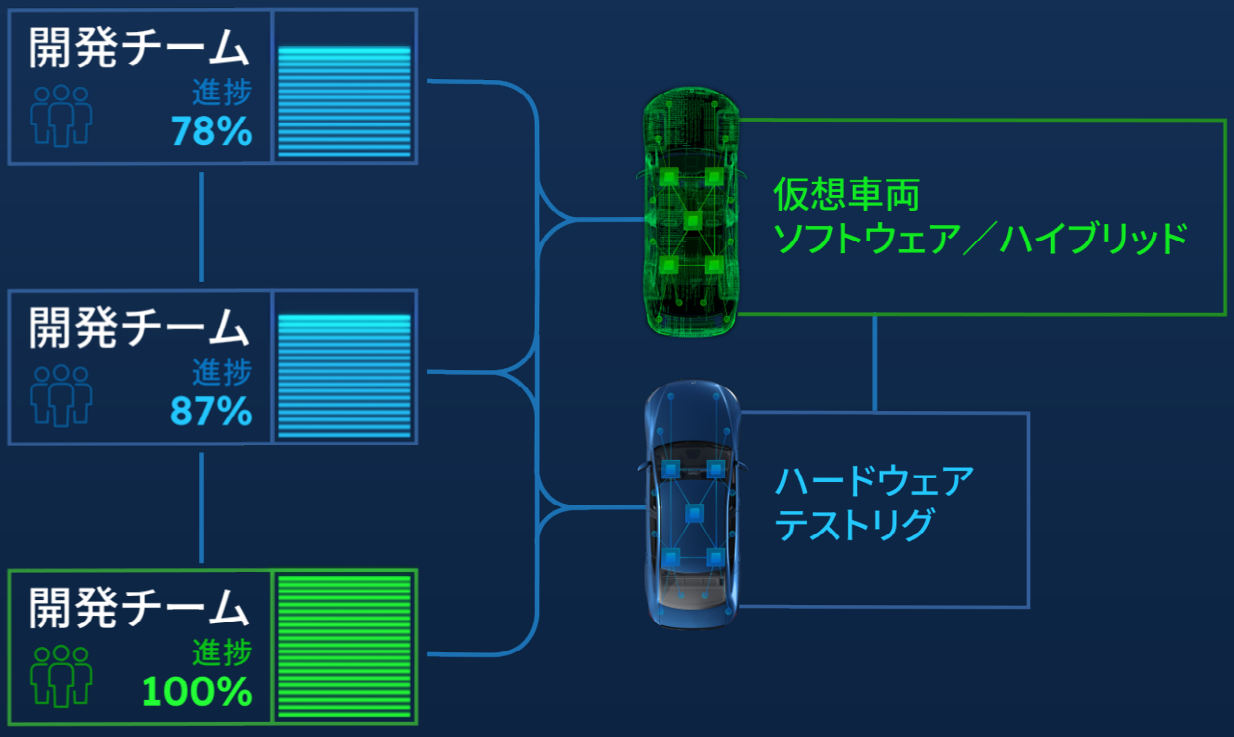
仮想車両開発によるシステム統合の革新



ホワイトペーパー

remotiveLabs

エグゼクティブサマリー

現代の車両におけるソフトウェアの急速な複雑化は、自動車システム開発のあり方を根本的に変えています。車両は、CAN（Controller Area Network）や車載イーサネットなどの異種ネットワークで相互接続された複数の ECU（電子制御ユニット）にまたがる分散型ソフトウェアアーキテクチャへの依存を強めています。ソフトウェア定義型車両（SDV）への移行が進むにつれ、ソフトウェアコンポーネントの数とその相互作用は増加の一途をたどっています。

従来の自動車開発プロセスでは、システム統合やテストにおいて物理的なハードウェアに大きく依存しています。ECU ソフトウェアは複数のチームやサプライヤーによって独立して開発されることが多く、システム統合はハードウェアが利用可能になって初めて行われます。このアプローチでは、統合課題の発見が遅れ、テスト環境の利用が制限され、開発中の迅速な反復が困難になるといった問題が頻繁に発生します。

E/E アーキテクチャそのものをソフトウェア上でインスタンス化し実行できる仮想環境を導入することで、チームはプログラム初日から――ECU ソフトウェアが存在しない研究・コンセプトフェーズから――設計の検証を開始できることが実証されています。先行開発から量産開発へとプログラムが進むにつれ、同じトポロジーに段階的に高忠実度のコンポーネントが追加され、システムレベルの統合テストは後工程の一回限りのイベントではなく、継続的な活動となります。このアプローチにより、開発チームは欠陥をより早期に検出し、分散チーム間の連携を改善し、限られたハードウェアリソースへの依存を軽減することができます。

RemotiveTopology は、AUTOSAR ARXML ファイルやネットワーク DBC ファイルなどの既存の車両設計成果物に基づいて仮想車両アーキテクチャ（virtual vehicle architecture）を構築・実行することで、自動車ソフトウェアのシステムレベルの仮想化を実現します。トポロジーは量産アーキテクチャの記述から直接生成されるため、仮想システムは量産車両とのアーキテクチャ的な一貫性を維持し、同一の量産通信（production communication）インターフェースとメッセージ定義を保持します。

これらの仮想車両トポロジー（virtual vehicle topology）内では、シンプルなモック（mock）から FMU（Functional Mock-up Unit）、Simulink モデル、さらには専用の vECU 実行プラットフォームで生成されたフル仮想 ECU（vECU）まで、さまざまな忠実度（fidelity）の ECU コンポーネントを、クラウドまたは開発者のワークステーション上のコンテナ化された環境で実行できます。RemotiveTopology は車載ネットワークの動作を再現する通信レイヤーを提供し、分散 ECU インスタンス間の信号交換を可能にします。

また、このプラットフォームはオープンインターフェース（open interfaces）を提供し、幅広いテスト・デバッグ・分析ツールとの統合を可能にします。この柔軟性により、開発チームは Pytest、Behave、Jupyter Notebook などのフレームワークを既存のワークフローに組み込むことができ、ベンダーロックイン（vendor lock-in）を回避しながら既存のエンジニアリングツールチェーンを活用できます。RemotiveStudio は、ブラウザ上で直接トポロジーの閲覧、信号データベースの検索、リアルタイム信号モニタリングを提供し、チームに稼働中のシステムに対する共有ビジュアルレイヤーを提供します。

RemotiveTopology はこれらの機能を Infrastructure-as-Code（IaC）として提供します。トポロジー定義、通信マッピング、テスト構成はバージョン管理されたテキスト成果物であり、マシン間で再現可能で、MCP API（Model Context Protocol）を通じて AI エージェントから直接アクセスできます。コンテナ化された実行環境と柔軟なツール統合と組み合わせることで、現代のソフトウェアエンジニアリングの手法に適合し、SDV の増大する複雑性に対応するスケーラブルな開発ワークフローの基盤を提供します。

1. はじめに：自動車ソフトウェア開発の進化

自動車業界は、これまでで最も大規模なソフトウェア変革の最中にあります。車両がSDV へと進化するにつれ、かつて固定されたハードウェアに組み込まれていた機能は、複数の ECU と複数の車載ネットワークにまたがる分散ソフトウェアによって提供――そして継続的に更新――されるようになっています。この変化は、車両が実現できることだけでなく、車両を開発する組織の働き方そのものを変えています。

本ホワイトペーパーでは、この変革がもたらす中核的なエンジニアリング課題――現代のソフトウェア開発が要求するスピードで、分散型車両ソフトウェアをシステムレベルで統合・検証・継続的にテストする方法――について考察します。従来のハードウェア中心の統合の限界を説明し、スケーラブルな代替手段としてシステムレベルの仮想化を紹介し、RemotiveTopology がこのアプローチを開発ライフサイクル全体――初期のコンセプトワークから量産開発、そして量産開始（SoP）後のソフトウェアアップデートまで――にわたってどのように支援するかを解説します。

2. ビッグバン統合から仮想開発へ

現代の車両は、数十の ECU にまたがるソフトウェアで構成されており、OEM チームや Tier1 サプライヤーが稼働中のシステムではなくインターフェース仕様に基づいて並行開発を行っています。各コンポーネントは個別に検証されますが、コンポーネント間の相互作用は、物理ハードウェア上でフル車両が組み立てられるまで検証できません――これは一般に「ビッグバン統合（big-bang integration）」と呼ばれるパターンです。

この後工程統合モデルは、自動車ソフトウェアプログラムにおける最もコストのかかる欠陥の根本原因です。インターフェースの不整合、タイミングおよびスケジューリングの競合、ネットワーク通信の設定ミス、起動時の依存関係の未充足――これらはすべて、独立して開発された ECU が初めて接続されたときに初めて表面化する傾向があります。その時点では、ハードウェアは不足し、複数のチームが同じ HIL（Hardware-in-the-Loop）ベンチやプロトタイプ車両へのアクセスを競い合い、すべての修正には組織の壁を越えた調整が必要となります。車両が SDV アーキテクチャ――サービス指向設計、集中型コンピュート、継続的なソフトウェアアップデート――へと移行するにつれ、検証すべき相互作用の数は、それをテストするための物理インフラの拡充を上回るペースで増加しています。

仮想開発環境は、システム統合をハードウェアの可用性から切り離すことで、この課題に対処します。仮想車両トポロジーは、ECU ソフトウェアが一行も書かれる前の研究・コンセプトフェーズにおいて、アーキテクチャファイルのみからインスタンス化できます。プログラムが先行開発から量産開発へと進むにつれ、個々のコンポーネントはより高忠実度の実装に段階的に置き換えられ、最終的には物理ハードウェアに置き換わります――その間、周囲のシステムトポロジーはそのまま維持されます。その結果、システムレベルの統合テストはプログラム初日から開始され、後工程のイベントではなく、継続的な活動として続きます。欠陥は、影響を受けるコンポーネントがまだ活発に開発されている段階――修正コストが最も低く、チーム間の調整がまだ容易な段階――で発見可能になります。

この現代的な開発手法への移行については、以下のオンラインビデオでもご紹介しています：https://www.youtube.com/watch?v=a29vfjl0bro

2.1 自動車ソフトウェア開発における仮想化

仮想化とは、ECU ソフトウェアをターゲットハードウェアプラットフォーム上ではなく、ホストコンピューティング環境内で実行する能力を指します。この文脈では、ECU ソフトウェアは実際の ECU ソフトウェアスタックの動作をエミュレートする vECU として実行される場合があります。

一般的な vECU には以下が含まれます：

» アプリケーションソフトウェア

- ミドルウェアコンポーネント（AUTOSAR サービスや Android Automotive OS など）
- 通信スタック
- 他の ECU とのソフトウェアインターフェース

物理的な車載ネットワークやハードウェアペリフェラルと直接やり取りする代わりに、vECU は仮想化環境が提供するシミュレートされたインターフェースを通じて通信します。

これにより、複数の ECU ソフトウェアコンポーネントを単一の開発ワークステーション、サーバー、またはクラウド環境内で同時に実行することが可能になります。

2.2 より早期のシステム統合の実現

仮想開発の主要な利点の一つは、開発ライフサイクルのより早い段階で統合テストを実施できることです。

物理的な ECU が統合ベンチに組み込まれるのを待つ代わりに、ソフトウェアコンポーネントを仮想システム環境内で接続・実行し、さまざまなシミュレーションレベルや仮想・物理ノードを利用可能な範囲で適宜組み合わせることができます。これにより、エンジニアは開発が進行中の段階で、異なる ECU 間の相互作用を観察し、潜在的な統合課題を特定することが可能になります。

早期統合により、チームは以下を検証できます：

- ECU 間の通信インターフェース
- サービス指向アーキテクチャ内のサービス相互作用
- 車載ネットワーク全体のデータフロー
- 起動および初期化シーケンス
- 分散型機能のエンドツーエンド動作

仮想システムは迅速に作成・変更できるため、エンジニアは課題の調査や新しい構成のテストにおいて素早く反復できます。

2.3 並行開発とコラボレーション

仮想開発環境は、開発チーム間のコラボレーションも改善します。

従来のワークフローでは、システムレベルのテストは限られた数のハードウェア統合ベンチに依存することが多く、複数のチーム間で共有する必要があるため、アクセスが制限され、開発速度が低下する可能性があります。

一方、仮想環境は容易に複製できます。複数の開発チームが同じ仮想車両アーキテクチャの独立したインスタンスを同時に実行できます。

これにより、以下が可能になります：

- 複数チームによる並行統合テスト
- 共有ハードウェアリソースに影響を与えない独立した実験
- より高速なデバッグサイクル
- より柔軟な開発ワークフロー

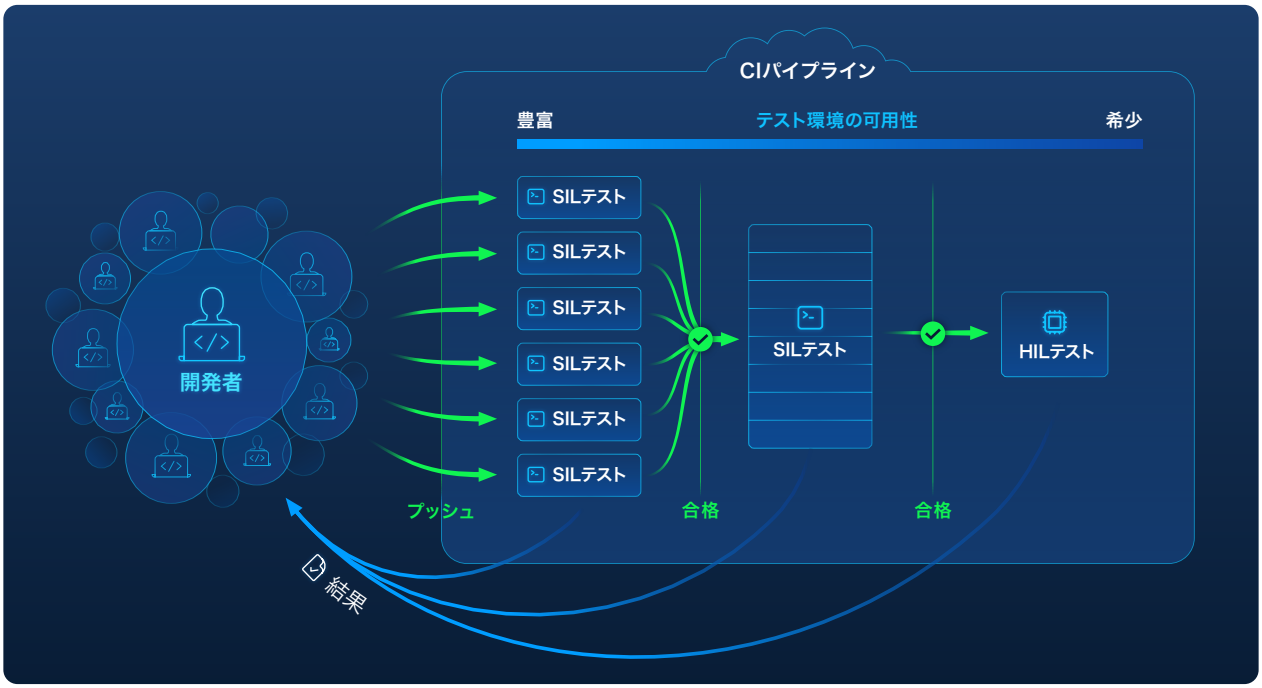
仮想環境はまた、OEM 開発グループや Tier1 サプライヤーを含む、チームや組織間でのシステム構成の共有を容易にします。

2.4 分散システムのための継続的インテグレーション

仮想開発の重要な利点として、ECU ソフトウェアテストを継続的インテグレーション（CI）パイプラインに統合できることが挙げられます。

多くの現代的なソフトウェア開発環境では、新しいコードがリポジトリにコミットされるたびに、ソフトウェアが自動的にビルド・テストされます。しかし、このアプローチを分散型自動車システムに適用することは、物理ハードウェアへの依存のため、従来は困難でした。

vECU とシステムレベルの仮想化環境を使用することで、ビルドプロセスの一環として仮想車両システムを自動的に組み立て、実行することが可能になります。自動テストにより、複数の ECU 間でのシステム動作を検証できます。



仮想環境は機能テストと統合テストを担い、HIL リグは実際のハードウェアを必要とする検証——タイミング、ジッター、電気レベルの動作——に活用されます

一般的な CI ワークフローには以下が含まれます：

1. ソースコードの変更をリポジトリにコミット
2. ECU ソフトウェアをコンパイルし、vECU としてパッケージ化
3. シミュレーション環境内で仮想車両システムをインスタンス化
4. 自動統合テストを実行
5. テスト結果を開発チームに報告

このアプローチにより、システムレベルの欠陥を導入直後に検出でき、後工程における統合障害のリスクを低減できます。

2.5 物理ハードウェアへの依存の軽減

最終的な検証や性能テストにおいて物理ハードウェアは依然として不可欠ですが、仮想開発環境により、早期のハードウェアベースのテストの必要性は大幅に軽減されます。これにより、限られたハードウェアリソースによるボトルネックが解消され、エンジニアリングチームは物理コンポーネントが必要なシナリオにハードウェアベースのテストを集中させることができます。

RemotiveTopology を使用する場合、仮想環境内のすべてのバス通信は量産車両プラットフォームと同一です――ネットワークパケットのエンコーディング、IP/MAC アドレス、VLAN などを含みます。これにより、仮想環境で機能を検証するために使用されたテスト資産を、物理ハードウェアでのテスト時にも再利用でき、テストの書き直しなしに仮想環境と物理環境の間を容易に移行できます。

2.6 スケーラブルなシステムレベル仮想化に向けて

個々の vECU の実行はより早期のテストに向けた重要なステップですが、効果的なシステムレベルの検証には車両アーキテクチャ全体をシミュレートする能力が必要です。

これには以下の表現が含まれます：

- 車両内の ECU のトポロジー
- ECU を接続する通信ネットワーク
- ネットワーク上で交換されるサービスと信号
- 分散ソフトウェアコンポーネント間の依存関係

これらの仮想車両アーキテクチャの管理と実行には、多数の相互作用するソフトウェアコンポーネントを統合運用できる専門的なツールが必要です。

RemotiveTopology などのプラットフォームは、システムレベルのオーケストレーションレイヤー（orchestration layer）を提供し、エンジニアが個別および統合された車両サブシステムの仮想表現を定義・実行できるようにします。

3. 仮想トポロジーにおける ECU コンポーネントの表現方法

仮想車両トポロジーには通常、2 種類のコンポーネントが含まれており、各表現方法の目的を理解する上でこの区別は重要です。

環境コンポーネント（environment component）は、テスト対象のデバイスまたはシステムを取り囲む ECU を表現します。その役割は、テスト対象のコンポーネントが量産通信を使用して動作・テストできるよう、リアルな信号動作、ネットワークトラフィック、システムコンテキストを提供することです。環境コンポーネントは、代替する ECU の内部動作を再現する必要はなく、説得力のある「隣接 ECU」であれば十分です。

DUT コンポーネント（DUT component）は、実際に検証対象となるソフトウェアまたはハードウェアを表現します。実際の量産コードまたはその高忠実度モデルを実行し、テストの対象となります。

ほとんどのトポロジーは両方を同時に含みます：数十の隣接 ECU の代わりにモックやビヘイビアモデル（behavioral model）が配置され、DUT 自体は vECU プラットフォーム内で量産ソフトウェアを実行するか、コンテナ内で Linux コンパイル済みバイナリとして動作するか、物理ハードウェアとして接続されます。以下のセクションでは、環境コンポーネントと DUT コンポーネントの両方を表現する方法を、忠実度の低いものから高いものへと順に説明します。実際には、ほとんどのトポロジーが複数の方法を同時に組み合わせています——このパターンはセクション 3.6 で説明します。

3.1 モック

最もシンプルな仮想化レベルがモックです。

モックは、簡略化されたプレースホルダー実装を使用して ECU またはソフトウェアコンポーネントを表現します。ECU の実際の内部機能を実装する代わりに、モックは他のコンポーネントが期待する外部インターフェースのみを提供します。例えば、テストシステムからの API 呼び出しに応じて、特定のレートと形式で信号値を送信する場合があります。

例えば、早期の機能開発において、HMI（ヒューマンマシンインターフェース）ECU がまだ実装されていないセンサー ECU からの信号に依存している場合、モックがこれらの信号をシミュレートすることで、HMI 開発を進めることができます。

モックは軽量で作成が容易なため、早期のアーキテクチャ探索やインターフェース検証に特に有用です。

ただし、モックにはいくつかの制限があります：

- ECU の内部動作を表現しない
- 実際の量産ソフトウェアを実行しない
- タイミング動作や複雑なロジックを正確に再現できない

そのため、モックは最終的なシステム動作の検証よりも、早期の統合実験に最適です。

3.2 ビヘイビアモデル

より高度な仮想化レベルがビヘイビアモデルです。

ビヘイビアモデルは、実際の量産ソフトウェアを実行することなく、ECU の機能的な動作をシミュレートしようとするものです。ECU の動作は、コンポーネントが入力に対してどのように応答し、出力を生成するかを近似するソフトウェアモデルによって表現されます。

ビヘイビアモデルには通常、以下が含まれます：

- 論理的意思決定
- ステートマシン
- 簡略化された制御アルゴリズム
- イベント駆動型動作

これらのモデルは軽量で、Python などの汎用プログラミング言語やスクリプティングフレームワークを使用して最小限の工数で実装できます。

ビヘイビアモデルは、ECU の詳細な実装がまだ利用できない場合に、システムレベルの相互作用を検証するのに有用です。コンポーネントの機能的動作を近似するため、システム内の他の ECU が欠落しているコンポーネントのよりリアルな表現と対話できるようになります。

例えば、車両リアライトコントロールモジュール（RLCM）のビヘイビアモデルは、受信した CAN バストラフィックをリアライトに送信する LIN メッセージに変換する動作をシミュレートする場合があります。

モックと比較して、ビヘイビアモデルは以下を提供します：

- ECU 間のより現実的な相互作用
- 機能レベルテストのより良いサポート
- システム状態のより柔軟なシミュレーション

ビヘイビアモデルは、通常のシステム実行範囲外の非定常シナリオのシミュレーションにも特に有用です。ECU が誤った出力を生成する場合、センサーが範囲外の値を報告する場合、走行中にドアが開けられるような安全上重要なイベント――これらは実際のハードウェアで再現することが困難または危険な状況ですが、車両が路上に出る前に徹底的にテストされなければなりません。ビヘイビアモデルは量産コードから派生するのではなくプログラム的に構築されるため、意図的に不正な出力を含むあらゆる出力を生成するよう構成でき、システム全体のエッジケースやフォールトハンドリングパスを実行するための自然なツールとなります。

この柔軟性には自然なトレードオフが伴います：ビヘイビアモデルは量産ソフトウェアを近似するものであり実行するものではないため、正確なタイミング、内部状態管理、コードパスの動作を完全に再現することはできません。量産レベルの忠実度を検証するには、以下のサブチャプターで説明する vECU などの高レベルの仮想化手法が必要です。

3.3 Simulink ベースモデルと FMU

多くの自動車 ECU は、特にパワートレイン、シャシー制御、先進運転支援システムなどのドメインにおいて、モデルベース開発アプローチを使用して開発されています。

これらの場合、ECU 機能は MathWorks 社の Simulink などのツールで開発されたモデルを使用して表現される場合があります。

Simulink モデルは、制御アルゴリズム、信号フロー、システムダイナミクスを表すブロック図を使用してシステム動作を記述します。これらのモデルはシミュレーション環境内で実行でき、コード生成やデプロイメントの前にシステム動作を評価できます。

仮想開発環境内で使用する場合、Simulink モデルにはいくつかの利点があります：

- 詳細な機能アルゴリズムの表現
- 高忠実度での制御動作のシミュレーション
- システムレベルダイナミクスの早期検証

Simulink ベースのコンポーネントは、制御アルゴリズムや信号処理機能の検証に特に価値があります。

例えば、電動パワーステアリングコントローラは、シミュレートされた車両入力に基づいてステアリングトルクを計算する Simulink モデルで表現される場合があります。

ただし、Simulink モデルをシステム統合テストに使用する場合にはいくつかの制限もあります：

- フル量産ソフトウェアスタックを含まない可能性がある

» ミドルウェアや通信レイヤーが簡略化されている場合がある

» 他のソフトウェアコンポーネントとの統合に追加のラッパーが必要な場合がある

これらの要因から、Simulink モデルはより大規模な仮想車両システムを構築する際に、他の仮想化アプローチと併用されることが多いです。

3.4 ECU ソフトウェアバイナリ（vECU）

ソフトウェアリアリズムの最高レベルは、シミュレートされた ECU ハードウェア上で vECU を使用することで達成されます。

vECU は、最終的な ECU ハードウェアの CPU アーキテクチャとハードウェアインターフェースをモデル化するホストコンピューティング環境内で、実際の ECU ソフトウェアスタックを実行します。

これにより、ECU ソフトウェアバイナリは高い動作精度を維持しながらシミュレーション環境内で実行できます。

» 一般的な vECU には以下が含まれます：

» アプリケーションソフトウェア

» AUTOSAR ランタイム環境、または Android Automotive OS

» 通信スタック

» オペレーティングシステムコンポーネント

» サービスインターフェース

vECU は通常、専門的な vECU 実行プラットフォーム（例：Vector vECU、Synopsys Silver）を使用して生成されます。これらのプラットフォームは、シミュレートされた ECU ハードウェア上で ECU ソフトウェアを実行し、シミュレートされた車載ネットワークに接続します。

vECU は実際の量産ソフトウェアバイナリを実行するため、以下が可能になります：

» ECU 間のリアルなソフトウェアインタラクション

» ミドルウェアおよび通信スタックの検証

» 量産レベルのコードパスの実行

» 統合欠陥の早期検出

このため、vECU は仮想開発の高度な統合・テストフェーズで使用されることが多いです。

3.5 最終 ECU ハードウェア

RemotiveTopology は、さまざまなレベルの仮想化に限定されるものではありません。利用可能な場合には、物理 ECU ハードウェアもトポロジーに組み込むことができます。Kvaser や National Instruments などの企業が、物理ハードウェアと仮想コンポーネント間の接続を可能にするハードウェアアダプターを提供しています。

3.6 ハイブリッド：複数の表現方法の組み合わせ

実際のところ、完全な仮想車両システムは複数の表現方法を混在させて構成されます――そして環境 /DUT の区別がこれを実用的なものにしています。一般的なトポロジーのコンポーネントの大半は環境表現です：数十の隣接 ECU の代わりに配置されたモックやビヘイビアモデルです。少数の DUT コンポーネントがより高い忠実度――vECU、Linux コンパイル済みバイナリ、または物理ハードウェアとして――実行され、実際の検証対象となります。

これらのアプローチを組み合わせることで、一部のコンポーネントがまだ利用できない段階でも仮想システムを構築できます。モックはソフトウェアが未作成の ECU の代わりを務め、ビヘイビアモデルや FMU がより完全な機能シミュレーションを可能にし、利用可能な場合には ECU ソフトウェアバイナリを vECU 上で実行でき、物理ハードウェアは開発のどのフェーズでも組み込むことができます。RemotiveTopology はこれらの異種コンポーネントを統一されたシステム環境内で統合運用し、プログラムの進行に伴って低忠実度の代替物をより高忠実度の実装に段階的に置き換えていきます——周囲のテストツールやスクリプトを再構築する必要はありません。

同じ論理で、仮想化と HIL テストの関係も説明できます。システムレベルの仮想化への移行は、HIL に反対する主張と誤解されることがあります。しかし、そうではありません。HIL リグは、実際のハードウェアと実際のタイミング動作を必要とする検証活動——精密なジッター分析、電気レベルの信号完全性、ハードウェア故障注入、およびデバイスがデプロイ時と全く同じ条件で動作する必要がある認証シナリオ——において依然として不可欠です。

仮想開発が変えるのは、そもそも HIL 上で実行する必要のあるテストの量です。機能検証——機能が分散 ECU 間で正しく動作するか、インターフェース契約が遵守されているか、起動シーケンスやサービスディスカバリーが設計通りに機能するか——には、実際のハードウェアは必要ありません。この作業が仮想環境に移行すると、HIL リグは HIL にしかできない作業に解放され、レイヤード・トポロジーが各検証活動に適切な手段を提供します：仮想で十分な場合は仮想、ハードウェアが必要な場合はハードウェア、テストが求める場合は両方を同じシステム内で使用します。

単一のシステムトポロジー内で複数のコンポーネント表現方法をサポートし、物理ハードウェアも同じ条件で統合することで、開発チームは開発ライフサイクル全体を通じて忠実度を段階的に向上させながら、ハードウェアベースのテストをそれが真に必要な活動に限定することができます。

4. RemotiveTopology と vECU 実行ツールの補完的役割

現代の仮想開発環境は、単一のツールで構成されるものではありません。通常、ソフトウェア開発スタックの異なるレベルで動作する技術を組み合わせて構築されます：個々の ECU ソフトウェアコンポーネントの実行に特化したものもあれば、車両システム全体の組み立てと管理に注力するものもあります。これらのツールカテゴリ間の関係を理解することは、効果的な仮想開発ワークフローを構築する上で不可欠です。

本章では、RemotiveTopology と専用の vECU 実行プラットフォーム（例：Synopsys Silver、dSpace VEOS、Vector CANoe）が、仮想開発環境内でどのように異なる補完的な機能を果たすかを説明します。これらのツールは競合するものではなく、開発スタックの異なる位置を占めています：RemotiveTopology は開発ライフサイクル全体にわたる水平的なオーケストレーションレイヤーとして機能し、vECU 実行プラットフォームは開発の特定フェーズに高忠実度の ECU シミュレーションを提供する垂直的な専門ツールです。

これらを組み合わせることで、業界で「レフトシフト（left shift）」と呼ばれる統合テストの前倒しが可能になります——物理ハードウェアが利用可能になる前の、課題の解決コストがまだ低い段階で、システムレベルの検証を実施できます。

4.1 水平オーケストレーションレイヤーとしての RemotiveTopology

最も高い抽象レベルで見ると、車両は CAN、LIN、車載イーサネットなどのネットワークを介して通信する、複数の ECU に分散された相互作用するソフトウェアコンポーネントのシステムとして捉えることができます。このシステムを検証するには、個々の ECU ソフトウェアコンポーネントを単独で実行するだけでは不十分であり、完全な車両アーキテクチャ全体を記述、組み立て、実行するメカニズムが必要です。

RemotiveTopology はこの能力を提供します。車両の E/E アーキテクチャをソフトウェアで定義する、永続的なシステムレベルレイヤーとして機能します：システム内の ECU、それらを接続する通信ネットワーク、コンポーネント間で交換される信号とサービス、分散ソフトウェア機能間の依存関係を定義します。このアーキテクチャは AUTOSAR ARXML やネットワーク定義ファイルなどの既存の量産設計成果物から直接導出されるため、仮想システムは実車と同じ構造と通信定義を反映します。

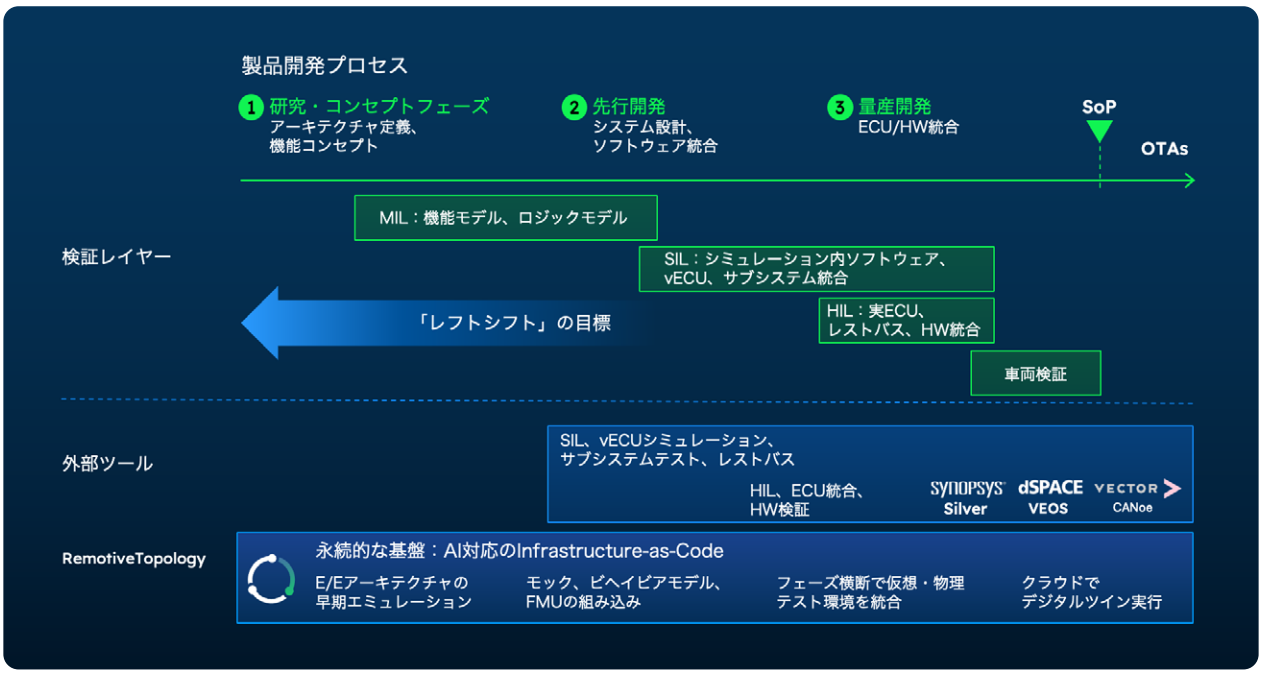
RemotiveTopology はコンポーネントレベルではなくシステムレベルで動作するため、製品開発プロセス全体にわたる永続的な基盤（persistent backbone）として機能します。研究・コンセプトフェーズでは、ドラフトアーキテクチャファイルのみからトポロジーを立ち上げ、モックを配置し、ECU ソフトウェアが一行も書かれる前にインタラクションパターンとインターフェース定義を検証するために使用できます。先行開発では、ソフトウェアの成熟に伴い、モックがビヘイビアモデル、FMU、vECU に段階的に置き換えられます。量産開発では、ハードウェアアダプターを通じて残りの仮想コンポーネントと並行して物理ハードウェアが導入されます。量産開始後は、同じトポロジーがデプロイされた車両アーキテクチャに対する OTA ソフトウェアアップデートの検証をサポートします。これらすべてのフェーズを通じて、トポロジー内の ECU コンポーネントは忠実度が進化しますが、システムアーキテクチャレイヤー自体は一貫性を保つため、コンセプトワーク中に開発されたテスト資産は SoP 後の検証まで再設計なしに再利用できます。

4.2 垂直専門ツールとしての vECU 実行プラットフォーム

RemotiveTopology がオーケストレーションインフラを提供する一方で、ECU ソフトウェアバイナリをシミュレートされたハードウェア上で実行する機能は提供しません。これは vECU 実行プラットフォームの役割です：ターゲット ECU の CPU アーキテクチャ、オペレーティングシステム、ハードウェアインターフェースをモデル化し、実際の量産ソフトウェアスタックを仮想環境でコンパイル・実行できるようにする専門ツールです。

複数の実績あるプラットフォームがこの機能を提供しています（例：Synopsys Silver、dSpace VEOS、Vector CANoe）。実際の量産ソフトウェアを実行するため、ミドルウェアの動作、通信スタック、量産レベルのコードパスの検証が可能です。

これらのプラットフォームは、ECU ソフトウェアが量産に近いバイナリとしてコンパイル・実行できるほど十分に成熟した SIL および HIL フェーズにおいて最も価値があります。高忠実度・高特異性のツールであり、モデル対象の ECU に対して正確な結果を生成しますが、その ECU が最終的に動作するシステムレベルのコンテキスト——周囲の ECU、ネットワーク、車両アーキテクチャ——を単体で提供するものではありません。



RemotiveTopology は初期のコンセプトから OTA アップデートまで、開発サイクル全体をカバーします

4.3 共有システム環境内での統合

これら 2 つのツールカテゴリの補完的な性質は、組み合わせたときに明確になります。RemotiveTopology がシステムコンテキストを提供し、vECU 実行プラットフォームが ECU レベルの忠実度を提供します。これらを合わせることで、異なるコンポーネントが異なる抽象レベルで同時に動作する仮想車両システムを構築できます。

実際には、開発チームはほとんどの ECU が軽量なモックで表現された仮想トポロジーから始め、開発の最も早い段階からシステムレベルの通信やインターフェース動作を検証できます。ECU ソフトウェアが成熟するにつれ、個々のモックはそれぞれの実行プラットフォーム上で量産ソフトウェアを実行する vECU インスタンスに段階的に置き換えられます。システムトポロジーはコンポーネントの実装とは独立して定義されているため、この置き換えはインクリメンタルに——1 つの ECU ずつ——周囲の仮想システムを中断することなく行えます。

このアーキテクチャにより、利用可能になった物理ハードウェアも同じ環境に参加できます。例えば、テスト対象のゾーンコントローラを、残りの ECU が仮想インスタンスとして動作し続ける仮想トポロジーに接続できます。システムトポロジーは実際のネットワークプロトコルを使用して物理コンポーネントと仮想コンポーネント間の通信を管理するため、テスト対象の ECU は量産車両と全く同じように仮想の隣接 ECU と対話します。

その結果、完全な仮想からハイブリッドの仮想・物理環境へと連続的にスケールし、同じテスト資産とシステム定義がこの移行を通じて持続する開発環境が実現します。

4.4 仮想開発スタックにおける各コンポーネントの役割

以下の表は、仮想開発環境に関与するさまざまな技術の役割と、各機能を実現するコンポーネントをまとめたものです。

機能	実現コンポーネント
システムプラットフォームのオーケストレーション	RemotiveTopology
仮想車載ネットワーク（ECU 間通信）	RemotiveTopology
モック	RemotiveTopology
ビヘイビアモデル	RemotiveTopology
FMUs	RemotiveTopology
ECU ソフトウェアバイナリの実行	vECU 実行ツール
仮想トポロジーと ECU ハードウェアの混在	RemotiveTopology とハードウェアアダプター（例：Kvaser、NI など）

このレイヤード構造により、異なる仮想化アプローチが同一のシステムトポロジー内で共存できます。開発チームは簡略化されたコンポーネント表現から統合を開始し、ECU ソフトウェアが利用可能になるにつれて段階的に高忠実度の実装に置き換えることができます。

5. RemotiveTopology の技術アーキテクチャ

仮想車両環境は、実車両のアーキテクチャを正確に表現しつつ、分散開発ワークフローをサポートするための十分な柔軟性を備えている必要があります。これを実現するには、標準化されたアーキテクチャ記述、コンテナ化された実行環境、スケーラブルな通信メカニズムの組み合わせが必要です。

RemotiveTopology は、実際の量産車両の構造を反映しつつ、仮想コンポーネントと開発ツールの柔軟な統合を可能にするシステムアーキテクチャを通じて、これらの要件に対応します。

本章では、仮想システムの定義方法、ECU コンポーネントの実行方法、分散コンポーネント間の通信の管理方法など、RemotiveTopology プラットフォームの技術的基盤について説明します。

5.1 量産設計ファイルからのトポロジー生成

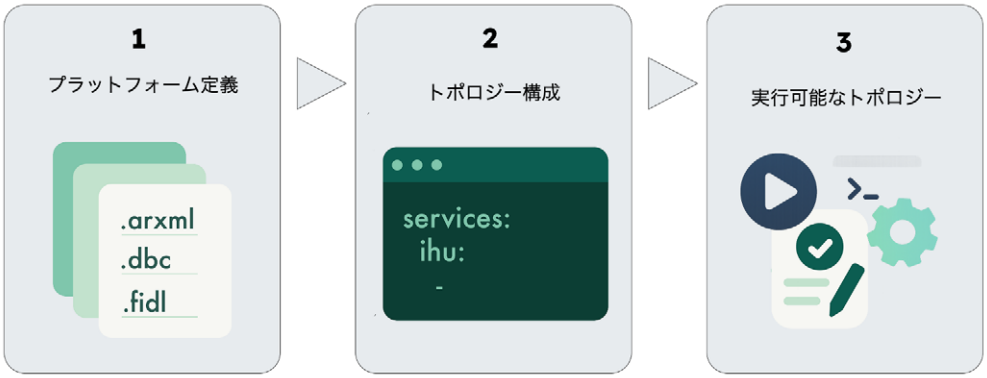
RemotiveTopology の重要な原則は、仮想車両環境が実際の量産車両のアーキテクチャと通信方法をできるだけ忠実に反映すべきであるということです。

開発者がシステムアーキテクチャを手動で定義する必要はなく、RemotiveTopology は既存の車両設計成果物から直接仮想車両トポロジーを生成します。

これらの成果物には通常、以下が含まれます：

- ARXML ファイルに格納された AUTOSAR アーキテクチャ記述
- DBC ファイルに格納された CAN バス定義
- LDF ファイルに格納された LIN バス記述
- FIBEX ファイルに格納された SOME/IP 定義

これらの設計ファイルをインポートすることで、RemotiveTopology は量産車両アーキテクチャの構造を反映した仮想システムトポロジーを自動的に構築できます。



RemotiveTopology は既存の設計成果物を使用して構築されます

このアプローチには 2 つの重要な利点があります：

» 第一に、仮想トポロジーが量産システムとアーキテクチャ的に同一になります。仮想環境で使用される ECU、通信チャネル、信号定義は、実車で使用されるものと直接対応します。

» 第二に、仮想環境内の通信は、量産車両用に定義されたものと同じネットワークメッセージと信号を使用します。これにより、仮想 ECU 間のインタラクションは、後に物理車載ネットワークで使用されるのと同じプロトコルとメッセージフォーマットに従います。

その結果、仮想環境で実行されるシステムレベルのテストは、最終車両に存在するのと同じ通信インターフェースを検証できます。

5.2 ECU コンポーネントのコンテナ化実行

仮想トポロジー内では、各 ECU コンポーネントが独自の隔離された実行環境内で動作します。各 ECU インスタンス——モック、ビヘイビアモデル、Simulink コンポーネント、vECU のいずれであっても——はコンテナ（container）イメージとしてパッケージ化されます。仮想トポロジーがインスタンス化されると、RemotiveTopology は必要なコンテナを起動し、定義されたシステムアーキテクチャに従ってそれらを接続します。

コンテナ化にはいくつかの重要な利点があります：

隔離性：各 ECU は隔離された実行環境内で動作します。ある ECU の依存関係やランタイム構成が他の ECU に干渉することはありません。

再現性：コンテナイメージは ECU ソフトウェアの正確なランタイム環境をキャプチャします。これにより、開発マシン、統合サーバー、クラウドインフラ間で同一のソフトウェア構成を一貫して実行できます。

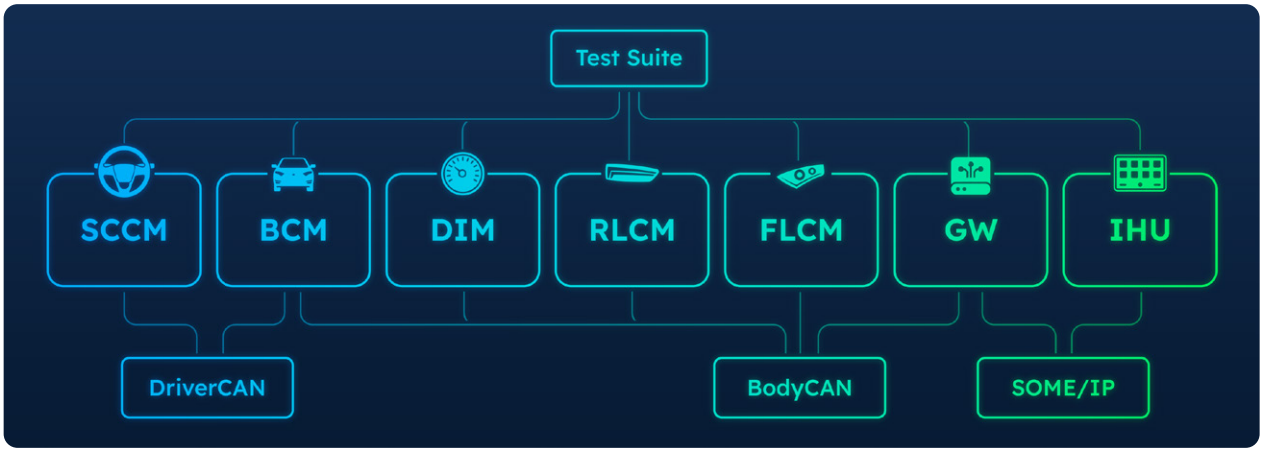
スケーラビリティ：各 ECU コンポーネントは標準的なコンテナとして実行されるため、トポロジーは確立されたオーケストレーションツールを使用してクラウドインフラにデプロイできます。チームは分散コンピューティングリソース全体で複数のテストシナリオを並列実行し、検証環境をオンデマンドでスケールアップ・ダウンし、専用ハードウェアなしで CI パイプライン用の新しいトポロジーインスタンスをプロビジョニングできます。開発者のラップトップで実行されるのと同じコンテナ化されたセットアップが、そのままクラウドで実行されます。

5.3 ECU 間通信

物理的な車両では、ECU は CAN、LIN、FlexRay、車載イーサネットなどの車載ネットワークを通じて通信します。仮想環境では、これらの通信メカニズムをソフトウェアで再現する必要があります。

RemotiveTopology は、仮想環境内で車載ネットワークメッセージングを再現する通信レイヤーを使用して、コンテナ化された ECU 間の通信を可能にします。通信定義は OEM の既存の設計ファイルに由来するため、仮想 ECU 間で交換されるメッセージは量産車両用に定義されたものと直接対応します。

重要なのは、RemotiveTopology がプロプライエタリなメッセージバスではなく、実際のネットワークプロトコルを使用していることです。パケットのエンコーディング、IP および MAC アドレッシング、VLAN タギングを含むすべてのバス通信が量産車両プラットフォームと一致します。これが、Wireshark や candump などのネットワークパケットモニタリング IT ツールで仮想ネットワークを検査でき、変換レイヤーなしで仮想コンポーネントと物理コンポーネントを同じトポロジー内で混在させることができる理由です。



複数の ECU と 3 つのバスで接続されたトポロジー

5.4 開発・テストツール向けオープンインターフェース

RemotiveTopology の重要な設計原則は、ツールチェーンの柔軟性です。

自動車開発組織は、さまざまなテストフレームワーク、デバッグツール、分析環境を使用しています。プロプライエタリなツールチェーンを要求すると、仮想開発プラットフォームの採用を大きく制限する可能性があります。

この課題に対処するため、RemotiveTopology はオープンインターフェースを提供し、チーム横断でエンジニアが社内およびサードパーティのツールを仮想開発ワークフローに統合し、共通のツーリングベースラインを確立して、チーム間のコラボレーションをよりスムーズにします。

例えば：

- 自動テストスクリプトとフレームワークが ECU サービスと対話可能
- デバッグツールが ECU 間のネットワークトラフィックを検査可能
- データ分析ツールがテストシナリオ中の信号動作を監視し、チームがシステムレベルのインタラクションをトレースし、ECU 境界を越えたパフォーマンスや正確性の問題を特定可能

これらのインターフェースはオープンであるため、開発チームは既存および将来のワークフローに最適なツールを自由に統合できます。

このアプローチはベンダーロックインを回避し、OEM やサプライヤーが既存のツールチェーンの根本的な変更を必要とせずに仮想開発環境を採用でき、複数のチームや大規模組織に共通環境を提供します。

オープンインターフェースに加えて、RemotiveLabs は RemotiveStudio を提供しています——トポロジー構造の探索、信号データベースの検索、構成可能なダッシュボードを

通じた稼働中のトポロジーからのライブ信号データの監視を行う Web ベース環境です。RemotiveStudio は稼働中の RemotiveTopology インスタンスに直接接続するため、トポロジーの詳細に精通していないエンジニアを含め、追加のツールやセットアップなしに車載ネットワークを視覚的に検査し、リアルタイムで信号動作をトレースし、チーム間でダッシュボードを共有できます。

5.5 柔軟な開発ワークフローのサポート

アーキテクチャ駆動型トポロジー生成、コンテナ化 ECU 実行、ソフトウェア定義通信ネットワーク、オープンツールインターフェースを組み合わせることで、RemotiveTopology は幅広い自動車開発ワークフローをサポートするスケーラブルな仮想開発環境を実現します。

これらの機能により、開発チームは以下が可能になります：

- 量産アーキテクチャ定義から直接仮想車両システムを構築
- 隔離されたコンテナ環境内で分散 ECU ソフトウェアを実行
- コンポーネント間の車載ネットワーク通信をシミュレート
- 複数ベンダーのテスト・デバッグ・分析ツールを統合

これらの機能を合わせることで、開発チームは物理ハードウェアとの量産アーキテクチャ的一貫性を維持しながら、開発プロセスのより早い段階でシステム統合と検証活動を実施でき、物理ハードウェアへの依存を軽減できます。

5.6 Infrastructure-as-Code と AI 対応アーキテクチャ

RemotiveTopology を多くの既存の自動車ツールと区別する重要な特性は、仮想車両環境を IaC として扱うことです。トポロジー定義、バス、通信マッピング、メッセージはすべてテキスト成果物として表現されます：バージョン管理され、マシン間で再現可能であり、エンジニアリングチームがコードに対して既に使用しているのと同じワークフローで変更可能です。

このアプローチには実用的な結果があります：

ベンダーロックインなし：コア機能は自己完結型です。トポロジーの立ち上げ、実行、動作の検査は、オープンインターフェースと標準ツールを通じて行われます。チームは必要に応じてサードパーティツールと統合でき、既存のツールチェーンを中断することなくプラットフォームをインクリメンタルに採用できます。

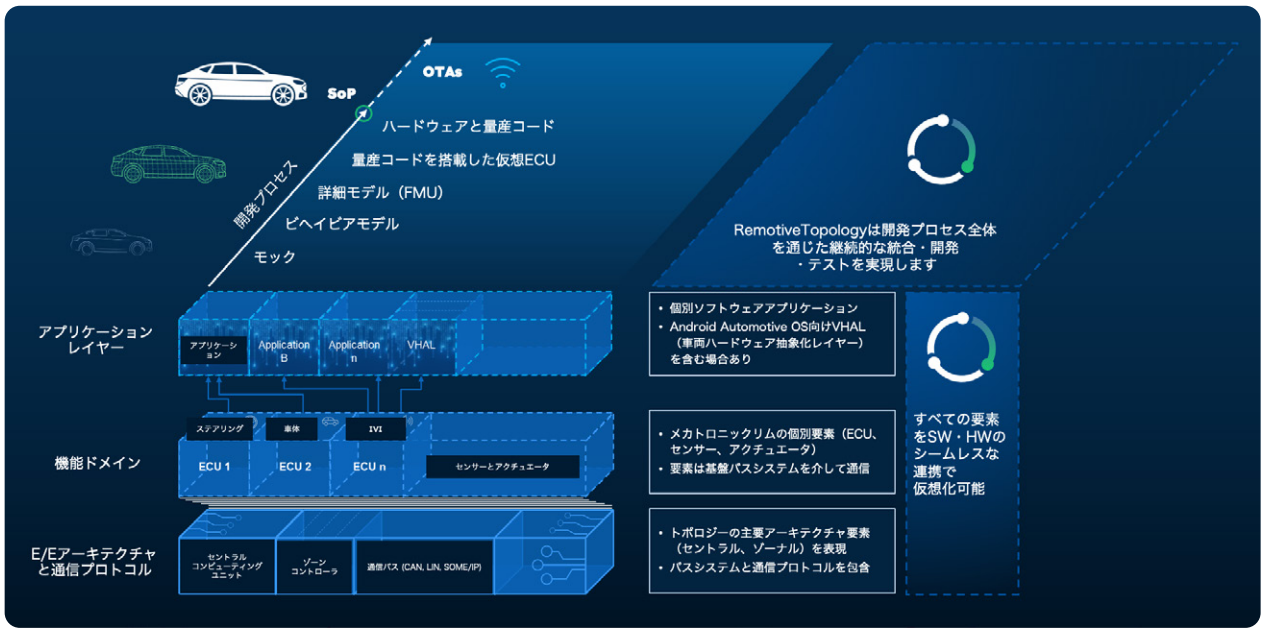
AI ネイティブな構造：トポロジーが完全に機械可読であるため、AI エージェントは RemotiveLabs コマンドラインツールや MCP API を通じて直接トポロジーと対話できます——信号のクエリ、アーキテクチャ構成の変更、稼働中のトポロジーに対するテストシナリオの生成や検証が、専用の統合作業なしに可能です。信号探索、アーキテクチャ最適化、検証スクリプティングが、専門的な活動ではなく、日常的なエージェント支援型の作業となります。

その生産性効果は具体的です。トポロジーに対してエージェント型 AI を使用し、正しい信号処理を備えた完全に統合された Android インストルメントクラスターをゼロから半日で生成し、中型 SUV 向けの完全な 20-ECU バックボーンと信号データベースを 1 日で作成しました。統合テストのレフトシフト効果と組み合わせることで、分散型車両ソフトウェアの構築・検証速度に段階的な変化をもたらします。

6. RemotiveTopology が可能にするユースケース

従来のツールは、開発プロセスの限られた領域をカバーする傾向があります：HIL プラットフォームはタイムラインの後半で ECU とアプリケーションをカバーし、vECU 実行ツールはプログラムの中盤以降で ECU をカバーし、コンセプトフェーズにはこれまで実行可

能な表現が存在しませんでした。RemotiveTopology は、永続的な基盤としてプロセス全体にわたります——スタックのあらゆるレイヤーとプログラムのあらゆるフェーズにおいて、初日から SoP 後の OTA アップデートまで対応します。

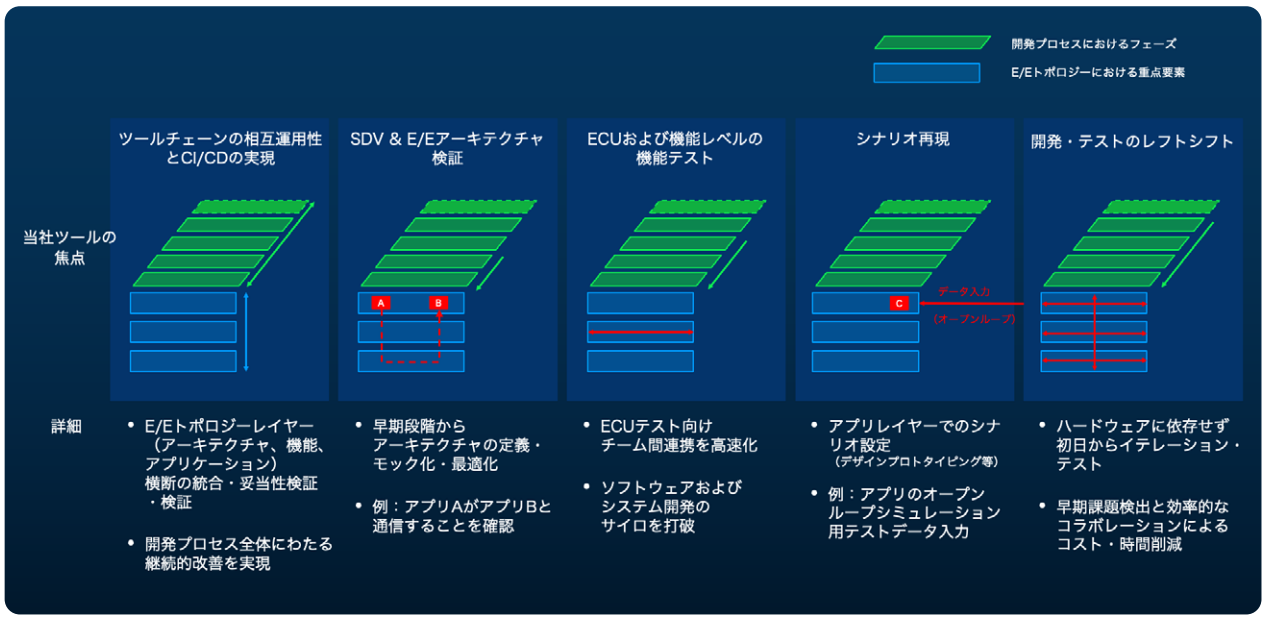


上の図は、RemotiveTopology がより広い自動車開発環境においてどこで動作するかを 2 つの次元で示しています：

» 垂直方向の自動車スタック——下部の E/E アーキテクチャと通信プロトコルから、ECU、上部のアプリケーションまで

» 水平方向の製品開発タイムライン——研究・コンセプトから先行開発、量産開発、SoP 後まで

この広範なカバレッジが、以下のユースケースを可能にしています。一部は、仮想化が検証タイムラインをコンセプトワークへと左方向に拡張することで初めて実現する早期フェーズのシナリオです。その他は、仮想化がハードウェアテストを補完する——置き換えるのではなく——中期・後期フェーズのシナリオであり、実際のシリコンを必要とする検証活動に HIL 容量を解放します。すべてのユースケースに共通するパターンがあります：仮想環境が機能・統合の作業を担い、同じトポロジー定義がプログラムのあらゆるステージを通じて引き継がれます。



以下の例は、コンセプトワークでのアーキテクチャ検証から SoP 後のシナリオまで、開発ライフサイクル全体にわたる RemotiveTopology の適用方法を示しています。

ユースケース

6.1 初日からの E/E アーキテクチャ検証

研究・コンセプトフェーズ・スピード・リスク軽減

目的

提案された E/E アーキテクチャ——その ECU トポロジー、通信マトリクス、インターフェース定義——を、ECU ソフトウェア開発が開始される前の開発プログラム早期に、実行可能なシステムとして動作させます。

課題

コンセプトフェーズで行われるアーキテクチャ上の決定（機能の ECU 間への分割方法、どの信号がどのネットワークを経由するか、サービスの発見方法、ドメイン間の依存関係など）は、プログラム内で見直すコストが最も高い決定です。従来、アーキテクチャの実行可能な表現が存在しないため、これらはレビューと専門家の判断によってのみ検証されてきました。そのため、アーキテクチャ上の欠陥は 12 〜 18 か月後の ECU 統合時に初めて表面化し、その時点での変更は複数のサプライヤーに連鎖的な影響を与えます。

アプローチ

ドラフトアーキテクチャファイルをインポートして、提案されたアーキテクチャの実行可能な表現を生成します。すべての ECU に軽量なモックを配置し、インタラクションシナリオ——起動シーケンス、クロスドメインサービス呼び出し、診断トラフィック——をトポロジーに対してスクリプト化し、アーキテクチャ全体を動作させます。

結果

アーキテクチャ上の課題——信号の欠落、あいまいなインターフェース、非現実的なタイミング仮定、循環的なサービス依存——がコンセプトワーク中に、アーキテクチャの変更コストがまだ低い段階で表面化します。チームは、ドキュメントとしてレビューされただけでなく、システムとして動作検証されたアーキテクチャを持って先行開発に入ります。

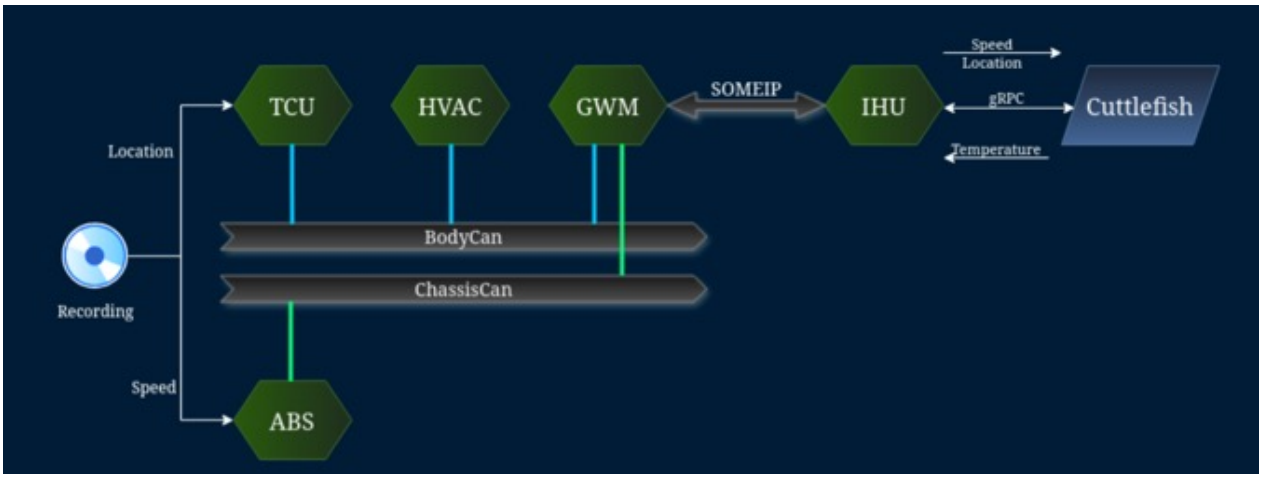
創出される価値

- » ECU 開発開始前にアーキテクチャ欠陥を検出——統合時と比較して変更コストは推定 10 ～ 100 分の 1
- » コンセプトフェーズのテストシナリオが先行開発から量産開発まで有効——忠実度の向上に伴う再設計が不要
- » サプライヤーは仕様書だけでなく、実行可能なリファレンスを受け取る
- » プログラム開始時から永続的なトポロジー基盤を確立

ユースケース

6.2 実車コンテキストを用いたインフォテインメントシステムの開発・デバッグ

早期開発・スピード・開発者生産性



VHAL 双方向統合による AAOS Cuttlefish 環境

目的

AAOS（Android Automotive OS）アプリケーションを含む HMI およびインフォテインメントシステムを、物理的な車両やリグに依存せず、実際の車両信号を使用して開発・テスト・デバッグします。

課題

AAOS 開発は、VHAL を介した正確な車両データに依存しています。従来、これには車両またはリグへのアクセスと手動の信号モニタリングが必要であり、フィードバックループの遅延、再現困難なバグ、発見が遅れる統合課題を引き起こしていました。

アプローチ

車両信号を VHAL レベルで双方向統合し、AAOS スタックに直接シミュレート、ストリーミング、記録、再生します。車両データは CAN の制限を超えてクラウドまたはイーサネット経由でアクセスされ、記録・再生機能により実環境の問題を決定論的に再現できます。同じセットアップがエミュレータ、仮想環境、ターゲットハードウェアのいずれでも動作し、開発・デバッグ・検証を通じた継続的なワークフローを実現します。

結果

開発者は早期の段階から実際のシステム動作と対話でき、統合時ではなく開発中に課題を発見します。開発からデバッグ、検証までの継続的なワークフローが実現されます。

創出される価値

- 実際または模擬信号からの即時フィードバックにより、イテレーションサイクルを高速化
- 記録・再生による再現可能なデバッグ
- 物理的な車両やリグへの依存を軽減
- 開発環境からオンターゲットテストへのシームレスな移行

拡張：デバッグからフルシステム統合へ

同じセットアップが、開発・デバッグからフルシステム検証ワークフローまでスケールします。チームは以下が可能です：

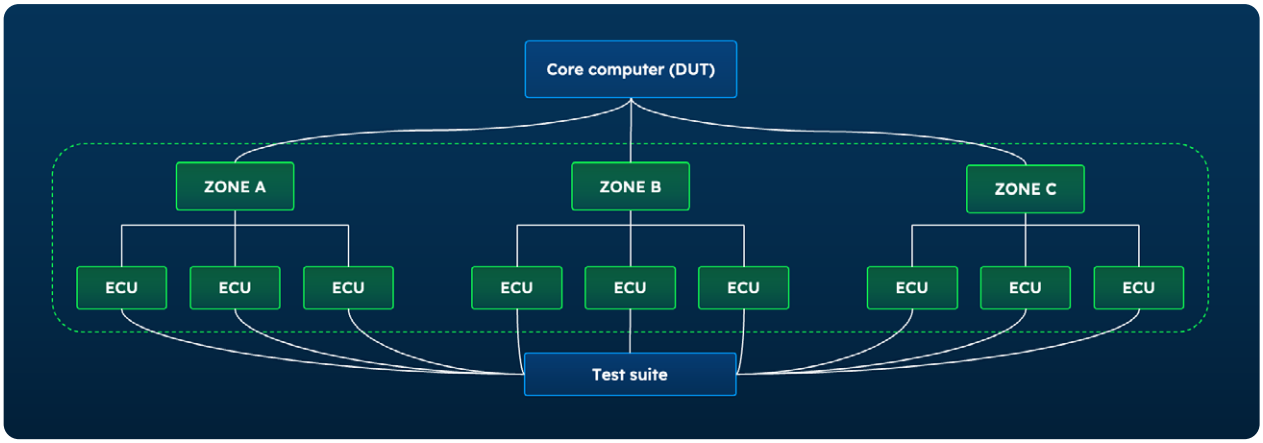
- 実環境の問題を開発環境に直接再生
- 模擬ユーザー行動と実際の車両信号を組み合わせ
- 仮想 AAOS テストからハードウェアまたは車両検証へ移行
- 最終検証が必要な場合に SIL/HIL セットアップへ橋渡し

これにより、開発→デバッグ→検証の継続的なワークフローが、すべて同一の信号駆動型基盤上で実現されます。

ユースケース

6.3 コアコンピュータテストのためのメカトロニックリムシミュレーション

早期開発・スピード・コスト



完全仮想メカトロニックリムにおけるコアコンピュータ（DUT）の検証

目的

新しい E/E アーキテクチャにおけるセントラルまたはゾーナルコンピュートユニットを検証します。

課題

セントラルコンピュートユニットのシステムレベル検証は、従来、周辺 ECU をハードウェアで再現する HIL リグに依存してきました。これらのリグは高価で数に限りがあり、通常は DUT の準備完了から数か月遅れて利用可能になるため、システムレベルのテストは欠陥の解決コストが最も高い後工程に押しやられます。

アプローチ

隣接 ECU をモックまたはビヘイビアモデルで置き換え、実際のネットワークプロトコルを使用したフル車両プラットフォーム通信を再現します。DUT は量産車両と全く同じように仮想の隣接 ECU と対話します。

結果

OEM は HIL へのアクセスを待つことなく、初日からシステムレベルの統合テストを開始できます。開発者は通信、依存関係、動作をはるかに早い段階で検証でき、ユニットテストと後工程のラボ検証との間のギャップを縮小します。

創出される価値

- » プログラム初日からシステムレベルの統合テストを開始——HIL リグ不要
- » HIL のスケジューリング圧力とコストを大幅に軽減
- » HIL チームだけでなく、すべての開発者がフル車両コンテキストにアクセス可能
- » ハードウェア到着時に同じトポロジーが引き継がれ、テストの再設計は不要

ユースケース

6.4 複数 ECU にまたがる車両ロジックのテスト

早期開発・スピード・開発者生産性

目的

複数の ECU にまたがる車両機能——パーソナライゼーションフロー、コンフォート機能、クロスドメインのインフォテインメントインタラクション——を、物理リグに依存せず、フルシステムコンテキストで検証します。

課題

マルチ ECU 機能の検証には、従来 HIL リグまたはテスト車両が必要であり、長いセットアップ時間、手動の信号設定、共有ハードウェアへの限られたアクセスにより、フィードバックループの遅延と開発者の自律性の低下を招いていました。

アプローチ

すべての関連 ECU を仮想化し、集中型の SIL セットアップでオーケストレーションします。モックが信号精度の高い動作を再現し、物理バスを必要とせず、開発者のラップトップや CI パイプラインから直接完全なエンドツーエンドテストを実行できます。

結果

チームは、ユーザーインタラクションからバックエンド ECU 応答まで、ハードウェア依存なしに完全な統合シナリオを実行できます。開発者は変更に対する即時フィードバックを得られ、複雑なシステム動作の継続的な検証が可能になります。

```yaml
# ECUs to instantiate
ecus:
  # a mock for SCCM
  SCCM:
    mock: {}
  # a python model for BCM
  BCM:
    models:
      bcm:
        type: container
        container:
          build:
            dockerfile: ../Dockerfile
          volumes:
            - ../ecus/bcm:/app/bcm
          working_dir: /app
          command: python -m bcm
  # no behavior needed for FLCM
  FLCM:
    channels:
      BodyCan0:

```

RemotiveTopology のインスタンス YAML を使用して、プラットフォームのどの部分をインスタンス化するか、各 ECU をどのように実行するかを記述できます。この例では、BCM ECU に Python で記述されたビヘイビアモデルが設定され、FLCM ECU はモックとして実行されます。

創出される価値

- » OEM 調査により、決定論的ソフトウェア環境でのエンドツーエンドテストが HIL リグテストと比較して 10 倍高速化（時間単位→分単位）

» リグの可用性に依存しない安定的で再現可能なテスト環境、ラップトップから CI パイプラインまでスケール

» 物理ハードウェアなしでリアルなシステム動作を実現

» 統合テストワークフローの完全な開発者オーナーシップ

» 大幅な時間削減（年間開発者一人あたり数百時間）

拡張：必要に応じて SIL から HIL へ

ハードウェア検証が必要な場合、同じセットアップをシームレスに HIL へ橋渡しできます。RemotiveTopology とハードウェアインターフェース（例：Kvaser、NI など）を組み合わせることで、チームは以下が可能です：

» 選択した仮想 ECU を実際のハードウェアノードに置き換え

» ハイブリッド SIL/HIL セットアップで特定の機能をテスト

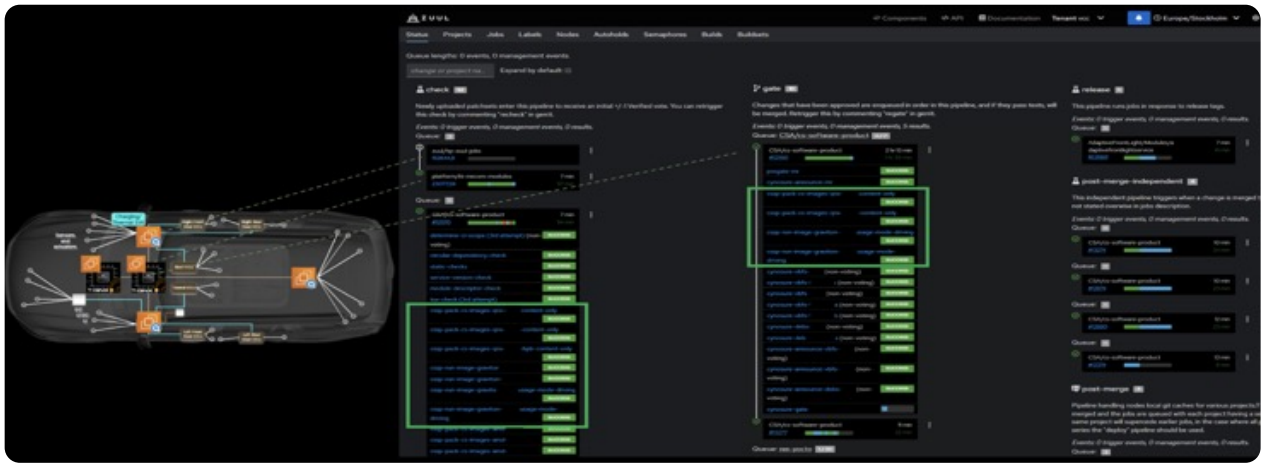
» 同じワークフロー、ツール、システム定義を維持

これにより、テスト環境を再構築することなく、**完全仮想開発からハードウェア検証までの柔軟な連続体**が実現されます。

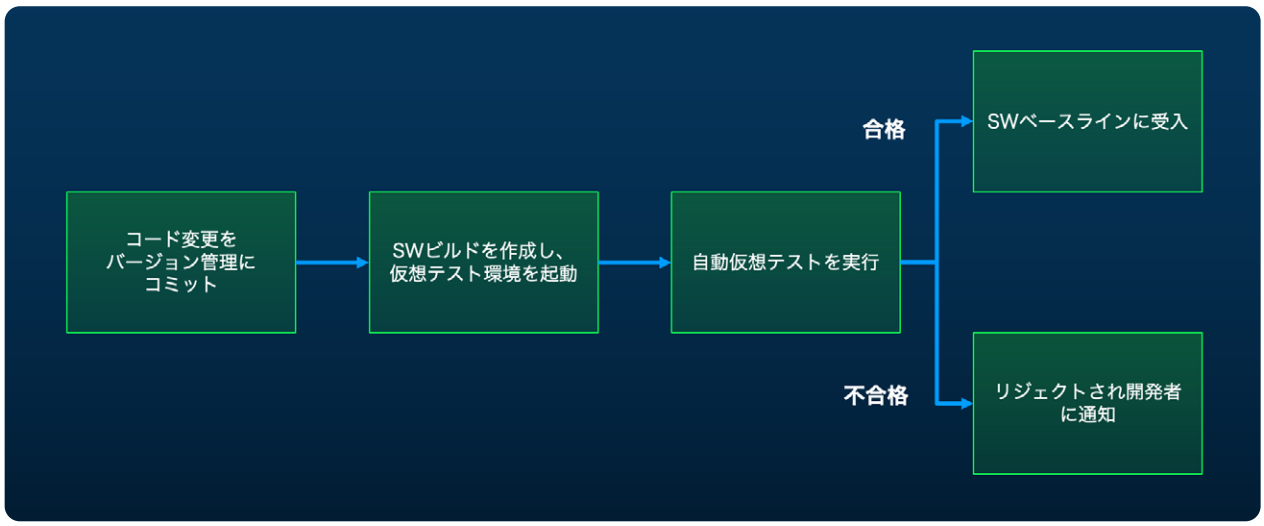
ユースケース

6.5 継続的インテグレーション（CI）パイプラインとの統合

全開発フェーズ・品質・スピード



OEM CI パイプラインに統合された RemotiveTopology



一般的な CI パイプラインの検証ロジック

目的

CI/CD ワークフローの一環としてシステムレベルのテストを可能にし、すべてのソフトウェアコミットが完全な仮想車両システムに対して自動的に検証されるようにします。

課題

数千人の開発者を抱えるプログラムでは、従来のシステムレベルテストがサポートする変更速度では、分散ソフトウェアスタックを検証できません——共有統合環境は希少であり、CI にはシステムレベルのゲーティングが欠如しています。

アプローチ

RemotiveTopology 環境が CI パイプラインに統合されます。コミットがバージョン管理に到達すると、仮想トポロジーが自動的にインスタンス化されます。個々の開発者がローカルで作成したテストが、中央パイプラインでそのまま実行されます。

結果

開発者は自分のマシンからシステムレベルのテストを実行できます。一貫した検証がすべてのチームで実行されます。不正なソフトウェアコミットは自動的に検出・拒否され、ハードウェアが関与する前にソフトウェア品質が大幅に向上します。

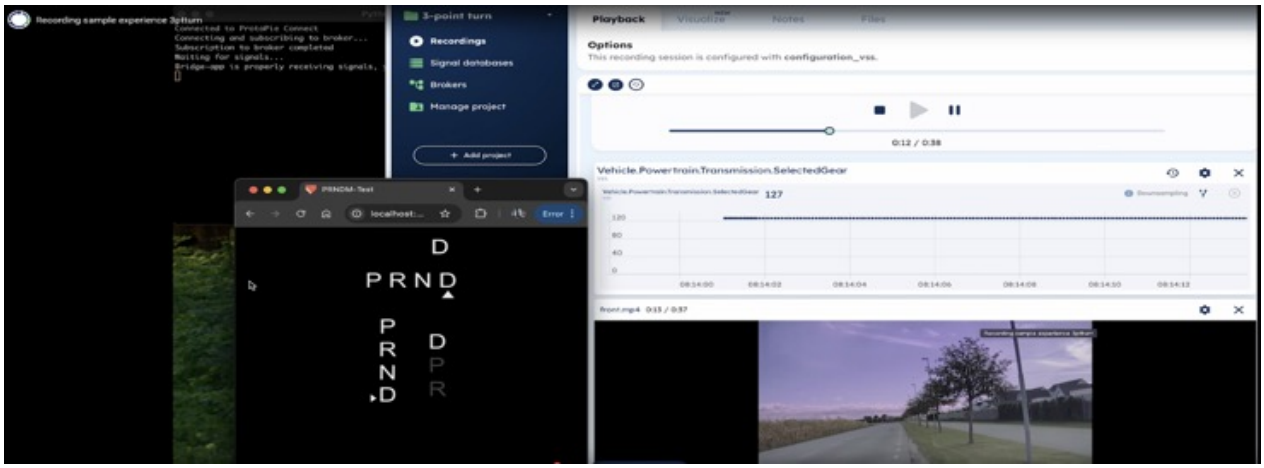
創出される価値

- 全開発者集団にわたる自動化されたシステムレベルゲーティング
- 統合への伝播前に不正コミットを検出。ある OEM では、この早期仮想テストプロセスにより約 30% のコード変更がリジェクトされたと報告
- 同じテスト資産がローカルと CI の両方で動作
- リグレッション欠陥の減少によるデプロイ頻度の向上

ユースケース

6.6 クラウドベース車両データを用いたプロトタイピング

早期開発・スピード・イノベーション



クラウドベースの車両信号データを ProtoPie ツールに供給――デザイナーが実際の車両データを使用して機能やデザインを検証できます。

目的

ユーザー中心のデザインにはリアルな車両データが必要ですが、ハードウェアプロトタイプは高価でイテレーションが遅く、デザイン・開発・エンジニアリングチーム間での共有が困難です。デザイナーは UX インタラクションの検証が難しく、デザイナーとエンジニアのギャップが拡大します。

アプローチ

実際の車両信号と同期されたビデオが共有クラウドリソースとして提供され、デザイナーと開発者が好みのツール（例：ProtoPie など）で直接リアルなデータを使用してプロトタイピングできます。

結果

高忠実度の UI/UX プロトタイピングがハードウェアなしで実現し、より高速なイテレーションと分野横断的な共有データ基盤が得られます。

創出される価値

- 物理車両なしで実際の車両信号に対して UI/UX を検証（ユーザーコメント：“I can test and validate assumptions, and provide valuable input earlier in the project”）
- ハードウェアプロトタイピングよりも高速なイテレーション
- 共有データ基盤上での多分野コラボレーション

ユースケース

6.7 ゾーンコントローラテスト：SIL からスケーラブルな HIL へ

中期→後期開発・コスト・スケーラビリティ

目的

テスト資産を再構築することなく、SIL から HIL への橋渡しを行い、ゾーンコントローラ向けのスケーラブルでコスト効率の高いハードウェアテスト環境を構築します。

課題

ゾーンコントローラ検証用の従来の HIL システムは高価で柔軟性に欠けます。SIL から HIL への移行には通常、環境の再構築とテストの再作成が必要です——そうでなければそのまま引き継がれるはずの作業です。

アプローチ

モジュラーハードウェアアダプター（例：Kvaser、NI）を介して、物理ハードウェアを既存の仮想トポロジーにインクリメンタルに導入します。SIL で作成されたテストはハードウェアの追加後もそのまま実行され、トポロジー定義は移行を通じて一貫性を保ちます。

モジュラーゾーンコントローラテスト——Kvaser との協業で開発——仮想ノードと物理ノードをスケーラブルなハイブリッドセットアップで統合。

結果

モジュラーでスケーラブルな HIL セットアップは従来のリグの数分の一のコストで構築でき、再設計なしにサブシステムからフルシステムテストまでスケールします。

創出される価値

- 従来のリグと比較して HIL コストを大幅に削減
- テスト資産を再利用しながら SIL から HIL へのスムーズな移行
- スケーラブルなセットアップ：開発者デスクレベルのテストから分散テストラックまで
- 仮想テストと物理テストにわたる一貫したワークフローによる高速イテレーション

拡張：オープンでモジュラーなハードウェアエコシステム

プラットフォームはオープンインターフェース上に構築されているため、チームが特定のベンダーのスタックにロックインされることはありません。標準プロトコル（CAN、LIN、イーサネット）とオープンブリッジング API（gRPC、Linux ツール）により、同じトポロジー内にあらゆるツールの組み合わせが共存できます——物理 I/O の隣に仮想 ECU、異なるベンダーのシミュレーションツールの並行稼働、Kvaser や NI などのハードウェアプロバイダーの必要に応じた統合。セットアップはオープン API と Python を通じて自動化でき、チームは DUT に適したツールの組み合わせを自由に選択し、周囲の環境を再構築することなく任意の部分を置き換えることができます。

ユースケース

6.8 軽量セットアップでの自動 ECU ソフトウェアテスト

中期開発・コスト・スケーラビリティ

目的

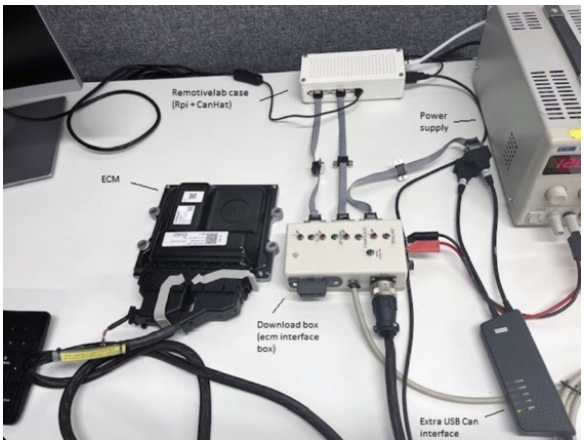
大規模な HIL インフラなしで、診断、サイバーセキュリティ、機能テストを含む幅広いテストタイプをサポートし、完全な CI 統合を備えた自動 ECU テストを実現します。

課題

多くの ECU テストシナリオはフル HIL リグを必要としませんが、自動化、CI 統合、幅広いテストカバレッジは必要です。従来のテストインフラは過剰（HIL）か不十分（デスクトップのみ）のいずれかでした。

アプローチ

RemotiveLabs ツールが軽量ハードウェア（例：Raspberry Pi）上で動作し、CAN インターフェースを介して ECU に接続され、リモートアクセスとスクリプティング用のオープン API が組み込まれています。同じセットアップをスケールで複製できます。



低コストハードウェアによる軽量・自動化 ECU テストセットアップ——単一リグから分散環境までスケール可能。

結果

完全自動化された ECU テストが低コストハードウェア上で実行され、リモート操作と簡易的な CI 統合が可能です。

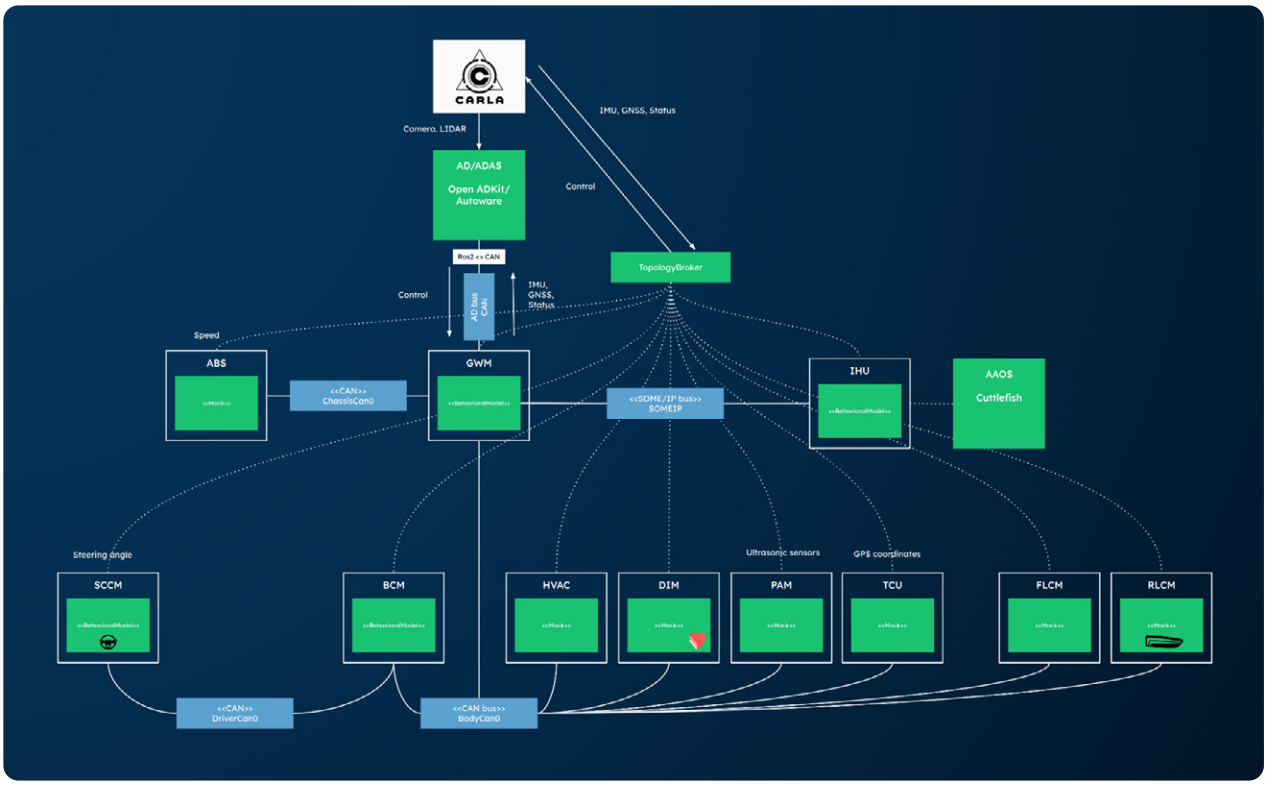
創出される価値

- 低単価かつ幅広いテストカバレッジでの自動化 ECU 検証（ユーザーコメント：“Raspberry Pi 使用時に最大 90% のコスト削減”）
- チーム間でのスケールと複製が容易
- 完全な CI 統合による高い自動化率

ユースケース

6.9 マルチドメインテスト：ADAS と IVI の統合

中期開発・品質・統合



マルチドメインの車両動作（ADAS シミュレーションと IVI の統合）を、早期開発段階からリアルな仮想 E/E アーキテクチャ内で検証

目的

複数のソフトウェアドメイン（ADAS、IVI など）間のインタラクションを統一された環境でテストします。

課題

ADAS と IVI 開発は通常、別々のツールと環境を持つサイロで行われ、フル車載ネットワークが含まれることは稀です。クロスドメインの課題は遅れて発見されることが多く、ハードウェア上でしか見つからないこともあり、コストのかかる統合のやり直しにつながります。

アプローチ

RemotiveTopology を使用して、フルの仮想 E/E アーキテクチャを SIL に導入します。ADAS シミュレーション（例：CARLA）と IVI（例：Cuttlefish 経由の Android）が同じ車両トポロジーのノードとして統合されます。ADAS の判断が車載ネットワークを通じて伝播し返されることで、シミュレータに依存しないクローズドループテストが可能になります。

結果

チームは、個別のドメインだけでなく、完全なシステム動作を SIL の段階で検証でき、ADAS、IVI、車載ネットワーク間の相互作用を可視化できます。

創出される価値

» 早期段階から ADAS と IVI を統合して検証し、必要な工数を削減

» クロスドメイン欠陥を修正コストが最も低い早期に発見

» システムテスト段階での統合やり直しを削減

» SIL、HIL、将来のプラットフォームで再利用可能な環境

ユースケース

6.10 AR/VR 開発のためのリアルタイムデータ統合

イノベーションフェーズ・研究・コラボレーション

目的

没入型の開発、テスト、検証のために、車両システムと AR/VR 環境間のリアルタイムインタラクションを実現します。

課題

車両信号、バックエンドソフトウェア、AR/VR ツールは異なる技術を使用しています。リアルタイム相互運用性の実現は困難であり、開発ツールとしての AR/VR の活用が制限されています。

アプローチ

開発者フレンドリーなフォーマットでのリアルタイムデータ合成とストリーミングが AR/VR プラットフォーム（例：Varjo XR）と統合され、車両信号、バックエンドロジック、没入環境間のリアルタイムデータ交換を実現します。

結果

AR/VR 環境が単なるプレゼンテーション層ではなく、開発ループの一部となります。車両データが AR/VR インタラクションを直接駆動し、開発、テスト、ステークホルダーとのエンゲージメントのための新しいワークフローが可能になります。

車両信号、バックエンドロジック、AR/VR をリアルタイムで接続することで、没入環境がデモ用の表示層ではなく、開発ループの一部となります。

創出される価値

» 没入環境でのリアルタイムかつインタラクティブなプロトタイピングと検証

» 車両動作からの即時フィードバックによる UX イテレーションの高速化

» シミュレートされたシナリオと実車両データの組み合わせが可能

» ドライバー行動とシステムインタラクションの理解の向上

6.11 ユースケースと開発フェーズ・価値の対応表

ユースケース	フェーズ	価値	主要メリット	KPI インパクト
初日からの E/E アーキテクチャ検証	研究・コンセプト	スピード・リスク軽減	ECU ソフトウェア開発開始前に提案 E/E アーキテクチャを実行可能なシステムとして検証	↓ 後工程のアーキテクチャ変更・ ↑ サプライヤーアラインメント・ ↑ コンセプトフェーズ欠陥検出
実車コンテキストを用いたインフォテインメント開発	早期開発	スピード・開発者生産性	車両やリグに依存せず、実際または模擬車両信号を用いた AAOS/HMI 開発	↓ デバッグサイクル時間・ ↑ 信号可視性・ ↑ 実環境課題の再現性
コアコンピュータテストのためのメカトロニックリムシミュレーション	早期開発	スピード・コスト	仮想車両コンテキストで単一 ECU を検証——実 DUT、仮想化された周辺——HIL リグ不要	↓ 統合リードタイム・ ↑ HIL 使用量・ ↑ 早期欠陥検出
マルチ ECU 機能検証	早期開発	スピード・開発者生産性	複数 ECU にまたがる機能のエンドツーエンドテスト、すべて仮想。ラップトップまたは CI からリグなしで実行	↓ フィードバックループ時間・ ↑ 開発者自律性・ ↑ 機能テストカバレッジ
CI パイプライン統合	全フェーズ	品質・スピード	継続的検証。1000 人以上の開発者にわたる自動化されたシステムレベルゲーティング	↑ テストカバレッジ・ ↓ リグレッション欠陥・ ↑ デプロイ頻度
クラウドベースプロトタイピング	早期開発	スピード・イノベーション	アクセス可能なリソースとして実信号を使用した UI/UX 検証	↓ プロトタイプサイクル時間・ ↑ イテレーション速度
ゾーンコントローラ（SIL → HIL）	中期→後期開発	コスト・スケーラビリティ	モジュラーでスケーラブルな HIL、仮想と物理のセットアップの混合。テスト資産の再利用	↓ HIL コスト・ ↓ ハードウェア依存・ ↑ テストスケーラビリティ
軽量 ECU テスト	中期開発	コスト・スケーラビリティ	低コストハードウェアでの完全自動 ECU テスト	↓ セットアップ単価・ ↑ 自動化率・ ↑ テスト頻度
マルチドメイン（ADAS + IVI）	中期開発	品質・統合	ドメインサイロの解消、早期クロスシステム検証	↓ クロスドメイン欠陥・ ↓ 統合やり直し
AR/VR リアルタイムデータ	イノベーション	イノベーション・コラボレーション	AR/VR への車両信号のライブ交換。効率的な開発ワークフローと可視化	↑ 開発者生産性・ ↑ ステークホルダーアラインメント

7. 自動車開発プログラムにおけるメリット

前述の章では、仮想システムレベル開発が現代の自動車ソフトウェアの統合課題にどのように対処するかを詳しく説明しました。プログラム意思決定者向けに、メリットを以下のように要約します：

1. **統合課題のより早期の検出。** 分散 ECU ソフトウェアは、物理ハードウェアが利用可能になるはるか前から相互作用でき、コンポーネントがまだ活発に開発されている段階で信号の不整合、インターフェースエラー、シーケンスの問題を表面化させます（第 2.2 章参照）。ある大手 OEM での顧客調査では、仮想エンドツーエンド統合テストがハードウェアベースのアプローチと比較して約 10 倍に加速されたことが示されています——時間単位ではなく分単位です。

2. **希少なハードウェアへの依存軽減。** 仮想トポロジーは開発者のワークステーション、CI サーバー、またはクラウドインフラ上で実行され、HIL ベンチを真に必要とする後工程のシナリオに解放します（第 2.5 章参照）。一般的な HIL リグは数千万円規模の設備投資を要し、仮想的に実行可能な機能テストに使用されている場合、リグもそれを使用するチームも最大の価値を生み出す活動に投入されていないことになります。

3. **スケーラブルな継続的インテグレーション。** システムレベルのテストがすべてのコミットで自動的に実行され、フルソフトウェアスタック全体のリグレッションを数週間ではなく数分で検出します（第 2.4 章参照）。

4. **チーム間コラボレーションの改善。** 共有された再現可能なトポロジー定義により、OEM チームと Tier1 サプライヤーに共通のシステム表現が提供され、組織の壁を越えた課題の再現とデバッグが容易になります（第 2.3 章参照）。

5. **オープンツール統合による柔軟性。** オープンインターフェースにより、チームは好みのテスト・デバッグツール（Pytest、Behave、Jupyter、Wireshark など）を使用でき、RemotiveStudio はブラウザ上で直接トポロジー閲覧とリアルタイム信号モニタリングを提供し、ベンダーロックインを回避しながら既存の投資を活用できます（第 5.4 章参照）。

6. **SDV 開発のための AI 対応基盤。** アーキテクチャ駆動型トポロジー生成、コンテナ化された実行、ハイブリッド仮想・物理セットアップが IaC として提供されます——バージョン管理され、再現可能で、コマンドラインツールや MCP API を通じて AI エージェントからアクセス可能——SDV プログラムがますます必要とする種類の基盤です（第 5.6 章参照）。

これらのメリットを総合すると、開発組織はシステムレベルの検証をライフサイクルのより早い段階にシフトし、希少な物理リソースへの依存を軽減し、自動車ソフトウェアワークフローを現代のソフトウェアエンジニアリングの手法に整合させることができます。

RemotiveTopology は現代の自動車ソフトウェア開発の基盤として、さまざまな開発チームや役割をサポートし、AI エージェントとの連携によりワークフローを加速します

8. Build vs. Buy: OEM が RemotiveTopology を採用する理由

自動車ソフトウェア開発がますます複雑化する中、OEM は戦略的な判断を迫られています：仮想開発インフラを社内で構築すべきか、それともそれに特化した企業から採用すべきか。

この問いは理論的なものではありません。大手 OEM はすでに社内でのシステムレベル仮想開発インフラの構築を試み、その結果として――能力が不十分であったために――RemotiveTopology のような専用プラットフォームへの移行を進めています。

本章では、ある大手欧州 OEM の経験を基に、社内ツール開発において直面した課題と、それが RemotiveTopology の採用にどのようにつながったかを検証します。

8.1 社内ツール開発の課題

この OEM は当初、仮想開発インフラを社内エンジニアリングチームで構築しようとしましたが、いくつかの根本的な課題に直面しました。

- 第一に、社内ツールは OEM の特定のパートナーシップやサプライヤーとのワークフローに合わせて設計されたため、真にオープンなエコシステムとはなり得ませんでした。このため、新しいサードパーティツールやテクノロジーとの統合が困難で、長期的なイノベーションとスケーラビリティが制約されました。
- 第二に、これらのツールは開発ライフサイクル全体をカバーできませんでした：モックベースの早期環境、中期の SIL、後期の HIL がすべて断片的な取り組みとなり、早期開発から量産レベルのテストまでの継続的な検証の実現が困難でした。
- 第三に、数千人の開発者が分散するグローバルチームと複数のサプライヤーネットワークへのスケーリングは、独自ツールでは実用的でないことが判明しました。
- 第四に、自動車開発が CI/CD パイプライン、クラウドネイティブインフラ、IaC を採用するにつれ、社内開発ツールは自動化、スケーラビリティ、統合の容易さにおいて常に後れを取りました。
- 第五に、社内ツールは一度きりの投資ではありません。車両プラットフォームが進化し新しい技術がスタックに加わるにつれ、社内ツールは量産中の旧アーキテクチャとの互換性を維持しながら、現行プログラムをサポートするために継続的に更新する必要があります。これにより年々増大する複合的な保守負担――車両ソフトウェア開発に投入すべきエンジニアリング能力と直接競合する経常的なコスト項目――が生まれます。

8.2 現代的な仮想開発プラットフォームの要件

上記の失敗モードから、この OEM のプラットフォーム要件として具体的な条件が形成されました。ソリューションは、すべてのチームが ECU ドメインやサプライヤーの境界を越えて同時に作業できる、一貫したインターフェースと通信モデルを持つ単一の統一車両トポロジーを提供する必要がありました。モックから vECU バイナリ、物理ハードウェアまでの完全なコンポーネント忠実度スペクトルをサポートし、開発の進行に伴いテスト資産を再設計なしに再利用できる必要がありました。ネイティブ CI/CD 統合は必須条件であり：この OEM では Zuul ベースのパイプラインとの互換性が明示的な要件であり、クラウドベースの実行と再現可能な環境により、開発者が自分のラップトップからシステムレベルのテストをトリガーできることが求められました。最後に、プラットフォームはオープンエコシステムの哲学に整合する必要がありました：オープン標準、モジュラーアーキテクチャ、ベンダーロックインの回避を重視し、長期的な柔軟性を確保します。

8.3 RemotiveTopology の採用

これらの要件に基づき、この OEM は社内ツールから RemotiveTopology へ移行しました。プラットフォームは社内開発の取り組みを悩ませていたコア制限のいくつかに対処しましたが、大規模なツールチェーン移行に伴う組織変革も必要でした。

最も即座に改善されたのは、開発ライフサイクル全体にわたるテスト資産の継続性でした。社内ツールでは早期のモックベース開発、SIL、HIL が断片的な環境だったのに対し、RemotiveTopology では同じテストケースを開発者のワークステーションから中央 CI パイプラインで実行されるハードウェア検証まで、すべてのステージで実行できました。これにより、チームがテスト環境間を移動するたびに必要だったやり直しが排除されました。

共有仮想トポロジーはまた、分散チームのコラボレーション方法も変えました。物理統合ベンチへのアクセスを競い合う代わりに、異なるサイトの開発チームがそれぞれ同じ車両アーキテクチャの独自インスタンスを実行できました。すべてのチームが同じシステム表現と同じソフトウェアベースラインに対して作業するため、チーム間の依存関係がより管理しやすくなり、組織の壁を越えた課題の再現とデバッグが容易になりました。

CI 統合も、プラットフォームが直接対処したもう一つの重要な要件でした。クラウドベースの実行により、CI パイプラインの一部として仮想車両システムを自動的にインスタンス化でき、自動検証が統合フェーズに到達する前にリグレッションを検出しました。テスト実行は、共有ハードウェアの予約を必要とするスケジュールされたイベントではなく、開発ワークフローの日常的なステップとなりました。

RemotiveTopology は実際の車両通信定義——CAN、LIN、SOME/IP——を使用するため、仮想環境は量産動作を忠実に反映します。ECU はモック化され、開発の成熟に伴いより高忠実度の実装に段階的に置き換えることができ、ECU 間で交換される信号は量産車両と同一です。これにより、早期の検証が単なる指標的なものではなく、有意義に正確なものとなりました。

最後に、プラットフォームのオープンアーキテクチャは、OEM のオープン標準とモジュラーツールチェーンの嗜好に整合しました。オープンソースのビルドインフラとオープン API のサポートにより、RemotiveTopology はチームにゼロから新しいプロプライエタリワークフローの採用を要求するのではなく、組織の既存エコシステムに統合できました。

8.4 OEM 向けの主要な教訓

この OEM の経験は、仮想システムレベルツールへのアプローチを検討している自動車開発組織にいくつかの教訓を提供しています。

社内インフラの構築は、当初見えるよりもコストがかかります。明確なプラットフォーム戦略がなければ、社内ツールはサイロで成長する傾向があり、一貫性のないインターフェース、ドメイン間での限られた再利用、増大する保守コストを生み出します。また、サードパーティ統合にも苦戦する傾向があります：ツールが OEM の既存のパートナーシップ

やワークフローに合わせて形作られるため、真にオープンなエコシステムとなることは稀であり、長期的なイノベーションとスケーラビリティが制約されます。

システムレベルの仮想化を適切に構築することも、本質的に困難です。マルチレベルの仮想化、分散システムオーケストレーション、CI/CD 統合、大規模コラボレーションを単一の統合されたプラットフォーム内でサポートするには、持続的なエンジニアリング投資と深い専門性が必要です。これを試みる組織は、通常、車両機能開発に必要なエンジニアリング能力と直接競合することになります。

一方、RemotiveTopology のような専用プラットフォームを採用することで、OEM は生産的な仮想開発により早く到達し、継続的なツール保守負担を軽減し、開発ワークフローを現代のソフトウェアエンジニアリングの手法に整合させることができます。この事例からの根本的な洞察はシンプルです：仮想開発インフラは競争上の差別化要因ではありません――その上で動作する車両ソフトウェアこそが差別化要因です。この区別を認識する組織は、最大の価値を生み出す領域にエンジニアリングの取り組みを集中させることができます。

この OEM における社内ツール開発から RemotiveTopology への移行は、より広い業界トレンドを示しています：仮想システムレベル開発は SDV の基盤インフラとなりつつあります。

複雑性が増し続ける中、スケーラブルで再利用可能な、開発者中心の環境を提供する能力が OEM にとって重要な差別化要因となるでしょう。