2008－SE－159 (25)
2008／3／18

形式的検証用モデル自動生成機能を持つ上流工程支援システムの開発

和田 昭彦　　深海 悟
大阪工業大学大学院 情報科学研究科
〒573-0196 大阪府枚方市北山 1-79-1
e-mail : fukami@is.oit.ac.jp

概要

産業界において，形式的検証技術，特にモデル検査技術に注目が集まっている．モデル検査技術は作成されたモデルに対し論理式で表現された検証式を満たすか否かを自動的に検証する技術である．しかし，モデル検査ツールのためのモデルを別途作成しなければならない問題がある．本稿では，この問題を解決する手段として，モデル検査ツール用のモデルを自動生成する機能を持つ，ソフトウェア開発の上流工程を支援するシステムを提案する．

Development of upper process support system with automatic generation function for formal method verification model

Akihiko Wada　　Satoru Fukami
Graduate School of Computer Science, Osaka Institute of Technology
1-79-1. Kitayama, Hirakata-City, Osaka, 573-0196, Japan

Abstract Formal verification, especially model checking is focused as an important technique in industry. Model checking is a technique that verifies automatically the model created for model checker with verification query. But if we want to use the model checking technique, we have to create a model for specific model checker. To solve this problem, we propose the upper process support system at software development with function that generates models for model checker automatically.

1. はじめに

産業界において，ソフトウェア品質確保の要求は非常に高まっている．それに対して，開発の短納期化が進み，品質確保とは相反する要求が求められている．そのような状況において，品質を確保するための検証技術である形式的検証技術，特にモデル検査技術[1]が注目を集めている．しかし，モデル検査技術を使用する際には，通常システム開発で作成する仕様書に加えてモデル検査ツール用のモデルを作成しなければならず，モデル検査技術導入の大きな壁となっている．

また，システム開発における下流工程を支援するツールは多数存在しているが，上流工程を支援するツールというものはあまり多くない．そのため仕様書のフォーマットは各使用者が独自に作成したものを使用することにより，各プロジェクトによって別フォーマットを使用している場合も少なくない．UML 図の描画ツールについても多数存在するが，描画ツールの域を超えるものは多くなく，仕様書でシステムの動的振る舞いを記述しても検証を行うことができない．

本稿ではこれら問題点を解決するために，システム開発の上流工程を支援するための統合ツールの提案・開発を行なった．本ツールは，IPA/SEC が提案している「組込みソフトウェア向け開発プロセスガイド」[2]に準拠した各種仕様書作成機能，UML の各種図面作成機能，これら仕様書や図面をプロジェクト単位で統合的に管理する機能，UML ステートマシン図からモデル検査ツール UPPAAL 用モデル自動生成機能を

持つ．本稿では，UML ステートマシン図からUPPAAL 用モデル自動生成機能を中心に報告する．

2. モデル検査適用における問題点

2.1. モデル検査技術

モデル検査技術とは，有限状態モデルで表現されたシステムの動的振る舞いを，時相論理式で表現された検証式を用いて自動検証を行なう技術である．ソフトウェアの設計をモデル検査ツールで検証する場合は，対象とするシステムの動的振る舞いをモデリングし，検証したい性質を時相論理式で表現し，モデル検査を行なう検証エンジンに入力する．モデル検査ツールは与えられた性質が成り立つかどうかを網羅的に検証し，成立するかしないかを報告する．また，成立しない場合は反例を表示する．

ソフトウェア開発において，あるタイミングのみにおいて生起する不具合動作を検証することは，通常のテストやレビューでは難しい．また，不具合の混入工程が上流工程であれば，開発手戻りが発生し，システム納期に間に合わない事態も発生しうる．それら不具合を通常のテストとは別にモデル検査技術を用いて網羅的に検証することで，システムの信頼性を向上させることができる．

モデル検査ツールは様々なものがあり，特性も各ツールによって異なる．有名な検査ツールにUPPAAL[3]がある．UPPAAL は有限状態モデルを GUI から作成する．そのため，使用者は視覚的に動作を確認しやすくなっている．また，各有限状態モデルに対し通信ラベルを設定することができるため，同期通信を行いながら動作するシステムの記述が容易になっている．また，性質の記述は，他のツールと比較して検証できる種類が少ないものの，ある状態に将来到達することが可能であるという性質や，デッドロックが発生するかどうかを検証する性質は記述可能であるため，十分実用に耐えうる．

2.2. 現行モデル検査の問題点

現在のモデル検査技術では，作成した仕様書モデルとは別にモデル検査ツール用のモデルを作成する必要がある．そのため，モデル検査ツール用のモデルを作成するために，通常のシステム開発とは別にモデリングをするための時間が必要となる．システム開発の短納期化が進んでいる現状を踏まえると，通常のシステム開発とは別にモデリングを行うための時間を容易に確保できる状態であるとはいい難い．

また現行モデル検査では，開発者が作成した仕様書からモデル検査用モデルを生成するため，仕様書でのモデルとモデル検査ツール用モデルの間に不一致が入り込む可能性がある．不一致が発生した場合，モデル検査を用いて検証を行なった結果が不適格でモデル検査を行なう意味自体がなくなってしまう．

モデル検査によるシステムの検証をより広く使用されるようにするためには，このような問題点を解決し，実ソフトウェア開発現場への導入を容易にする必要があると考えられる．

もちろん，モデル検査技術を導入するには，一定の知識を必要とするが，従来からソフトウェア開発現場で行われているモデリング技術が活用できることも事実である．そこで，開発現場でも普及が進んでいる UML ステートマシン図からモデル検査ツール入力ファイルを自動生成するツールについて動作確認を行い，適用可能性について検討した

3. 開発システム概要

本システムの目的は，これまで開発者各自が独自で作成してきた仕様書をフォーマット提供することにより，統一的な仕様書として扱うことができるようにすると共に，上流工程において作成されるシステムの動的振る舞いを表すステートマシン図からモデル検査ツールUPPAALの入力ファイルを自動生成することにより，開発者に対するモデル検査導入時の負担を軽減することである．

本章では，開発したシステム(図1)の機能と，それら機能が必要とされる理由について述べる．

3.1. 仕様書作成支援機能

本システムが提供する仕様書作成画面を用いて仕様書を作成する．「組込みソフトウェア向け開発プロセスガイド」におけるソフトウェア・エンジニアリング・プロセスの上流工程にあたるSWP1(ソフトウェア要求定義)，SWP2(ソフトウェア・アーキテクチャ設計)，SWP3(ソフトウェア詳細設計)の各工程で用いられる仕様書を中心にサポートしている．

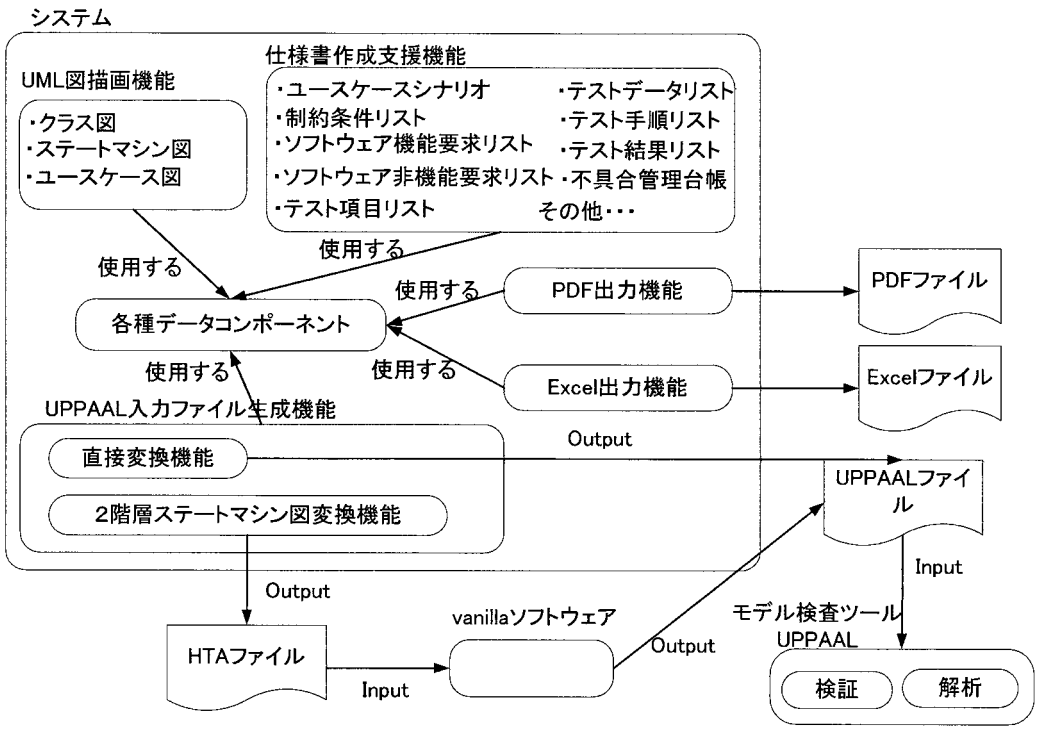

図 1 システム構成図

通常各種仕様書を作成する場合は，開発者が各自 Excel 等を用いてフォーマットを作成して仕様書を記述する．しかし，作成される仕様書にデータの関連性を明記するのは難しい．ユースケースシナリオを例に挙げると，シナリオはシーケンシャルな情報を持つべきだが，Excel 等で作成すると順序情報はシナリオ自体に含めることができず，文字列であること以上の情報を持つことができない．本システムを用いて作成されるユースケースシナリオでは，連続するシナリオに順序番号を割り当てており，各シナリオにおける順序情報を格納している．そのため，Excel 等で作成される仕様書とは異なり，各種データ間の相互関連情報も含むことができる．また，作成される各種仕様書には，通常仕様書間の相互関連が発生する(図 2)．しかし，各種仕様書がExcel等で書かれた場合に相互関連を明示することは難しい．例えば，本システムで作成することができるテストデータリストは，事前に作成されたテスト項目リストを参照に作成されるが，Excel でテスト項目リストを作成している場合，開発者が Excel ファイルを確認しながらテストデータを作成することになり，漏れが発生する可能性がある．本システムでは相互関連する仕様書間においてデータを再利用することにより，テスト項目すべてに対して漏れなくテストデータを作成することができる．

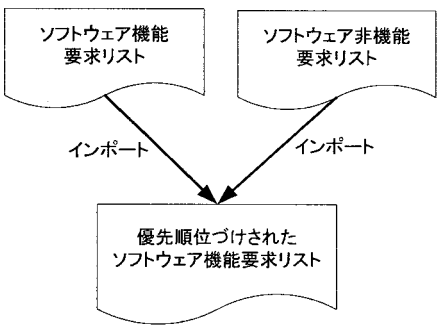

図 2 仕様書相互関連の例

3.2. UML 図描画機能

UML 図のユースケース図・クラス図・ステートマシン図を現在描画することができる．各 Diagram の描画アイテムの詳細編集にはポップアップ画面

を用いることで，複数種類の描画アイテム編集画面を同時に開くことができる．また，アイテム同士を結ぶ遷移が不正確である場合，警告を表示することで不正な関連を防ぐことができる．また，これまでモデリングツールを用いて作成していた UML 図を，本上流工程支援システムで仕様書と同じ環境で作成することにより，一元管理を行うことができる．

3.3. UPPAAL 入力ファイル生成機能

本システムの UML 図描画機能を用いて作成されたステートマシン図からモデル検査ツール UPPAAL 入力ファイルを自動生成することができる．生成されたファイルを入力ファイルとして UPPAAL で検証を行い，性質が成り立つかどうかを検証する．

変換には2種類の手法を用いる．2種類の手法について以下に述べる．

① 直接変換手法

階層構造を持たないステートマシン図を UPPAAL 入力ファイルに変換する．しかし，ステートマシン図と UPPAAL 入力ファイルのデータ構造は異なる．ステートマシン図には通常の状態と複合状態・擬似状態（初期状態・フォーク・ジョイン）・終了状態などがある．それに対し，UPPAAL 入力ファイルは状態を表す location データのみである．二つのファイル間におけるマッピングは，図3に示す規則に従って変換を行なう．生成される UPPAAL ファイルのバージョンは uppaal-1.5 である．

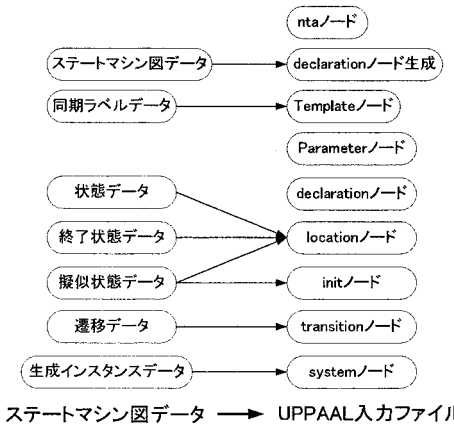

図 3 直接変換ルール

② 2階層ステートマシン図変換手法

複合状態（階層構造）を持つステートマシン図を UPPAAL 入力ファイルに変換する．複合状態を含む階層構造モデルから UPPAAL 入力ファイルに変換する手法は存在するが，複雑すぎるため現実的ではない．そこで，ステートマシン図を階層構造を持つ時間オートマトンである HUPPAAL[4]に変換し，HUPPAAL から UPPAAL 入力ファイルに変換する2段階変換手法を用いる．

階層構造モデルである複合状態を持つステートマシン図から HUPPAAL モデルへ変換する（図 4）．状態は図 5 に示すとおりマッピングによって変換される．この中で複合状態は特殊な状態として取り扱う．

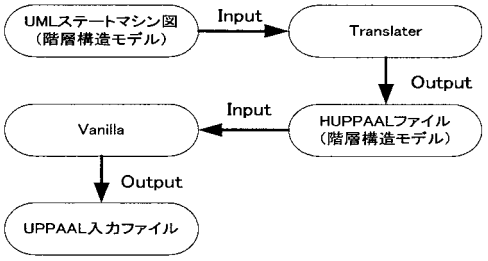

図 4 2 階層ステートマシン図変換フロー

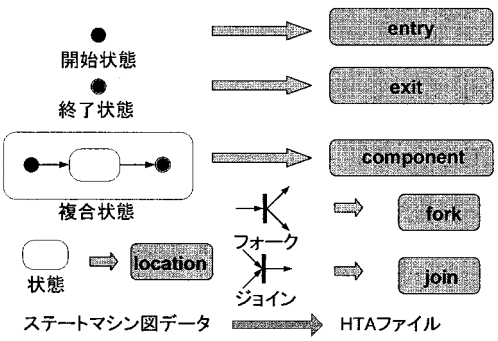

図 5 2 階層ステートマシン図変換手法
マッピングルール

複合状態は各種状態を内包することができるため，1 つのステートマシンとして取り扱う．

これらルールに沿って HUPPAAL ファイルを生成する（図 6）．

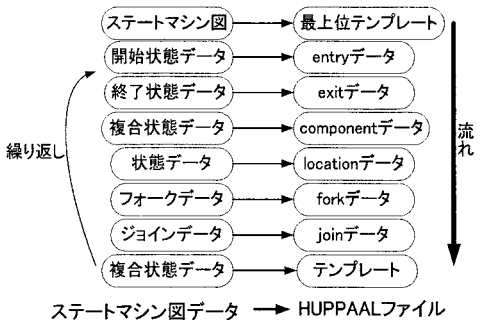

図 6 HUPPAAL 生成の流れ

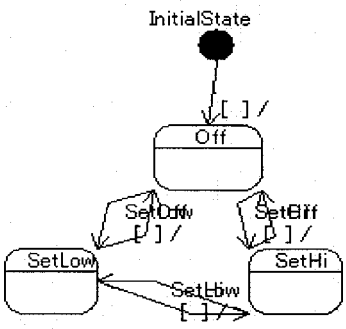

図 7 ユーザの振る舞い

生成された HUPPAAL ファイルを vanilla[5] で読み込むと，階層構造を持たない Flat な時間オートマトンである UPPAAL 入力ファイルが出力される．

3.4. 各種データコンポーネント

本システムで取り扱うデータをまとめたパッケージである．仕様書ごとに最上位データクラスであるドキュメントクラスが存在し，その中に各仕様書で保持するデータクラスが配置されている．すべてのデータは XML 形式で保存されており，ファイルを開いた時点で XML ファイルを読み込み，ドキュメントクラスに仕様書情報が格納される．すべてのドキュメントクラスはマネージャクラスによって制御され，各機能はマネージャクラスに指示を出すことによってデータを使用する．

4. 適用例

本研究で作成したシステムを用いて，動作検証を行なった．

動作検証の対象は，シンプルな扇風機の動作を記述したものである．対象の扇風機は Off スイッチと風の強度を表す Low スイッチ・Hi スイッチとモータから成る．実際のユーザの動作を表したモデルを加えた5つのステートマシン図から構成される（図 7～図 11）．

ユーザの振る舞いは，各スイッチをどれでも選択可能であることを表している．

モータはユーザの動作に対してモータの強弱・Off が切り替わる．

Off スイッチ・強スイッチ・弱スイッチはユーザの動作に対して On・Off の値をとる．

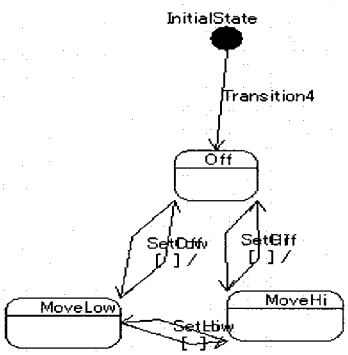

図 8 モータのステートマシン図

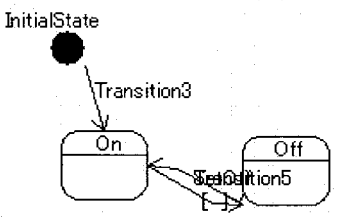

図 9 Off スイッチのステートマシン図

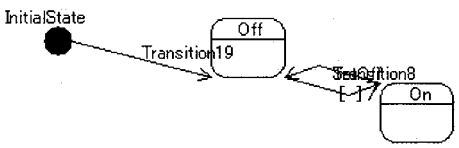

図 10 強スイッチのステートマシン図

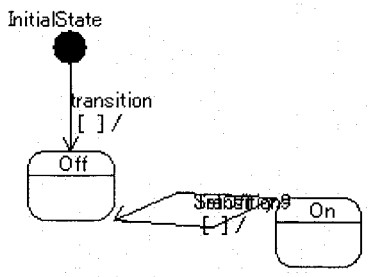

図 11　弱スイッチのステートマシン図

これら5つのステートマシン図に同期ラベルを付け足したものを UPPAAL 入力ファイルとして出力し，UPPAAL を起動させた．ステートマシン図から生成された5つのテンプレートが表示されている（図 12～図 16）．同期ラベルは，「SetLow」「SetHi」「SetOff」をユーザの振る舞いから他の振る舞いに対して送信する形で設定を行った．

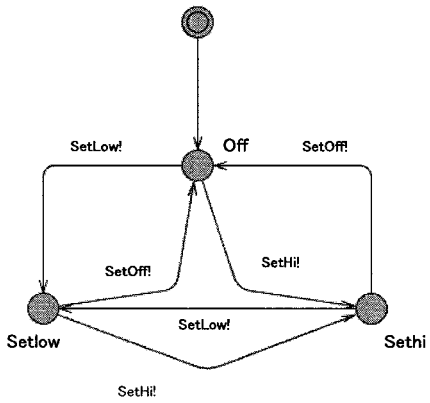

図 12　ユーザの振る舞い(UPPAAL)

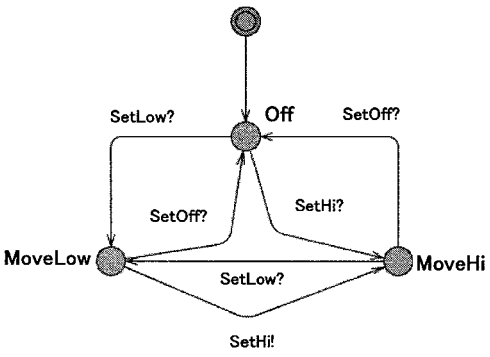

図 13　モータの振る舞い(UPPAAL)

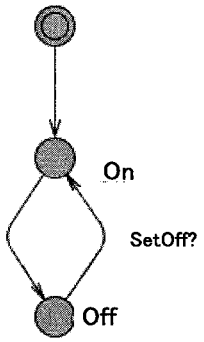

図 14　Off スイッチの振る舞い(UPPAAL)

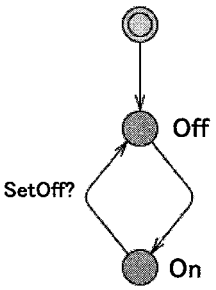

図 15　強スイッチの振る舞い(UPPAAL)

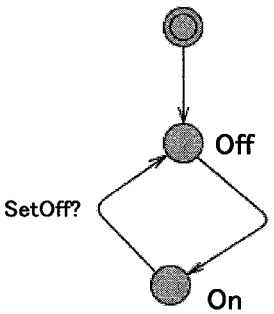

図 16　弱スイッチの振る舞い(UPPAAL)

実際に UPPAAL を動作させてみた結果，ステートマシン図から生成されたファイルはシステムエラーを吐き出すことなく正常に動作することが確認された．しかし，Simulaterを動作させた際に，作成したステートマシン図では遷移が不足していることが分かった．各スイッチの状態において，その状態にとどまり続けるための自己遷移が発生しなけ

ればいけないという遷移や，Off スイッチのステートマシン図において、弱スイッチや強スイッチがOn になった時点で On から Off に切り替わるために遷移が複数存在する必要があるというものである．この問題点においては，実際の開発ではスイッチが押下されない限り自動的に状態が切り替わることは考えられない．そのため、バグであるとは断定できないが、仕様書として曖昧さが残るためにステートマシン図を作りなおし，厳密な動作定義を行う必要があると考えられる．

本システムを用いて作成されたステートマシン図を UPPAAL 入力ファイルに自動変換し、シミュレーションを行った結果、本適用実験においてステートマシン図の曖昧表記を発見することができた．

5. まとめ

本稿では本システムをシステム開発の上流工程で用いることで，モデル検査技術をシステム開発に導入することができることを確認した．しかし，研究を進めるうちに新たな問題点及び必要機能があることが明確になった．

- 階層構造を持つステートマシン図の内部での変換

現在，階層構造を持つステートマシン図の UPPAAL ファイルへの変換には，外部ソフトウェア vanilla を用いている．そのため，複合状態を含むステートマシン図を複数同時に変換することができない．この問題は本システム内部に変換機能を取込むことで解決できる．

- ステートマシン図のユーザ指定ラベルにおいての特殊指定事項記述

ステートマシン図からUPPAAL入力ファイルを生成する際，生成支援機能の一部として画面から同期チャネルを設定している．しかし，本来そのような同期通信チャネルの設定はステートマシン図描画時に設定されるべきである．

- 直接変換手法における設定保存

直接変換手法において，UPPAAL 入力ファイル生成時に設定した情報は，保存されず生成された時点で破棄される．しかし，スパイラルモデルで開発が進むプロジェクトなどでは，以前に生成した UPPAAL ファイルの設定などが情報として必要となる．そのため，設定ファイルを保存し，それまでの履歴として取り扱うことができる状況になることが望まれる．

- 検証式作成支援機能の実装

UPPAAL では，モデル情報を持つファイルとは別に検証式情報を持つ query ファイルが存在している．現在のシステムでは，検証式を作成する画面は存在しているものの，内部実装が追いついていないため，検証式を支援できていない．ソフトウェア開発者にとってモデル検査を行なう際に必要となる論理式を使用する場面は，検証式を作成するフェーズであり，検証式作成支援機能を実装することによってよりソフトウェア開発者にとってモデル検査を使用しやすくすることができる．

- 他のモデリングツールとの連携機能

本システムを用いて作成された UML 図のみならず，他ツールで作成された UML 図をインポートすることができれば，より多くのソフトウェア開発者に使用されると考えられる．現在有名な無償モデリングツールに JUDE, ArgoUML などが挙げられる．これらモデリングツールとの連携をとることで本システムが持つ UPPAAL 入力ファイル自動生成機能を用いてモデル検査を容易に行なえると考えられる．

今後はこれら機能を実現することでより有用なツールに進化させていく予定である．

参考文献

[1] E. M. Clarke Jr., O. Grumberg, D. A. Peled: Model Checking, The MIT Press (1999).

[2] 独立行政法人情報処理推進機構 ソフトウェア・エンジニアリング・センター：組み込みソフトウェア向け開発プロセスガイド，(株)翔泳社(2006).

[3] G. Behrmann, A. David, and K. G. Larsen. :A Tutorial on Uppaal, In proceedings of the 4th International School on Formal Methods for the Design of Computer,Communication, and Software Systems (SFM-RT'04). LNCS 3185 (2004).
下記にも詳しい情報がある.
http://www.it.uu.se/research/group/darts/uppaal/documentation.shtml

[4] A. David and M.O. Moller : From HUppaal to Uppaal: A Translation from Hierarchical Timed Automata to Flat Timed Automata, BRICS Report Series RS-01-11, BRICS (2001).

[5] M.O.Moller: Vanilla-1 Translation from HUppaal to Uppaal,
http://www.brics.dk/~omoeller/hta/vanilla-1/ (2001).