リレーションシップ駆動
要件分析

モデリング編　ver. 1.0

ValueSource technologies from SAPPORO
▶▶▶ http://www.vsa.co.jp

リレーションシップ駆動要件分析

モデリング編

この資料の内容

概要

この資料で説明するリレーションシップ駆動要件分析（RDRA）の背景、位置付け、前提事項、および要件定義のための考え方の枠組みについて述べます。

要件定義で記述すべき事項

網羅的に要件を定義していくための出発点として、システムを語る上での基本的な問いかけを整理します。

リレーションシップ駆動要件分析（RDRA）

リレーションシップ駆動要件分析のプロセスについて説明します。

主要なダイヤグラムとその関係

この手法で使用する各種ダイヤグラムと全体像を示します。

ダイヤグラムと詳細説明

RDRA を構成するモデル要素と対応する構成概念を整理し、サンプルを使用して具体的な記述例を示します。

要件定義の整合性を確保するには

要件定義の整合性を確保する手段として、関係ダイヤグラムを使用したトレーサビリティチェックについて説明します。

まとめ

この資料全体の結論をまとめます。

1. 概要

1-1. この手法の背景と使用目的

『要件定義にまつわる問題とその解決に向けて』では、要件定義の本来の目的は、そのシステムが価値を生み出す根拠となる基本コンセプトを表現することであり、そのためには、表現力、網羅性、整合性という特性を兼ね備えた成果物で要件を記述する必要があることを述べました。また、そうした目標を達成するには、そこに至るまでの道筋、すなわち作業の進め方を規定するプロセスが重要であり、その手段として当社が提唱している「リレーションシップ駆動要件分析」手法の概要について説明しました。

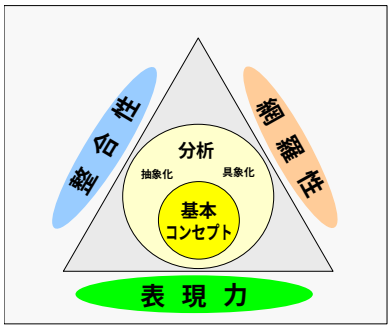

図 1:要件定義に求められるもの

モデリング編では、この手法に従って実際に要件モデリングを行い、最終的な成果物を完成させるための具体的な方法と関連する知識、ノウハウなどについて説明します。

1-2. 位置付けと前提事項

この手法では、あらかじめ以下の点が明確になっていることが前提となります。

1-2-1. ビジネス上の価値

システムを実現するにあたっては要件定義に先立ってそのシステムが提供するビジネス上の価値が明確になっている必要があります。その領域はどのようにビジネスを行い、何によって収益を上げるのかが明確になっていなければなりません。そのうえでシステム化対象の要件を策定することになります。

1-2-2. 要件定義の対象

この手法は上記のビジネスの価値（ゴール）を策定するところは対象としません。あくまでもそのゴールはある程度見えており、その後のシステム化のところを対象とします。

既にシステム化されて、その有効性や意義がはっきりしているものや、必要な機能がある程度明確になっているものを対象とします。そのようなシステムに新しい要求を追加して、新たなシステム像を定義するような場合を想定しています。具体的な例としては、パッケージソフトのメジャーバージョンアップ、既存システムの改善などが該当します。

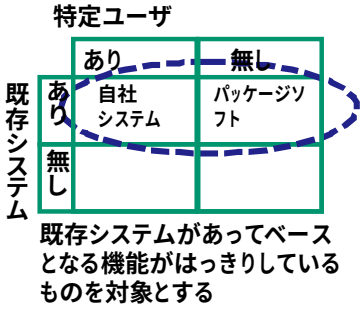

図 2:手法の対象範囲

1-2-3. 守備範囲

この手法は、ソフトウェア開発全体のうち要件定義の部分を対象とします。どのようにシステムを実現するかという方針は、アーキテクチャとして別に検討します。要件分析では、システムが何を行うのかを明らかにすることが目的です。つまり、How ではなく What（システムが何を行う必要があるのか）に焦点を当てます。

システムの実現方法については別途アーキテクチャとして定義され、アーキテクチャと要件要件分析とを基にシステムの設計が行われます。

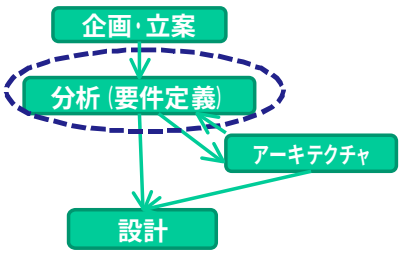

図 3:工程の関係

1-3. 要件定義のための考え方の枠組み

『要件定義にまつわる問題とその解決に向けて』でも説明したように、要件定義という作業をうまく進めるには、考え方の枠組みが必要です。また、その最終目標となる成果物は、網羅性、整合性、表現力という3つの特性を兼ね備えた形で、システムの基本コンセプトを明らかにするものでなければなりません。

1-3-1. 網羅性

まず、どのような事柄を記述すれば要件を定義したことになるのか、という基本的な問いかけがあり、それに基づいて要件定義として作成すべき資料が決まります。この手法では、思考を視覚化する手段として UML を利用するので、作成すべき資料は UML ベースの基本ダイヤグラムとしてリストアップされます。このセットに基づいて要件を定義することが、網羅的にシステムの要件を洗い出すことにつながります。

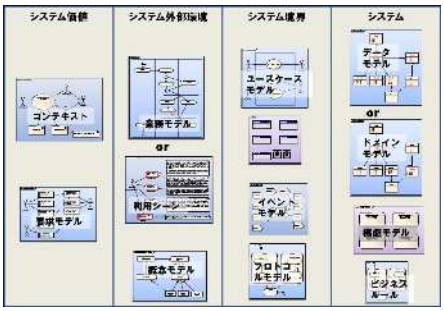

図 4:基本ダイヤグラム

1-3-2. 整合性

枠組みとして規定されたダイヤグラムに基づいて要件をモデル化しただけでは、モデル間に不整合が存在する可能性があります。複数のビューから定義した要件を相互に関係付け、矛盾がないかどうかをチェックすることで全体としての整合性を確保することができます。この手法では、上記の基本ダイヤグラムに加えて、ビューの異なるモデルを 1 つの図に混在させてその関連を描く関係ダイヤグラムを使用します。

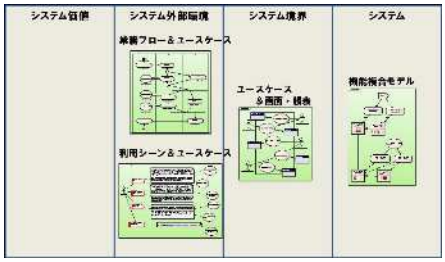

図 5:関係ダイヤグラム

1-3-3. 表現力

上記の成果物や活動を支える基盤となるのが UML の表現力です。『要件定義にまつわる問題とその解決に向けて』で見たように、要件定義を UML モデリングで行うことによって、プロジェクトのさまざまな局面で有利に作業が進められるようになるほか、最終的な成果物そのものの表現力を高めることができます。

1-3-4. 考え方の枠組み

上記のような特徴を備えた UML ベースの表現手段を用いながら、次のような考え方で要件を定義していきます。

* システムの外部環境を定めそこからシステムの内部へブレークダウンする考え方
* 要件を表現するモデルのセットとの各々のモデルの位置づけの明確化
* 関係ダイヤグラムを用いたトレーサビリティチェックの手法
* 反復プロセスによる網羅性、整合性を意識した洗練化

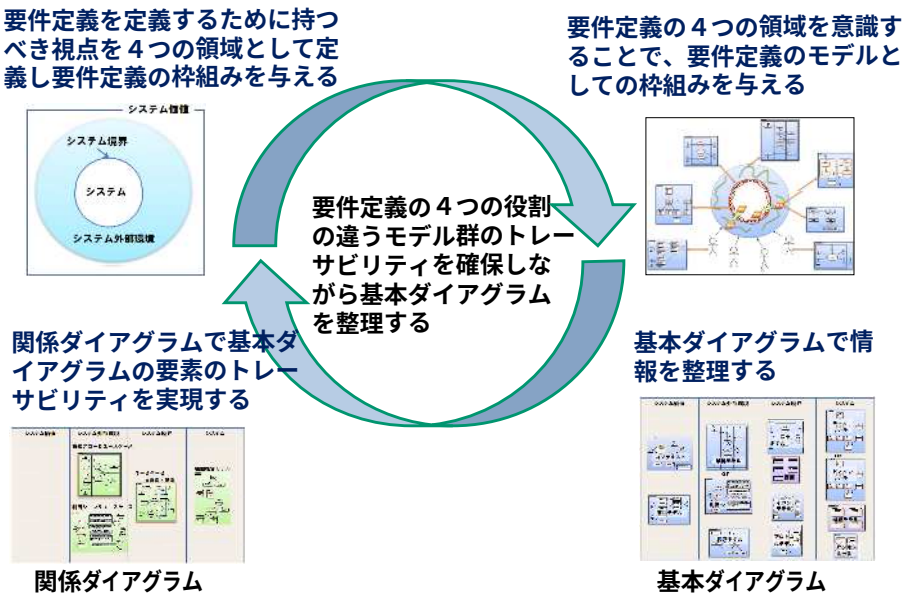

図 6:要件定義のためのモデリングの枠組み

2. 要件定義で記述すべき事項

網羅的に要件を定義していくための出発点として、システムを語る上での基本的な問いかけを以下にまとめます。これらの問いに答えられれば、要件定義ができたと考えることができます。

2-1. システム価値

* Q: 要求の元となる関係者（ロール）にはどのようなひとがいるか
* Q: どのような外部システムと連携するのか
* Q:どのような機能要求があるか、それは誰（ロール）が要求しているのか
* Q: どのような非機能要求があるか、それは誰（ロール）が要求しているのか

2-2. システム外部環境

* Q: どのような業務がありどのような流れになるのか
* Q: 業務にどのような人、組織が関わるが
* Q: 業務の中で使われる情報にはどのようなものがあるのか
* Q: このシステムはどのようなシーンで使われるのか
* Q: 対象となる業務ではどんな用語や概念が使用されているのか

2-3. システム境界

* Q: システム化範囲はどこまでか。その時のシステムとの接点はどのような処理になるのか
* Q: 画面・帳表はいくつあるのか、また各々の画面の主要な項目は何か
* Q: 画面・帳表はどのユースケースで使われるのか
* Q: 外部システムからのイベントにはどのようなものがあるのか
* Q: 外部システムとのインターフェイスはどのようなルールになっているのか

2-4. システム

* Q: どのようなソフトウェア機能（典型的な処理イメージ、画面・帳表との連携）があるのか
* Q: 扱うデータにはどのようなものがあるのか → データの種類、主な項目、データの量、成長量

3. リレーションシップ駆動要件分析 (RDRA)

ここでは、要件分析の具体的な手順について説明します。

3-1. 手法の概説

前のセクションでは、要件定義として網羅すべき事項を列挙しました。しかし、要件定義において本当に重要なことは、関係者の合意を得ながら「どのようなシステムを構築するのか」ということを分析的に整理し、明確にシステムのイメージを固めていくことです。そのためには、関係者を動かし、あるべき姿を徐々に明確化していくこと、またその過程で現れる問題に対応しながらプロジェクトを進めていくことが必要になります。

その際に必要となるアイデアの表現手段がモデルです。モデルを通じて現状を把握し、モデルとしてアイデアを視覚化し、そのモデルをベースに議論を行って、共通の認識基盤のもとで相互の合意をとりながらシステムの要件を固めていきます。モデルを使用することで対象を分析的に検討できるほか、個々のダイヤグラムとして表されたモデル間の関連をトレースすることで、全体の整合性を保ちながら要件を洗練させていくことができます。大切なことは、個々のダイヤグラムを完成させることではなく、ダイアグアム間の関係を重視しながら、関係者の共通の認識を固めていくことです。そうすることで、本当に価値のある要件定義のドキュメントを完成させることができます。

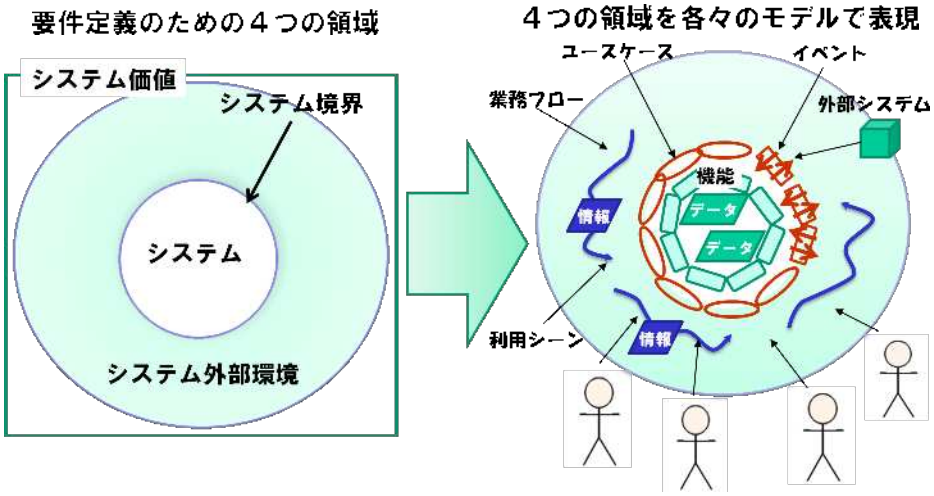

図 7:要件定義のための 4 つの領域

要件定義は単純なドキュメントの作成過程ではなく、対象を分析し、定義し、少しずつシステムの要件を固めていく作業です。そのためには以下の 2 つの点を意識しながら進めることが重要です。

* システムが対象とする業務または利用シーンの範囲
* 段階的につながりをもってシステム要件につなげる道筋

この 2 つを念頭にシステムの範囲を捉え、システム要件として詳細化していくために、大きく次の 4 つの領域として捉えることが有効です。

1. **システム価値**（コンテキストモデル、要求モデル）
2. **システムの外部環境**（業務フロー、利用シーン、概念モデル）
3. **システム境界**（ユースケース、画面・帳表、プロトコルモデル、イベントモデル）
4. **システム**（データ／ドメイン、機能、ビジネスルール）

細かい要求や漠然とした要求からシステム化対象領域を把握します。

3-1-1. システム価値：システムと関係を持つ対象とそこからの要求をとらえる

まず「システム価値」の視点として、システムと関係を持つ外部の人やシステム、およびシステムに対する要求の洗い出しを行います。

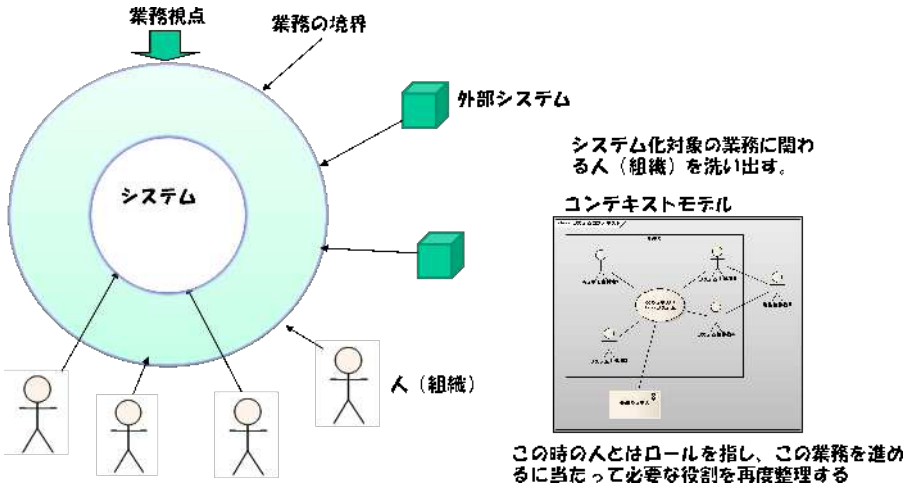

図 8:関係者を捉える

要求を分析するには、まずその要求を発生させる元となる存在を把握する必要があります。なぜなら、これらとうまく協調して価値を実現することがシステムの目標であり、そのために必要な条件を明らかにすることが要件定義という作業の目的だからです。このためには、システムの外側にある対象、具体的にはシステム化対象の業務に関わる人や組織、外部システムなどを洗い出す必要があります。こうして得られた結果は、**コンテキスト図**としてまとめます。

ロールの洗い出しができたら、それらに対応する実際の人や組織から要求を集めていきます。関係者の要求をもれなくヒアリングし、要求の粒度を調整しながら、本質的な要求を抽出し、要件へと洗練化していきます。要求の洗い出しと構造化には**要求モデル**を使用します。

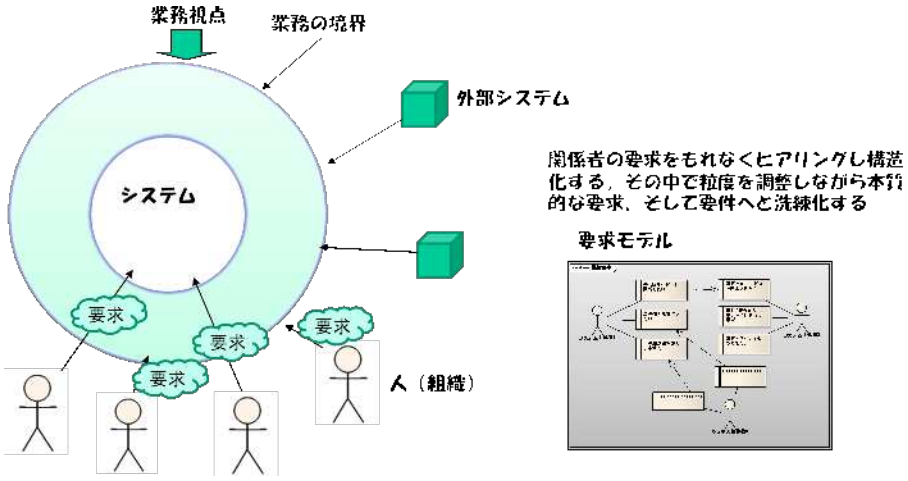

図 9:要求を捉える

3-1-2. システム外部環境:システムの外部環境をとらえる

次に、システムを取り巻く外部環境を明確にします。具体的には、業務フローや利用シーン、および関連する概念をモデルを作成します。

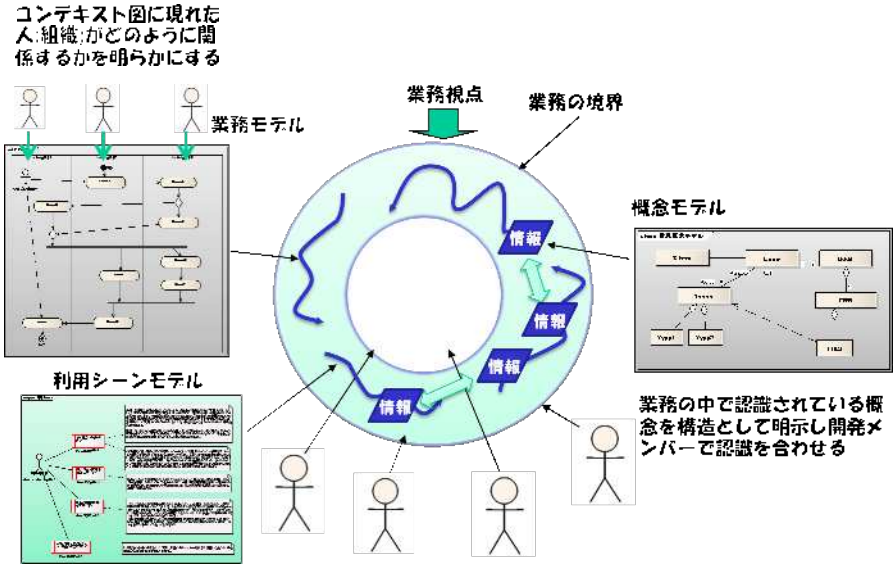

図 10:システムの外部環境を捉える

システムが果たすべき役割は、業務を支援してその業務によって生み出される価値を高めることです。この対象となる業務の流れを**業務フロー**としてモデル化します。他方、パッケージソフトの開発やサービス提供など、特定の業務を前提にすることが困難な場合は、そのシステムが使われる典型的な状況を物語風の文章による**利用シーン**として表現することで、同等の目的を果たすことができます。

「外部環境」の視点ではもう一つ、業務の中で認識されている概念を明らかにします。それらの名前、構造、他の概念との関係を**概念モデル**のクラス図としてモデル化します。この成果物をプロジェクトの関係者に明示することで認識を合わせ、共通の用語と概念に基づいて議論ができるようにします。

3-1-3. システム境界：内部と外部の境界部分を明らかにする

さらに、システムの内部と外部との境界部分へと分析を進めていきます。「システム境界」の視点では、ユースケースモデル、画面・帳表モデル、プロトコルモデル、イベントモデルを明らかにします。

ユースケースモデルでは、人間とシステムとがどのようなシステム機能（ユースケース）を通じて関係しあっているのかを表現します。すべてのユースケースが洗い出されれば、人または組織から見たシステムの境界が明確になったことになります。

人間がシステムと接する部分は、具体的には画面や帳表ということになります。システム境界を詳細に記述するには、これらの仕様をモデル化して表現します。このために**画面モデル**、**帳表モデル**を作成します。

他方、人間以外の外部システムとの関係は、外部システムの連携で発生する状態遷移を状態マシン図として整理し、そこで発生するイベントを洗い出すことによって外部システムとのインターフェイスが明らかになります。洗い出されたイベントの一覧によって、外部システムとのインターフェイス仕様が明らかとなります。この目的で作成する状態マシン図を**プロトコルモデル**と呼び、洗い出したイベントの一覧を**イベントモデル**と呼びます。

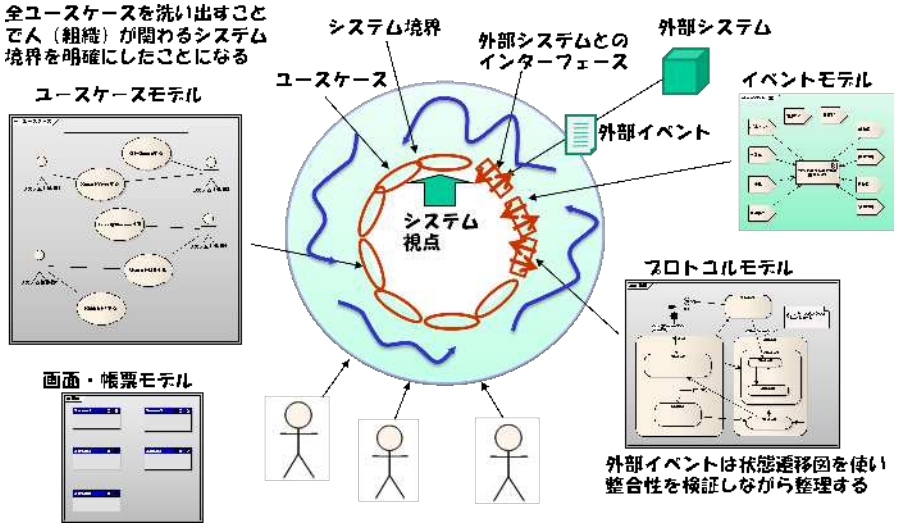

図 11：システム境界を捉える

3-1-4. システム：システム内部の構造を明らかにする

システム境界が明らかになったら、最後に内部の構造を明らかにしていきます。「システム」の視点からは、データモデル（またはドメインモデル）、機能モデル、ビジネスルールを作成していきます。

業務として扱う情報や、システムとして扱う情報を**データモデル**として明らかにします。一般にデータモデルはデータベースなどを使用して永続化することを前提としたデータを対象としますが、それに限らない、システムが保持するデータ全般のデータ構造やソフトウェア構造を**ドメインモデル**として表現することも有効です。

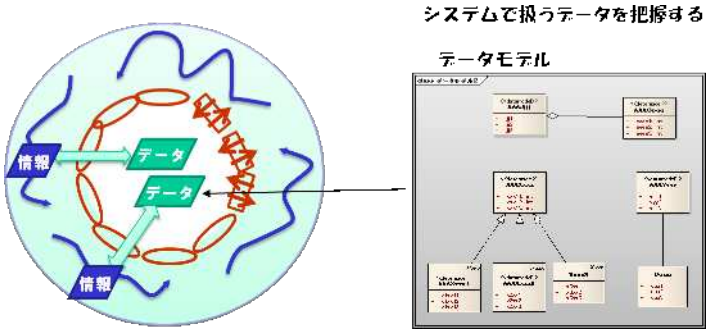

図 12：データを捉える

システムが提供する機能は機能モデルとして明らかにします。この機能とは、システムの外部から見た業務としての機能ではなく、システムの構成要素として提供されるものを指し、複数のユースケースや外部イベントとつながることができる機能です。

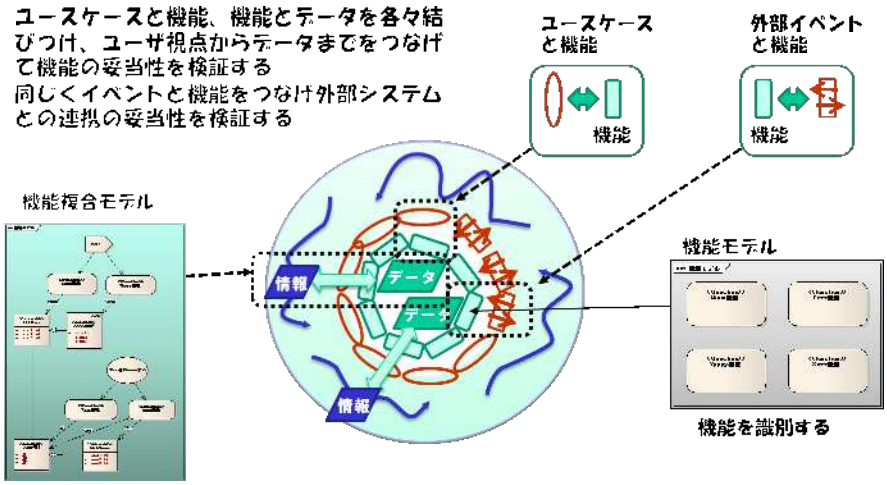

図 13：機能を捉える

対象とする業務そのものに起因する状態変化があれば、状態マシン図を使用したビジネスルールとしてモデル化します。この状態マシン図は、業務そのものに関するルールを記述することが目的で、システム境界でイベントの洗い出しを目的として作成するプロトコルモデルとは区別します。

3-2. モデリングの進め方

以上により、システム境界とそこで扱われる情報が明確化され、システムの外枠が確定します。これはシステムとして提供すべき機能とデータをより詳細に検討していくための基盤となり、それらを網羅的に行うための枠組みを提供するものでもあります。

こうして作成した個々の基本モデルを出発点として、各ダイヤグラム間の関係を当てはめ、各ダイヤグラム間の関係をトレースし、モデル全体として不整合や漏れがないかどうかをチェックします。こうすることで、全体としての整合性を保ち、かつ要件の抜けや漏れが生じないようにしながら、要件定義を洗練させていくことができます。

こうした一連の作業を反復しながらモデルを洗練させていきます。

各々のプロジェクトの状況に応じて、要件定義の進め方を検討します。その中で捉えるべき事、整理すべき事を洗い出します。その上で、ダイヤグラム間の整合性をトレースするモデル間の構成を設計し、モデリングを通じて要件定義を進めていくための支援をします。

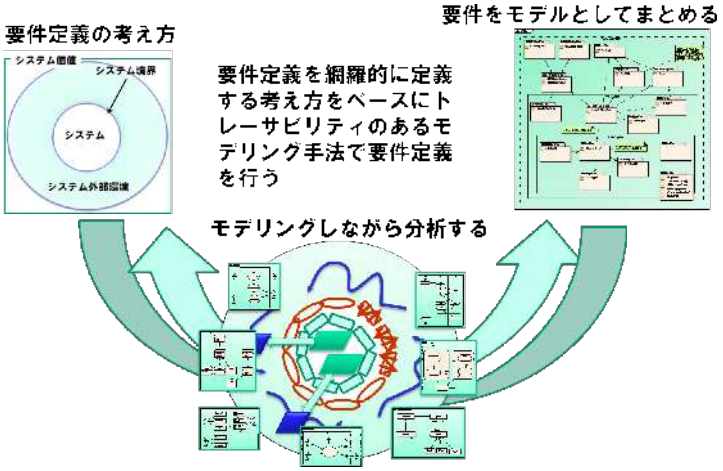

図 14:要件定義の考え方とモデルの関係

4. 主要なダイヤグラムとその関係

4-1. 基本ダイヤグラムと関係ダイヤグラム

この手法においては、基本ダイヤグラムに関係ダイヤグラムを組み合わせてモデルの整合性をチェックします。

基本ダイヤグラムは、次のような役割があります。

* モデリングの主たる対象となる
* 各々の視点における情報をもつ（業務フロー：Xxx 作業）
* 洗練化の対象となる
* 継続的に保守する
* 要件そのものを表す
  これらのモデルを説明できれば要件定義できたと考える

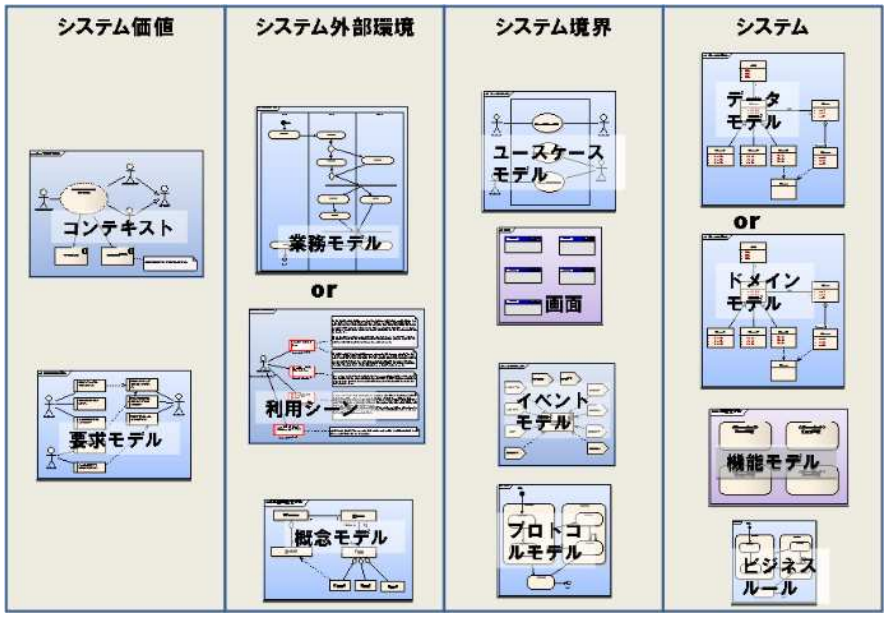

図 15:基本ダイヤグラム

他方、関係ダイヤグラムは次のような目的で使用します。

* 基本ダイヤグラムの関係を整理する
* 基本ダイヤグラムでは表せない関係を補足する
* 基本ダイヤグラムの検証に使用

主な関係ダイヤグラムとしては次のようなものがあります。

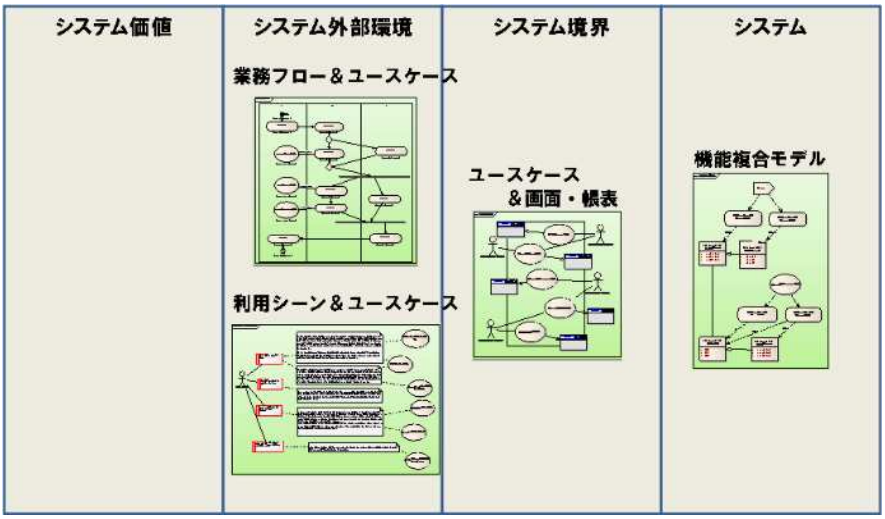

図 16:関係ダイヤグラム

4-2. 各ダイヤグラムの関係性

4-2-1. システム価値からのトレーサビリティ

システム価値に基づくトレーサビリティは、次のように整理することができます。

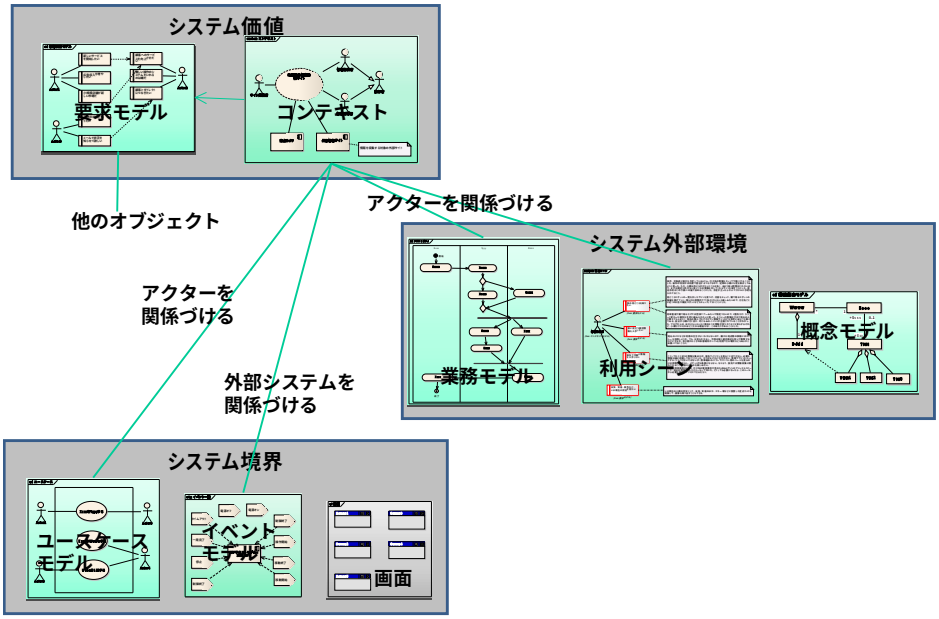

図 17:システム価値からの関係性

コンテキスト図で取り上げたアクターが要求元になり、それを要求モデル内のアクターと要求の関係付けで表します。同じくユースケースに関わるアクターはユースケースに、利用シーンに関わるアクターは利用シーンモデルで関係付けます。

業務モデルにおいては、レーンの名前にアクターの名前を対応付けます。コンテキスト図の外部システムは何かしらのインターフェイスを持つので、それはイベントモデルにおいて外部システムとイベントを関連付けます。

また、要求アイコンは他のすべてのアイコンと結びつけることができます。つまり、その要求を考慮したところ（アイコン）に関係付けます。

4-2-2. 外部環境と境界のトレーサビリティ

外部環境とシステム境界との間のトレーサビリティは、次のように整理できます。

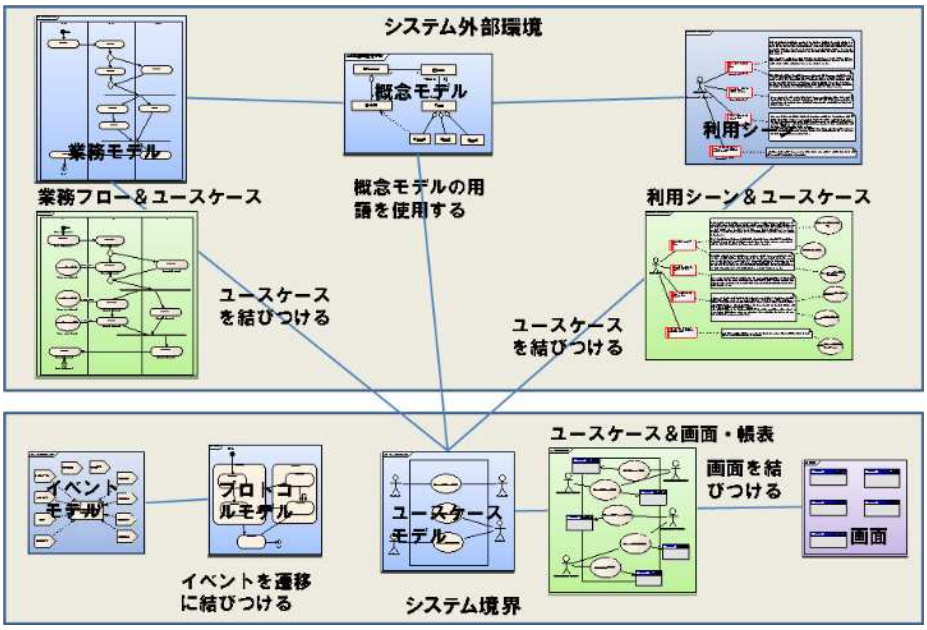

図 18:システム外部環境とシステム境界の関係性

業務モデル、ユースケースモデル、利用シーンモデルで使用する概念は、概念モデルで示した言葉を使用します。

業務フローの各アクティビティの中で、システムと関係を持つものについては業務フロー＆ユースケースダイヤグラムを使い、アクティビティとユースケースを結びつけます。

同じく利用シーンと結びつくユースケースは、利用シーン＆ユースケースダイヤグラムを使い、各利用シーンとユースケースを結びつける。また、画面・帳表とユースケースとの関係をユースケース＆画面帳表ダイヤグラムで表します。

イベントモデルのイベントと、プロトコルモデルの遷移には、状態マシン図の遷移に対応する名前を付けます。

4-2-3. 機能からのトレーサビリティ

機能に基づくトレーサビリティチェックは、次のようにまとめることができます。

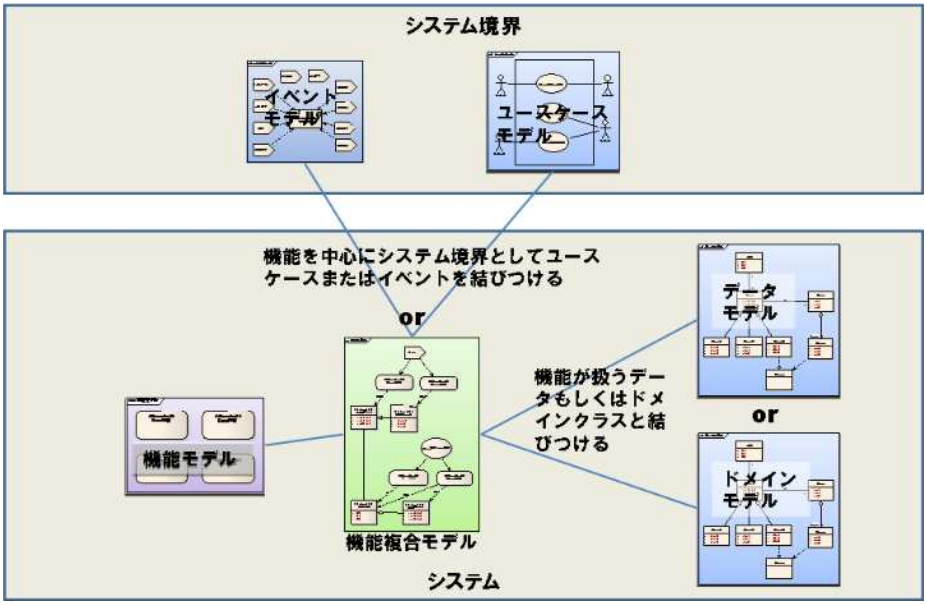

図 19:システム境界とシステムの関係性

システム領域内の各ダイヤグラムとシステム境界のダイヤグラムを結びつけるために、機能複合モデルを作成します。これにより、システム境界につながる機能とデータを明らかにすることができます。

このように、コンテキストで洗い出された各アクターの要求から、システムの機能、データまでを各ダイヤグラムでつなぎ、トレーサビリティを確保します。

5. ダイヤグラムの詳細説明

この手法では、上記の基本的な問いかけについて、それぞれに対応する視点からシステムの要件をモデル化して表現するための構成概念とダイヤグラムが用意されています。以下、それぞれに対応する基本的な問いかけと答え方、およびモデルの作成方法について、具体例を挙げながら順に説明します。

なお、この資料でサンプルとして示すモデルは、UML モデリングツールの Enterprise Architect (EA) を使用して作成しており、表記法の説明も基本的に EA を前提としています。

5-1. サンプル題材について

以下では「地域観光情報ポータルサイト」のシステム開発を例に、モデルの作成例を示していきます。サンプルでは、個々のモデルの例として断片的に取り上げるだけなので、最初に題材の全体概要を示しておきます。

5-1-1. サービスの目的

雑多な観光情報を収集整理し、より使いやすい形で提供すること。

5-1-2. サイトの機能

情報の収集

* 人からの情報収集
  何らかの動機づけによって情報を送付してもらう際の道具としての機能
* ネットからの情報収集
  ネット上に公開されている情報を自動的に収集する機能
  HTML のクローリング、RSS フィードの取り込み、Web サービスの利用
* 情報収集機器の制御
  定点カメラを制御してブログ等で利用できる地域の現在の写真をキャプチャする

情報の加工と整理

* データベース化
  いつ、どこで、だれが、なにを、の観点から情報を探し出せるように

情報の再発信

* 使いやすい情報検索インターフェイス
* 多様なアクセスへの対応
  Web、携帯、メール、他言語への翻訳

以上のような題材で要件定義モデルを行っていく場合の例をサンプルとして示しながら、この手法の構成概念および対応するモデルについて以下説明します。

5-2. システム価値

「要求」の視点では、システムと関係を持つ外部の人やシステム、およびシステム対する要求の洗い出しを行います。

5-2-1. コンテキストモデル

目的

この手法では、システムを取り巻く外部環境をコンテキストという言葉で表してモデル化します。コンテキストモデルは、最初に示した基本的な問いかけのうち次の2つに答えるものです。

* 要求の元となる関係者（ロール）にはどのようなひとがいるか
* Q: どのような外部システムと連携するのか

コンテキストのモデリングでは、ヒアリングやレビューで意見聴取を行う相手となるステークホルダ（利害関係者）、およびシステムの動作に関わる外部システムの洗い出しを行います。これがシステムに対する要求の発生元となります。対象毎に要求を洗い出しながら要件を定義することにより、システムの仕様決定や運用に影響を及ぼす外部の人間やシステムの範囲を把握し、網羅的に要求を捉えるための基盤とします。

コンテキスト図には運用時と開発時の 2 つに分けて作成します。運用時のコンテキスト図は、運用時に関わりのある人や組織を明示します。要求をヒアリングする時にヒアリングもれを防ぐためにも、運用時に関わる人や組織を把握することが重要で、そのために運用時のコンテキスト図を作成します。開発時のコンテキスト図では、開発時に関わりのある人を明示します。開発時に関わりのある人を洗い出しておくことで、開発時のコミュニケーションを円滑に進めるべき相手を把握できます。

モデリングの手引き

前述のように、運用時におけるシステムの外部環境を捉えるための運用時コンテキストだけでなく、仕様決定や開発プロジェクトの遂行に影響を及ぼすステークホルダを捉えるための開発時コンテキストも作成するようにします。開発時コンテキストは、開発作業そのものを円滑に進めていく上で重要な意味を持ちます。たとえば、プロジェクトメンバーが複数の役割を兼任する場合などにきちんと工数が確保できるようにするには、その人のマネージャーもステークホルダに含めておけば、キックオフミーティング等でそのマネージャに参加してもらったり、後日位置づけを確認するなどの対応を忘れずに行うことができます。

また、モデリングに際しては、境界線を利用して関わりの深い存在とそうでないものを分ける、システムとして影響を及ぼせるものとそうでないものを分ける、影響力の強いものと弱いものを分ける、などの工夫をします。

ダイヤグラムの作成では、UML のユースケース図で使われるアクターを人や組織に、外部システムをコンポーネント図のコンポーネントに、コラボレーションをコンテキストにそれぞれ対応づけてダイヤグラムを作成します。また、登場した人や組織、外部システムの間になんらかのつながりがある場合は、関連線を結んで関係を整理します。

サンプル

題材システムの運用時コンテキストモデルの例を示します。

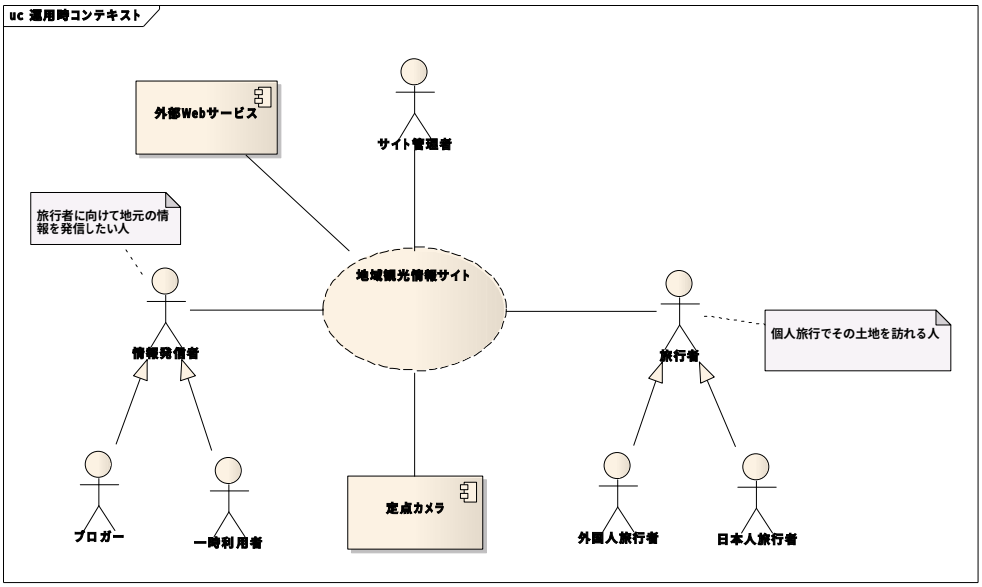

図 20:コンテキストモデル

このサンプルでは、まず 3 つのステークホルダを識別し、そのうち 2 つをさらに特化して合計 7 つのロールをアクターとして表しています。システムと連携する外部システムとして、外部 Web サービスと定点カメラがあり、それぞれコンポーネントのアイコンで表しています。

他のモデルとの関係

関係する外部システムを認識することで、システム連携のために必要なIOを識別する切り口が明確になります。外部システムに依存する項目を明確にし、外部システムに提供する機能、依存する外部システムの機能などを明らかにします。

外部システムの連携については、通常、外部システムからの送受信をイベントと捉え、受信するものを受信イベント、送信するものを送信イベントと捉えます。そうすることで受信をオブジェクトが用意するメソッドの呼び出し、送信を相手オブジェクトのメソッド呼び出しと考えることができます。

5-2-2. 要求モデル

目的

コンテキストが明らかになったら、次に要求の洗い出しを行います。要求モデルは次の基本的な問いかけに答えるものです。

* Q:どのような機能要求があるか、それは誰（ロール）が要求しているのか
* Q: どのような非機能要求があるか、それは誰（ロール）が要求しているのか

要件定義ではまず、顧客やユーザーの要望（「こうできるようにしたい」という思い）を洗い出し、それに基づいて、より具体的な要求（要望に応えるためにシステムができなければならないこと）が何なのかを検討します。その上で、他のステークホルダとの関係やアーキテクチャ上の制限なども加味し、最終的なシステムの要件（プロジェクトとして対応することを決定した事項）を定義していきます

箇条書きされた機能要求は網羅性や整合性が取りにくいだけでなく、本当に網羅的に書こうとすると膨大な量になってしまいます。その点、図的な表現は、言葉による箇条書きよりも内容を正確に記述でき、網羅性、整合性も取りやすいと言えます。

要望、要求、要件

要求の抽象度を表すため、次のように用語を使い分けるのも有効です。

* 要望：こうできたらいいな、思いつきレベル
* 要求：検討対象として合意されているもの
* 要件：今回の開発で実施すると決めたもの

定義の詳細や具体的な運用はプロジェクトの合意事項として決めるようにします。

機能要求と非機能要求

要求モデルは機能要求と非機能要求に分けて作成します。いずれも後述の要求アイコンを使用してモデル化します。

機能要求は、どんな要求があるのかを要求モデルとして明らかにした上で、各要求の詳細は、ユースケース図、機能ダイヤグラム、画面・帳表、外部イベント（入力、出力）、データモデル、状態遷移図（ビジネスルール）などの個別のダイヤグラムを利用して整理します。

非機能要求は、要求モデルの中で要求アイコンを使用して要求モデルの中で整理します。各要求の間の関係を整理することで、要求の優先順位、重要性などをがわかります。また、非機能要求を構造化することで、後でアーキテクチャの策定に対応しやすくなります。

抽象度、粒度、優先度

要求のモデリングは、抽象度、粒度、優先度に注意して行います。

抽象度は要件定義に適した詳細レベルとなるように注意します。たとえば、特定のソフトウェア機能を前提とする画面仕様などは、要件定義とは別の資料で定義するようにします。反対に、読む人によって記述の解釈にばらつきが生じる場合、要件定義として抽象的すぎることの兆候ですから、もう一段の具体化を検討します。

要件の粒度は、ステークホルダやユーザーがシステムの責務や役割として認識できるレベルにします。そうすることで、システムの仕様が元の要件を満たしているかどうかを直接ステークホルダに確認してもらうことができます。逆に言えば、それが要件を出す目的です。

複数の要望や要求から最終的な要件を導き出すには、各要求間の優先順位を明確にする必要があります。要求の優先順位を決定する要因としては、要求の種類（機能要求か非機能要求か）、要求を発しているステークホルダの位置付け（重要度）、および他の要求との関係（依存、類似、競合）などが挙げられます。こうした要因をきちんと把握し、合理的な判断に基づいて要件を定義する必要があります。そのためには、ゴール分析の考え方を適用して曖昧さを排除したり、各要求間の関係（依存、汎化、集約など）を図に表して思考を整理するなどの方法が有効です。

モデリングの手引き

UML には要求モデルのためのダイヤグラムがないので、この手法では、要求を表す専用のアイコンにクラス図と同様の関連（関係）の表記法を組み合わせたダイヤグラムで要求モデルを表現します。アクターごとの要求を洗い出し、要求アイコンを使用してダイヤグラム上に表現していきます。要求はアクターから出されるものなので、どのアクターにも結びつかない要求がある場合は、その理由を考える必要があります。

時間をかけて要求を洗い出したのに、開発現場では全く無視されている案件をよく目にします。要求を有効なものにするためには、必要最低限の要求にとどめることが重要です。数ばかり多く出しても、かえって重要な部分がわからなくなります。ポイントは目的を意識して方向性に沿ったものだけを洗い出すことです。

要求とシステム仕様との違いを意識して作業します。ソフトウェアは特定の要求を実現するための手段（How）であり、何を実現すべきか（What）という要求とは異なります。何らかのソフトウェアを想定した実現すべき事項は、要求ではなくソフトウェア仕様に含めるようにします。要求は、このソフトウェア仕様を作成するための方向性を示すものと捉えるとわかりやすいでしょう。ある場面において取り得る手段が複数ある場合に、適用すべき手段を検討するよりどころとして要求を捉えることができます。逆に言えば、要求は、設計、実装、テストの各段階で絶えず考慮されるようなものにします。

要求を整理するにはゴール分析の考え方を利用するのが有効です。たとえば、要求の内容が漠然としている場合、より具体的にどんな要求を満たせば元の要求が満たされたことになるのかを考えます。逆に、具体的な要求が多数出された場合は、それらを抽象化するとどのような要求として整理されるのかを考えます。ある要求が別の要求の実現を前提としている場合は、要求の依存関係を捉えることができます。このように、抽象化、具象化、依存などの関係を意識して全体の構造化し、整理していくことで、要求の優先順位を出したり、要求間の競合を発見することも可能になります。

上位のステークホルダでは要求が抽象的な場合が多く、いろいろな機能を横断する要求が多いため、言葉で要求をまとめるのが有効です。最終的には、上位の要求項目とシステム要件（各モデル）とを突き合わせることにより、システムの仕様と要求の整合が取れているかを検証します。他のダイヤグラムとの関連をトレースすることで、要求がどこまで実現されているのかを検証できます。こうした作業の繰り返しにより、要求そのものを洗練化していきます。

サンプル

題材システムの機能要求モデルのダイヤグラムを示します。

ステークホルダのアクターに関連づける形で要求アイコンを描き、要求の内容を関係に記述しています。漠然とした要求にはゴール分析の考え方を適用し、具体的に何ができればその要求が満たされたことになるのかを洗い出しています。この例では、UML の集約としてこうした関係を表しています。また、ある要求と、その要求の存在に依存して存在する要求との関係を、依存関係として破線の矢印で表しています。

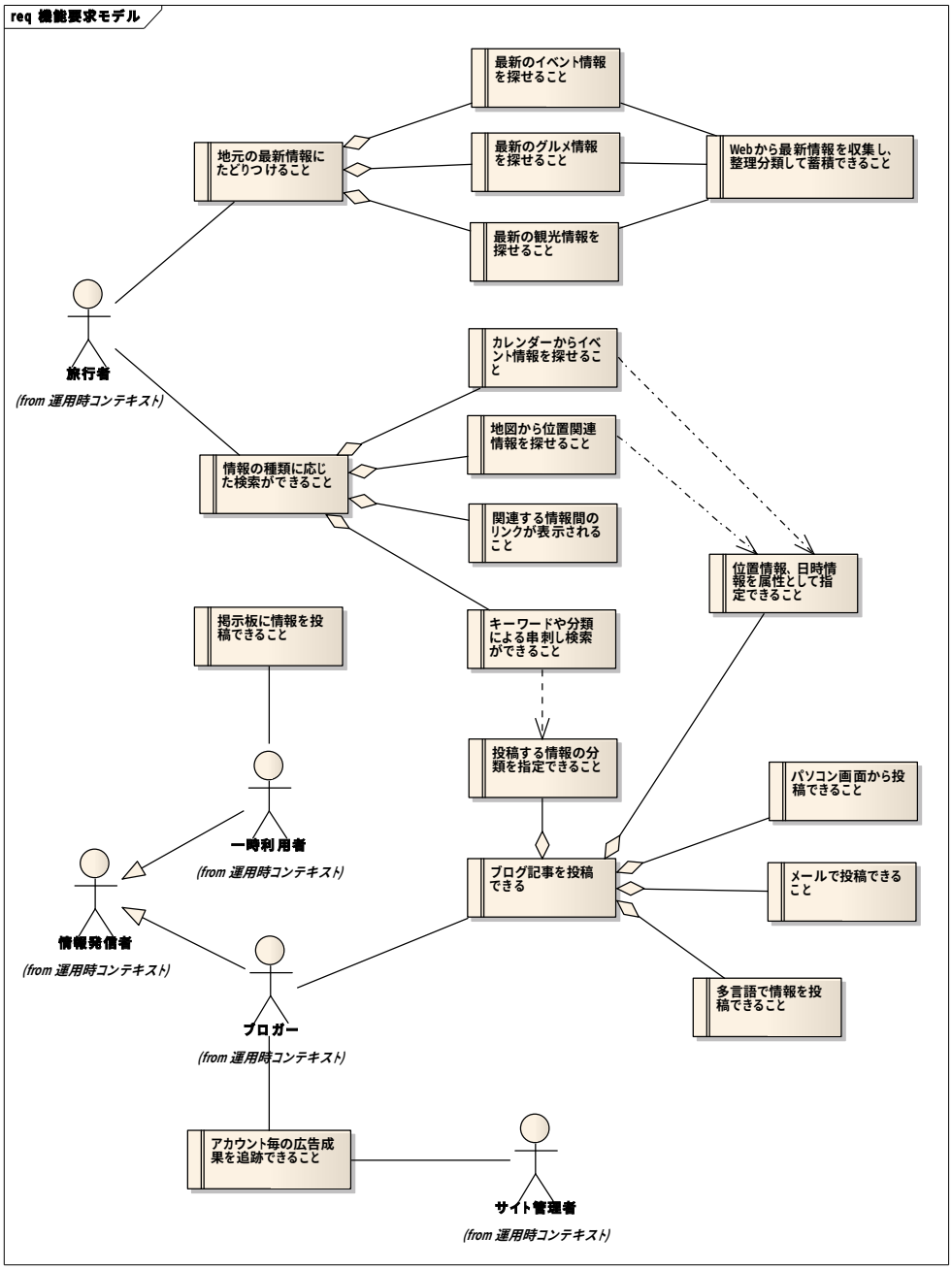

図 21:要求モデル

要求を洗い出すときのポイント

システム開発を行う場合、システムの機能に関心が向かいがちですが、機能に先立つ要求を整理することは整合性の取れたシステムを作るために必要です。そのシステムの利用者、価値を享受するユーザが何を望んでいるのかを整理することは間違ったシステムを作らないために必須の要件です。要求を整理し本質的な要求を見つけ出すことが重要です。そして端的な言葉でこのシステムの最重要な要求を洗い出し共有することが重要です。

5-3. 外部環境

「外部環境」の視点では、システムを取り巻く外部環境として、業務フロー、利用シーン、および関連する概念をモデル化します。

5-3-1. 業務フロー

目的

要求モデルは次の基本的な問いかけに答えるものです。

* Q: どのような業務がありどのような流れになるのか
* Q: 業務にどのような人、組織が関わるが
* Q: 業務の中で使われる情報にはどのようなものがあるのか

ここでの業務とは、特定の価値の創造を目的として行われる一連の活動のことです。この業務の流れをモデル化することで、システム化の対象となる世界でどのような価値が生み出されているのか、そのためにどのような業務が行われているのか、またコンテキストモデルで把握したアクターがどの業務に関係しているのか、といったことを明らかにします。なお、業務フローは特定の担当者の作業フローとは区別します。通常、複数の作業フローを組み合わせることで一つの業務フローが完成します。

まだシステム化されていない業務の場合、業務フローは、システム化の範囲やシステム化に伴う業務フローの変更などを検討するための基礎資料となります。他方、既にシステム化されている業務の場合は、新しい形に変えていくための準備として現状を把握するための基礎資料となります。いずれの場合も、システム化に関するアイデアをエンドユーザーとの間で交換するための数少ない情報媒体としての役割があるため、モデルはエンドユーザーに理解できる言葉と概念で記述するようにします。

モデリングの手引き

業務フローのモデルは UML のアクティビティ図を利用して表します。アクティビティ図では、処理の流れの実行主体をパーティション（レーン）として表現できますが、業務フローのモデリングにおいては、コンテキストモデルで洗い出したステークホルダがこれに対応します。ダイヤグラムでは、アクターのアイコンを使用して、どのステークホルダがどのフローを実行しているのかがわかるようにします。

業務を捉えることの意味

システムの要件定義を進めるに当たってビジネス的な価値を意識することが重要です。つい作ることに目が奪われ，なぜそのシステムが必要なのかを忘れてしまうことが多いからです。システムが使われる業務を把握することはシステム化の第一歩です。

作業にあたっては，ユーザ価値に結びつく業務の流れを把握（創造）するようにします。業務がどのような人や組織によって運用されどのような情報により進められているかを知ることはシステム化が必要な部分を洗い出す上で非常に有効です。

サンプル

下記は、題材そのものの業務ではありませんが、題材のような Web サイト構築の業務フローをアクティビティ図で表したものです。こうした業務をシステム化する場合、このような形で業務フローを整理することで対象業務を明らかにします。

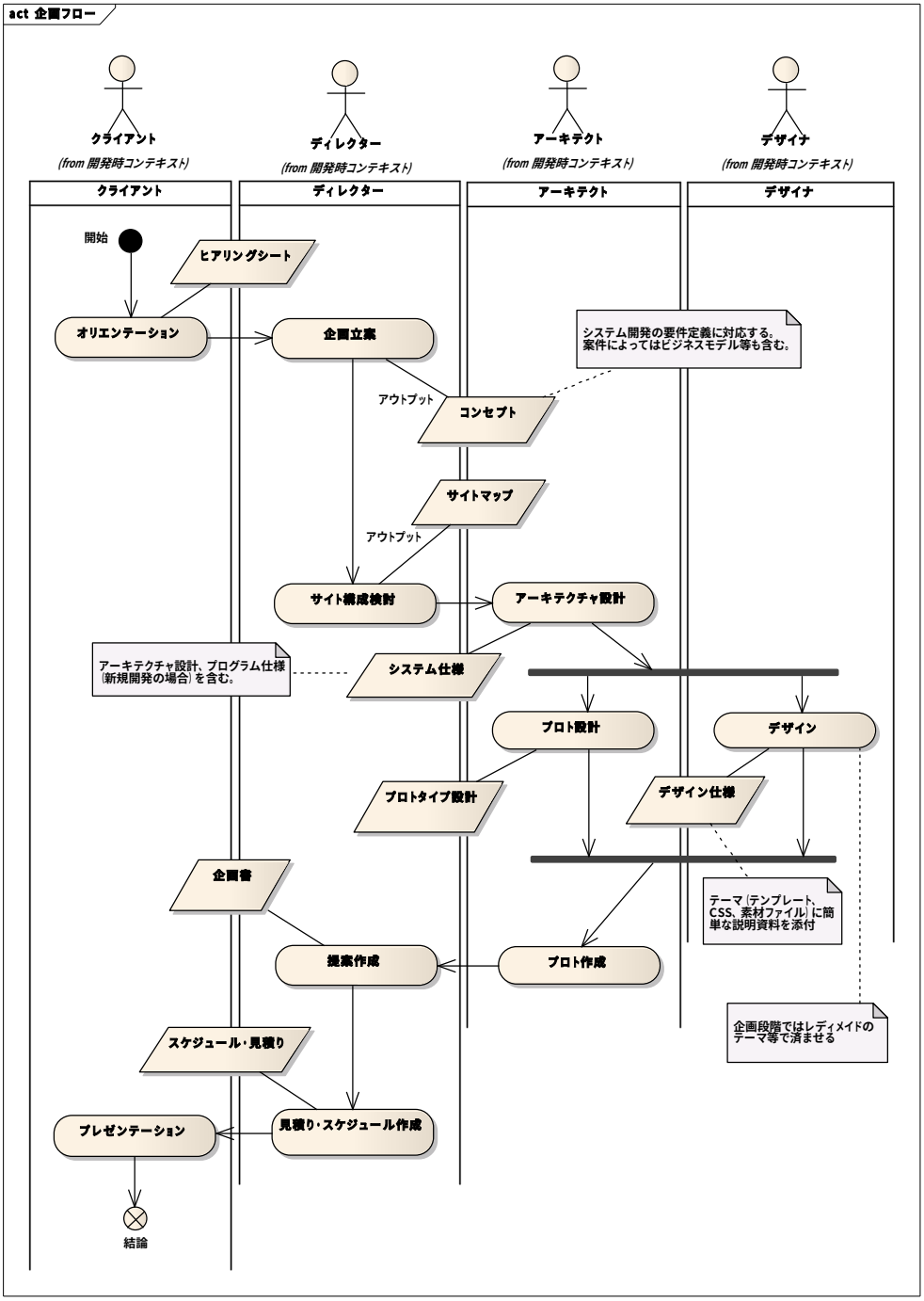

図 22:業務モデル

この例では、コンテキスト図のアクターをレーンに対応づけていますが、そうした表記が難しい場合やそのような対応関係になっていない場合は、アクターを図中の適切な位置に埋め込んでもかまいません。

5-3-2. 情報一覧 (オプション)

目的

情報一覧は、次の基本的な問いかけに答えるものです。

* Q: 業務の中で使われる情報にはどのようなものがあるのか

業務で扱う情報を洗い出します。必ずしもシステム化対象のものでなくてもかまいません。たとえば、伝票やメモ、壁に貼られて共有するデータなど、人が作業をする上で必要とする情報も含まれます。ここで洗い出された情報がユースケースに結びつく場合、画面や帳表など入出力部分のモデルとして、対応する項目を別に整理します。

情報一覧は、全くシステム化されていない分野をシステム化する場合など、扱われる情報を最初から導き出すようなときにを作成するモデルで、既に情報が明らかになっている更新案件などでは省略することもあります。

モデリングの手引き

通常、業務フローのアクティビティに結びつける形で洗い出されます。ひと通り洗い出しが終わった後、それらをまとめる形で整理します。

この段階では、情報の項目の型やサイズなど詳しい仕様を定義する必要はなく、項目の名前と意味が明らかになっていれば十分です。

また、アクティビティと情報をつなぐ関連名には「in」と「out」を付け、アクティビティの入力情報なのか出力情報なのかを明示します。

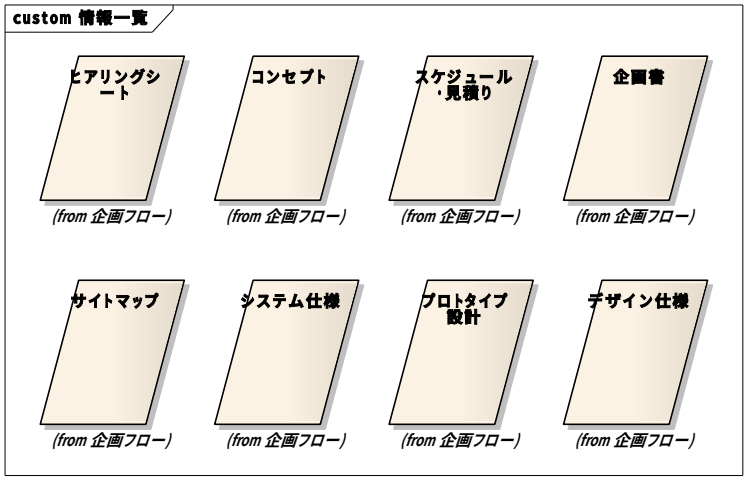

図 23:情報モデル

5-3-3. 利用シーンモデル

目的

利用シーンは、次の基本的な問いかけに答えるものです。

* Q: このシステムはどのようなシーンで使われるのか

汎用的に使用されることを前提としたパッケージソフトやツール、サービスなどの提供を目的とするプロジェクトでは、特定の業務フローのモデルを描くことが難しいこともあります。このような場合は、そのシステムが使われる場面や状況を物語風の文章で表した利用シーンを、ステークホルダのアクター別にノートとして記載する手法が有効です。

利用シーンは、複数メンバーで開発する場合にソフトウェアの機能を決める土台となる情報を合意する目的で使用できます。利用シーンが明確になると、ソフトウェアの機能をイメージしやすくなります。特に、ユーザーの視点からシステムの価値を考える場合に利用シーンが有効です（これに対して、業務フローはサービス提供側の作業イメージとして作られる傾向があります)。

モデリングの手引き

このシステムがユーザーにとってどのような場面で使われ、それがどのように役に立つものなのかを示します。そして、ユーザーが抱える問題に対して、どのような形で問題を解決するのかを示します。表現方法としては、具体的な問題解決の様子がイメージしやすいように物語風に記述します。

ダイヤグラムでは、UML のアクターに対応付けた要求アイコンとして記載します。コンテキストモデルで洗い出したアクター別に記述していきます。複数の利用シーンがある場合は、利用シーン別に要求アイコンを分けて書くと、後でユースケースに結びつけやすくなります。

内容は、利用シーンがイメージできるように記述します。抽象的な書き方では、その後のブレークダウンが困難になるので注意してください。

他のモデルとの関係

利用シーンはユースケースと関連づけることができます。この場合、利用シーンごとにユースケースを導き出します。１つの利用シーンに複数のユースケースを対応づけることもできます。また、複数の利用シーンで利用されるユースケースもありえます。ユースケースと利用シーンの記述が同じになっている場合は、利用シーンの物語としての膨らませ方が足りない可能性があります。どのような場面でシステムが利用されるのかを具体的にイメージできることが重要です。機能ベースで出した利用シーンは、視点がサービス提供側寄りとなり、どうしてもシーズ中心になりやすいという傾向があります。

利用シーンを捉える意味

業務フローが描けない場合でも，利用シーンを記述することで，そのシステムが生み出す価値を表現でき，またアクター（ステークホルダ）との対応関係も明らかにすることができます。

業務フローでも利用シーンでも，そのシステムがどのような場面で利用され，どのような価値を生み出しているのかを捉えることが重要です。

サンプル

旅行者というステークホルダからの要求の利用シーンモデルの例を示します。要求モデルで洗い出した要求に対応づける形で、その要求に対応してシステムがどのような場面でどう使われるのかを UML のノートとして記述しています。

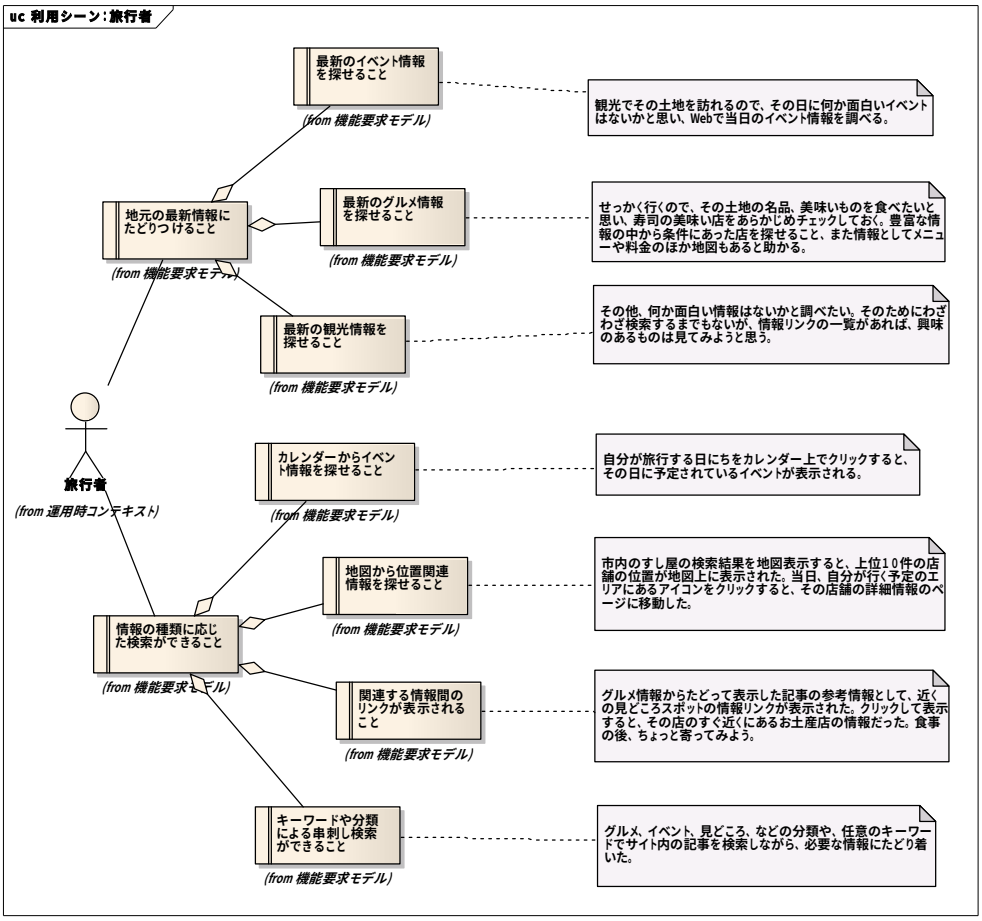

図 24:要求モデル

また、関係ダイヤグラムでユースケースとの対応関係を明らかにすることもできます。

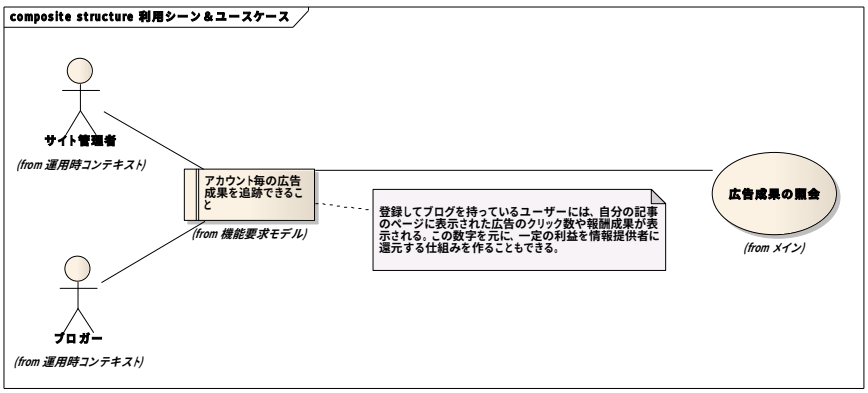

図 25:利用シーン＆ユースケースモデル

5-3-4. 概念モデル

目的

概念モデルは、次の基本的な問いかけに答えるものです。

* Q: 対象となる業務ではどんな用語や概念が使用されているのか

上流工程における打合せの難しさは、共通した概念の欠如が原因である場合が少なくありません。これは、同じ話でも、使う言葉の意味やその背後にある概念が異なると、解釈が違ってくることがあるからです。たとえば、中古車販売会社と自動車修理工場では「車」というものに対する概念が異なるかもしれません。あるいは、在庫管理業務と販売管理業務とでは、まったく別の視点で「車」という対象を捉えている可能性があります。こうした概念の相違を放置したままいくら議論をしてみても、コミュニケーションは成立しません。

話がなかなか噛み合わない時はまず、概念を整理しておくことが重要です。そのためには用語の意味を整理します。その業務の中で使用される用語が表す意味を統一することで、システムの要件や仕様を語るための言葉と概念の基盤が整備されることになります。

モデリングの手引き

モデリングに際しては、すべての概念をモデル化する必要はなく、共有する必要のある概念のみを扱うようにします。それほど厳密さを求める必要はなく、関係者の間で共通する概念が確認できれば目的は果たせたことになります。

概念モデルは、UML のクラス図で表現します。個々の概念が内部に保持する情報や機能をクラスとして表現するほか、汎化、特化、依存、集約、実現など、一般的なクラス間の関係を使用して、構成概念の持つ意味をより明確化します。

ソフトウェア機能を意識した概念モデルだけでなく、利用者の視点から捉えた概念を整理することも重要です。そうすることで、利用者の要求が新たに見えてくることもあります。

サンプル

この例は、クローラ、クローラターゲット、クロールリストなどの用語とその意味を概念モデルとして表したものです。用語に対応する概念をクラスとして表し、クラス間の関連として概念の意味的なつながりを表現しています。また、必要に応じて、クラスの属性や操作を記述することもできます。

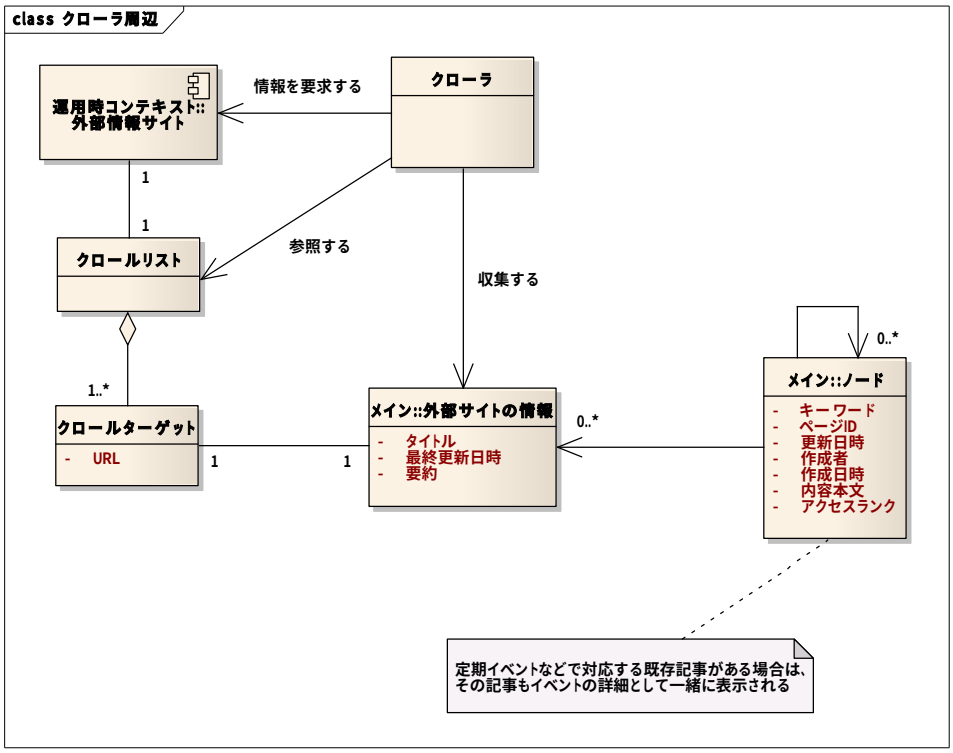

図 26;概念モデル

システムの外部環境を捉えることの意味

システムを取り巻く環境を知ることで、実際にそのシステムで実現すべきことや、実現に当たっての制約などを認識できます。そもそもシステムは周りの環境（人（組織）や外部システム）に適応し何らかの価値を提供する必要がありますから、そのためにもシステムが存在する環境を知ることがシステムの要件を決めるための第一歩となります。システムに関わる外部環境を明確にすることがシステムの網羅性を確保するための出発点となります（この外部環境を満たせばシステムに必要な機能は満たした、と考えることができる）。

5-4. システム境界

「システム境界」の領域では、システムの内部と外部の境界に位置するものとして、ユースケース、画面･帳表、プロトコル、イベントルールを明らかにします。

5-4-1. ユースケースモデル

目的

ユースケースモデルは、次の基本的な問いかけに答えるものです。

* Q: システム化範囲はどこまでか。その時のシステムとの接点はどのような処理になるのか

システムの要件定義では、システムの範囲がどこまでなのかを明確にする必要があります。このためにユースケース図を使用します。ユースケース図では、どのようなアクターがどのようにシステムに関わるのかを示します。この手法では、ユースケースを人間とシステムとの対話に関する範囲に限定します。外部システムとの連携に関する部分は、後述のプロトコルモデルとイベントルールを使用して明らかにします。

モデリングの手引き

ユースケース図を描く目的はシステム境界を明らかにすることなので、それ以外の情報は別のダイアログに記載し、それらを参照する形でユースケース図に結びつけるようにします。たとえば、ユーザーインターフェイスの振る舞いは「画面定義書」などとして、より詳細なドキュメントを別資料として作成します。ユースケース図に直接記述する情報としては、アクターとシステムの接点に焦点を合わせます。つまり、ユースケースをアクターとシステムの関わりを示すものとして位置づけ、ソフトウェアとしての機能には深入りしないようにします。ソフトウェアの機能については、機能モデルとして別に抽出します。システム境界とそこで使われるソフトウェアの機能は視点が異なるため、意識的に分けて表現する必要があります。

ユースケース記述を書く場合は、概念モデルの用語か、後述のデータモデルで定義されている用語を使うようにします。逆に、ユースケース記述で使いたい用語の意味に曖昧さを感じた場合は、これらのモデルとして明確に定義しておく必要があります。

ユースケース記述のイベントフローは、アクターが業務上意味のある何らかの価値を感じられる粒度にするのが原則です。この手法では、業務フロー、画面定義、機能（ソフトウェア機能）など別の視点から作成するモデルが多数あるので、それらと重複しない内容にします。たとえば、画面に関する情報は画面モデルに記述し、ソフトウェア的な機能は機能モデルに記述しますから、それらには触れない程度の粒度でイベントフローを書くことになります。

他のモデルとの関係

ダイヤグラムの作成では、単に一般的なユースケース図の要素（アクター、ユースケース）を描くだけでなく、コンテキスト、業務フローのほか、画面、帳表など、他のダイヤグラムとも結びつけながらシステム境界を表現することが有効です。たとえば、アクターにはコンテキスト図のアクターを利用し、ユースケースには業務フローのアクティビティをそれぞれ対応させ、外部との境界には画面などのユーザインターフェースや帳表を結びつけることができます。こうすることで、常に他のモデルとの関係を意識しながら、より整合性を確保しやすい形でシステム境界を定義していくことができます。

システム境界を捉えることの意味

システムがカバーする範囲を知ることは、システム対象の規模を見積もることになります。そしてシステムが提供する機能の範囲を明確にすることでもあります。

サンプル

ユースケース図の例を示します。アクターが利用するシステム機能を洗い出すことで、システムの外部仕様としてシステム境界が明らかになります。

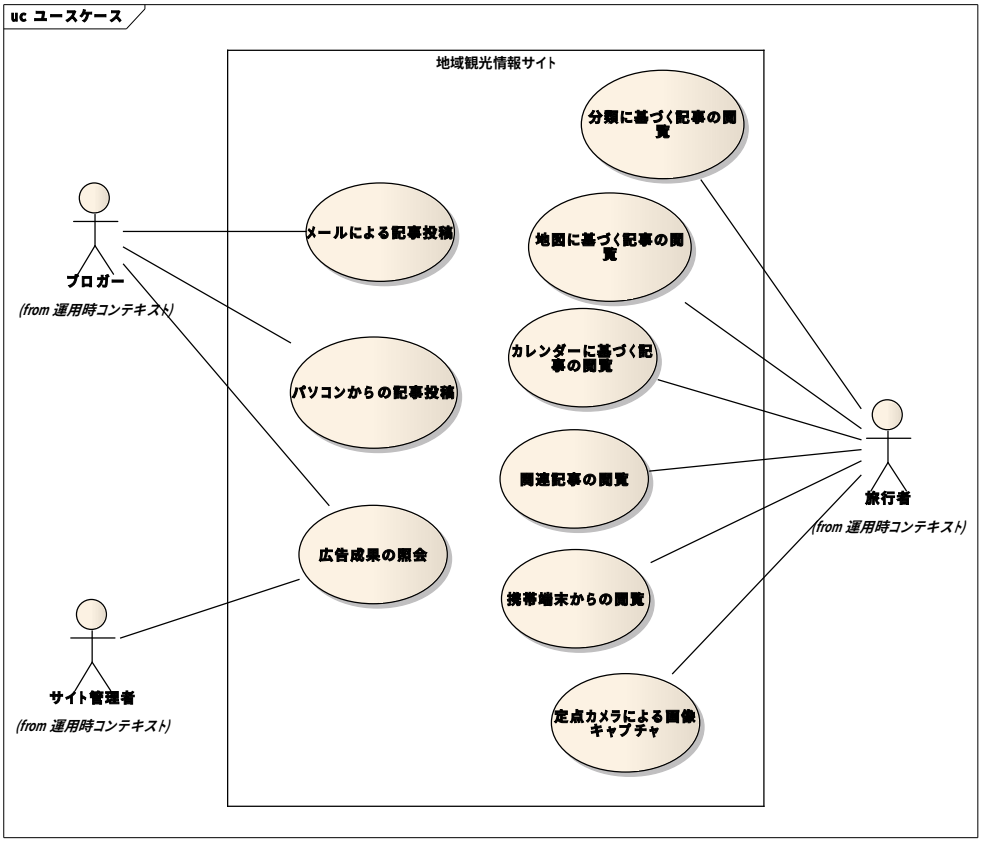

図 27:ユースケースモデル

5-4-2. 画面・帳表モデル

目的

画面（または帳表）モデルは、次の基本的な問いかけに答えるものです。

* Q: 画面・帳表はいくつあるのか、また各々の画面の主要な項目は何か
* Q: 画面・帳表はどのユースケースで使われるのか

業務に結びつく情報の入出力として、ユースケースに結びつく画面や帳表を洗い出します。モデリングに関しては画面も帳表も同じなので、以下では画面モデルのみ記述します。

モデリングの手引き

主要な画面について、次の情報をダイヤグラムとして整理します。

* 何画面あるのか
  ダイヤグラムの中の画面を数えると主要な画面の数が分かる
  画面の数、項目数は見積もりの重要な指標になる
* 画面はどのユースケースで使われるのか
  画面とユースケースを結びつける
  ユースケースと結びつく画面が業務に結びつく
* 画面・帳表にはどのようなものがあるのか
  画面・帳表モデルでは主要項目を網羅する

UML には画面や帳表を直接表現するための表記法は用意されていないので、クラスアイコンを使用したり、モデリングツールで提供される専用のアイコンなどを利用します。どのような方法を採るかは、プロジェクト内で合意した上で決定します。

サンプル

Web システムの画面に対応する表示ページを、専用のステレオタイプを指定したクラスアイコンで表現した例を示します。表示する項目や型、サイズなどを定義できます。

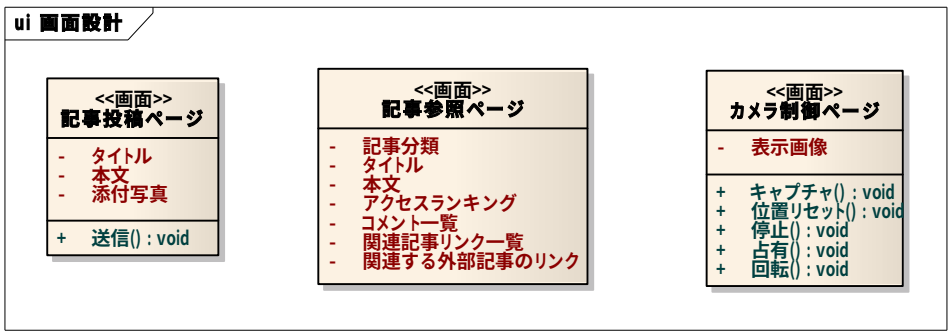

図 28: 画面・帳表モデル

画面や帳表を捉える意味

ユーザがシステム価値を得るのは画面や帳表などのIOと外部システムとの連携しかありません。そのIOを認識することは要件定義の重要な側面です。

5-4-3. プロトコルモデルとイベント一覧

目的

プロトコルモデルとイベント一覧は、次の基本的な問いかけに答えるものです。

* Q: 外部システムからのイベントにはどのようなものがあるのか
* Q: 外部システムとのインターフェイスはどのようなルールになっているのか

外部システムとの連携は、相手が人間ではないので、状態マシン図を使用してルール化することができます。データのライフサイクルや、この条件では機能しないが、ある条件の時だけできることがある、などの複雑な条件は状態マシン図で整理します。ここで識別される状態とは、外部システムのと該当システムの両方を含んだものになります。つまり、外部システムが持っているインターフェースをこのシステムでどのように使用するか（制約も含め）を状態としてルール化するのが、このモデリングの目的です。

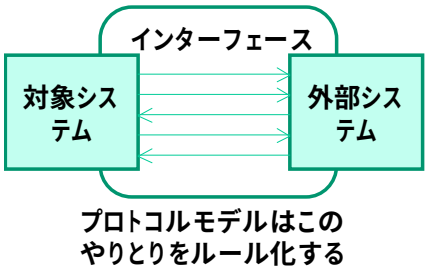

図 29: プロトコルモデルの表現対象

ルールの記述には、自然言語、デシジョンテーブルなどいくつかの手段がありますが、状態マシン図が最も網羅的で正確にルールを記述できます。

モデリングの手引き

状態遷移を引き起こすイベントには、外部に提供するものと外部に依存するものがあります。外部に提供するイベントとしては、情報提供、イベント通知があり、外部に依存するイベントとしては、外部機能の利用（呼出）があります。これらを整理してイベントルールのダイヤグラムとしてまとめ、情報提供はアクティビティアイコンを使って表現します。また通常、既存の外部システムには複数のインターフェースがあるので、そのインターフェースの中から使用するものをルール化することで提供されるソフトウェア機能を明らかにします。

なお、外部システムでも多くの Web サービスのように 1 回の呼び出しで完結する場合は状態マシン図を作成する必要はありません。

また、外部システムが既存のシステムとして存在する場合は、インターフェイスとしてイベントが先に決まっていることもあります。その場合は、このシステムに関係するイベントだけを取り上げ、その関係をプロトコルモデルとして明確にします。

サンプル

サンプル題材の定点カメラの制御についてプロトコルモデルとイベント一覧を作成した例を示します。

状態マシン図を作成する中で、それぞれの遷移に対応するイベントを識別します。状態図がひととおり完成したら、これらのイベントを抽出してイベント一覧にまとめます。このイベント一覧の作成によって、外部システムのインターフェイスをルール化されたことになります。

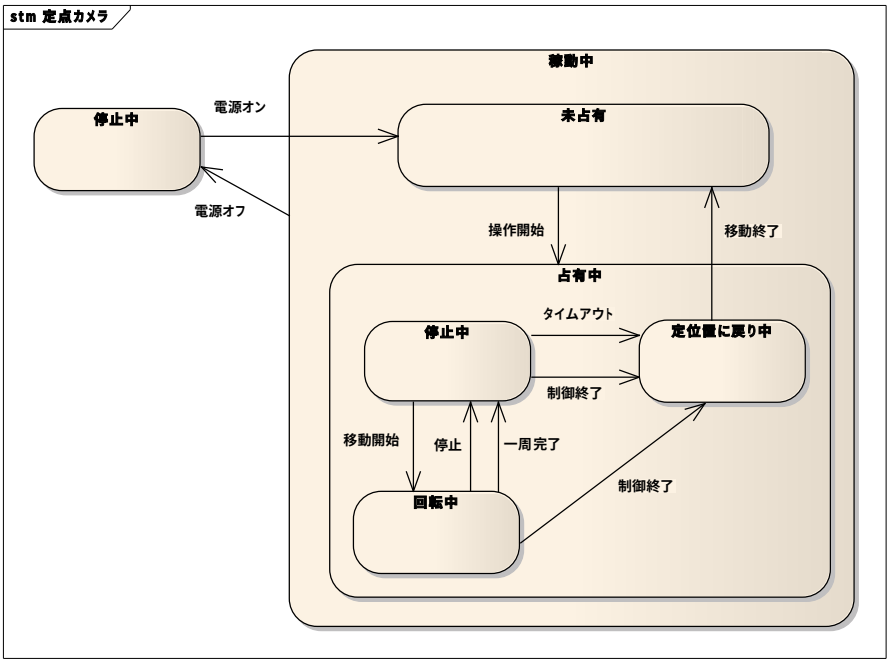

図 30:プロトコルモデル

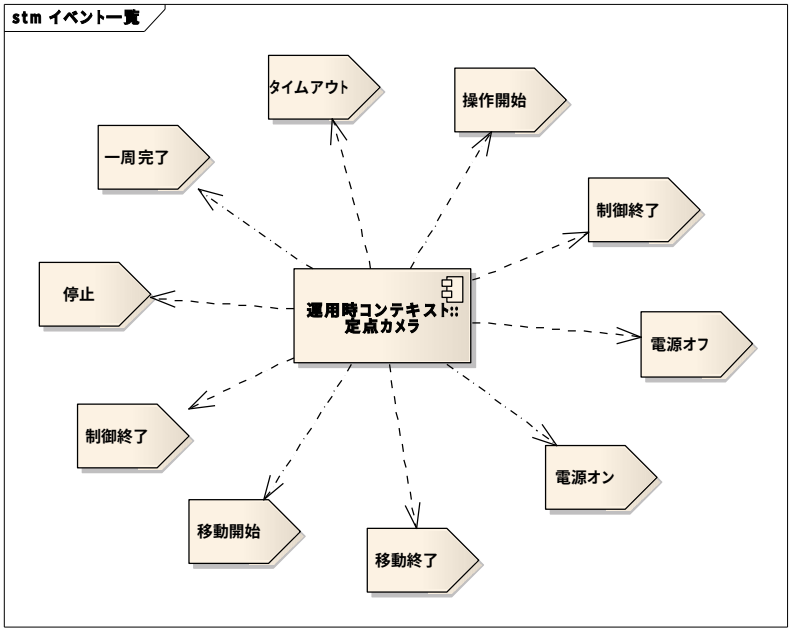

図 31:イベントモデル

5-5. システム

最後にシステムの内部構造を明らかにしていきます。「システム」の視点からは、データモデル（またはドメインモデル）、機能モデル、ビジネスルールを作成していきます。

5-5-1. データモデル／ドメインモデル

目的

データモデルは次の基本的な問いかけに答えるものです。

* Q: 扱うデータにはどのようなものがあるのか

→ データの種類、主な項目、データの量、成長量

システムが扱う情報、つまり画面や帳表で洗い出された入出力項目をデータモデルとして整理します。システムを構成する各機能の前提となる共通の情報構造をモデルとして明確にしておくことで、システム全体の整合性を確保することが目的です。モデリングの初期段階では、業務中に出てくる情報を概念的に扱うレベルにとどめます。作業が進んでシステムの機能が見えてくるに従って、それぞれの機能が使用する具体的なデータ構造へと洗練していくことができます。

モデリングの手引き

データモデルは UML のクラス図を使って表現します。クラス図は概念モデルでも使用しますが、概念モデルのクラス図は用語や概念の意味を明確にする目的で作るもので、データモデルのクラス図とは目的が違うことに注意してください。概念モデルと同じ構造でデータモデルを表現できる場合、両者のクラス図が重複していても問題ありません。

各データの属性は主要なものだけでよく、個々のデータの詳しい内容よりむしろデータ間の関係が重要です。また、データモデルは ER 図として表現することも可能ですが、その場合はテーブル設計の ER 図ではなく、論理モデルとしての ER 図を作成するようにします。テーブル設計のレベルまでブレークダウンしてしまうと、システムの実装に必要な情報までモデルに入り込んでしまい、その後の洗練化に悪い影響を及ぼす可能性があるからです。

異なる機能の間で整合が取れているかどうかは、(1) 各機能が扱うデータの把握（CRUD の把握）、(2) 個々のデータのライフサイクルの整理（上記 CRUD の整合性の確認）の 2 点から検証します。

サンプル

サンプル題材のデータモデルを示します。

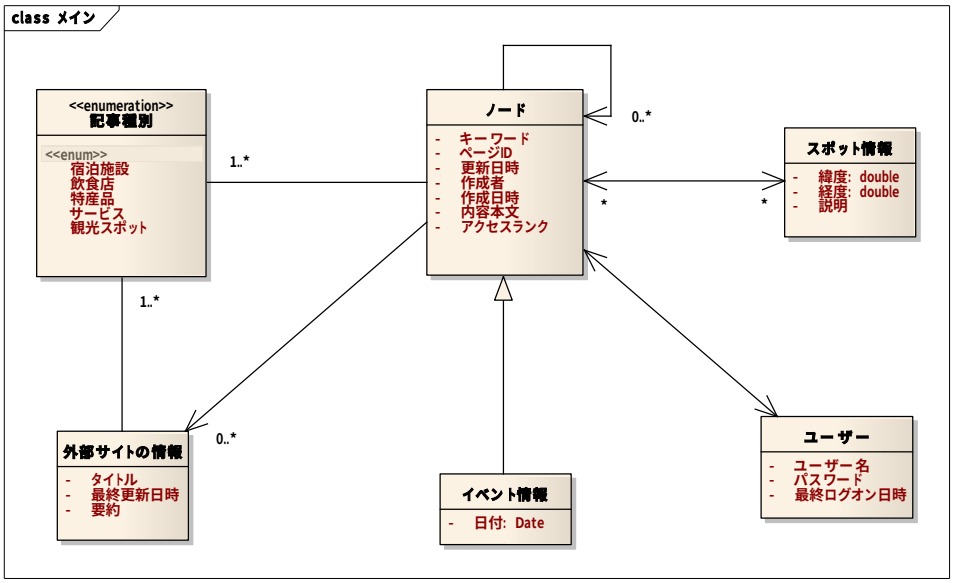

図 32:データモデル

永続化される情報を前提に、システム内で扱うデータの構造をクラス図で表しています。

データを捉えるポイント

通常のビジネスシステムでは必ずデータの操作が含まれます。入出力部分として識別されたものの背後にどのようなデータが存在し、それらがどのような構造になり、異なるタイミングで存在する入出力の整合性をどのような構造で保つかはシステムの構造の骨格をなします。

5-5-2. 機能モデル

目的

機能モデルは、次の基本的な問いかけに答えるものです。

* Q: どのようなソフトウェア機能（典型的な処理イメージ、画面・帳表との連携）があるのか

洗い出された機能をもとに典型的な機能の処理イメージ、および画面や帳表との連携を洗い出します。ここで扱う機能とは、ソフトウェアとして実現する機能を指します。ただし、ここで洗い出した機能がそのまま実装につながるわけではありません。具体的な実装は、アーキテクチャの設計に基づいて決まるので、モデル化した機能をブレークダウンするような形で実装をイメージしないことが重要です。

モデリングの手引き

システムのソフトウェア機能は機能ダイヤグラムにまとめられ、機能はアクティビティアイコンを使用します。このダイヤグラムの役割は、ユースケースを実現するためのシステム内の機能を明らかにすることです。機能は複数のユースケースから利用でき、またイベントとつながる機能も記述します。

UML モデリングツールの Enterprise Architect を使う場合、機能をアクティビティをとして扱うことで、状態マシン図のアクションとして利用可能となります。

サンプル

題材システムの機能モデルを示します。作図には Enterprise Architect を使用しており、個々の機能をアクティビティとして表現しています。こうすることで、状態マシン図側で遷移時のアクションを指定する際に、これらの機能を選択して指定できるようになります。

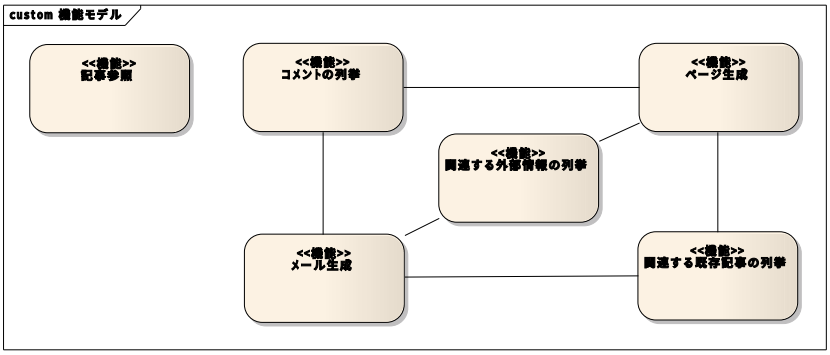

図 33:機能モデル

機能を中心に、画面、ユースケース、データ、ドメインなどを結び付けた機能複合モデルという関係ダイヤグラムを作成することで、全体の整合性をチェックすることができます。例を示します。

次の図では、各機能とユースケースやドメインとの関係が表されています。ユースケースの代わりにイベントをつなげる場合もあります。

モデリングの手引き

機能複合モデルでは、機能を中心にその機能がどのようにシステム境界（ユースケース、イベント）から使われ、そしてどのようにデータと関わるのかを表現します。データとの関わりにおいては、関連名として「CRUD」を記述するのが効果的です。

画面、ユースケース、機能、データまでをつなげることで、画面からデータまでのつながりを通してその機能の役割を認識します。また、項目レベルでは画面項目はデータ項目で満たされているかを確認します。

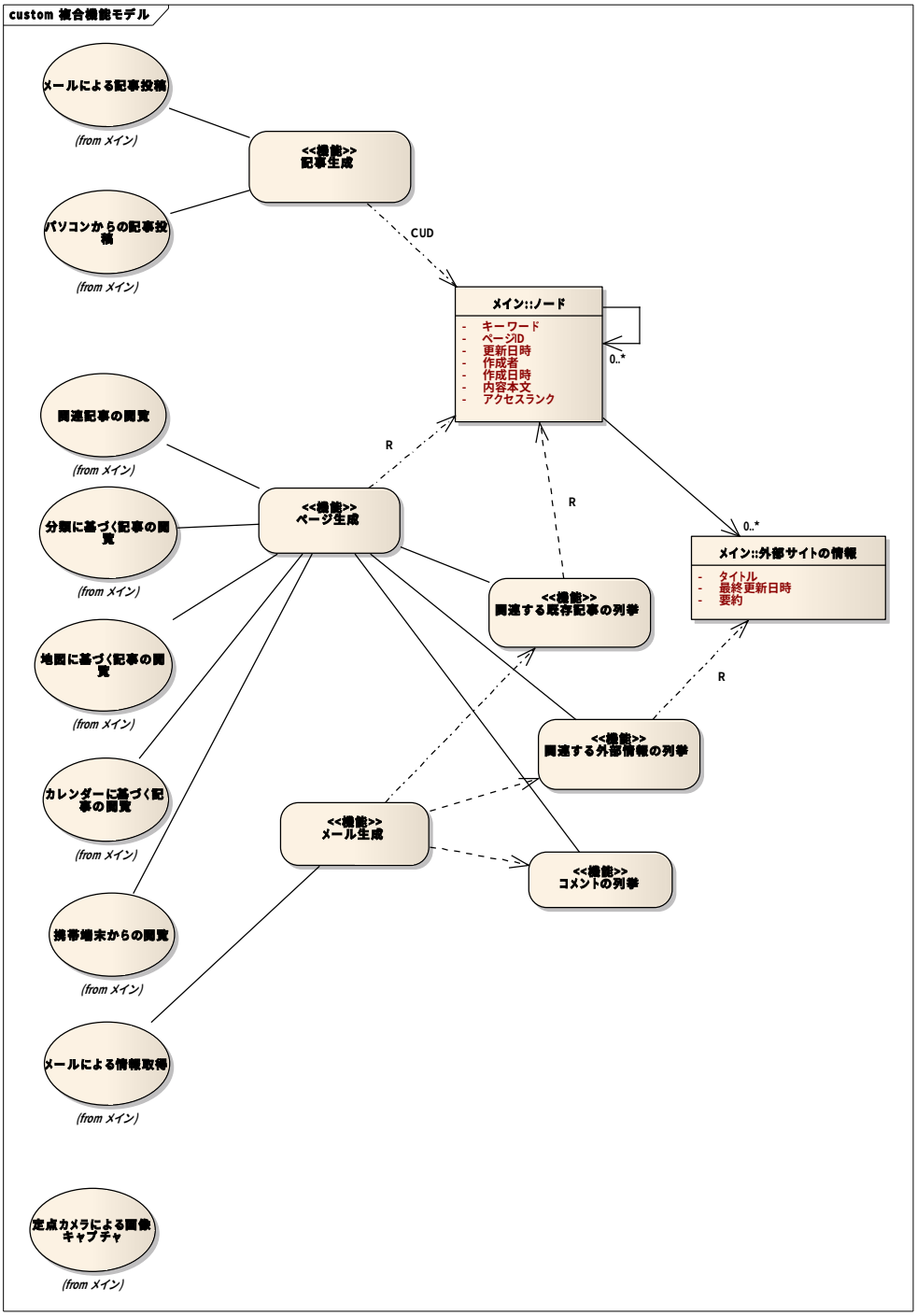

図 34:機能複合モデル

機能を捉えるポイント

システムがそもそもどのようなソフトウェア的な価値をもつかを認識することは、その内部構造を考える出発点になります。

5-5-3. ビジネスルール

目的

状態を捉えることによって複雑なビジネスルールを明確に定義できます。ビジネスルールは状態マシン図を使用して表現します。状態マシン図は網羅的に抜けのないようにルールを記述できる優れた表現力をもつ手段です。ルールとして明示しておきたいことがある場合は、対象を状態として捉えられるか検討し、その状態を識別できる場合は状態マシン図に表現しておくことでルールを記述できます。

状態マシン図は、状態や状態遷移のほかに、遷移のきっかけとなるイベントも識別します。対象の状態とイベントが識別できる場合は、このイベントが発生したときの扱いとしてルールを記述することができます。

モデリングの手引き

ビジネスルールの識別に際しては、ビジネスルールを実現するための処理手順や手続きであるソフトウェア上のルールとビジネスルールとを混同しないように注意します。

サンプル

題材システムが外部サイトにアクセスする際の認証に伴う状態を識別するための状態マシン図を示します。

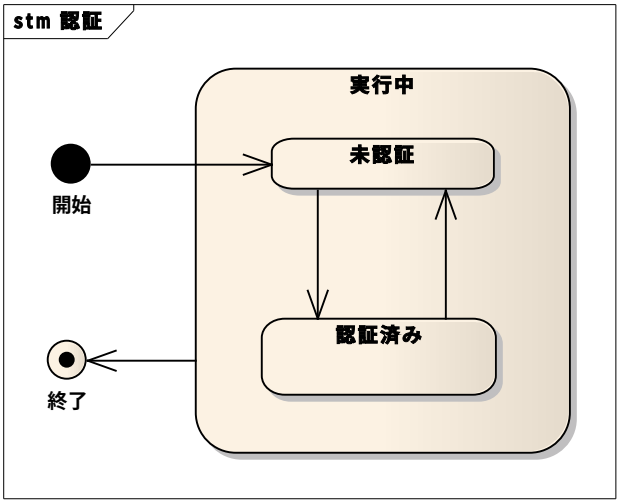

図 35:状態遷移で表現したビジネスルール

6. 要件定義の整合性を確保するには

この手法で使用する各ダイヤグラムは、必ず他のダイヤグラムと関係付けられており、その関係性をチェックすることで整合性をチェックできるようにしてあります。

6-1. 網羅的に要件定義を行う基本的な考え方

この手法におけるモデリングでは、以下に示す要件定義のための考慮点をモデルとして表現する必要があります。これらの視点を押さえながら要件定義としてのモデリングを行うことで網羅的で整合性の取れた要件定義が可能になります。以下に、その 4 つの切り口から網羅性と整合性を確保するための考え方を示します。

視点	問いかけ	整合性の根拠
業務の境界線を理解する	何をもって業務の境界線と考えるか？	・登場人物が全て出ている レビューなどで関係者に確認を取る ・その登場人物が行う作業が明確になっている
	どのようにして業務が正しいと判断するのか？	・業務の結果価値を作り出していることを確認 要求と突き合わせる ・PDCA の枠組みを使って業務のサイクルを検証する 業務のまとまり毎に PDCA が行われていることを確認する
システム化対象を明確にする	何をもってシステム化対象を網羅したと考えるか	・業務につながるユースケースを全て捉えている ・そのユースケースに必要な入出力を捉えている ・その IO で業務が回ることを確認している ・全ての外部コンポーネントの全イベントを捉えている
	どのようにして、そのユースケースが正しいことを確認するのか？	・要求と突き合わせて実現されていることを検証する ・ユーザーが理解（検証）できるモデルとして表現し、直接ユーザーに確認を依頼する 業務フローと突き合わせて説明することで、業務視点からユースケースの正しさを検証する
	外部イベントを正しく処理しているとどのように確認するのか？	・プロトコルモデルとイベントモデルの対応関係に基づく
	全てのユースケース、外部イベントを整合を合わせて洗い出したとどのように考えるか？	・ユースケースからの機能をイベントととらえて、プロトコルモデルで整合性を確認する ・すべての外部イベントを状態遷移に対応づける
機能を把握する	機能はどのように把握するのか？	・ユースケースに必要な機能を洗い出す ・イベント毎に必要な機能を洗い出す
	全て網羅したと判断できる根拠は何か？	・ユースケースが満たすソフトウェア機能を洗い出した ・外部イベントにつながるイベントを全て捉えた ・データと突き合わせて、データ視点から必要な機能をすべて洗い出した
	洗い出した機能が正しいとどのように確認するのか？	・データと突き合わせてデータの整合性を確認する ・全てのイベントをプロトコルモデルで表現し、状態との整合性を確認する
データを把握する	データとして把握すべき事は何か？	・構造を理解する ・各々のデータの項目を理解する ・ライフサイクルを把握する ・データ量を把握する
	データを把握する必要がある理由は何か？	・各機能間の整合性を確認するため 最終アウトプットに必要なデータがあること 画面・帳表に必要な機能があること

表 1

6-2. 整合性をチェックするポイント

要求からシステムの要件へとトレーサビリティを持って情報がつながっているかどうかに基づいて整合性を確認します。そのために、業務の境界を押さえ、システム化対象を明確にし、その実現のための機能とデータを明らかにします。

6-1 節で挙げた基本的な考え方をモデルに当てはめて、モデル別のチェックポイントを表2に示します。なお、ここに挙げたものをすべて関係づける必要はなく、対象に応じて必要なもののみを選択します。

視点	対象	意味	チェック方法
要求モデル	その他オブジェクト	要求が他のオブジェクトにつながっているとその要求を満たしていると考えられる	どこにもつながっていない要求はその要求が考慮されていない可能性がある。
コンテキスト	ユースケース	アクターが関わるユースケースの確認	ユースケースに関わらないアクターがある場合はユースケースに抜けがある可能性がある。
	利用シーン	アクターに関わる利用シーンの確認	利用シーンに関わらないアクターがある場合は利用シーンに抜けがある可能性がある。
	イベント	イベントは全て外部システムから発生すると考える	外部システムに関わらないイベントは不正確な可能性がある。
	業務フロー	アクターに対応するレーン	アクターがレーンとして出ていない場合はアクティビティが抜けている可能性がある。
利用シーン	ユースケース	利用シーンはユースケースを使って実現される	利用シーンに結びつくユースケースがない場合はユースケースが抜けている可能性がある。
ユースケース	画面・帳表、機能	ユースケースと画面の関係	ユースケースは画面か帳表、もしくは機能、データのどれかと結びついているはず。
	業務フロー 利用シーン	ユースケースは必ずアクティビティもしくは利用シーンと結びつく	ユースケースに関わるアクティビティがない場合はアクティビティが抜けている可能性がある。 ユースケースに関わる利用シーンがない場合は利用シーンが抜けている可能性がある。
イベント	機能モデル	イベントに結びつくアクションは機能と考える	アクションは機能として実現されるので、アクションに関わる機能がない場合は機能が抜けている可能性がある。
	プロトコルモデル	イベントはプロトコルモデルの遷移として使われる	遷移に使われていないイベントがある場合は状態の洗い出しが抜けている可能性がある。
画面	ユースケース	画面はユースケースで使われる	画面に結びつくユースケースがない場合は場合はユースケース抜けている可能性がある。
機能モデル	ユースケース、イベント、データ	実現するソフトウェアの機能として関係するものがつながっているかを目で確認する	ダイアグラムを目でチェックする。 ・画面の項目に対応するデータの存在 ・データの関係 (CRUD) は正確か？ ・ユースケースやイベントとの対応関係
データモデル	機能モデル	データを扱う機能を確認する	データ毎に CRUD が実現する機能を確認する。Create、Refarence、Update、Delete が各々満たされていないとデータのライフサイクルが完結しない可能性がある。
ドメインモデル	機能モデル	ドメインクラスを扱う機能を確認する	ドメインクラスに関わる機能がない時な機能が抜けている可能性がある。

表 2

7. まとめ

この資料では、当社が提唱するリレーションシップ駆動要件分析（RDRA）によるモデリングの手順と基本的な考え方を説明しました。

まず、要件定義で記述すべき事項をリストアップした後、それらを洗い出してドキュメントにまとめるための具体的な手段を提供する、リレーションシップ駆動要件分析（RDRA）のプロセスについて説明しました。

次に、この手法で実際に作成するダイヤグラムを示し、それぞれの手引きとサンプルを示しました。

最後に、こうして洗い出したモデルの間の整合性を確保する方法として、各基本ダイヤグラム間のトレーサビリティチェックの方法について説明しました。

この資料では、基本的な手順を中心に RDRA の説明をしました。この手法を実際のプロジェクトに適用する際のヒントやテクニックなどについては、さらに「実施にあたって」の資料を参照してください。