組み込みシステムのテスト
― 適切なテスト単位について

Vincent Encontre

Rational Software ホワイト・ペーパー

TP 317, 11/01

Rational®
the software development company

目次

概要

このホワイト・ペーパーでは、組み込みシステムのテストについて紹介します。また、組み込みシステムの問題がテストのプロセスや技術にどのような影響を与えるか、Rational Test RealTime によってこれらの問題がどのように解決されるかについて説明します。

共通の概念に関する定義

最初にいくつかの定義を行って、共通の概念について合意しておきます。

まず、「テスト」について定義します。テストとは、アプリケーション (そのコンポーネントを含む) の動作、性能、堅固さが、期待される基準に適合するかどうかを確認する、一つにまとめられたプロセスです。主な基準の 1 つは、通常は暗黙的ですが、アプリケーションに可能な限り障害がないことです。したがって、期待される動作、性能、堅固さは形式的に記述し、計測可能にする必要があります。デバッグは、文字どおり、障害 (バグ) を取り除くことを意味し、テスト・プロセスの一部と見なされます。

続いて、「組み込みシステム」を定義します。組み込みシステムについて正確に定義することは難しく、多くの議論がありますが、ここではいくつかの例を示します。組み込みシステムは、私たちの生活に浸透している「インテリジェント」な装置の中に存在しています。例えば、携帯電話やそれらを支える無線システム、PalmPilot、電子メールを転送しているインターネット・ルーター、大画面のホーム・シアター・システム、航空交通管制ステーション (そして、そこで監視される予定より遅れている飛行機) などがあります。現在では、これらの装置の価値の 90% はソフトウェアが構成しています。

図 1: 組み込みソフトウェアに向かう世界

すべてではありませんが、ほとんどの場合、組み込みシステムは「リアルタイム」なシステムです。「リアルタイム」と「組み込み」という用語は、相互に置き換えて使用される場合があります。リアルタイム・システムは計算の正しさが、論理的な正しさだけでなく、結果を出す時間にも依存するシステムです。システムの時間に関する制約に適合しない場合は、システム障害が発生することになります。安全重視システムでは、障害の発生は許されません。**したがって、「組み込みシステムの時間的な制約をテストすることは、機能的な動作をテストすることと同じく重要です。**

組み込みシステムのテスト・プロセスは、ほかの種類のアプリケーションの多くで使用されるテスト・プロセスと似ていますが、組み込み分野における重要な問題の影響を受けます。

- アプリケーションの開発プラットフォームと実行プラットフォームの相違
- 多様な実行プラットフォームとクロス開発環境
- 広範囲の配置アーキテクチャー
- さまざまな実装方法の共存
- 実行プラットフォームにおけるリソースと時間の制約
- 明確な設計モデルの不足
- 品質と保証の基準に関する問題

より詳細な情報については、後で説明します。

これらの問題は、組み込みシステムのテストと評価に大きく影響を与えます。組み込みシステムのテストが難しく、テストが現在の開発作業で最大の欠点の 1 つである理由も示しています。最近の研究 (「参考資料」[R1] を参照) によると、組み込みシステム開発プロジェクトの 50% 以上でスケジュールに数カ月の遅れが生じています。また、開発の 50% 以上の労力をテストに費やした場合でも、機能と性能の両方について予測と実際の差が 20 % 以内に収まった設計は 44% にすぎません。これらのことは、テストの難しさを考えると無理もありません。

このホワイト・ペーパーでは、次の内容を説明します。

- 一般的なテストの反復を詳しく説明します。ここから、望ましい最小限のテスト技術を導き出します。
- この反復をインスタンス化し、複雑な組み込みシステムのテストを処理します。 これらの考察に基づいて、理想的なテスト技術に機能を追加します。
- 組み込みシステムの開発とテストを困難にしている原因を説明します。
- テスト技術で実行される機能のリストに、これらの問題をどのように追加するかを評価します。

この理想的な技術の大部分を実装している製品例として、Rational Test RealTime を使用します。

一般的なテストの反復

テスト単位の準備

どのようなテストでも最初に必要な手順は、テスト単位を識別することです。「単位」という用語は、コンポーネント、ユニット、システムなどの一般的な定義を持たない用語を避けるために使用しています。例えば、UML では、コンポーネントは、制御スレッド (タスクや UNIX プロセス) を持つ個々のアプリケーションを表します。また、「単位」という用語は、粒度という意味でも使用しています。これは、1 行のコードから大規模な分散システムまで、テスト可能なさまざまな要素を表します。

テストする単位を識別し、テスト可能な単位またはテスト単位 (GuT: granule under test) に変換します。この手順は、テスト単位を「スタブ」や (場合によっては) 「アダプター」によって独立させ、環境から分離することで構成されます。スタブは、テスト単位とアプリケーションの残りとの間で、双方向のアクセスをシミュレートするコードです。

続いて、テスト・ドライバーを作成します。テスト・ドライバーは適切な入力で GuT を呼び出し、出力情報を評価して、期待される応答と比較します。テスト・ドライバーが GuT を呼び出すには、アダプターが使用されます。呼び出しと評価は、GuT のゲートを通した特定のパスに従います。これらは制御と観測ポイント (PCO) と呼ばれ、電気通信業界で使用される用語です。PCO は、GuT の境界または内部に設定できます。

C 関数のテスト単位に対する PCO は、次のようになります。

- 観測ポイントがテスト単位の内側にある場合:関数の特定のコード行のカバレッジ

- 観測ポイントがテスト単位の境界にある場合:関数が返すパラメーター値
- 制御ポイントがテスト単位の内側にある場合:ローカル変数の変更
- 制御ポイントがテスト単位の境界にある場合:実パラメーターによる関数呼び出し

スタブとテスト・ドライバーは、アプリケーションを構成する別の部分である場合もあります (利用可能な場合)。必ずしも、C 関数や C++ クラスをテストするためだけに作成する必要はありません。GuT は、アプリケーションのほかの部分からアクセスまたは呼び出されて、スタブまたはテスト・ドライバーとして動作することができます。スタブとテスト・ドライバーは、GuT のテスト・ハーネス環境を構成します。

テスト・ケースの記述

テスト・ケースを記述することは、次の点を解決するということです。

- 適切な PCO — 機能、構造、負荷など、実行するテストの種類に依存します。
- その利用方法 — 送信される情報、期待される情報、その順序を決定します。

テストの種類、送信される情報と期待される情報は、GuT に対して設定される要件によって決まります。安全重視システムの場合は、開発プロセスに形式的で正確な要件が不可欠です。形式的な要件はテストに対する重要な動機付けとなりますが、システムの重要な欠陥を発見するテストが明示的に示されているとはかぎりません。形式的な要件を使用し、特定の要件が不足する場合、重要度の低いアプリケーションでは、テスト担当者は適切なテストのセット (「テスト・アイデア」と呼ばれます) を行うよう考慮して、GuT をテストするためのいくつかの要件を記述する必要があります。これらの要件は、利用可能な PCO を使用する形式的なテスト・ケースに変換される必要があります。ここでは、わかりやすくするために、「形式的」という用語で「実行可能」であることを表しています。

通常、要求はそれ自体が形式的でないため、形式的なテスト・ケースには変換されません。テスト・ケースが要求を正確に反映していないために、この変換プロセスにエラーが混入する場合があります。仕様を表現する言語は、UML の導入によってより形式的になります。 形式的な要求に基づくテスト・ケースを表現することで、変換に伴う危険を避けることができます。Rational QualityArchitect と Rational Test RealTime の一部に、これらのモデルに基づくテスト技術の使用例があります。

残念ながら、特に組み込みシステムの場合は、すべての要求を UML で表現できるとは限りません。テスト・ケースを形式的に作成する最も一般的な方法は、C や C++ などのプログラミング言語を使用することです。C や C++ は広く知られています (したがって、学習曲線は短縮されます) が、PCO の定義や予測される GuT の動作などのテスト・ケースのニーズを考慮するには適していません。したがって、総合的なテスト・ケースを作成することはできません。この問題は、特定の高レベルなテスト言語で設計することによって処理されています。これらの言語は、データに重点を置いたテストやトランザクションに基づくテストなど、特定のテスト領域にうまく適応します。Rational Test RealTime は、ネイティブな 3GL と専用の高レベル・スクリプト言語を組み合わせ、短縮された学習曲線と効率的なテスト・ケースの作成という両方の利点を利用しています。

テスト・ケースを作成する効果的で生産性の高い方法には、セッション・レコーダーを使用する方法もあります。GuT が (手動または将来的な環境によって) 呼び出されるときに、特定の観測ポイントは GuT の入出力情報を記録します。これらは適切なテスト・ケースに自動的に変換され、後で再現できます。Rational Rose RealTime に、これらのセッション・レコーダーの例があります。 モデルの実行により、トレースの実行を反映する UML シーケンス図が生成されます。 作成されたシーケンス図は、Rational QualityArchitect RealTime でテスト・ケース モデルとして使用されます。

各テスト・ケースでは、テストを実行できるように Gut を特定の開始状態にする必要があります。テスト・ケースのこの部分は、プリアンブルと呼ばれます。どのような結果になっても、効果的なテストの最後では次のテスト・ケースを実行できるように、テスト・ケース・スクリプトで GuT を最終的な安定状態にする必要があります。テスト・ケースのこの部分は、ポストアンブルと呼ばれます。

テストの配置と実行

作成されたテスト・ケースは、テスト・ドライバーとスタブの (操作に対しての) 情報部分として変換され統合されます。スタブの記述はテスト・ケースに欠かせない部分であることに注意してください。テスト・ケースは、テスト・ハーネスの実行中に実行されます。

テスト結果の観測

テストの実行結果は、観測ポイントを通して監視できます。

テスト単位の境界では、一般的な観測ポイントは次のようになります。

- 関数または受信メッセージから返された「パラメーター」
- *グローバル変数の値*
- 情報の順序とタイミング

テスト単位の内部では、一般的な観測ポイントは次のようになります。

- ソース・コード・カバレッジ。GuT のどのソフトウェア部分が実行されたかについての詳細が得られます。
- 制御グラフ。実行されるさまざまな論理分岐に従います。
- 情報フロー。GuT の各部分間での情報のやり取りを、時間に関して視覚化します。一般的に、このようなフローは Rational Test RealTime の UML シーケンス図として表されます。
- リソースの使用状況。 GuT の各部分にかかる時間、メモリー・プール管理、イベント処理のパフォーマンスなど、さまざまな機能外情報を示します。

これらのすべての観測は、1 つのテスト・ケースに対して収集したり、テスト・ケースのセットに対して集計することができます。

次の手順の決定

すべてのテスト・データが収集されて統合されると、1 つ以上のテスト・ケースが失敗したか、すべてのテストを通過したかの 2 種類の結果になります。

テスト・ケースは、さまざまな理由で失敗します。

- 要求への不適合 (クラッシュしないという暗黙的な要件を含みます) — 実装に戻るか、さらに悪い場合は設計に戻って、GuT の問題を修正する必要があります。
- テスト・ケースの誤り — この状況は、予想以上に頻繁に発生します。テストは、ソフトウェアと同じように、最初から予測どおりに動作するとは限りません。テスト・ケースを変更して問題を修正します。
- テスト・ケースを実行できない — ソフトウェアと同様に、すべてが正しいと思われる場合でも、テスト・ハーネスを GuT に配置したり、開始または接続できない場合があります。

すべてのテストが通過した場合は、次のような処理を考えます。

- テストの再評価 — テスト・プロセスの初期の場合、テストの値と目標が疑われる場合もあります。特に開発の初期の段階では、テストでは問題が発見されるものと想定されます。
- テスト・ケースの追加 — GuT の信頼性が向上します。信頼性は要求の一部であるという異論もあり、それは正しいと言えるでしょう。しかし、多くの場合、信頼性のレベルはテスト・ケースのセット全体によって GuT のカバレッジのレベルに直接関連付けられています。

- カバレッジのうち、コード・カバレッジを最も広範囲に使用する — Rational Test RealTime で実装されます。コード・カバレッジに基づくテストにより、要求で同意されるレベルまでカバレッジ・レベルを上げる追加テスト・ケースを定義できます。テストのこの部分は、構造テストと呼ばれます。テスト・ケースは GuT の内容に基づき、要求には直接依存しません。
- テスト単位を合成してテスト範囲を拡大する — この次のトピックで説明しているように、システムのより大きな部分に対して、この一般的なプロセスを適用します。

いつテストを終了するか

この問題は、ソフトウェアの開発者にとっての永遠の課題であり、ここで解決すべき目標ではありません。しかし、テスト中のシステムがどれほど安全性を重視するかを考慮するために、発見的な方法を使用することができます。安全重視のシステムと考えることができるかどうかに注目します。安全重視でないシステムの場合は、いくらか主観的な基準に基づいてテストを終了できます。 例えば、市場への投入時期、予算、「それで十分」 の考え方などです。一方、安全重視のシステムでは失敗は許されません。テストの終了は、このような基準では決定できません。後の「品質と保証の基準に関する問題」で、この問題に対する推奨事項が説明されています。

一般的なテスト技術の要件

前に説明した一般的なテストの反復から、テスト・ツールに必要な最小限の機能について推測することができます。ツールには次の機能が必要です。

- GuT の定義と分離の支援
- テスト・ケースの表記法の提供 (PCO の定義、GuT に送られる情報、期待される GuT からの情報の定義、プリアンブル/ポストアンブルの定義をサポートする、3GL または視覚的な高レベル・スクリプト)
- 要求またはテスト・アイデアから、テスト・ケースを正確に引き出す支援
- セッション・レコーダーを使用してテスト・ケースを実装するための代替方法の提供
- テスト・ケースの配置と実行のサポート
- 観測結果のレポート
- 成功または失敗の評価

Rational Test RealTime はこれらの機能をサポートし、これらの要件だけでなく組み込みシステム分野の複雑なシステムのテストも処理します。

複雑なシステムのテストにおける 6 つの段階的手順

複雑なシステムの一般的なアーキテクチャーと実装

組み込みシステムは、単純な 8 ビットのマイクロコントローラーから、マルチプロセッサー・プラットフォームで構成される大規模な分散システムまで、極端に異なるアーキテクチャーで構成される複雑なシステムです。ただし、これらのシステムの 3 分の 2 は、市販または社内開発のリアルタイム・オペレーティング・システム (RTOS) 上で動作し、RTOS のタスクやプロセスに広がるスレッドの概念を実装します (スレッドは、独立した制御のフローのテスト単位です)。UML では、この概念はコンポーネントと呼ばれます。また、RTOS で管理されるタスク セットを実行する独立した処理単位は、ノードと呼ばれます。ノード間の通信は、通常、TCP/IP などのメッセージをやり取りするプロトコルを使用して実行されます。

組み込みシステムの開発者の大部分は、プログラミング言語に C、C++、Ada、Java を使用しています (2002 年の調査では、70% の開発者が C、60% が C++、20% が Java、5% が Ada を使用しています。「参考資料」[R1] を参照してください)。組み込みシステムでは、複数の言語が使用されることは珍しくありません。特に C と C++ や、C と Java の組み合わせがよく使用されます。C は、プラットフォームの細部に対してより効果的で密接であると考えられます。Java や

C++ は、オブジェクト指向の概念によって、おそらくより生産的であると考えられます。ただし、組み込みシステムのプログラマーは、オブジェクト指向の愛好家でないことにも注意してください。

組み込みシステムにおけるテスト単位は、次のいずれかになります。ここでは、複雑さの簡単な順に示しています。

- C 関数または Ada プロシージャー
- C++ または Java クラス
- C または Ada のモジュール (のセット)
- C++ または Java クラスのクラスター
- RTOS タスク
- ノード
- 完全なシステム

最小規模の組み込みシステムでは、完全なシステムは C モジュールのみで構成され、RTOS 関連のコードは統合されません。最大規模のシステム (分散システム) では、ネットワーク・プロトコルによってさらに複雑さが増します。

次に、この一般的なアーキテクチャーがさまざまなテスト手順に与える影響を示します。

段階的な 6 つのテスト手順

テスト単位の種類に応じて、また後で説明する業界での一般的な方法に従って、アプリケーションの動作、性能、堅固さが期待される基準を満たしているかどうかを確認するには、6 つのテスト手順が必要です。これらの手順は次のとおりです。

- ソフトウェア単体テスト
- ソフトウェア統合テスト
- ソフトウェア有効性テスト
- システム単体テスト
- システム統合テスト
- システム有効性テスト

ソフトウェア単体テスト

GuT は、分離された C の関数または C++ のクラスです。GuT の目的に応じて、テスト・ケースは次のいずれかで構成されます。

- データに重点を置いたテスト — 関数パラメーター値に与えるデータを、広範囲に変動させます。
- シナリオに基づくテスト — C++ メソッドの呼び出しシーケンスを変えて、要求で見つかったすべてのユースケースを実行します。

観測ポイントは、戻り値のパラメーター、オブジェクト・プロパティーの評価、ソース・コード・カバレッジです。単体テストにはホワイトボックス・テストが使用されるので、テスト担当者は、GuT の内容について熟知している必要があります。単体テストは、開発者の責務です。

複雑な組み込みシステムでは小さなエラーを追跡するのは困難であるため、単体テスト・レベルで、これらのエラーを見つけて取り除く作業を行う必要があります。

ソフトウェア統合テスト

GuT は、関数のセットまたはクラスのクラスターです。統合テストの中心となるのはインターフェースの検証です。同じ種類の制御ポイントが、データに重点を置いたメイン関数呼び出しやメソッド呼び出しのシーケンスをテストしている単体テストに適用されます。 観測ポイントでは、情報フローの図を使用して、低レベルのテスト単位間のやりとりに焦点を置きます。

GuT が意味を持ち始めると (エンドツーエンドの各テスト・シナリオを GuT に適応できるようになると)、すぐに性能テストを実行できます。これは、アーキテクチャーの有効性に関する適切な指標になります。機能テストについては、早ければ早いほど有効です。以降の各手順には、性能テストが含まれます。この手順でも、ホワイトボックス・テストが使用されます。ソフトウェア統合テストは開発者の責務です。

ソフトウェア有効性テスト

GuT は、コンポーネント内のすべてのユーザー・コードです。このテストは、ソフトウェア統合の最終段階と考えることができます。ユースケースは、エンド・ユーザーのシナリオに近づき、実装の詳細から離れていきます。GuT はシステム全体で重要な部分であるため、観測ポイントにはリソースの使用状況に関する評価が含まれます。ここでも、この手順をホワイトボックス・テストとして考えます。ソフトウェア有効性テストも開発者の責務です。

システム単体テスト

GuT は、完全なシステム・コンポーネントであり、ソフトウェア有効性テストでテストされたユーザー・コードと、すべての RTOS とプラットフォームに関する部分 (タスク処理メカニズム、通信、割り込みなど) となります。制御ポイントのプロトコルは、関数やメソッドの呼び出しではなく、例えば RTOS メッセージ・キューを使用して送受信されるメッセージです。

メッセージ送受信の方法に見られる対称性から、テスト・ドライバーとスタブの区別は、この段階では不適切であると考えられます。ここでは、これらを「仮想テスター」と呼びます。それぞれは、GuT に対する別のシステム・コンポーネントとして置き換え、動作させることができます。「シミュレーター」や「テスター」は、「仮想テスター」と同じ意味です。仮想テスターの技術は、多数の RTOS やネットワーク・プロトコルに適合するように用途が広いものとなります。これ以降、テスト・スクリプトは GuT を目的の初期状態にした後、メッセージのサンプルを順序どおりに生成します。 受信したメッセージは、期待されるメッセージの内容と比較し、タイミングの制限と受信日を比較して検証します。テスト・スクリプトは、さまざまな仮想テスターに配置されます。システム・リソースが監視され、組み込みシステムの実行を支えるシステムの能力が評価されます。この手順から、テスト方法としてグレイボックス・テストが使用されます。必要となるのは、GuT に対するインターフェースの知識だけです。システム単体テストは、組織に応じて、開発者または専任のシステム統合チームの責務となります。

システム統合テスト

GuT は、単一のノード内のコンポーネントのセットから、最終的には、すべてのシステム・ノードを含む分散ノードのセットまでです。PCO は、RTOS イベントやネットワーク・メッセージなどの、RTOS やネットワーク関連の通信プロトコルです。コンポーネントに加え、仮想テスターもノードのロールになります。ソフトウェア統合の場合と同様、焦点はさまざまなインターフェースを検証することです。テスト方法として、グレイボックス・テストが使用されます。システム統合テストは、システム統合担当チームの責務です。

システム有効性テスト

GuT は、最終的に完全な組み込みシステム全体になります。この最終的な手順の目標は次のとおりです。

- *エンド・ユーザーの機能要求を満たす。*エンド・ユーザーは、電気通信ネットワークのデバイス (例えば、組み込みシステムがインターネット・ルーターの場合) か人間 (システムが消費者デバイスの場合)、またはその両方 (エンド・ユーザーが管理できるインターネット・ルーター) になります。
- *負荷や堅固さなどの、最終的な機能外テストの実行。*仮想テスターは、負荷をシミュレートするように複製したり、システムに障害を起こすようにプログラムできます。
- *ほかの接続機器との相互運用性の確認。*適用可能な相互接続標準への適合を確認します。

これらの目標を詳細に説明することは、このホワイト・ペーパーで扱う内容の範囲を超えています。これらの手順では、ブラック・ボックス・テストが使用されます。テスト担当者は、繰り返されるユースケースと危険なユースケースの両方に集中します。

6 つの手順の順序の決定

これらの 6 つの段階的な手順をどのように使用するかを考えます。次の項目を決定するための基準は数多くあります。

- これらのすべての手順をシステムに適用するかどうか
- 選択した手順のすべてをシステムの全部に実行するか、一部に対してだけ実行するか
- 選択した手順を、選択した部分に適用する順序

この基準の収集に基づく決定は、開発中の組み込みシステムの種類に大きく左右されます。すなわち、安全性を重視するかどうか、市場投入までの時間、配置数の規模などによります。この選択プロセスを支援するガイドラインが、今後作成される予定です。これらのホワイト・ペーパーには、Rational Developer Network を通じてアクセスできるようになります。

これらのテスト手順を処理する方法として、これらを 1 つのモデルとして融合することもできます。有効性テストは、より大きな GuT の単体テストと考えることができます。統合は、有効性テスト中にも確認できます。GuT へのアクセス方法、挿入する PCO の種類とその場所、テスト・ケースから (おそらく、非常に離れた) GuT の特定の部分を目標にする方法に関する問題です。これらについては、別の機会に議論します。

複雑なシステムのテスト技術に対するその他の要件

複雑な組み込みシステムのテストを処理するには、テスト技術に次の機能を追加する必要があります。

- *複数の種類の制御ポイントの管理* — GuT を呼び出すには、関数呼び出し、メソッドの呼び出しとメッセージの引き渡しや、Ada、C、C++、Java などの異なる言語を通じたリモート・プロシージャー・コール (RPC) を使用します。
- *さまざまな観測ポイントの提供。*例えば、パラメーターとグローバル変数の検査、アサーション、情報、管理パスの追跡などです。または、コード・カバレッジの記録やリソースの使用状況の監視です。これらの各観測ポイントでは、予測に対して実際の結果を評価する機能を提供します。

テスト・プロセスとテスト技術に対する組み込みシステムの問題の影響

ここでは、組み込みシステムに特有の問題に注目し、テスト・ツール (Rational Test RealTime) で使用されるテスト技術に、これらの問題がどのような影響を与えるかを評価します。

アプリケーションの開発プラットフォームと実行プラットフォームの相違

組み込みシステムの定義の 1 つは、次のようになります。

> *組み込みシステムは、アプリケーションの配置先や対象プラットフォームと異なるプラットフォームで設計する必要があるソフトウェア・システムです。*

開発側の視点では、通常、プラットフォームは Windows、Solaris、HP-UX などのオペレーティング・システムを意味します。ほかの IT システム分野に比べると、組み込みシステムの分野では UNIX と Linux ユーザーの比率が高くなっています ([R1] では 40% となっています)。ターゲット側では、プラットフォームはこのホワイト・ペーパーの最初に示したような装置を表します。

なぜ制限があるのでしょうか。ターゲット・プラットフォームはエンド・ユーザー (人間またはほかのデバイス・セット) に対して独占的に最適化され構成されているため、開発を行うために必要なコンポーネント (キーボード、ネットワーク、ディスクなど) を備えていません。

プラットフォームが 2 つ存在するという問題に対処するために、テスト・ツールには、開発プラットフォームから実行プラットフォームに可能な限り透過的で効率的にアクセスする方法が必要になります。実際には、これらのアクセスの複雑さは、

ユーザーから見えなくする必要があります。アクセスには、テスト・ケース情報のダウンロード、テストの実行のリモート監視 (開始、同期、停止)、テストの結果と観測のアップロードが含まれます。Rational Test RealTime では、すべてのターゲット・プラットフォームへのアクセスは Target Deployment 技術によって制御されます。

Rational Test RealTime は Windows、Solaris、HP、Linux 上で動作する、デバイス、組み込みシステム、インフラストラクチャー業界の主要企業で使用されている開発プラットフォームです。

多様に展開する実行プラットフォームとクロス開発環境

実行プラットフォームは、単純な 8 ビットのマイクロコントローラー・ベースの基板から、大規模な分散ネットワーク・システムまで多岐にわたります。すべてのプラットフォームで、さまざまなチップやシステムのベンダーに合わせて、異なるアプリケーション開発用ツールが必要になります。同じ組み込みシステム中で複数のプラットフォームを使用することも一般的になっています。

一般的に、このような環境における開発は、「クロス開発環境」と呼ばれます。さまざまな実行プラットフォームがあることは、対応するコンパイラー、リンカー、ローダー、デバッガーなどの開発ツールのセットが、多数存在していることを意味します。

その結果として、まず、観測ポイントの技術はソース・コード・ベースのみになります。Rational PurifyPlus ファミリーで使用され、簡単なネイティブ・コンパイラーのセットで利用可能なオブジェクト・コード挿入 (OCI: Object Code Insertion) 技術と対照的に、Rational Test RealTime では、ソース・コードのインスツルメンテーションを行って数値的な要素を処理します。この技術に関する 10 年の経験から、効率の高いコードのインスツルメンテーションが可能になりました。

実行プラットフォームの多様性は、クロス開発ツールのベンダーが統合開発環境 (IDE) によって複雑さを隠蔽し、開発者の作業しやすい環境を整えるという結果にもつながりました。追加されるツールが対応する IDE に密接に統合されることは、強く要求されることです。例えば、Rational Test RealTime は、WindRiver 社の Tornado や GreenHills 社の Multi IDE とうまく統合されます。

また、動的なコンピューター・チップ業界の影響を受けて、新しい実行プラットフォームと関連開発ツールが頻繁にリリースされるという特徴もあります。これは、新しいアーキテクチャーにテスト技術を素早く柔軟に対応させる必要があることを意味します。新しいターゲット・プラットフォームに対する Rational Test RealTime のターゲット配置は、通常 1 週間以内 (場合によっては 2 日以内) に完了します。

実行プラットフォームにおけるリソースと時間の制約

組み込みシステムの定義では、システム上で実行されるアプリケーションの利用可能なリソースに制限があります。これは、特に利用できる RAM が 1 KB 未満のごく小規模なプラットフォームの場合、開発環境への接続が JTAG プローブ、エミュレーター、シリアル・リンクでしか確立できない場合、マイクロプロセッサーにジョブを処理する以上の速度がない場合に言えることです。テスト・ツールでは、開発プラットフォームで用意したテスト・データを (通常は耐えられないほど) 低速な接続リンクで送信するか、テスト・データをターゲット・プラットフォーム上でテスト・ドライバーによって変換するかという、難しい選択を行う必要があります。

Rational Test RealTime で使用される技術は、ターゲット・システムにテスト・ハーネスを組み込む方法です。この方法では、利用できるクロスコンパイラーを使用して、あらかじめアプリケーション・ プログラミング言語 (C、C++、Ada) に変換されたテスト・データをテスト・ハーネス内にコンパイルし、このテスト・ハーネス・オブジェクト・ファイルをアプリケーションの残りの部分にリンクします。makefile で Rational Test RealTime コマンド行インターフェースを使用することで、この一連のビルドはユーザーに対してトランスペアレントになります。最適化されたコード生成、適切なリンク・マップ割り当て、少量の占有スペースなどの機能により、ターゲット・プラットフォームでテスト・ハーネスが必要とするリソースは最小化されます。さらに、次のような利点があります。

- タイミングの正確さが向上します。また、ターゲット内に配置することにより、性能に対するリアルタイムの影響が減少します。
- テスト・ケースの実行中に接続リンクでデータ情報が循環するのを避けることで、ホストとターゲット間の通信が最小化されます。必要に応じて、観測データを RAM に保存し、テスト・ケースが確実に完了したときに、デバッガーまたはエミュレーターを利用してアップロードします。

クロス開発環境は使用しやすくなっていますが、現在出荷されている組み込みシステムの大部分において、開発とテストの難しさはやはり頭痛の種です。Rational Test RealTime の目標は、組み込みシステムの開発者が行う面倒な作業手順を簡単にすることです。

明確に設計されたビジュアル・モデルの不足

組み込みシステムの開発者はコードを好みます。残念ながら、高等教育の卒業生はビジュアル・モデリングに関するトレーニングが不十分で、コードこそが「本物」であると信じる傾向にあります。UML などの言語を使用したビジュアル・モデリングは、設計に文章以外で表現された知識を統合する方向に大きく進展していますが、組み込みシステムのプログラマーの大部分は、ほとんどの作業を単純な従来のプログラミング言語で行いたいと考えています。それほど多くの人が Java を設計に使用しているわけではありません。

開発者が自分の好きな方法で作業できるように、Rational Test RealTime ではテスト・テンプレート・ジェネレーターや API ラッパーなどのウィザードが用意され、アプリケーションのソース・コードに基づいてテスト・ケースを設計できるようになっています。テストを確実にアプリケーションに適用できることが、コード ベースのテスト作成プロセスの主な利点です。欠点は、確認すべき要求をテスト・ケースが反映しているかどうかの保証がないことです。ビジュアル・モデリングを広く採用することで、要求とテスト・ケースの間のギャップが埋められることは明らかです。Rational Test RealTime は、Rational Rose RealTime の UML シーケンス図をテスト・ケース・コンパイラーに渡すことで、この技術を簡単に使用できるようにしています。

品質と保証の基準に関する問題

組み込みシステムの特定の分野 (安全重視システム) では、失敗は許されません。これらのシステムは、原子力、医療、航空電子工学の業界で使用されています。過去の障害ゼロの目標を基に、航空機業界や政府機関 (米国の連邦航空局) は共同で「Software Considerations in Airborne Systems and Equipment Certification (RTCA の DO-178B 規格、「参考資料」[R2] を参照)」の作成に取りかかりました。これは、世界中の航空電子工学業界において、安全重視ソフトウェア開発の一般的な標準となっています。この標準は最も厳しいソフトウェア開発標準の 1 つとして、自動車、医療 (FDA は DO-178B に近い標準をリリースしたばかりです)、防衛、運輸ビジネスなど、人命重視システムが製造されているほかのセクターでも部分的に採用されるようになっています。

DO-178B では、システムの機能障害を引き起こすソフトウェアの異常動作に関して、ソフトウェアを 5 つの重大性レベルに分類します。最も危険なのはレベル A 機器で、ここで発生する障害は、システム全体に破滅的な障害をもたらします。DO-178B には、特にテスト領域に関して、レベル A 機器を十分に安全な状態にするための正確な手順が記述されています。Rational Test RealTime は、レベル A 機器を含むすべてのレベルにおいて、DO-178B で必須とされるすべてのテスト要件を満たしています。

まとめ

このホワイト・ペーパーでは、組み込みシステムのテストに使用される 6 つの段階的な手順に関する、プロセスの概要を説明しました。このプロセス (ごく小規模なシステムから非常に大規模なシステムまでの、すべての範囲を考慮に入れます) と組み込みシステムの特性と制約を考慮すると、組み込みシステムのテストを処理するために理想的な技術で必要とされる要件がわかります。組み込みシステムの分野に対して Rational が新しく提供する Rational Test RealTime は、この理想的な技術のほとんどを備えています。

このホワイト・ペーパーでは、一般的な組み込みシステムのテストを紹介しました。さまざまなトピックに焦点を当てたその他の資料もすぐに用意されます。

用語

用語集については、Paul Szymkowiak が作成しました。

ソフトウェア開発の世界で使用されている多くの用語には、異なる定義があります。このホワイト・ペーパーで使用されている用語は、リアルタイム組み込みシステムの分野に関わっている方にはおそらく見慣れたものでしょう。ただし、ソフトウェア・テストの分野で使用されている別の同義語または関連用語があります。これらの対応について、次の表に示します。

このホワイト・ペーパーで使用されている用語	同義語/関連用語
テスト単位 (GuT) :テストの目的で、環境から分離されたシステムの要素。	関連用語:テスト項目、対象、テスト対象
制御と観測ポイント:テストの特定のポイント。このポイントで、テスト環境の観測が記録されるか、テストの制御のフローに関する決定が行われます。	最も近い同義語:**検証ポイント**
ポストアンブル:システムを特定の安定した終了状態に導くために実行される処理。次のテストを実行するために必要です。	事後条件、リセット
プリアンブル:システムを特定の安定した開始状態に導くために実行される処理。テストを実行するために必要です。	前提条件、設定
テスト・ハーネス:1 つ以上のテスト・ドライバー、テスト固有のスタブ、関連する一連のテストを実行する目的で組み合わされたインスツルメンテーションを集めたもの。	関連用語:テスト・セットまたはテスト・ドライバー、テスト・スタブ
仮想テスター:(1) テスト中にテスト単位とやり取りを行う GuT の外部にあるもの。(2) 実行するテストのインスタンス。一般的には、個人の作業を表します。	関連用語:テスト・スクリプトまたはテスト・ドライバー、テスト・スタブ、シミュレーター、テスト担当者

参考資料

[R1] "*Critical Issues Confronting Embedded Solutions Vendors*", *Electronics Market Forecast*, http://www.electronic-forecast.com/, April 2001.

[R2] **DO-178B, *Software Considerations in Airborne Systems and Equipment Certification*, Radio Technical Commission for Aeronautics — RTCA, http://www.rtca.org/, January 1992.**

著者について

Vincent Encontre は、フランスのツールーズにある Rational の新しいエンジニアリング・センターに勤務する、Embedded and RealTime、Automated Testing Business Unit の責任者です。外出中、会議への出席中、膨大な数の電子メールへの回答中以外は、Rational の製品であり、次世代の Rational 製品で再利用可能な技術である Rational Test RealTime に携わっています。Vincent は、組み込みモデリングやテスト技術、これらの最善の実践原則について幅広い経験を持っています。Rational と ATTOL 社の前、Vincent は 13 年の間 Philips 社と Verilog 社に勤務し、より優れたソフトウェアをより迅速に開発できると考えられた、ソフトウェア開発ツールを開発、販売、サポートしてきました。

休日は、3 人の息子や友人たちとサッカーを楽しみ、フランス南部で芸術的な生活術を (手遅れになる前に) 楽しんでいます。

Rational®
the software development company

Dual Headquarters:
Rational Software
18880 Homestead Road
Cupertino, CA 95014
Tel: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
Tel: (781) 676-2400

Toll-free: (800) 728-1212
E-mail:info@rational.com
Web:www.rational.com
International Locations:www.rational.com/worldwide