のサブゴールに分解する.
制御不能駆動	・関心状態の制御能力がない時に外部に制御を割振るパターン ・「可能な制御情報での制御」と「可能な情報での制御依頼」のサブゴールに分解する.

作モデルは，関連するそれぞれの要素を含んでモデル化される．

ゴールの洗練化（ゴール分解）を的確に実施することは難しく，ある程度の訓練が必要である．サブゴールの粒度，詳細化の意味・意図，サブゴール間の相互関係など戦術を持って，一貫した洗練化を実施することが必要となる．無造作なゴールの洗練（分解）は，要求の抽出または分析における混乱を引き起こす．ゴールモデルの精緻化（洗練）において洗練パターンの利用は非常に効果的なガイダンスになると，Lamsweerde[20] は紹介している．それによると，洗練パターンはゴールによるモデルを具体的な仕様にインスタンス化する時の洗練や抽象化の一般的な指針となる．すなわち，マイルストーン駆動など 6 種類の戦術に基づいた典型的なゴール分解パターンからなり，表 2.1 に示される．洗練パターン名がそのまま戦術を表わしている．

洗練パターンは，パターン名称，適用可能条件，一次の AND グラフ（ゴール分解モデル）などによって構成されている．さらに，一次の AND グラフ（ゴール分解モデル）によるパターンは，状態を示すパラメータを使って一般的に定義され，そのパラメータのイン

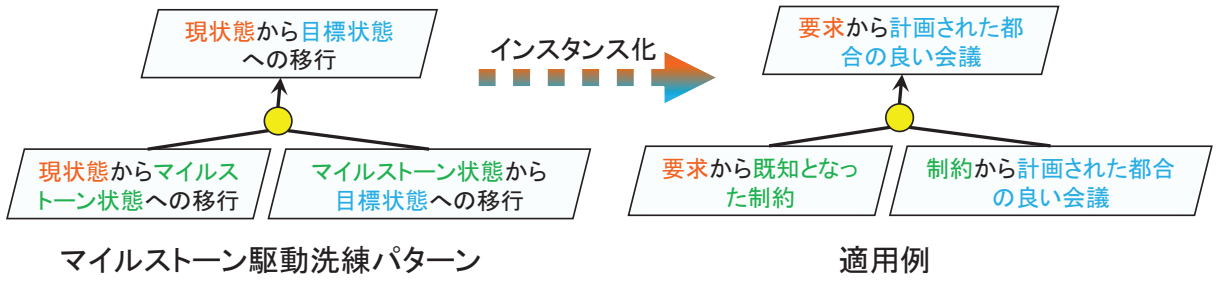


図 2.2: マイルストーン駆動洗練パターン適用例（出典 [20]）

スタンス化によって具体的なモデルに適用される．図 2.2 に，マイルストーン駆動洗練パターンの具体的なモデルへの適用例を示す．状態を示すパラメータとして，現状態, 目標状態, およびマイルストーン状態が記述されている．図の左側がマイルストーン駆動洗練パターンの AND グラフである．親ゴール「現状態から目標状態への移行」は，サブゴール「現状態からマイルストーン状態への移行」とそれを受けて実現されるサブゴール「マイルストーン状態から目標状態への移行」にマイルストーン駆動で分解されている．このマイルストーン駆動洗練パターンを具体的なモデルに適用するためにインスタンス化する．すなわち，現状態を会議開催の要求がなされた状態，目標状態を都合の良い会議が計画された状態，マイルストーン状態を制約が既知となった状態にそれぞれインスタンス化すると，親ゴール「要求から計画された都合の良い会議」をマイルストーン駆動洗練パターンを使って分解したサブゴール「要求から既知となった制約」と「制約から計画された都合の良い会議」が得られる．マイルストーン駆動以外の洗練パターンも同様に適用できる．

2.2 ユースケースモデルとロバストネス図

ユースケースモデルはユースケース図とユースケース記述で構成される．ユースケース図の構成要素は，サブジェクト，アクター，およびユースケースである．サブジェクトはそのユースケース図が対象としているシステムの範囲を矩形で示したものであり，アクターはそのシステムを利用したり利用されたりするシステム外部の存在（役割）をスティックマンで表現したものである．ユースケースはアクターに対するシステム機能（振舞い）を示し，楕円で表現される．ユースケースはサブジェクトの矩形の中に記述され，サブジェ

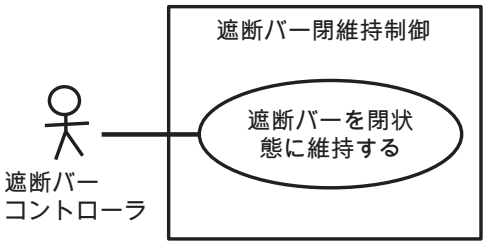


図 2.3: 遮断バー状態閉維持のユースケース図例

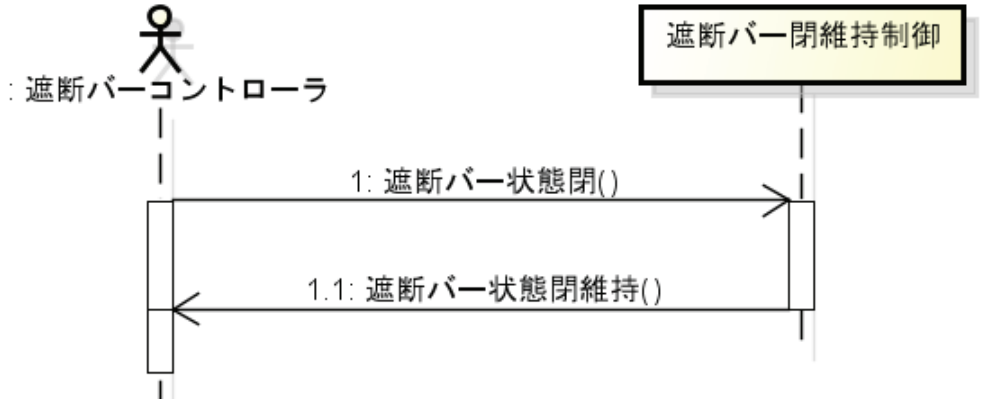


図 2.4: 遮断バー状態閉維持のイベントフロー図例

クトの外部に配置されたアクターと関連を示す直線で結ばれる．すなわち，ユースケース図はアクターとシステム機能とのインタラクションを示す．

ユースケース記述には，ユースケース図の説明としてユースケースのシナリオや事前・事後条件などが記述される．シナリオはイベントフローで表現されることが多く，ユースケース記述の中心的な情報である．本稿でも，ユースケース記述に該当するものとして，イベントフロー図を採用している．イベントフロー図は，イベントフローをシーケンス図の表記を使った図で記述したものである．図 2.3 と図 2.4 に，図 2.1 のゴール G31「遮断バー状態閉維持」に関するユースケース図とユースケース記述（イベントフロー図）の例をそれぞれ示す．

ユースケースモデルは，オブジェクト指向開発における要求定義モデルとして頻繁に使用される．例えば，代表的なオブジェクト指向開発手法として実務への適用実績も多い ICONIX プロセス [51] では，抽出した要求をユースケースモデルで定義してレビューする．

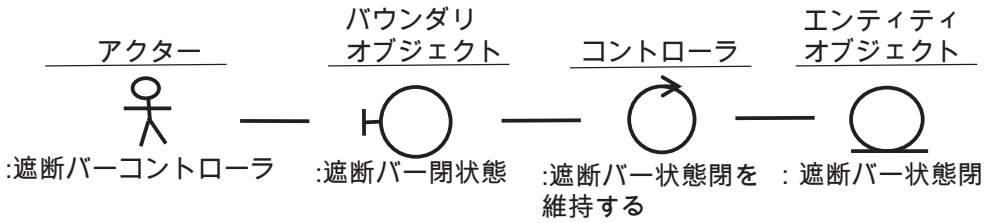


図 2.5: 遮断バー状態閉維持のロバストネス図例

しかし，ユースケースモデルは要求を利用者等のアクターとシステムとの間のインタラクションとして定義するものであり，要求を体系的，論理的に抽出する方法を提供するものではない．従って，要求抽出にはユースケースモデルよりも体系的，論理的に要求を抽出，分析できるゴール指向要求分析手法 KAOS の方がより適していると言える．

ロバストネス図は，ユースケース記述からオブジェクトと振舞いを抽出しその関係を図で表現したものである．ユースケース記述のイベントフローと 1 対 1 に対応させ，その内容を図に張り付ける．ユースケースごとに作成する．ユースケースモデルで定義された要求情報を設計モデルであるクラス図やシーケンス図に仲介するためのものであり，予備設計に含まれる [51].

イベントフローはアクターとシステム間インタラクションの流れを表現している．それと対応するロバストネス図も，システムがハンドリングするエンティティオブジェクト，システムとシステム外部とのインターフェイスであるバウンダリオブジェクト，システムの処理や機能を示すコントローラ，およびシステムとインタラクションを行うアクターなどのシンボルを使って，イベントフローに基づくシンボル間の関連を模式図的に表現する．ロバストネス図の例として，図 2.1 のゴール G31「遮断バー状態閉維持」に関するものを図 2.5 にを示す.

2.3 ICONIX プロセス

ICONIX プロセスはユースケースとコードの間に存在する領域にフォーカスした，最小限かつ効率的なアプローチであり，ユースケース駆動による UML を用いたオブジェクト指向分析／設計のプロセスである [33, 51]．抽象的かつ本質的なユースケースからコードを

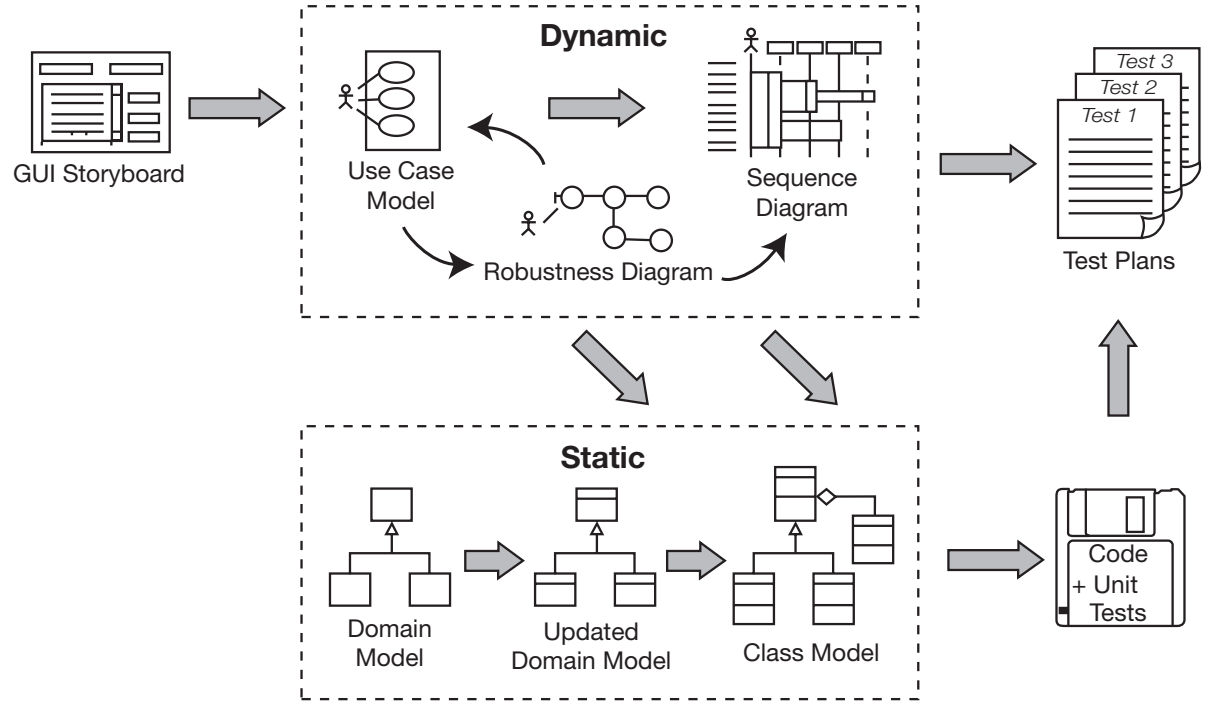


図 2.6: ICONIX プロセス　，出典 [33]

生成するためにはロバストネス分析が不可欠であり，それによって要求と設計を結ぶことができるとの考えに基づいており，多くの実務的なプロジェクトで実践されている．要求定義工程から設計工程への移行部分までを重点に，ICONIX プロセスの要点を説明する．

ICONIX プロセスの全体像を図 2.6 に示す．図 2.6 にあるように，ICONIX プロセスは動的な側面のワークフロー（上側の破線矩形部分：Dynamic）と静的な側面のワークフロー（下側の破線矩形部分：Static）からなり，お互いのプロセスの途中成果を反映させながら進めていく．動的な側面のワークフローの入力（出発点）はユースケースモデルであり，静的な側面のワークフローの入力（出発点）はドメインモデルである．動的な側面のワークフローの左側枠外にある GUI のストーリーボードは，ユースケースの根拠となる振舞い要求を抽出するためのツールとして使う．

ICONIX プロセスの開始は，静的な側面のワークフローのドメインモデルの作成である．ドメインモデルは対象ドメインで使用する用語を体系化したもの（用語集）であり，静的な側面のワークフローの成果物としてクラス図に進化していく．ここで整理した用語を使っ

て，ストーリーボード等を参照しながらユースケースモデルを作成する作業が，動的な側面のワークフローの出発点となる．

ユースケースモデルは振舞いに対する要求をユースケース図とユースケース記述で定義したものであり，ユースケース図に記されたユースケースが要求機能に相当する．ユースケースの詳細な振舞いは，アクターとシステムとのインタラクションとして，ユースケース記述の中で詳細に説明される．このユースケースモデリングによって見直された用語の体系はドメインモデルに反映され，ドメインモデルはユースケースモデルと整合するように更新される．ここで説明したユースケースモデルとドメインモデルが要求定義工程の成果物となる．

動的な側面のワークフローにおける設計工程の成果物はシーケンス図であり，要求を定義するユースケースモデルとそのシーケンス図の間を仲介するのがロバストネス図である．ロバストネス図の作成は予備設計に位置づけられる．ロバストネス図はユースケース記述をオブジェクトの絵として表現したものであり，ユースケース記述から抽出したオブジェクトを，ユースケース記述に記載されたシナリオ（イベントフロー）どおりに貼り付けて作成する．シーケンス図は，ロバストネス図で抽出したドメインオブジェクト等（最終的にはクラスに対応する）の相互作用を時系列に並べたものである．ロバストネス図で表現されたシナリオをそのまま時系列の相互作用に置換えた記述となっている．シーケンス図は詳細設計に位置づけられる．

ICONIX プロセスの範疇には含まれないが，ソフトウエアアーキテクチャの設計はロバストネス図と並行して実施されるべきものとされる．ソフトウエアアーキテクチャの設計プロセスがICONIX プロセスに含まれていないのは，ソフトウエアアーキテクチャが個々のプロジェクトに依存する場合が大きいためである．すなわち，同様なシステムであっても，その実現を図るプロジェクトの構成員，スケジュール，技術的背景，またはそのほかの様々な制約等により，ソフトウエアアーキテクチャの設計プロセスは異なったものになる．しかし，いずれにしても，ソフトウエアアーキテクチャの設計はロバストネス図の作成完了と同時に終える必要があり，ロバストネス図分析の成果からシーケンス図を導出するときにソフトウェアアーキテクチャ設計の成果も合わせて反映される．例えば，ライフラインとしてソフトウェアアーキテクチャ上必要なクラスが追加・変更・削除されたり，特

有な相互作用が追加されたりする．同様に，ロバストネス図分析によって定義されたドメインオブジェクトは，ソフトウェアアーキテクチャやシーケンス図に反映されクラスへと抽象化される．その結果，ドメインモデルはクラス図へと更新される．

このようにしてこれまでのプロセスに従い，要求定義モデルを入力とした予備設計・設計工程の成果として，動的な側面のワークフローではシーケンス図が作成され，静的な側面のワークフローではクラス図が作成される．最終的に，クラスはコードとして実装され，シーケンス図に基づいた設計駆動のテストでその有効性が確認される．

以上説明したように，ICONIX プロセスでは，ユースケースで定義した要求を出発点としてロバストネス図で予備設計を行い，それを介して，クラス図，シーケンス図等での詳細な設計へと進むプロセスが提供されている．すなわち，ICONIX はユースケース駆動プロセスであり，獲得された要求がユースケースモデルに適切に反映されていれば，要求を正しく反映した設計を期待できる．

本論文は，KAOS による要求モデルを ICONIX の動的な側面のワークフローに漏れなく反映させるための仕組みを提案するものである．

第3章　関連研究

ゴール指向要求分析手法は，最終ゴール（システム目標）を矛盾なく実現する要求モデルを構築できると言われている．ゴール指向による要求分析･定義モデルを，次工程の設計モデルに，一貫性を持って効率良く反映させる試みがいくつかなされてきた．しかし，以下に言及するとおり，これらの関連研究においては，依然として課題が残っている．本研究は，これらの課題の解決を目的とするものである．

3.1　KAOSモデル指向設計

KAOSでの要求分析に基づいたオブジェクト指向設計へのアプローチが研究されてきた．Lamsweerdeは，機能要求/非機能要求ゴールからソフトウェアアーキテクチャを体系的に構築できる，アーキテクチャ設計へのゴール指向アプローチを提案している[19]．そこでは，次のような手順でKAOSによる要求からアーキテクチャを導出する．

1. 要求からソフトウェア仕様を導出する．この時，ドメインオブジェクトを抽象化オブジェクトにマッピングするが，その一貫性を保つために非機能ゴールを導入し，さらにそれに責任を持つ入出力エージェントを導く．

2. ソフトウェア仕様から抽象アーキテクチャドラフト（データフローアーキテクチャ）を導出する．エージェントをコンポーネントに再編し，コンポーネント間のデータフローを明示する．

3. 抽象アーキテクチャドラフトを，ドメインのアーキテクチャ制約に適合するように，アーキテクチャスタイルを選定して洗練する．

4. 上記で得たアーキテクチャを様々な非機能要求に適合するようにさらに洗練する．

この研究では KAOS 要求モデルからアーキテクチャを導出しているので本稿よりも範囲が広いが，手順 1，2 の要求（KAOS ゴールモデル）からコンポーネント間のデータフローを導出する部分がほぼ対応する．ここでは，人手による作業を想定しており，そのための具体的なガイドラインを提示している．例えば，それぞれのゴールの実現に責任を持つソフトウェアエージェントがモニタ/制御する変数を抽出し，それら変数に対する各々のソフトウェアエージェントの依存性を根拠としてデータフローアーキテクチャを導出している．さらに，ゴールからユースケースモデルの導出方法を解説した書籍 [20] では，ゴールモデルから要求ゴールを操作可能とする操作モデルを導出し，そのモデル中に定義されたそれぞれの操作と環境エージェント（当該操作の入/出力オブジェクトをモニタ/制御する）を関連付けることによってユースケース図を定義している．操作モデルの作成は事前/事後条件等をゴールから推察するなど，ガイドラインに沿った人手によって実施する．これらふたつの研究は，どちらも具体的なガイドラインを提示しているが，それでもすべて人手による作業であり，作業者の知識や経験則に影響されてしまうところは多い．

また，Heaven and Finkelstein[12] は，KAOS プロファイル（ゴール，要求，エージェント，イベント等）を UML モデル図中にステレオタイプとタグで表現して，UML によるモデルの中に KAOS モデリングを直接組込むアプローチを提案している．KAOS モデルは，例えば，ユースケースモデル，シーケンス図，オブジェクト図などで表現される．また，UML で記述した KAOS モデリングと通常の KAOS モデルは相互にマッピングできる．しかし，著者らも指摘しているとおり，KAOS モデルは要求定義モデルであり，UML による設計モデルとは粒度に差がある．例えば，KAOS モデルにおける Kagent クラスは，UML による設計レベルでは複数のクラスに分解される．従って，UML に組込んだ KAOS モデルを設計レベルに詳細化する手順が必要になる．

Robinson and Elofson[32] は，UML による既存手法と既存のゴール指向要求分析手法を統合し，ゴールから UML による設計仕様を導くアプローチを定義している．そのアプローチは次の五つのアクティビティからなる．

(1) システムコンテキストの抽出→ (2) システムゴールの定義→ (3) 要求の導出→ (4) ユースケースの導出→ (5) UML 定義モデルの導出

このゴール指向 UML モデリングアプローチでは，組織間の関係，ひとつの組織における

タスク間の関係，タスク設計仕様，それぞれを規定するユースケースを，ゴールモデルの階層構造から導出している．サブゴールまたはサブ要求ゴールをユースケースステップとして捉え，サブゴールから組織間の関係を示すユースケース，サブ要求ゴールからタスク間の関係を示すユースケースを定義している．ゴール階層を生成するのに洗練パターンをよく使うと説明しているが，ゴール階層からのユースケースの導出はサブゴール（サブ要求ゴール）をユースケースステップに対応させるとしている．ゴール階層はユースケースのアクターアクションを明らかにするとも述べているが，具体的な手順は示されていない．サブゴールからシナリオを抽出し，それからユースケースステップを導出するアルゴリズム的な手順を提示できれば，設計者の知識や経験への依存度をより低減するアプローチになると推察される．

Quartel ら [31] は，KAOS 等既存の要求分析手法を活用して，動機分析からユースケースを導く ARMOR という手法を提案している．エンタープライズアーキテクチャのゴール指向要求分析に使用する．そこでは，ステークホルダの動機分析から始める．まず，ステークホルダの関心事を洗い出し，それらについて SWOT 分析等の評価を実施する．さらに，その評価結果に基づき，それを改善するのに必要なハードゴールまたはソフトゴールを導く．例えば，評価の結果，弱点が見つかればその弱点を解消するためのゴールを抽出し，脅威が見つかればその脅威を排除するためのゴールを設定する．次に，それらゴールを洗練して，サブゴール，システム要求，ユースケースを導出し，さらにそれらを実現するためのビジネスサービスやプロセスを特定している．ゴールの洗練は KAOS の考え方を利用している．このモデリングをステークホルダを頂点とする階層に従って実施することで，ステークホルダの関心事から特定されたサービス/プロセスへと追跡可能である．ユースケースは対応するゴールを満足させるものとして導いているが，具体的な手順が示されていないため，設計者の知識や経験的な部分に依存する結果となる．

以上の従来研究は，ともにゴールを洗練・分析してユースケースを導出するというプロセスである．これらのプロセスではゴールからユースケースを経験則によって導いているが，本論文の提案アプローチでは変換テンプレートを仲介したアルゴリズムによる変換アプローチを定義し，知識や経験則をできるだけ排したより負荷の少ない一般的なアプローチとなっている．

3.2　KAOS への洗練パターン適用

Darimont ら [7] は，時相論理で定義した KAOS ゴールの一般的な洗練パターンと，それを操作可能とするイベントによる刺激－応答に基づく操作パターンを提案している．そこでは，ドメインに依存しない一次の洗練パターン，例えば，マイルストーン駆動，ケース分解などをゴールモデルの定義に則って時相論理で定義している．この洗練パターンは非操作なパターンである．第一ステージとして，これらの洗練パターンを利用して要求定義ゴールモデルを構築する．次のステージで，刺激-応答パターンを使って，第一ステージでモデリングした非操作な洗練パターンのゴールを操作的なパターンに変換する．この刺激ー応答パターンも時相論理で定義してある．この結果，ゴールによる要求定義モデルをエージェントに割振るべき操作要求に変換でき，設計工程へと移行できる．

このように，時相論理による洗練パターンおよび刺激ー応答操作パターンを使うと，要求定義から操作要求へと時相論理を用いて定義・変換されるので，矛盾なく一貫した設計が期待できる．しかし，著者らも言及している通り，実務の設計者にとって，時相論理を習得しモデリングに使用する負荷は大きく，彼らは時相論理を使って要求を論理的に記述して分析することなしに，このようなアプローチを利用したいと望んでいる．

この洗練パターンは，本稿の提案アプローチでも採用しているが，ゴールモデル作成者の負担となるため形式手法による記述は使っていない．形式手法に依らないで，洗練パターンを使った規則的な手順で KAOS ゴールモデルから振舞い要求を抽出するアプローチを提案している．

3.3　i*，Tropos による一貫した開発手法

本稿のベースになっている KAOS は代表的なゴール指向要求分析手法のひとつであるが，KAOS と同様，ゴール指向要求分析手法の代表的な手法として i*[36] がある．i*は，アクター間の依存関係とアクター内の依存関係を，それぞれ意図ネットワークとして表現している．

Bresciani ら [3] や Mylopoulos ら [25] は，上流から下流までのソフトウェア開発を，i*[36] をベースにしたエージェント指向開発に基づいてサポートする手法として，Tropos を導入

している．Bresciani らによると，Tropos によるエージェント指向開発は，ソフトウェア開発工程を前期要求工程から後期要求工程，アーキテクチャ設計工程，詳細設計工程と実装工程に分け，アクターに主眼を置いた意図ネットワーク（依存関係）を使ってモデルを詳細化することで，各工程に一貫した開発手法を提供している．

その中では，i*のモデリング機能を使って，ステークホルダ (アクター）およびその意図（ゴール）の特定とそれらのゴール分析結果を，アクター間の依存関係を示す SD(Strategic Dependency) モデルとアクター内部の依存関係を示す SR(Strategic Rationale) モデルで表現している（前期要求工程）．さらにそれを，対象システムを示すアクター（以降，対象アクター）とその他のアクター間の詳細な依存関係分析や対象アクターの詳細な SR 分解によって system-to-be（システムのあるべき姿）へと詳細化し，すべての機能要求や非機能要求を具体化する（後期要求工程）．次に，アクターをマッピングしたエージェントの依存関係からそれぞれのエージェントに機能を割付け（アーキテクチャ設計工程），さらにエージェント UML（AUML）を使ってその機能を詳細に仕様化している（詳細設計工程）．AUML では，エージェント間のコミュニケーションプロトコルをアクティビティ図やシーケンス図で表現する．

このように，Tropos は i*のモデリング機能や AUML の図表示などを使って要求定義から設計・実装までを一貫して実施するものであるが，i*によるモデリングや AUML による設計をどのように実施するかについてはアルゴリズム的な手順はない．すなわち，Tropos による成果は実施者に依存し，知識や経験則の影響を大きく受けてしまう．

時間的に動的な関係を扱えるよう Tropos を拡張したものとして，Formal Tropos（FT）がある．KAOS で時間関係を形式的に記述するのに使う時相論理を扱えるよう，Tropos を拡張したものである．Fuxman ら [9] は FT を前期要求工程に適用した．FT でモデリングした前期要求モデルを，モデル検査器を使って自動的にチェックし繰り返し洗練する，T-tool フレームワークを提案している．そこでは，i*モデルの親ゴール-子ゴール間の手段-目的関係等で暗黙的に表現されているオブジェクト間の動的制約を導出し，時相論理で形式的に定義している．

前期要求工程で抽出された時間的な動的制約を時相論理で定義することで，i*で分析・定義された要求を矛盾なく設計工程に引き継ぐことができ，要求定義工程から設計・実装

工程までの一貫した実施を期待できる．しかし，一般的な設計者にとって，形式的な FT を習得し，時相論理式を使ってモデルを記述して分析することに対する負荷は大きい．

これまでに述べたように，i*，Tropos による開発は，要求定義から設計へ一貫性を持って移行する仕組みを持っている．しかし，アルゴリズム的な手順ではなく，手法に基づいた手動による作業であるため，設計者の知識や経験がその成果に及ぼす影響は大きい．FT はそれを改善するために，要求定義モデルをアルゴリズム的に設計に引き継ぐように時相論理を用いて工夫したものであるが，実務の設計者にとって時相論理や FT の習得に対する負荷は大きい．これに対して，本稿では，時相論理などの形式手法を使うことなく，要求定義モデルをアルゴリズム的（規則的）に設計に引き継ぐアプローチを提案している．

第4章　ゴール指向要求分析駆動による UML設計手法の全体像

要求定義工程と設計工程との間にはギャップがあり，要求定義情報のいくらかが抜け落ちたり曲解されてしまうと言われている．本章では，要求定義工程の成果である要求定義モデルを，設計工程の出発点となるユースケースモデルや予備設計モデルとしてのロバストネス図に変換するアプローチを提案する．この提案アプローチを用いたゴール指向要求分析駆動による UML 設計手法の全体像を図 4.1 に示す．

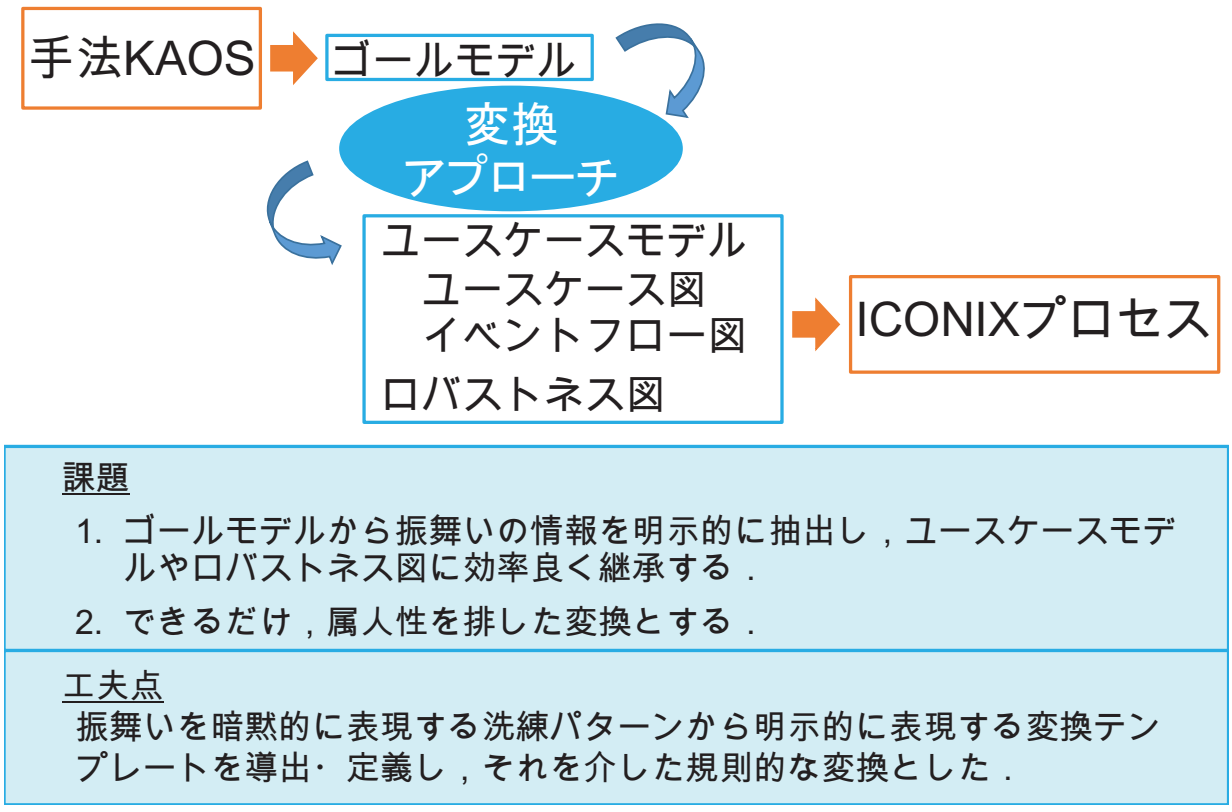


図 4.1: ゴール指向要求分析駆動による UML 設計手法の全体像

本提案アプローチを利用すると，要求定義工程と設計工程それぞれに実績のある従来技術を効率よく活用することができる．要求定義工程と設計工程の実績ある従来技術として，それぞれゴール指向要求分析手法 KAOS とユースケース駆動開発プロセス ICONIX を対象としている．すなわち図 4.1 に示すように，KAOS による要求分析・定義の成果であるゴールモデルを，設計工程の ICONIX における入力モデルであるユースケースモデル（ユースケース図，ユースケース記述：イベントフロー図），さらに予備設計モデルであるロバストネス図へとそれぞれ規則的に変換する．この結果，ユースケースモデルやロバストネス図にゴール指向による要求分析・定義の成果が反映され，ICONIX による設計プロセスがゴール指向要求分析の成果によって駆動される．

KAOS ゴールモデルは，システムによって実現すべきゴールを階層的に構成した宣言的モデルである．振舞いを明示的に表現するものではないが，ゴールは振舞いの結果として得られるものなので，振舞いを暗黙的・暗示的に表現していると言える．それを踏まえ，ゴールモデルから振舞いのシナリオを明示的に抽出し，ユースケースモデルやロバストネス図にそれを如何に効率よく継承するかが図 4.1 に示す課題 1 である．また，ゴールモデルによる振舞いのシナリオは要求として分析・定義されたものであり，要求定義工程からの一貫した要求の情報として，設計者によって変化することなく，ユースケースモデルやロバストネス図に反映されるべきものである．すなわち，ゴールモデルからユースケースモデルやロバストネス図への変換は，できるだけ設計者の知識や経験に依存しないように属人性を排したものにする必要がある．これがもうひとつの課題 2 である．これらふたつの課題に対する対策として，図 4.1 の工夫点に示すように，基本的な振舞いを暗示する洗練パターンを根拠に，振舞いのシナリオを明示的に表現する変換テンプレートをモデルごとに導出し，それらを介した規則的な変換とした．変換テンプレートは当該モデルの規則に従って振舞いのシナリオを定義している．このことから，変換元モデルの変換テンプレートを変換先モデルの変換テンプレートにマッピングすることによって，振舞いのシナリオを規則的に効率よく継承できる．この工夫によって，情報の効率的な継承と変換における属人性の排除が可能となった．また，すべてのゴール分解を洗練パターンで網羅できるとは言えないが，洗練パターンはごく基本的な振舞いについてパターン化されているので，それらを組み合せることによって多彩な振舞いのゴールモデリングが可能であり，要求モ

デリングにおける大部分のゴール分解を網羅できる．

本提案アプローチは，本田らの議論 [45, 13, 43] を整理統合して発展させたものである．

本章では，まず最初に 4.1 節で，従来技術と提案技術を組み合せて要求定義モデルをユースケースモデルとロバストネス図に変換するアプローチの全体像を説明した後，次の 4.2 節で，変換の重要なキーである変換テンプレートの構成について説明する．なお，KAOS モデルからユースケースモデルまたはロバストネス図への変換方法は第 5 章で詳しく説明する．

4.1 変換アプローチの全体像

KAOS ゴールモデルからターゲットモデルであるユースケースモデル（ユースケース図，イベントフロー図）やロバストネス図への変換の意味・必要性について説明する．ゴールモデルからユースケースモデルやロバストネス図への変遷を図 4.2 に示す． 洗練パターン

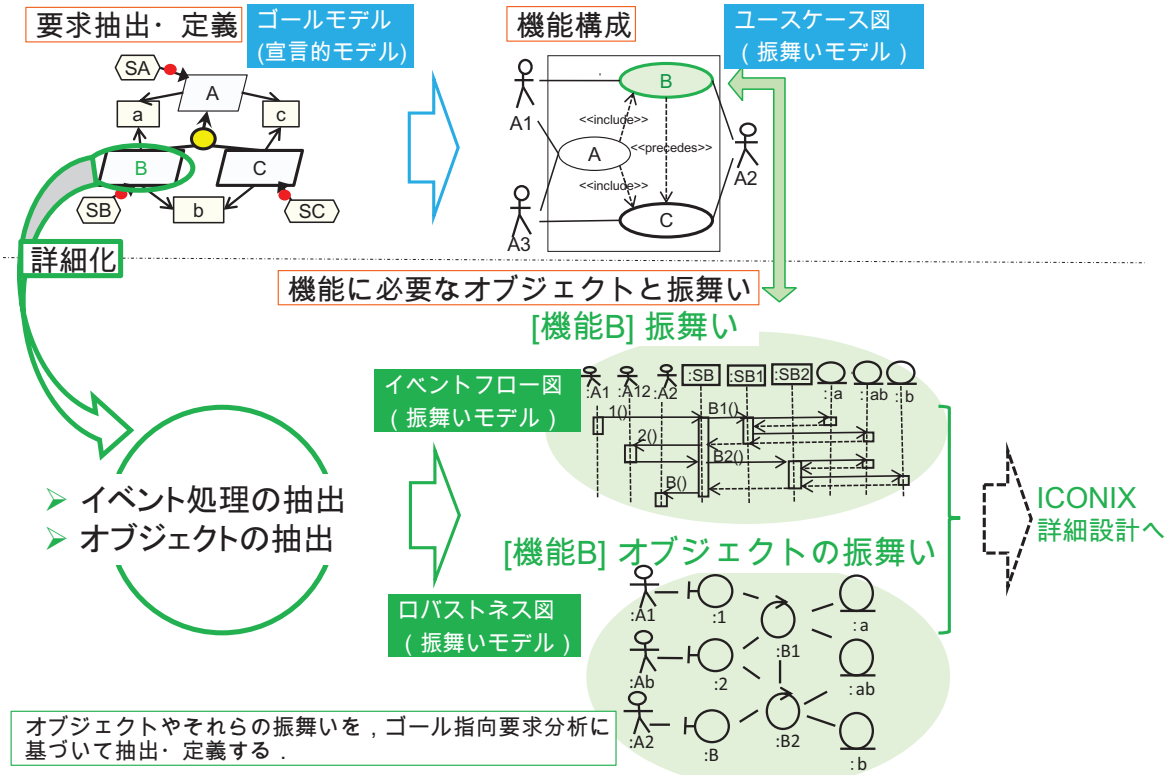


図 4.2: ゴールモデルからユースケースモデル，ロバストネス図への変遷

を組合せて構成されている KAOS ゴールモデルを前提としている．ゴールモデルは実現すべき最終ゴール（最終目標）を階層的に AND 分解した宣言的モデルであるため，振舞いを明示的に表現するものではない．しかし，個々の洗練パターン自体は振舞いを暗黙的に表現しているため，振舞いを抽出して明示的に表現することは可能である．一方，ターゲットモデルであるユースケース図とイベントフロー図，およびロバストネス図は振舞いモデルである．

まず，図 4.2 上側のゴールモデルからユースケース図への変換について考える．ユースケース図は，アクターによる機能の使用関係とともに機能間の関連を示す振舞いモデルである．ユースケースによって機能を示しているが，これはゴールモデルのリーフゴールである要求ゴールに対応する．従って，要求ゴールを実現する振舞いを抽出することによってユースケースに変換することができる．さらに，要求ゴール間の関連をユースケース間の関連に置き換えることができれば，ゴールモデルをユースケース図に変換することができる．そこで，宣言的モデルであるゴールモデルからゴールを実現する振舞いを抽出し，ユースケース図に反映する手段が必要になる．つまり，ゴールモデルの暗黙的な振舞いの情報を明示的に表現できれば，それを利用してユースケース図への規則的な変換が可能となる．

次に，図 4.2 下側のユースケースのイベントフロー図とロバストネス図を導出する手段について考える．イベントフロー図は，ユースケース記述の主な情報であるイベントフロー（振舞いのシナリオ）を図で表現したものである．本稿ではシーケンス図の表記を使って記述している．また，ロバストネス図は，同じくユースケース記述のイベントフロー（振舞いのシナリオ）からオブジェクトや機能を抽出し，振舞いのシナリオに沿って関連付けたものである．これらの図は，ユースケースにおける振舞いのシナリオ，およびそれに基づくオブジェクトと機能の関連を詳細に説明していることから，詳細設計の出発点に位置づけることができる．

通常，ユースケースの振舞いを詳細に分析してユースケース記述（イベントフロー図）を導出する．また，そのユースケース記述を具体的に読み解いた中からオブジェクトと機能，およびそれらの関連を導出し，ロバストネス図を作成する．ところで，上述したように，ユースケースは要求ゴールに対応する．この関係から，ユースケースの代わりに要求

分析の成果である要求ゴールをさらに分析して，イベントフロー図とロバストネス図を導出すればより効果的であると考えられる．要求分析手法 KAOS を利用して要求ゴールを直接詳細化できることが，その効果の理由である．KAOS はユースケース図よりも要求分析に適している．そこで，図 4.2 を例に取ると，ゴールモデルのリーフゴールである要求ゴール B を KAOS を活用して詳細化し，イベント処理とオブジェクトを抽出する．それらの抽出結果が振舞い図として得られれば，イベントフロー図とロバストネス図に効率よく変換できる．

これまでに，KAOS ゴールモデルからユースケースモデル（ユースケース図，イベントフロー図）やロバストネス図への変換の意味・必要性について説明した．次にそれを受けて，具体的な変換の手段と流れを説明する．図 4.3 は，KAOS ゴールモデルからユースケース図への変換の流れを示している．図にあるとおり，KAOS ゴールモデルは宣言的な要求

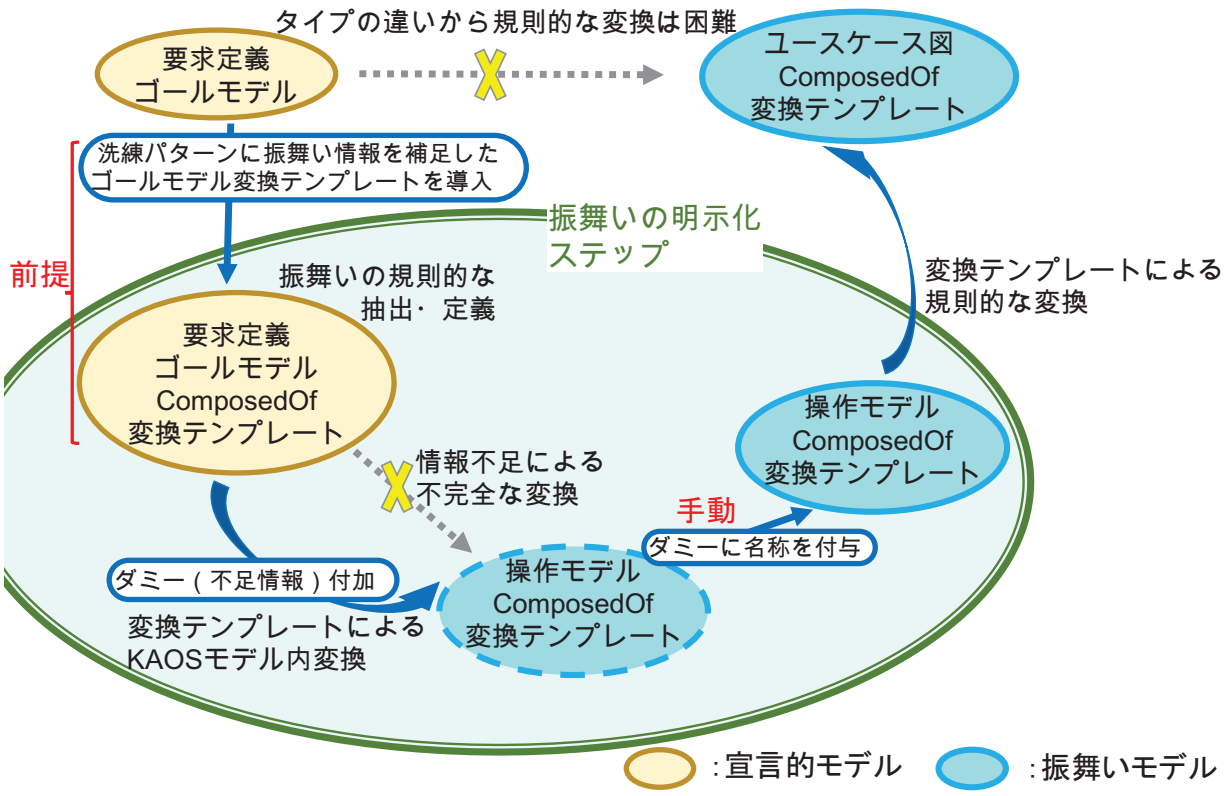


図 4.3: ゴールモデルからユースケース図へのモデル変換の流れ

定義モデルである．これに対して，ユースケース図は振舞いモデルであるため，規則的に

直接変換することは難しい．この対策として，KAOS で振舞いモデルとして定義されている操作モデルを利用する．すなわち，ゴールモデルで暗黙的に表現される振舞いを操作モデルで明示的にモデル化し，宣言的モデルを振舞いモデルに変換する．操作モデルは要求ゴールを実現するための操作のモデルであると KAOS では定義されているので，暗黙的な振舞いを明示化するのに適している．そこで，要求ゴールを操作（要求ゴールを実現するための操作）に置換え，ゴールモデルにおける洗練パターンの振舞いを操作間の関連に置換える．こうして，ゴールモデルを操作モデルに変換できる．この結果，ユースケースと親和性が高い操作モデルを介して，KAOS ゴールモデルをユースケースモデル（ユースケース図とイベントフロー図）やロバストネス図に変換することができる．

これらの変換を規則的に行うため，洗練パターンの振舞いを，より具体的に定型的な振舞いとして，各モデルで明示化した変換テンプレートを新規に導入する．ゴールモデルの変換テンプレートは，洗練パターンのゴールモデルにエンティティやソフトウェアエージェントなどオブジェクトモデルや責任モデルの要素を付加したものである．さらに，操作モデルへの規則的な変換をするために必要な，振舞いに関する情報，例えば洗練パターンの種類，マイルストーン駆動のサブゴールの数や駆動の順番，ケース分解のケース数，またはエンティティのゴールに対する入出力関係など，が補足されている．提案アプローチに対する入力としての要求定義モデルは，このような変換テンプレートで構成されたゴールモデルが前提となる．

操作モデルの変換テンプレートは，洗練パターンをイベント駆動による操作とエンティティの入出力などで定義したものであるが，ゴールモデルでは抽出できない環境エージェントとイベントを含んでいる．従って，そのままでは情報不足による不完全な変換となってしまうため，これらの要素を所定の位置にダミーとして付加して変換を完結させる．これらのダミーとして付加された環境エージェントとイベントに，適切な名称を手動で付与することにより，操作モデルの導出が完了する．

この「要求定義ゴールモデル ComposedOf 変換テンプレート」から「操作モデル ComposedOf 変換テンプレート」までは，図にあるとおり「振舞いの明示化ステップ」である．この明示化ステップにおけるモデル変換は，上述したように，操作を駆動するもしくは操作の結果として出力されるイベントと，それらイベントを発生させるもしくはイベントか

ら影響を受ける環境エージェントが情報として追加される垂直変換である．すなわち，モデルの形態が変わるとともに情報が追加される．ここで導出された操作モデルにはユースケース図に必要な情報が含まれているため，ユースケース図への規則的な変換が可能となる．このように，操作モデルからユースケース図への変換は，情報の追加がなくモデルの形態のみが変わる水平変換である．

続いて，図 4.4 に，KAOS ゴールモデルからイベントフロー図，ロバストネス図への変換の流れを示す．リーフゴールに要求ゴールが抽出されている「要求定義ゴールモデル

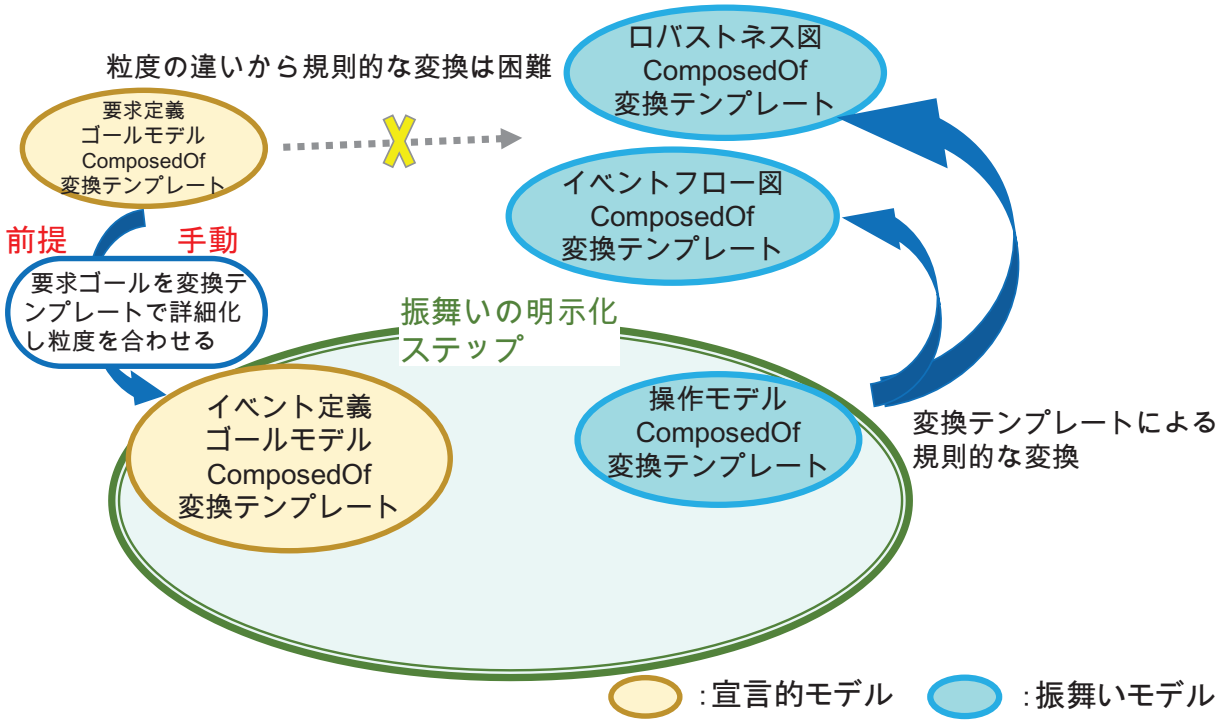
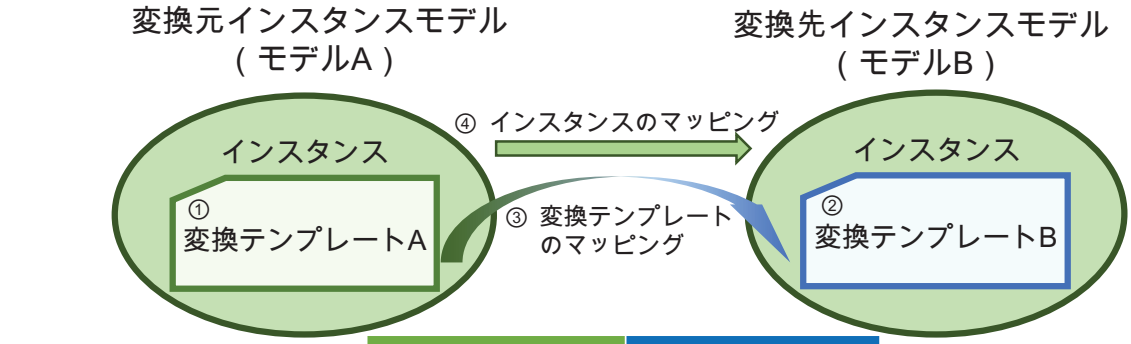


図 4.4: ゴールモデルからイベントフロー図，ロバストネス図への変換の流れ

ComposedOf 変換テンプレート」は，図にあるとおり粒度の違いからイベントフロー図やロバストネス図への規則的な変換は困難である．このため，「要求定義ゴールモデル ComposedOf 変換テンプレート」で抽出された要求ゴールを詳細化し，イベントフロー図やロバストネス図と粒度を合わせる必要がある．詳細化は，インタラクションを明確にするために，単一のイベントに対応するゴールが抽出されるまで階層的に実施する．また，変換テンプレートを介した変換とするために，前提として詳細化にはすべて変換テンプレート

を使う．こうして，要求ゴールを詳細化したゴールモデルが「イベント定義ゴールモデル ComposedOf 変換テンプレート」である．このように詳細化して粒度を合わせることができれば，ユースケース図への変換時に実施した「振舞いの明示化ステップ」を使って,「操作モデル ComposedOf 変換テンプレート」に変換できる．この操作モデルにはイベントフロー図，ロバストネス図それぞれに必要な情報が含まれているため，それぞれへの規則的な変換が可能となる．従って，操作モデルからイベントフロー図，ロバストネス図それぞれへの変換は水平変換となる．

次に，変換テンプレートを介したモデルの規則的な変換について，その意味を説明する．図 4.5 はそれをまとめたものである．図に示すように，変換テンプレートで構成されたモ



モデルA	モデルB
KAOS ゴールモデル	KAOS 操作モデル
KAOS 操作モデル	ユースケース図
KAOS 操作モデル	イベントフロー図
KAOS 操作モデル	ロバストネス図

変換テンプレート
洗練パターンのシナリオを，各モデルの表現規則に従い一般的に定義．シナリオはマイルストーンなど6種類があり，明示的に表現される．ごく基本的な振舞いのシナリオである．

モデルAからモデルBへの規則的な変換
① モデルAの変換テンプレートを特定
② モデルBに①と同じ種類の変換テンプレートを適用
③ 変換テンプレートのマッピング
④ モデルAのインスタンスをモデルBの変換テンプレートの要素にマッピング

図 4.5: 変換テンプレートによる規則的な変換

デル A からモデル B に変換する場合を考える．モデル A とモデル B の組合せは，図中央の表にあるように KAOS ゴールモデルから KAOS 操作モデルへの変換など 4 通りがある．モデル A，モデル B ともに，変換テンプレートを使って階層的に構成され，インスタンス

化される．変換テンプレートは，洗練パターンのシナリオを各モデルの表現規則に従い一般的に定義したものである．シナリオはマイルストーンなど 6 種類があり，各モデルの表現規則に従い，明示的に表現される．ごく基本的な振舞いのシナリオであるため，それらを組み合わせることによりさまざまなシナリオを記述できる．このような変換テンプレートのマッピングの結果，詳細な振舞いのシナリオを失うことなくモデル変換できる．変換テンプレートの詳細は 4.2 節で説明する．

変換テンプレートを使った規則的な変換は①～④の手順で実施する．モデル A とモデル B は，ともに同じ洗練パターンに対応した（同じ種類の）それぞれの変換テンプレートを使って，同一の振舞いのシナリオを表現している．そこで，まず手順①で，変換元であるモデル A の変換テンプレートを特定し，それと同じ種類の変換テンプレートを変換先モデルであるモデル B に適用する（手順②）．それに基づき手順③でモデル A からモデル B に適用した変換テンプレートのマッピングを行う．最後に手順④で示すように，同じ位置にマッピングされた同種類の変換テンプレートの要素にモデル A のインスタンスをマッピングする．以上が変換の手順である．

このようにして，ゴール指向に基づいたユースケースモデルとロバストネス図を作成することができる．整理すると，入力となる KAOS ゴールモデルは，洗練パターンによるゴール分析でシステム範囲が決定され，関連するエージェントやエンティティ等オブジェクトを含めてモデル化されたものが対象である．すなわち，目標ゴールを実現するためのシナリオが，洗練パターンの組合せによる要求ゴール間の関連として暗黙的に表現されている．一方，出力となるユースケース図では，ユースケースとアクターとの関連およびユースケース間の関連が表現される．そのユースケース記述として，ユースケースのシナリオがイベントフロー図で表現されている．事前・事後条件はアクター間のメッセージや参照する情報に含めるようにした．同じく出力となるロバストネス図では，ユースケース記述の導出元である要求ゴールの振舞いのシナリオからオブジェクト，処理を示すコントローラ，およびアクターが抽出され，それぞれの関連が表現される．この結果，ユースケース記述とオブジェクトとを対応付けることができる [51].

提案アプローチは，ユースケース図に変換する STEP1～5 とイベントフロー図に変換する STEP6～8，およびロバストネス図に変換する STEP6,7,9 で構成される．なお，イベン

トフロー図，ロバストネス図それぞれへの変換ステップのうち，STEP6，7 は共通であるため一度実施すればよい．これら変換プロセスの全体イメージを図 4.6 に，変換処理全体のフローを図 4.7 にそれぞれ示す．図 4.6 と図 4.7 の中の数字は，ステップ番号（ex.STEP1.）を示している．また，図 4.6 の各ステップの変換を示す矩形の下に表示している（垂直），（水平）は，その変換がそれぞれ “垂直変換”，“水平変換” であることを示している．

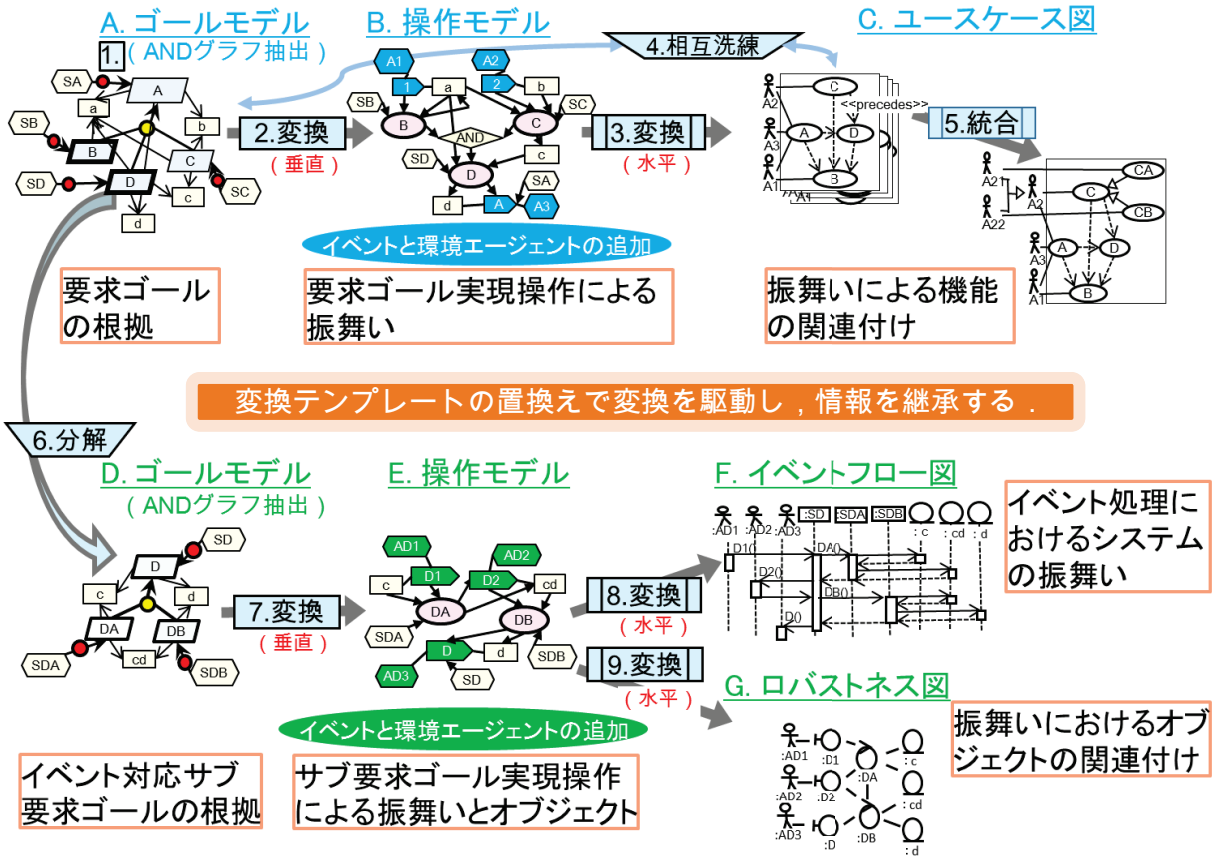


図 4.6: ゴールモデルからユースケースモデル，ロバストネス図への変換プロセス

図 4.6 の全体イメージにおいて，上段は要求定義の KAOS ゴールモデルからユースケース図への変換の流れを示している．ゴールモデルのリーフゴールが要求（要求ゴール）であり，操作からユースケースへと変換され，それぞれ変換テンプレートで他のモデル要素と関連付けられる．ゴールモデルでは要求ゴールが導出された根拠が，操作モデルでは要求ゴールを実現するために必要な操作による振舞いのシナリオが，ユースケース図では振舞いに必要とされる機能の関連付けがそれぞれ表現される．

一方，下段はユースケースそれぞれのユースケース記述（イベントフローで代表する）を導出する流れとなっている．要求ゴールはさらにイベント対応のサブ要求ゴールに分解され，上段と同様に，操作からイベントフロー図のメッセージへまたはロバストネス図のコントローラへと変換され，それぞれ変換テンプレートで他のモデル要素と関連付けられる．ゴールモデルではサブ要求ゴール（イベントに相当）が導出された根拠が，操作モデルではサブ要求ゴールを実現するために必要な操作による振舞いと関連するオブジェクトが，イベントフロー図ではイベント処理におけるシステムの振舞いが，ロバストネス図では振舞いに沿ったオブジェクトの関連付けが表現されている．それぞれのモデルは変換テンプレートを使って構成されているため，その置換えでモデルの変換と変換における情報の継承が可能になる．

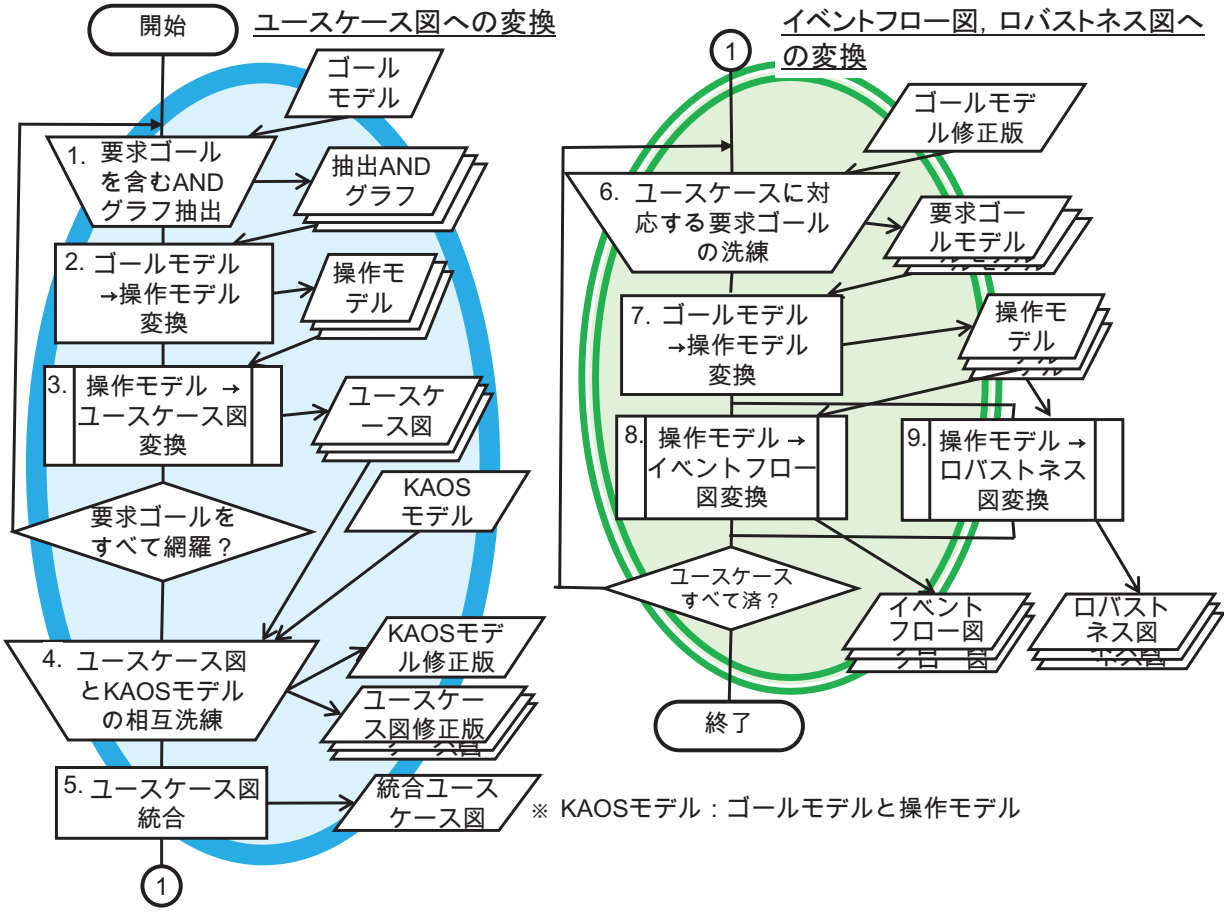


図 4.7: ゴールモデルからユースケースモデル，ロバストネス図への変換フロー

図 4.7 の変換処理全体のフローは，各ステップに対して入力されるモデルと結果として出力されるモデルを，変換ステップの流れに沿って示している．左側がユースケース図への変換のフロー，右側が要求ゴールの分解からユースケースのイベントフロー図，ロバストネス図それぞれへの変換のフローを示している．

各ステップの自動／手動の区別，新規／既存の区別，および概要は次のようになる．ここで，先頭の各数字はステップ番号に相当し，“新規” と書かれたステップは，提案アプローチにおいて新規に導入するステップを表わす．情報がすべて揃っているため，規則的に処理または変換できる部分は自動としている．また，経験則や知識に基づいた創意工夫が必要となる作業は手動で実施する．なお，自動で変換した結果に対して手動による作業が必要な場合は，自動後手動とした．以降，“ひとつの親ゴールとその直下のサブゴールによる AND グラフ” を，説明のため “一次の AND グラフ” と呼ぶ．

1. 一次の AND グラフ抽出（自動，既存）

2. ゴールモデル → 操作モデル変換（自動後手動，新規）

3. 操作モデル → ユースケース図変換（自動，新規）

4. ユースケース図 → KAOS モデル相互洗練（手動，既存）

5. ユースケース図の統合（自動，新規）

6. 洗練パターンによる要求ゴール分解（手動，既存）

7. ゴールモデル → 操作モデル変換 (自動後手動，新規）

8. 操作モデル → イベントフロー図変換（自動，新規）

9. 操作モデル → ロバストネス図変換（自動，新規）

図 4.7 に示すように，STEP 1 〜3 では，STEP1 で抽出した一次の AND グラフごとにユースケース図への変換を繰り返す．個々に変換されたユースケース図は，最終的に STEP5 で階層構成に合成され統合される．さらに，要求ゴールそれぞれをシングルイベント対応に分解して詳細化するが（STEP6），STEP6,7,8,9 はそれぞれ要求ゴールごとに実施され，す

べての要求ゴールを網羅するまで繰り返される．その結果，STEP8,9 の出力モデルであるイベントフロー図，ロバストネス図は，詳細化された要求ゴールごとに出力される．なお，ここでは省くが，密接に関連している階層については，イベントフロー図またはロバストネス図を合成して統合することでより理解し易くなる．

STEP2，STEP3，STEP7，STEP8，STEP9 では，それぞれ変換テンプレートを使ったモデル間の変換を実施する．これらの変換を通して，洗練パターンによる要求の実現シナリオが変換テンプレートを介して継承される．変換テンプレートについては，次節で詳述する．

それぞれの変換では，QVT[27] による変換規則に従って，変換テンプレートの置換えと情報のマッピングを実施する．その詳細は第 5 章で説明する．この結果，洗練パターンによって表現された要求定義モデルにおけるシナリオの継承と，変換における属人性の排除が期待される．このように，シナリオや情報を継承すべき作業は属人性を排除すべく自動で実施する．

提案アプローチの各ステップ（STEP1〜STEP8，STEP9）を以下に概説する．

・STEP1

リーフゴール（要求ゴール）を探索し，それをサブゴールとして含む一次の AND グラフを抽出する．これは，KAOS ゴールモデルを一次の AND グラフ（ゴールモデルの変換テンプレートで表現される）に分解することを示しており，この一次の AND グラフごとに変換を行う．

・STEP2

抽出した一次の AND グラフそれぞれを，操作モデルの変換テンプレートを利用した変換規則に従って，機械的に操作モデルに変換する．この変換は，暗黙的に振舞いを表現しているゴールの関連を明示的に振舞いを表現している操作の関連に置換えている．この STEP の変換結果では，振舞いのシナリオを明確にするために，環境エージェントとイベントが所定の位置にダミーとして配置される．それらについては，振舞いのシナリオに沿った名称が手動で追記されることによって，操作モデルとして完成する．この手動部分は，ゴールモデルにより暗黙的に表現された振舞いのシナリオを，操作モデルでより明確に表現するための補足作業である．ダミーの環境

エージェントとイベントを使った変換結果がそのガイダンスとなる．設計者の意図が反映される部分である．

・STEP3

それぞれの操作モデルを，ユースケース図の変換テンプレートを利用した変換規則に従って，機械的にユースケース図に変換する．すなわち，ゴールの実現操作をアクターから見た操作に変換する．ここで得られるユースケース図は，STEP1 で抽出された一次の AND グラフそれぞれに対応している．ユースケース図は，システム全体の機能構成を俯瞰するのに有効である [42]．

・STEP4

それぞれのユースケース図を全体として俯瞰し，必要があれば修正する．例えば，共通ユースケースの抽出，包含・拡張・汎化／特化関係の見直し，ユースケースの分割・結合等の修正である．修正結果を KAOS モデルに反映し，それぞれのユースケース図を確定する．

・STEP5

これまでに得られた各階層一段ごとのユースケースについて，上位階層のユースケースに下位階層のユースケース（サブユースケース）を代入することによって機械的に統合する．ここまでの STEP で，要求ゴール間の関連に相当するユースケース図を得ることができる．以降の STEP では，個々のユースケースのユースケース記述に相当するイベントフロー図とロバストネス図を求める．

・STEP6

一般的には，要求ゴールを抽出できたときにゴールの分解は終了する [20]．しかし，サブの要求ゴールを見つけることができれば，それをユースケースステップとして，ユースケース記述を定義できる [32]．これに基づき，要求ゴールにおけるシステムと外部環境とのインタラクションシナリオを明確にするために，ひとつのイベントだけに関わるゴールをサブゴールとする AND グラフを抽出するまで要求ゴールを階層的に洗練する．このゴール洗練にも，洗練パターン（表 2.1）を利用する．すなわち，変換テンプレート (図 4.8) の “ゴールモデル” 変換テンプレートを使う．このように

して分解した要求ゴールごとに，STEP1 と同様に AND グラフを一段ずつ抽出し，以降のステップでイベントフロー図またはロバストネス図に変換する．このステップでの作業は，KAOS の手法を活用した人手による作業である．要求ゴールの洗練に，KAOS 利用についての知識と経験が活用される．

・STEP7

STEP2 と同じアルゴリズムで，一段ごとの AND グラフをそれぞれ操作モデルに変換する．

・STEP8

それぞれの操作モデルを，イベントフロー図の変換テンプレートを利用した変換規則に従って，機械的にイベントフロー図に変換する．すなわち，ゴールを実現する操作の流れを，ゴールをイベントに対応させたイベントフローに置換えている．

・STEP9

ロバストネス図への変換では，STEP7 で作成した操作モデルを入力とする．すなわち，STEP8 と同様に，ロバストネス図の変換テンプレートを利用した変換規則に従って，それぞれの操作モデルを機械的にロバストネス図に変換する．変換においては，エンティティやイベント，操作を，それぞれオブジェクト，コントローラに置換え，これらオブジェクトやコントローラをシナリオに沿って関連づけている．

以上説明したように，STEP5 でユースケース図が得られ，STEP8 でユースケースそれぞれのイベントフロー図が，また STEP9 でユースケースそれぞれに対応するロバストネス図が得られる．

4.2 変換テンプレート

変換テンプレートは，洗練パターンの意味（シナリオ）を，ゴールモデル，操作モデル，ユースケース図，イベントフロー図，およびロバストネス図それぞれで表現したものである．ゴールモデルから操作モデル，操作モデルからユースケース図，操作モデルからイベントフロー図，操作モデルからロバストネス図それぞれの変換に使用する．ゴールモデル

の洗練パターンに対応したゴールモデル，操作モデル，ユースケース図，イベントフロー図，およびロバストネス図それぞれの変換テンプレートを図 4.8 に示す．洗練パターンは，

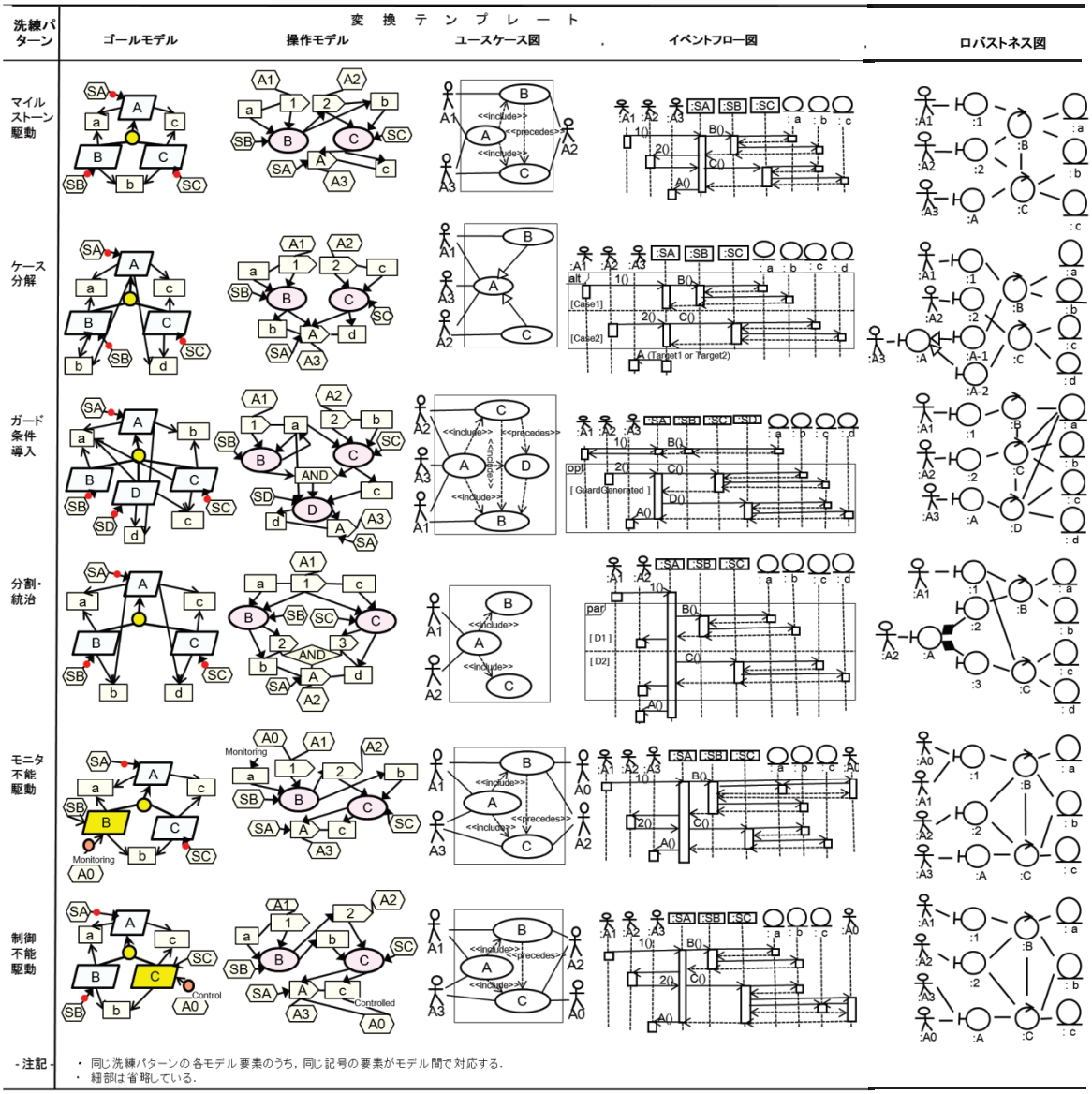


図 4.8: KAOS モデル→ユースケースモデル，ロバストネス図変換テンプレート一覧

戦術的な振舞いのシナリオに基づいたゴール分解方法の一般的表現である．マイルストーン駆動，ケース分解，ガード条件導入，分割・統治，モニタ不能駆動，および制御不能駆動の基本的な AND 分解の 6 パターンからなる．図の各行が洗練パターンそれぞれに対応する．変換テンプレートは，洗練パターンによる戦術的なシナリオを，ゴールモデル，操

作モデル，ユースケース図，イベントフロー図，およびロバストネス図それぞれのモデルの規則に従って表現したものである．洗練パターンに準じた一般的な表現になっているが，振舞いのシナリオを明確に定義するためのそれぞれのモデル要素を使って構成している．図の各列がそれぞれのモデルに対応する．洗練パターンの意味する戦術的な振舞いのシナリオをそれぞれのモデルの変換テンプレートで定義することにより，それらをモデル間で継承できる．また，変換テンプレートで置換えた後，モデル要素を具体的な情報でインスタンス化することにより，モデル変換が可能となる．変換テンプレートの活用に際し以下を考慮する必要がある．モデル間で同じ種類を 1 対 1 に対応付ける必要があるため，複数種類の変換テンプレートを複合的に構成すると変換に使用できなくなる．従って，複合的に構成された変換テンプレートは階層的に分解し，一次の AND グラフとして抽出できるように再構成する必要がある．また，環境エージェント，イベント，エンティティはそれぞれひとつとして変換テンプレートを定義しているため，抽象化するか変換後に手動追加する必要がある．

KAOS モデルのメタモデル（図 4.9），ユースケースモデルのメタモデル（図 4.10），およびロバストネス図のメタモデル（図 4.11）に基づいて，これら変換テンプレートを定義した．KAOS モデルのメタモデル（図 4.9）は，Heaven and Finkelstein[12] の KAOS モデルのメタモデルをベースにして，RefinementPattern の関連とその特化パターン，KAgent の特化オブジェクト，および KEvent と KAction（Operation）間の cause 関係についての情報を，KAOS モデルにおけるこれらの関連を明示するために，追記したものである．また，Action は Operation と同等であるため，KAction に（KOperation）を説明として追記した．KAOS モデルのメタモデル（図 4.9）は，次のような関係を示している．Goal は RefinementPattern で関連付けられる．その関連に基づき，Goal は KObject に関心を持つ．要求は Goal を分解したものなので，要求も同様に RefinementPattern および Kobject と同様な関連を持っている．KAction（KOperation）は Requirement を操作可能化し，そのために KObject を入出力する．すなわち，KObject に対して，Goal における関心は KOperation における入出力の関係となる．また，KAgent，KEvent，KEntity は KObject である．KEvent は KOperation の原因となり，KAgent は KOperation の実行，能力，あるいは責任をもつ．

ユースケースモデルのメタモデル（図 4.10）は，OMG の UMLV2.4.1, formal/2011-08-06,

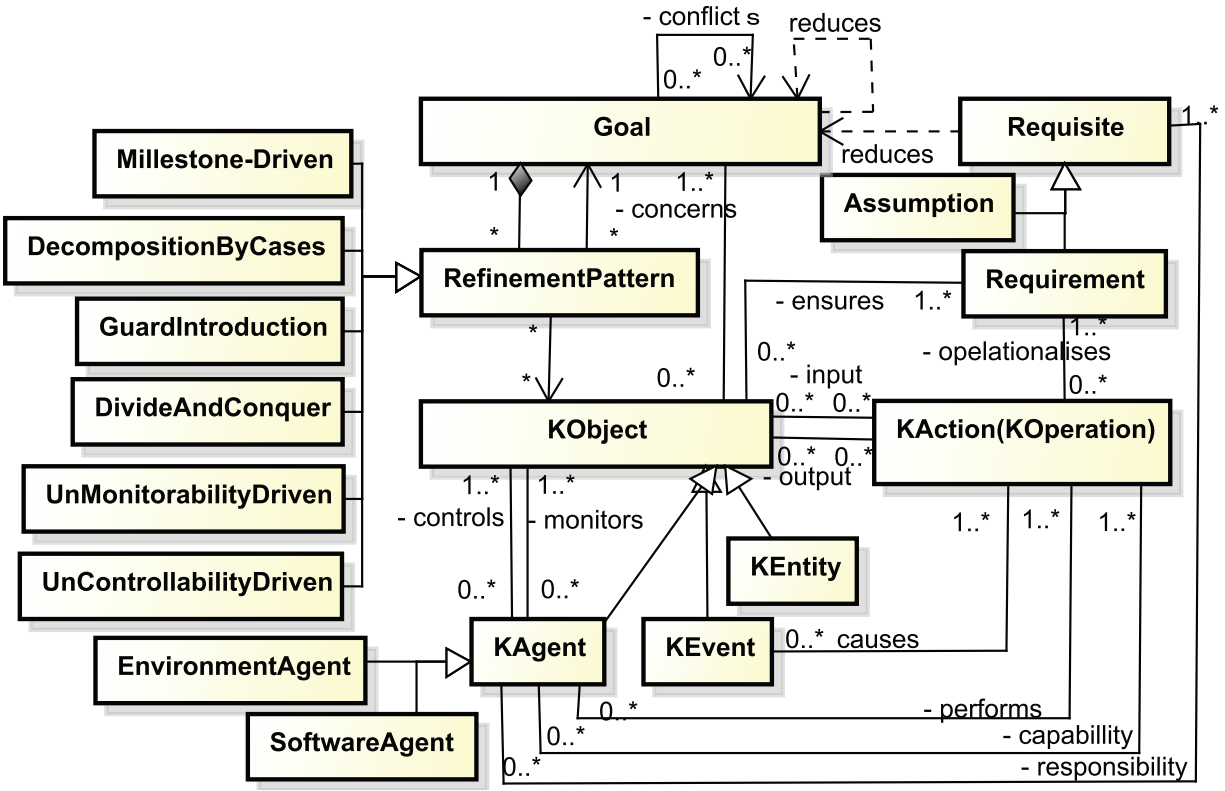


図 4.9: KAOS モデルのメタモデル（[12] に説明のため一部追記）

Superstructure Specification[28] で定義している Use case のメタモデルに，Event Flow のメタモデルを UseCaseDescription を介して関連付けた．Event Flow のメタモデルは UML Superstructure Specification では定義されていないので，Eclipse Modeling Framework（EMF）で使用されている Sequence Diagram のメタモデル [23] を用いた．要素間の関連性を明示するために，UseCase と Actor，UseCase と Generalize/Specialize，UseCase と Depend/Precede それぞれの関連性，及び Lifeline の特化要素を追記した．また，UseCase に対して，Sequence Diagram を Event Flow として関連付けるために，UseCaseDiscription を UseCase の構成要素として追記し，SeqDiagram をさらにその Event Flow を説明する構成要素として関連付けた．

このようにして構成したユースケースモデルのメタモデルは，次のような関係を示している．各々の UseCase は，DirectedRelationship に汎化される Extend,Include,Generalize/Specialize, Depend, および Precede などによってお互いに関連付けられ，Actor との関連を持つ．Se-

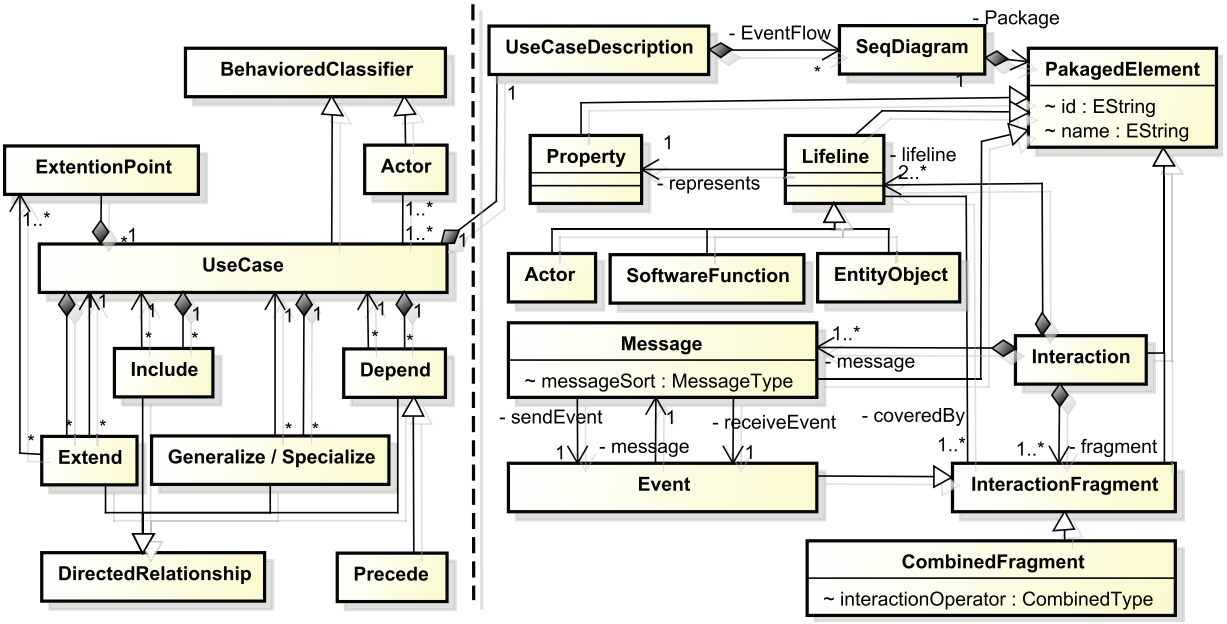


図 4.10: ユースケースモデルのメタモデル（[28][23] を一部省略/詳細化）

qDiagram(EventFlow) は，Lifeline, InteractionFragment, および Message などを使って Interaction を表現する．Lifeline としては，Actor, SoftwareFunction, EntityObject などがある．CombinedFragment もまた InteractionFlagment である．

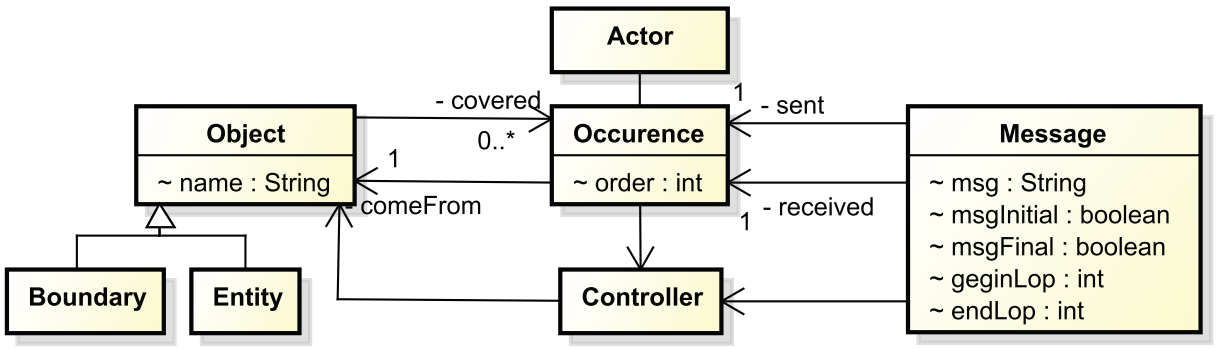


図 4.11: ロバストネス図のメタモデル（[24] に一部追記）

ロバストネス図のメタモデル（図 4.11）は，Merah らが UML2 コミュニケーション図の形式手法への変換を議論している論文 [24] 中で定義しているコミュニケーション図のメタモデルに，特化オブジェクトとして BoundaryObject と EntityObject を追記して関連付け，さ

らに Controller を追記して Object，Occurence，および Message に関連付けた．すなわち，Object に起因，または Object がハンドリングする Occurence に対応して Message が送受信され，それに従って Controller が処理する．Controller は BoundaryObject と EntityObject を操作する．

それぞれのメタモデルで規定された関係を使って，変換テンプレート（図 4.8）を構成した．概要を説明すると，ゴールモデルと操作モデルの変換テンプレートでは，ゴールやそれを操作可能化する操作を RefinementPattern を使って関連付け，さらにメタモデルに準じてオブジェクトを関連付けた．同様に，ユースケース図，イベントフロー図の変換テンプレートでは，それぞれ DirectedRelationship，Interaction を使って，洗練パターンのシナリオに合うようにユースケースまたはイベントの流れを関連付けた．また，ロバストネス図の変換テンプレートでは，Occurence を使って，Object，Controller，および Actor を洗練パターンのシナリオに合わせて関連付けた．

振舞いについては，次のふたつを基本の流れとした．

1. エンティティ入力→ゴール/操作/処理→エンティティ出力

2. イベントによる操作起動→操作→操作結果イベント出力

1 項は，ゴール/操作/処理に対してエンティティを入力し，その結果としてのエンティティを出力する基本的な流れを示している．一方，2 項は，操作について，その起動はイベントに起因し，操作の結果としてイベントを出力する基本的な流れを示す．本稿では，前者を開始イベント，後者を結果イベント，また操作と操作を結合するイベントを中間イベントと呼ぶ．さらに，それぞれのイベントに関係する環境エージェントが存在することを基本としている．

以降，どのように構成したかをそれぞれのモデルごとに詳述する．

4.2.1　ゴールモデルの変換テンプレート

Lamsweerde[20] は，抽象ゴールをサブゴールに分解する AND パターンとして，洗練パターンを定義している．洗練パターンによるシナリオは，主にサブゴールの名称と構成で表現される．親ゴールの実現シナリオをより明確に表現するために，ゴール実現に責任を

持つソフトウェアエージェント，ゴール実現を期待される環境エージェント，およびモニタや制御の関心事としての入出力エンティティを，ゴールに関連付けることは有効である[18]．親ゴールの実現シナリオをより明確に表現するために，Lamsweerde の洗練パターン（ゴールモデル）にソフトウエアエージェント，環境エージェント（一部），およびエンティティを関連付けたものを，ゴールモデルの変換テンプレートとした．さらに，操作モデルへの規則的な変換をするために必要となる，洗練パターンの種類, マイルストーン駆動におけるサブゴールの数と実現順番，ケース分解のケース数，分割・統治の分割数，ゴールのエンティティに対する関心内容（入力 or 出力），またはサブゴールの役割などの振舞いに関する情報を持つように拡張されている．

KAOS モデルのメタモデル（図 4.9）を参照して，ゴールモデルの変換テンプレートをどのように構成したかを説明する．ソフトウェアエージェントはゴール実現の主体であるため，親ゴール，各子ゴールとそれぞれを実現するソフトウェアエージェントとを responsibility 関係で接続する．期待ゴールに対しては，その実現を委ねた環境エージェントを assignment 関係で接続し，さらにその期待に対するシステムの受取側としてのソフトウェアエージェントを responsibility 関係で接続する．エンティティはゴール実現に必要な入出力としてゴールと concerns 関係で接続されるが，親ゴールに対して concerns 関係で接続されたエンティティは，その関心部分を引継いだ子ゴールに対しても同じ concerns 関係で接続される．また，サブゴールへの分解にともない抽出されたエンティティが，主にサブゴール等と concerns 関係で接続される．これらの関係は，KAOS モデルのメタモデル（図 4.9）に基づいている．なお，図 4.9 では，エージェント（KAgent）は要件（Requisite）と responsibility 関係にあるが，Requisite はゴール（Goal）を reduces したものなので，Goal も KAgent と responsibility の関係を持つ．KAgent と Requisite 間の responsibility 関係は，KAgent が EnvironmentAgent の時の assignment 関係を含むものとした．

次に，図 4.8 のゴールモデルの列を参照しながら，ゴール変換パターンへの洗練パターンの反映内容を説明する．

○ マイルストーン駆動

順番にマイルストーンを実現していく Milestone-Driven の RefinementPattern で親 Goal と子 Goal を関連付けた．ゴール B は，エンティティ a を入力とし，現状態から

マイルストーン状態を実現するとともにエンティティb を出力とする．ゴール C はエンティティb を入力としてマイルストーン状態から目標状態を実現するとともにエンティティc を出力とする．なお，以降，エンティティの入力は参照も含むものとする．

○　ケース分解

独立したケースごとのサブゴール達成が親ゴールの目標達成に相当する Decomposition-by-Case の RefinementPattern で親 Goal と子 Goal を関連付けた．サブゴール B と C はそれぞれのケースにおいて実現すべきゴールであり，それぞれの実現が親ゴール A の実現に相当する．エンティティa と b はそれぞれサブゴール B における入力と出力であるが，親ゴール A に対しても同様である．これは，サブゴール B の実現が親ゴール A の実現に含まれることによる．エンティティc と d，サブゴール C，および親ゴール A の関係についても同様である．

○　ガード条件導入

現状態時でのガード条件発生で目標を達成する GuardIntroduction の RefinementPattern で親 Goal と子 Goal を関連付けた．サブゴール C は，エンティティb と c をそれぞれ入力と出力とし，ガード条件を実現する．サブゴール D は，ガード条件実現に相当するエンティティc を入力とし，目標実現に相当するエンティティd を出力とする．サブゴール B は，サブゴール D による目標状態が達成されていない限り現状態を維持する．エンティティa は現状態に相当し，すべてのサブゴールから入力として扱われる．また，エンティティa，b，d は，それぞれ a：入力・出力，b：入力，d：出力として親ゴールからも同様に扱われる．

○　分割・統治

単純なサブゴールに分割しそれをすべて実現することによって親ゴールを実現する DivideAndConquer の RefinementPattern で親 Goal と子 Goal を関連付けた．サブゴール B と C は，親ゴール A をふたつに分割したものである．従って，親ゴール A は，サブゴール B と C がともに実現された状態である．ゴールとエンティティの関係はケース分解と同じく以下のようになる．エンティティa と b はそれぞれ入力と出力としてサブゴール B から扱われるが，親ゴール A に対しても同様である．これは，

サブゴールBの実現は親ゴールAの実現に含まれることによる．エンティティcとd，サブゴールC，および親ゴールAの関係についても同様である．

○ モニタ不能駆動

モニタ不能な機能を環境エージェントに委託することによって親ゴールを実現するUnMonitorabilityDrivenのRefinementPatternで親Goalと子Goalを関連付けた．サブゴールBは，エンティティaを入力（モニタ）することによって，そのモニタ情報エンティティbを出力する．但し，対象システムとしてエンティティaのモニタ機能を保有していない場合であり，サブゴールBを期待として環境エージェントA0に委ねる．環境エージェントA0は，入力（モニタ）したエンティティaを対象システムが取扱可能な形式に変換し，ソフトウェアエージェントSBを介してエンティティbとして出力する．サブゴールCは，エンティティbを入力して目標状態を実現し，エンティティcを出力する．親ゴールAは，エンティティaを入力（モニタ）して目標状態を実現し，エンティティcを出力する．

○ 制御不能駆動

制御不能な機能を環境エージェントに委託することによって親ゴールを実現するUnControllabilityDrivenのRefinementPatternで親Goalと子Goalを関連付けた．対象システムとして保有していない制御機能を使って，目標状態を実現する場合である．その制御機能による目標状態の実現を環境エージェントA0に委ねる．サブゴールBは，エンティティaを入力することによって，システムとして可能な範囲で制御処理を実施し制御情報エンティティbを出力する．ソフトウェアエージェントSCは，サブゴールCを期待して，環境エージェントA0に制御情報エンティティbを渡して実際の制御を委ねる．期待Cの実現結果として，実制御結果に相当するエンティティcが出力される．

4.2.2 操作モデルの変換テンプレート

操作モデルの変換テンプレートは，親ゴールを実現する操作の流れを構成している．ゴールモデルでは，エンティティをゴールに対する入出力として関連付けたが，操作モデルで

は，具体的に当該ゴールを実現する操作により入出力される．各操作は，必要なエンティティを入出力することによってそれぞれのサブゴールを実現する．その結果，最終的に親ゴール実現に相当する結果イベントを出力する．また，操作の流れと利害関係者を明示するために，操作を起動する開始イベント，操作間をつなぐ中間イベント，最終結果となる結果イベント，およびそれらイベントと関連付けられる環境エージェントも構成要素として記述している．操作モデルの変換テンプレートは，洗練パターンのシナリオに従ってそれぞれ構成されており，それによってゴールモデル洗練パターンの意味を継承している．

ゴールモデル変換パターンと同様に，KAOS モデルのメタモデル（図 4.9）を参照して，操作モデルの変換テンプレートをどのように構成したかを説明する．操作（KAction(KOperation)）は要求（Requirement）ひいてはゴール（Goal）を操作可能化し，さらにゴール（Goal）は洗練パターン（RefinementPattern）を構成するので，操作（KAction(KOperation)）は洗練パターン（RefinementPattern）を operationalises するように構成できる．その過程で，操作（KAction(KOperation)）はエンティティ（KEntity）またはイベント（KEvent）をオブジェクト（KObject）として input/output する．また，イベント（KEvent）は，操作（KAction(KOperation)）を causes する．それぞれのイベント（KEvent）に対しては，それを controls（例えば，発生させる）したり monitors（例えば，モニタ結果で影響を受ける）する環境エージェント（EvironmentAgent）を関連付け，操作（KAction(KOperation)）に対しては，それを主体的に実行するソフトウエアエージェント（SoftwareAgent）を関連付ける．なお，結果イベントについては，ソフトウェアエージェントと環境エージェントの両方を関連付けている．

図 4.8 の操作モデルの列を参照しながら，操作モデル変換パターンへの洗練パターンの反映内容を説明する．

○　マイルストーン駆動

図 4.8 のマイルストーン駆動ゴールモデルを操作可能化するように操作モデルの変換テンプレートを構成した．マイルストーンに対応するイベント 2 を仲介として操作 B，C を逐次駆動し，親ゴールに相当するイベント A を出力するマイルストーン駆動である．

操作 B は環境エージェント A1 に起因するイベント 1 で起動され，エンティティ a

を入力して，操作結果としてイベント 2 とエンティティb を出力する．イベント 2 は，続けて操作 B を起動する．操作 B は，エンティティb を入力して，操作結果としてイベント A とエンティティc を出力する．A2，A3 はそれぞれイベント 2，イベント A に関連するい環境エージェントである．また，SB，SC は，それぞれ操作 B，操作 C を実行するソフトウェアエージェントである．ここで，イベント A に関連付けられている SA は，イベント A に相当する親ゴールの実現操作を実行するソフトウェアエージェントを示している．

○ ケース分解

それぞれのケースに対応して操作 B または操作 C を実行するケース分解である．それぞれのケースにおける親ゴールの状態に相当するイベント A をケースに準じて出力する．このように，操作 B と操作 C からそれぞれ出力するイベント A はケースに準じた状態となるが，どちらの状態も親ゴールに含まれた状態に相当している．

なお，A1 と 1 など環境エージェントとイベントとの関係，a と B などエンティティと操作との関係，SB と B などソフトウェアエージェントと操作との関係，および SA と A のソフトウェアエージェントと結果イベントの関係は，マイルストーン駆動の場合と同等であるため説明を省略する．ガード条件導入～制御不能駆動についても同様である．

○ ガード条件導入

操作 B で現状態を維持しているときに，ガード条件発生を操作 C が認識すると，それによって操作 D が親ゴールに相当するイベント A を出力する．

現状態時にガード条件が発生すると操作 D を実行するというガード条件導入シナリオを表現するために，操作 B と操作 C のそれぞれの出力に対して AND イベントをとり，両者がともに正の時に操作 D が実行される流れになっている．

○ 分割・統治

サブゴールをそれぞれ独立に実現する操作 B と C の出力イベントを AND で統合して，親ゴールに相当するイベント A を出力する．

直接では実現困難な親ゴールを，便宜的に独立なふたつのサブゴールに分割し，そ

れぞれを実現した結果を統合する操作の流れとした．それを表現するため，イベント1でそれぞれのサブゴールを実現する操作B,Cをともに起動し，それぞれの操作から出力される結果のイベント2,3を多項関係ANDでまとめて最終的な結果イベントAに統合する振舞いとした．

○　モニタ不能駆動

環境エージェントA0によるモニタ情報であるエンティティaを操作Bで取得し，それを使って操作Cが親ゴールに相当するイベントAを出力する．

環境エージェントA0と，システムの利害関係者である環境エージェントA1が関連するイベント1によって操作Bを起動し，モニタ情報であるエンティティaを取り込み，必要とされる情報としてエンティティbを出力する．操作Cはそれを入力として，イベントAを出力する流れとなっている．

○　制御不能駆動

操作Bで定義した制御内容を，操作Cで環境エージェントA0に委託し，その結果親ゴールに相当するイベントAを出力する．

環境エージェントA1に起因するイベント1で起動された操作Bは，システムに可能な形式で制御内容を規定しエンティティbとして出力する．操作Cはそれに引き続いて駆動され，入力した制御内容（エンティティb）に基づいて環境エージェントA0に制御を委託する．その結果として，期待通りに制御された状態としてイベントAとエンティティCが得られる．

4.2.3　ユースケース図の変換テンプレート

ともに振る舞いを定義することからユースケースと操作は親和性が高い．ユースケースにアクターを関連付け，親とサブユースケースの分解関係を明確にするため，ケース分解の場合を除いて，親ユースケースはサブユースケースを <<include>> する．ケース分解の場合は，サブユースケースそれぞれがそのケースにおける親ユースケースに相当するので，親ユースケースとサブユースケースは汎化/特化の関係とした．さらに，それらユースケースを洗練パターンのシナリオに従って関連付けることによって，ゴールモデル洗練パ

ターンの意味を継承する．また，アクターとユースケースは一般的に関連付けられているが，インスタンス化するときに同じ役割のアクターはひとつにまとめることができる．

具体的に，図 4.8 のユースケース図の列を参照しながら説明する．

○ マイルストーン駆動

サブユースケースＢとＣを <<precedes>> で関連付けマイルストーンによる駆動順序を定義している．

○ ケース分解

サブゴールそれぞれの達成が各ケースにおける親ゴールの達成に相当するので，サブユースケースＢとＣそれぞれと親ユースケースは汎化/特化の関係で関連付けている．サブユースケースＢとＣはお互いに独立である．

○ ガード条件導入

サブユースケースＣでのガード条件発生認識結果を受けて，サブユースケースＤで目標を達成するユースケースの実行順序とするため，サブユースケースＣとＤを <<precedes>> で関連付けている．さらに，それらの実行は現状態であることが条件となるので，サブユースケースＣとＤはともに現状態を維持するサブユースケースＢに依存する構成とした．

○ 分割・統治

分割されたサブユースケースＢとＣの実行結果を親ユースケースＡで統合する構成とした．サブユースケースＢとＣはお互いに独立である．

○ モニタ不能駆動

アクター A0 によるモニタ情報をサブユースケースＢで取得し，その結果を受けてサブユースケースＣで目標を達成する構成としている．そのため，サブユースケースＢとＣを <<precedes>> で関連付けその実行順序を明示した．

○ 制御不能駆動

サブユースケースＢで仮想的に制御した内容をサブユースケースＣで具体的な制御

としてアクター A0 に委託するので，サブユースケース B と C を <<precedes>> で関連付けその実行順序を明示した．

4.2.4　イベントフロー図の変換テンプレート

イベントフロー図の表記はシーケンス図に準じている．ライフラインとしてアクター，ソフトウエア機能，エンティティを配置している．さらに，ソフトウエア機能をメインとサブの機能に分け，洗練パターンの親子関係を継承できるようにした．メインの機能はサブの機能を起動し，洗練パターンのシナリオに従って，イベントフローを駆動する．サブの機能はエンティティを入出力し具体的なイベント処理を実行する．洗練パターンそれぞれのシナリオを，主要なメッセージを添えた相互作用フラグメントの組み合わせで表現することにより，ゴールモデルの洗練パターンの意味を継承している．サブの機能におけるイベントフロー相互の関係が単純に時系列でない場合は，分岐，条件判断，並行処理などの制御構造を定義する適切な複合フラグメントを利用してその振舞いのシナリオを表現している．また，4.2 節の最初の方で述べた振舞いについての基本的な流れである以下のふたつのフローに基づいて構成している．

1. エンティティ入力→ゴール/操作/処理→エンティティ出力

2. イベントによる操作起動→操作→操作結果イベント出力

具体的に，図 4.8 のイベントフロー図の列を参照しながら説明する．なお，図中の “A()”，“B()”，“C()”，“1()”，および “2()” はそれぞれメッセージを示す．

○　マイルストーン駆動

マイルストーンの順番に従ったイベントの流れを，InteractionFragment を使って，単純に時系列なフローとして表現している．

○　ケース分解

複合フラグメントの条件分岐（alt）を使い，ケースをガード条件としてそれごとの相互作用を表現している．

○ ガード条件導入

複合フラグメントの特定条件（opt）を使い，ガード条件発生時の相互作用を表現している．ガード条件が発生するまでは，現状を維持している．

○ 分割・統治

分割されたイベントが個々に実行され，最後に統合される必要があるため，複合フラグメントの並行処理（par）を使って相互作用を表現している．分割されたイベントフロー相互の順序性はない．

○ モニタ不能駆動

イベントフローとしては単純に時系列な相互作用であるが，アクター A0 でモニタされた情報をサブ機能 SB で取扱可能な情報として取得した後，サブ機能 SC で処理に使用する流れになっている．

○ 制御不能駆動

イベントフローとしては単純に時系列な相互作用であるが，サブ機能 SB で使用可能な情報を使って仮想的に処理した制御内容を，サブ機能 SC でアクター A0 に委託するフローとなっている．

4.2.5 ロバストネス図の変換テンプレート

操作モデルで操作の流れとして表現された洗練パターンのシナリオを，そのままオブジェクトの絵として貼り付けた．アクターとのインタラクションをバウンダリオブジェクトを介してコントローラに伝え，コントローラはそのインタラクションに必要なエンティティを入出力する．この関係が操作モデルのシナリオに従って関連付けられている．この結果，ゴールモデル洗練パターンの意味を操作モデルを介して継承している．

具体的に，図 4.8 のロバストネス図の列を参照しながら説明する．

○ マイルストーン駆動

コントローラ B によりマイルストーンを実現し，コントローラ C でマイルストーンから目標を達成するように構成している．エンティティオブジェクト a，b は，それ

ぞれマイルストーン実現に関する入力，出力の情報である．また，エンティティオブジェクトbはマイルストーンからの目標達成に関する入力の情報にもなる．その結果目標達成に相当するエンティティcが出力される．現状態，マイルストーン状態，または目標状態ではバウンダリオブジェクト1，2，Aをそれぞれ介して，アクターA1，A2，A3とそれぞれインタラクションを持つ．

○　ケース分解

コントローラBとCがお互いに独立なケースに相当する．それぞれのコントローラにはバウンダリオブジェクトとエンティティオブジェクトが関連付けられている．それぞれのケースの目標達成状態のインタラクションに対応するバウンダリオブジェクトA-1とA-2それぞれは，ともに目標達成状態のインタラクションに相当するバウンダリオブジェクトAの特化状態に相当する．それぞれのケースの場合，現状態と目標状態において，コントローラB，Cはそれぞれバウンダリ1とA-1，バウンダリ2とA-2を介してアクターA1とA3，アクターA2とA3とそれぞれインタラクションを持つ．

○　ガード条件導入

コントローラBが目標状態にない時の現状維持，コントローラCがガード条件発生，コントローラDが現状態時にガード条件発生で目標状態達成に相当する．それぞれのコントローラにはバウンダリオブジェクトとエンティティオブジェクトが関係付けられている．現状態に対応するエンティティaはコントローラBによって維持されるが，それをコントローラCとDも参照する．コントローラCによって，エンティティbを入力してガード条件が発生されると，エンティティcが出力される．ガード条件発生に相当するエンティティcが入力されることにより，コントローラDによって目標状態は達成される．現状態，ガード条件発生状態，または目標状態において，コントローラB，C，またはDはそれぞれバウンダリオブジェクト1，2，またはAを介してアクターA1，A2，またはA3とインタラクションを持つ．

○　分割・統治

達成するに難しい目標を達成が容易なふたつの目標に分割し，それぞれの達成結果

を統合することによって最終目標とするシナリオになっている．コントローラ B と C が分割した目標の達成に相当する．それぞれのコントローラにはバウンダリオブジェクトとエンティティオブジェクトが関係付けられる．コントローラ B は，現状態でのインタラクションをバウンダリオブジェクト 1 を介してアクター A1 と実施し，それによって分割された目標達成に向けて処理が開始される．コントローラ B はエンティティオブジェクト a を入力して分割目標達成に相当するエンティティオブジェクト b を出力する．コントローラ C に相当するもうひとつの分割目標についても同様である．また，分割目標の達成状態に相当するバウンダリオブジェクト 2 と 3 は，最終目標状態に相当するバウンダリオブジェクト A とコンポジションの関係にある．最終目標状態が達成状態であるためには，分割目標それぞれが達成状態でなければならないからである．目標状態では，コントローラ B と C はバウンダリオブジェクト A を介してアクター A2 とインタラクションを持つ．

○ モニタ不能駆動

コントローラ B は，外部のアクター A0 に委託したモニタ結果を取扱い可能な情報として取込む機能に相当する．従って，コントローラ B は，バウンダリオブジェクト 1 を介して，モニタ委託先アクター A0 とシステム利用アクター A1 の両方とインタラクションを持つ．アクター A0 によってモニタされた情報（エンティティオブジェクト a）は，コントローラ B によって取扱可能なエンティティオブジェクト b となり，コントローラ C によって目標達成のために使用される．エンティティオブジェクト c は目標達成状態に相当し，コントローラ C によって出力される．モニタ委託状態（現状態），モニタ委託結果状態，目標達成状態では，それぞれバウンダリオブジェクト 1，2，A を介して，アクター A0 と A1，A2，A3 とインタラクションを持つ．

○ 制御不能駆動

コントローラ C は，外部のアクター A0 にシステムとして不能な制御を委託する機能に相当する．従って，コントローラ C は，バウンダリオブジェクト A を介して，制御委託先アクター A0 とシステム利用アクター A3 の両方とインタラクションを持つ．コントローラ B は，システムとして可能な範囲で仮の制御を実施する機能に相当する．制御要求情報に相当するエンティティオブジェクト a を入力し，可能な範囲での

仮の制御結果に相当するエンティティオブジェクト c を出力する．コントローラ C は，エンティティオブジェクト c に従って，アクター A0 に制御を委託する．制御要求状態（現状態），仮の制御結果状態，制御委託状態では，それぞれバウンダリオブジェクト 1，2，A を介して，アクター A1，A2，A0 と A3 とインタラクションを持つ．

各変換ステップでは，変換元モデルの変換テンプレートを変換先モデルの同一洗練パターンに基づく変換テンプレートで置換えた後，モデル要素のマッピングを行いインスタンス化することで変換を実施する．

第5章　モデル変換アルゴリズムと疑似コードによる具体化

ソムバットら [4, 5] は，操作モデルで表現されたエージェントによる操作の実行関係，操作による出力エンティティとそれを入力する操作の関係，操作に起因したイベントとそれに基づき実行される操作の関係，ヒューマンエージェントにより発生させられたイベントと操作の関係，ひとつのイベントが発生させる複数操作の関係それぞれをクラス図におけるクラスとそのオペレーションによる関係にマッピングし，操作モデルからクラス図を導出している．さらに，洗練パターンによるゴール洗練モデルから読取った OCL 制約とその中に含まれる変数を，クラスのオペレーションと属性に割り付けている．この，洗練パターンから操作の制約，すなわち操作の流れを導出するという考え方，および操作モデルの操作とエンティティまたはイベントとの関係をクラスとオペレーションの関係にマッピングするという考え方を，本稿では理論的に見直し体系化して発展させている．

4.1 節で概説した提案アプローチの各ステップのうち，自動もしくは自動後手動により処理されるのは，STEP1，STEP2(STEP7)，STEP3，STEP5，STEP8，STEP9 である．それらのアルゴリズムについて解説する．

その中で，STEP1 と STEP5 を除く，STEP2(STEP7)，STEP3，STEP8，STEP9 はモデル変換である．モデルの変換規則は，Query/View/Transformation(QVT)[27] を使って定義しており，その概要についてまず解説する．本提案アプローチにおけるモデル変換の変遷と QVT 変換規則との対応をまとめると図 5.1 になる．各モデルを角丸四角形で示し，その中にモデル要素名を記入している．モデルを表現している角丸四角形のさらに内側に配置している角丸四角形は，変換テンプレートを構成する関連要素である．すなわち，当該モデルの関連要素によって振舞いシナリオを表現するように構成されたものが，そのモデルの変換テンプレートということになる．ユースケース図を例に取ると，変換テンプレー

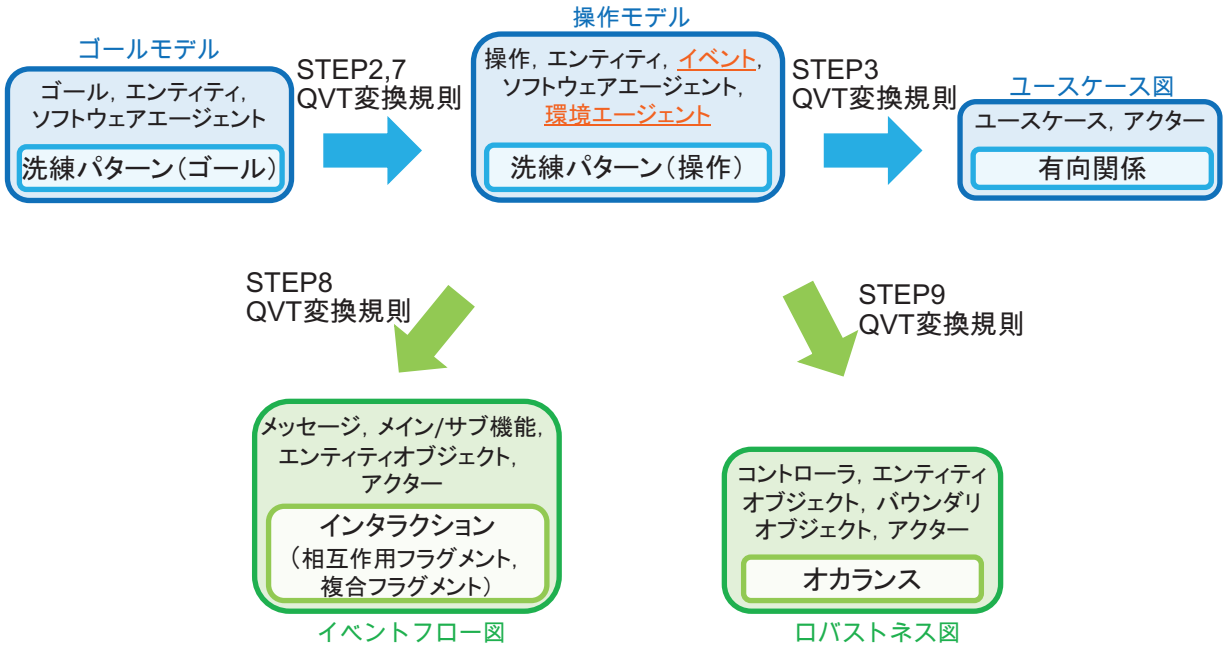


図 5.1: QVT 変換規則によるモデル変換

トは関連要素の有向関係で関連付けられている．また，矢印で変換元モデルと変換先モデルを示している．各 QVT 規則によるモデル変換は，振舞いシナリオを表現する内側の角丸四角形部分（変換テンプレート）を置換え，さらにモデル要素をマッピングすることによって実現される．

ゴールモデルを操作モデルに変換する STEP2,7 の QVT 変換規則は，変換テンプレートに相当する洗練パターン（ゴール）から洗練パターン（操作）への置換えと，ゴールを操作になどのモデル要素間のマッピングを定義している．ここで，操作モデルのモデル要素で下線で表示している イベント と 環境エージェント は，対応する要素がゴールモデルにないため操作モデルではダミーとして扱う．操作モデルをユースケース図に変換する STEP3 の QVT 変換規則は，変換テンプレートに相当する洗練パターン（操作）から有向関係へのマッピングと，操作をユースケースになどのモデル要素間のマッピングを定義している．さらに，操作モデルをイベントフロー図に変換する STEP8 の QVT 変換規則は，変換テンプレートに相当する洗練パターン（操作）からインタラクションへのマッピングと，操作をメッセージになどのモデル要素間のマッピングを定義している．最後に，操作モデルを

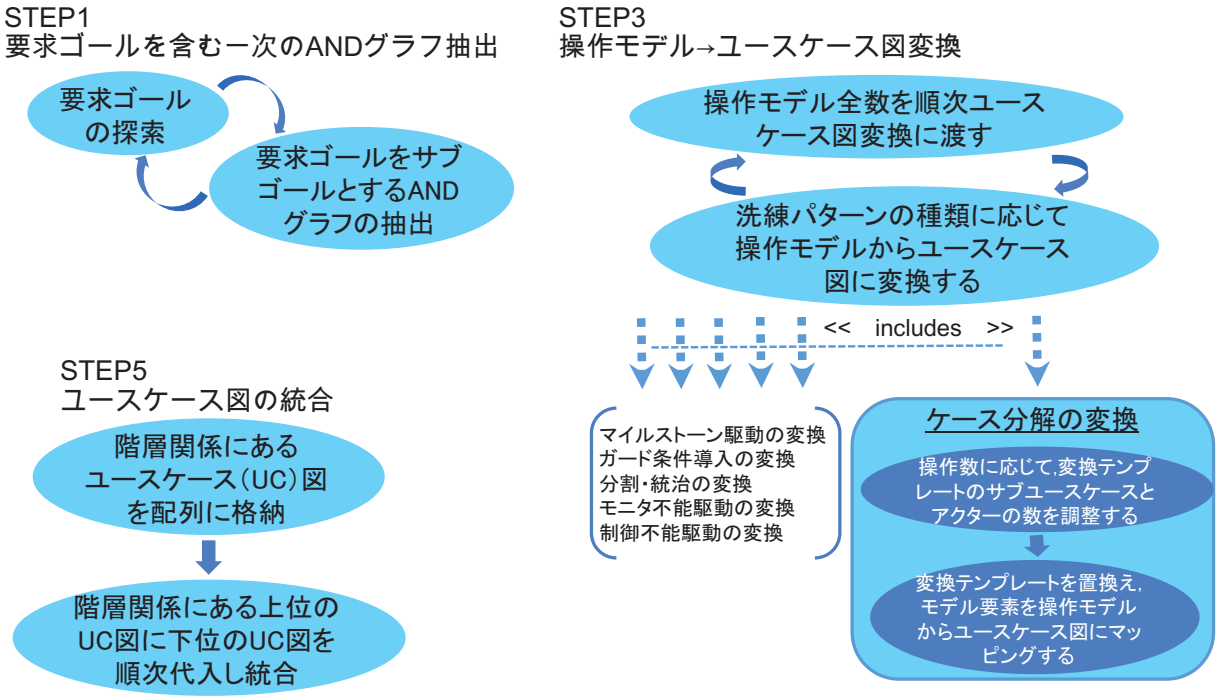


図 5.2: 疑似コードによるアルゴリズムの具体化

ロバストネス図に変換する STEP9 の QVT 変換規則は，変換テンプレートに相当する洗練パターン（操作）からオカランスへのマッピングと，操作をコントローラになどのモデル要素間のマッピングを定義している．

さらに，自動もしくは自動後手動処理可能な次のステップについては，上述した QVT 変換規則に基づき，疑似コードによりアルゴリズムを具体化した．次に，この概要について解説する．STEP1 での要求ゴールを含む一次の AND グラフ抽出，STEP2(7) でのゴールモデルから操作モデルへの自動後手動による変換，STEP3，8，9 での操作モデルからそれぞれユースケース図，イベントフロー図，ロバストネス図への自動変換，STEP5 でのユースケース図の自動統合のアルゴリズムそれぞれが対象である．ただし，STEP1 と STEP5 については，モデル変換ではないため QVT 変換規則は用いていない．

各ステップのアルゴリズムについての概要を図 5.2 に示す．STEP1 では，要求ゴールを探索し，それをサブゴールとする一次の AND グラフを抽出する．これをすべての要求ゴールについて繰り返すが，一次の AND グラフが重複して抽出されないようにする．STEP5 では，まず STEP3 で一次の AND グラフごとに変換したユースケース図を整理し，階層関

係にあるユースケース図を各行に格納した配列を作成する．次に，各行ごとに上位のユースケース図に下位のユースケース図を順次代入して統合する．モデル変換については，STEP3のアルゴリズムを代表して概説する．STEP2(7)，STEP8，STEP9についての概要は省略するが，STEP3と同様である．STEP3では，一次のANDグラフごとにユースケース図への変換を実施する．変換は，洗練パターンの種類に応じた変換機能を呼出して実行する．すなわち，図5.2のようにケース分解の洗練パターンを例に取ると，インクルードされたケース分解の変換機能を呼出す．ケース分解の変換では，ユースケース図の変換テンプレートはケースが2の場合を構成しているので，まずインスタンスに合わせケース数を調整する．次に，操作モデルの変換テンプレートをケース数を合わせたユースケース図の変換テンプレートで置換え，モデル要素をマッピングする．これで変換が完了する．ケース分解以外の場合もほぼ同様な変換手順になっている．

以下，STEP1，STEP2(7)，STEP3，STEP5，STEP8，およびSTEP9について，各STEPごとに節を分けて，QVT変換規則と疑似コードで具体化したアルゴリズムを詳述する．ただし，STEP1とSTEP5はモデル変換ではないので，疑似コードによるアルゴリズムの説明のみである．STEP2(7)，3，および8ではケース分解洗練パターン，またSTEP9ではマイルストーン駆動洗練パターンの変換アルゴリズムを抜粋して説明しているが，他の洗練パターンについても同様なアルゴリズムで変換できる．このように，大部分が自動化可能なアルゴリズムとなっている．

5.1 要求ゴールを含む一次のANDグラフ抽出（STEP1）

要求ゴールをサブゴールに持つ一次のANDグラフを，図5.3のアルゴリズムで抽出する．トップゴールからサブゴールへと要求ゴールを捜索し，重複しないように1段ずつANDグラフを抽出する（1–7行）．当該ゴールが要求ゴールではない場合は，下位の階層へと辿り，そのサブゴールを捜索する（8–11行）．

```
1  SAG()：SAG1(G0：トップゴール)
2  SAG1(G)：if(G：要求ゴール){
3                  if( GをサブゴールとするANDグラフは未抽出){
4                        AG[ i ]：ANDグラフ配列 = GをサブゴールとするANDグラフ；
5                       i = i +1；
6                        return AG[ ]；
7                    } else return;
8                  } else {
9                        for (each G’：Gのサブゴール)
10                           SAG1(G’)；
11                       }
```

図 5.3: 一次の AND グラフ抽出アルゴリズム

5.2 ゴールモデル→操作モデル変換（STEP2(7)）

図 5.4 に示した QVT による変換規則に従い，ゴールモデルから操作モデルに変換する．例えばマイルストーン駆動の場合，マイルストーンを順次達成して目標状態を達成するので，各操作を逐次に実行するモデルに変換する．上段の左右のモデルはそれぞれゴールモデルと操作モデルにおける変換テンプレートのメタモデルである．ともに，KAOS モデルのメタモデル（図 4.9）に基づいている．すなわち，ゴールは洗練パターンで関連づけられ，ゴールを分解して得られる要求（要求ゴール）はアクション（操作）によって操作できるようになるという関係から，操作も洗練パターンで関連付けることができる．また，エージェント，イベント，エンティティなどオブジェクトは，KAOS モデルのメタモデルに基づき，洗練パターンを介してゴールまたは操作に関連付けられる．下段の When 節と Where 節は，上段左右に記述されているモデルの要素間のマッピング規則を示している．

前 STEP で抽出した AND グラフを操作モデルの変換テンプレートで置換えた後，下段の When 節と Where 節の規則に従ってモデル要素間の情報をマッピングすると変換が終了する．ただし，イベントとそれに関連付けられる環境エージェントはこのステップで初めて抽出されるため，それらの名称を最後に手動で入力する．

次に，ゴールモデルから操作モデルへの変換アルゴリズムを具体化した疑似コードを示す．変換アルゴリズムの抜粋として，ケース分解洗練パターンの変換部分を図 5.5 に示す．

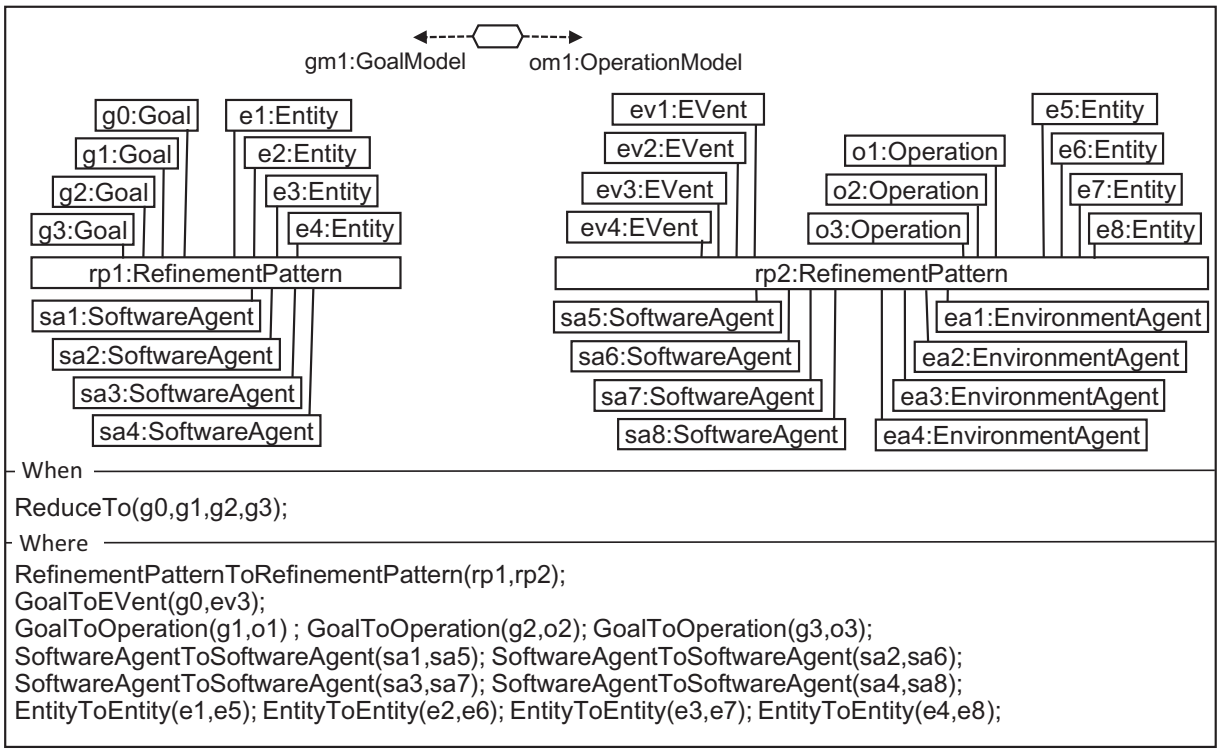


図 5.4: ゴールモデル→操作モデル変換規則（QVT）

STEP1 で抽出したゴールモデルの一次の AND グラフを，変換テンプレート図 4.8 の “ゴールモデル→操作モデル” で置換え（3,4 行および 21,22 行等），モデル要素間で情報をマッピングする（29–32 行）．次にイベントと環境エージェントの名称を入力する（33,34 行）．これらの名称の入力は，自動でダミーのイベントや環境エージェントを配置した後，それに対し手動で入力する．変換テンプレートよりもケース数が多かった場合には，AND グラフのサブゴールの情報に応じ，必要なモデル要素を追加する（25–28 行）．

5.3 操作モデル→ユースケース図変換（STEP3)

STEP2 で変換した操作モデルを，QVT による変換規則（図 5.6）に従って，ユースケース図に変換する．図 5.6 における上段左右のモデルは，それぞれ操作モデルとユースケース図の変換テンプレートのメタモデルである．左側のモデルは，図 5.4 の出力側のモデルと同じものである．右側のモデルは，ユースケースモデルのメタモデル（図 4.10）におけ

```
1//   TO( ): Transformation from AND graphs to Operation models with AG[ ]
2      TO(AG[ ] ) :
3        for each AG'：AG[ ]に蓄積されている抽出ANDグラフ　{
4            switch (洗練パターン)　{
5
・・・・・
21            case ケース分解：
22               ケース分解操作モデル変換テンプレート{
23  //             andg.sgn：ANDグラフのサブゴール数
24  //             rpg.sgn：ゴール洗練パターンのサブゴール数
25                 if (andg.sgn  > rpg.sgn ) {
26                    操作, ケース毎の開始イベント , 操作入出力エンティティ, ソ
                      フトウエアエージェントをandg.sgn – rpg.sgnだけ追加する ;
27                    環境エージェントを追加イベントの分だけ追加する ;
28                        }
29                 サブゴールをそれを実現する操作にマッピングする ;
30                 親ゴールを結果イベントにマッピングする ;
31                 エンティティをそのまま対応させる ;
32                 ソフトウエアエージェントをそのまま対応させる ;
33                 開始イベントの名称を入力する ;
34                 環境エージェントの名称を入力する ;
35               }
36             OG[ no ] : 変換結果操作モデル配列 = ケース分解操作モデル変換テンプ
                 レート ;
37             no = no + 1 ;
38            break ;
・・・・・
90          }
91     }
92    return OG[ ] ;
```

図 5.5: ゴール→操作モデル変換アルゴリズムの抜粋（ケース分解）

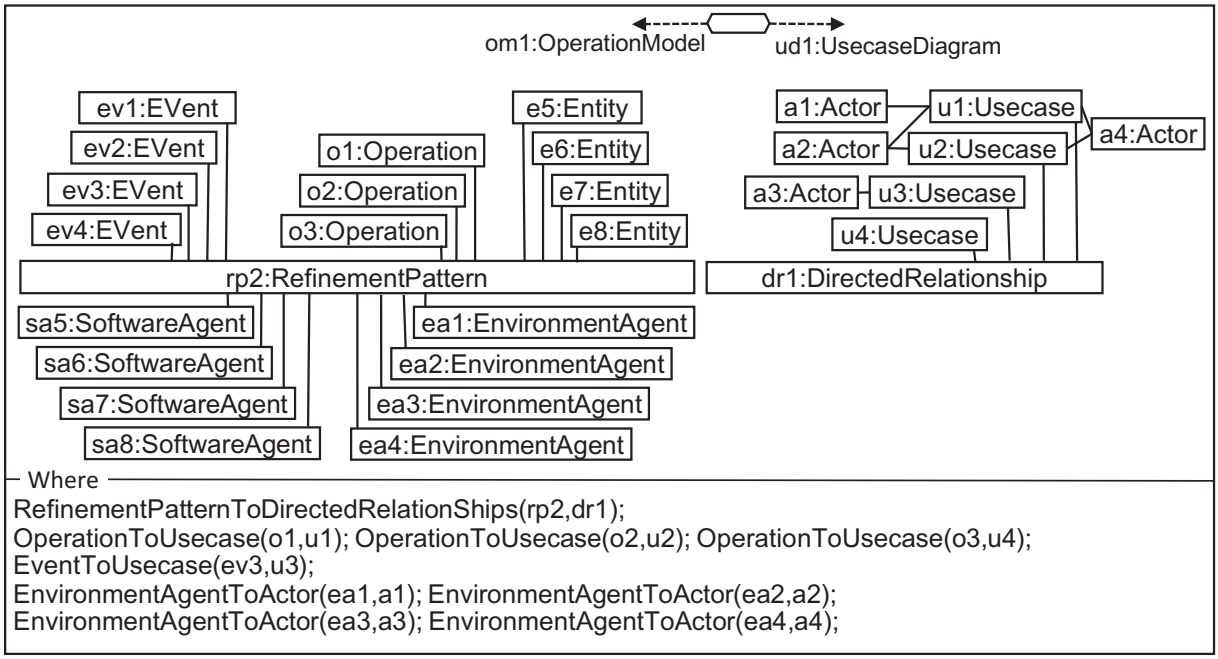


図 5.6: 操作モデル→ユースケース図変換規則（QVT）

る UseCase の部分（図の左側部分）に基づいている．すなわち，洗練パターンの意味（シナリオ）に合うように，DirectedRelationship の関係（Precedes 等）を使ってユースケースを関連付け，そのユースケースにアクターを関連付けている．下段の Where 節の規則に従って，左右モデルのモデル要素間をマッピングする．

次に，操作モデルからユースケース図への変換アルゴリズムを疑似コードにより具体化する．変換アルゴリズムの抜粋として，ケース分解洗練パターンの場合の変換部分を図 5.7 に示す．STEP2 の変換結果操作モデルを，変換テンプレート図 4.8 の “操作モデル→ユースケース図” で置換え（3,4 行および 19,20 行等），モデル要素間で情報をマッピングする（17–29 行）．変換テンプレートよりも操作数が多かった場合には，必要なモデル要素を追加する（23–26 行）．

5.4 ユースケース図の統合（STEP5)

STEP3 または STEP4 での変換結果である各階層一段ごとのユースケースについて，上位階層のユースケースにそれを分解した下位階層のユースケース（サブユースケース）を

```
1//    TU( OG[ ] ): Transformation from Operation models to Use case models with  OG[ ]
2     TU( OG[ ] ) :
3       for each OG'：OG[ ]に蓄積されている変換結果操作モデル{
4           switch (洗練パターン)  {
・・・・・
19           case  ケース分解：
20                 ケース分解ユースケース図変換テンプレート{
21  //                 og.opn：変換結果操作モデルの操作数
22  //                 op.opn：操作モデル変換テンプレートの操作数
23                   if (og.opn > op.opn)  {
24                     サブユースケースとそれに関連付けられたアクターを
                       og.opn - op.opnだけ追加する；
25                     親ユースケースと追加サブユースケースの間に
                       汎化/特化関係を追記する；
26                   }
27                  操作をそれを使用するサブユースケースにマッピングする；
28                  結果イベントをそれを生成する親ユースケースにマッピングする；
29                  環境エージェントをそのままアクターに置換える；
30                }
31             UCD[ nu ]：変換結果ユースケース図配列 = ケース分解ユース
               ケース図変換テンプレート；
32             nu = nu + 1；
33             break；
・・・・・
73         }
74     }
75   return UCD [  ]；
```

図 5.7: 操作モデル→ユースケース図変換アルゴリズムの抜粋（ケース分解）

代入し統合する．これを全階層について実施することにより，最下層のユースケース間の関連に統合することができる．ケース分解洗練パターンでは，サブゴールそれぞれの達成は各ケースにおける親ゴールの達成に相当するので，親ゴールとサブゴールは汎化/特化の関係にある．従って，ケース分解の場合については，この関係をユースケースの統合前後でも明示するために，親ユースケースとサブユースケースを汎化/特化の関係で関連付け，統合後もこの関係を明記する．

次に，ユースケースの統合アルゴリズムを疑似コードにより具体化する．AND グラフ 1 段ごとに作成したユースケース図を，図 5.8 のアルゴリズムで統合する．まず，2 次元配列配列 RUC の各行に階層関係のある変換結果ユースケース図を分配する（4,5 行）．第 1 引数の上位ユースケース図に第 2 引数の下位ユースケースをマージする（6–13 行）．マージアルゴリズム MC() の説明は 14 行以下に示している．上位または下位ユースケースがケース分解パターンである場合，および下位ユースケースが分割・統治パターンである場合は，汎化／特化または集約関係を明示するために下位ユースケース図の親子関係を明示する（16–24 行）．そのほかのパターンについては，親ユースケースを子ユースケースで置換える（24–29 行）．

5.5　操作モデル→イベントフロー図変換（STEP8)

STEP7 で変換した操作モデルを，QVT による変換規則（図 5.9）に従って，イベントフロー図に変換する．例えば，環境エージェント，ソフトウェアエージェントをそれぞれアクターとソフトウェア機能にマッピングし，イベントと操作をメッセージにマッピングする．図 5.9 上段左，右のモデルは，それぞれ操作モデルとイベントフロー図の変換テンプレートのメタモデルである．左側のモデルは，図 5.4 の出力側のモデルと同じものである．右側のモデルは，ユースケースモデルのメタモデル（図 4.10）における UseCaseDescription の部分（図の右側部分）に基づいている．すなわち，アクター，ソフトウェア機能（メインソフトウェア機能またはサブソフトウェア機能），エンティティオブジェクトからなるライフラインと，それらライフライン間のメッセージからなるインタラクションを使って，洗練パターンに合ったイベントフローを構成する．また，インタラクションはインタラクションフラグメントで構成され，複合フラグメントもインタラクションフラグメントのひ

```
1//  MUC( UCD[ ]  ): Integration Use case diagrams with hierarchical relations
2     MUC( UCD[ ] ) :
3  //    RUC[ ] [ ] : 各行にツリー関係にある変換結果ユースケース図を分配
4        for each UCD' : UCD[ ]の要素である変換結果ユースケース図
5          RUC [ i : ユースケース図のツリー番号 ] [ j : 変換結果ユースケース図番号 ] = UCD' ;
6        for each i {
7            for each j
8                  if ( RUC [ i ] [ j ] は上位のユースケース図を持つ) {
9  //      MC( ) : < 第1引数 : 上位のユースケース図 > に < 第2引数 : 下位のユースケース > を
                   マージして , その結果を第1引数に代入する
10                   MC1(UCD'' : UCD'の上位ユースケース図, UCD') ;
11                   RUC [ i ] [ j'' : 上位ユースケース図の格納位置 ] = UCD'' ;
12                  }
13              }
14    MC( ) : MC1( UCD1 : 上位ユースケース図, UCD2 : 下位ユースケース図 )
15    MC1(UCD1,UCD2) :
16          if ((UCD1 || UCD2) == ケース分解パターン)  {
17                replace (UC1 : UCD1中のUCD2の親ユースケース) with UCD2 ;
18                UC1とUCD2のアクターを汎化/特化で関連付ける ;
19                UC1とUCD2のエンティティを汎化/特化で関連付ける ;
20                 } else if (UCD2 == 分割・ 統治パターン) {
21                       replace (UC1 : UCD1中のUCD2の親ユースケース) with UCD2 ;
22                       UC1とUCD2のアクターを集約関係で関連付ける ;
23                       UC1とUCD2のエンティティを集約関係で関連付ける ;
24                         } else {
25 //                          SUCs : UCD2のサブユースケース群
26                           replace (UC1 : UCD1中のUCD2の親ユースケース) with SUCs ;
27                          UCD2のアクターを関連付ける ;
28                          UCD2のエンティティを関連付ける ;
29                              }
```

図 5.8: ユースケース図統合アルゴリズム

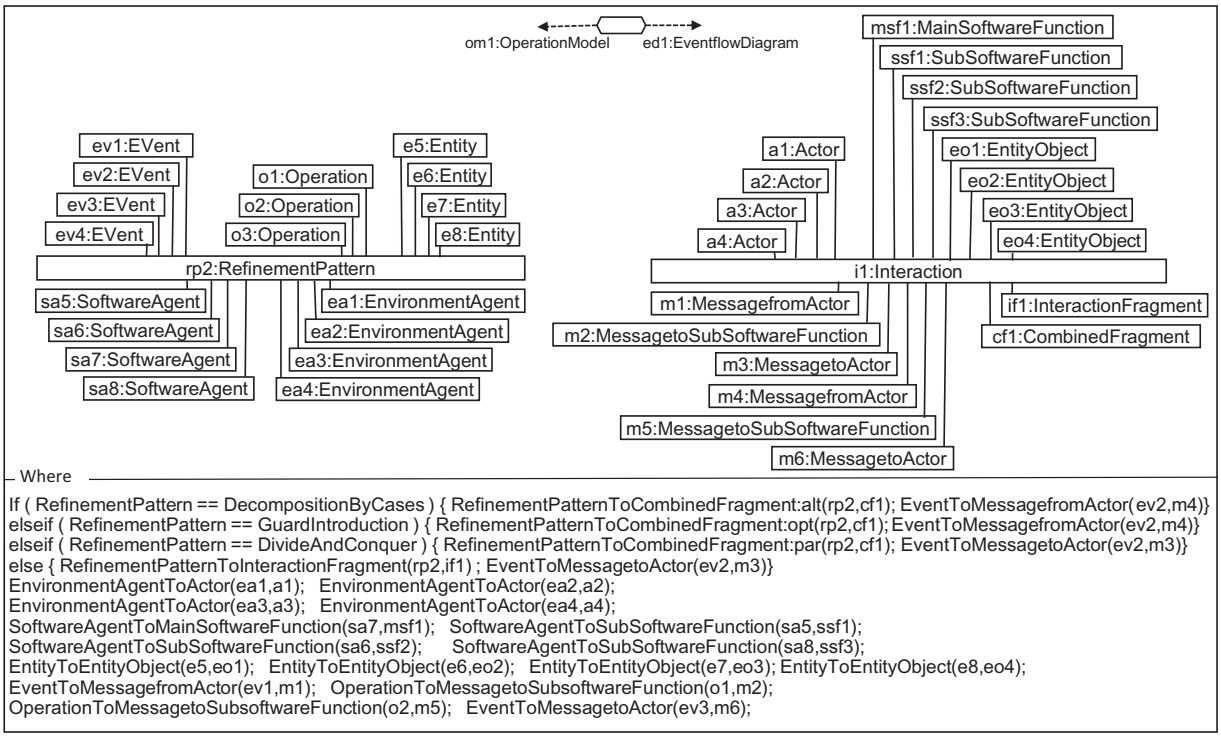


図 5.9: 操作モデル→イベントフロー図変換規則（QVT）

とつである．洗練パターンの種類によって，単純なインタラクションフラグメントを使ったり，複合フラグメントを使ったりする．すなわち，ケース分解の場合はオルタネートの複合フラグメント，ガード導入の場合はオプションの複合フラグメント，分割・統治の場合はパラレルの複合フラグメント，その他の場合は単純なインタラクションフラグメントをそれぞれ使う．下段の Where 節の規則に従って，左右モデルのモデル要素間をマッピングする．

次に，操作モデルからイベントフロー図への変換アルゴリズムを疑似コードにより具体化する．変換アルゴリズムの抜粋として，ケース分解洗練パターンの変換部分を図 5.10 に示す．STEP7 の変換結果操作モデルを，変換テンプレート図 4.8 の “操作モデル→イベントフロー図” で置換え（3,4 行および 22,23 行等），モデル要素間で情報をマッピングする（31–37 行）．変換テンプレートよりも操作数が多かった場合には，必要なモデル要素を追加する（26–30 行）．

```
1//   TE( OG[ ] ) ： Transformation from Operation models to Event flow diagrams with OG[ ]
2     TE( OG[ ] ) :
3       for each OG'：OG[ ]に蓄積されている変換結果操作モデル  {
4           switch (洗練パターン)  {
・・・・・
22           case ケース分解：
23             ケース分解イベントフロー図変換テンプレート{
24  //                  og.opn：変換結果操作モデルの操作数
25  //                  op.opn：操作モデル変換テンプレートの操作数
26                      if ( og.opn > op.opn ) {
27                          ライフラインとして，アクター，サブソフトウエア構成機能，
                            エンティティをog.opn - op.opnだけ追加する；
28                           altの相互作用オペランドをog.opn - op.opnだけ追加する；
29                          上の行で追加した相互作用オペランドとライフラインについて，
                             定型的なイベントフローシーケンスを追記する；
30                            }
31                         ソフトウエアエージェントをその振舞いを定義するサブソフト
                             ウエア構成機能にマッピングする；
32                         結果イベントを生成するソフトウエアエージェントをその振舞いを
                             定義するメインソフトウエア構成機能にマッピングする；
33                       環境エージェントをそのままアクターに置換える；
34                       エンティティをそのまま対応させる；
35                       各ケースの開始イベントをアクターからメインソフトウエア構成機能
                             への駆動メッセージにマッピングする；
36                       操作をメインソフトウエア構成機能から各サブソフトウエア構成機能
                             への駆動メッセージにマッピングする；
37                       終了イベントをメインソフトウエア構成機能からアクターへの
                             実行結果メッセージにマッピングする；
38                     }
39               EFD[ ne ] : 変換結果イベントフロー図配列 = ケース分解イベントフロー図変換
                   テンプレート；
40               ne = ne + 1 ;
41               break ;
・・・・・
100          }
101        }
102      return EFD [ ] ;
```

図 5.10: 操作モデル→イベントフロー図変換アルゴリズムの抜粋（ケース分解）

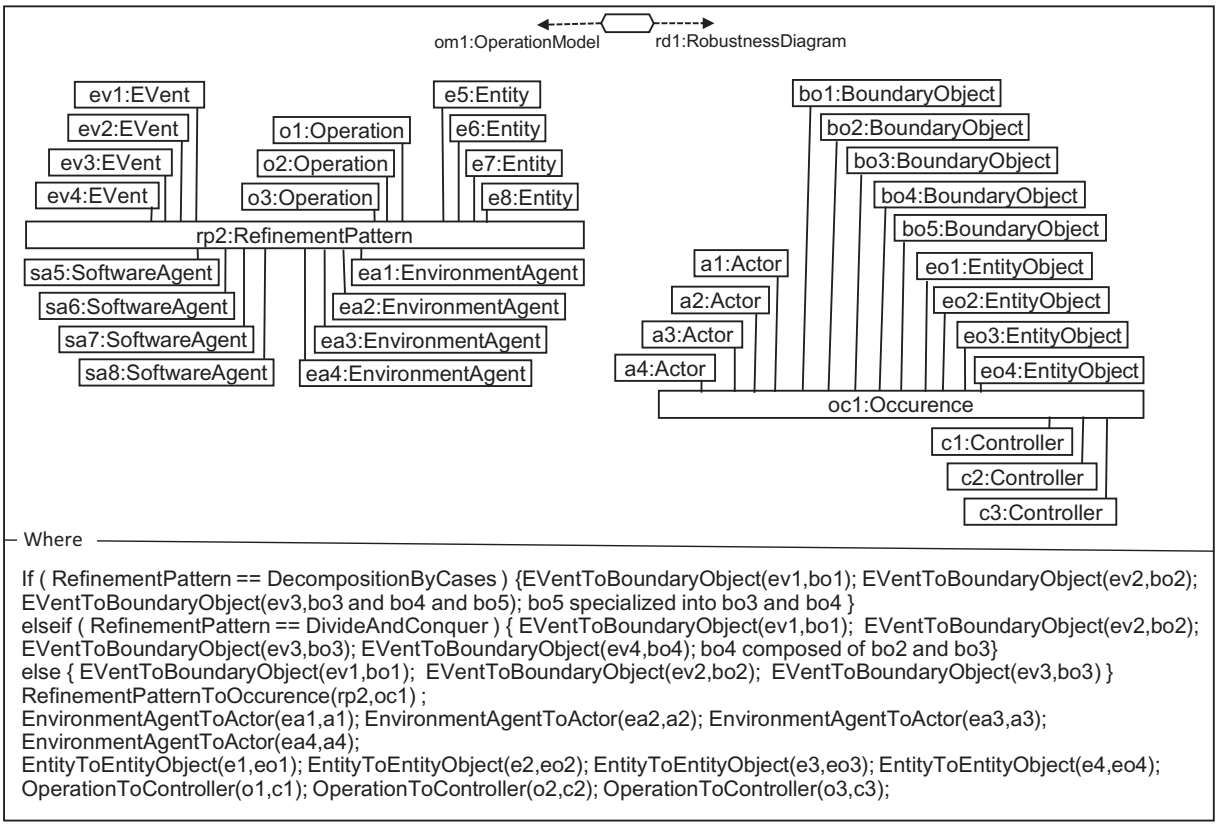


図 5.11: 操作モデル→ロバストネス図変換規則（QVT）

5.6　操作モデル→ロバストネス図変換（STEP9）

STEP7 で変換した操作モデルを，QVT による変換規則（図 5.11）に従って，ロバストネス図に変換する．すなわち，洗練パターンを Occurence にマッピングし，環境エージェント，イベント，エンティティ，操作をそれぞれアクター，バウンダリオブジェクト，エンティティオブジェクト，コントローラにマッピングする．図 5.11 上段左，右のモデルは，それぞれ操作モデルとロバストネス図の変換テンプレートのメタモデルである．左側のモデルは，図 5.4 の出力側のモデルと同じものである．右側のモデルは，ロバストネス図のメタモデル（図 4.11）に基づいている．すなわち，Occurence は洗練パターンのシナリオに従って構成され，それに対してアクター，バウンダリオブジェクト，エンティティオブジェクト，コントローラ等が関係付けられている．なお，ケース分解と分割・統治の洗練パターンについては，結果状態のインタラクションに相当するバウンダリオブジェクトの

構成を特有なものとする．ケース分解については，それぞれのケースにおける目標達成が親ゴールの目標達成に相当するので，親ゴールの結果状態に相当するバウンダリオブジェクトとそれぞれのケースにおける結果状態に相当するバウンダリオブジェクトを汎化／特化の関係で関連付ける．分割・統治については，分割されたそれぞれの目標は全体目標を構成するので，親ゴールの結果状態に相当するバウンダリオブジェクトとそれぞれの分割目標の結果状態に相当するバウンダリオブジェクトをコンポジションの関係で関連付ける．このようにして，下段の Where 節の規則に従って左右モデルのモデル要素間をマッピングする．

次に，操作モデルからロバストネス図への変換アルゴリズムを疑似コードにより具体化する．図 4.8 の “操作モデル→ロバストネス図” 変換テンプレートに基づき，操作モデルからロバストネス図に変換する詳細なアルゴリズムを示す．変換アルゴリズムの抜粋として，マイルストーン駆動洗練パターンの変換部分を図 5.12 に示し説明する．STEP7 で配列 OG に蓄積した変換結果操作モデルそれぞれについて，以下の変換処理を実行する（3 行）．図 4.8 にある「マイルストーン駆動」のロバストネス図で操作モデルを置換え（4–6 行等），モデル要素間で情報をマッピングする（13–16 行）．また，変換テンプレートよりも操作数が多かった場合には，必要なモデル要素を追加する（7–12 行）．最後に，変換結果ロバストネス図を配列 RBD に蓄積する．

5.7 変換アルゴリズムの適用範囲

これまでに，新規に提案するそれぞれの変換アルゴリズムを説明した．そこでは変換テンプレート（図 4.8）や QVT による変換規則（図 5.4，図 5.6，図 5.9，図 5.11）は，マイルストーン駆動洗練パターンにおけるマイルストーンがひとつの場合など各洗練パターンの基本的な型に対する変換を定義している．マイルストーン駆動のマイルストーンが 2 つ以上の場合，ケース分解で 3 ケース以上の場合，分割・統治で 3 分割以上の場合など基本的な型からはずれる場合は，基本的な変換の枠組みを決定した後，ゴールモデルの情報に基づきマイルストーンを増やす等拡張することで対応している．

また，各ゴール，各イベントにそれぞれソフトウエアエージェント，環境エージェントを

```
1//   TR( OG[ ] ) : Transformation from Operation models to Robustness diagrams
2     TR( OG[ ] ) :
3       for each OG': OG[ ]に蓄積されている変換結果操作モデル{
4           switch (洗練パターン)  {
5           case  マイルストーン駆動 :
6                 マイルストーン駆動ロバストネス図変換テンプレート{
7 //                    og.opn：変換結果操作モデルの操作数
8 //                    op.opn：操作モデル変換テンプレートの操作数
9                     if (og.opn > op.opn) {
10                        コントローラ，バウンダリ，アクター，エンティティを
                            og.opn - op.opnだけ追加する ;
11                        上の行で追加したオブジェクトについて，定型的な関係を追記する;
12                      }
13                    環境エージェントを対応するアクターにそれぞれマッピングする;
14                    エンティティを対応するエンティティにそれぞれマッピングする;
15                    開始，中間，終了のイベントを対応するバウンダリにそれぞれマッピン
                        グする;
16                    操作を対応するコントローラにそれぞれマッピングする;
17                }
18          RBD[ nr ] : 変換結果ロバストネス図配列 = マイルストーン駆動ロバストネス図変換
              テンプレート ;
19          nr = nr + 1 ;
20            break;
・・・・・
85        }
86    }
87    return RBD[ ] ;
```

図 5.12: 操作モデル→ロバストネス図変換アルゴリズムの抜粋（マイルストーン駆動）

ひとつずつ割り当てているが，それらは同一エージェントであっても良い．また，親ゴールに複数のエージェントが関与する場合やゴールに関連するオブジェクトが複数ある場合は，できるだけそれらを汎化した名称にしたり，それらを併記した名称とすることで対処できる．逆にない場合には，とりあえずダミーとして付与し，STEP8 の終了後に見直すことによって必要であれば修正する．なお，STEP6 の要求ゴールの分解はゴールとイベントが 1 対 1 になると終了するので，ゴールに対応するイベントは必ず存在する．

提案する変換アルゴリズムは，洗練パターンを用いてモデリングしたゴールモデルを対象としている．洗練パターンを使用していない AND/OR 分解の部分は対象外のため変換されない．その詳細は本稿の範囲外となるので省略するが，STEP4 または STEP8，STEP9 の終了後に見直しが必要である．OR 分解の場合は，代替処理になるので修正せずそのまま分離するのが適切であると考える．AND 分解の場合は，必須処理になるので，ユースケースもしくはイベント処理，コントローラに関連する機能として追加するのが適切であると考える．

また，アクターとユースケースとの関連で多重度を明記する場合があるが，現時点では対応していない．しかし，環境エージェントとそれに関するイベントの組と操作との関連に多重度の情報を付加することによって対応可能と思われる．これは操作モデルを拡張することになるが，今後の検討課題としたい．

第6章　ゴール指向UML設計手法の適用事例

4章から5章で，KAOSモデルの成果物である要求定義モデルを，設計工程への入力となるユースケースモデルや予備設計結果であるロバストネス図に変換するアプローチを提案し，その新たな変換アルゴリズムを具体的に説明した．ここでは，その提案アプローチの適用方法を具体的に示すために，踏切制御に関する遮断バー安全制御のシステムを事例として提案アプローチを適用し，その適用手順と結果をSTEPを追いながら順番に説明する．本事例は，KAOSゴールモデルの例として2章で説明している．

遮断バー安全制御のゴールモデル例である図2.1について，不要な説明を省き一部のゴールを追加したものを図6.1に示す．このゴールモデルは，表2.1の洗練パターンを使って，トップゴールは階層的に洗練され，リーフゴールとして要求ゴールが抽出されている．Gxxxは要求ゴールを含むゴールであり，Pxxは洗練パターンによる一次のANDグラフである．G0，G1，G21，G221は，順に，最上位の層，2番目の層，3番目の層，4番目の層にそれぞれ属している．P0，P2とP3，P22とP32は，順に，最上位から2番目の層への，2番目の層から3番めの層への，3番めの層から4番目の層へのAND分解である．P0は分割・統治，P2とP3はガード条件導入，P22とP32はケース分解の洗練パターンをそれぞれ適用している．なお，P0は分割・統治の洗練パターンであるため，G1，G2，G3はそれぞれ要求ゴールのグループを構成するとみなすことができ，それぞれのグループの要求ゴールがすべて満たされることによってG0は実現される．最下層のG1，G21， G23，G221，G222，G31，G33，G321，G323が抽出された要求ゴールである．要求ゴール間の関連は，このように洗練パターンによる階層構造によって示される．例えば，G221とG23はP22とP2によって関連付けられている．

図6.1を適用事例として，提案アプローチによる変換の流れを次節から具体的に説明する．

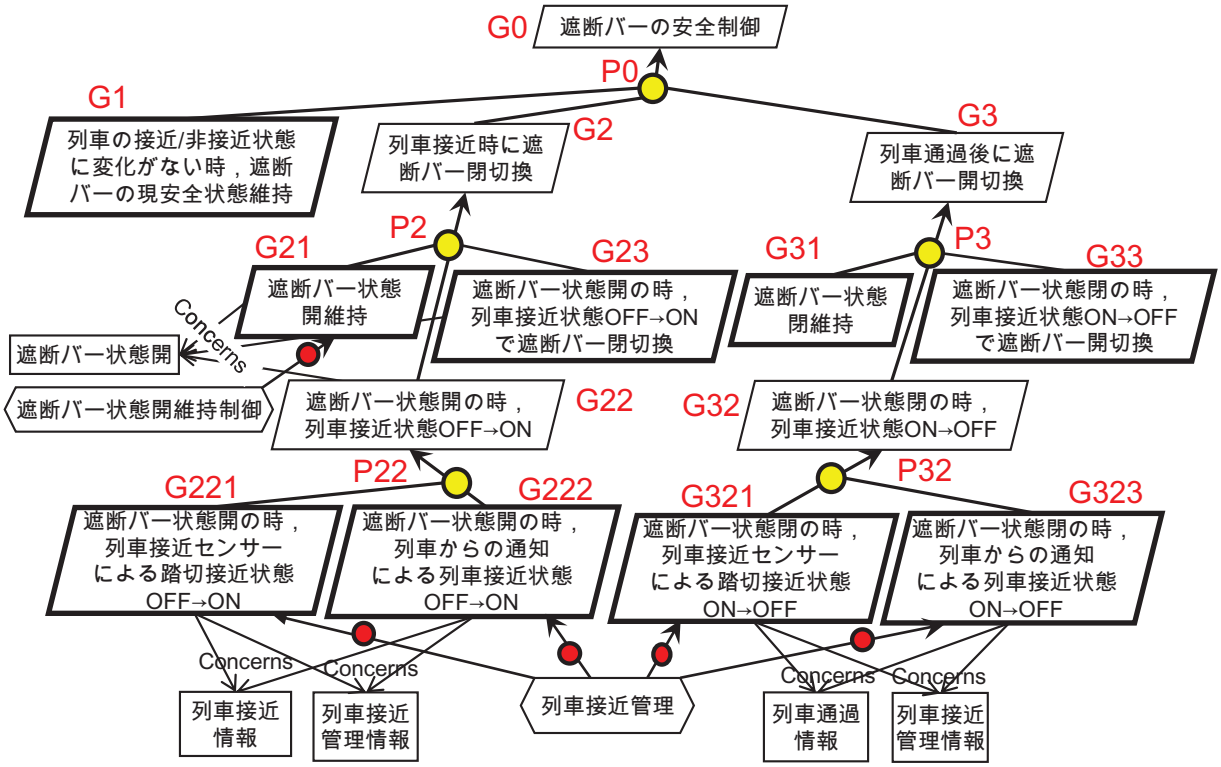


図 6.1: 遮断バーの安全制御モデル

6.1　要求ゴールを含む一次の AND グラフ抽出 (STEP1)

図 6.1 において，要求ゴールを含む AND グラフ一段の抽出例として，G2 を分解している P2 と P22 がある．複雑化による図の認識悪化を防ぐためゴールに関連するソフトウェアエージェントやエンティティを一部省略している．P2 はガード条件導入の洗練パターンによる分解である．遮断バー状態開の維持が達成されているときに，列車接近状態の OFF から ON への切換えが達成されると遮断バー状態閉への切換が達成されるシナリオになっている．また，P22 はケース分解洗練パターンによって分解されている．列車接近状態の OFF から ON への切換は，列車接近センサーの検出によるケースと接近列車からの通知によるケースのふたとおりである．

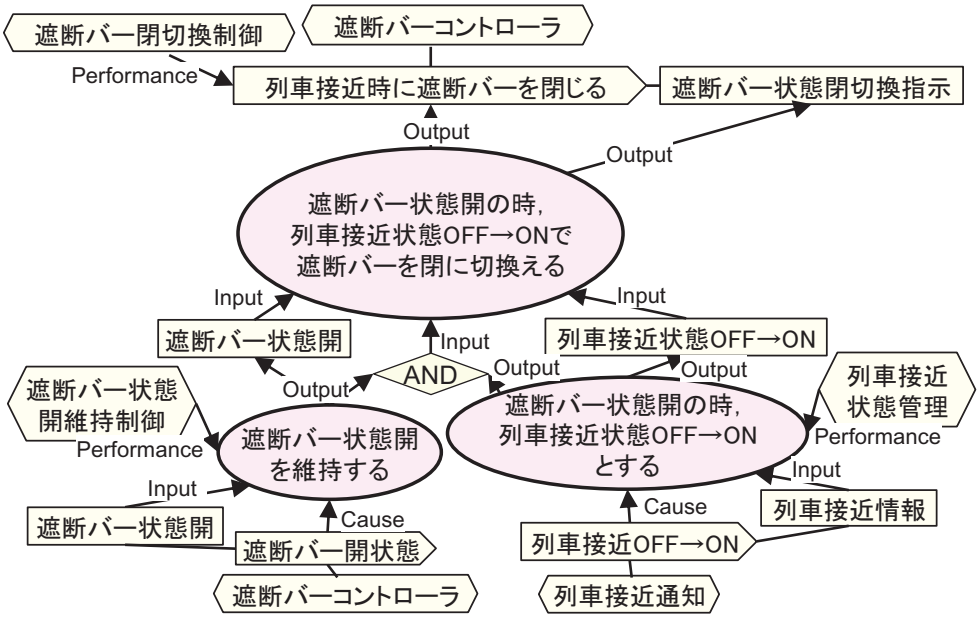


図 6.2: 遮断バー閉切換操作モデル

6.2 ゴールモデル→操作モデル変換（STEP2）

P2，P22 は，QVT 変換規則（図 5.4）に従って，それぞれ図 6.2，図 6.3 の操作モデルに変換される．これらの変換において，イベントと環境エージェントそれぞれはダミーとして配置される．その後，それぞれ内容に基づいた名称が手動で入力される．例えば，図 6.2 の操作「遮断バー状態開を維持する」を起動するイベント「遮断バー開状態」とそれを発動する環境エージェント「遮断バーコントローラ」などである． これら操作モデル図の上下位置関係は，ゴールモデル図 6.1 の上下位置関係と対応が取りやすいように，下から上の方向に開始イベント，操作，結果イベントの順に配置している．図 6.1 のゴールモデルでは，複雑化による図の認識悪化を防ぐためゴールに関連するソフトウェアエージェントやエンティティを一部省略しているが，操作モデル図 6.2 と図 6.3 はそれらが省略されていないゴールモデルからの変換結果を示している．

ガード条件導入の変換テンプレートを適用した図 6.2 では，遮断バー状態が開に維持されている時に，列車接近状態が OFF → ON に変わったら（ガード条件発生）, 遮断バーを閉に切換える操作の流れになっている．

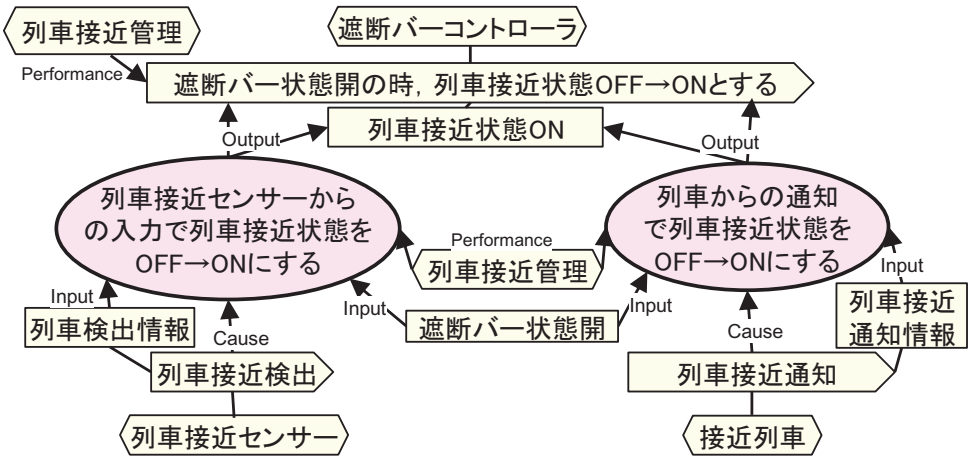


図 6.3: 列車接近状態 ON 操作モデル

また，ケース分解の変換テンプレートを導入した図 6.3 では，列車接近センサーからの検出情報入力の場合または列車からの接近通知の場合それぞれにおいて（ケース分解），列車接近状態が OFF → ON に変わることが示されている．

操作の階層構造について説明すると，図 6.2 が上の階層でその下の階層に図 6.3 がある．すなわち，図 6.2 の操作「遮断バー状態開の時，列車接近状態 OFF → ON とする」を図 6.3 の 2 つの操作で置き換えることによって，要求ゴールに対応する操作間の関連が明確になる．このように，ゴールモデルの洗練パターンのシナリオが操作モデルに継承されている．

以降，説明のため，最初の操作の要因となるイベントを開始イベント，操作の結果発生し次の操作を駆動するイベントを中間イベントもしくは駆動イベント，および一連の操作の最終結果として出力される操作結果のイベントを結果イベントと呼ぶ．

6.3　操作モデル→ユースケース図変換（STEP3）

次に，操作モデルを洗練パターンに基づいてユースケース図に変換する．操作モデル図 6.2，図 6.3 は，QVT 変換規則（図 5.6）に従って，それぞれ図 6.4，図 6.5 のユースケース図に変換される．操作モデルの操作はシステムをどのように操作してゴールを実現するかを定義しているため，ユースケースにマッピングすることができる．洗練パターンによる階層ごとにユースケース図に変換される．

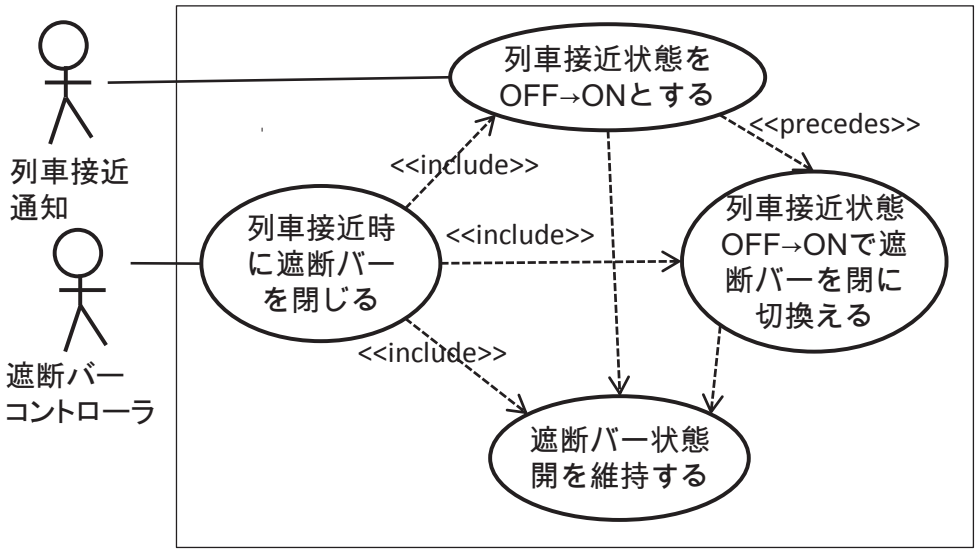


図 6.4: 遮断バー閉切換ユースケース図

具体的にはまず，洗練パターンに対応するゴールモデルから操作モデルへの置換えと同様に，操作モデルからユースケース図に置換えられる．この場合も，洗練パターンごとに対応するユースケース図変換テンプレートを利用する．次に，結果イベントが親ユースケースに，操作がサブユースケースにそれぞれマッピングされる．親ユースケースはサブユースケースを <<include>> している．また，サブユースケースは洗練パターンの手続に合うようにお互いに関連している．さらに，開始イベント，中間イベント，または結果イベントに関連付けていた環境エージェントは，ユースケースに関連するアクターにマッピングされる．最下層のユースケースが要求ゴールに対応する．

図 6.4 は，ガード条件に対応するユースケース「列車接近状態を OFF → ON にする」が先行して実行されれば，ユースケース「列車接近状態 OFF → ON で遮断バーを閉に切換える」が実行されることを示している．このとき，この 2 つのユースケースはユースケース「遮断バー状態開を維持する」に依存している．ガード条件導入洗練パターンのシナリオが，操作モデルを介してユースケース図に継承されているのが分かる．

図 6.5 は，ユースケース「列車接近状態を OFF → ON とする」は「列車接近センサーからの入力で列車接近状態を OFF → ON にする」と「列車接近からの通知で列車接近状態を OFF → ON にする」のふたつのユースケースに特化できることを示している．ケース分解

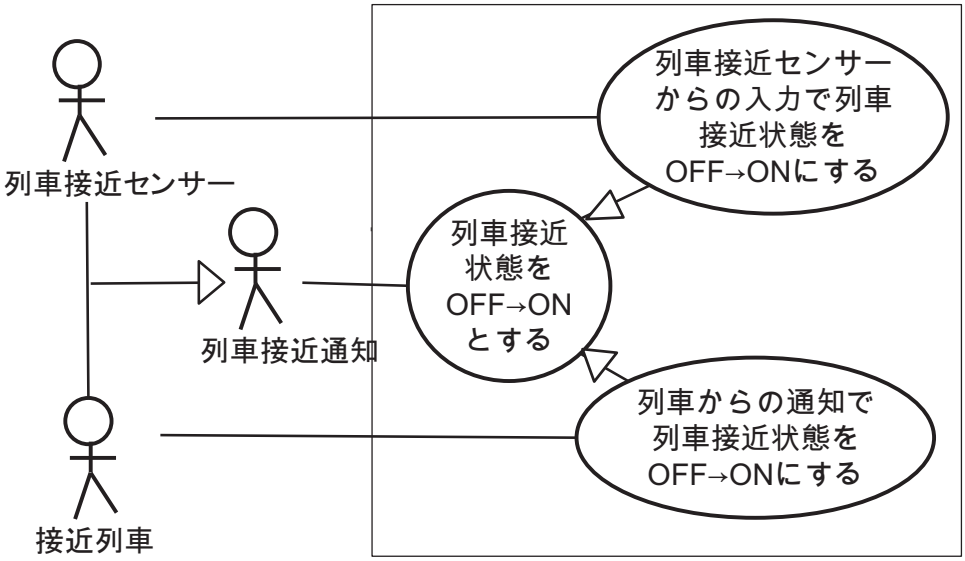


図 6.5: 列車接近状態 ON ユースケース図

による洗練パターンは汎化／特化関係のユースケースに変換される．

6.4　ユースケース図と KAOS モデルの相互洗練（STEP4）

前節で導出した図 6.4 と図 6.5 のユースケース図は，KAOS モデル上の要求ゴールをそのままの構成でユースケース図にマッピングしたものであるから，KAOS モデルのセマンティクスを引き継いでいると言える．しかし，ユースケース図としては，共通ユースケースの抽出，ユースケースの分割／統合，アクターの構成，アクターとユースケースの関連等もう少し洗練が必要である．

ここでは，共通ユースケースの抽出を例にとって説明する．前節で導出した図 6.4 のユースケース図を見るとユースケース「遮断バー状態開を維持する」がある．同様に，本稿では割愛したが図 6.1 のゴールモデル全体を変換したユースケース図を見ると，G1 に対応するユースケース「列車の接近/非接近状態に変化がない場合は遮断バーの現安全状態を維持する」と G31 に対応するユースケース「遮断バー状態閉を維持する」を類似のユースケースとして見つけることができる．これらを見比べると，遮断バーのあるべき状態に合わせて遮断バーの現状態を維持するユースケース「遮断バーの現状態を維持する」を抽出する

ことができる．これをゴールモデルにフィードバックすると，図 6.6 のように G21 のゴールを遮断バーの状態が開であることを確認するゴールと遮断バーの現状態を維持する共通ゴールに分割・統治洗練パターンで分解できる．ゴール「遮断バーの現状態維持 (C)」の (C) は共通ゴールであることを示している．図 6.6 のように，ゴールモデルは要求ゴール

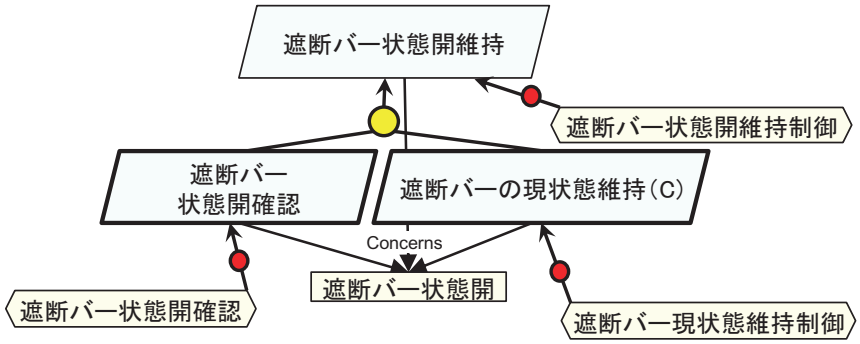


図 6.6: 遮断バー開維持改訂ゴールモデル

に対して責務を負うソフトウェアエージェントや要求ゴールが関心を持つエンティティを含めて記述されていることから，責任モデルとオブジェクトモデルの要素も含んでいることがわかる．従って，これらのモデル要素の修正も合わせて実施する．

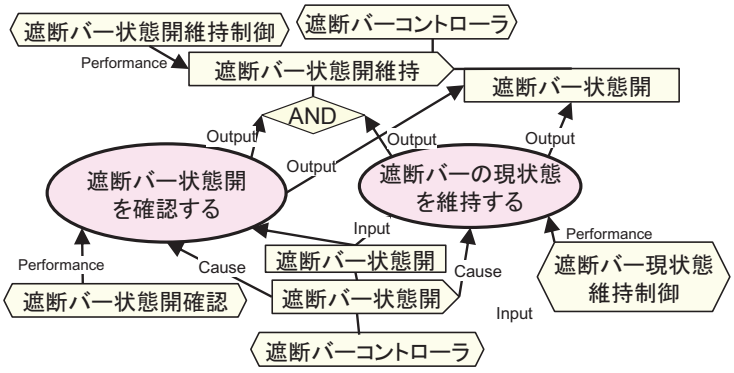


図 6.7: 遮断バー開維持改訂操作モデル

改めて，図 6.6 を分割・統治の洗練パターンに基づいて操作モデル，さらにユースケース図に変換する．その結果，操作モデルの図 6.7 がまず得られ，次にそれから変換されたユースケース図の図 6.8 が得られる．図 6.7 の操作モデルは，遮断バー状態開を確認しな

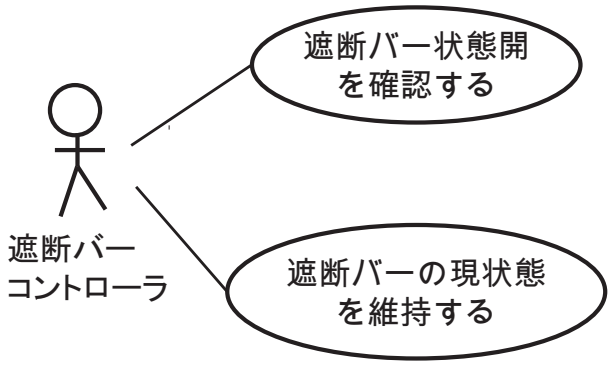


図 6.8: 遮断バー開維持改訂ユースケース図

がら遮断バーの現状態を維持するという分割・統治の操作パターンを示し，図 6.8 のユースケース図では親ユースケースは省略されているが，遮断バーの状態が開であることを確認しながら，遮断バーの現状態を維持する共通機能が使用されることを示している．

6.5　ユースケース図の統合（STEP5）

最下層のユースケースを上位のユースケースに代入していくことにより，最終的なユースケース図に変換できる．上位のユースケースも洗練パターンによって関連付けられているため，下位のユースケースを上位のユースケースに代入していくことにより，最下層ユースケースそれぞれが洗練パターンの組み合わせによって関連付けられる．このようにして，ユースケース図は統合される．それぞれのユースケースは要求ゴールに対応している．具体例で説明する．図 6.8 のユースケース図が図 6.4 のユースケース図に組込まれ，さらにケース分解 AND グラフ P22 から変換したユースケース図（図 6.5）が組込まれると，図 6.1 の G2 を P2 と P22 で詳細化した系統のユースケース図として図 6.9 に統合される．

6.6　洗練パターンによる要求ゴール分解（STEP6）

KAOS による要求モデルで最下層のリーフゴールとして抽出した要求ゴールを，さらにソフトウェアを構成する機能が果たすべきサブゴールに洗練する．この洗練にも，表 2.1 の洗練パターンを使う．要求ゴールをサブゴールに洗練する目的は，要求ゴールの実現手

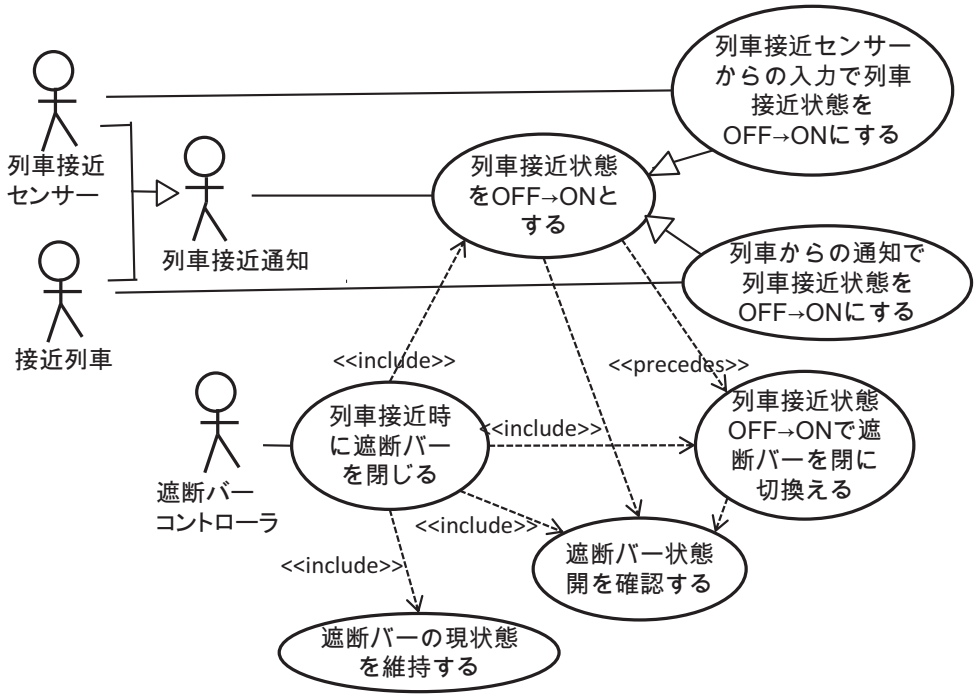


図 6.9: 遮断バー閉切換改訂ユースケース図

順を示すイベントフローを導出することである．従って，イベントフローが明確になるまで洗練する．さらなる洗練が必要かどうかの判断基準はイベントフローが明確になっているかどうかであり，それぞれのサブゴールがひとつのイベントに明確に対応するまで洗練する．要求ゴールまたはサブゴールに複数のイベントフローが暗示的に含まれている場合はさらに洗練が必要であると判断する．この要求ゴールの洗練には，Girier ら [11] がプロトタイプを紹介している KAOS モデリングの視覚化ツール Visual-K を使用すると効率的である．Visual-K には，マウス操作によるズームイン・アウト，下位ゴールの折りたたみ，特定ノードの拡大・縮小，追加ノード間エッジの自動描画などの視覚化機能があり，洗練対象の要求ゴールに注目できることで，効率的な洗練作業が可能となる．また，システムが対象としているドメインについて，設計者が精通しているとは限らない．そのような場合，吉田ら [37, 52] が提案しているドキュメントから機能要求を半自動で抽出するアプローチを使うと，サブゴールの候補を抽出するのに効果的である．前節で作成した図 6.9 に含まれるユースケース「列車接近センサーからの入力で列車接近状態を OFF → ON にする」を例にとり説明する．

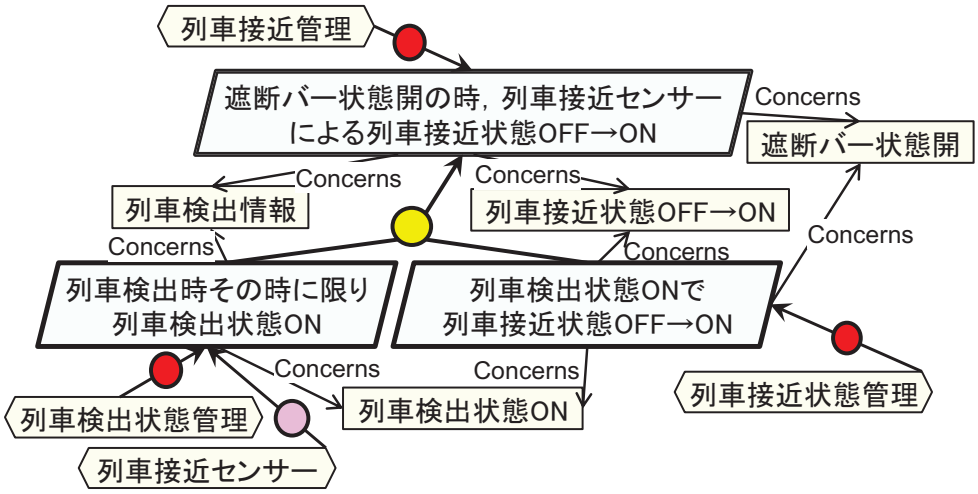


図 6.10: センサーによる接近検出ゴールモデル

G221「遮断バー状態開の時，列車接近センサーによる列車接近状態 OFF → ON」は，ソフトウェアでは検出不能な列車接近を環境エージェント「列車接近センサー」による検出に委ねるサブゴール「列車検出時その時に限り列車検出状態 ON」とその列車検出状態 ON に従って接近状態を管理するサブゴール「列車検出状態 ON で列車接近状態 OFF → ON」にモニタ不能駆動洗練パターンを使って分解でき，その結果は図 6.10 である．その中で，例えば，ゴール「列車検出時その時に限り列車検出状態 ON」はソフトウェアエージェント「列車検出状態管理」によって実現され，環境エージェント「列車接近センサー」が提供するエンティティ「列車検出情報」を参照して「列車検出状態 ON」を出力する．同様にソフトウェアエージェント「列車接近状態管理」は，エンティティ「列車検出状態 ON」を参照し「列車接近状態 OFF → ON」を出力することによってゴール「列車検出状態 ON で列車接近状態 OFF → ON」を実現する．以上が，要求モデルで抽出した要求ゴールをさらに個々のソフトウェア構成機能で実現すべきゴールに詳細化する方法である．

6.7　ゴールモデル→操作モデル変換（STEP7）

次に，要求ゴールを洗練したゴールモデルを 6.2 節と同様な手順で操作モデルに変換する．ゴールモデル図 6.10 を，QVT 変換規則（図 5.4）に従って，モニタ不能駆動の操作モデルに変換すると図 6.11 になる．モニタ不能な部分を列車接近センサーに委ねたことに基

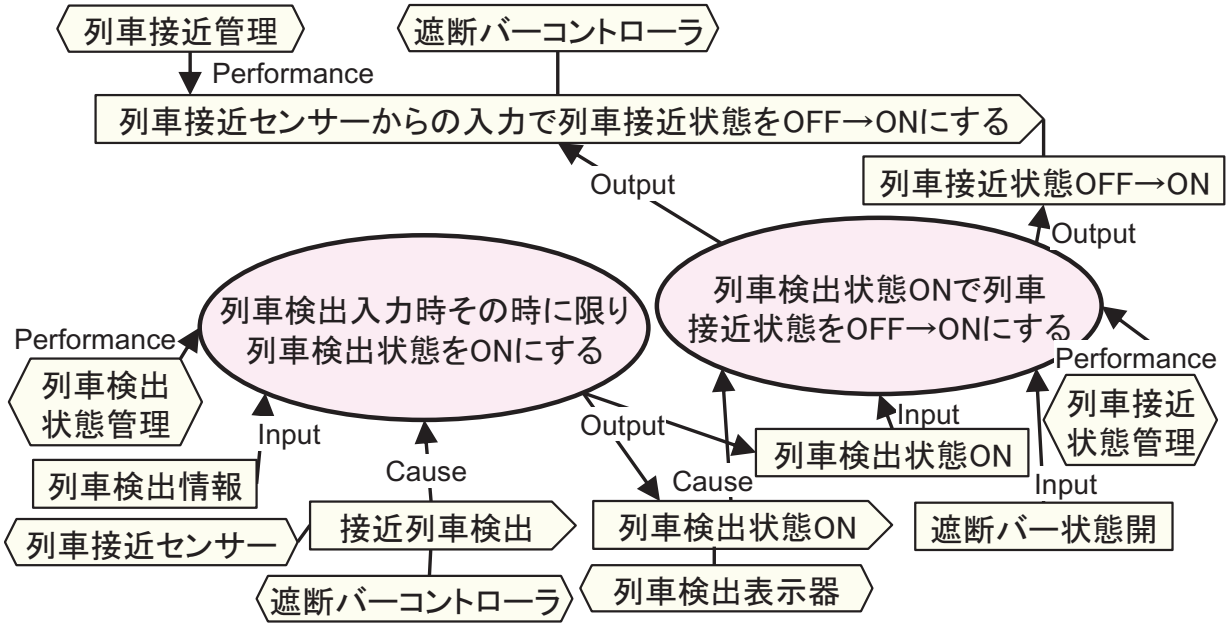


図 6.11: センサーによる接近検出操作モデル

づく操作「列車検出入力時その時に限り列車検出状態を ON にする」を受けて操作「列車検出状態 ON で列車接近状態を OFF → ON にする」を実行することを示しており，前 STEP で要求ゴールを分解したモニタ不能駆動洗練パターンのシナリオを継承している．ここで，サブゴール実現に関する開始，中間，結果の各イベントと，それらイベントに関連する環境エージェントである遮断バーコントローラと列車検出表示器が，6.2 項同様，ダミーとして変換された後，手動で名称が入力される．

6.8 操作モデル→イベントフロー図変換（STEP8）

6.7 節で作成した操作モデル図 6.11 を，QVT 変換規則（図 5.9）に従って，モニタ不能駆動のイベントフロー図に変換すると図 6.12 になる．列車接近センサーからの「列車接近検出」メッセージ入力を処理した後,「列車検出状態 ON で列車接近状態を OFF → ON にする」を処理する流れになっている．この結果から，ゴールモデルのモニタ不能駆動洗練パターンのシナリオを，操作モデルを介して継承していることがわかる．

変換内容を以下に示す．

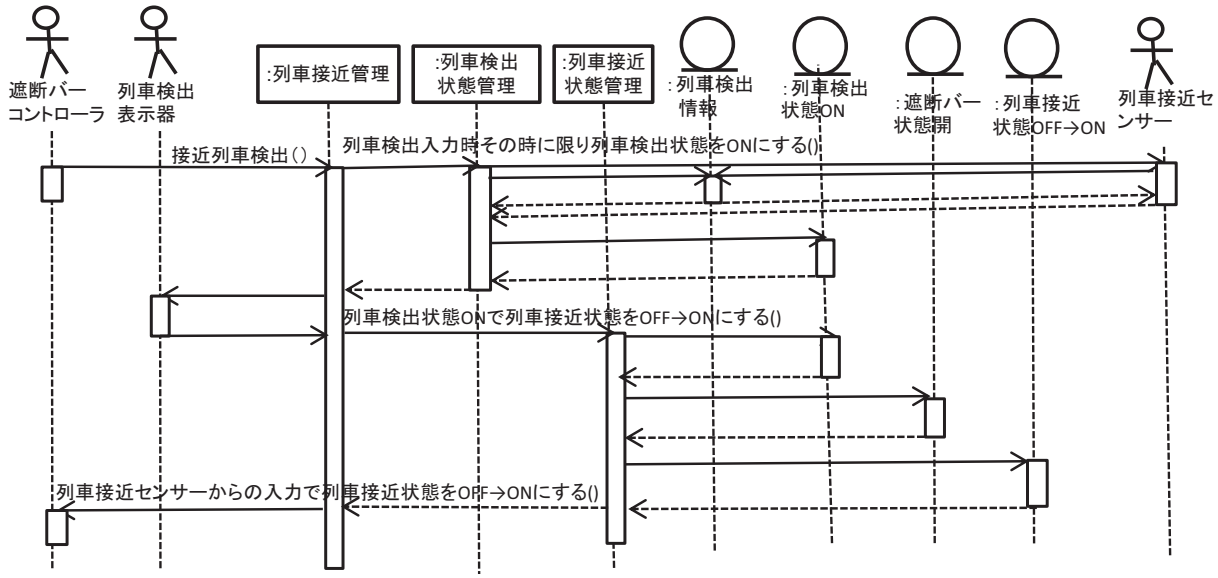


図 6.12: センサーによる接近検出イベントフロー図

1. モニタ不能駆動のイベントフロー図変換テンプレートへの置換え

2. ライフラインへの要素のマッピング

   - 図 6.11 で操作を駆動するイベント「接近列車検出」，中間イベント「列車検出状態」，最終結果の結果イベント「列車接近センサーからの入力で列車接近状態を OFF → ON にする」にそれぞれ関連付けられた環境エージェント「列車接近センサー」,「列車検出表示器」，「遮断バーコントローラ」がライフラインのアクターにマッピングされる.
   - 結果イベント「列車接近センサーからの入力で列車接近状態を OFF → ON にする」を実行するソフトウェアエージェント「列車接近管理」がライフラインのメインソフトウェア構成機能にマッピングされる,
   - 操作「列車検出入力時その時に限り列車検出状態を ON にする」，操作「列車検出状態 ON で列車接近状態を OFF → ON にする」を実行するそれぞれのソフトウェアエージェント「列車検出状態管理」と「列車接近状態管理」がそれぞれライフラインのサブソフトウェア構成機能にマッピングされる.
   - 操作によって入出力されている 4 つのエンティティ「列車検出情報」,「列車検出

状態 ON」,「遮断バー状態開」，および「列車接近状態 OFF → ON」がそのままライフラインのエンティティにそれぞれマッピングされる．

3. 操作シーケンスの描画とメッセージのマッピング

- 操作モデルの操作手続きが，メインのソフトウェア機能からサブソフトウェア構成機能へのメッセージにマッピングされる．
- イベントが，メインソフトウェアのライフラインに対する入出力シーケンスへのメッセージにマッピングされる．

シナリオは，次のようになる．アクター「: 列車接近センサー」からメインのソフトウェア構成機能「列車接近管理」にイベント「接近列車検出」をマッピングしたメッセージを送り操作の起動を要求する．メインの機能「列車接近管理」はその要求を受けてサブのソフトウェア構成機能である「列車検出状態管理」に，操作「列車検出入力時その時に限り列車検出状態を ON にする」をマッピングしたメッセージを送って処理を起動する．サブの機能「列車検出状態管理」はそのメッセージに従った処理を実行しエンティティ「: 列車検出情報」を入力してエンティティ「: 列車検出状態 ON」を出力する．サブの機能「列車検出状態管理」のフローを受けて，メインの機能「列車接近管理」は次に続くサブの機能「列車接近状態管理」に操作「列車検出状態 ON で列車接近状態を OFF → ON にする」をマッピングしたメッセージを送り処理を起動する．サブの機能「列車接近状態管理」はそのメッセージに従った処理を実行しエンティティ「: 列車検出状態 ON」と「遮断バー状態開」を入力してエンティティ「: 列車接近状態 OFF → ON」を出力する．最後にメインの機能を介して最終結果をアクター「遮断バーコントローラ」に通知する．

6.9 操作モデル→ロバストネス図変換 (STEP9)

さらに，6.7 節で作成した操作モデル図 6.11 を，QVT 変換規則（図 5.11）に従って，モニタ不能駆動のロバストネス図に変換すると図 6.13 になる．

図 4.8 の変換テンプレートの対応に従って操作モデルは，ロバストネス図に置換えられ，続いて対応するモデル要素がマッピングされる．ロバストネス図に含まれるモデル要素は，

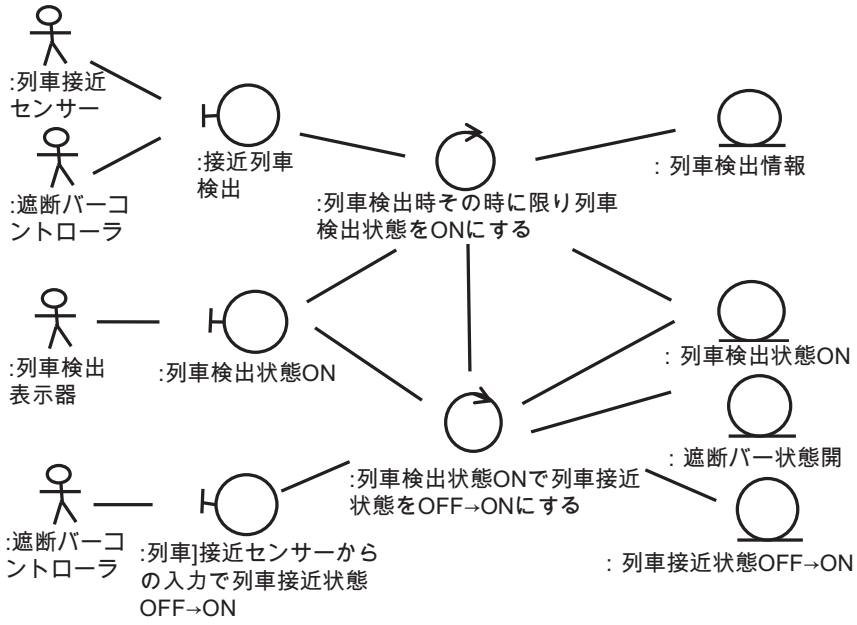


図 6.13: センサーによる接近検出ロバストネス図

バウンダリオブジェクト，エンティティオブジェト，コントローラ，およびアクターである．バウンダリオブジェクトはシステムと会話するアクターとの境界になるので，アクターに起因する開始イベント，システムによる操作の結果発生する結果イベント，およびある操作の結果発生し次の操作の原因となる中間イベントによってマッピングされる．また，エンティティオブジェクトは操作によって入出力されるエンティティによって，コントローラは処理や機能に相当するので操作によって，またアクターは環境エージェントによってそれぞれマッピングされる．

図 6.13 の表現内容は以下のようになる．アクター「列車接近センサー」はバウンダリ「接近列車検出」を介してコントローラ「列車検出入力時そのときに限り列車検出状態を ON にする」を駆動し，当該コントローラは，バウンダリ「列車検出状態 ON」を介して，その結果をエンティティ「：列車検出状態 ON」として出力したことを通知し列車検出を表示している．続けて，このバウンダリ「列車検出状態 ON」を介してコントローラ「列車検出状態 ON で列車接近状態 OFF → ON にする」が駆動され，バウンダリ「列車接近センサーからの入力で列車接近状態 OFF → ON」を介して，その結果をエンティティ「：列車接近状態 OFF → ON」として出力されたことが通知されている．これらのバウンダリは

外部世界との接点になるので，操作モデルにおけるイベントがマッピングされている．また，バウンダリを介して起動されるコントローラは，同様に操作がマッピングされている．コントローラと入出力で関連付けられているエンティティは，同様に操作モデルにおける入出力エンティティがそれぞれマッピングされたものである．

第7章　適用実験による評価

7.1　米国 ATM システムのユースケースモデリングの評価

7.1.1　実験内容

提案アプローチを適用することにより，ゴールモデルを半自動的にユースケースモデルに変換できる．ユースケースモデリングの初心者を被験者として提案アプローチの適用実験を実施した．実験により，ゴールモデルの洗練パターンによる要求定義シナリオはモデル変換において逐次継承されているか（洗練パターンの継承），またモデル変換において属人性は排除されているか（属人性の排除）を評価した．

実験では，Bjork[2] が米国 Gordon College のサイトに公開している ATM System に関する教育用題材を事例とした．銀行側のシステムは含まれない．要求記述とそれを分析したユースケース図，および個々のユースケースのイベントフロー図またはコラボレーション図，さらに詳細設計から実装までのオブジェクト指向ソフトウェア開発の全工程に渡りその成果物が提供されている．また，学内に限らず教育用に利用することを広く許諾されており，利用者からのフィードバックも反映される仕組みとなっている．このサイトに提供されているこれら成果物のうち，ATM システムのユースケース図とシステムスタートアップ，システムシャットダウン，およびセッション部分のイベントフロー図を基準のユースケースモデル（以降，基準モデルと呼ぶ）とした．

これら基準モデルは Gordon College 内外に広く提供され，利用者からのフィードバッグが反映されていることから，利用者に依存する内容が平均化され一般化された内容になっているものと想定できる．利用者に依存する内容としては，間違ってモデル化された内容のみならず，例外的に創意工夫に優れた内容もあり得るが，これらはすべて基準モデルから外れたものとまず判断する．このように，適用結果のユースケースモデルと基準モデルとの差異は属人性に由来するとし，その差異がモデル変換に起因していないことを検証す

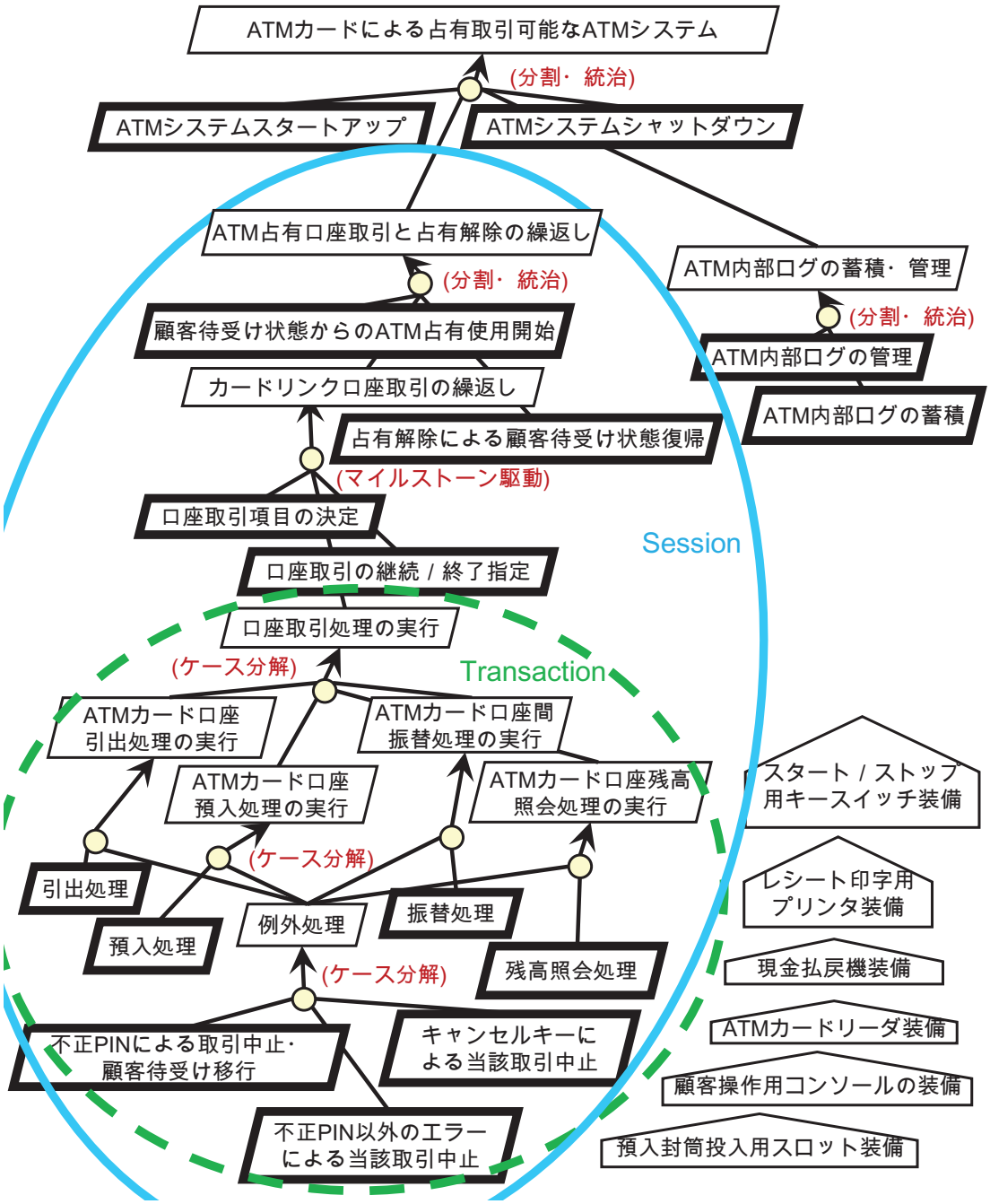


図 7.1: ATM カードによる占有取引可能な ATM システムゴールモデル（一部省略）

ることによって，モデル変換における属人性の排除を評価した．さらに，基準モデルから外れているものの創意工夫が見られ，一般的なモデルとは異なった選択肢による実現手法と評価できる内容についても，採用に値するモデルとして考察を加えた．

実験手順としては,「ATM カードによる占有取引可能な ATM システム」をトップゴールとするゴールモデル（図 7.1）を要求記述書に基づき準備し，被験者はそれに対して提案アプローチを適用することで，ユースケースモデルに変換した．すなわち，被験者は，要求記述書を参照しながら，STEP2，7 の一部と STEP4，6 を手動で実施し，その他の自動化可能な部分はアルゴリズムに従って変換した．

なお，複雑化を避けるため，ATM 内部ログ関連の機能は対象範囲外とした．また例外処理の分解は省略し，これをリーフゴールとして扱った．図 7.1 については，オブジェクトや操作の関連付けは図の複雑化を防ぐため省略している．

被験者は情報工学系の大学院学生 2 名（A，B）であるが，要求分析および KAOS ゴールモデルについては未経験であり，ユースケースモデリングについては講義による知識を有している程度である．また，ATM システムについては，日本国内 ATM の一般ユーザとして有する知識のみである．

7.1.2 評価の方法と判断基準

洗練パターンの継承評価については，ゴールモデル図 7.1 を構成する洗練パターンによるシナリオがユースケース図への各変換においてそれぞれ継承されているか，および STEP6 で要求ゴールを分解した洗練パターンによるシナリオがイベントフロー図への各変換においてそれぞれ継承されているかを評価した．この場合の判断基準は，変換規則を遵守した変換であることと変換結果が洗練パターンの意味を正しく表現できているかどうかである．

モデル変換における属人性の排除評価については，基準モデルに対する変換結果モデルの適合率と再現率を算出し，非適合・非再現内容の発生原因と属人性との関係について評価した．適合率と再現率は，ユースケース図またはイベントフロー図を主要なモデル要素に分け，それごとに適合するか否かを判定し適合数と再現数をカウントして計算した．ユースケース図のモデル要素は，ユースケースと，ひとつのアクターとユースケースの組のふたつであり，イベントフロー図のモデル要素はイベントごとの処理とした．これらのモデ

ル要素はモデルそれぞれの特徴を直接的に表現するものである．

被験者モデルのモデル要素が，基準モデルのモデル要素に対して，表記が異なっている場合であっても同等な意味，役割，もしくは振舞いを意図している場合には適合しているとし，被験者のモデル要素の中に適合しているものがある場合は再現しているとした．また要求の定義であるため，ユースケース等の内容で適合しているか否かを判断し，セッション/トランザクションどちらの制御機能に含まれるか等の機能割当ては判断基準とはしなかった．後述する “表 7.2 ATM システムユースケース図作成結果の比較” と “表 7.4 イベントフロー図作成結果の比較” では，‘正’ の欄に適合を ‘○’，非適合を ‘× ’ で示している．さらに本稿では，基準モデルのモデル要素としては適合していないが，ATM システムのモデル要素としては有効であり適合していると見做せるものを ‘△’ で示した．

基準モデルに対する被験者 A モデルの適合率は “被験者 A モデルのモデル要素のうちで適合している要素の総数÷被験者 A モデルのモデル要素の総数” としている．また再現率は “基準モデルのモデル要素のうち被験者 A モデルのモデル要素によって適合された要素の総数÷基準モデルのモデル要素の総数 ” としている．

この場合の適合か否かの判断基準は次のように定めた．ユースケースは ATM システムとのインタラクション要求仕様を規定するので，そのグループ分けが ATM システムの要求機能上大きな差とならない場合は，どちらのグループであってもインタラクション自体が適合していれば適合とする．ユースケースとアクターについては，表記が異なっていても同じ意味・内容を意図しているのであれば適合とした．イベントフローは，ユースケースにおけるイベントの流れを振舞いに対する要求として規定するものである．従って，イベントの内容やイベントの順番は重要であるが，各イベントの実現手順は，設計時に最終決定されるため，厳密に規定する必要はない．この理由により，イベントについては，アクターとソフトウエア機能によるイベントの処理内容が基準モデルと同じ意図を表現しているものであれば適合とした．また，アクターやソフトウエア機能の表記が異なっていても，そのイベントに対して同じ意味を持っていたり複数の意味を包含している場合は適合とした．例えば,「スタートアップ実行」イベントにおいて，アクターの「OperatorPanel」,「オペレータ，キースイッチ」,「オペレータ（キースイッチを包含していると見做せる）」それぞれは，このイベントに対して同じ意味を持っている．また，ソフトウエア機能の「ATM」

表 7.1: 事例における洗練パターンの種類と数

洗練パターン ＼ 変換STEP	STEP2, 7	STEP3	STEP8
マイルストーン駆動	15	1	14
ケース分解	6	5	1
ガード条件導入	2	0	2
分割・統治	2	2	0
モニタ不能駆動	4	0	4
制御不能駆動	0	0	0
パターン延数	29	8	21

STEP2,7
ゴールモデル→操作モデル
STEP3
操作モデル→ユースケース図
STEP8
操作モデル→イベントフロー図

と「スタートアップ」もこのイベントに対して同じ意味を持つ．この結果，被験者 A，B によるスタートアップ実行は基準モデルに対して適合していると判断できる．

7.1.3 評価結果

○ 洗練パターンの継承評価

事例における各変換で扱っている洗練パターンの種類と数を表 7.1 に示す．これに基づき変換した．以下，各モデル変換のステップごとに，洗練パターンの継承評価について説明する．

まず，ユースケース図への一連の変換ステップについて検証する．ゴールモデル（図 7.1）における対象範囲としてログ関係と例外処理の分解を対象外としたので，図 7.1 において，変換対象とするゴールモデルは分割・統治（2），マイルストーン駆動（1），およびケース分解（5）の洗練パターンで構成されている．ここで，“（ ）” 内は使用数を示す．

STEP2：上記洗練パターンの AND グラフは一段ずつ，ゴールモデルから操作モデルへの QVT による変換規則（図 5.4）に従い操作モデルに変換された．分割・統治，マイルストーン駆動，ケース分解のゴール洗練パターンによる AND グラフは，それぞれの洗練パターンを継承した操作モデルのパターンに正しく変換されており，規則に従った変換となっている．例えば，マイルストーン駆動洗練パターンの場合の変換結果は,「口座取引処理を選択する」→「口座取引処理を実行する」→「口座取引継続を指定する」の順番で操作するように変換されており，ゴールモデルのマイルストーン駆動洗練パターンの意味（シナ

リオ）を正しく継承している．操作モデルに変換後，被験者 A，B それぞれによって，洗練パターンのシナリオに合わせたイベント名称と環境エージェント名称が手動で記入された．この部分は手動のため，被験者 A と B で多少の差が出ている．

STEP3：STEP2 の出力として一次の AND グラフごとに得られた操作モデルは，STEP3 での操作モデルからユースケース図への QVT による変換規則（図 5.6）に従い，ユースケース図に変換された．それぞれ分割・統治，マイルストーン駆動，ケース分解の洗練パターンを継承したパターンに正しく変換されている．例えば，マイルストーン駆動洗練パターンの場合の変換結果では，<<precedes>> を使って，操作手順と同じく「口座取引処理を選択する」→「口座取引処理を実行する」→「口座取引継続を指定する」の順でユースケースが正しく駆動され，それらユースケースに関連するアクターにはそれぞれ環境エージェントが正しくマッピングされている．

STEP5：一次の AND グラフごとに変換されたそれぞれのユースケース図がひとつに統合された．被験者 A，B の統合結果ユースケース図はともに，最下層のユースケース（要求ゴールに相当する）がケース分解，マイルストーン駆動，および分割・統治に相当するパターンの組合せで関連付けられており，正常に統合されていることを確認できた．

ここまでが，ユースケース図への変換ステップについての検証結果である．つづいて，イベントフロー図への一連の変換ステップについて検証する．

STEP6：被験者 A，B は洗練パターンを使って要求ゴールを分解した．被験者 A がその分解に使用した洗練パターンは，マイルストーン駆動（8），ケース分解（1），モニタ不能駆動（4）の 3 種類であり，被験者 B が使用した洗練パターンはマイルストーン駆動（6）とガード条件導入（2）である．ここで，“（　）” 内は使用数を示す．

STEP7：ゴールモデルから操作モデルへの変換は，STEP2 と同様に，QVT による変換規則（図 5.4）に従って正常に変換された．例えば，被験者 A は，キー操作スイッチ ON 切換のモニタを外部に委ねるモニタ不能駆動洗練パターンを使って，「ATM システムスタートアップ」を分解している．被験者 A のモニタ不能駆動洗練パターン変換例では，キー操作スイッチ ON 情報を受取った後，ATM を起動する操作シナリオになっている．また同様に，被験者 B は，ATM がストップ状態のときに，キー操作スイッチ ON が発生したら ATM を起動するガード条件導入洗練パターンを使って同じ「ATM システムスタートアッ

プ」を分解している．このように，被験者 A，B の操作モデルへの変換結果は，ともに洗練パターンのシナリオを継承していることを確認した．

STEP8：一次の AND グラフごとに変換された操作モデルはそれぞれイベントフロー図に変換された．イベントフロー図への QVT による変換規則（図 5.9）に従った正常な変換である．例えば，STEP7 の例で説明すると，被験者 A の変換されたイベントフロー図は，キー操作スイッチからの ON 情報を取り込むイベントに続けて，ATM を起動するイベントを処理している．被験者 B モデルの場合は，ATM ストップ状態の時に，キー操作スイッチが ON に切り換ったイベントに続けて ATM を起動するイベントを処理する流れとなっている．このように，被験者 A，B の操作モデルからの変換結果はともに，洗練パターンのシナリオを継承したイベントフロー図であることを確認できた．同じ要求ゴールの分解について，被験者 A，B は異なる洗練パターンを適用しているが，その結果，適用された洗練パターンによるシナリオが継承されたイベントフロー図にそれぞれ変換されている．

例題の妥当性：本事例において各変換で扱っている洗練パターンの種類と数は表 7.1 のようになっているが，各変換にすべての洗練パターンが出てくるような事例を見つけるのは難しい．また，STEP7 と 8 でどの洗練パターンが扱われるかは，被験者が STEP6 でどの洗練パターンを使って分解したかに依存する．このため，STEP7，8 で洗練パターンを網羅できる保証はない．こうしたことから，適用評価の事例で全洗練パターンを網羅するのは困難である．

しかし，各変換は洗練パターンの種類によらず，それぞれひとつの変換規則に従っている．すなわち，STEP2，7 では図 5.4，STEP3 では図 5.6，および STEP8 では図 5.9 の QVT による変換規則に従ってそれぞれ変換されている．それぞれの変換規則は各洗練パターンを抽象化したメタモデルの領域で規定されているので，洗練パターンによる違いは変換規則の基本的な部分には影響しない．STEP8 のイベントフロー図への変換規則（図 5.9）では，モデル要素のマッピング方法が洗練パターンによって一部異なっている．だが，この違いは変換規則にとって本質的なものではなく基本的な部分に変わりはない．このことから，提案したそれぞれの QVT による変換規則を評価するのに，すべての洗練パターンを網羅することは必須ではないと考えられる．表 7.1 に示す通り，STEP2，7，STEP3，STEP8 で確認対象となっている洗練パターンの延数は，それぞれ 29，8，21 となっており量的に

表 7.2: ATM システムユースケース図作成結果の比較

サイト提供教材(基準モデル)			被験者Aモデル					被験者Bモデル				
ユースケース		アクター	ユースケース		正	アクター	正	ユースケース		正	アクター	正
System Startup		Operator	システムスタートアップ		○	顧客, 銀行, オペレータ	①	システムスタートアップ		○	顧客, 銀行, オペレータ	①
System Shutdown		Operator	システムシャットダウン		○	同上	①	システムシャットダウン		○	同上	①
Session		Customer	占有使用開始		○	同上	①	占有使用開始		○	顧客, 銀行	①
			占有使用解除		○	同上	①	占有使用解除		○	同上	①
(transaction type選択) (another transaction指定)		Customer, Bank	セッション	取引処理選択	○	顧客, 銀行	②	セッション	取引処理選択	○	同上	②
				取引継続指定	○	顧客, 銀行, オペレータ	②		取引継続指定	○	同上	②
Transaction	Withdrawal		トランザクション(口座取引処理実行)	引出処理	○	顧客, 銀行	②	トランザクション(口座取引処理実行)	引出処理	○	同上	②
	Deposit			預入れ処理	○	同上	②		預入れ処理	○	同上	②
	Transfer			振替処理	○	同上	②		振替処理	○	同上	②
	Inquiry			残高照会処理	○	同上	②		残高照会処理	○	同上	②
	Invalid PIN			例外処理	○	顧客, 銀行, オペレータ	②		例外処理	○	同上	②

注. ○:正解, ×:不正解, ○で囲った数字:正解の組数(例. 被験者Aのシステムスタートアップとアクターの組は全部で3組あり, 正解が1組, 不正解が2組である.)

も十分である.

以上述べたことを前提にしたとしても，さらに補足的な評価として洗練パターンの変換を網羅的に確認したい場合は，類似のパターンで代替的に確認し結果を推定することが可能である．すなわち，STEP2，7 の制御不能駆動は類似のパターンであるモニタ不能駆動で代替確認でき．STEP3 では，同様に，ガード条件導入とモニタ不能駆動および制御不能駆動はマイルストーン駆動で代替確認できる．また，STEP8 では，分割・統治はマイルストーン駆動で，また制御不能駆動はモニタ不能駆動で代替確認できる．

これまで述べたように，提案している QVT 変換規則によるゴールモデルからユースケースモデルへの変換の評価はこの適用事例で十分であると考える．

○ 属人性の排除評価

ユースケース図の評価： 表 7.2 は，ユースケース図のユースケースとアクターについて，基準モデルと被験者 A，B モデルの変換結果を比較したものである．関連するユースケースとアクターは同じ行に並べて表示している．また，‘正’ の欄に基準モデルに対してそのモデル要素が適合か否かを ‘○’ と ‘× ’ で示した．アクターの右側の ‘正’ の欄は，ひとつのアクターとユースケースの組が適合するか否かを示しており，‘○’ で囲った数字は適合している組の数である．表 7.2 に基づき計算した適合率と再現率を表 7.3 にまとめた．ユースケース（UC）については，被験者 A，B モデルともに，適合率，再現率は 100%である．一方，アクターとユースケースの組（[Actor，UC]）は，再現率はともに 100%だが，適合率については被験者 A モデルの 64.3%に対し被験者 B モデルは 75.0%と差が出ている．

UC についての一見不適合に見える箇所は，被験者 A，B モデルともに同じで,「取引処理

表 7.3: ATM システムユースケース図要素の適合，再現数

基準モデルに対して	基準モデル UC	基準モデル [Actor, UC]	被験者Aモデル UC	被験者Aモデル [Actor, UC]	被験者Bモデル UC	被験者Bモデル [Actor, UC]
適合数			11	18	11	18
非適合数			0	10	0	6
抽出要素計	8	13	11	28	11	24
適合率(%)			100	64.3	100	75.0
再現数			8	13	8	13
非再現数			0	0	0	0
基準モデル要素計	8	13	8	13	8	13
再現率(%)			100	100	100	100

注1. UC：ユースケース，[Actor, UC]：関連するActorとUCの組
注2. Transaction（口座取引処理実行）のUCは，Withdrawal（引出処理）等の各処理に分割した.

選択」と「取引継続指定」がセッションの機能として規定されているところである．基準モデルでは，ユースケースとして明示されていないが，セッションとトランザクションの境界に位置する機能として説明されトランザクションに割当てられている．被験者 A，B がこのふたつをセッションのユースケースとしたのは，入力であるゴールモデル（図 7.1）の段階でセッションの機能として割当てられていたからであり，属人性の影響ではない．また，このふたつの機能がセッションとトランザクションのどちらに割当てられても，システムの振舞い要求定義への影響はないと思われる．従って，このふたつは適合となり適合率は 100%となる． [Actor，UC] の適合率について検証する．被験者 A，B モデルとも，再現率が 100%であることから，アクターの冗長的な抽出が適合率悪化の原因であると考えられる．STEP2 の最後に環境エージェントの名称を手動で入力し，それがアクターにマッピングされる．このことから，この環境エージェント名称の手動入力が適合率悪化の根本的原因であり，唯一属人性が混入するところである．モデル変換における属人性の混入はないと言える．環境エージェント名称の付与規則を細かく規定することによって属人性の混入を抑制できると考えられる.（今後の研究課題）

イベントフロー図の評価： 表 7.4 は，イベントフロー図のイベント処理について，基準モデルと被験者 A，B の変換結果を比較したものである．‘正’ の欄に，基準モデルに対して被験者 A または B のイベント処理が適合か否かを ‘○’‘× ’ および ‘△’ で示した．‘△’ は，基準モデルには適合していないが，システム的には有効なイベント処理を示す．基準モデ

ルのイベント3および16に記された×は，内部処理として記述されているため，イベント処理としては除外したことを示している．それに対して被験者A，Bは，銀行またはオペレータ間のイベント処理として新たに抽出している．表7.4に基づき計算した適合率と再現率を表7.5にまとめた．‘△’は不適合として計算している．被験者A，Bモデルの適合率はそれぞれ62.5%，78.6%であり，再現率はそれぞれ76.9%，84.6%である．

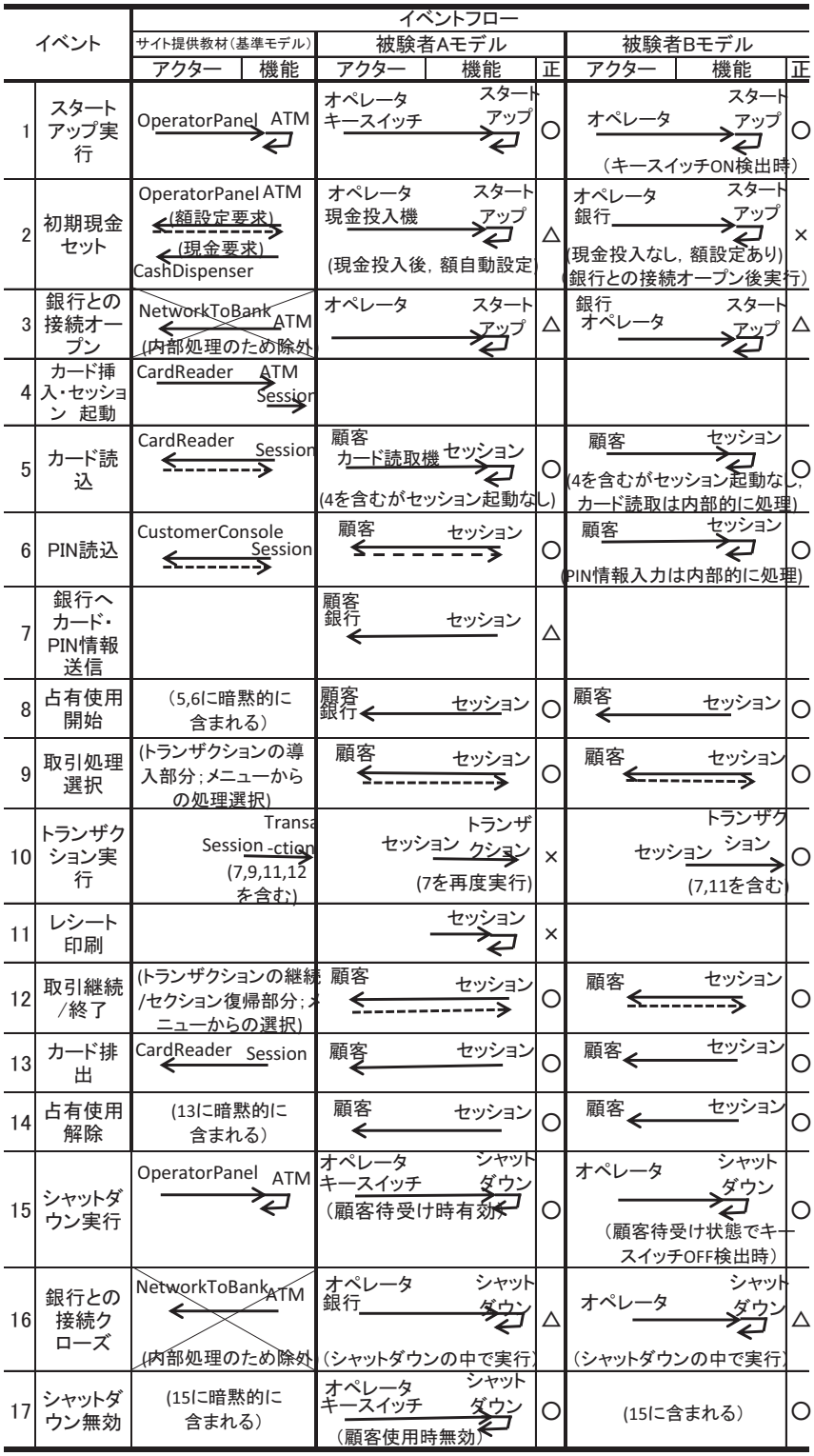
被験者Aモデルの不適合等の内容を検証する．イベント2（△）では，基準モデルは手動で現金額を設定しているのに対し，被験者Aモデルは現金投入後自動で設定している．イベント3，16（△）は，基準モデルではイベント処理としては扱われていないものである．被験者Aモデルは，イベント7をトランザクションの中と重複して実行している．カード・PIN情報を入力時点で銀行に送信することによって，いろいろなチェックに使用できるという拡張性を，要求ゴール分解作業の中で考慮したということである．ゴール分解に被験者の考えが反映されたもので，ゴールモデリングにおける個人差の許容範囲内と思われる．逆に，モデル中にイベントを明示できるという操作モデルの特徴を活用している．このように，これら‘△’は基準モデルと厳密には異なるが，システム的には有効であると言える．イベント10，11（×）は，レシート印刷をトランザクションの中ではなくセッションで印刷しているので不適合とした．また，イベント4のセッション起動は再現されていない．

次に，被験者Bモデルの不適合等の内容を検証する．イベント2（×）では現金投入が記述されていない．イベント3，16（△）の不適合，およびイベント4の不再現については被験者Aモデルの場合と同様である．

上述した被験者A，Bモデルの基準モデルに対する不適合，不再現内容の発生原因を調べてみると，STEP6での要求ゴールの分解結果やSTEP7でのイベントや環境エージェントの名称付与結果に原因がみられる．このことから，STEP7，8のモデル変換における属人性の影響は排除されていることが確認された．STEP6と，STEP7（STEP2と同じ作業）におけるイベントや環境エージェントの名称付与は，ゴールモデリングに対する熟練度やATMシステムのドメイン知識など属人的な要素に大きく影響される部分である．

しかし，これらの部分は逆にゴール指向モデリングの有効な特徴を発揮できる部分でもあり，‘△’のイベント抽出にその効果が表れている．ここで，‘△’を適合と見做すと，被験者A，Bモデルの適合率はそれぞれ87.5%，92.9%，再現率は84.6%，84.6%となって，被験

表 7.4: ATM システムイベントフロー図作成結果の比較



注1. 複雑化を避けるため，細部を省略した概略フローを示している.

注2. 機能：ソフトウエア機能，○：適合，×：不適合，△：不適合だがシステム的には適合

表 7.5: ATM システムイベントフロー要素の適合，再現数

基準モデルに対して	イベント処理		
	基準モデル	被験者Aモデル	被験者Bモデル
適合数		10	11
不適合数		6	3
イベント処理総数	13	16	14
適合率(%)		62.5	78.6
再現数		10	11
不再現数		3	2
再現対象数	13	13	13
再現率(%)		76.9	84.6

注. △は不適合としてカウントした.

者 B モデルの再現率は変化しないが，それ以外は大幅に改善する．このことから，STEP2（STEP7）と同様に，STEP6 のゴール分解についても，洗練パターンのシナリオに沿った分解規則を細かく規定することによって，属人性をさらに抑制しつつゴール指向分析の有効性を効果的に追及できると考えられる.（今後の研究課題）

上述したユースケース図，イベントフロー図の評価結果から，STEP2，3，7，8 のモデル変換に起因する属人性は排除されていることが確認された．手動作業部分において混入する属人性については，STEP4 もしくは STEP5，または STEP8 終了後の部分的な見直しと修正で対処できると考える.

妥当性の脅威

何を正解モデルとするかが妥当性の脅威となる．実務面でのゴール指向要求分析手法の活用実績は少なく，提案する一連のモデル変換への入力と出力の正解モデルを決定するのは難しい．本稿では，要求記述書から入力となるゴールモデルを作成し，その変換結果を既存のオブジェクト指向設計によるユースケースモデル（基準）と比較し評価した．さらに，その差を属人性の影響として評価した．この入力と出力の正解モデルの内容は，各モデル変換の妥当性評価への影響は少ないと思われるが，最終変換結果であるユースケース図やイベントフロー図の評価には影響を与える．本稿では，その影響をできるだけ緩和できるように判定条件を考慮した.

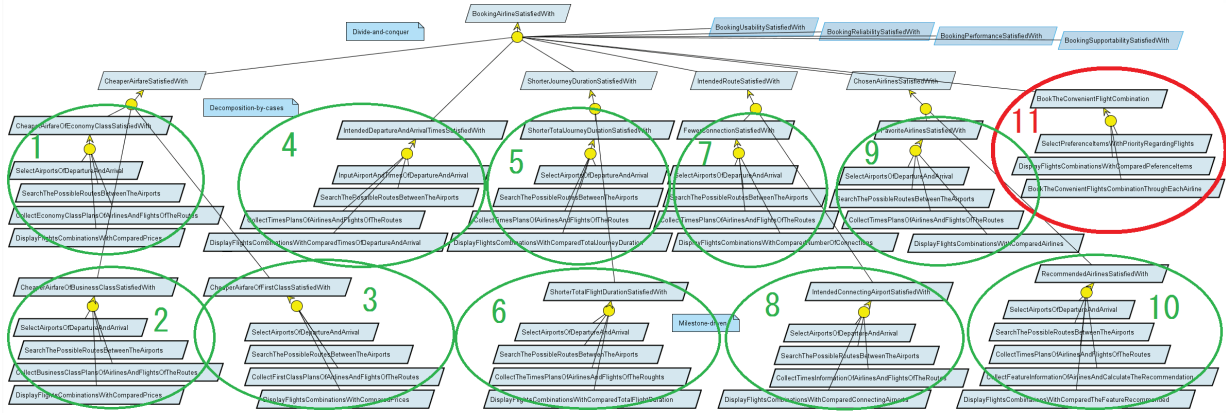


図 7.2: [BookingAirlineSatisfiedWith] Goal model

7.2 国際航空券予約システムのロバストネス分析の評価

7.2.1 実験内容

国際航空券予約システムを事例として提案アプローチを適用し，ゴールモデルをロバストネス図に変換した．その結果において，ゴールモデルとロバストネス図のモデル要素を比較し，その対応を評価した．

まず，国際航空券予約システムのゴールモデルについて説明する．航空券を予約するときには，様々な優先度が存在する．例えば，好きな航空会社，時間優先の旅行，決められた到着時刻などである．航空券予約システムは，このような様々な要望に対応したサービスを提供する必要がある．図 7.2 は,「満足できる航空券予約」をトップゴールとして，国際航空券予約システムに対する要求を分析し定義したゴールモデルである．ゴールモデル全体を表示しているため細部の判別が困難になっているが，まずは全体構成の形を見てほしい．実験内容にかかわる変換の説明は，赤い円 11 で囲まれた AND グラフを例にとって行う．

図 7.2 では複雑な図となるため省略しているが，ゴール実現に責任を持つエージェントやゴールが関心を持つエンティティを含めたゴールモデルとなっている．このゴールモデルは三階層である．最下層の親ゴールが要求ゴールであり，それぞれ実現責任を持つ単独のエージェントが関連付けられる．最下層のサブゴールとそれらの関連は，要求ゴールをどのように実現するかをイベントに対応した振舞いシナリオで示している．図 7.2 の右側に

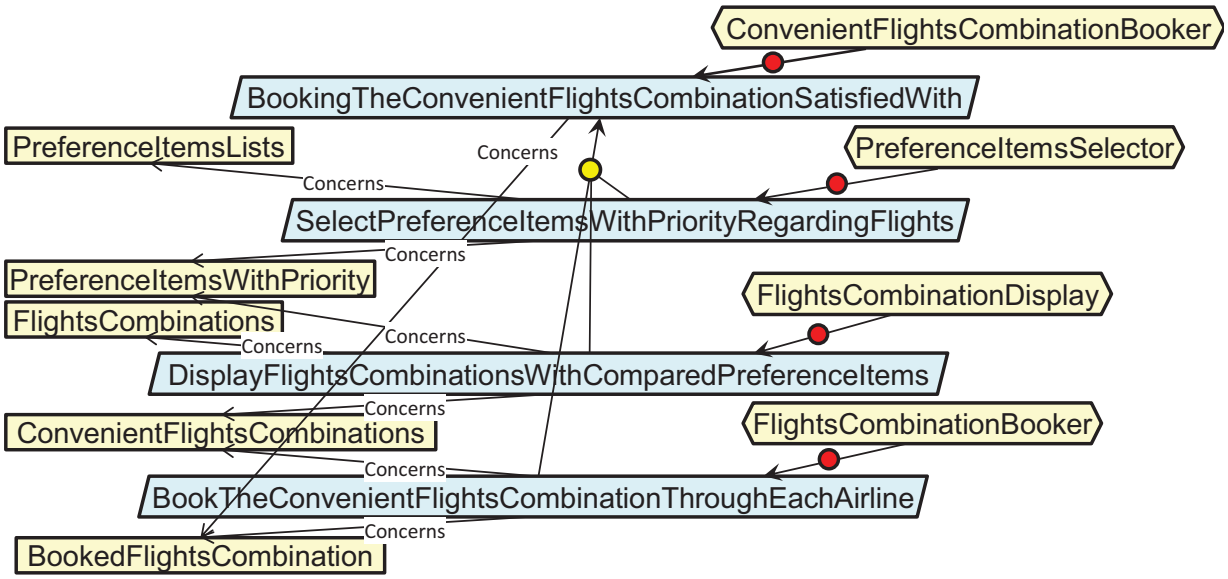


図 7.3: [BookTheConvenientFlightsCombination] Goal model

ある赤い円 11 で囲っている箇所は，要求ゴール「満足な便利な航空券組合せの予約」をマイルストーン駆動の洗練パターンで洗練化した部分である．この部分を拡大すると，図 7.3 になる．各ゴールに関連するソフトウェアエージェントとエンティティも明示してある．

この部分を例にとり，ゴールモデルからロバストネス図への変換内容を説明する．ゴールモデルの詳細は以下のようになる．まず，サブゴール「航空券の優先項目と優先度の選択」を実現し，次にサブゴール「航空券組合せとその優先項目比較の表示」を実現した後，最後にサブゴール「航空会社を通した便利な航空券組合せの予約」を実現することで，親ゴールの実現を果たす．サブゴール「航空券の優先項目と優先度の選択」は，エンティティ「優先項目リスト」と「優先項目と優先度」をそれぞれ入力と出力とし，サブゴール「航空券組合せとその優先項目比較の表示」は，エンティティ「優先項目と優先度」および「航空券組合せ」と「便利な航空券組合せ」をそれぞれ入力と出力とする．またサブゴール「航空会社を通した便利な航空券組合せの予約」は,「便利な航空券組合せ」と「予約された航空券組合せ」をそれぞれ入力と出力とする．それぞれのサブゴール実現責任は，実現の順番に,「優先項目セレクター」,「航空券組合せ表示」,「航空券組合せブッカー」のそれぞれのソフトウェアエージェントが負っている．親の要求ゴール「便利な航空券組合せの予約」は，エンティティ「航空券組合せ」および「優先項目リスト」と「予約された航空券組合せ」をそれぞれ入力と出力とし，ソフトウェアエージェント「便利な航空券組合せブッ

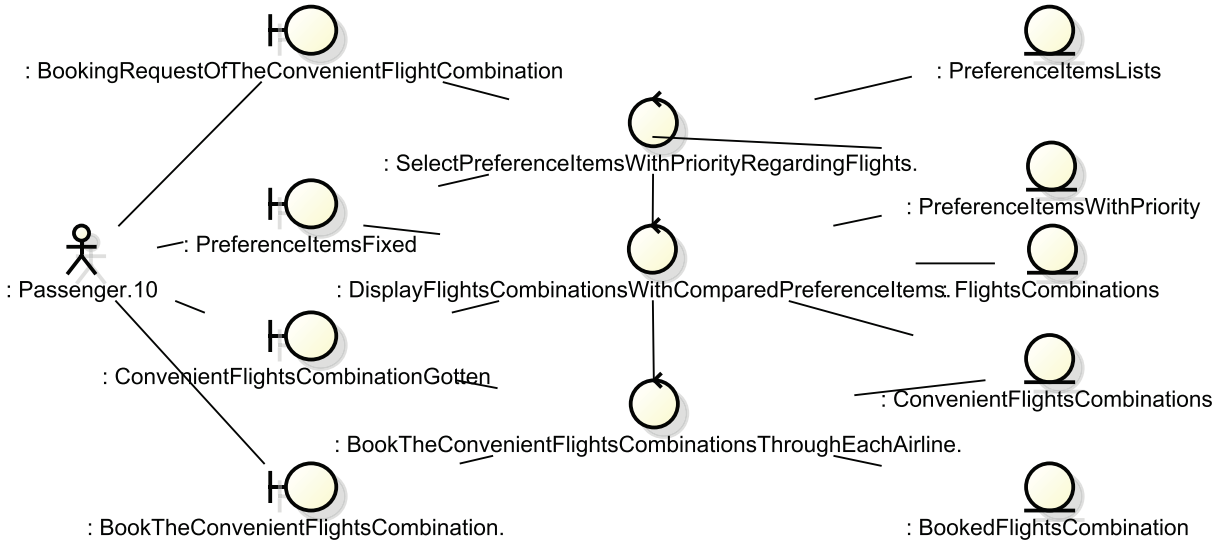


図 7.4: [BookTheConvenientFlightsCombination] Robustness diagram

カー」が実現責任を負っている．

ゴールモデルからロバストネス図へは以下の手順で変換した．図 7.3 のゴールモデルを，STEP7（操作モデルへの QVT 変換規則図 5.4 による），STEP9（ロバストネス図への QVT 変換規則図 5.11 による）に従って，ロバストネス図（図 7.4）に変換した．

変換結果ロバストネス図（図 7.4）の内容を説明する．アクターは旅行者である．三個のコントローラが中心になる．それぞれのコントローラは処理要求と処理結果出力に相当するバウンダリオブジェクトと関連付けられ，さらに入力エンティティと出力エンティティと関連付けられている．一番上のコントローラ「航空券の優先項目と優先度を選択する」はバウンダリオブジェクト「便利な航空券組合せの予約要求」を介して処理要求され，エンティティ「優先項目リスト」を入力して処理結果であるエンティティ「優先度付き優先項目」を出力する．その結果は，バウンダリオブジェクト「優先項目決定」を介してアクターに通知される．さらに，そのバウンダリオブジェクトを介して，次の処理をコントローラ「優先項目比較付の航空券組合せを表示する」に要求する．このコントローラは，エンティティ「優先度付き優先項目」と「航空券組合せ」を入力し，エンティティ「便利な航空券組合せ」を出力する．その結果は，バウンダリオブジェクト「得られた便利な航空券組合せ」を介してアクターに通知される．さらに，そのバウンダリオブジェクトを介して，次の処理をコントローラ「便利な航空券組合せを各航空会社を通して予約する」に要求す

る．この最後のコントローラは，エンティティ「便利な航空券組合せ」を入力し，エンティティ「予約された航空券組合せ」を出力する．その結果は，バウンダリオブジェクト「便利な航空券組合せを予約する」を介してアクターに通知され，一連の処理は終了する．このように，このロバストネス図による処理はマイルストーン駆動により流れているのが理解できる．

ここまでで，図 7.2 の赤円 11 の要求ゴールの洗練化部分を例にとって，ゴールモデルからロバストネス図への変換手順について説明したが，緑の円で囲った 1 〜10 の要求ゴールを洗練した部分についても同様に変換した．緑円 1 〜10 も，赤円 11 と同じくマイルストーン駆動洗練パターンによって洗練化されている．また，それらの変換結果であるそれぞれのロバストネス図も，図 7.4 と同様に，マイルストーン駆動による処理の流れになっていた．

7.2.2　評価の方法と判断基準

提案アプローチは，KAOS による要求定義モデルの洗練パターンによる振舞いの情報を変換テンプレートを介して変換していくことにより，ロバストネス図に継承することを目的のひとつとしている．ここでは，この洗練パターンの継承について評価した．評価の方法として，要求ゴールを洗練化したゴールモデル（図 7.2 の円 1〜11）とそれを変換したロバストネス図とを，処理手続の形態（洗練パターンまたは変換テンプレート）や対応するモデル要素についてそれぞれ比較した．対応するモデル要素は,「ゴールモデルのモデル要素-ロバストネス図のモデル要素」で示すと，サブゴール-コントローラ，エンティティ-エンティティオブジェクト，イベント-バウンダリオブジェクト，環境エージェント-アクターとなる．なお，環境エージェントとイベントは操作モデルのモデル要素である．

判断基準として，処理手続の形態やモデル要素の内容，数が同等であれば，正常に継承できていると判断した．内容は名称によって表現されているので，ここでは「名称（内容）」としてそのことを表わした．KAOS モデルとロバストネス図では，サブゴールとコントローラなどモデル要素の種類が異なるものがある．その場合は同じ名称にはならないが，同等な意味であれば継承していると判断した．

7.2.3 評価結果

図 7.2 の円 11 のゴールモデルを抜粋し，処理手続の形態および対応するモデル要素すべてについて，ゴールモデルとそれを変換したロバストネス図の比較結果を表 7.6 にまとめた．最上段の表がすべての要素の比較結果をまとめたものである．その中で，スペース不

表 7.6: KAOS モデルとロバストネス図の要素比較

比較項目	洗練パターン			サブゴール			エンティティ			イベント			環境エージェント		
	変換テンプレート			コントローラ			エンティティオブジェクト			バウンダリオブジェクト			アクター		
	手続数	パターン等価性		個数	名称等価性		個数	名称等価性		個数	名称等価性		個数	名称等価性	
Goal	3	MDRP	○	3	表a	○	5	表b	○	4	表c	○	1	旅行者	○
Rob.	3	MDP		3	表a		5	表b		4	表c		1	旅行者	

名称 (内容) 比較

表a	対応要素	1	2	3
Goal	サブゴール	航空券の優先項目と優先度の選択	航空券組合せとその優先項目比較の表示	航空会社を通した便利な航空券組合せの予約
Rob.	コントローラ	航空券の優先項目と優先度を選択する	優先項目比較付の航空券組合せを表示する	便利な航空券組合せを各航空会社を通して予約する

表b	対応要素	1	2	3	4	5
Goal	エンティティ	優先項目リスト	優先度付き優先項目	航空券組合せ	便利な航空券組合せ	予約された航空券組合せ
Rob.	エンティティオブジェクト	優先項目リスト	優先度付き優先項目	航空券組合せ	便利な航空券組合せ	予約された航空券組合せ

表c	対応要素	1	2	3	4
Goal	イベント	便利な航空券組合せの予約要求	優先項目決定	得られた便利な航空券組合せ	満足のいく便利な航空券組合せの予約
Rob.	バウンダリオブジェクト	便利な航空券組合せの予約要求	優先項目決定	得られた便利な航空券組合せ	満足のいく便利な航空券組合せの予約

注: Goal --- KAOS ゴールモデル，Rob.--- ロバストネス図，MDRP --- マイルストーン駆動洗練パターン，MDP --- マイルストーン駆動変換テンプレート

足のため記述できない名称（内容）の比較は，表 a，表 b，または表 c として「名称（内容）比較」の下側に別出しした．最上段の表では，比較項目として，比較しているゴールモデルとロバストネス図の要素（ex. ゴールモデルの洗練パターンとロバストネス図の変換テンプレート）と比較項目（ex. 手続数とパターン等価性）を示し，Goal と Rob. の行にそれぞれの比較項目について具体的に実験で得られたデータを示している．表 a には 3 個づつのサブゴールとコントローラについて，表 b には 5 個づつのエンティティとエンティティオブジェクトについて，また，表 c には 4 個づつのイベントとバウンダリオブジェクトについて，それぞれ具体的に得られた名称（内容）を比較して表記した．

この円 11 についての比較結果表 7.6 は，ゴールモデルとそれを変換したロバストネス図について，処理手続の形態およびすべてのモデル要素の比較結果が同等であることを示している．他の要求ゴールに関する円 1 ～10 のゴールモデルについても，円 11 と同様に，処理手続の形態およびすべてのモデル要素について同等であることを確認できた．この結果，ゴールモデルからロバストネス図へと洗練パターンによる情報が正常に継承されていると判断した．

なお，今回の国際航空券予約システムの事例で要求ゴールの洗練に使用した洗練パターンはマイルストーン駆動洗練パターンのみである．このマイルストーン駆動洗練パターンによる評価結果を，他の洗練パターンに適用できるかどうかについて述べる．第 5 章で説明しているゴールモデルから操作モデル（5.2 節），操作モデルからロバストネス図（5.6 節）への QVT による変換規則を見ると，ほとんどが洗練パターンによらない変換規則となっている．ただ，操作モデルからロバストネス図への変換規則において，ケース分解と分割・統治の洗練パターンについては，一部のバウンダリオブジェクトのパターンが特殊なものとなっている．しかし，米国 ATM システムのユースケースモデリングの評価に関する 7.1.3 節で議論した例題の妥当性 の場合と同様に，この部分の特殊性もロバストネス図への QVT 変換規則（図 5.11）にとって本質的なものではなく，本例題による評価で十分であると考える．

とは言え，このふたつの洗練パターンについて，厳密に評価したい場合は別途実施する必要がある．但し，この場合も，使用しているロバストネス図の変換テンプレート（図 4.8 のロバストネス図の列）からある程度予測することは可能である．ケース分解洗練パターンについては，最終結果のバウンダリオブジェクトに対し，それぞれのケースの結果のバウンダリオブジェクトが汎化／特化の関係になっている．これは，各ケースのサブゴール実現が親ゴールの実現に相当することとシナリオ的に合致する．また，分割・統治洗練パターンについては，最終結果のバウンダリオブジェクトに対し，それぞれの分割の結果のバウンダリオブジェクトがコンポジションの関係になっている．これは，各分割のサブゴールそれぞれがすべて実現されることが親ゴールの実現に相当することとシナリオ的に合致する．これらのことから，ケース分解洗練パターンと分割・統治洗練パターンについても，ゴールモデルからロバストネス図に洗練パターンによる情報が継承されると期待できる．

本例題によるロバストネス図への変換に関する評価では，属人性の排除評価は実施していない．今後の実施を検討する必要があるが，ロバストネス図への変換規則（図 5.11）とイベントフロー図への変換規則（図 5.9）の類似性から，イベントフロー図への変換における属人性の排除効果と同様に，ロバストネス図への変換についても属人性の排除効果が見込まれると容易に期待できる．

第8章　結論

本章では，本研究における課題，提案内容，評価内容についてまとめ，最後に，今後の課題について述べる．

8.1　本研究のまとめ

要求定義工程と設計工程との間には，要求定義情報の一部が抜け落ちてしまうギャップが存在すると言われている．本節では，そのギャップを橋渡しすることを目的とした時の課題，その課題を解決するための提案内容，および評価内容と結果についてまとめる．

8.1.1　課題・目的

要求定義工程，設計工程にはそれぞれゴール指向要求分析手法，UML によるオブジェクト指向設計手法という有効な手法が存在する．しかし，ゴール指向要求分析手法の成果物としての要求定義情報をオブジェクト指向設計手法に漏れなく体系的に反映する仕組みには，知識や経験則など設計者に依存しているなど課題が残っている．従って，論理的，体系的に抽出した要求定義を設計に反映し実装する仕組みを構築することは，ソフトウエア開発における基本的な課題のひとつであると考えられる．そして，この基本的な課題を解決するための要件は以下である．

1. 要求定義工程における体系的，論理的な要求の抽出・分析・定義
2. 設計・実装工程の開始点としての要求定義を，設計から実装へと体系的に反映し具体化していく仕組み
3. 要求定義工程の成果を，設計・実装工程への入力である要求定義に，体系的に反映する仕組み

本稿で提案するアプローチはこれらの要件を満たすためのものである．また要件 3．については，KAOS モデルをユースケースモデルやロバストネス図に体系的に変換するアプローチを提案する．

本研究の課題は，これら要件において，それぞれの工程で有効性が確認され実務にも使用されている既存の手法を最大限に活用し，体系的・論理的に抽出し定義した要求を漏れなく設計し実装する仕組みを構築することである．

その課題の解決に向けて，要件の 1．と 2．を満足させるために，要求定義工程，設計・実装工程それぞれに実績のあるゴール指向要求分析手法 KAOS，UML による設計・実装プロセス ICONIX を利用する．さらに，要件 3．を満たすために，KAOS による要求定義モデルを ICONIX プロセスへの入力であるユースケースモデルや予備設計モデルであるロバストネス図に体系的に変換する仕組みを構築することが，本研究の目的となる．

8.1.2　提案内容

KAOS の成果物である要求定義モデルを，ICONIX プロセスの入力モデル（ユースケースモデル）や予備設計モデル（ロバストネス図）に体系的に変換するアプローチを提案する．これによって，KAOS による要求定義モデルの情報を ICONIX プロセスに体系的に反映する．提案アプローチの概要は以下のとおりである．

1. KAOS のゴールモデル（要求定義モデル）を入力とし，操作モデルに変換する（責任モデル，オブジェクトモデルの要素を含む）．
2. 操作モデルをユースケース図に変換する．
3. ユースケースに相当する要求を示すゴールを，シングルイベントに対応するシナリオに洗練し，操作モデルに変換する．
4. シナリオの操作モデルをイベントフロー図に変換する．
5. シナリオの操作モデルをロバストネス図に変換する．

全体的には半自動的な変換であるが，モデル変換の部分は QVT による変換規則に従い規則的に実施できる．提案アプローチによる一連のモデル変換は，洗練パターンの意味（振舞いのシナリオ）を継承するとともに，属人性を排除する効果がある．

提案アプローチの特徴は次の 2 点である．

- KAOS モデリングと，ユースケースモデル（ユースケース図，イベントフロー図）やロバストネス図への変換は，変換テンプレート（洗練パターンの振舞いシナリオを継承する）を使って実施する．
- シングルイベントに着目して，ユースケースにおけるアクターとシステム間のインタラクションを抽出し，システム機能の振舞いを特定する．

この洗練パターンをベースに KAOS モデルを作成することによって，ユースケースモデルやロバストネス図への変換を容易にする．また，操作の原因や結果に対応するシングルイベントをキーとすることでインタラクション抽出の根拠となり，それに対応するシステム機能の振舞いを特定できる．

8.1.3 評価内容

米国 ATM システムのユースケースモデリングおよび国際航空券予約システムのロバストネス分析による提案アプローチの適用実験を実施した．

米国 ATM システムのユースケースモデリングにおける適用評価では，ユースケースモデリングの初心者を被験者として，洗練パターンによる要求定義シナリオはモデル変換において逐次継承されているか（洗練パターンの継承），またモデル変換において属人性は排除されているか（属人性の排除）を評価した．

具体的には，洗練パターンの継承評価については，ユースケース図，イベントフロー図それぞれへのモデル変換の各 STEP において，洗練パターンによるシナリオがそれぞれ継承されているかを評価した．判断基準は，変換規則を遵守した変換であることと変換結果が洗練パターンの意味を正しく表現できているかどうかである．属人性の排除評価については，基準モデルに対する変換結果モデルの適合率と再現率を算出し，非適合・非再現内容の発生原因と属人性との関係について評価した．

前者の評価結果として，STEP2，3，5，7，8 のそれぞれのモデル変換において，QVT による変換規則を遵守した変換であること（STEP5 は対象外）と，変換後のモデルが洗練パターンによる振舞いのシナリオを正しく表現していることを確認した．また，後者の評価結果としては，基準モデルと適用結果について，ユースケース図とイベントフロー図の非適合・非再現の発生原因は，人手による環境エージェントの名称付与や要求ゴールの洗練部分にあり，モデル変換部分には混入していないことを確認した．

国際航空券予約システムのロバストネス分析における適用評価では，ゴールモデルとそれに対する提案アプローチの適用結果であるロバストネス図のそれぞれのモデル要素を比較し，洗練パターンによるゴールモデルの情報がロバストネス図に継承されているかを評価した．

具体的には，11 個の要求ゴールそれぞれを洗練した AND グラフに提案アプローチを適用してロバストネス図に変換し，適用前後の AND グラフ（ゴールモデル）とロバストネス図それぞれの処理手続の形態とモデル要素の名称と数が同等であるかどうかを評価した．

評価結果としては，対応する処理手続の形態とモデル要素はすべて同等であることが確認された．この結果，ゴールモデルからロバストネス図への情報は正常に継承されていると判断できた．QVT による要素のマッピングにおいて，ケース分解と分割・統治の洗練パターンについては特殊な変換部分が存在するが，変換規則の基本的な部分ではなく，その変換テンプレートから判断すると，問題なく継承されると期待できる．

このロバストネス図への変換については属人性の排除に関する評価は実施していないが，ゴールモデルから操作モデルへの変換はユースケースモデリングの場合と同じ変換規則であり，操作モデルからロバストネス図への変換も操作モデルからイベントフロー図への変換規則と類似の規則であるため，同様に属人性も排除されていると期待できる．しかし，厳密には適用評価が必要となり，課題として残っている．

8.2 今後の課題

8.2.1 KAOS の利用

提案アプローチでは，ユースケース記述として，ユースケースの説明をイベントフロー図で表現する．しかし，ユースケースの説明は，自然言語のユースケース文によってなされる場合も多い．ユースケース文で表現する場合は，操作モデルの操作手順を文章に変換する必要があるが，5.2 節までの手順はそのまま使用できる．操作モデルからユースケース文への変換は，操作モデルの要素を操作手続に従って組み合わせて文章を構成することで可能と思われるが，詳細は今後の課題としたい．

KAOS モデリングから UML モデリングへのシームレスな移行の出発点として，操作モデルからユースケースモデル，ロバストネス図それぞれへの変換を選んだ．この選択は評価実験の結果からも正しいと思うが，KAOS モデリングと UML モデリングのシームレスな統合を実現するためには，さらに多様な移行関係も必要になる．UML による設計プロセスのひとつである ICONIX による設計を想定しているが，ユースケースモデリングとロバストネス図を介してクラス図やシーケンス図へと詳細化して行くなかで，これらの図へ KAOS モデルの持つ意味を反映していく手段が得られれば，より密接に KAOS と ICONIX を統合することができる．例えば，オブジェクトモデルを洗練し，クラス図を導出することが考えられる．これらについても，今後の課題としたい.

米国の銀行 ATM システムを事例とした適用評価では，提案アプローチの効果的な適用を確認した．適用結果のユースケースモデルは，基準モデルに対する適合率，再現率ともにほぼ妥当な値であり，提案アプローチの有効性を示している．適合率や再現率のマイナス要因は，手動によるゴールモデリング結果に由来しており，被験者の知識や熟練度に起因している．これらは，ゴールモデリングの実施要領を詳細に規定すれば改善できると思われるが，今後の研究課題である．

提案アプローチの STEP4 において，導出したユースケース図と KAOS モデルとの相互洗練は手動で実施したが，これをある程度定型化することも今後の研究課題である．また，今回の評価は二例に関するものであり，その他の事例についての評価も今後検討していきたい．

8.2.2　要求定義モデル駆動アーキテクチャ設計

対象システムのアーキテクチャをどう考えるかはKAOSゴールモデルの構造に暗示的に現れる．提案アプローチで導出する親ユースケースは，ゴールモデルのリーフゴールである要求ゴールに対応するが，そのアーキテクチャの構成員としての役割を担うことになる．提案アプローチでは要求ゴールはさらにイベントに対応するサブゴールに洗練されているが，上述と同様にこの洗練も要求ゴールがどのようなアーキテクチャで実現されようとしているかを暗示的に示している．要求ゴールとサブゴールの関係はそのまま親ユースケースとサブユースケースに引継がれる．また，親ユースケースのシステムでの位置付けも，KAOSゴールモデルにおける要求ゴールの位置付けをそのまま引継いでいる．つまり，対象システムのKAOSゴールモデルとそのリーフゴール（要求ゴール）のユースケースモデルにアーキテクチャに関する情報が機能構成として暗示的に含まれていることになる．その情報をどう抽出して，アーキテクチャ決定にどう反映するかは，本論文の成果を踏まえた次のステップとして進めていきたい．

また，堀田ら [48, 14, 49] は，KAOSゴールモデルをBPMNモデル [26] に変換し，BPMNモデル構築の難しさを緩和するとともに，要求分析の成果物とBPMNモデルとの整合性を確保する手法を提案している．中村ら [54, 53] は，プロダクトライン [30] において，複数製品それぞれの要求をモデリングしているゴールモデルからフィーチャモデルを作成するために，ゴールの可変性分析（共通機能と可変機能への分類）を効率よく実施するための共通ゴール判別手法を提案している．前者はゴールモデルをビジネスプロセスのアーキテクチャにどう反映するかであり，後者はゴールモデルをプロダクトラインにおけるフィーチャモデルのアーキテクチャにどう反映するかである．これらに，本論文提案アプローチの考え方を導入することによって，さらに有効性が向上する可能性が考えられる．これも，次ステップの課題である．

ソフトウェアに対する要求には機能要求と非機能要求がある．機能要求とはシステムが果たすべき機能的効果であり（入出力，処理，振る舞い），非機能要求とはシステムが機能要求を満たす上での特性である．非機能要求は品質要求（品質特性 [17] など）と種々の制約からなり，性能（効率性），信頼性，セキュリティ，使用性，保守性などがある．[39, 46]

設計工程では，機能要求のみならず非機能要求を効率よく実現するために，対象システ

ムのアーキテクチャを決定する必要がある．アーキテクチャの決定は機能実現のための設計と並行して実施される．それぞれの非機能要求を満足するように構成されたアーキテクチャパターンを選定し，対象システムへの適用に向けてインスタンス化する設計方法がよくとられる [50, 34, 1, 10]．その具体化のなかでは，実行時に解る品質（性能，セキュリティ，可用性，機能性，ユーザビリティ），実行時に解らない品質（改変性，移植性，再利用性，完全化可能性（integrability），テスト可能性），およびアーキテクチャ固有の品質（概念の統合性，正当性および完全性，構築可能性）を要求定義を根拠として適切に実現することが求められる [16]．

ICONIX のプロセスでは，ユースケースモデルを入力として，ロバストネス分析とアーキテクチャ検討を並行して実施する．ロバストネス分析は，システムの振舞い，入出力，処理などの機能要求に関する部分を，オブジェクト間のコミュニケーションとしてモデル化したものである．アーキテクチャは，上述したように非機能要求を実現できるように構築されなければならない．KAOS モデルは，機能要求のみならず，非機能要求も一緒に機能要求に関連付けてモデル化することができる．そこで，KAOS モデルにおける機能要求と非機能要求との関連から最適なアーキテクチャパターンを選定し，ロバストネス図で抽出されたオブジェクトを使ってインスタンス化することにより，アーキテクチャを決定する方法が考えられる．ICONIX の次の詳細設計プロセスとして，ロバストネス図のオブジェクト間のコミュニケーションを，アーキテクチャの決定に基づいたシーケンス図に発展させる．

このような，本研究の成果を踏まえたアーキテクチャ設計，さらにアーキテクチャに基づいた詳細設計への移行方法を，次のステップの研究課題としたい．

謝辞

本研究は JSPS 科研費 24300005, 26330081, 26870201 の助成を受けたものです．

本研究にあたり，ご多忙の中適切なご指導をくださった大須賀 昭彦 教授，田原 康之准教授に感謝いたします．同じく，大阪大学に転出された中川 博之准教授には，転出前後に渡り，ご多忙の中具体的にご指導頂きました．感謝申し上げます．さらに，協力研究員である東芝ソリューション販売（株）平山 秀昭氏と（株）東芝 早瀬 健夫氏には，貴重なご意見とご指導を頂き感謝致します．また，様々な協力をしてくださった大須賀・田原研究室の皆様に感謝の意を表します．そして，投稿した論文等に対して，国内外の多くの査読者から様々なコメントをいただき，大いに研究の参考になりました．

審査を快く引き受けてくださいました大学院 情報システム学研究科の田中 健次 教授，古賀 久志 准教授，石川 冬樹 客員准教授に感謝申し上げます．先生方には，論文のまとめ方，提案手法の説明方法，または実験結果のまとめ方などに関して多大なご指導をいただきました．

本研究では，KAOS ツールとして国立情報学研究所 GRACE センター所有の「K-tool」を使用させて頂きました．当ツールの利用に関してご協力を賜りました，国立情報学研究所 GRACE センター長/東京大学 本位田真一教授を始め関係者の方々に深く感謝致します．

参考文献

[1] LenBass(著), PaulClements(著), RickKazman(著), 前田卓雄 (訳), 佐々木明博 (訳), 加藤滋郎 (訳), 新田修一 (訳), 吉野圭一 (訳). 実践ソフトウエア アーキテクチャ. 日刊工業新聞社 日本, 2005.

[2] Russel C. Bjork. Atm simulation links - by topic. `http://www.cs.gordon.edu/courses/cs211/ATMExam-ple/index.html`.

[3] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. In *AAMAS 8*, pp. 203–236. Kluwer Academic Publishers, 2004.

[4] Sombat Chanvilai, Kozo Honda, Hiroyuki Nakagawa, Yasuyuki Tahara, and Akihiko Ohsuga. Goal-oriented Approach to Creating Class Diagrams with OCLConstraints. In *SAC2012*, pp. 1051–1056. ACM, 2012.

[5] Sombat Chanvilai, 本田耕三, 中川博之, 田原康之, 大須賀昭彦. Support for Generating UML Class Diagrams and OCL Constraints from KAOS Model. 37 ソフトウエア工学の基礎 18 FOSE2011, pp. 157–162. 日本ソフトウエア科学会, 2011.

[6] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. On Non-Functional Requirements in Software Engineering. In *Mylopoulos Festschrift LNCS 5600*, pp. 363–379. Springer-Verlag, 2009.

[7] Robert Darimont and van Lamsweerde. Formal Refinement Patterns for Goal-Driven Requirements Elaboration. In *SIGSOFT'96*, pp. 179–190. ACM SIGSOFT Software Engineering Notes, 1996.

[8] D.C. ゴーズ (著), G.M. ワインバーグ (著), 黒田純一郎 (監訳), 柳川志津子 (訳). 要求仕様の探検学. 共立出版 日本, 1995.

[9] Ariel Fuxman, Lin Liu, John Mylopoulos, Marco Pistore, Marco Roveri, and Paolo Traverso. Specifying and analyzing early requirements in Tropos. In *RE2004 9*, pp. 132–150. Springer-Verlag, 2004.

[10] F. ブッシュマン (著), R. ムニエ (著), H. ローネルト (著), P. ロンメルラード (著), M. スタル (著), 金澤典子 (訳), 水野貴之 (訳), 桜井麻里 (訳), 関富登志 (訳), 千葉寛之 (訳). ソフトウエアアーキテクチャ. 近代科学社 日本, 2006.

[11] Yoko Girier, Kozo Honda, Hiroyuki Nakagawa, Yasuyuki Tahara, and Akihiko Ohsuga. Visual-K:A Prototype for a Visualization Tool Modeling Goal-Oriented RE Methodology KAOS. In *ICSII 2011*, pp. 315–326. ASME, 2011.

[12] W Heaven and A Finkelstein. UML profile to support requirements engineering with KAOS. *IEE Proc-Softw*, Vol. 151, No. 1, pp. 10–27, 2004.

[13] Kozo Honda, Hiroyuki Nakagawa, Yasuyuki Tahara, and Akihiko Ohsuga. Goal-Oriented Robustness Analysis. In *Proceedings of the Tenth JCKBSE*, pp. 171–180. IOS Press, 2012.

[14] Hiroki Horita, Kozo Honda, Yuichi Sei, Hiroyuki Nakagawa, Yasuyuki Tahara, and Akihiko Ohsuga. Transformation Approach from KAOS Goal Models to BPMN Models Using Refinement Patterns. In *SAC2014*, pp. 1023–1024. ACM, 2014.

[15] IBM. Ibm rational unified process (rup). `http://www-01.ibm.com/software/rational/rup/`.

[16] IEEE, ACM, 松本吉弘（訳）. ソフトウエアエンジニアリング基礎知識体系：SWEBOK2004. オーム社 日本, 2005.

[17] ISO/IEC. Iso/iec 9126-1:2001 Software engineering – Product quality – Part 1: Quality model. `http://www.iso.org/iso/catalogue_detail.htm?csnumber=22749`.

[18] Axel Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *RE'01*, pp. 249–263. IEEE, 2001.

[19] Axel Lamsweerde. From System Goals to Software Architecture. In *LNCS2804*, pp. 25–43. Springer-Verlag, 2003.

[20] Axel Lamsweerde. *Requirements Engineering :From System Goals to UML models to Software Specification*. WILEY West Sussex England, 2009.

[21] Emmanuel Letier. Reasoning about Agents in Goal-Oriented Requirements Engineering. *Phd Thesis 2001*, pp. 1–68, 2001.

[22] Emmanuel Letier and van Lamsweerde. Reasoning about partial goal satisfaction for requirements and design engineering. In *SIGSOFT '04/FSE-12*, Vol. 29, pp. 53–62. ACM SIGSOFT Software Engineering Notes, 2004.

[23] Chao Li, Liang Dou, and Zongyuan Yang. A metamodeling level transformation from UML sequence diagrams to Coq. In *ICTCS 2014*, pp. 147–157. CEUR, 2014.

[24] Elkamel Merah, Halima Saidi, Nabil Messaoudi, and Allaoua Chaoui. Design of ATL Rules for Transforming UML 2 Communication Diagrams into Buchi Automata. *International Journal of Software Engineering and Its Applications*, Vol. 7, No. 2, pp. 19–34, 2013.

[25] John Mylopoulos, Mannuel Kolp, and Jaelson Castro. UML for Agent-Oriented Software Development: The Tropos Proposal. In *UML2001,LNCS2185*, pp. 422–441. Springer-Verlag, 2001.

[26] OMG. Business Process Model and Notation (BPMN). `http://www.omg.org/spec/BPMN/index.htm`.

[27] OMG. Documents associated with Meta Object Facility (MOF) 2.0 Query/View/Transformation, v1.0. `http://www.omg.org/spec/QVT/1.2/`.

[28] OMG. Documents associated with uml v2.4.1. http://www.omg.org/spec/UML/2.4.1/.

[29] OMG. Unified modeling language. http://www.uml.org/.

[30] Klaus Pohl, Gunter Bockle, and Frank van der Linden. *Software Product Line Engineering Foundations, Principles, and Techniques*. Springer-Verlag, Berline Heidelberg, 2005.

[31] Dick Quartel, Wilco Engelsman, Henk Jonkers, and Marten van Sinderen. A goal-oriented requirements modelling language for enterprise architecture. In *EDOC2009*, pp. 3–13. IEEE, 2009.

[32] William N. Robinson and Greg Elofson. Goal Directed Analysis with Use Cases. *Journal of Object Technology*, Vol. 3, No. 5, pp. 125–142, 2004.

[33] Dung Rosenberg and Matt Stephens. *Use Case Driven Object Modeling With UML; Theory and Practice*. Apress, Berkeley, CA 94710 USA, 2007.

[34] NickRozanski(著), EoinWooz(著), 榊原彰 (監訳), 牧野祐子 (訳). ソフトウエアシステム アーキテクチャ構築の原理. SB クリエイティブ 日本, 2014.

[35] Matt Stephens and Doug Rosenberg. ICONIX Process. http://iconixprocess.com/iconix-process/.

[36] Eric S.Yu. Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. In *3rdRE*, pp. 226–235. IEEE, 1997.

[37] Yutaka Yoshida, Kozo Honda, Yuichi Sei, Hiroyuki Nakagawa, Yasuyuki Tahara, and Akihiko Ohsuga. Towards Semi-Automatic Identification of Functional Requirements in Legal Texts for Public Administration. In *JURIX2013*, pp. 119–124. IOS Press, 2013.

[38] Eric Yu. i* an agent- and goal-oriented modelling framework. http://www.cs.toronto.edu/km/istar/.

[39] 一般社団法人情報サービス産業協会 REBOK 企画 WG. 要求工学知識体系：REBOK 第 1 版. 近代科学社 日本, 2011.

[40] 長瀬株式会社テクノロジックアート. UML2 ハンドブック. 翔泳社 日本, 2004.

[41] 山内亨和（監修）, 株式会社オージス総研オブジェクトの広場編集部. その場で使えるしっかり学べる UML2.0. 近代科学社 日本, 2008.

[42] 児玉公信. UML モデリング入門. 日経 BP 社 日本, 2009.

[43] 本田耕三, 平山秀昭, 中川博之, 田原康之, 大須賀昭彦. ゴール指向洗練パターンパターン駆動によるユースケースモデリング. 電子情報通信学会論文誌 D, Vol. J99-D, No. 3, pp. 238–254, 2016.

[44] 本田耕三, 中川博之, 田原康之, 大須賀昭彦. ケーススタディ：kaos/uml モデリング. 36 ソフトウエア工学の基礎 17 FOSE2010, pp. 155–160. 日本ソフトウエア科学会, 2010.

[45] 本田耕三, 中川博之, 田原康之, 大須賀昭彦. 洗練パターンによるゴール指向ユースケースモデリング. In *SES2014*, pp. 45–50. IPSJ/SIGSE, 2014.

[46] 山本修一郎. ～ゴール指向による!!～ システム要求管理技法. 株式会社ソフト・リサーチ・センター 日本, 2007.

[47] 神崎善司. 要件定義マニュアル. 秀和システム 日本, 2008.

[48] 堀田大貴, 本田耕三, 平山秀昭, 清雄一, 中川博之, 田原康之, 大須賀昭彦. リファイメントパターンを利用した kaos ゴールモデルから bpmn モデルへの変換. コンピュータソフトウェア, Vol. 32, No. 4, pp. 141–160, 2015.

[49] 堀田大貴, 本田耕三, 清雄一, 中川博之, 田原康之, 大須賀昭彦. リファインメントパターンを利用した kaos ゴールモデルから bpmn モデルへの変換. 39 ソフトウエア工学の基礎 20 FOSE2013, pp. 273–282. 日本ソフトウエア科学会, 2013.

[50] 岸智二, 野田夏子, 深澤良彰. ソフトウエアアーキテクチャ. 共立出版 日本, 2005.

[51] ダグ・ローゼンバーグ (著), マット・ステファン (著), 三河 淳一 (監訳), 船木 健児 (翻訳), 佐藤竜一 (翻訳). ユースケース駆動開発実践ガイド. 翔泳社 日本, 2007.

[52] 吉田豊, 本田耕三, 清雄一, 中川博之, 田原康之, 大須賀昭彦. 法令から機能要求を抽出するための支援ツールの実装と評価. 39 ソフトウエア工学の基礎 20 FOSE2013, pp. 155–160. 日本ソフトウエア科学会, 2013.

[53] 中村祐貴, 本田耕三, 中川博之, 田原康之, 大須賀昭彦. ゴールモデルの構造に基づいた共通ゴール判別手法の提案. 38 ソフトウエア工学の基礎 19 FOSE2012, pp. 63–68. 日本ソフトウエア科学会, 2012.

[54] 中村祐貴, 本田耕三, 中川博之, 田原康之, 大須賀昭彦. ソフトウェア再利用に向けた共通ゴール判別手法の提案. コンピュータソフトウェア, Vol. 31, No. 2, pp. 67–83, 2014.

[55] 吉田裕之, 山本里枝子, 上原忠弘, 田中達雄. UML によるオブジェクト指向開発実践ガイド. 技術評論社 日本, 1999.

研究業績

学術雑誌

1. 本田 耕三, 平山 秀昭, 中川 博之，田原 康之，大須賀 昭彦: ゴール指向洗練パターン駆動によるユースケースモデリング, 電子情報通信学会論文誌, Vol.J99-D, No.3, pp.238-254, Mar.2016.

2. 堀田 大貴, 本田 耕三, 平山 秀昭, 清 雄一, 中川 博之, 田原 康之, 大須賀 昭彦： リファイメントパターンを利用した KAOS ゴールモデルから BPMN モデルへの変換, コンピュータソフトウェア, Vol.32, No.4(2015), pp.141-160, 2015 年 12 月.

3. 中村 祐貴, 本田 耕三, 中川 博之, 田原 康之, 大須賀 昭彦： ソフトウェア再利用に向けた共通ゴール判別手法の提案, コンピュータソフトウェア, Vol.31, No.2(2014), pp.67-83, 2014 年 6 月.

国際会議

4. Hiroki Horita, **Kozo Honda**, Yuichi Sei, Hiroyuki Nakagawa, Yasuyuki Tahara, Akihiko Ohsuga: Transformation Approach from KAOS Goal Models to BPMN Models Using Refinement Patterns, Proceedings of the 29th Annual ACM Symposium on Applied Computing(SAC'14), pp.1023-1024, ACM, March 2014.

5. Yutaka Yoshida, **Kozo Honda**, Yuichi Sei, Hiroyuki Nakagawa, Yasuyuki Tahara, Akihiko Ohsuga: Towards Semi-Automatic Identification of Functional Requirements in Legal Texts for Public Administration, Proceedings of The 26th International

Conference on Legal Knowledge and Information Systems (JURIX2013), pp.175-184, IOS Press, December 2013.

6. **Kozo Honda**, Hiroyuki Nakagawa, Yasuyuki Tahara, Akihiko Ohsuga: Goal-Oriented Robustness Analysis, Proceedings of the 10th Joint Conference on Knowledge-Based Software Engineering(JCKBSE 2012), pp.171-180, IOS Press, August 2012.

7. Sombat Chanvilai, **Kozo Honda**, Hiroyuki Nakagawa, Yasuyuki Tahara, Akihiko Ohsuga: Goal-oriented Approach to Creating Class Diagrams with OCL Constraints, Proceedings of 27th ACM Symposium On Applied Computing(SAC2012), pp.1051-1056, ACM, March 2012.

8. Yoko Girier, **Kozo Honda**, Hiroyuki Nakagawa, Yasuyuki Tahara, Akihiko Ohsuga: Visual-K:A Prototype for a Visualization Tool Modeling Goal-Oriented RE Methodology KAOS, Proceedings of 2nd International Conference on Measurement and Control Engineering (ICMCE 2011), pp.315-326, ASME, October 2011.

査読付国内シンポジウム

8. 本田 耕三, 中川 博之, 田原 康之, 大須賀 昭彦：洗練パターンによるゴール指向ユースケースモデリング, 第 20 回 ソフトウエアエンジニアリングシンポジウム (SES 2014) 論文集, pp.45-50, 2014 年 9 月.

9. 堀田 大貴, 本田 耕三, 清 雄一, 中川 博之, 田原 康之, 大須賀 昭彦： リファインメントパターンを利用した KAOS ゴールモデルから BPMN モデルへの変換, 第 20 回 ソフトウエア工学の基礎 (FOSE 2013) ワークショップ論文集, pp.273-282, 2013 年 11 月.

10. 吉田 豊, 本田 耕三, 清 雄一, 中川 博之, 田原 康之, 大須賀 昭彦： 法令から機能要求を抽出するための支援ツールの実装と評価, 第 20 回 ソフトウエア工学の基礎 (FOSE 2013) ワークショップ論文集, pp.119-124, 2013 年 11 月.

11. 中村 祐貴, 本田 耕三, 中川 博之, 田原 康之, 大須賀 昭彦： ゴールモデルの構造に基づいた共通ゴール判別手法の提案, 第 19 回 ソフトウエア工学の基礎 (FOSE 2012) ワークショップ論文集, pp.63-68, 2012 年 12 月.

12. Sombat Chanvilai, 本田 耕三, 中川 博之, 田原 康之, 大須賀 昭彦： Support for Generating UML Class Diagrams and OCL Constraints from KAOS Model, 第 18 回 ソフトウエア工学の基礎 (FOSE 2011) ワークショップ論文集, pp.157-162, 2011 年 11 月.

13. 本田 耕三, 中川 博之, 田原 康之, 大須賀 昭彦: ケーススタディ:KAOS/UML モデリング, 第 17 回 ソフトウエア工学の基礎 (FOSE 2010) ワークショップ論文集, pp.155-160, 2010 年 11 月.

関連論文の印刷公表の方法及び時期

学術雑誌

1. 全著者名：本田 耕三, 平山 秀昭, 中川 博之，田原 康之，大須賀 昭彦.
   論文題目：ゴール指向洗練パターン駆動によるユースケースモデリング
   印刷公表の方法及び時期：電子情報通信学会論文誌, Vol.J99-D, No.3, pp.238-254, Mar.2016, 2016 年 3 月
   （第 4 章及び第 5 章及び第 6 章及び第 7 章と関連）

国際会議

2. 全著者名：**Kozo Honda**, Hiroyuki Nakagawa, Yasuyuki Tahara and Akihiko Ohsuga.
   論文題目：Goal-Oriented Robustness Analysis
   印刷公表の方法及び時期：Proceedings of 10th Joint Conference on Knowledge-Based Software Engineering(JCKBSE 2012), pp.171-180, IOS Press, August 2012.
   （第 4 章及び第 5 章及び第 7 章と関連）

3. 全著者名：Sombat Chanvilai, **Kozo Honda**, Hiroyuki Nakagawa, Yasuyuki Tahara and Akihiko Ohsuga
   論文題目：Goal-oriented Approach to Creating Class Diagrams with OCL Constraints
   印刷公表の方法及び時期：Proceedings of the 27th ACM Symposium On Applied Computing(SAC2012), pp.1051-1056, ACM, March 2012.
   （第 5 章と関連）

4. 全著者名：Yoko Girier, **Kozo Honda**, Hiroyuki Nakagawa, Yasuyuki Tahara and Akihiko Ohsuga

論文題目：Visual-K:A Prototype for a Visualization Tool Modeling Goal-Oriented RE Methodology KAOS

印刷公表の方法及び時期：Proceedings of 2nd International Conference on Measurement and Control Engineering (ICMCE 2011), pp.315-326, ASME, October 2011.

（第 6 章と関連）

本論文との関連の詳細

章	節		関連論文番号	関連する内容
4 章	4.1 節		1	変換アプローチの全体像，変換テンプレート
	4.2 節	4.2.1〜4.2.4 節	1	変換テンプレート
			2	変換テンプレート
		4.2.5 節	2	変換テンプレート
5 章	5.1 節		1	AND グラフ抽出アルゴリズム
	5.2 節		1	QVT 変換規則と変換アルゴリズム
			2	変換アルゴリズム
			3	操作とイベント，エンティティの関係からのオペレーション特定
	5.3 節		1	QVT 変換規則と変換アルゴリズム
			2	変換アルゴリズム
	5.4 節		1	統合アルゴリズム
	5.5 節		1	QVT 変換規則と変換アルゴリズム
			2	変換アルゴリズム
			3	洗練パターン操作からのシナリオ導出
	5.6 節		2	変換アルゴリズム
	5.7 節		1	変換アルゴリズムの適用範囲
6 章	6.1〜6.5 節		1	適用手順と結果
	6.6 節		1	適用手順と結果
			4	要求ゴール分解の視覚化支援ツール
	6.7〜6.8 節		1	適用手順と結果
7 章	7.1 節		1	適用実験の実施と評価
	7.2 節		2	適用実験の実施と評価

著者略歴

本田 耕三（ほんだ こうぞう）

1953 年　　長崎県島原市に生まれる
1972 年 3 月　長崎県立島原高等学校 卒業
1972 年 4 月　国立九州大学 工学部 電気工学科 入学
1976 年 3 月　国立九州大学 工学部 電気工学科 卒業
1976 年 4 月　日本電気株式会社 入社
2009 年 4 月　国立大学法人 電気通信大学 大学院 情報システム学研究科
社会知能情報学専攻 博士前期課程 入学
2011 年 3 月　国立大学法人 電気通信大学 大学院 情報システム学研究科
社会知能情報学専攻 博士前期課程 修了
2011 年 3 月　日本電気株式会社 退職
2011 年 4 月　国立大学法人 電気通信大学 大学院 情報システム学研究科
社会知能情報学専攻 博士後期課程 入学
2017 年 3 月　国立大学法人 電気通信大学 大学院 情報システム学研究科
社会知能情報学専攻 博士後期課程 修了予定