

性能モデル検証

平成23年度シラバス

2011年1月28日

国立情報学研究所

トップエスイープロジェクト

代表者 本位田 真一

1. 講座名

性能モデル検証

2. 担当者

長谷川 哲夫、宇佐美 雅紀、他

3. 本講座の目的

本講座では、ネットワーク家電の制御ソフトウェアを題材とした産業ソフトウェアの性能モデル検証問題を扱う。近年ますます大規模化・複雑化するソフトウェアの設計において、特に性能面に関して、その動作の正しさを保証することがより困難になっているが、設計モデルの性能面に関する動作の正しさを自動的に検証する性能モデル検査ツールUPPAALを使用し、実際のシステム開発に適用する方法を習得する。

性能モデル検査ツールを使うためのいくつかのステップにはノウハウが必要となる。特に、ソフトウェアの設計モデルから性能モデル検査ツールの入力モデルへの変換ステップ、モデル検査結果の分析と設計モデルへのフィードバックのステップが重要である。そこで本講座では、設計モデルとして、標準のモデル記述言語UMLを想定し、UPPAALの入力モデルへの変換手法を習得する。UMLのモデルとUPAALLの入力モデルの違いを明確にすることで正確な入力モデル作成の理解を深める。さらに、実際に組込ボード上でのソフトウェア開発を通して、ソフトウェア設計検証の難しさを体感した上で、UPPAALツールの適用における難しさを解決するための検証プロセス、およびノウハウを習得する。また、グループ討議を通じた例題演習により、議論を通してモデル検査技術の理解を深め、検証プロセスの実用的ノウハウを体得できる効果が期待できる。

4. 本講座のオリジナリティ

ソフトウェアの時間性能のモデル検査は、「設計モデル検証(基礎編)」講座、および「設計モデル検証(基礎編)」講座で習得する3ツール(SPIN、SMV、LTSA)を使用しても、原理的には可能である。しかし、これらの講座ではその手法に言及しておらず、また、検証モデルが大幅に複雑化しそれに伴い検証効率も悪化するため、実質的にはUPPAALなどの性能モデル検査ツールの使用が必須となる。

また、UPPAALの講座も存在するが、UPPAALがベースとする理論とツールの使用方法にとどまっており、ソフトウェア設計開発プロセスの中への適用に触れておらず、受講生が習得した内容を速やかに開発現場で適用するための障害となっていた。本講座では、設

計モデルである UML を起点として、検証モデルの作成、検証結果の分析と設計モデルへの反映に関するノウハウを習得することにより、本講座受講後 UPPAAL ツールを速やかに開発現場で適用できるように配慮している。表 1 に、既存の講座の問題点と、本講座における解を示す。

表 1 既存の講座の問題点と、本講座における解

既存の講座の問題点	本講座における解
モデル検査ツール独自の言語・記法のみを使用するため、そのような言語や記法を使用しない、現場での開発工程に組み入れるのが難しい	設計モデル UML を基に、モデル検査の実施から検証結果の分析・設計モデルへの反映をサポートするため、現実的な開発プロセスへのシームレスな組入れが可能
モデル検査手法が解決する、現状のソフトウェア開発における問題点を体感できないため、モデル検査の有用性を実感するのが難しい	同じ情報(UML 設計モデルと C 実装)を用いて、テスト・デバッグとモデル検査の両方を実習するため、従来のテスト・デバッグに比べての、モデル検査の有用性を体感可能
組込みシステムをターゲットとした講座であっても、高々PC 上での実習しか行わないため、組込みシステムに対する有効性を実感するのが難しい	実際の組込みボードを使用した実習を行うため、組込みシステムに対する有効性を体感可能
SPIN、SMV、LTSA といったツールを使用して、性能面のモデル検証を行った場合、状態爆発などにより、現実には検証不可能なことが多いので、現実のシステム開発への適用が難しい	性能面のモデル検証を主要な特徴とするツール UPPAAL を使用するので、現実的なシステムに対する性能モデル検証に容易に適用可能
時間制約特有のノウハウを習得していないと、UPPAAL を利用しても、検証や設計モデル修正が困難な場合が多いので、やはり現実のシステム開発への適用が難しい	時間パラメータの分類や、時間制約に関する検証エラー、および設計誤りの分類といった、時間制約の扱いに関するノウハウを習得するので、現実的なシステムに対する性能モデル検証に容易に適用可能

5. 本講座で扱う難しさ

近年、家電の制御ソフトウェアは大規模化、複雑化が進み、複数の家電機器がネットワークに接続され、相互に連携して動作することで 1 つの機能を実現する、ネットワーク家電市場が急速に立ち上がりつつある。ネットワーク家電では、他の機器との連携動作のためのプロトコルなどが必要になるため、制御ソフトウェアが非常に複雑になり、また不安定なネットワーク環境などを考慮する必要がある。さらに AV 制御など、リアルタイム性が要求される機能が多いので、性能面の性質が重要である。このような環境に対して、高性能、高信頼でかつ安全なソフトウェア設計を実施するために、全動作パターンを人手で検証する従来型の方法論は膨大な工数を必要とし、もはや現実的手段ではない。こ

なお性能面の性質の検査は、従来はテスト・デバッグで実施してきた。しかし、次のような問題点により、テスト・デバッグでは十分実施しきれなくなっている。第 1 に、性能面に関し、別個に検査すべきテストケースは、どのパラメータがどのようにテスト結果に影響するかの場合分けが多岐にわたるため、膨大な件数が必要になる。第 2 に、システムをテストケースで設定した時間パラメータ通りに動作させるためには、ハードウェアや OS も含めたチューニングが必要なため、テストケース通りに動作させるのは困難である。

6. 本講座で習得する技術

モデル検査は、システム上で起こり得る状態を網羅的に調べることによって設計誤りを発見する自動検証手法の一種である。近年、性能面も含めて、システムの動作に関する性質について、時間オートマトンの理論に基づく高品質なモデル検査ツールが開発され利用されて、自動検証が可能になっている。

しかし、大規模ソフトウェア開発にモデル検査を利用する場合には、検証の目的や範囲を明確化した上で、戦略的な検証実施計画を立てることが求められる。ソフトウェアの設計検証には、ソフトウェア開発プロセスと同様に、様々な角度から検証問題を捉えるという実践的なスキルと経験が求められる。さらに性能面の性質、特に時間制約の検証については、時間制約特有のノウハウを習得していないと、検証や設計モデル修正が困難な場合が多い。

本講座では、性能面の性質に対する設計検証に関わる活動を、ソフトウェア設計開発プロセスの1つと捉えた検証プロセスについて学ぶ。具体的には、設計モデル UML をベースにした検証プロセスとして、性能面に関する性質を表現可能なオートマトンベースの検証モデルの作成方法、ツールによる検証結果の分析と設計モデルへの反映に関するノウハウを学ぶ。を中心とする。ここで、性能面に関する性質を表現可能なオートマトンとしては、リアルタイム性(時間制約)については時間オートマトン、また信頼性については確率オートマトンがある。しかし本講座では、優れたツール UPPAAL が存在することから、時間オートマトンによる時間制約性質の検証に絞る。

7. 前提知識

本講座の受講生は、以下の項目を習得済みであることが望ましい。

- 以下に関する基礎理論
 - 並行プログラム：非決定性、資源共有・排他制御、通信、安全性、生存性、公平性
 - 時間オートマトン：意味論(形式言語理論)、通信
 - 時相論理、特に時間 CTL：構文、意味論、パターン(安全性、生存性など)
- UML：特に状態チャートの意味論

なお、これらの項目は、全て「基礎理論」講座で習得可能である。

8. 講義計画

- ・ 概要

- 第1週：導入、UPPAAL チュートリアル(1)
- 第2週：UPPAAL チュートリアル(2)
- 第3週：UPPAAL 時間オートマトンの記述と意味(1)
- 第4週：UPPAAL 時間オートマトンの記述と意味(2)
- 第5週：UML Profile for SPT と時間オートマトン、開発プロセスと検証プロセス
- 第6週：ボード上でのテスト実習
- 第7週：UPPAAL によるギア制御例題の検証(1)
- 第8週：UPPAAL によるギア制御例題の検証(2)
- 第9週：UPPAAL によるオーディオプロトコル例題の検証(1)
- 第10週：UPPAAL によるオーディオプロトコル例題の検証(2)
- 第11週：UPPAAL によるオーディオプロトコル例題の検証(3)
- 第12週：課題による実習(1)
- 第13週：課題による実習(2)
- 第14週：課題による実習(3)
- 第15週：課題による実習(4)

- ・ 詳細

- 第1週：導入、UPPAAL チュートリアル(1)
 - 導入
 - ◇ 設計モデル検証(基本編)では何ができるか
 - ◇ なぜ性能モデル検証が必要か
 - ◇ モデル検査ツール UPPAAL の特徴
 - UPPAAL チュートリアル
 - ◇ 時間オートマトンの基礎
 - 演習：時間オートマトンによる時間表現(個人実習、宿題)
- 第2週：UPPAAL チュートリアル(2)
 - UPPAAL チュートリアル(UPPAAL 操作方法説明と個人実習)
 - ◇ UPPAAL 時間オートマトン(検証モデル)の作成
 - ◇ シミュレーション機能
 - ◇ モデル検査機能
 - 演習：検証によるエラー発見・修正(個人実習、宿題)

- 第 3 週: UPPAAL 時間オートマトンの記述と意味(1)
- プロセス間通信、ロケーションと変数
 - 演習: 検証モデル作成(宿題)
- 第 4 週: UPPAAL 時間オートマトンの記述と意味(2)
- 検証式、オブザーバ
 - 演習: 検証モデル作成 (宿題)
- 第 5 週: UML Profile for SPT と時間オートマトン、開発プロセスと検証プロセス
- UML Profile for SPT と UPPAAL 時間オートマトン
 - 組込みリアルタイムシステムの開発プロセス例と検証プロセスとの関係
 - ◇ 開発プロセスと各ステップにおける成果物一般論
 - ギア制御例題の理解
- 第 6 週: ボード上でのテスト実習
- ギア制御例題を用いた、実機でのテスト実習
 - ◇ 実機操作チュートリアル
 - ◇ 実機を用いたテスト実習
 - 演習: テスト結果解析(グループ実習、宿題)
- 第 7 週: UPPAAL によるギア制御例題の検証(1)
- 検証プロセス
 1. 検証モデル作成
 2. 検証式作成
- 第 8 週: UPPAAL によるギア制御例題の検証(2)
- 検証プロセス続き
 1. 検証→エラー検出
 2. 反例分析手法
- 第 9 週: UPPAAL によるオーディオプロトコル例題の検証(1)
- オーディオプロトコル例題の説明と検証モデル作成
- 第 10 週: UPPAAL によるオーディオプロトコル例題の検証(2)

- 反例分析 手法

第 11 週：UPPAAL によるオーディオプロトコル例題の検証(3)

- 原因分析と、検証モデルの修正 (グループ実習)
- 結果のプレゼンテーション

第 12 週：課題による実習(1)

- ブロックソーター例題の説明
 - ◇ 実習用実機 LEGO Mindstorm NXT のチュートリアル
- 実習：設計モデル (ステートチャート) の作成 (グループ実習)

第 13 週：課題による実習(2)

- 実習：UPPAAL による検証モデル作成と検証式作成 (グループ実習)
 - ◇ 設計モデル (ステートチャート) → UPPAAL の検証モデル
- 実習：UPPAAL による検証 (グループ実習)

第 14 週：課題による実習(3)

- 実習：実装、ステートチャート/UPPAAL モデル/C 言語との比較 (グループ実習)
- 実習：時間パラメータの調整 (グループ実習)

第 15 週：課題による実習(3)

- 実習結果のプレゼンテーション準備 (グループ実習)
- 結果のプレゼンテーションとディスカッション

9. 教育効果

本講座を受講することにより、実際のシステム開発、特に組込みソフトウェアの開発に適用可能な、性能モデル検証プロセスを習得できる。その結果、開発現場において、モデル検査ツール UPPAAL を活用することにより、信頼性の高いシステムを、効率的に開発することができるようになる。

10. 使用ツール

UPPAAL : モデル検査ツール

- 使用する上での難しさ
 - 検証モデルの作成が難しい
 - ◇ 特に時間制約の設定
 - 検証式の作成が難しい
 - ◇ 論理式の構成
 - ◇ ツールで扱える論理式の表現力の限界
 - 検出されたエラーに対する修正が難しい
 - ◇ 反例トレースの分析による原因追跡
 - ◇ 原因の除去、特に時間制約の修正
- 使用上必要なノウハウ
 - 時間制約の UPPAAL における表現
 - ◇ 状態における不変条件
 - ◇ 遷移におけるガード条件、変数更新
 - 時間制約の検証式による表現
 - ◇ 限界境界値 : たとえば、 t ミリ秒以内に完了する必要があるが、最悪 t ミリ秒ぎりぎりかかることがある
 - UPPAAL 検証式の表現力の補完
 - ◇ オブザーバプロセス : ある検証対象 (オートマトン) が特定のイベント列のパターンを実行可能かを調べるために、検証対象と同期を取って動作させるプロセス
- 選択理由、実用性
 - オートマトンモデルに基づいている
 - ◇ 「設計モデル検証」講座からのシームレスな移行が可能
 - 性能モデル検査ツールとして唯一充実している
 - ◇ 完成度の高さ
 - ◇ 視覚的環境
 - ◇ アクティブな開発体制
 - ◇ 比較的豊富な適用実績

1.1. 実習

ブロックソーターの設計検証プロセスを体験することで、時間制約を綱領した設計検証の難しさと効果を実感し、ノウハウを身につける。実習は 3~4 名程度の少人数で構成されたグループ単位で行い、グループ毎に 1 台の LEGO Mindstorm 製ブロックソーター（図 1）を使用する。まず、提示された要求を満たすように設計検証の対象となるブロックソーターの設計モデル（状態遷移図を用いる）を設計する。次に要求の充足、特に時間制約の充足を確認するための検証項目の抽出を行ったうえで、設計モデルを基に UPPAAL による検査モデルを作成する。検証項目を検査式に変換し、UPPAAL を用いて検査モデルを検証する。検証により発見された誤りは検出、修正する。最後に、検証済みの状態遷移図に基づき、C 言語による実装を行い、実際にブロックソーターを動作させる。一連の工程を通じて、設計モデル、検査モデル、実装の対応関係を確認する。各実習は、グループ内で議論しながら行い、議論を通じてモデル検査により時間制約を検証するプロセスへの理解を深める。

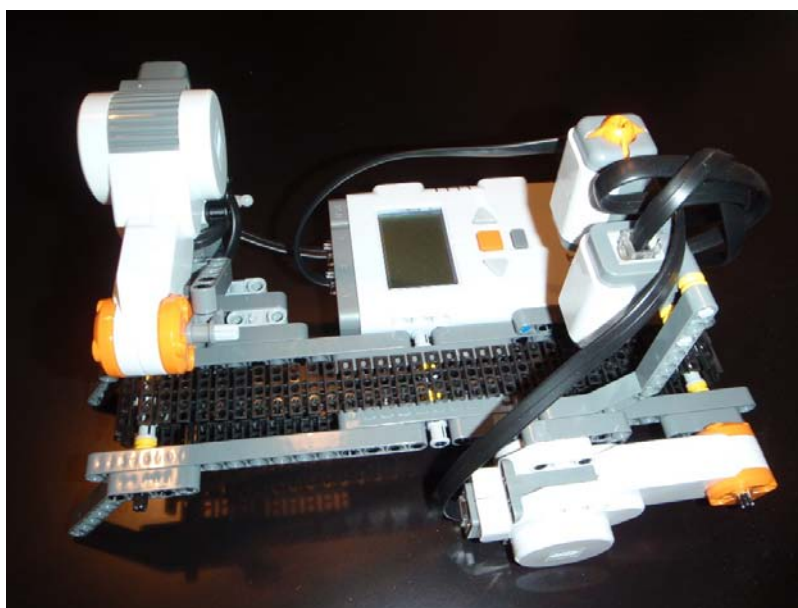


図 1 ブロックソーター実機

12. 評価

演習課題レポート、プレゼン発表、出席日数を総合して評価する。

1.3. 教科書/参考書

- B. Berard, et. al, “Systems and Software Verification: Model-Checking Techniques and Tools,” Springer Verlag, 2001.
モデル検査手法を利用したソフトウェア検証について、入門から実践までの一通りが述べられており、この講義に最適である。
- G. Behrmann et al, “A Tutorial on Uppaal”, Proc. of SFM-RT'04, 2004
モデル検査ツール Uppaal に関する理論的背景とツールの利用法が述べられており、Uppaal を用いた演習に最適である。