

静的モデル・ハーネス — ドメインの構造を記述する

2026-10-05
SimpleModeling

静的モデル・ハーネスは、ドメインの概念・関係・規則をモデルとして記述し、検査と実行へ接続する基盤です。本連載の第3回では、ソフトウェア工学で実用化された構成方法を活用し、AIによる生成・変更にも業務上の制約を働かせる仕組みを説明します。

1 静的モデル・ハーネスの位置付け

1.1 シリーズにおける位置付け

第1回ではAI開発ハーネスの全体像を、第2回ではモデル駆動開発を基盤とするモデル・ハーネスを扱いました。第3回はその静的な側面として、ソフトウェアを構成する概念、型、同一性、関係、所有、制約を記述します。第4回では、同じ対象のイベント、状態遷移、ワークフローへ進みます。

静的モデルと動的モデルは、同じモデルを構造と振る舞いから捉える二つの側面です。ドメイン・モデルとアプリケーション・モデルを分ける目的・関心の軸とは異なります。ドメインにも振る舞いがあり、アプリケーションにも静的な構造があります。

1.2 前回シリーズの静的モデルを引き継ぐ

前回シリーズの「モデルの構造とビュー」と「構造基盤としてのオブジェクトモデリング」では、概念、型、関係、責務、制約を形式的に記述し、目的に応じたビューから確認する基盤を示しました。今回の静的モデル・ハーネスは、その構造記述を検査と実行へ接続し、AIによる生成・変更に継続的な制約を働かせるものです。

オブジェクト・モデルは形式的な構造を、文芸モデルは意図や判断を文章として、知識モデルは意味関係と根拠を保持します。BoKで管理する用語集の名称、識別子、定義を共有することで、三つのモデルの対応を保ちます。クラス図などのビューは、元のモデルから目的に合った構造を選んで示します。意味の正本はモデルと用語集です。

1.3 ドメイン知識に近い抽象度で記述する

モデル駆動開発では、ソフトウェアをドメイン知識とのインピーダンス・ミスマッチが少ない形で記述します。注文、注文明細、商品、金額といった概念と、その関係や規則を直接表すことで、プログラミング言語の実装詳細へ展開する前に、業務としての意味を確認できます。

SimpleModelingでは、CMLで記述した実行可能モデルをTextusが直接実行します。モデルだけでは不足する実装はAIがプログラミング言語で自動補完するため、開発者はモデルによる記述を中心に作業できます。この分担によって、モデルは開発者にとって事実上のプログラミング言語になります。補完した実装もモデルの契約に従う必要があります。

2 静的モデルで記述する構造

業務の対象をソフトウェアとして扱うには、対象の同一性、共通する性質、他の対象との関係を定める必要があります。静的モデルは、これらを型、分類、所有、多重度などの形で記述します。

2.1 永続する対象と値を区別する

SimpleModeling では、エンティティはプログラムのライフサイクルを超えて、同じ対象として存在し続ける永続オブジェクトです。状態が変化しても、その対象としての同一性を保ちます。注文は実行が終了した後も存続し、次の実行で配送先を変更しても同じ注文として扱います。この同一性をモデルに記述し、生成・変更した実装を検査する基準にします。

値オブジェクトは、金額や期間などの値を表現します。同じ値であれば同等に扱い、個々のオブジェクトが同じ対象として存続しているかは問いません。金額を数値と通貨の組として記述すれば、値の同等性と演算の条件を明示できます。型と値の制約をモデルに置き、具体的な値の検証を必要な実行境界へ接続します。

2.2 分類・構造・その他の関係を見渡す

オブジェクト・モデルの静的構造における関係は、分類、構造（所有）、その他の関係を基本に整理します。分類は is-a 関係、構造は has-a 関係に対応します。この全体像を踏まえて、個々の関係に役割、多重度、所有やライフサイクルの条件を記述します。

分類では汎化を基本に、パワータイプとトレイトで分類や共通性の表現を補います。全体と部分の構造では集約関係と合成関係を使い、共有と所有を区別します。それ以外の対象間の結び付きは、関連関係として役割を明示します。トレイトの合成は共通性を補う仕組みとして扱い、適用を一律に is-a 関係とみなすことは避けます。

2.3 分類と共通の性質を明示する

分類と共通性の記述では、汎化を基本にします。共通する性質を上位概念にまとめ、下位概念で具体化します。SimpleModeling ではプログラミング言語との相性を考慮し、汎化の階層に単一継承を採用しています。多重継承が担う機能の一部を、パワータイプによる分類軸の表現と、トレイトによる共通の部分モデルの合成で補うモデル構成です。

パワータイプは、顧客種別や商品カテゴリのような分類軸と、その軸で使用する区分を表します。単一継承による基本の分類に加えて、複数の観点から対象を分類するために利用します。分類軸ごとの意味をモデルに置き、区分の追加や実装への対応を設計します。

トレイトは、汎化の階層を横断する共通の特徴、責務、制約などを、再利用可能な部分モデルとして定義します。複数のクラスへこの部分モデルを合成し、多重継承が担う共通性の表現の一部を補います。プログラミング言語の trait や interface への対応は、実現方針として定めます。

2.4 分類と全体・部分から業務の構造を捉える

業務の対象を理解するとき、「何の一種か」「何から成り立つか」は基本となる問いです。前者は対象の共通性と違いを、後者は対象を構成するまとまりを明らかにします。オブジェクト・モデルでは、この二つを is-a 関係と has-a 関係として重視します。

例えば、注文は複数の注文明細から成り立ちます。注文と明細を一つのまとまりとして捉えることで、明細を追加する、数量を変える、合計金額を求めるといった業務の操作を、その構造に沿って考えられます。何を一つの対象として扱い、その内側に何を含めるかという判断が、モデルの骨格を作ります。

一方、明細が指す商品は、注文を構成する部分とは役割が異なります。商品は注文とは独立して管理され、多くの注文から参照されます。対象の内側に持つものと、独立した対象として関わるものを分けることで、変更や管理の責任も明確になります。

静的モデル・ハーネスは、こうして捉えた分類やまとまりをモデルに残し、AI による実装や変更にも、その構造を保つ基準を与えます。

2.5 集約モデル・ビューモデル・表示モデル

SimpleModeling では、エンティティや値オブジェクトを基礎に、集約モデル、ビューモデル、表示モデルを用意しています。業務上のまとまりをどう構成するか、その情報をどのような形で利用するか、利用者にどう提示するかを、それぞれ明示的なモデルとして記述します。

集約モデルは、関連するエンティティや値オブジェクトを、業務上の整合性を保つ一つのまとまりとして表すモデルです。中心となるルートと構成要素を定め、そのまとまりの状態、操作、不変条件を記述します。例えば注文では、注文をルートとして注文明細をまとめ、明細の数量や金額と注文全体の合計が整合するようにします。個々のオブジェクトの関係に加えて、まとまり全体として守る規則と振る舞いを表現できます。

ビューモデルは、エンティティなどが持つ情報を、利用目的に合わせた構造へ投影するモデルです。必要な属性や関連を選び、業務上の意味を保ちながら、利用する側に適した情報のまとまりを作ります。例えば注文の概要には注文番号、顧客名、合計金額を、詳細には注文明細や配送先も含めます。元のモデルと各ビューの対応を記述することで、用途ごとに必要な情報の形を定められます。

表示モデルは、ビューモデルの情報を、利用者へ提示するための構造として表すモデルです。一覧、詳細、項目のまとまり、見出し、表示する値、操作の入口などを記述します。例えば注文番号を一覧項目の見出しにし、合計金額を表示項目に、注文の詳細を開く操作を入口として構成します。業務上の項目を画面上の役割に対応付けることで、表示を担う仕組みへ渡せます。

注文という業務上のまとまりを集約モデルで捉え、注文概要や注文詳細という情報の形をビューモデルで作成し、それを一覧や詳細画面として提示する構造を表示モデルで定めます。三つのモデルとその対応を明示することで、AI による生成・変更でも、業務の規則、情報の意味、画面上の役割を保つ基準が得られます。

2.6 用途に応じたステレオタイプでエンティティを分類する

エンティティ・オブジェクトは、業務で扱うドメイン・オブジェクトを記述するために用います。ドメイン・オブジェクトには、用途や性質に応じた構造と振る舞いがあります。SimpleModeling では、その特徴をステレオタイプとして定義します。

資源を表す Resource、処理の単位を表す Task は、エンティティ・オブジェクトをさらに分類するステレオタイプです。対象に合ったステレオタイプを選ぶことで、その用途に適した構造と振る舞いをモデルに取り込めます。資源には所有・共有・利用の条件、処理には入力・結果・必要な資源などの契約を記述し、実行方針を決める際に参照する情報を保持します。

静的モデルは、処理が利用する資源や入出力の関係を定めます。処理の開始順序、待機、再試行、終了条件は、動的モデルで記述します。

2.7 注文モデルで構造上の条件を確認する

注文識別子を持つ注文が、1 件以上の注文明細を合成関係で所有し、各明細が一つの商品を参照するモデルを例にします。明細は同時に一つの注文にだけ属し、商品は複数の明細から参照できます。金額は数値と通貨を持つ値オブジェクトです。

この構造では、各明細の所有者は一つの注文に定まります。同じ明細を二つの注文が同時に所有することは制約違反になります。一方、商品は複数の明細から参照できます。所有する部分と、独立した対象への参照を区別することで、注文のまとまりを明確にします。

3 モデルをハーネスとして機能させる

実用化された概念・部品・パターンをモデルの枠組みとして活用し、その規則を処理系の検査と実行へ接続します。AI による生成・変更に制約を働かせ、静的検査、実行時検査、意味のレビューを組み合わせて、ユースケースとの対応を確認します。

3.1 モデルの検査と実行を開発工程に組み込む

エンタープライズ・アプリケーション分野では、ソフトウェア工学の長い蓄積を通じて、多くの概念、部品、パターンが実用化されてきました。SimpleModeling は、これらをモデルとして記述し、そのまま実行できる基盤を提供します。

CML はモデルの要素と構造を文法として定めます。処理系はその記述を読み、対応する検査を行い、Textus は実行可能モデルを直接実行します。AI が生成・変更したモデルと補完実装をこの経路に通すことで、モデルの契約に反する成果物を開発・実行の境界で検出できるようにします。この仕組みが、モデル駆動開発の基盤を AI 活用のハーネスとして機能させます。

このハーネスの枠組みを補助線にすることで、実績ある概念と構成方法に沿って、整合性のあるオブジェクト・モデルを構築できます。AI も、その枠組みの中でモデルの記述や実装の補完を進めるため、既存の仕組みで解決できる部分に独自の構造や処理を持ち込むことを抑えられます。

ハーネスとして制約が働くためには、対象の規則がモデルに表現され、その規則を扱う検査または

実行機構が接続されている必要があります。ビルド時のモデル検査と実行境界の検査を工程に組み込み、違反の解消を次工程へ進む条件にします。制約が実際に働く範囲は、接続した検査・実行機構が対応する規則の範囲です。

3.2 静的検査・実行時検査・AI レビューを組み合わせる

モデル定義の型や参照先、関係の宣言など、実行前に確かめられる条件を静的検査の対象にします。実データの値、存在する対象間の多重度、同時更新時の所有など、実行状態に依存する条件は実行時の検査や制御へ接続します。規則の性質に応じて、静的検査と実行時の検査を分担させます。

textus-cbd-support によるレビューは、実行可能モデルにも直接実行しないモデルにも適用し、ビルド時にモデル上の矛盾を検査するために利用できます。処理系が機械的に判定する条件と、AI が意味関係から指摘する不整合候補を区別して結果を確認します。モデルが業務の事実と目的に合っているかは、人間による判断も必要です。

3.3 ユースケースと静的モデルの整合を確かめる

自然言語のユースケースを人間が読み、設計や実装の手掛かりにする運用に加えて、CML による文法定義と AI の活用により、ユースケースもモデル検査の対象になります。textus-cbd-support のレビューでは、シナリオが参照する概念、関係、責務と静的モデルの対応を確認できます。実行できるかどうかと、検査の対象にできるかどうかは別の性質です。

注文を確定するユースケースなら、対象の注文を特定できること、明細と商品の関係が定義されていること、金額の意味が共通であることを確認します。シナリオが明細の共有を前提とし、静的モデルが排他的な所有を定めていれば、両者を調整する必要があります。要求との追跡性は、記述した構造が何を実現するためのものかを確認する役割を持ちます。

4 モデルを実装と変更管理に活用する

静的モデルに記述した同一性や所有関係は、Textus がオブジェクトを保存・復元するときや、更新範囲を定めるときに利用できます。また、用語を通じて関連する文書や知識との対応を保持することで、モデルを変更した際に、見直すべき補完実装、検査、業務上の説明をたどれます。

4.1 静的モデルを永続オブジェクトへ接続する

静的モデルでは、記述した対象を永続オブジェクトとして扱えることが重要です。同一性、属性、関係、所有構造を保存・復元・更新へ接続し、処理を終えた後にも同じ対象を継続して扱えるようにします。

リレーショナルデータベースを利用する場合、オブジェクトとテーブル、属性と列、識別子とキー、オブジェクト間の関係と参照の対応を定める OR マッピングを使用します。SimpleModeling では Textus が OR マッピングを自動化し、静的モデルの記述を永続化へ接続します。汎化や所有の表現も、この対応を考える際の基礎になります。

注文の例では、注文識別子による同一性を保って、注文と明細を保存・復元します。集約モデルの

整合性境界と、永続化時の更新単位を対応付けます。履歴保存、削除、並行更新などの業務規則を明示し、採用する保存方式や処理系の対応範囲に合わせて実現します。

4.2 モデルの意味を実行方針へ接続する

モデル要素の分類、同一性、所有、関係は、処理系が利用する実行可能なメタデータにもなります。エンティティの同一性を永続化やキャッシュのキーへ、所有の境界を更新やロックの単位へ、資源と処理の契約をスケジューリングや資源割当てへ接続する際の判断材料にできます。モデルの意味と実行方針の対応を定義することが前提です。

分割配置、並行性、観測にもモデルの識別子や境界を利用できます。永続化方式やロック方式は、性能、信頼性、セキュリティなどの要求を踏まえて選び、処理系が対応する範囲で実現します。モデルの意味を方針の選択と適用へ結び付けることが、この接続の役割です。

4.3 用語を介して知識モデル・文芸モデルと連携する

静的モデルの概念や関係は、用語を媒介に知識モデルと文芸モデルへ連携します。用語集の名称、定義、識別子を共有し、モデル要素と、業務の意味や判断の根拠を説明する文書を対応付けます。例えば「注文」の成立条件を文書で説明し、同じ用語を注文の型、属性、関係へ結び付けます。

知識モデルでは、用語に対応する RDF ノードを通じて文書とオブジェクト・モデルの意味関係を保持します。用語やモデルの変更時には、関連する文書、構造、知識の対応をたどり、整合を確認できます。実際のガードは、用語の参照とモデル間の対応を検査するツールを開発工程に接続して実現します。

ビジネス・モデル、要求モデル、ドメイン知識のうち文章で表す内容も、文芸モデルとして静的モデルにつなぎ、開発中に利用し続けます。この連携を基盤に、知識化したオブジェクト・モデルと文芸モデルを MCP で生成 AI へ提供することを当面の目標とします。その提供環境と知識ハーネスの詳細は第 8 回で扱います。

4.4 モデルの変更を実装と検査へ反映する

静的モデルを変更するときは、対象の意味と規則を確認し、CML の記述、AI が補完する実装、検査を対応させます。例えば金額に通貨を追加する変更は、属性の追加だけでなく、値の同等性、加算規則、保存形式、ユースケース上の前提にも影響します。モデルの差分を基準に影響範囲を確認し、ビルド時のモデル検査と実行時の検証へ進みます。

レビューでは、同じ意味に読めるかという明確性、定義が衝突しないかという一貫性、目的に必要な内容があるかという妥当性、現実の事実や規則に合うかという正確性を確認します。AI は候補の作成と不整合の検出を支援し、人間は同一性、所有、ライフサイクルなどの業務上の判断を行います。

5 まとめと動的モデルへの接続

静的モデル・ハーネスは、ソフトウェア工学の蓄積から実用化された概念・部品・パターンを、検査・実行できるモデルの枠組みとして活用します。型、同一性、分類、所有、関係、多重度を明示し、

実績ある構成方法に沿ってオブジェクト・モデルを構築します。AI による生成・変更もこの枠組みに通すことで、契約違反を検出し、不要な独自構造や処理の持ち込みを抑えます。

第 4 回では、ここで定めた対象がどのイベントを受け、どの条件で状態を変えるかを状態機械とワークフローで記述します。第 5 回以降はアーキテクチャ、プログラミング、ランタイムへの具体化へ進みます。用語集を介したモデル間の意味の連携を踏まえ、共通知識とプロジェクト固有の知識を生成 AI へ提供する知識ハーネスは第 8 回で扱います。