ランタイム・ハーネス

2026-11-02
SimpleModeling

SimpleModeling では、モデル構築・プログラミング・実行にハーネスを重ね、AI の逸脱を抑え、適切な実装へ導きます。ランタイム・ハーネスは、モデルと部品の契約を実行時に適用する仕組みです。Textus/CNCF の通常の作法に沿って実装することで、品質を支える共通機構の恩恵を受けられます。

1　ランタイム・ハーネスとは

プログラムを動かす実行基盤（ランタイム）が、使える操作とその実行方法を管理し、動作を定めた規則の範囲に保つ仕組みを、本連載では「ランタイム・ハーネス」と呼びます。

本連載では、AI ハーネスの役割を、望ましくない行動を抑える「制約」、適切な操作や実装へ導く「誘導」、検証結果を修正へ返す「フィードバック」として捉えます。これらを組み合わせ、AI が目的に沿った結果へ収束することを支えます。

ハーネスの役割と実行基盤との関係

役割	実行基盤との関係
制約	モデルの規則・部品の契約・権限に適合しない操作を制限する。
誘導	モデル操作、API・SPI、UoW によって、業務を実現する操作と接続方法を明確にする。
フィードバック	実行結果と記録を検証に用い、問題を修正へ返す。ワークフローで再検証や停止の条件を定め、試行錯誤の範囲を管理する。

Textus/CNCF が備える認可、更新管理、状態遷移の管理などは、元々セキュリティや信頼性などの品質属性を達成するために実行基盤へ導入した仕組みです。SimpleModeling では、これらを AI のハーネスとしても活用します。共通の仕組みへ制御を任せることで、AI が個別に判断・実装する範囲を減らし、業務固有の処理に集中しやすくします。

1.1　ハーネス全体での位置付け

AI による開発では、モデル構築、プログラミング、実行の各段階にハーネスを重ねます。一つのハーネスでは制約できない範囲を他のハーネスで補い、AI の誤った判断や生成が不適切な動作につながることを抑えます。

モデル構築・プログラミング・実行に重ねるハーネス

段階とハーネス	制約する範囲
モデル構築：モデル・ハーネス（第 2〜4 回）	業務の構造・振る舞い・規則をモデルに記述し、モデルの文法や整合性を検査する。その定義を生成と実行にも用いる。
プログラミング：プログラミング・ハーネス（第 6 回）	型・合成規則・API で補完コードの構造と使える操作を制約し、モデルと実行基盤の作法に従う実装へ導く。
実行：ランタイム・ハーネス（第 7 回）	モデルと部品の契約に従って処理を動かし、実行時の権限・状態・更新・資源利用を管理する。

例えば、型や契約に適合するコードでも、呼出し元の権限や対象の現在の状態によっては操作を許可できません。ランタイム・ハーネスは、モデルと契約を実際の動作へ適用し、こうした実行時の条件も確認します。プログラミング・ハーネスが実行基盤の管理を受ける構造へ実装を導くことで、AI の補完コードにも共通の制御が働きます。

これらの役割と組合せを定めるのが、第 5 回のアーキテクチャ・ハーネスです。知識ハーネスは AI が判断に用いる前提を揃え、観測モデルと検証ハーネスは実際の振る舞いを要求と照合するための情報と確認手段を提供します。

1.2 実行時の構造と制約点

Textus/CNCF では、生成モデルと補完コードをコンポーネントの内部に置きます。部品どうしは、提供する操作を表す API と、必要な機能を接続する SPI で連携します。各部品のモデルと操作は、共通の実行管理によって扱われます。

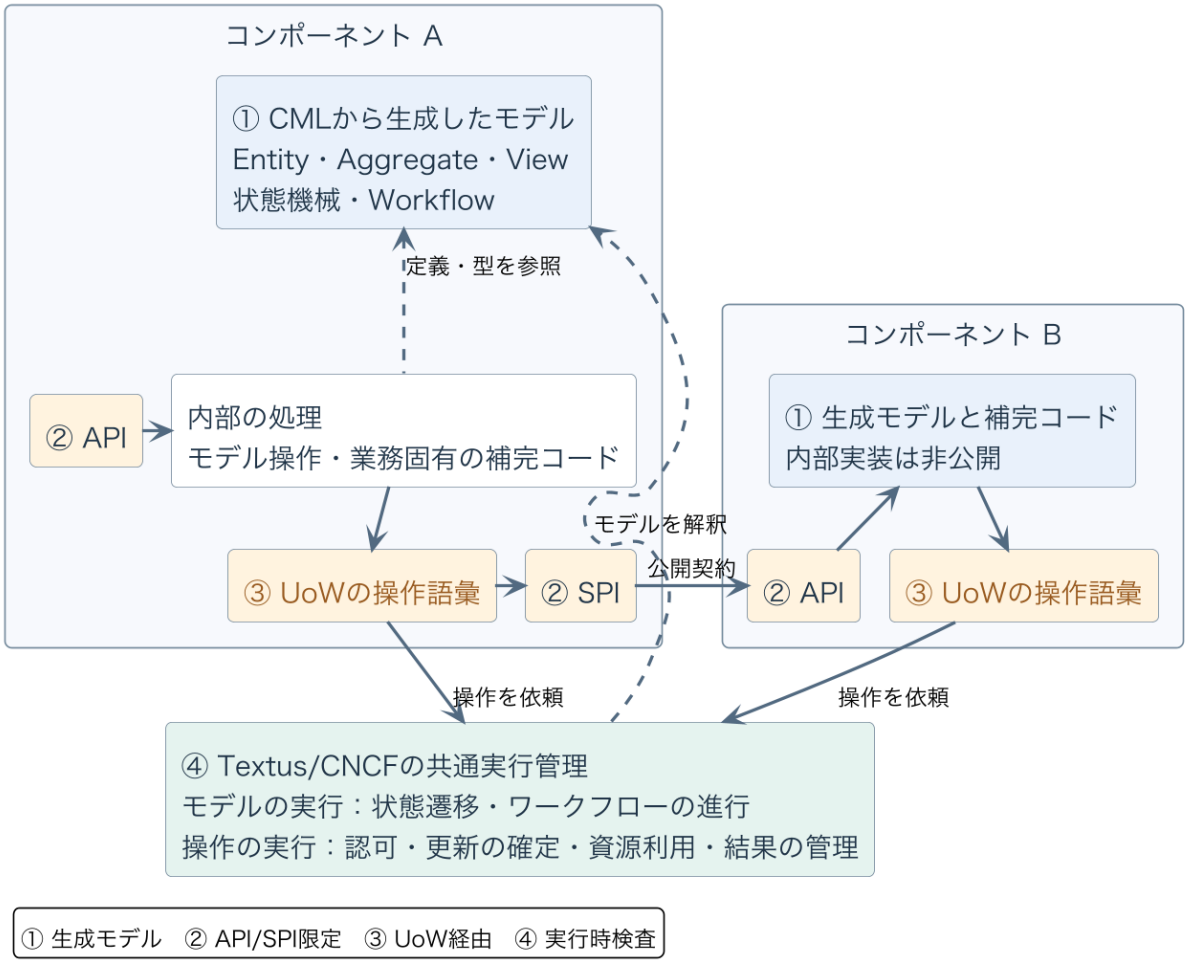


図 1 実行時の構造と制約点（B の内部は簡略化）。枠は部品の境界、実線は契約・操作の接続、破線はモデル参照を表します。①〜④は下表の制約点に対応します。

1.3　この構造がハーネスとして働く理由

ハーネスとして働くのは、使える接点や操作が構造によって限定され、その経路に実行時の検査と管理が組み込まれているためです。図の各制約点は、次の範囲を受け持ちます。

構造上の制約がハーネスとして働く理由

図の制約点	制約が働く理由
① モデルは CML から生成	コードと実行基盤が同じモデルの定義・型・操作を使うため、補完コード側に別のデータ構造や進行規則を作る範囲が限定される。
② 外部接続は API・SPI のみ	他の部品の非公開実装を利用側へ渡さず、公開された契約で接続するため、内部処理を直接呼ぶ実装を抑えられる。
③ 内部の作用は UoW 経由	保存・参照・外部呼出しを基盤が提供する操作で記述するため、個別コードが実行管理を迂回する範囲を限定できる。

④ 実行時に規則を適用	基盤が操作を実行する際に、認可、許される状態遷移、更新の確定、資源利用の規則を適用する。コードから要求されても、規則に適合しない操作は拒否される。

これらの制約は重なって働きます。開発者がモデルと部品の作法に沿って実装すると、AI の補完コードにも共通の認可や資源管理、状態・進行の管理が適用されます。品質対策を個別に組み込む負担を減らし、業務のモデルと処理に集中できます。

2　モデルに記述した規則を実行する

CML には、業務データとその操作を構造モデルとして、処理の進行を動的モデルとして記述します。Textus/CNCF はその定義を実行に用いるため、AI がモデルとは別にデータ管理や進行管理を作り直す範囲を減らせます。

2.1　CML から生成したモデルを実行の定義にする

Entity、Aggregate、View、Display などのモデルは CML から自動生成したものを使用し、開発者や AI が同じモデルを別に実装することはしません。Textus/CNCF は、生成された定義、型、操作と接続情報を受け取り、そのモデルに従って実行します。モデルの直接実行とは、モデルの意味を別の手続きへ人手で書き直さず、実行基盤が扱うことです。

モデルが提供する構造と操作をそのまま利用し、業務固有の計算や連携だけを所定の拡張点で補完します。モデルと補完コードが別々の構造を持つ箇所を減らし、モデルの変更を生成物と実行定義へ反映できます。

2.2　Entity を基盤に Aggregate と View を構成する

構造モデルでは、Entity を業務データの基盤とし、更新系を Aggregate、参照系を View で記述します。Aggregate は業務上の更新と不変条件を扱い、View は参照に必要な情報を構成します。View の外側には Display を配置し、参照結果を表示に適した形へ投影します。

例えば、注文の状態変更は Aggregate の業務操作、一覧に必要な情報は View、項目の表示は Display で扱います。表示の都合で業務状態を書き換えたり、一覧表示の処理に更新規則を重ねて実装したりする必要がありません。

Entity を基盤として更新・参照・表示を分担することで、AI も処理の置き場所を判断しやすくなります。対応するモデル操作を使うと、基盤が提供する保存・参照・表示の仕組みを利用でき、モデルの役割に沿う実装が実行時の管理にもつながります。

2.3　振る舞いと判断もモデルに従って実行する

Textus/CNCF では、状態機械を振る舞いの実行管理の基盤としています。状態・イベント・遷移条件をモデルに明示し、基盤が許された遷移を実行します。AI が状態管理や分岐を個別に実装する範囲が減り、条件の扱い漏れや不正な状態変更を抑えられます。

その上に、アプリケーション寄りの動的モデルとしてワークフローを構成します。開発者や AI は業務の順序や分岐を記述し、進行に伴う状態管理を基盤へ委ねます。業務に近い抽象度で処理を表しながら、状態機械による定型的な実行管理を利用でき、モデルに沿った安定した動作を得やすくなります。

ワークフローの各処理をコンポーネントの公開操作へ接続します。更新や参照を伴う操作では、Aggregate や View のモデル操作を使います。これにより、業務の進行とデータの操作が、それぞれのモデルと部品の契約に従って実行されます。

業務規則をルール・エンジンで評価する場合も、判断結果と実行する操作を分けます。評価から得た ActionPlan は、コンポーネントの公開 Operation や Event、Job の実行へ接続します。モデル化した判断を使いながら、部品の契約と基盤の実行管理も維持する構成です。

3　部品の接続と内部の操作を実行基盤が管理する

コンポーネントの外部との連携は API・SPI、内部のデータ操作や外部への作用は UoW の語彙を通します。どちらも実行基盤が管理する接点となるため、AI の補完コードも部品の契約と共通の制御に従います。

3.1　外部との接点を API と SPI に限定する

コンポーネントが外部と接点を持つのは、API と SPI を通してです。API は部品が提供する操作の契約を表し、SPI は部品が必要とする機能を提供者へ接続する契約を表します。相手の内部実装を呼び出したり、接続先の処理を利用側へ複製したりせず、定められた入出力と意味で連携します。

業務固有の外部連携も、この接点として実装します。通信方式や提供者の詳細を SPI の実装側へまとめることで、業務処理は同じ契約を利用できます。AI が変更する範囲が、契約に従う利用側か、契約を実現する提供側かに分かれます。

3.2　実行時にも公開契約と内部実装を分ける

CAR はモデルに対応する部品の契約、実装、依存関係をまとめます。利用側へ渡すのは公開契約の API であり、提供側の内部実装は利用側のライブラリーにしません。実行時にも契約 API の型を共有し、実装はコンポーネントごとに保持します。

基盤は宣言された依存関係と契約を使って部品を構成します。必要な API がない場合や契約が適合しない場合に、内部実装を探して代わりに接続することはしません。型と部品検査によって定めた利用範囲が、実行時の部品構成でも維持されます。

公開操作を使うことで、利用側は部品の内部実装に依存せずに処理を再利用できます。基盤はこの接続関係を実行時にも維持するため、部品の独立性を保ったまま共通の制御を適用できます。

3.3　内部の操作を UoW の語彙で記述する

コンポーネント内部では、保存、参照、外部呼出しなど、基盤が管理する操作を UoW の語彙で記述します。補完コードが実行方法を自由に選ぶのではなく、使える操作を定め、その実行を基盤へ渡

します。純粋な業務計算は、その入出力の間に組み込みます。

Free Monad で記述・合成した操作を実行基盤が解釈するときに、認可、更新の確定・中止、資源の終了管理を適用します。UoW は、生成モデルと API・SPI の契約に沿って構成した処理を、基盤が管理する作用へ接続します。

4　入力の検査とエラー処理を統一する

入力の検査と結果の返却も、コンポーネントごとの独自方式にしません。CNCF のプロパティで入力を扱い、Consequence で成功と失敗を表し、Conclusion で失敗の内容を揃えます。

4.1　プロパティとアプリカティブな合成で入力を扱う

パラメタは CNCF が提供するプロパティ経由で扱い、必要な値の取得と検査の結果を Consequence として組み合わせます。入力項目が互いに独立している場合はアプリカティブな合成を使い、各項目の結果を集約します。例えば、注文番号の不足と数量の不正を同じ入力の問題としてまとめて扱えます。

各項目の取得と検査を Consequence として合成するため、AI が独自のパラメタ解析やエラー収集の制御を組み立てる範囲が減ります。検査を通った入力を業務処理へ渡し、入力の問題は共通の結果体系で返すことで、検査と処理の接続を統一できます。

4.2　Consequence と Conclusion で結果と失敗を統一する

業務処理の成功と失敗は Consequence で扱い、失敗の内容は Conclusion で構造化します。入力の問題、業務条件の不成立、認可の拒否、実行時の失敗を共通の結果体系へ載せることで、部品ごとに独自の例外やエラー文字列の扱いを設計する範囲を減らします。

呼出し元への結果の伝達、診断、実行基盤による失敗の扱いに同じ情報を使います。AI が書く処理もこの結果体系に従って合成されるため、部品を組み合わせても、システム全体で一貫したエラー処理を行えます。

5　ハーネスに沿う実装で品質を支える

モデルと部品の標準操作には、認可、更新管理、資源管理、観測、障害対応が結び付いています。開発者が対策を一つずつ選んで実装しなくても、業務処理の実行とともにこれらの制御が働きます。それぞれが支える品質属性は次のように整理できます。

ハーネスの共通機構が支える品質属性

品質属性	ハーネスとの関係
データ整合性・正確性、信頼性、予測可能性・再現性	モデルの規則と状態遷移、共通の更新・結果管理に従って、振る舞いを揃える。
セキュリティ、観測可能性、監査可能性・追跡可能性	標準の実行経路で保護と記録を適用し、操作の主体・対象・結果を把握できるようにする。

性能・応答性、拡張性・弾力性	参照・表示のモデルと共通の実行管理を使い、負荷に応じた処理と容量の制御を利用する。
可用性・耐障害性、回復可能性	部品の境界と状態・進行の管理によって、障害の影響を限定し、復旧につなげる。
保守性・変更容易性、検証容易性、再利用性・組合せ容易性、相互運用性・接続先の変更容易性	型付き契約と業務・実行制御の分離によって、部品の変更、検証、再利用、外部接続を共通の構造で扱う。
導入・運用容易性、資源効率・運用コスト	標準設定と共通の資源管理を使い、用途に必要な構成と負担を調整する。

一つの共通機構が複数の品質に寄与し、複数の機構が同じ品質を支えます。メッセージによる処理の分離に、進行管理、容量制御、障害対応を組み合わせることで、応答性・耐障害性・弾力性・メッセージ駆動というリアクティブな四特性も支えます。

必要な処理量や可用性の水準は、用途に応じた配備と設定で調整します。

6　AI の作業をワークフローの管理下で実行する

AI 自身が業務や開発の実行に参加する場合にも、同じ構成を適用できます。担当する判断や生成をワークフロー内の処理として位置付け、AI は定められた入力・制約・完了条件に従って結果と根拠を返します。全体の順序と継続・再開は基盤が管理する分担です。

実行基盤は、Consequence ／ Conclusion による構造化された結果と実行記録を、検証へ渡す情報として提供します。検証側で要求や完了条件と照合し、問題の箇所と理由を AI へ返すことで、修正する範囲を絞れます。結果を伝える仕組みと、その結果が要求を満たすかを判定する検証を組み合わせます。

このフィードバックを収束へつなぐには、修正と再検証の手順、再試行の上限、停止や人による確認へ移る条件をワークフローに定めます。結果の型や実行管理だけで正解が得られるわけではありません。検証と継続判断を組み込むことで、修正を際限なく繰り返すことを抑え、必要に応じて方法を見直せます。

7　まとめ

ランタイム・ハーネスは、モデルと部品の契約を実行時に適用し、AI の逸脱を抑え、適切な操作へ導きます。実行結果と記録を検証・修正へつなぐことで、目的に沿った結果への収束も支えます。開発者は通常の作法に沿って業務を記述することで、安定した動作と品質の恩恵を受けられます。

次回の第 8 回では、AI が判断や生成に用いる知識を整える知識ハーネスを扱います。